MAC5711 Análise de Algoritmos

Slides de Paulo Feofiloff

[com erros do coelho, cris e alair]

Slides de Paulo Feofiloff

[com erros do coelho, cris e alair]

"A análise de algoritmos é uma disciplina de engenharia. Um engenheiro civil, por exemplo, tem métodos e tecnologia para prever o comportamento de uma estrutura antes de construí-la.
Da mesma forma, um projetista de algoritmos deve ser capaz de prever o comportamento de um algoritmo antes de implementá-lo."

Avisos

- Página da disciplina:

  http://paca.ime.usp.br/

- Paca: Cadastro, fórum, entregas de trabalho
- Monitor: Omar Gaudio
  (?)
- Livros:
  - CLRS = Cormen, Leiserson, Rivest, Stein, *Introduction to Algorithms*
  - AU = Aho, Ullman, *Foundations of Computer Science*
  - TAOCP = Knuth, *The Art of Computer Programming*
- Tarefas
- Alunos especiais: formulário, comissão.

MAC5711

Continuação natural de MAC5710 Estrutura de Dados e sua Manipulação.

A disciplina

- estuda algoritmos eficientes e elegantes para alguns problemas computacionais básicos;
- prova a correção de algoritmos iterativos a partir de suas relações invariantes;
- explora a estrutura recursiva dos problemas para construir algoritmos eficientes;
- formaliza o conceito de desempenho (assintótico) de algoritmos;
- calcula o desempenho de vários algoritmos básicos.

Principais tópicos

- Elementos de análise assintótica (notação O, Ω e Θ)
- Solução de recorrências
- Análise da correção e desempenho de algoritmos iterativos
- Análise da correção e desempenho de algoritmos recursivos
- Análise de pior caso e análise probabilística
- Algoritmos de busca e ordenação
- Algoritmos de programação dinâmica
- Algoritmos gulosos
- Algoritmos para problemas em grafos
- Análise amortizada de desempenho
- Introdução à teoria da complexidade: problemas completos em NP

Introdução à AA

CLRS 2.1–2.2

AU 3.3, 3.6

Exercício 1.A

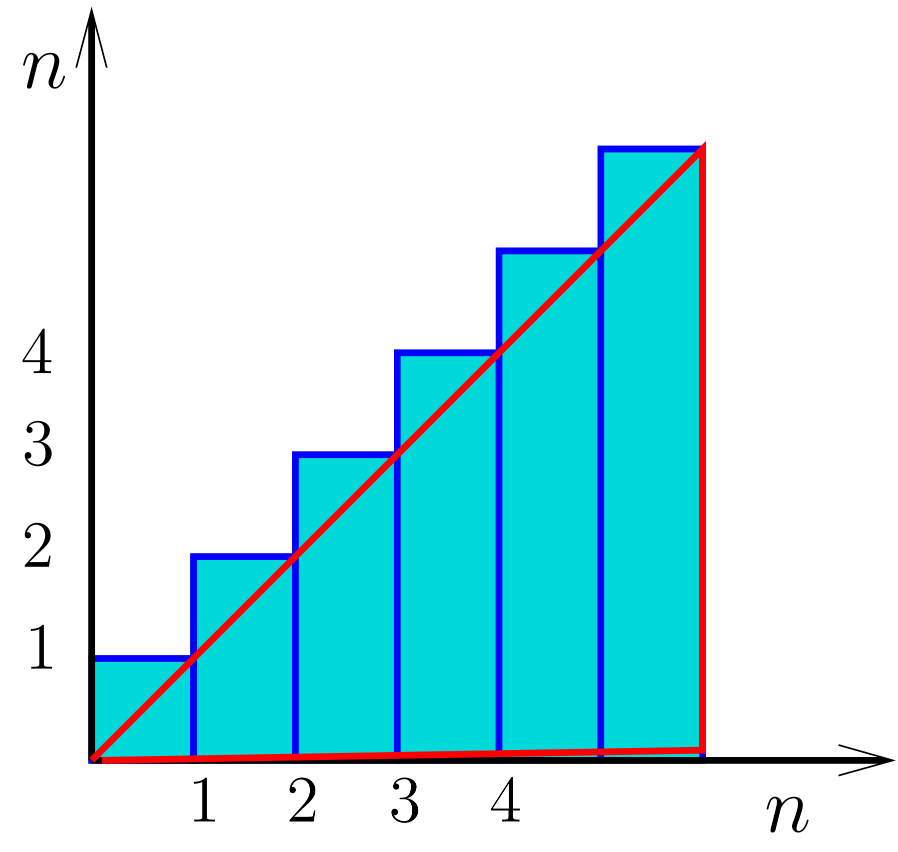
Quanto vale S no fim do algoritmo?

1 $S \leftarrow 0$
2 **para** $i \leftarrow 2$ **até** $n - 2$ **faça**
3   **para** $j \leftarrow i$ **até** n **faça**
4         $S \leftarrow S + 1$

Escreva um algoritmo mais eficiente que tenha o mesmo efeito.

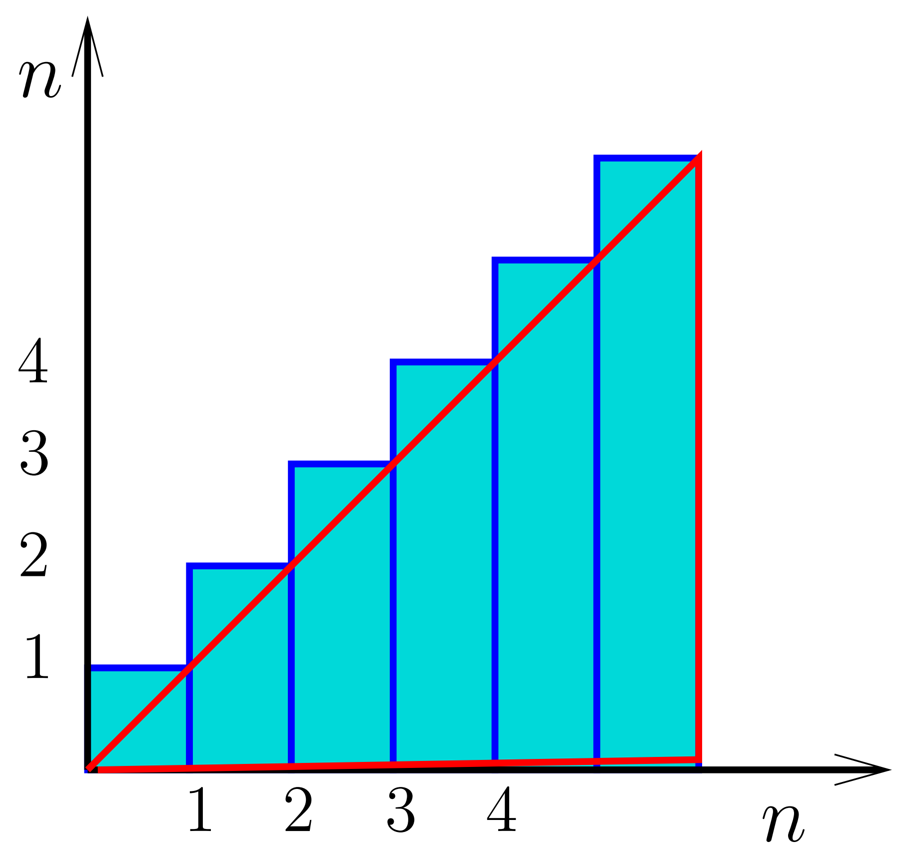
$$1 + 2 + \cdots + (n-1) + n = ?$$

Carl Friedrich Gauss, 1787



$$1 + 2 + \cdots + (n-1) + n = ?$$

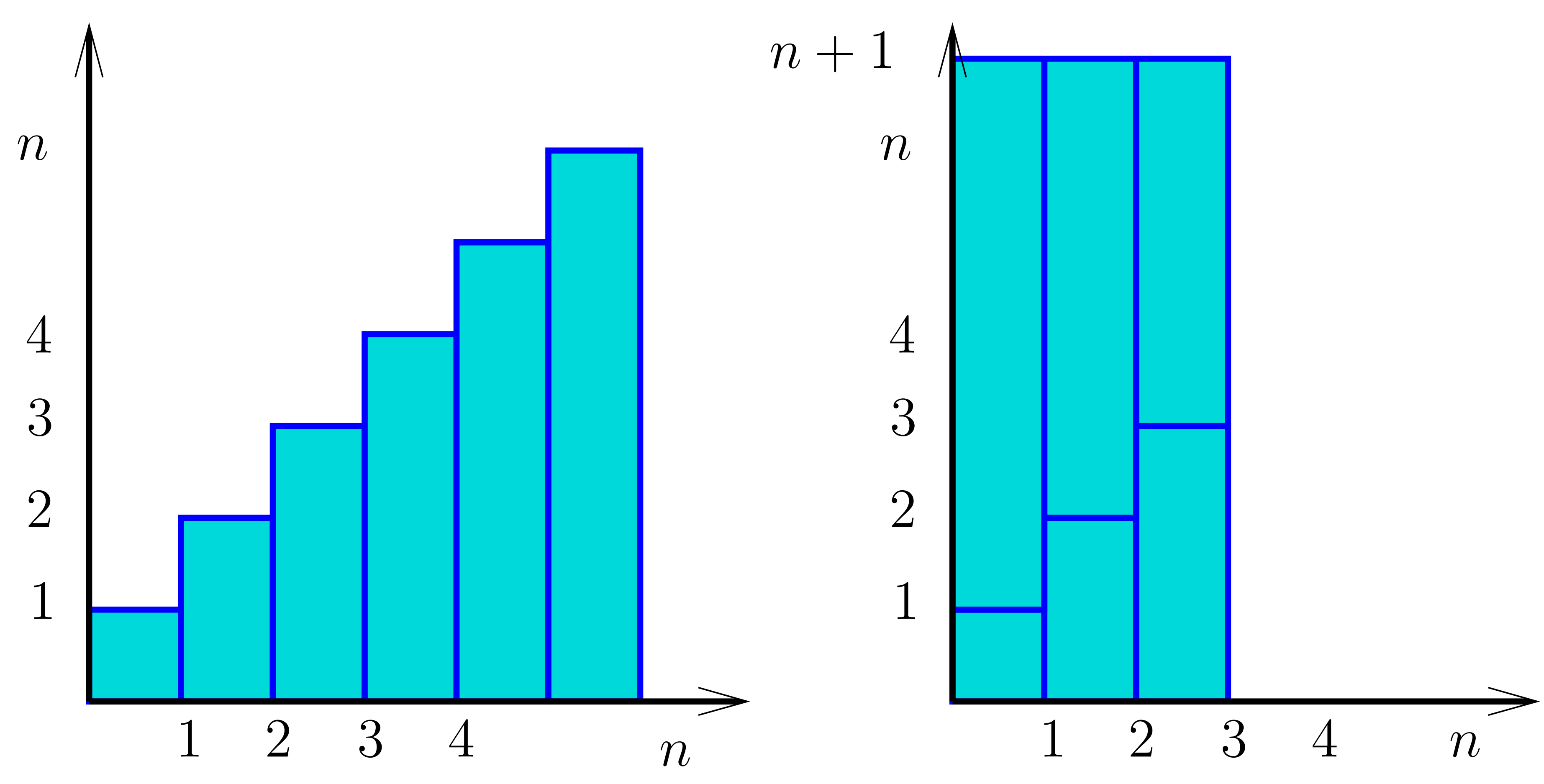
Carl Friedrich Gauss, 1787



$$\frac{n^2}{2} + \frac{n}{2} = \frac{n(n+1)}{2}$$

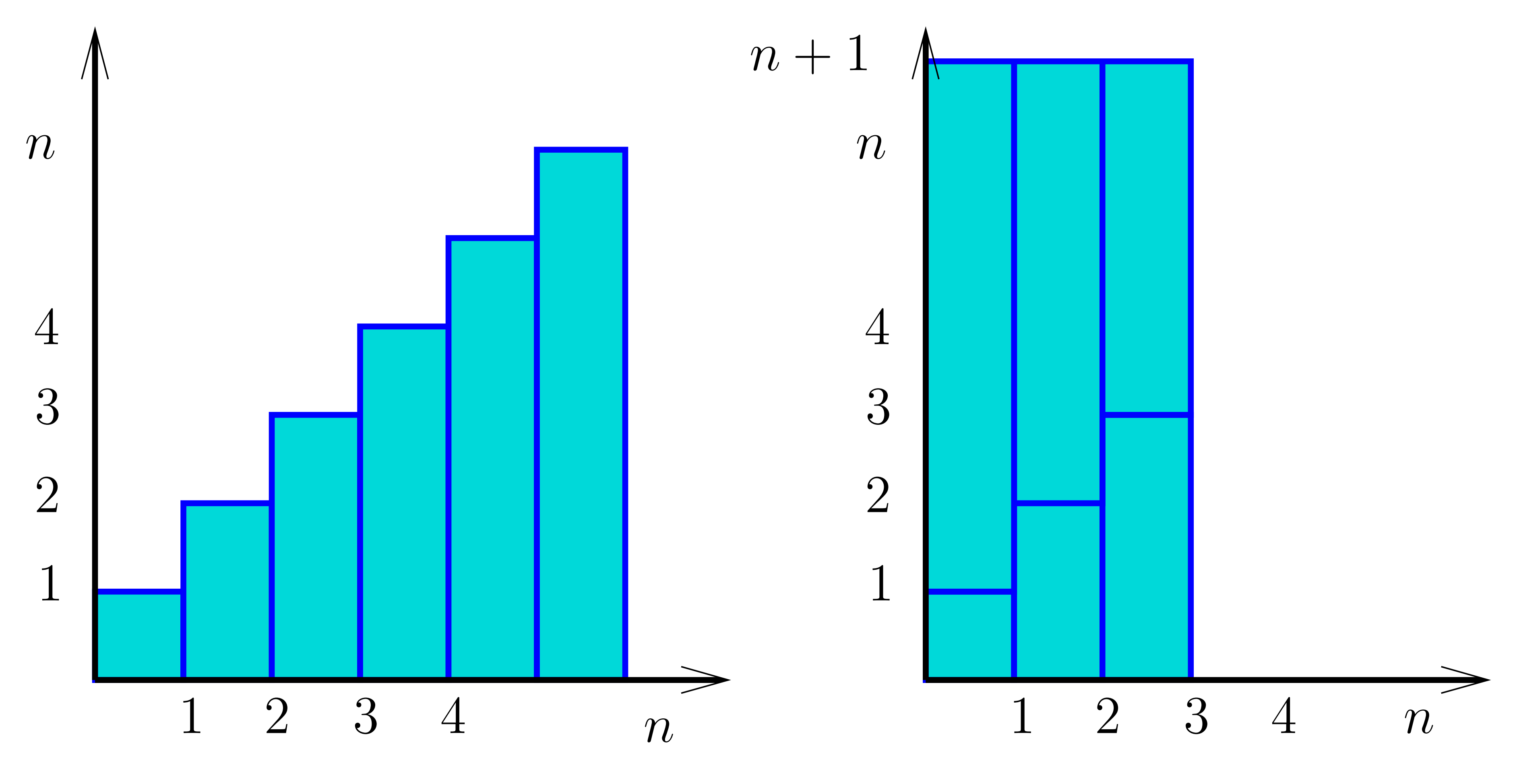
$$1 + 2 + \cdots + (n-1) + n = ?$$

Carl Friedrich Gauss, 1787



$$1 + 2 + \cdots + (n-1) + n = ?$$

Carl Friedrich Gauss, 1787



$$(n+1) \times \frac{n}{2} = \frac{n(n+1)}{2}$$

Solução

Se $n \geq 4$ então no fim da execução das linhas 1–4,

$$
\begin{aligned}
S &= (n-1) + (n-2) + \cdots + 4 + 3 \\
&= (n+2)(n-3)/2 \\
&= \tfrac{1}{2}n^2 - \tfrac{1}{2}n - 3\,.
\end{aligned}
$$

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de mode que ele fique crescente.

Entra:

Ordenação

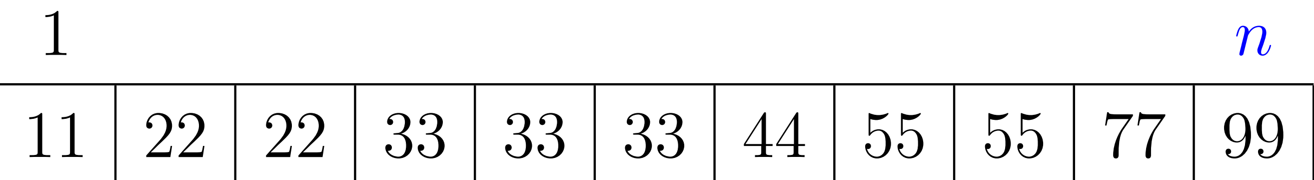
$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de mode que ele fique crescente.

Entra:

1										n
33	55	33	44	33	22	11	99	22	55	77

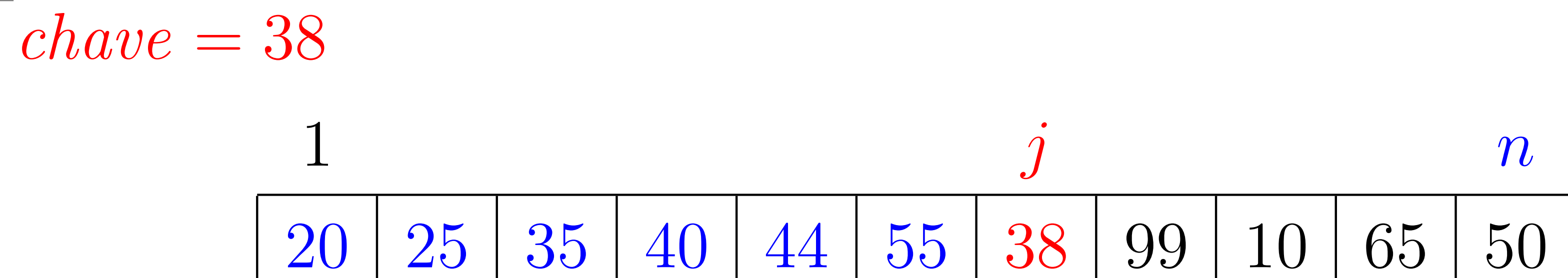
Sai:



1										n
11	22	22	33	33	33	44	55	55	77	99

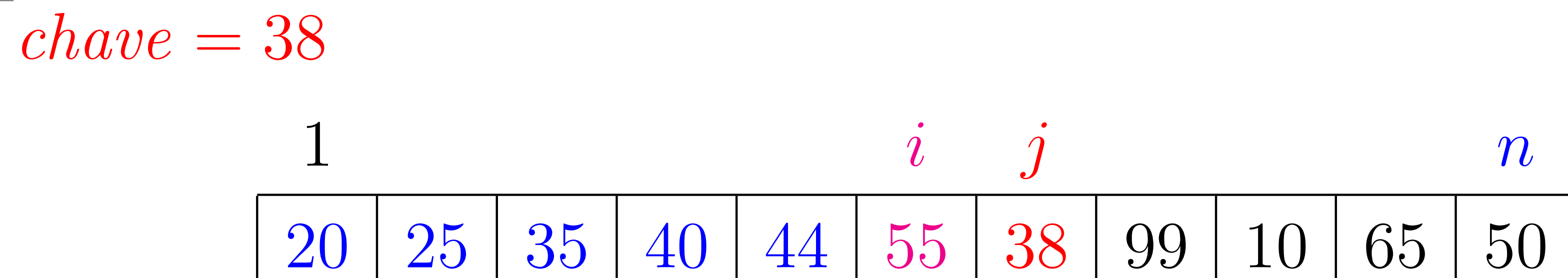
Ordenação por inserção

$chave = 38$

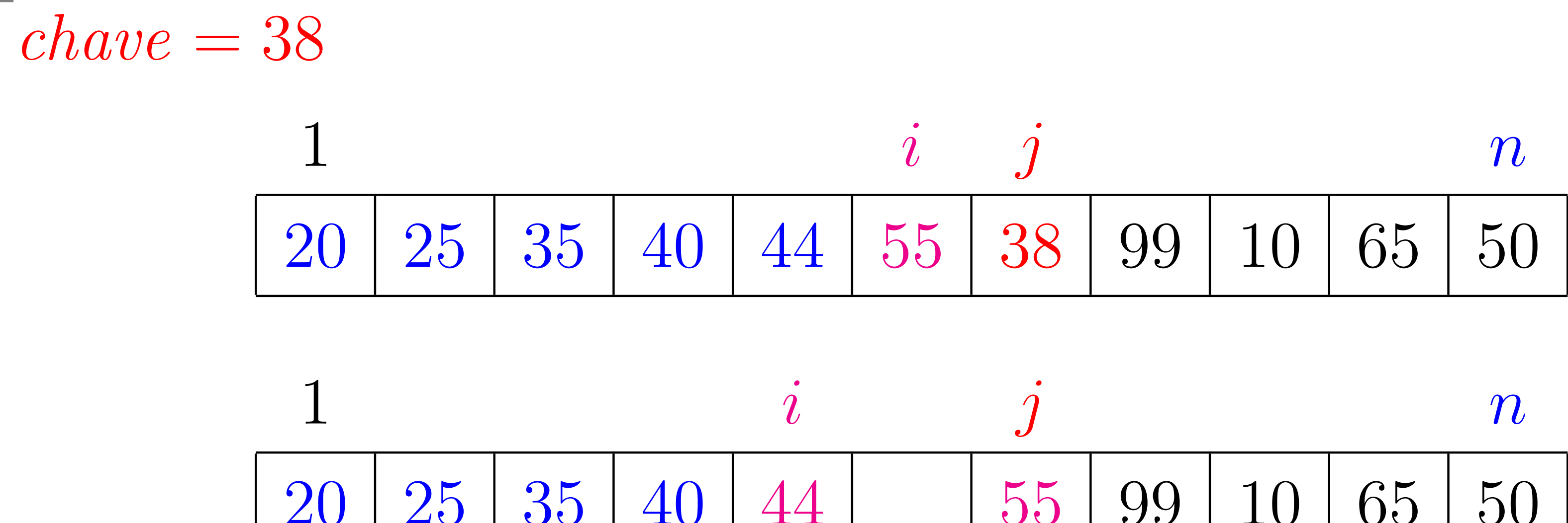


Ordenação por inserção

$chave = 38$



Ordenação por inserção



Ordenação por inserção

$chave = 38$

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

1				i		j				n
20	25	35	40	44		55	99	10	65	50

1			i			j				n
20	25	35	40		44	55	99	10	65	50

Ordenação por inserção

$chave = 38$

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

1				i		j				n
20	25	35	40	44		55	99	10	65	50

1			i			j				n
20	25	35	40		44	55	99	10	65	50

1		i				j				n
20	25	35		40	44	55	99	10	65	50

Ordenação por inserção

$chave = 38$

1					i	j				n
20	25	35	40	44	55	38	99	10	65	50

1				i		j				n
20	25	35	40	44		55	99	10	65	50

1			i			j				n
20	25	35	40		44	55	99	10	65	50

1		i				j				n
20	25	35		40	44	55	99	10	65	50

1		i				j				n
20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

$chave$	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

$chave$	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

$chave$	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

$chave$	1								j		n
10	20	25	35	38	40	44	55	99	10	65	50

Ordenação por inserção

$chave$	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

$chave$	1								j		n
10	10	20	25	35	38	40	44	55	99	65	50

Ordenação por inserção

$chave$	1						j			n
99	20	25	35	38	40	44	55	99	10	65	50

$chave$	1							j		n
10	10	20	25	35	38	40	44	55	99	65	50

$chave$	1								j	n
65	10	20	25	35	38	40	44	55	99	65	50

Ordenação por inserção

$chave$	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

$chave$	1								j		n
10	10	20	25	35	38	40	44	55	99	65	50

$chave$	1									j	n
65	10	20	25	35	38	40	44	55	65	99	50

Ordenação por inserção

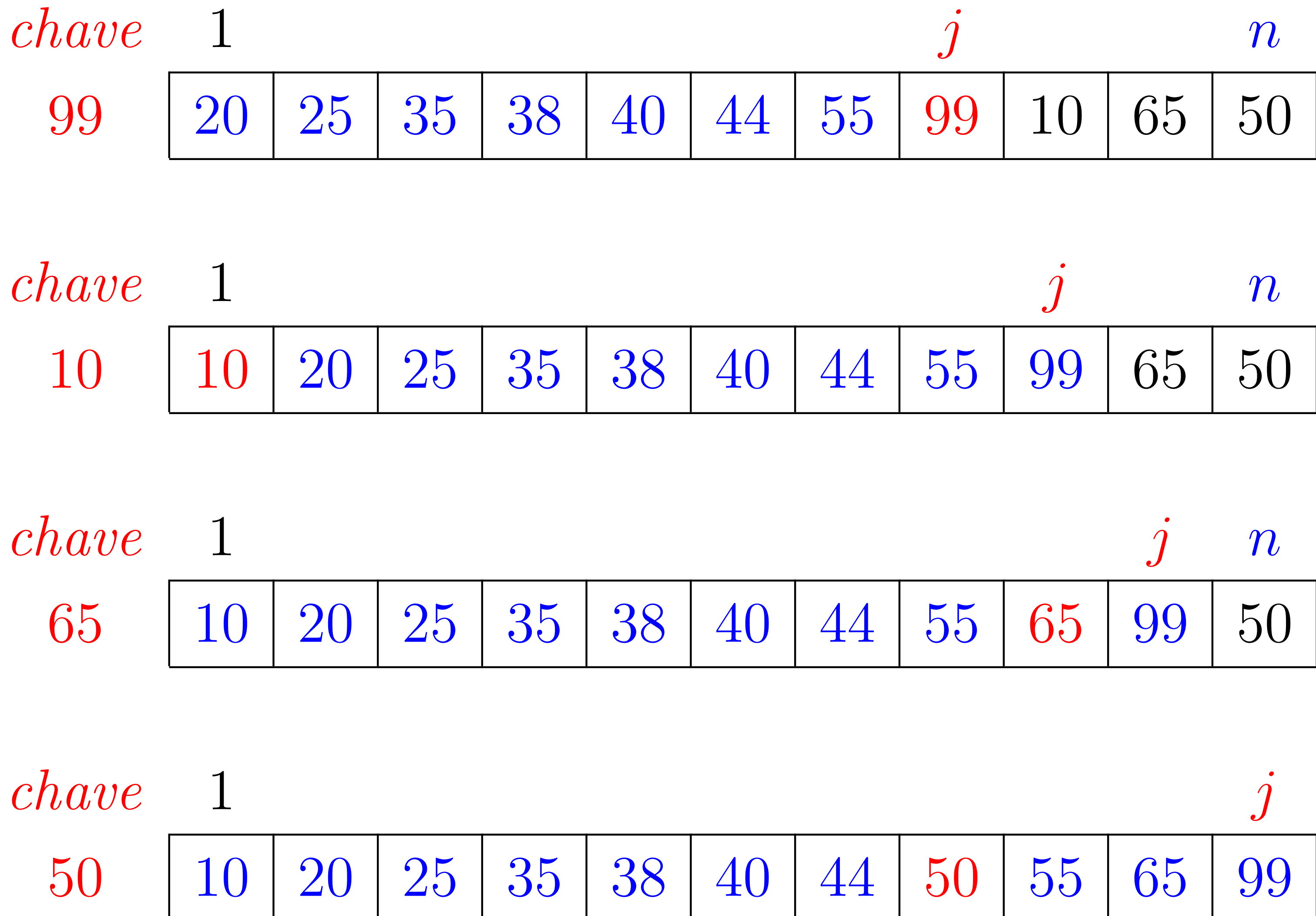
$chave$	1							j			n
99	20	25	35	38	40	44	55	99	10	65	50

$chave$	1								j		n
10	10	20	25	35	38	40	44	55	99	65	50

$chave$	1									j	n
65	10	20	25	35	38	40	44	55	65	99	50

$chave$	1										j
50	10	20	25	35	38	40	44	55	65	99	50

Ordenação por inserção



Ordenação por inserção

Algoritmo rearranja $A[1\,..\,n]$ em ordem crescente.

ORDENA-POR-INSERÇÃO (A, n)

1 **para** $j \leftarrow 2$ **até** n **faça**

2 $\quad chave \leftarrow A[j]$

3 $\quad i \leftarrow j - 1$

4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**

5 $\quad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca

6 $\quad\quad i \leftarrow i - 1$

7 $\quad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere

Ordenação por inserção

Algoritmo rearranja $A[1 \mathinner{.\,.} n]$ em ordem crescente

ORDENA-POR-INSERÇÃO (A, n)
0 $\quad j \leftarrow 2$
1 $\quad$ **enquanto** $j \leq n$ **faça**
2 $\qquad chave \leftarrow A[j]$

3 $\qquad i \leftarrow j - 1$
4 $\qquad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**
5 $\qquad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca
6 $\qquad\quad i \leftarrow i - 1$

7 $\qquad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere
8 $\qquad j \leftarrow j + 1$

O algoritmo faz o que promete?

Correção do algoritmo!

O algoritmo faz o que promete?

Correção do algoritmo!

Relação invariante chave:

(i0) na linha 1 vale que: $A[1\,..\,j-1]$ é crescente.

1						j				n
20	25	35	40	44	55	38	99	10	65	50

O algoritmo faz o que promete?

Correção do algoritmo!

Relação invariante chave:

(i0) na linha 1 vale que: $A[1\mathinner{\ldotp\ldotp} j{-}1]$ é crescente.

1						j				n
20	25	35	40	44	55	38	99	10	65	50

Supondo que a invariante vale.
Correção do algoritmo é evidente.

No início da última iteração das linhas 1–7 tem-se que $j = n+1$. Da invariante concluí-se que $A[1\mathinner{\ldotp\ldotp} n]$ é crescente.

Mais invariantes

Na linha 4 vale que:

(i1) $A[1\,..\,i]$ e $A[i+2\,..\,j]$ são crescentes

(i2) $A[1\,..\,i] \leq A[i+2\,..\,j]$

(i3) $A[i+2\,..\,j] > chave$

(i4) $A[1\,..\,i] + A[i+2\,..\,j] + chave$ não muda

$chave$	1		i				j				n
38	20	25	35		40	44	55	99	10	65	50

Mais invariantes

Na linha 4 vale que:

(i1) $A[1\,..\,i]$ e $A[i+2\,..\,j]$ são crescentes

(i2) $A[1\,..\,i] \leq A[i+2\,..\,j]$

(i3) $A[i+2\,..\,j] > chave$

(i4) $A[1\,..\,i] + A[i+2\,..\,j] + chave$ não muda

$chave$	1		i				j				n
38	20	25	35		40	44	55	99	10	65	50

invariantes (i1),(i2) e (i3)
$+$ condição de parada do **enquanto** da linha 4
$+$ atribuição da linha 7 $\Rightarrow$ validade (i0)

Demonstre!

Correção de algoritmos iterativos

Estrutura "típica" de demonstrações da correção de algoritmos iterativos através de suas relações invariantes consiste em:

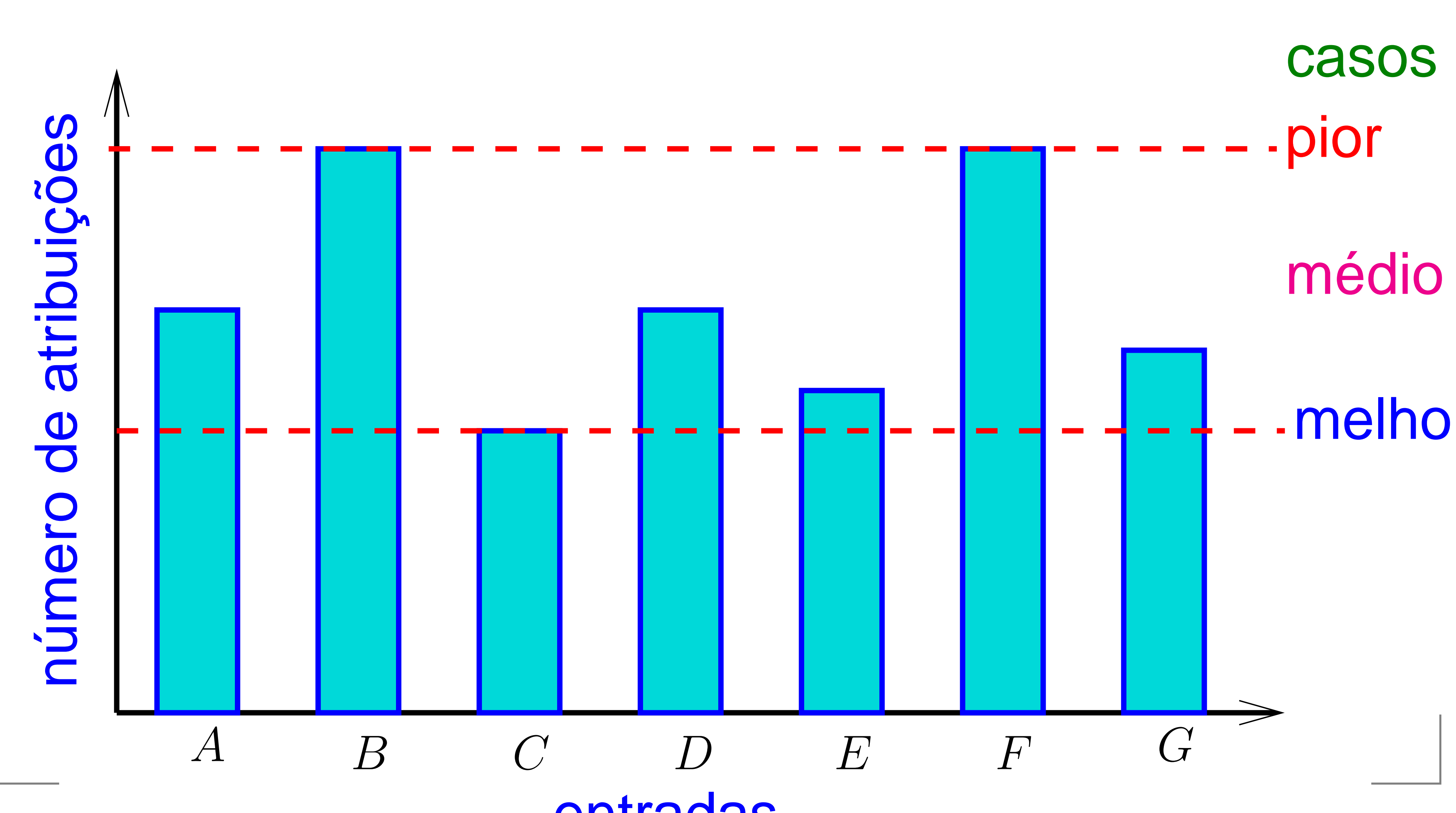
1. verificar que a relação vale no início da primeira iteração;
2. demonstrar que

   se a relação vale no início da iteração, então ela vale no final da iteração (com os papéis de alguns atores possivelmente trocados);
3. concluir que, se relação vale no início da última iteração, então a a relação junto com a condição de parada implicam na correção do algoritmo.

Quantas atribuições ($\leftarrow$) algoritmo faz?

Quantas atribuições ($\leftarrow$) algoritmo faz?

Número mínimo, médio ou máximo?
Melhor caso, caso médio, pior caso?



Quantas atribuições (←) algoritmo faz?

LINHAS 3–6 $(A, j, chave)$

3 $\quad i \leftarrow j - 1 \qquad \triangleright\ 2 \leq j \leq n$

4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**

5 $\qquad A[i+1] \leftarrow A[i]$

6 $\qquad i \leftarrow i - 1$

linha	atribuições (número máximo)
3	?
4	?
5	?
6	?

total ?

Quantas atribuições (←) algoritmo faz?

LINHAS 3–6 $(A, j, chave)$

3 $\quad i \leftarrow j - 1 \qquad \triangleright\ 2 \leq j \leq n$

4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**

5 $\qquad A[i+1] \leftarrow A[i]$

6 $\qquad i \leftarrow i - 1$

linha	atribuições (número máximo)
3	$= 1$
4	$= 0$
5	?
6	?

total ?

Quantas atribuições (←) algoritmo faz?

LINHAS 3–6 $(A, j, chave)$

3 $\quad i \leftarrow j - 1 \qquad \triangleright\ 2 \leq j \leq n$

4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**

5 $\qquad A[i+1] \leftarrow A[i]$

6 $\qquad i \leftarrow i - 1$

linha	atribuições (número máximo)
3	$= 1$
4	$= 0$
5	$\leq j - 1$
6	?

total ?

Quantas atribuições (←) algoritmo faz?

LINHAS 3–6 $(A, j, chave)$

3 $\quad i \leftarrow j - 1 \qquad \triangleright \; 2 \leq j \leq n$

4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**

5 $\qquad A[i+1] \leftarrow A[i]$

6 $\qquad i \leftarrow i - 1$

linha	atribuições (número máximo)
3	$= 1$
4	$= 0$
5	$\leq j - 1$
6	$\leq j - 1$

total $\leq 2j - 1 \leq 2n - 1$

Quantas atribuições ($\leftarrow$) algoritmo faz?

ORDENA-POR-INSERÇÃO (A, n)

1 **para** $j \leftarrow 2$ **até** n **faça** $\triangleright$ $j \leftarrow j + 1$ escondido

2 $chave \leftarrow A[j]$

3 LINHAS 3–6 $(A, j, chave)$

7 $A[i + 1] \leftarrow chave$

linha	atribuições (número máximo)
1	?
2	?
3–6	?
7	?

total ?

Quantas atribuições (←) algoritmo faz?

ORDENA-POR-INSERÇÃO (A, n)

1 **para** $j \leftarrow 2$ **até** n **faça** $\triangleright$ $j \leftarrow j + 1$ escondido

2 $chave \leftarrow A[j]$

3 LINHAS 3–6 $(A, j, chave)$

7 $A[i+1] \leftarrow chave$

linha	atribuições (número máximo)
1	$= n - 1 + 1$
2	$= n - 1$
3–6	$\leq (n-1)(2n-1)$
7	$= n - 1$

total $\leq 2n^2 - 1$

Análise mais fina

linha	atribuições (número máximo)
1	?
2	?
3	?
4	?
5	?
6	?
7	?
total	?

Análise mais fina

linha	atribuições (número máximo)
1	$= n - 1 + 1$
2	$= n - 1$
3	$= n - 1$
4	$= 0$
5	$\leq 1 + 2 + \cdots + (n{-}1) = n(n-1)/2$
6	$\leq 1 + 2 + \cdots + (n{-}1) = n(n-1)/2$
7	$= n - 1$
total	$\leq n^2 + 3n - 3$

$n^2 + 3n - 3$ **versus** n^2

n	$n^2 + 3n - 3$	n^2
1	1	1
2	7	4

$n^2 + 3n - 3$ **versus** n^2

n	$n^2 + 3n - 3$	n^2
1	1	1
2	7	4
3	15	9
10	127	100

$n^2 + 3n - 3$ **versus** n^2

n	$n^2 + 3n - 3$	n^2
1	1	1
2	7	4
3	15	9
10	127	100
100	10297	10000
1000	1002997	1000000

$n^2 + 3n - 3$ **versus** n^2

n	$n^2 + 3n - 3$	n^2
1	1	1
2	7	4
3	15	9
10	127	100
100	10297	10000
1000	1002997	1000000
10000	100029997	100000000
100000	10000299997	10000000000

n^2 domina os outros termos

Exercício 1.B

Se a execução de cada linha de código consome 1 unidade de tempo, qual o consumo total?

ORDENA-POR-INSERÇÃO (A, n)
1 **para** $j \leftarrow 2$ **até** n **faça**
2 $\quad chave \leftarrow A[j]$

3 $\quad i \leftarrow j - 1$
4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**
5 $\quad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca
6 $\quad\quad i \leftarrow i - 1$

7 $\quad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere

Solução

linha	todas as execuções da linha
1	$= \quad n$
2	$= \quad n-1$
3	$= \quad n-1$
4	$\leq \quad 2+3+\cdots+n \;=\; (n-1)(n+2)/2$
5	$\leq \quad 1+2+\cdots+(n-1) \;=\; n(n-1)/2$
6	$\leq \quad 1+2+\cdots+(n-1) \;=\; n(n-1)/2$
7	$= \quad n-1$
total	$\leq \quad (3/2)n^2 + (7/2)n - 4$

Exercício 1.C

Se a execução da linha i consome t_i unidades de tempo, para $i = 1, \ldots, 7$, qual o consumo total?

ORDENA-POR-INSERÇÃO (A, n)
1 **para** $j \leftarrow 2$ **até** n **faça**
2 $\quad chave \leftarrow A[j]$

3 $\quad i \leftarrow j - 1$
4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**
5 $\quad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca
6 $\quad\quad i \leftarrow i - 1$

7 $\quad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere

Solução para $t_i = 1$

linha	todas as execuções da linha
1	$= \quad n$
2	$= \quad n - 1$
3	$= \quad n - 1$
4	$\leq \quad 2 + 3 + \cdots + n \; = \; (n-1)(n+2)/2$
5	$\leq \quad 1 + 2 + \cdots + (n-1) \; = \; n(n-1)/2$
6	$\leq \quad 1 + 2 + \cdots + (n-1) \; = \; n(n-1)/2$
7	$= \quad n - 1$
total	$\leq \quad (3/2)n^2 + (7/2)n - 4$

Solução

linha	todas as execuções da linha	
1	$= \quad n$	$\times t_1$
2	$= \quad n-1$	$\times t_2$
3	$= \quad n-1$	$\times t_3$
4	$\leq \quad 2+3+\cdots+n \;=\; (n-1)(n+2)/2$	$\times t_4$
5	$\leq \quad 1+2+\cdots+(n-1) \;=\; n(n-1)/2$	$\times t_5$
6	$\leq \quad 1+2+\cdots+(n-1) \;=\; n(n-1)/2$	$\times t_6$
7	$= \quad n-1$	$\times t_7$

total $\leq$?

Solução

linha	todas as execuções da linha				
1	$=$	n			$\times t_1$
2	$=$	$n-1$			$\times t_2$
3	$=$	$n-1$			$\times t_3$
4	$\leq$	$2+3+\cdots+n$	$=$	$(n-1)(n+2)/2$	$\times t_4$
5	$\leq$	$1+2+\cdots+(n-1)$	$=$	$n(n-1)/2$	$\times t_5$
6	$\leq$	$1+2+\cdots+(n-1)$	$=$	$n(n-1)/2$	$\times t_6$
7	$=$	$n-1$			$\times t_7$

total $\leq$ $((t_4+t_5+t_6)/2)\times n^2$

$+$ $(t_1+t_2+t_3+t_4/2-t_5/2-t_6/2+t_7)\times n$

$-$ $(t_2+t_3+t_4+t_7)$

Solução

linha	todas as execuções da linha		
1	$=$	n	$\times t_1$
2	$=$	$n-1$	$\times t_2$
3	$=$	$n-1$	$\times t_3$
4	$\leq$	$2+3+\cdots+n \;=\; (n-1)(n+2)/2$	$\times t_4$
5	$\leq$	$1+2+\cdots+(n-1) \;=\; n(n-1)/2$	$\times t_5$
6	$\leq$	$1+2+\cdots+(n-1) \;=\; n(n-1)/2$	$\times t_6$
7	$=$	$n-1$	$\times t_7$
total	$\leq$	$c_2 \times n^2 + c_1 \times n + c_0$	

c_2, c_1, c_0 são constantes que dependem da máquina.

n^2 é para sempre! Está nas entranhas do algoritmo!

Exercícios

Exercício 1.D (bom!)
Em função de n, quanto vale S no fim do seguinte algoritmo? Dê também uma cota superior simples e próxima.

1 $S \leftarrow 0$
2 $i \leftarrow n$
3 **enquanto** $i > 0$ **faça**
4   **para** $j \leftarrow 1$ **até** i **faça**
5         $S \leftarrow S + 1$
6     $i \leftarrow \lfloor i/2 \rfloor$

Exercício 1.E
Prove a invariante do algoritmo ORDENA-POR-INSERÇÃO.

Exercício 1.F
Quantas comparações faz o algoritmo ORDENA-POR-INSERÇÃO, no pior caso, ao receber um vetor $A[1\,..\,n]$?

Exercício 1.G
Quantas atribuições faz o algoritmo abaixo?

$s \leftarrow 0$
para $i \leftarrow 1$ **até** n **faça**
    $s \leftarrow s + i$
devolva s

Escreva um algoritmo melhorado que produza o mesmo efeito com menos atribuições.

Exercícios

Exercício 1.H (AU 3.7.1)
O algoritmo abaixo opera sobre um vetor $A[1\,..\,n]$. Quantas atribuições ele faz no pior caso? Quantas comparações?

1 $s \leftarrow 0$
2 **para** $i \leftarrow 1$ **até** n **faça**
3 $\quad s \leftarrow s + A[i]$
4 $m \leftarrow s/n$
5 $k \leftarrow 1$
6 **para** $i \leftarrow 2$ **até** n **faça**
7 $\quad$ **se** $(A[i] - m)^2 < (A[k] - m)^2$
8 $\quad\quad$ **então** $k \leftarrow i$
9 **devolva** k

Exercício 1.I (AU 3.7.2)
O fragmento abaixo opera sobre uma matriz $A[1\,..\,n, 1\,..\,n]$. Quantas atribuições faz?

1 **para** $i \leftarrow 1$ **até** $n - 1$ **faça**
2 $\quad$ **para** $j \leftarrow i + 1$ **até** n **faça**
3 $\quad\quad$ **para** $k \leftarrow i$ **até** n **faça**
4 $\quad\quad\quad A[j,k] \leftarrow A[j,k] - A[i,k] \cdot A[j,i]/A[i,i]$

Exercícios

Exercício 1.J (AU 3.7.3*)

Quantas atribuições faz o algoritmo?

SUMPOWERSOFTWO (n)

1 $s \leftarrow 0$
2 **para** $i \leftarrow 1$ **até** n **faça**
3 $\quad j \leftarrow i$
4 $\quad$ **enquanto** $2\lfloor j/2 \rfloor = j$ **faça**
5 $\quad\quad j \leftarrow \lfloor j/2 \rfloor$
6 $\quad\quad s \leftarrow s + 1$
7 **devolva** s

Chão, teto, log etc

CLRS 3.2, A.1

AU 2.9

Definições

$\lfloor x \rfloor$:= inteiro i tal que $i \leq x < i + 1$

$\lceil x \rceil$:= inteiro j tal que $j - 1 < x \leq j$

Diz-se que

$\lfloor x \rfloor$ é o chão de x

$\lceil x \rceil$ é o teto de x

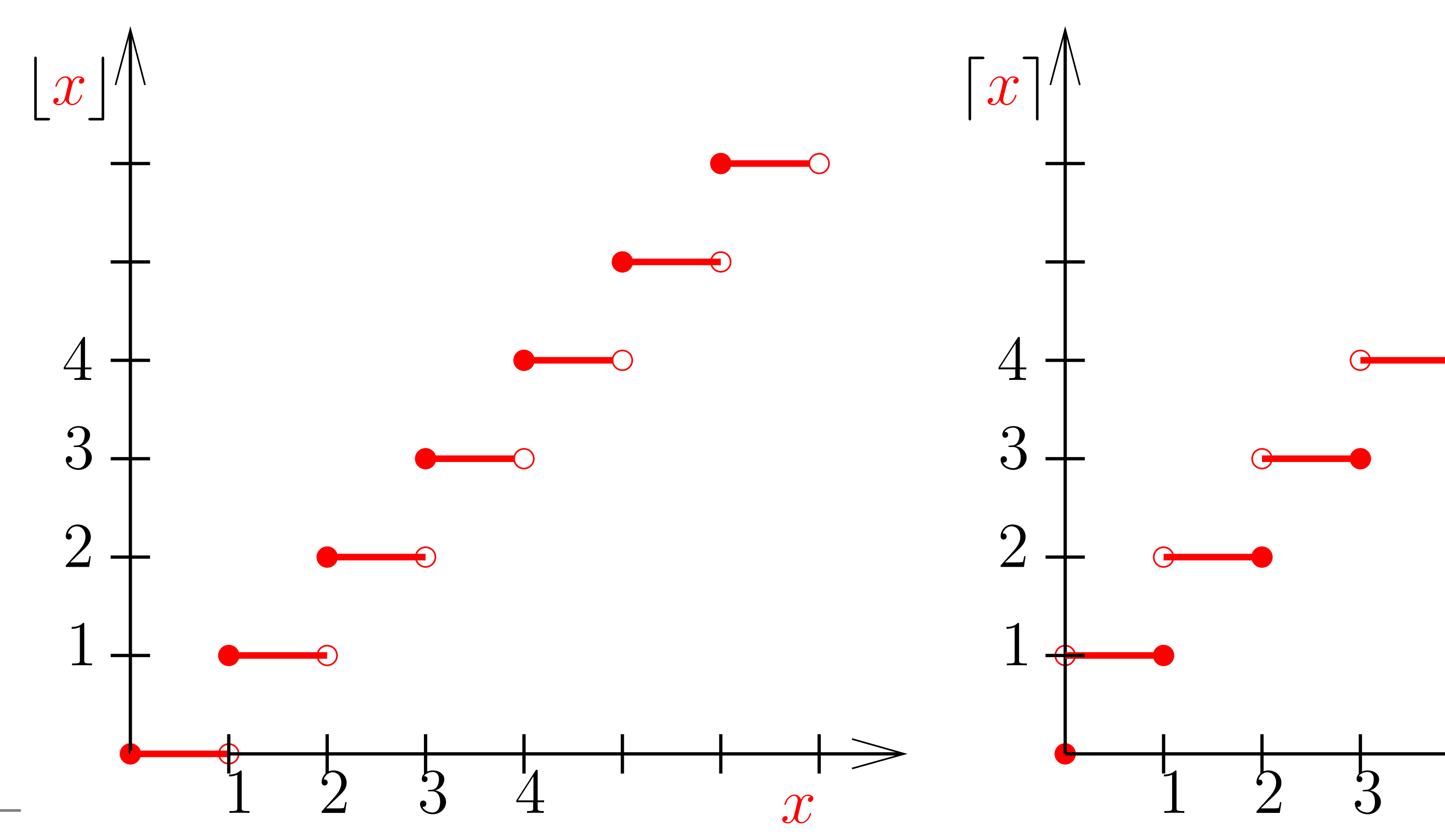
Exercício A1.A

Desenhe os gráficos das funções $\lfloor x \rfloor$ e $\lceil x \rceil$ para x não-negativo.

Exercício A1.A

Desenhe os gráficos das funções $\lfloor x \rfloor$ e $\lceil x \rceil$ para x não-negativo.

Solução



Exercícios

Exercício A1.B

Mostre que para qualquer inteiro $n \geq 1$.

$$\frac{n-1}{2} \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2} \qquad \text{e} \qquad \frac{n}{2} \leq \left\lceil \frac{n}{2} \right\rceil \leq \frac{n+1}{2}$$

para qualquer inteiro $n \geq 1$.

Exercício A1.C

É verdade que $\lfloor x \rfloor + \lfloor y \rfloor = \lfloor x + y \rfloor$ para quaisquer x e y? É verdade que $\lceil x \rceil + \lceil y \rceil = \lceil x + y \rceil$ para quaisquer x e y?

Exercício A1.D

Desenhe os gráficos das funções $\lg x$ e 2^x para x inteiro não-negativo.

Exercícios

Exercício A1.E
Explique o significado da expressão $\log_{3/2} n$.

Exercício A1.F
Calcule $5^{\log_5 n}$, $\log_3 3^n$ e $\lg 2^n$.

Exercício A1.G
Qual a relação entre $\log_8 n$ e $\log_2 n$?

Exercício A1.H
Se $i := \lfloor \lg n \rfloor$, qual a relação entre n e 2^i ?
Se $j := \lceil \lg n \rceil$, qual a relação entre n e 2^j ?

Exercício A1.I
É verdade que $\lfloor \lg n \rfloor + 1 = \lceil \lg(n+1) \rceil$ para todo inteiro $n \geq 1$?

Exercício A1.J
Escreva um algoritmo que calcule $\lfloor \lg n \rfloor$.

Exercício A1.L
Mostre que para qualquer número real x tem-se $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$.

Exercício A1.M
Mostre que $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$ para todo inteiro positivo n.

Mais exercícios

Exercício A1.N

Mostre que $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor$, com igualdade se e somente se $x + y - 1 < \lfloor x \rfloor + \lfloor y \rfloor$. Encontre uma fórmula análoga para $\lceil \cdot \rceil$.

Exercício A1.O

Se c é inteiro e x é racional, é verdade que $\lceil cx \rceil = c\lceil x \rceil$?

Exercício A1.P

Use a notação $\lfloor \ \rfloor$ para representar o resto da divisão de n por 7.

Exercício A1.Q

É verdade que $\lceil 2\lceil 2n/3 \rceil/3 \rceil = \lceil 4n/9 \rceil$? É verdade que $\lfloor 2\lfloor 2n/3 \rfloor/3 \rfloor = \lfloor 4n/9 \rfloor$?

Exercício A1.R

É verdade que $\lfloor \lfloor n/2 \rfloor/2 \rfloor = \lfloor n/4 \rfloor$?

Exercício A1.S

Se n, a, b são inteiros positivos, é verdade que $\lfloor \lfloor n/a \rfloor/b \rfloor = \lfloor n/ab \rfloor$?

Exercício A1.T

É verdade que $\lfloor \lg n \rfloor \geq \lg(n-1)$ para todo inteiro $n \geq 2$? É verdade que $\lceil \lg n \rceil \leq \lg(n+1)$ para todo inteiro $n \geq 1$?

Mais exercícios ainda

Exercício A1.U

Prove que para qualquer número racional $x > 1$ tem-se

$$\lfloor \lg x \rfloor \leq \lg \lfloor x \rfloor \leq \lg x \leq \lg \lceil x \rceil \leq \lceil \lg x \rceil$$

Exercício A1.V

Quanto vale $1/2 + 1/4 + 1/8 + \cdots + 1/2^n + \cdots$?

Melhores momentos

AULA PASSADA

Chão e teto

$\lfloor x \rfloor$:= inteiro i tal que $i \le x < i + 1$

$\lceil x \rceil$:= inteiro j tal que $j - 1 < x \le j$

Exercício A1.B

Mostre que

$$\frac{n-1}{2} \le \left\lfloor \frac{n}{2} \right\rfloor \le \frac{n}{2} \qquad \text{e} \qquad \frac{n}{2} \le \left\lceil \frac{n}{2} \right\rceil \le \frac{n+1}{2}$$

para qualquer inteiro $n \ge 1$.

Ordenação por inserção

Algoritmo rearranja $A[1\,..\,n]$ em ordem crescente.

ORDENA-POR-INSERÇÃO (A, n)
1 **para** $j \leftarrow 2$ **até** n **faça**
2 $\quad chave \leftarrow A[j]$

3 $\quad i \leftarrow j - 1$
4 $\quad$ **enquanto** $i \geq 1$ **e** $A[i] > chave$ **faça**
5 $\quad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca
6 $\quad\quad i \leftarrow i - 1$

7 $\quad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere

Invariantes

Correção de algoritmos iterativos e invariantes

Relação invariante chave:

(i0) na linha 1 vale que: $A[1\,..\,j-1]$ é crescente.

1						j				n
20	25	35	40	44	55	38	99	10	65	50

Supondo que a invariante vale. Correção do algoritmo é evidente!

No início da última iteração das linhas 1–7 tem-se que $j = n+1$. Da invariante concluí-se que $A[1\,..\,n]$ é crescente.

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha	todas as execuções da linha
1	$= \quad n$
2	$= \quad n-1$
3	$= \quad n-1$
4	$\leq \quad 2+3+\cdots+n \ = \ (n-1)(n+2)/2$
5	$\leq \quad 1+2+\cdots+(n-1) \ = \ n(n-1)/2$
6	$\leq \quad 1+2+\cdots+(n-1) \ = \ n(n-1)/2$
7	$= \quad n-1$
total	$\leq \quad (3/2)n^2+(7/2)n-4$

$(3/2)n^2 + (7/2)n - 4$ **versus** $(3/2)n^2$

n	$(3/2)n^2 + (7/2)n - 4$	$(3/2)n^2$
64	6364	6144
128	25020	24576
256	99196	98304
512	395004	393216
1024	1576444	1572864
2048	6298620	6291456
4096	25180156	25165824
8192	100691964	100663296
16384	402710524	402653184
32768	1610727420	1610612736

$(3/2)n^2$ domina os outros termos

Consumo de tempo

Se a execução da linha i consome t_i unidades de tempo, para $i = 1, \ldots, 7$, o consumo totalde tempo é:

linha		todas as execuções da linha			
1	$=$	n			$\times t_1$
2	$=$	$n - 1$			$\times t_2$
3	$=$	$n - 1$			$\times t_3$
4	$\leq$	$2 + 3 + \cdots + n$	$=$	$(n-1)(n+2)/2$	$\times t_4$
5	$\leq$	$1 + 2 + \cdots + (n-1)$	$=$	$n(n-1)/2$	$\times t_5$
6	$\leq$	$1 + 2 + \cdots + (n-1)$	$=$	$n(n-1)/2$	$\times t_6$
7	$=$	$n - 1$			$\times t_7$

total $\leq$ $((t_4 + t_5 + t_6)/2) \times n^2$

$+ \quad (t_1 + t_2 + t_3 + t_4/2 - t_5/2 - t_6/2 + t_7) \times n$

$- \quad (t_2 + t_3 + t_4 + t_7)$

Consumo de tempo

linha	todas as execuções da linha			
1	$=$	n		$\times t_1$
2	$=$	$n-1$		$\times t_2$
3	$=$	$n-1$		$\times t_3$
4	$\leq$	$2+3+\cdots+n$	$=\ (n-1)(n+2)/2$	$\times t_4$
5	$\leq$	$1+2+\cdots+(n-1)$	$=\ n(n-1)/2$	$\times t_5$
6	$\leq$	$1+2+\cdots+(n-1)$	$=\ n(n-1)/2$	$\times t_6$
7	$=$	$n-1$		$\times t_7$

total $\leq$ $c_2 \times n^2 + c_1 \times n + c_0$

c_2, c_1, c_0 são constantes que dependem da máquina.

n^2 é para sempre! Está nas entranhas do algoritmo!

Notação assintótica

CLRS 3.1

AU 3.4, p.96–100 (muito bom!)

Funções de n

$5n^2 - 9n$ $\quad$ $4n + 8$ $\quad$ $\lfloor n/3 \rfloor + 4$ $\quad$ $2\sqrt{n} + 7$

2^{n-3} $\quad$ $\lg n \quad (= \log_2 n)$

Qual é maior: $n^2 - 9$ ou $4n + 8$?

Funções de n

$5n^2 - 9n$ $\quad$ $4n + 8$ $\quad$ $\lfloor n/3 \rfloor + 4$ $\quad$ $2\sqrt{n} + 7$

2^{n-3} $\quad$ $\lg n$ (= $\log_2 n$)

Qual é maior: $n^2 - 9$ ou $4n + 8$?

Depende do valor de n!

Funções de n

$5n^2 - 9n$ $\quad$ $4n + 8$ $\quad$ $\lfloor n/3 \rfloor + 4$ $\quad$ $2\sqrt{n} + 7$

2^{n-3} $\quad$ $\lg n$ $\quad$ $(= \log_2 n)$

Qual é maior: $n^2 - 9$ ou $4n + 8$?

Depende do valor de n!

Qual cresce mais?

Funções de n

$5n^2 - 9n$ $\quad$ $4n + 8$ $\quad$ $\lfloor n/3 \rfloor + 4$ $\quad$ $2\sqrt{n} + 7$

2^{n-3} $\quad$ $\lg n \quad (= \log_2 n)$

Qual é maior: $n^2 - 9$ ou $4n + 8$?

Depende do valor de n!

Qual cresce mais?

Comparação assintótica, ou seja, para n ENORME.

Funções de n

$5n^2 - 9n$ $\quad 4n + 8$ $\quad \lfloor n/3 \rfloor + 4$ $\quad 2\sqrt{n} + 7$

2^{n-3} $\quad \lg n \quad (= \log_2 n)$

Qual é maior: $n^2 - 9$ ou $4n + 8$?

Depende do valor de n!

Qual cresce mais?

Comparação assintótica, ou seja, para n ENORME.

$\lg n$ $\quad 2\sqrt{n} + 7$ $\quad \lfloor n/3 \rfloor + 4$ $\quad 4n + 8$ $\quad 5n^2 - 9n$

2^{n-3}

Notação O

Intuitivamente...

$$
\begin{aligned}
\mathrm{O}(f(n)) &\approx \text{funções que não crescem mais rápido que } f(n) \\
&\approx \text{funções menores ou iguais a um múltiplo de } f(n)
\end{aligned}
$$

n^2 $(3/2)n^2$ $9999n^2$ $n^2/1000$ etc.

crescem todas com a mesma velocidade

Notação O

Intuitivamente...

$O(f(n)) \approx$ funções que não crescem mais rápido que $f(n)$

$\approx$ funções menores ou iguais a um múltiplo de $f(n)$

$n^2 \qquad (3/2)n^2 \qquad 9999n^2 \qquad n^2/1000$ etc.

crescem todas com a mesma velocidade

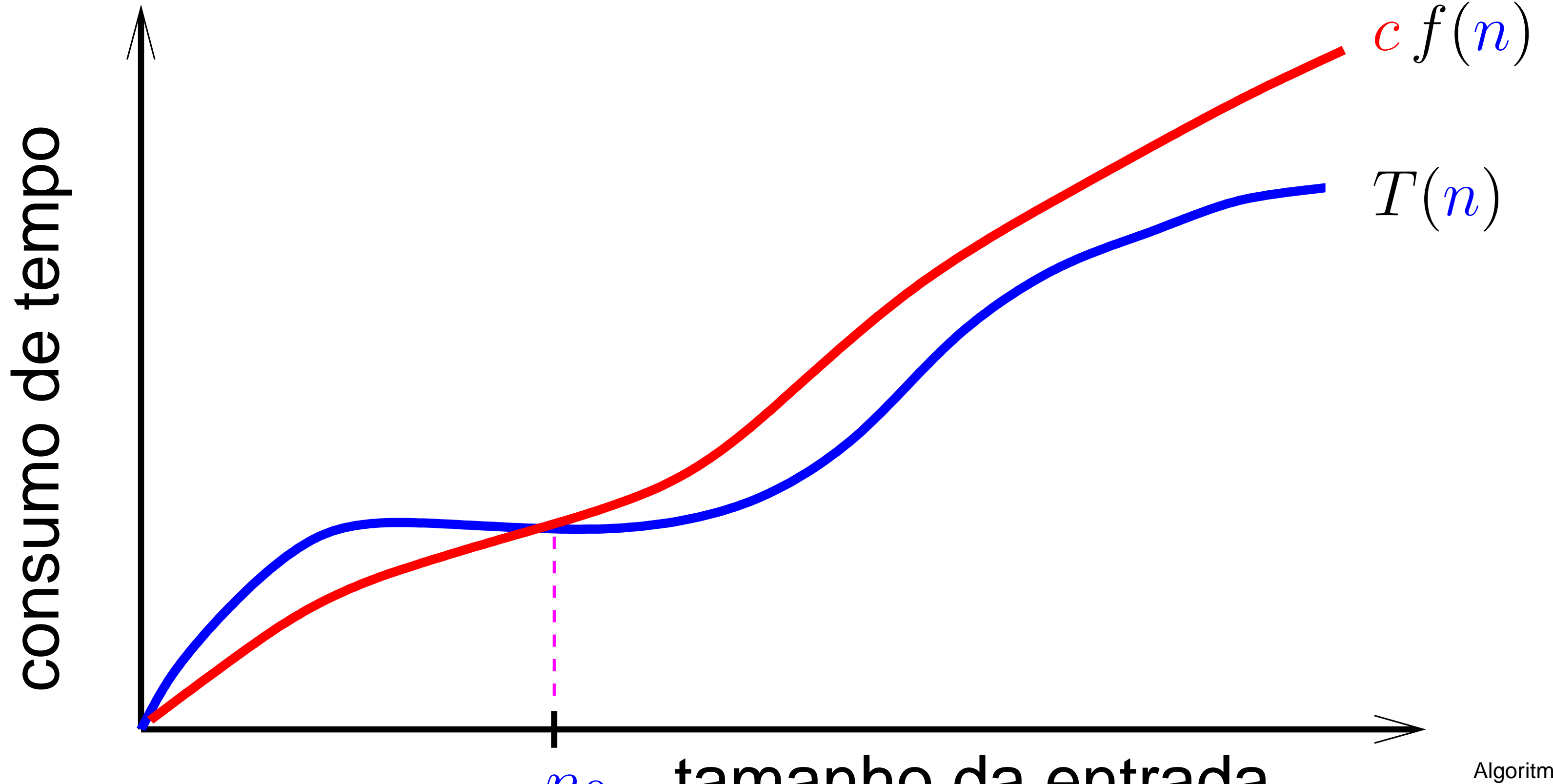
- $n^2 + 99n$ é $O(n^2)$
- $33n^2$ é $O(n^2)$
- $9n + 2$ é $O(n^2)$
- $0{,}00001n^3 - 200n^2$ não é $O(n^2)$

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros nos reais. Dizemos que $T(n)$ **é** $\mathrm{O}(f(n))$ se existem constantes positivas c e n_0 tais que

$$T(n) \;\leq\; c\, f(n)$$

para todo $n \geq n_0$.

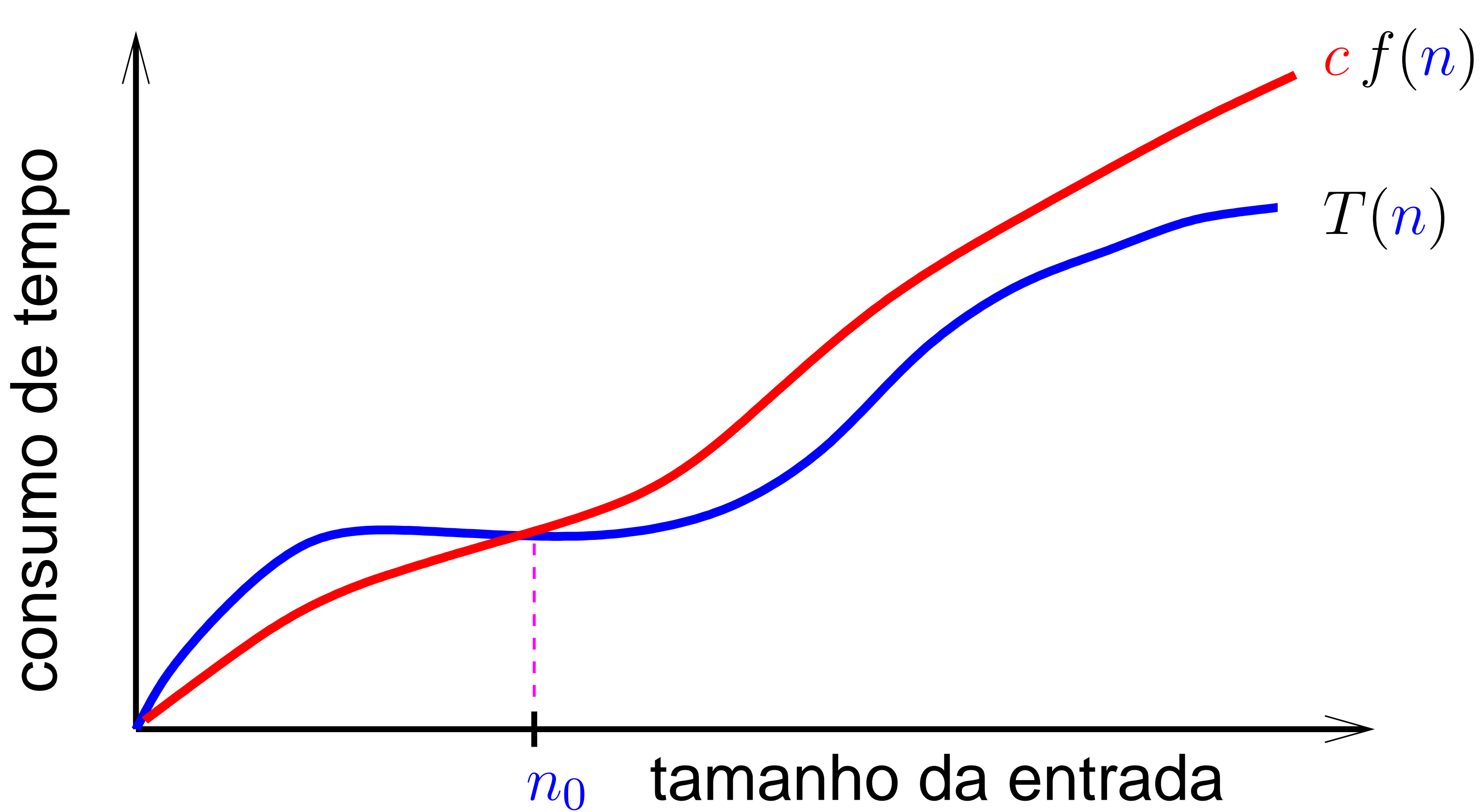


Mais informal

$T(n)$ é $\mathrm{O}(f(n))$ se existe $c > 0$ tal que

$$T(n) \leq c\, f(n)$$

para todo n suficientemente GRANDE.



Exemplos

$T(n)$ é $\mathrm{O}(f(n))$ lê-se "$T(n)$ é O de $f(n)$" ou
"$T(n)$ é da ordem de $f(n)$"

Exemplos

$T(n)$ é $\mathrm{O}(f(n))$ lê-se "$T(n)$ é O de $f(n)$" ou
"$T(n)$ é da ordem de $f(n)$"

Exemplo 1

Se $T(n) \leq 500f(n)$ para todo $n \geq 10$, então $T(n)$ é $\mathrm{O}(f(n))$.

Exemplos

$T(n)$ é $\mathrm{O}(f(n))$ lê-se "$T(n)$ é O de $f(n)$" ou
"$T(n)$ é da ordem de $f(n)$"

Exemplo 1

Se $T(n) \leq 500f(n)$ para todo $n \geq 10$, então $T(n)$ é $\mathrm{O}(f(n))$.

Prova: Aplique a definição com $c = 500$ e $n_0 = 10$.

Exemplos

$T(n)$ é $\mathrm{O}(f(n))$ lê-se "$T(n)$ é O de $f(n)$" ou
"$T(n)$ é da ordem de $f(n)$"

Exemplo 1

Se $T(n) \leq 500 f(n)$ para todo $n \geq 10$, então $T(n)$ é $\mathrm{O}(f(n))$.

Prova: Aplique a definição com $c = 500$ e $n_0 = 10$.

Exemplo 2

$10n^2$ é $\mathrm{O}(n^3)$.

Exemplos

$T(n)$ é $\mathrm{O}(f(n))$ lê-se "$T(n)$ é O de $f(n)$" ou
"$T(n)$ é da ordem de $f(n)$"

Exemplo 1

Se $T(n) \leq 500 f(n)$ para todo $n \geq 10$, então $T(n)$ é $\mathrm{O}(f(n))$.

Prova: Aplique a definição com $c = 500$ e $n_0 = 10$.

Exemplo 2

$10n^2$ é $\mathrm{O}(n^3)$.

Prova: Para $n \geq 0$, temos que $0 \leq 10n^2 \leq 10\, n^3$.

Outra prova: Para $n \geq 10$, temos $0 \leq 10n^2 \leq n \times n^2 = 1\, n^3$.

Mais exemplos

Exemplo 3

$\lg n$ é $\mathrm{O}(n)$.

Mais exemplos

Exemplo 3

$\lg n$ é $\mathrm{O}(n)$.

Prova: Para $n \geq 1$, tem-se que $\lg n \leq 1\, n$.

Mais exemplos

Exemplo 3

$\lg n$ é $\mathrm{O}(n)$.

Prova: Para $n \geq 1$, tem-se que $\lg n \leq 1\, n$.

Exemplo 4

$20n^3 + 10n \log n + 5$ é $\mathrm{O}(n^3)$.

Mais exemplos

Exemplo 3

$\lg n$ é $\mathrm{O}(n)$.

Prova: Para $n \geq 1$, tem-se que $\lg n \leq 1\, n$.

Exemplo 4

$20n^3 + 10n \log n + 5$ é $\mathrm{O}(n^3)$.

Prova: Para $n \geq 1$, tem-se que

$$20n^3 + 10n \lg n + 5 \leq 20n^3 + 10n^3 + 5n^3 = 35\, n^3.$$

Outra prova: Para $n \geq 10$, tem-se que

$$20n^3 + 10n \lg n + 5 \leq 20n^3 + n\, n \lg n + n \leq 20n^3 + n^3 + n^3 = 22\, n^3.$$

Mais exemplos ainda

Exemplo 5

$3 \lg n + \lg \lg n$ é $O(\lg n)$.

Mais exemplos ainda

Exemplo 5

$3 \lg n + \lg \lg n$ é $\mathrm{O}(\lg n)$.

Prova: Para $n \geq 2$, tem-se que

$$3 \lg n + \lg \lg n \leq 3 \lg n + \lg n = 4 \lg n.$$

[Note que $\lg \lg n$ não é definida para $n = 1$.]

Mais exemplos ainda

Exemplo 5

$3\lg n + \lg\lg n$ é $\mathrm{O}(\lg n)$.

Prova: Para $n \geq 2$, tem-se que

$$3\lg n + \lg\lg n \leq 3\lg n + \lg n = 4\lg n.$$

[Note que $\lg\lg n$ não é definida para $n = 1$.]

Exemplo 6

10^{100} é $\mathrm{O}(1)$.

Mais exemplos ainda

Exemplo 5

$3\lg n + \lg\lg n$ é $\mathrm{O}(\lg n)$.

Prova: Para $n \geq 2$, tem-se que

$$3\lg n + \lg\lg n \leq 3\lg n + \lg n = 4\lg n.$$

[Note que $\lg\lg n$ não é definida para $n = 1$.]

Exemplo 6

10^{100} é $\mathrm{O}(1)$.

Prova: Para $n \geq 1$, tem-se que

$$10^{100} = 10^{100}\, n^0 = 10^{100} \times 1.$$

[Note que n não precisa aparecer, já que estamos lidando com funções constantes.]

Uso da notação O

$\mathrm{O}(f(n)) = \{T(n) : \text{existem } c \text{ e } n_0 \text{ tq } T(n) \leq c f(n), n \geq n_0\}$

"$T(n)$ é $\mathrm{O}(f(n))$" deve ser entendido como "$T(n) \in \mathrm{O}(f(n))$".

"$T(n) = \mathrm{O}(f(n))$" deve ser entendido como "$T(n) \in \mathrm{O}(f(n))$".

"$T(n) \leq \mathrm{O}(f(n))$" é feio.

"$T(n) \geq \mathrm{O}(f(n))$" não faz sentido!

"$T(n)$ é $g(n) + \mathrm{O}(f(n))$" significa que existe constantes positivas c e n_0 tais que

$$T(n) \leq g(n) + c\, f(n)$$

para todo $n \geq n_0$.

Nomes de classes O

classe	nome
$\mathrm{O}(1)$	constante
$\mathrm{O}(\lg n)$	logarítmica
$\mathrm{O}(n)$	linear
$\mathrm{O}(n \lg n)$	$n \log n$
$\mathrm{O}(n^2)$	quadrática
$\mathrm{O}(n^3)$	cúbica
$\mathrm{O}(n^k)$ com $k >= 1$	polinomial
$\mathrm{O}(2^n)$	exponencial
$\mathrm{O}(a^n)$ com $a > 1$	exponencial

Exercícios

Exercício 2.A

Prove que $n^2 + 10n + 20 = \mathrm{O}(n^2)$

Exercício 2.B

Prove que $300 = \mathrm{O}(1)$

Exercício 2.C

Prove que $\lceil n/3 \rceil = \mathrm{O}(n)$

É verdade que $n = \mathrm{O}(\lfloor n/3 \rfloor)$?

Exercício 2.D

Prove que $\lg n = \mathrm{O}(\log_{10} n)$

Exercício 2.E

Prove que $n = \mathrm{O}(2^n)$

Exercício 2.F

Prove que $\lg n = \mathrm{O}(n)$

Exercício 2.G

Prove que $n/1000$ não é $\mathrm{O}(1)$

Exercício 2.H

Prove que $\frac{1}{2} n^2$ não é $\mathrm{O}(n)$

Mais exercícios

Exercício 2.I

Suponha T definida para $n = 0, 1, \ldots$

Se $T(n) = \mathrm{O}(1)$, mostre que existe c' tal que $T(n) \leq c'$ para todo $n \geq 0$.

Se $T(n) = \mathrm{O}(n)$, mostre que existe c' tal que $T(n) \leq c'n$ para todo $n \geq 1$.

Exercício 2.J

Prove que $n^2 + 999n + 9999 = \mathrm{O}(n^2)$.

Exercício 2.K

Prove que $\frac{1}{2}n(n+1) \ = \ \mathrm{O}(n^2)$.

Exercício 2.L

É verdade que $\frac{1}{100}n^2 - 999n - 9999 = \mathrm{O}(n)$? Justifique.

Exercício 2.M

Suponha que $f(n) = n^2$ quando n é par e $f(n) = n^3$ quando n é ímpar.

É verdade que $f(n) = \mathrm{O}(n^2)$?

É verdade que $f(n) = \mathrm{O}(n^3)$?

É verdade que $n^2 = \mathrm{O}(f(n))$?

É verdade que $n^3 = \mathrm{O}(f(n))$?

Mais exercícios ainda

Exercício 2.N
É verdade que $n^2 = \mathrm{O}(2^n)$?

Exercício 2.O
É verdade que $\lg n = \mathrm{O}(\sqrt{n})$?

Exercício 2.P
Suponha $f(n) = 64n \lg n$ e $g(n) = 8n^2$, com n inteiro positivo.
Para que valores de n temos $f(n) \leq g(n)$?

Exercício 2.Q (bom!)
Suponha T e f definidas para $n = 1, 2, \ldots$ Mostre que se $T(n) = \mathrm{O}(f(n))$ e $f(n) > 0$ para $n \geq 1$ então existe c' tal que $T(n) \leq c' f(n)$ para todo $n \geq 1$.

Exercício 2.R (bom!)
Faz sentido dizer "$T(n) = \mathrm{O}(n^2)$ para $n \geq 3$"?

Mais exercícios ainda ainda

Exercício 2.S

É verdade que $2^n = \mathrm{O}(n)$?
É verdade que $n = \mathrm{O}(\lg n)$?
Justifique.

Exercício 2.T

É verdade que $n + \sqrt{n}$ é $\mathrm{O}(n)$?
É verdade que n é $\mathrm{O}(\sqrt{n})$?
É verdade que $n^{2/3}$ é $\mathrm{O}(\sqrt{n})$?
É verdade que $\sqrt{n} + 1000$ é $\mathrm{O}(n)$?

Exercício 2.U

É verdade que $\lg n = \mathrm{O}(n^{1/2})$?
É verdade que $\sqrt{n} = \mathrm{O}(\lg n)$?
É verdade que $\lg n = \mathrm{O}(n^{1/3})$?
Justifique. (Sugestão: prove, por indução, que $\lg x \leq x$ para todo número real $x \geq 1$.)

Exercício 2.V

É verdade que $\lceil \lg n \rceil = \mathrm{O}(\lg n)$?

Análise com notação O

CLRS 2.1–2.2

AU 3.3, 3.6 (muito bom)

Ordenação por inserção

Algoritmo rearranja $A[p\,..\,r]$ em ordem crescente

ORDENA-POR-INSERÇÃO (A, p, r)
1 **para** $j \leftarrow p + 1$ **até** r **faça**
2 $\quad chave \leftarrow A[j]$

3 $\quad i \leftarrow j - 1$
4 $\quad$ **enquanto** $i \geq p$ **e** $A[i] > chave$ **faça**
5 $\quad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca
6 $\quad\quad i \leftarrow i - 1$

7 $\quad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere

Quanto tempo o algoritmo consome?

Ordenação por inserção

Algoritmo rearranja $A[p\,..\,r]$ em ordem crescente

ORDENA-POR-INSERÇÃO (A, p, r)
1 **para** $j \leftarrow p+1$ **até** r **faça**
2 $\quad chave \leftarrow A[j]$

3 $\quad i \leftarrow j-1$
4 $\quad$ **enquanto** $i \geq p$ **e** $A[i] > chave$ **faça**
5 $\quad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca
6 $\quad\quad i \leftarrow i-1$

7 $\quad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere

Quanto tempo o algoritmo consome?

"Tamanho" do problema: $n := r - p + 1$

Consumo de tempo

linha	consumo de todas as execuções da linha
1	?
2	?
3	?
4	?
5	?
6	?
7	?
total	?

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\mathrm{O}(n)$
2	$\mathrm{O}(n)$
3	$\mathrm{O}(n)$
4	$n\mathrm{O}(n) = \mathrm{O}(n^2)$
5	$n\mathrm{O}(n) = \mathrm{O}(n^2)$
6	$n\mathrm{O}(n) = \mathrm{O}(n^2)$
7	$\mathrm{O}(n)$
total	$\mathrm{O}(3n^2 + 4n) = \mathrm{O}(n^2)$

Justificativa

Bloco de linhas 4–6 é executado $\leq n$ vezes;
cada execução consome $O(n)$;
todas juntas consomem $nO(n)$.

Justificativa

Bloco de linhas 4–6 é executado $\leq n$ vezes;
cada execução consome $\mathrm{O}(n)$;
todas juntas consomem $n\mathrm{O}(n)$.

Êpa!

Quem garante que $n\mathrm{O}(n) = \mathrm{O}(n^2)$?
Quem garante que $\mathrm{O}(n^2) + \mathrm{O}(n^2) + \mathrm{O}(n^2) = \mathrm{O}(3n^2)$?
Quem garante que $\mathrm{O}(3n^2 + 4n) = \mathrm{O}(n^2)$?
Veja exercícios de Mais notação O.

$n\mathrm{O}(n) = \mathrm{O}(n^2)$

"$n\mathrm{O}(n) = \mathrm{O}(n^2)$" significa "$n\mathrm{O}(n) \subseteq \mathrm{O}(n^2)$".

Ou seja, se $T(n)$ é $\mathrm{O}(n)$, então $n\,T(n)$ é $\mathrm{O}(n^2)$.

$n\mathrm{O}(n) = \mathrm{O}(n^2)$

"$n\mathrm{O}(n) = \mathrm{O}(n^2)$" significa "$n\mathrm{O}(n) \subseteq \mathrm{O}(n^2)$".
Ou seja, se $T(n)$ é $\mathrm{O}(n)$, então $n\,T(n)$ é $\mathrm{O}(n^2)$.

De fato, se $T(n)$ é $\mathrm{O}(n)$ então existem constantes, digamos 10 e 10^{100}, tais que

$$T(n) \;\leq\; 10\,n$$

para todo $n \geq 10^{100}$. Desta forma,

$$n\,T(n) \;\leq\; n\,10\,n \leq 10\,n^2$$

para todo $n \geq 10^{100}$. Logo $n\,T(n)$ é $\mathrm{O}(n^2)$.

$n\mathrm{O}(n) = \mathrm{O}(n^2)$

"$n\mathrm{O}(n) = \mathrm{O}(n^2)$" significa "$n\mathrm{O}(n) \subseteq \mathrm{O}(n^2)$".
Ou seja, se $T(n)$ é $\mathrm{O}(n)$, então $n\,T(n)$ é $\mathrm{O}(n^2)$.

De fato, se $T(n)$ é $\mathrm{O}(n)$ então existem constantes positivas c e n_0, tais que

$$T(n) \;\leq\; c\,n$$

para todo $n \geq n_0$. Desta forma,

$$n\,T(n) \;\leq\; n\,c\,n \leq c\,n^2$$

para todo $n \geq n_0$. Logo $n\,T(n)$ é $\mathrm{O}(n^2)$.

Conclusão

O algoritmo ORDENA-POR-INSERÇÃO consome $\mathrm{O}(n^2)$ unidades de tempo.

Notação O cai como uma luva!

Exercício

Exercício 3.A

Problema: rearranjar um vetor $A[p \,..\, r]$ em ordem crescente. Escreva um algoritmo "de seleção" para o problema. Analise a correção do algoritmo (ou seja, encontre e prove as invariantes apropriadas). Analise o consumo de tempo do algoritmo; use notação O.

Mais notação assintótica

CLRS 4.1
AU 4.5, p.101–108

Exercícios

Exercício 4.A
Interprete e prove a afirmação $\mathrm{O}(n^2) + \mathrm{O}(n^2) + \mathrm{O}(n^2) = \mathrm{O}(3n^2)$.

Exercício 4.B
Interprete e prove a afirmação $n\mathrm{O}(n) = \mathrm{O}(n^2)$.

Exercício 4.C
Interprete e prove a afirmação $\mathrm{O}(3n^2 + 4n) = \mathrm{O}(n^2)$.

Exercício 4.D (propriedade transitiva)
Suponha $T(n) = \mathrm{O}(f(n))$ e $f(n) = \mathrm{O}(g(n))$.
Mostre que $T(n) = \mathrm{O}(g(n))$.
Dê um exemplo interessante.

Exercício 4.E (regra da soma, caso especial)
Suponha que $T(n) = \mathrm{O}(f(n))$ e mostre que $T(n) + f(n) = \mathrm{O}(f(n))$.
Dê um exemplo interessante.

Exercício 4.E' (regra da soma, geral)
Suponha $T_1(n) = \mathrm{O}(f_1(n))$ e $T_2(n) = \mathrm{O}(f_2(n))$. Se $f_1(n) = \mathrm{O}(f_2(n))$, mostre que $T_1(n) + T_2(n) = \mathrm{O}(f_2(n))$.

Mais exercícios

Exercício 4.F
O que significa "$T(n) = n^2 + \mathrm{O}(n)$"?
Mostre que se $T(n) = n^2 + \mathrm{O}(n)$ então $T(n) = \mathrm{O}(n^2)$.

Exercício 4.G
O que significa "$T(n) = n\mathrm{O}(\lg n)$"? Mostre que $T(n) = n\mathrm{O}(\lg n)$ se e só se $T(n) = \mathrm{O}(n \lg n)$.

Exercício 4.H
Interprete e prove a afirmação $7 \cdot \mathrm{O}(n) = \mathrm{O}(n)$.

Exercício 4.I
Interprete e prove a afirmação $\mathrm{O}(n) + \mathrm{O}(n) = \mathrm{O}(n)$.

Exercício 4.J
Prove que $\mathrm{O}(n) = \mathrm{O}(n^2)$. É verdade que $\mathrm{O}(n^2) = \mathrm{O}(n)$?

Exercício 4.K
Interprete e prove a afirmação $(n + 2) \cdot \mathrm{O}(1) = \mathrm{O}(n)$.

Mais exercícios ainda

Exercício 4.L

Interprete e prove a afirmação $\underbrace{\mathrm{O}(1) + \cdots + \mathrm{O}(1)}_{n+2} = \mathrm{O}(n)$.

Exercício 4.M

Prove que $\mathrm{O}(1) + \mathrm{O}(1) + \mathrm{O}(1) = \mathrm{O}(1)$.
É verdade que $\mathrm{O}(1) = \mathrm{O}(1) + \mathrm{O}(1) + \mathrm{O}(1)$?

Exercício 4.N

Interprete e prove a afirmação $\mathrm{O}(f) + \mathrm{O}(g) = \mathrm{O}(f + g)$.

Melhores momentos

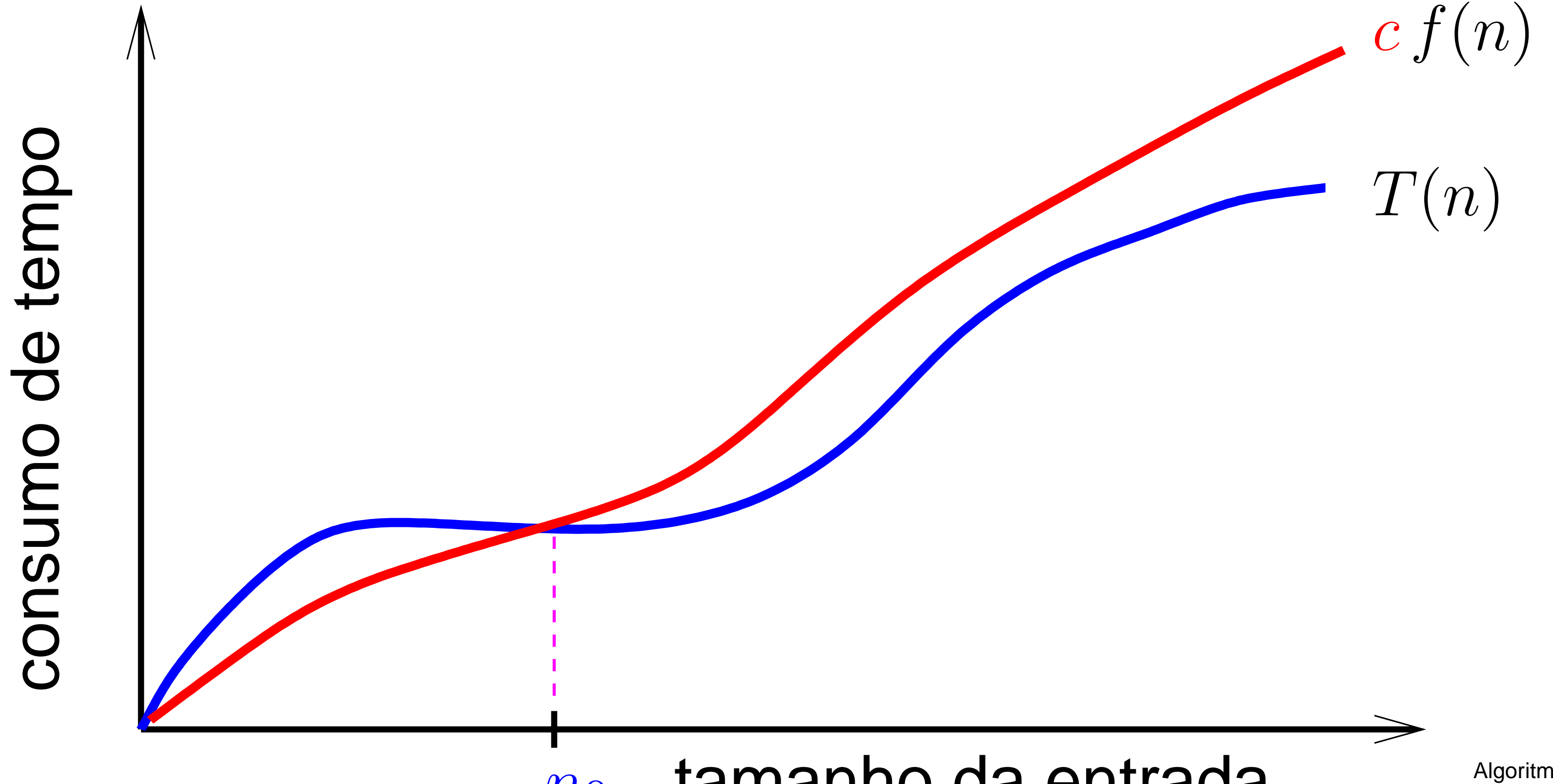
AULA 2

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais. Dizemos que $T(n)$ é $\mathrm{O}(f(n))$ se existem constantes positivas c e n_0 tais que

$$T(n) \leq c\,f(n)$$

para todo $n \geq n_0$.



Notação O

Intuitivamente...

$\mathrm{O}(f(n))$ $\approx$ funções q não crescem mais rápido que $f(n)$
$\approx$ funções menores ou iguais a um múltiplo de $f(n)$

n^2 $(3/2)n^2$ $9999n^2$ $n^2/1000$ etc.

crescem todas com a mesma velocidade, são todas $\mathrm{O}(n^2)$.

- $n^2 + 99n$ é $\mathrm{O}(n^2)$
- $33n^2$ é $\mathrm{O}(n^2)$
- $9n + 2$ é $\mathrm{O}(n^2)$
- $0{,}00001n^3 - 200n^2$ **não é** $\mathrm{O}(n^2)$

Uso da notação O

$\mathrm{O}(f(n)) = \{T(n)$: existem c e n_0 tq $T(n) \leq cf(n), n \geq n_0\}$

"$T(n)$ é $\mathrm{O}(f(n))$" deve ser entendido como "$T(n) \in \mathrm{O}(f(n))$".

"$T(n) = \mathrm{O}(f(n))$" deve ser entendido como "$T(n) \in \mathrm{O}(f(n))$".

"$T(n) \leq \mathrm{O}(f(n))$" é feio.

"$T(n) \geq \mathrm{O}(f(n))$" não faz sentido!

"$T(n)$ é $g(n) + \mathrm{O}(f(n))$" significa que existem constantes positivas c e n_0 tais que

$$T(n) \leq g(n) + c\, f(n)$$

para todo $n \geq n_0$.

Ordenação por inserção

Algoritmo rearranja $A[p \,..\, r]$ em ordem crescente

ORDENA-POR-INSERÇÃO (A, p, r)
1 **para** $j \leftarrow p + 1$ **até** r **faça**
2 $\quad chave \leftarrow A[j]$

3 $\quad i \leftarrow j - 1$
4 $\quad$ **enquanto** $i \geq p$ **e** $A[i] > chave$ **faça**
5 $\quad\quad A[i+1] \leftarrow A[i]$ $\quad \triangleright$ desloca
6 $\quad\quad i \leftarrow i - 1$

7 $\quad A[i+1] \leftarrow chave$ $\quad \triangleright$ insere

Quanto tempo o algoritmo consome?
"Tamanho" do problema: $n := r - p + 1$

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\mathrm{O}(n)$
2	$\mathrm{O}(n)$
3	$\mathrm{O}(n)$
4	$n\mathrm{O}(n) = \mathrm{O}(n^2)$
5	$n\mathrm{O}(n) = \mathrm{O}(n^2)$
6	$n\mathrm{O}(n) = \mathrm{O}(n^2)$
7	$\mathrm{O}(n)$
total	$\mathrm{O}(3n^2 + 4n) = \mathrm{O}(n^2)$

Justificativa

Bloco de linhas 4–6 é executado $\leq n$ vezes;
cada execução consome $\mathrm{O}(n)$;
todas juntas consomem $n\mathrm{O}(n)$.

Êpa!
Quem garante que $n\mathrm{O}(n) = \mathrm{O}(n^2)$?
Quem garante que $\mathrm{O}(n^2) + \mathrm{O}(n^2) + \mathrm{O}(n^2) = \mathrm{O}(3n^2)$?
Quem garante que $\mathrm{O}(3n^2 + 4n) = \mathrm{O}(n^2)$?
Veja exercícios de Mais notação O.

Conclusão:

O algoritmo consome $\mathrm{O}(n^2)$ unidades de tempo.

Notação O cai como uma luva!

$n\mathrm{O}(n) = \mathrm{O}(n^2)$

"$n\mathrm{O}(n) = \mathrm{O}(n^2)$" significa "$nO(n) \subset \mathrm{O}(n^2)$".
Ou seja, se $T(n)$ é $\mathrm{O}(n)$, então $nT(n)$ é $\mathrm{O}(n^2)$.

De fato, se $T(n)$ é $\mathrm{O}(n)$ então existem constantes, digamos 10 e 10^{100}, tais que

$$T(n) \ \leq \ 10\,n$$

para todo $n \geq 10^{100}$. Desta forma,

$$n\,T(n) \ \leq \ n\,10\,n \leq 10\,n^2$$

para todo $n \geq 10^{100}$. Logo $n\,T(n)$ é $\mathrm{O}(n^2)$.

$n\mathrm{O}(n) = \mathrm{O}(n^2)$

“$n\mathrm{O}(n) = \mathrm{O}(n^2)$” significa “$nO(n) \subset \mathrm{O}(n^2)$”.
Ou seja, se $T(n)$ é $\mathrm{O}(n)$, então $nT(n)$ é $\mathrm{O}(n^2)$.

De fato, se $T(n)$ é $\mathrm{O}(n)$ então existem constantes positivas c e n_0, tais que

$$T(n) \ \leq \ c\,n$$

para todo $n \geq n_0$. Desta forma,

$$n\,T(n) \ \leq \ n\,c\,n \leq c\,n^2$$

para todo $n \geq n_0$. Logo $n\,T(n)$ é $\mathrm{O}(n^2)$.

AULA 3

Mais análise com notação O

CLRS 2.1–2.2

AU 3.3, 3.6 (muito bom)

Análise da intercalação

Problema: Dados $A[p\,..\,q]$ e $A[q+1\,..\,r]$ crescentes, rearranjar $A[p\,..\,r]$ de modo que ele fique em ordem crescente.

Para que valores de q o problema faz sentido?

Entra:

	p				q				r
A	22	33	55	77	99	11	44	66	88

Análise da intercalação

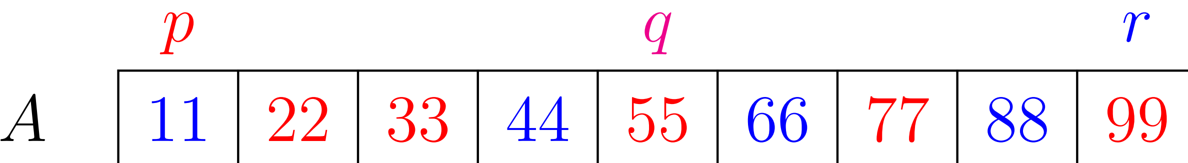
Problema: Dados $A[p\,..\,q]$ e $A[q+1\,..\,r]$ crescentes, rearranjar $A[p\,..\,r]$ de modo que ele fique em ordem crescente.

Para que valores de q o problema faz sentido?

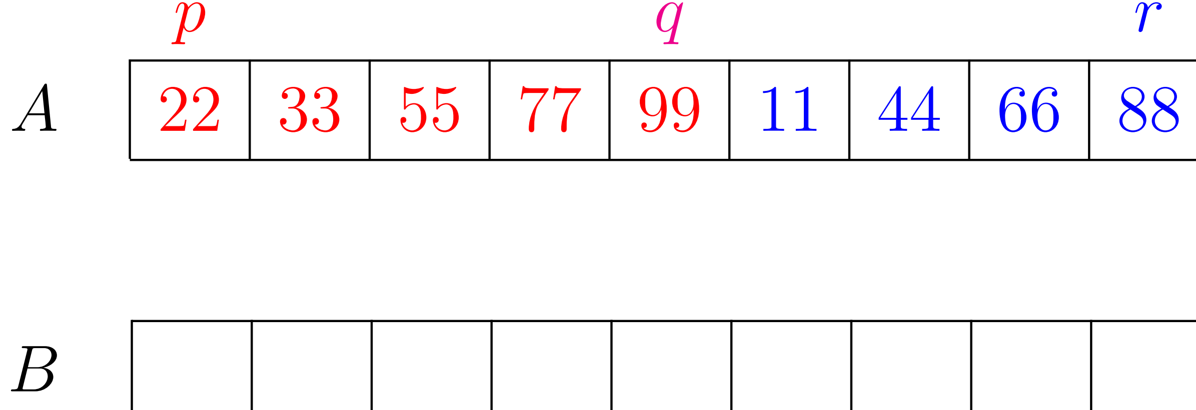
Entra:

	p				q				r
A	22	33	55	77	99	11	44	66	88

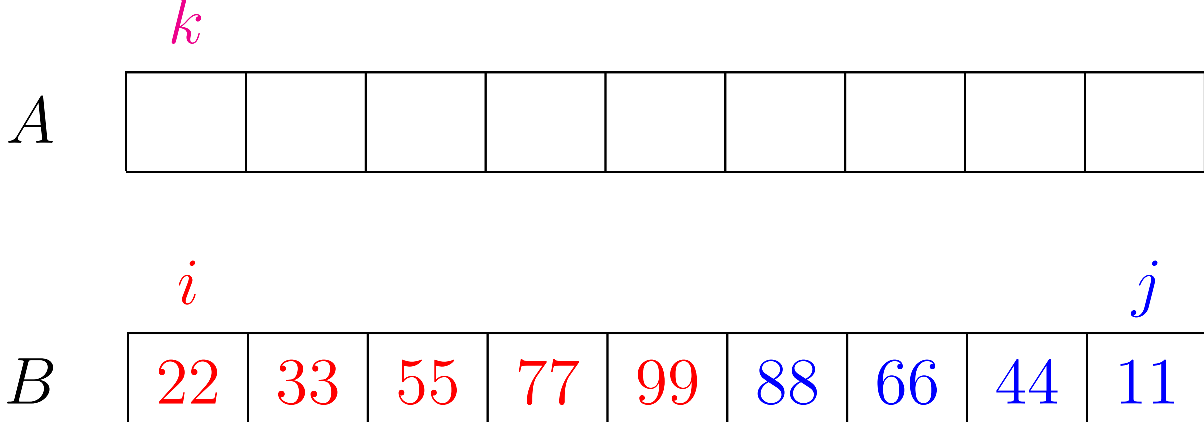
Sai:



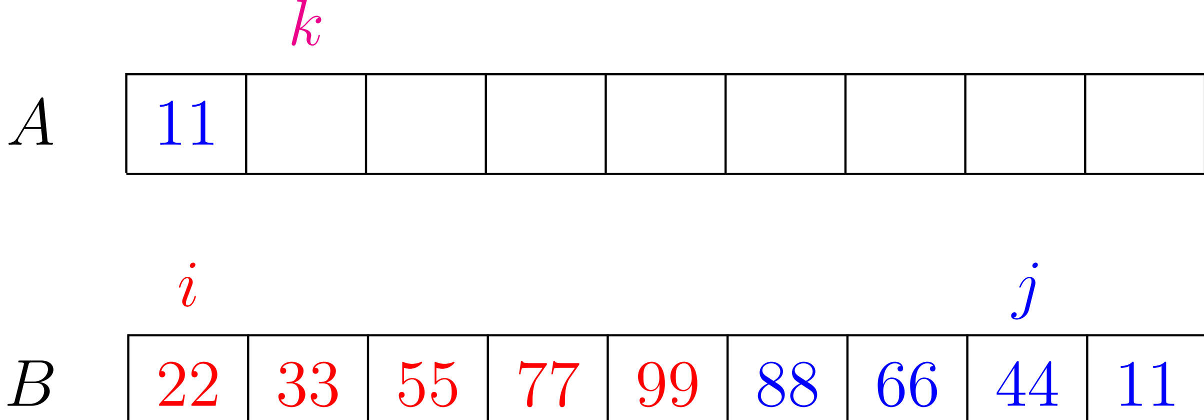
Intercalação



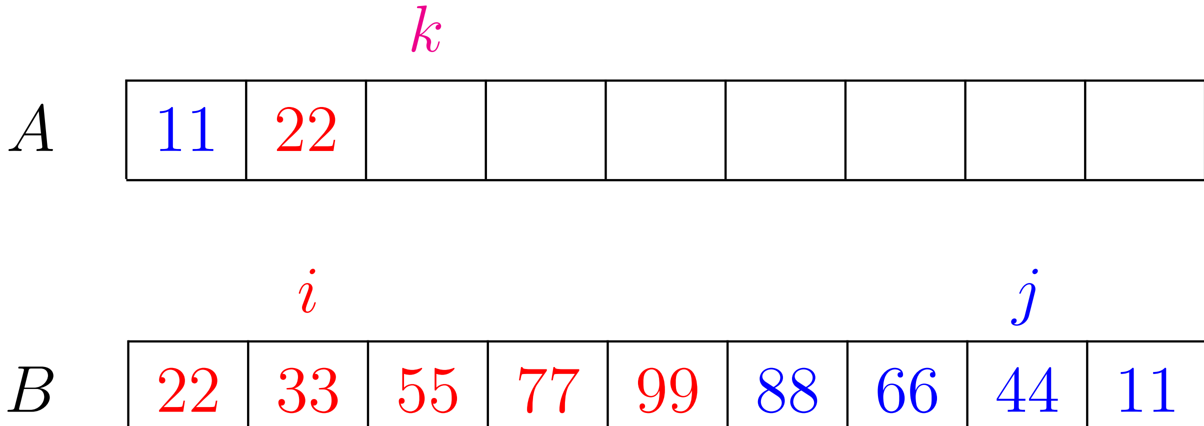
Intercalação



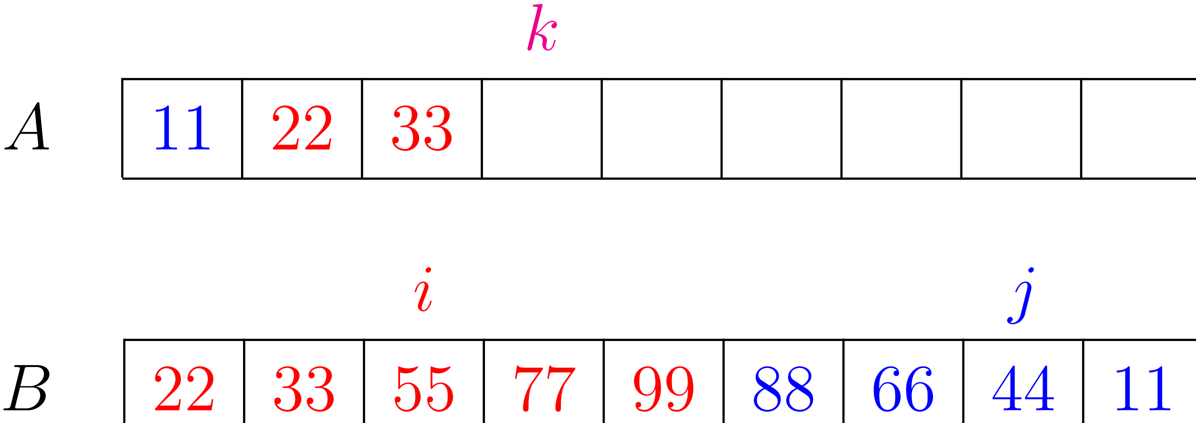
Intercalação



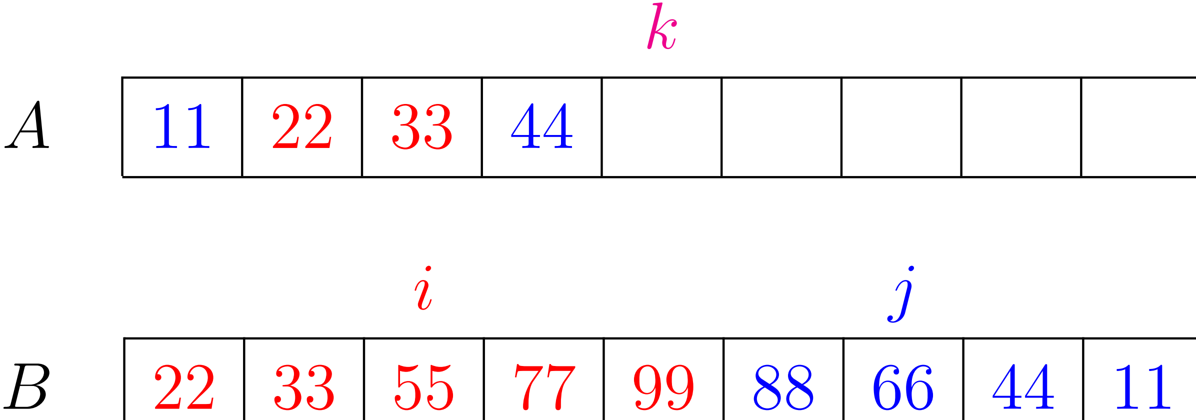
Intercalação



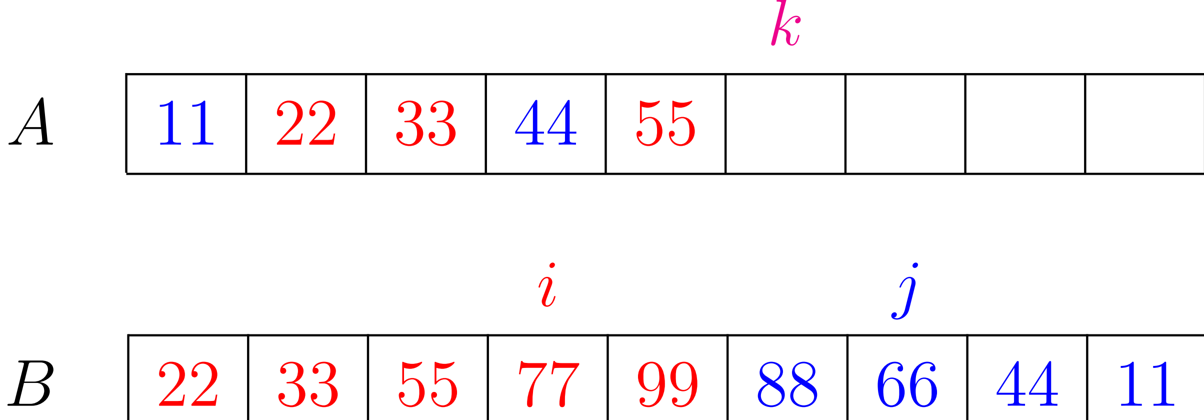
Intercalação



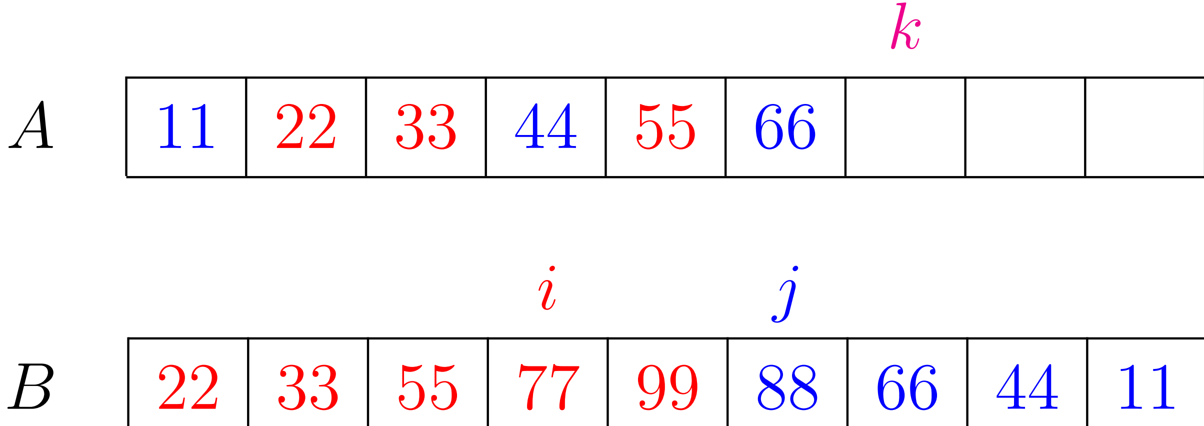
Intercalação



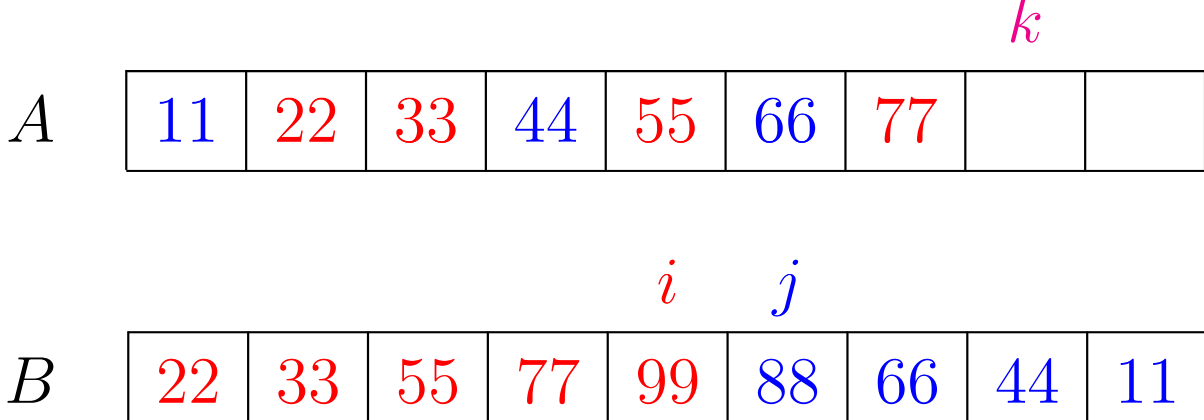
Intercalação



Intercalação



Intercalação



Intercalação

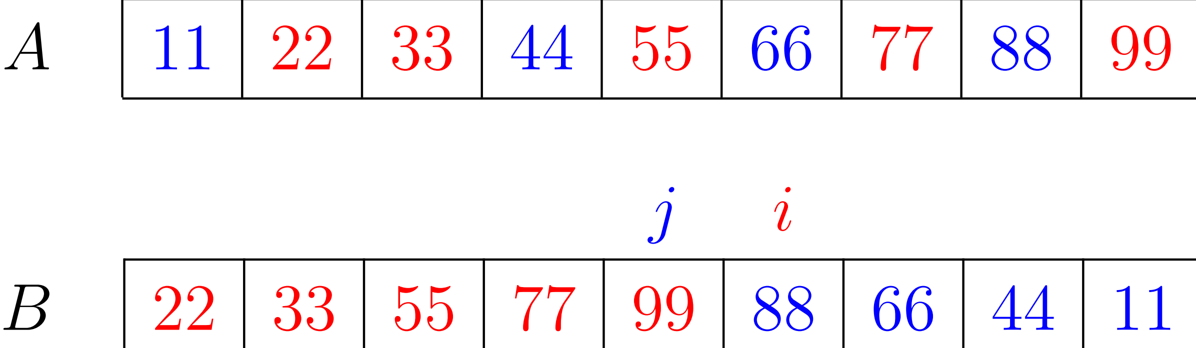
k

A	11	22	33	44	55	66	77	88	

$i = j$

B	22	33	55	77	99	88	66	44	11

Intercalação



Intercalação

INTERCALA (A, p, q, r)

```
0    ▷ B[p . . r] é um vetor auxiliar
1    para i ← p até q faça
2        B[i] ← A[i]
3    para j ← q + 1 até r faça
4        B[r + q + 1 − j] ← A[j]
5    i ← p
6    j ← r
7    para k ← p até r faça
8        se B[i] ≤ B[j]
9            então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha	todas as execuções da linha
1	?
2	?
3	?
4	?
5	?
6	?
7	?
8	?
9–12	?
total	?

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é ($n := r - p + 1$):

linha	todas as execuções da linha
1	$= \quad q - p + 2 \;=\; n - r + q + 1$
2	$= \quad q - p + 1 \;=\; n - r + q$
3	$= \quad r - (q + 1) + 2 \;=\; n - q + p$
4	$= \quad r - (q + 1) + 1 \;=\; n - q + p - 1$
5	$= \quad 1$
6	$= \quad 1$
7	$= \quad r - p + 2 \;=\; n + 1$
8	$= \quad r - p + 1 \;=\; n$
9–12	$= \quad 2\,(r - p + 1) \;=\; 2n$
total	$= \quad 8n - 2(r - p + 1) + 5 = \; 6n + 5$

Conclusão

O algoritmo INTERCALA consome $6n + 5$ unidades de tempo.

Consumo de tempo em O

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha
1–4	?
5–6	?
7	?
8	?
9–12	?
total	?

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha
1–4	$\mathrm{O}(n)$
5–6	$\mathrm{O}(1)$
7	$n\mathrm{O}(1) = \mathrm{O}(n)$
8	$n\mathrm{O}(1) = \mathrm{O}(n)$
9–12	$n\mathrm{O}(1) = \mathrm{O}(n)$
total	$\mathrm{O}(4n + 1)$ = $\mathrm{O}(n)$

Conclusão

O algoritmo INTERCALA consome $\mathrm{O}(n)$ unidades de tempo.

Também escreve-se

O algoritmo INTERCALA consome tempo $\mathrm{O}(n)$.

Exercício

Exercício 5.A

Analise a correção e o consumo de tempo do seguinte algoritmo:

EXERC (A, p, r)

1 $q \leftarrow \lfloor (p + r)/2 \rfloor$

2 ORDENA-POR-INSERÇÃO (A, p, q)

3 ORDENA-POR-INSERÇÃO $(A, q + 1, r)$

4 INTERCALA (A, p, q, r)

Irmãos de O

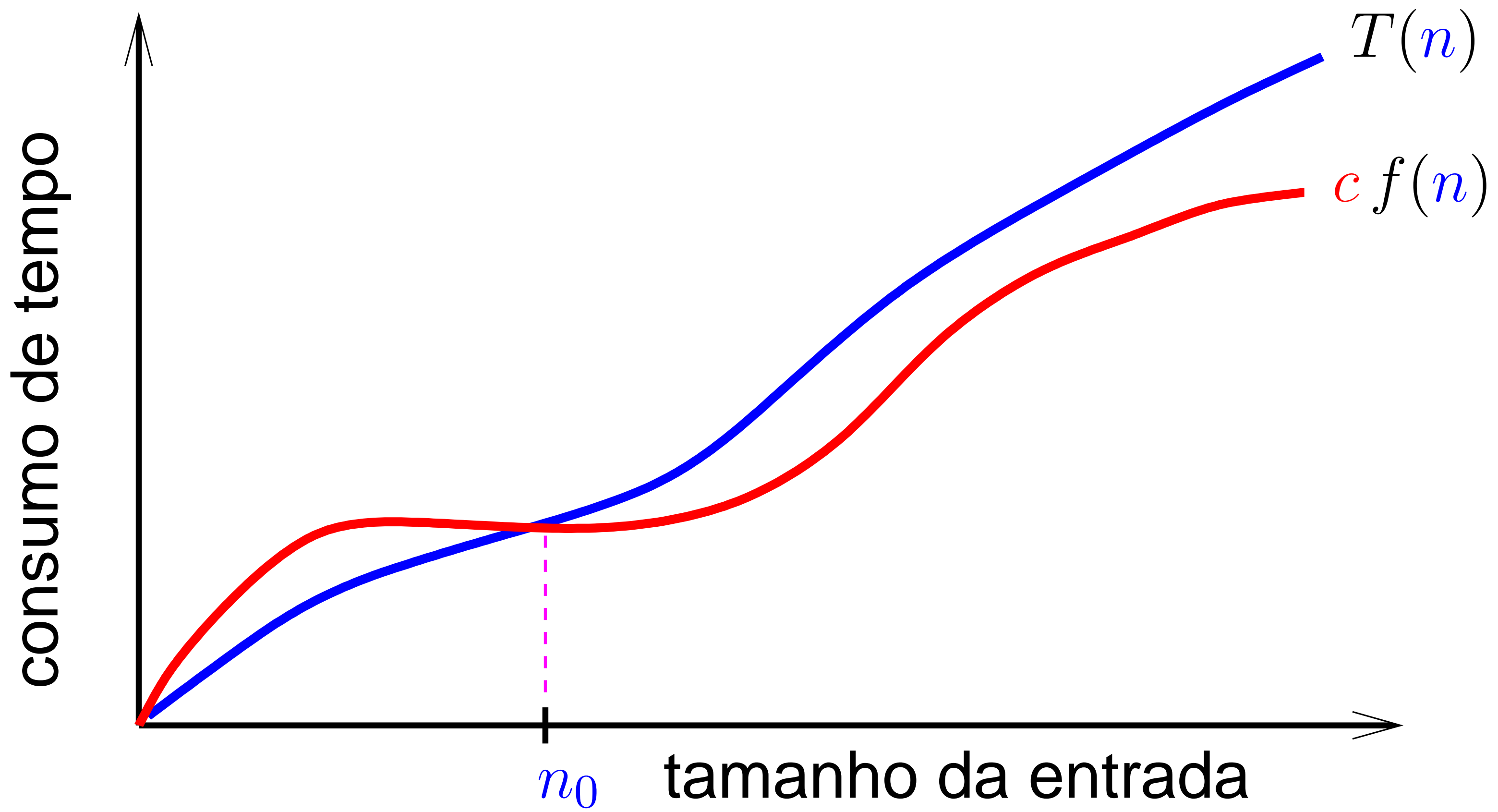
CLRS 3.1

Definição

Dizemos que $T(n)$ é $\Omega(f(n))$ se existem constantes positivas c e n_0 tais que

$$c\, f(n) \;\leq\; T(n)$$

para todo $n \geq n_0$.

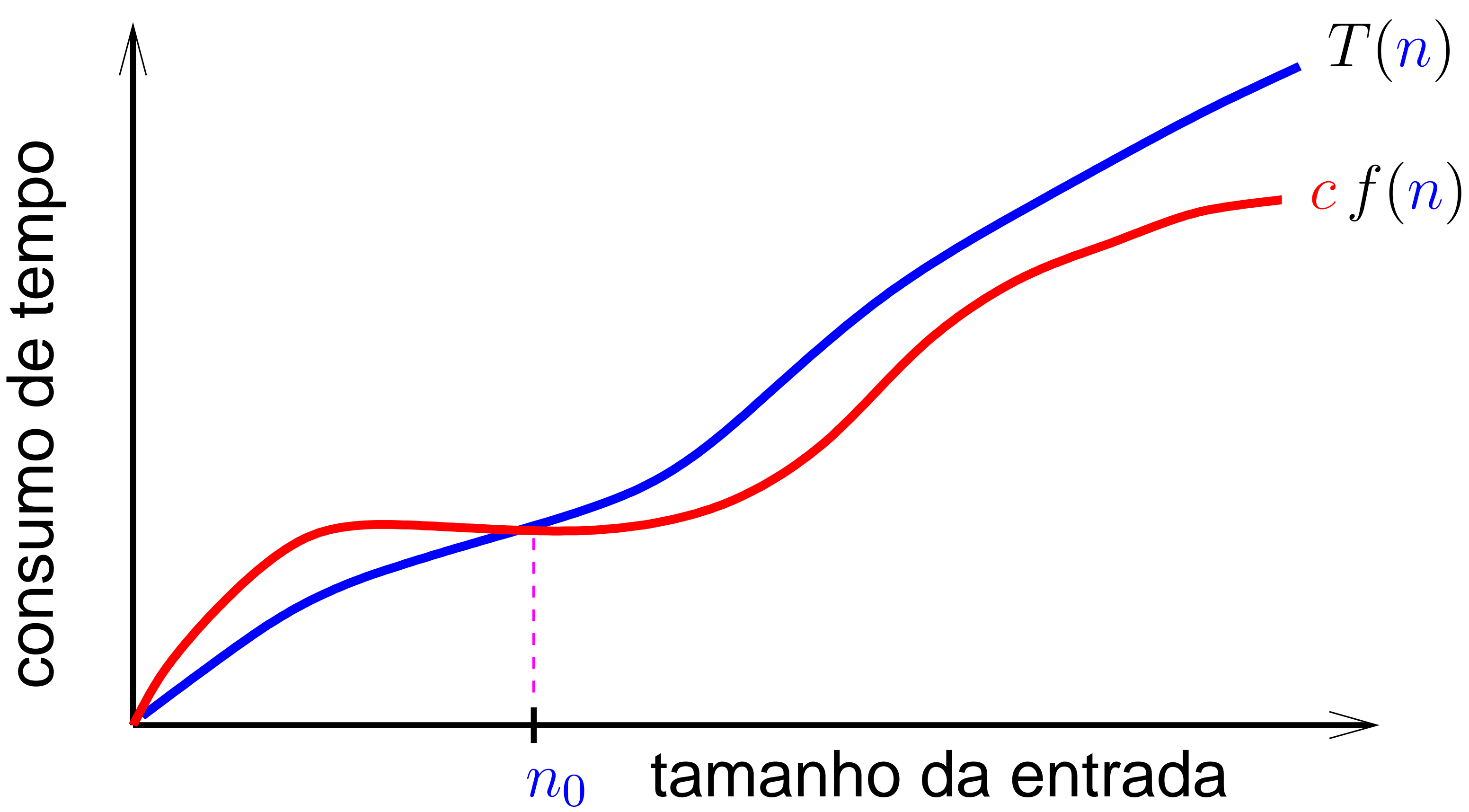


Mais informal

$T(n) = \Omega(f(n))$ se existe $c > 0$ tal que

$$c\, f(n) \leq T(n)$$

para todo n suficientemente GRANDE.



Exemplos

Exemplo 1

Se $T(n) \geq 0.001n^2$ para todo $n \geq 8$, então $T(n)$ é $\Omega(n^2)$.

Exemplos

Exemplo 1
Se $T(n) \geq 0.001n^2$ para todo $n \geq 8$, então $T(n)$ é $\Omega(n^2)$.

Prova: Aplique a definição com $c = 0.001$ e $n_0 = 8$.

Exemplo 2

O consumo de tempo do INTERCALA é $\mathrm{O}(n)$ e também $\Omega(n)$.

Exemplo 2

O consumo de tempo do INTERCALA é $\mathrm{O}(n)$ e também $\Omega(n)$.

linha	todas as execuções da linha
1	$= \quad q - p + 2 = n - r + q + 1$
2	$= \quad q - p + 1 = n - r + q$
3	$= \quad r - (q + 1) + 2 = n - q + p$
4	$= \quad r - (q + 1) + 1 = n - q + p - 1$
5	$= \quad 1$
6	$= \quad 1$
7	$= \quad r - p + 2 = n + 1$
8	$= \quad r - p + 1 = n$
9–12	$= \quad 2\,(r - p + 1) = 2n$
total	$= \quad 8n - 2(r - p + 1) + 5 = 6n + 5$

Exemplo 3

Se $T(n)$ é $\Omega(f(n))$, então $f(n)$ é $\mathrm{O}(T(n))$.

Exemplo 3

Se $T(n)$ é $\Omega(f(n))$, então $f(n)$ é $\mathrm{O}(T(n))$.

Prova: Se $T(n)$ é $\Omega(f(n))$, então existem constantes positivas c e n_0 tais que

$$c\,f(n) \;\leq\; T(n)$$

para todo $n \geq n_0$. Logo,

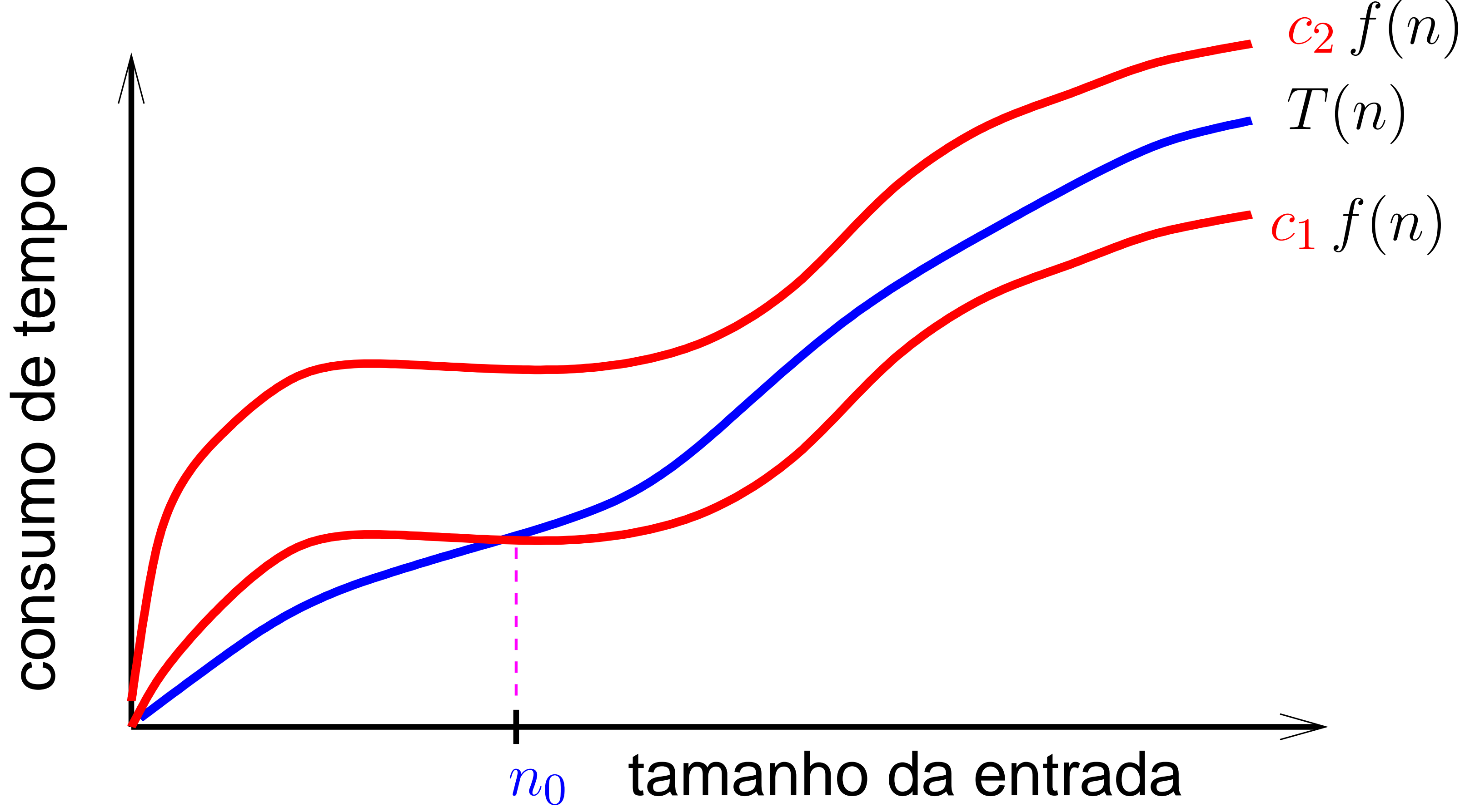
$$f(n) \;\leq\; 1/c\,T(n)$$

para todo $n \geq n_0$. Portanto, $f(n)$ é $\mathrm{O}(T(n))$.

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais.
Dizemos que $T(n)$ é $\Theta(f(n))$ se

$$T(n) \text{ é } \mathrm{O}(f(n)) \text{ e } T(n) \text{ é } \Omega(f(n)).$$

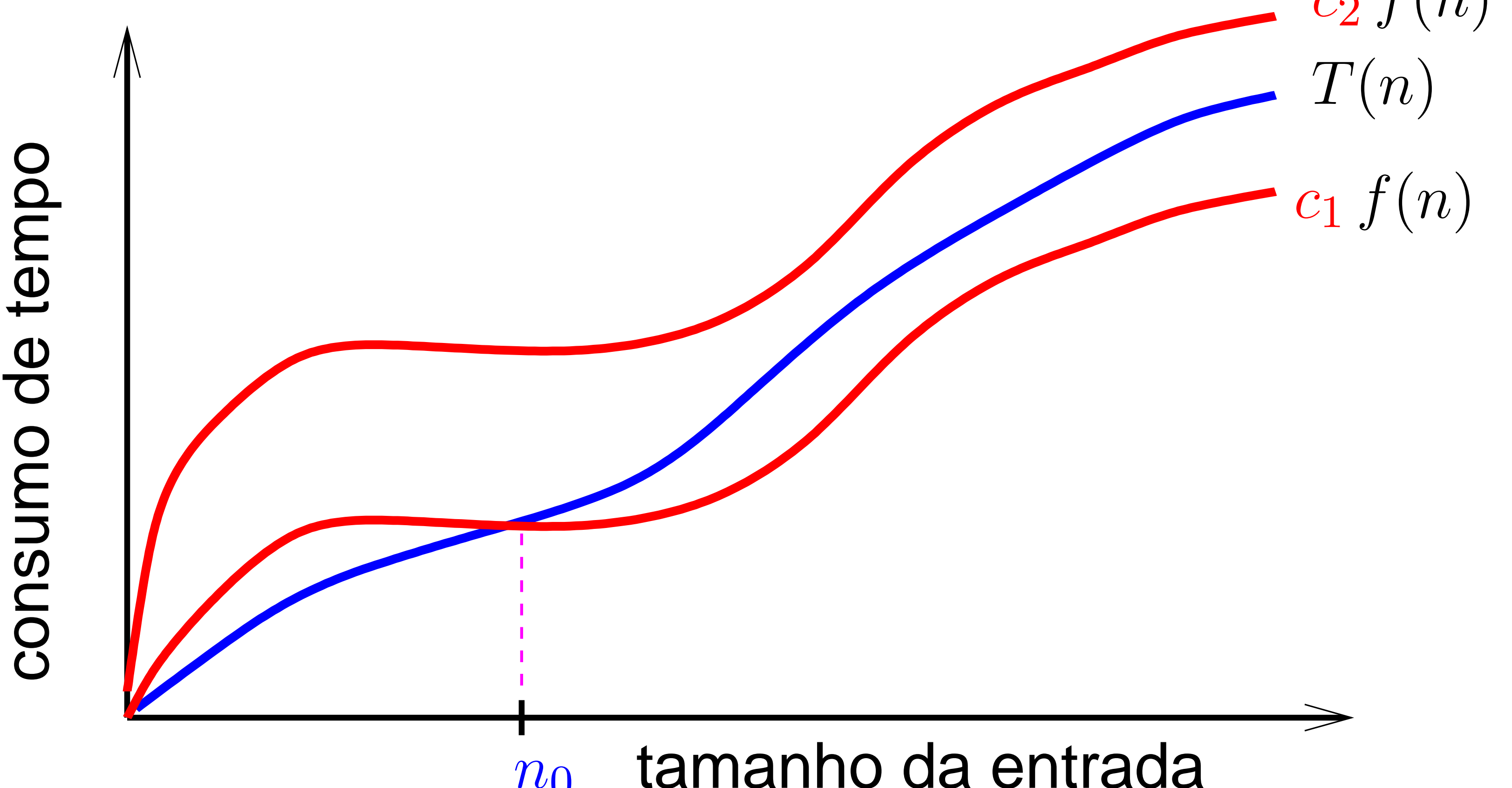


Definição

Dizemos que $T(n)$ é $\Theta(f(n))$ se se existem constantes positivas c_1, c_2 e n_0 tais que

$$c_1\, f(n) \;\leq\; T(n) \;\leq\; c_2\, f(n)$$

para todo $n \geq n_0$.



Intuitivamente

Comparação assintótica, ou seja, para n ENORME.

comparação	comparação assintótica
$T(n) \leq f(n)$	$T(n)$ é $\mathrm{O}(f(n))$
$T(n) \geq f(n)$	$T(n)$ é $\Omega(f(n))$
$T(n) = f(n)$	$T(n)$ é $\Theta(f(n))$

Tamanho máximo de problemas

Suponha que cada operação consome 1 microsegundo ($1\mu s$).

consumo de tempo(μs)	Tamanho máximo de problemas (n)		
	1 segundo	1 minuto	1 hora
$400n$	2500	150000	9000000
$20\,n\,\lceil \lg n \rceil$	4096	166666	7826087
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

Michael T. Goodrich e Roberto Tamassia, *Projeto de Algoritmos*, Bookman.

Crescimento de algumas funções

n	$\lg n$	$\sqrt{n}$	$n \lg n$	n^2	n^3	2^n
2	1	1,4	2	4	8	4
4	2	2	8	16	64	16
8	3	2,8	24	64	512	256
16	4	4	64	256	4096	65536
32	5	5,7	160	1024	32768	4294967296
64	6	8	384	4096	262144	1,8 10^{19}
128	7	11	896	16384	2097152	3,4 10^{38}
256	8	16	1048	65536	16777216	1,1 10^{77}
512	9	23	4608	262144	134217728	1,3 10^{154}
1024	10	32	10240	1048576	1,1 10^{9}	1,7 10^{308}

Nomes de classes Θ

classe	nome
$\Theta(1)$	constante
$\Theta(\log n)$	logarítmica
$\Theta(n)$	linear
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	quadrática
$\Theta(n^3)$	cúbica
$\Theta(n^k)$ com $k \geq 1$	polinomial
$\Theta(2^n)$	exponencial
$\Theta(a^n)$ com $a > 1$	exponencial

Palavras de Cautela

Suponha que $\mathcal{A}$ e $\mathcal{B}$ são algoritmos para um mesmo problema. Suponha que o consumo de tempo de $\mathcal{A}$ é "essencialmente" $100\,n$ e que o consumo de tempo de $\mathcal{B}$ é "essencialmente" $n \log_{10} n$.

Palavras de Cautela

Suponha que $\mathcal{A}$ e $\mathcal{B}$ são algoritmos para um mesmo problema. Suponha que o consumo de tempo de $\mathcal{A}$ é "essencialmente" $100\,n$ e que o consumo de tempo de $\mathcal{B}$ é "essencialmente" $n \log_{10} n$.

$100\,n$ é $\Theta(n)$ e $n \log_{10} n$ é $\Theta(n \lg n)$.
Logo, $\mathcal{A}$ é assintoticamente mais eficiente que $\mathcal{B}$.

Palavras de Cautela

Suponha que $\mathcal{A}$ e $\mathcal{B}$ são algoritmos para um mesmo problema. Suponha que o consumo de tempo de $\mathcal{A}$ é “essencialmente” $100\,n$ e que o consumo de tempo de $\mathcal{B}$ é “essencialmente” $n \log_{10} n$.

$100\,n$ é $\Theta(n)$ e $n \log_{10} n$ é $\Theta(n \lg n)$.
Logo, $\mathcal{A}$ é assintoticamente mais eficiente que $\mathcal{B}$.

$\mathcal{A}$ é mais eficiente que $\mathcal{B}$ para $n \geq 10^{100}$.

10^{100} = um *googol*
$\approx$ número de átomos no universo observável
= número ENORME

Palavras de Cautela

Conclusão:

Lembre das constantes e termos de baixa ordem que estão "escondidos" na notação assintótica.

Em geral um algoritmo que consome tempo $\Theta(n \lg n)$, e com fatores constantes razoáveis, é bem eficiente.

Um algoritmo que consome tempo $\Theta(n^2)$ pode, algumas vezes ser satisfatório.

Um algoritmo que consome tempo $\Theta(2^n)$ é dificilmente aceitável.

Do ponto de vista de AA, eficiente $=$ polinomial.

Exercícios

Exercício 6.A

Já sabemos que ORDENA-POR-INSERÇÃO é $\mathrm{O}(n^2)$.
Mostre que o algoritmo é $\Omega(n)$.

Exercício 6.B

Mostre que ORDENA-POR-INSERÇÃO é $\Omega(n^2)$
no pior caso.

Exercício 6.C

Mostre que ORDENA-POR-INSERÇÃO é $\mathrm{O}(n)$
no melhor caso.

Mais exercícios

Exercício 6.D

Prove que $n^2 + 10n + 20 = \Omega(n^2)$. Prove que $n^2 - 10n - 20 = \Theta(n^2)$.

Exercício 6.E

Prove que $n = \Omega(\lg n)$.

Exercício 6.F

Prove que $\lg n = \Theta(\log_{10} n)$.

Exercício 6.G

É verdade que $2^n = \Omega(3^n)$?

Exercício 6.H

É verdade que $2n^3 + 5\sqrt{n} = \Theta(n^3)$?

Mais exercícios ainda

Exercício 6.I

Suponha que os algoritmos $\mathcal{A}$ e $\mathcal{B}$ só dependem de um parâmetro n. Suponha ainda que $\mathcal{A}$ consome $S(n)$ unidades de tempo enquanto $\mathcal{B}$ consome $T(n)$ unidades de tempo. Quero provar que algoritmo $\mathcal{A}$ é pelo menos tão eficiente quanto o algoritmo $\mathcal{B}$ (no sentido assintótico). Devo mostrar que existe $f(n)$ tal que

$S(n) = \mathrm{O}(f(n))$ e $T(n) = \mathrm{O}(f(n))$?

$S(n) = \mathrm{O}(f(n))$ e $T(n) = \Omega(f(n))$?

$S(n) = \Omega(f(n))$ e $T(n) = \mathrm{O}(f(n))$?

$S(n) = \Omega(f(n))$ e $T(n)\Omega(f(n))$?

Que devo fazer para mostrar que $\mathcal{A}$ é mais eficiente que $\mathcal{B}$?

Exercício 6.J

Mostre que o consumo de tempo do algoritmo INTERCALA é $\Theta(n)$, sendo n o número de elementos do vetor que o algoritmo recebe.

Recursão

CLRS 2.3
AU 2.6, 2.7, 2.9

"To understand recursion, we must first understand recursion.”

Recursão

Recursão: resolve um problema a partir das soluções de seus subproblemas

Recursão

Recursão: resolve um problema a partir das soluções de seus subproblemas

Problema: Rearranjar $A[p \mathinner{.\,.} r]$ de modo que ele fique em ordem crescente.

Entra:

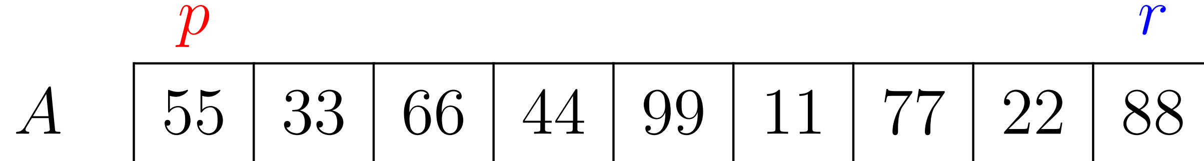
	p								r
A	55	33	66	44	99	11	77	22	88

Recursão

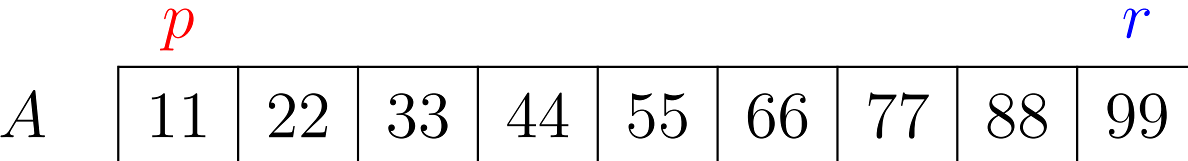
Recursão: resolve um problema a partir das soluções de seus subproblemas

Problema: Rearranjar $A[p\,..\,r]$ de modo que ele fique em ordem crescente.

Entra:



Sai:



Divisão-e-conquista

Algoritmos por divisão-e-conquista têm três passos em cada nível da recursão:

Dividir: o problema é dividido em subproblemas de tamanho menor;

Conquistar: os subproblemas são resolvidos recursivamente e subproblemas “pequenos” são resolvidos diretamente;

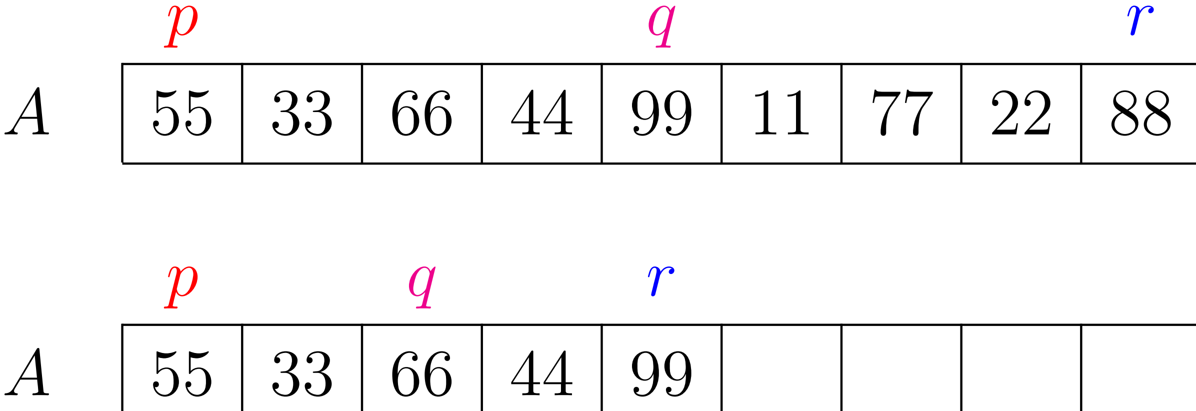
Combinar: as solução dos subproblemas são combinadas para obter uma solução do problema original.

Exemplo: ordenação por intercalação (Merge-sort).

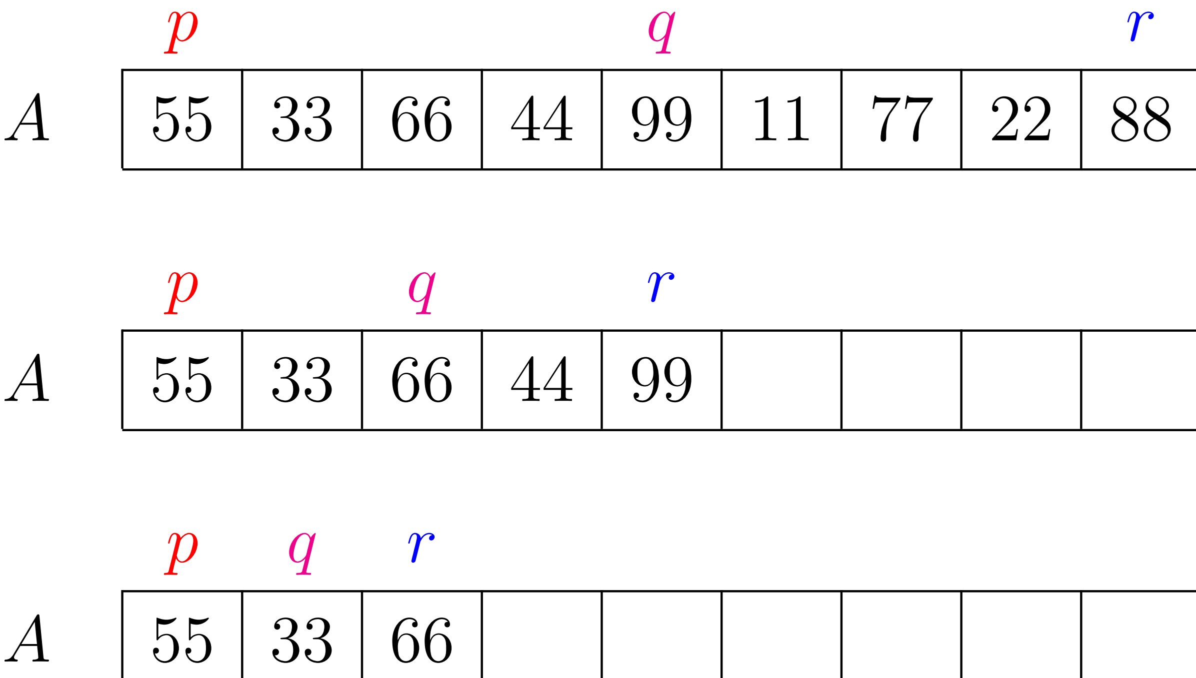
Merge-Sort

	p				q				r
A	55	33	66	44	99	11	77	22	88

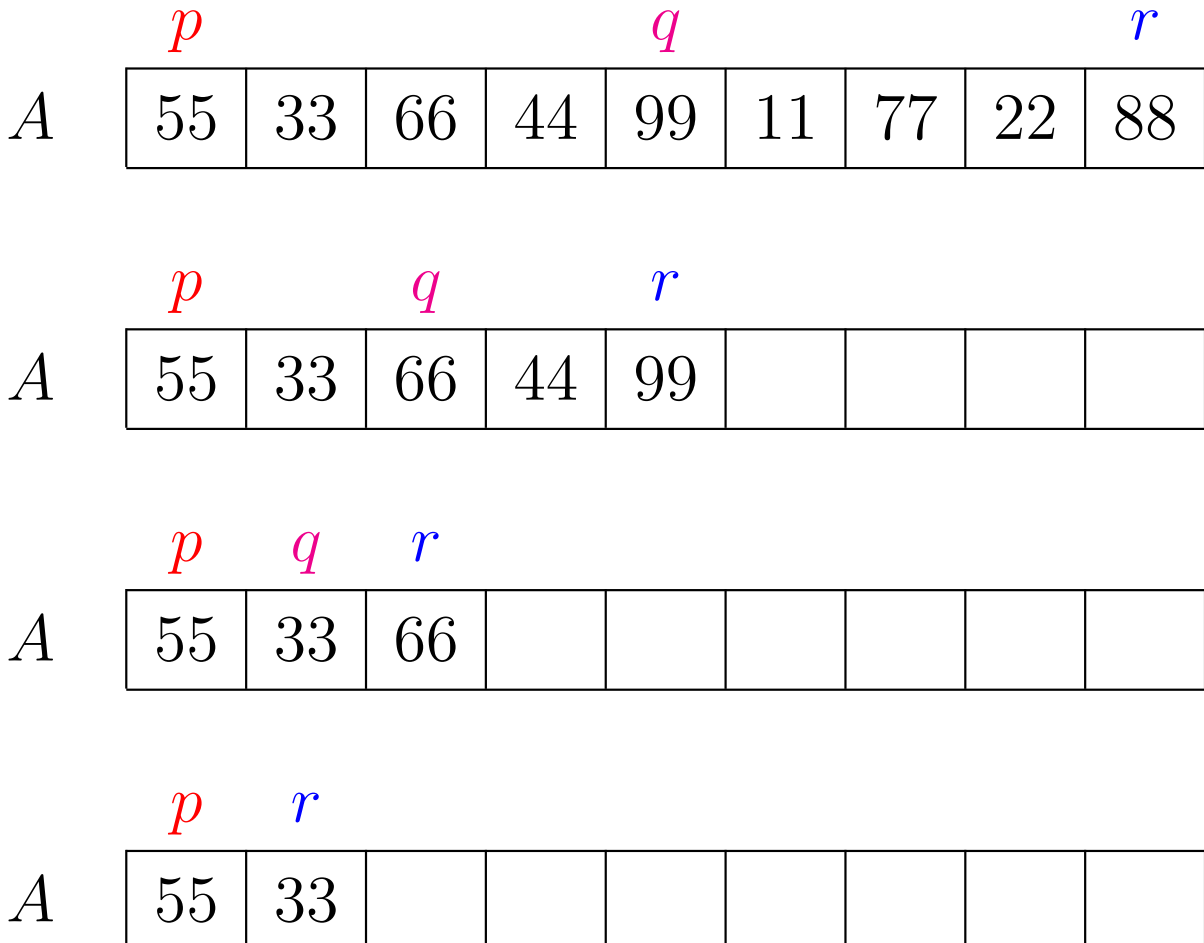
Merge-Sort



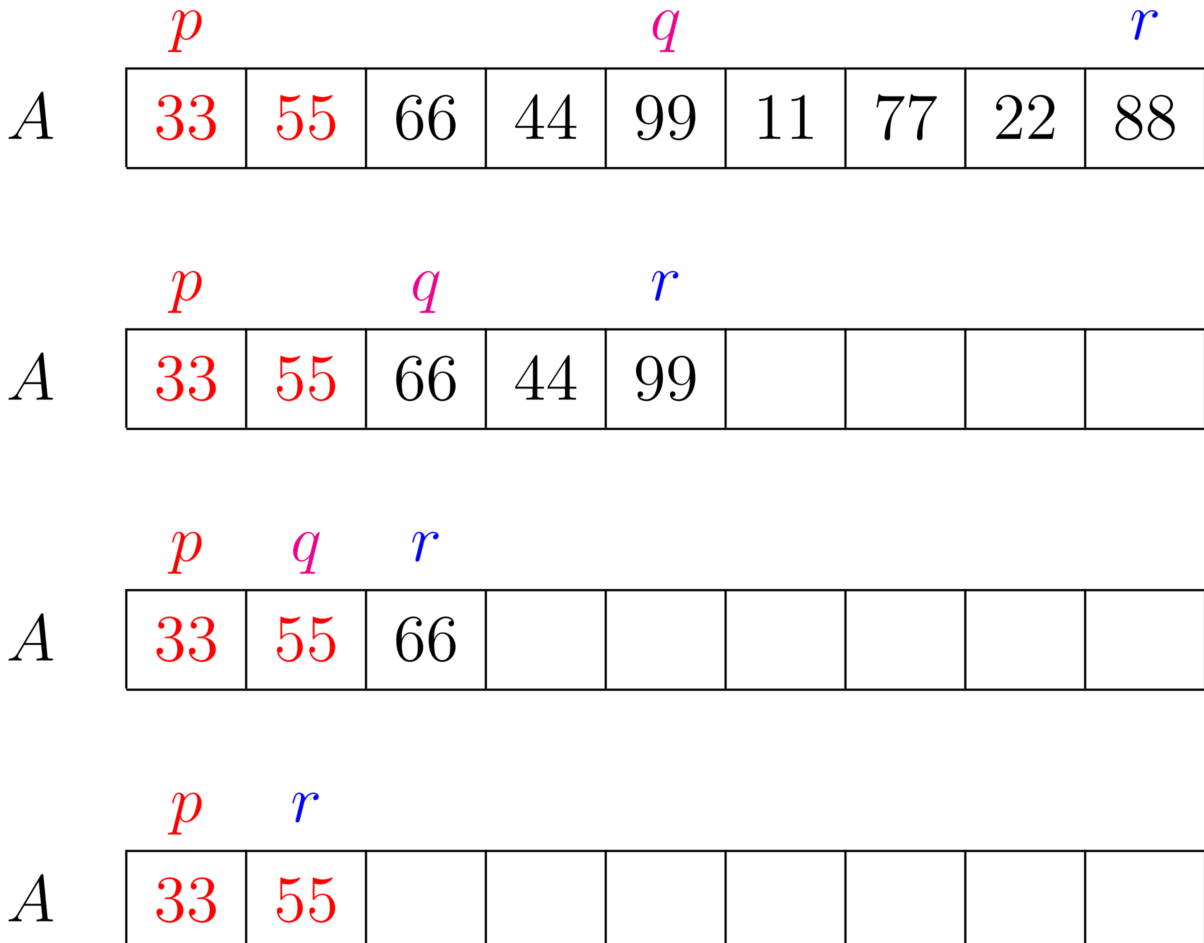
Merge-Sort



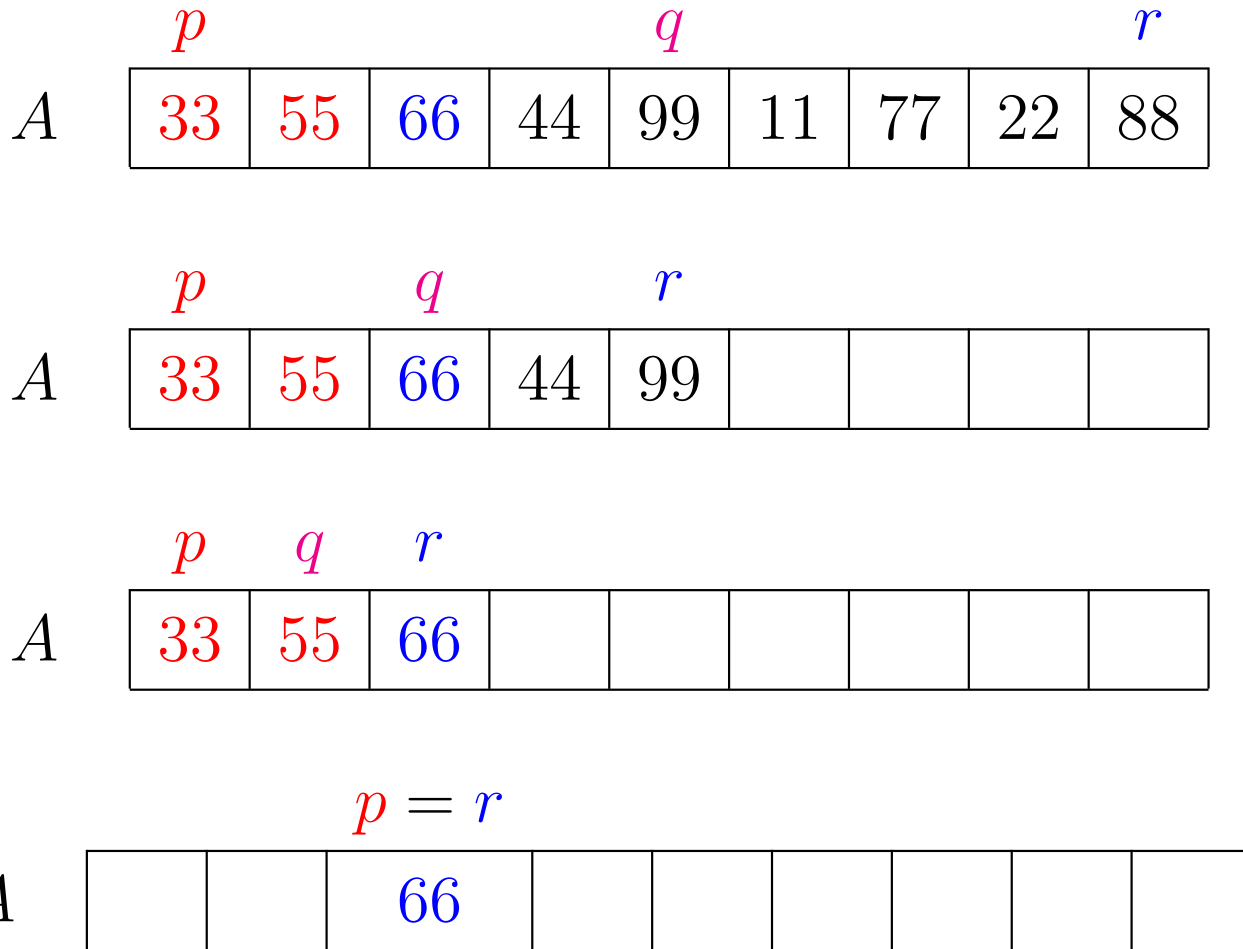
Merge-Sort



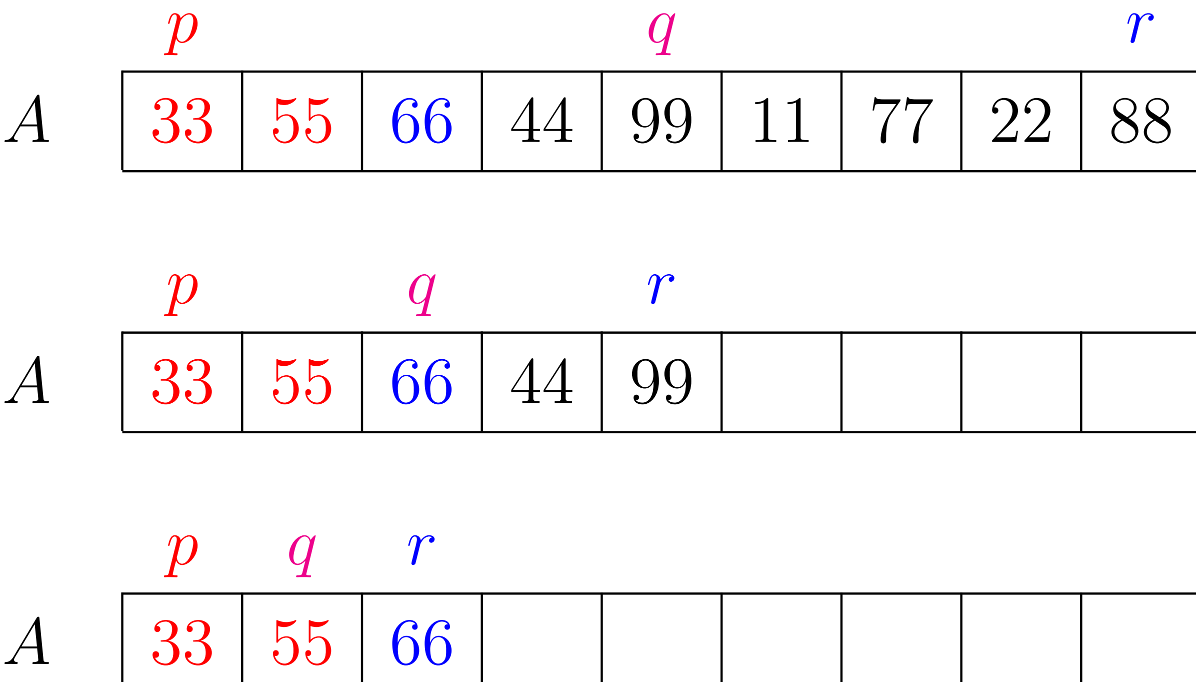
Merge-Sort



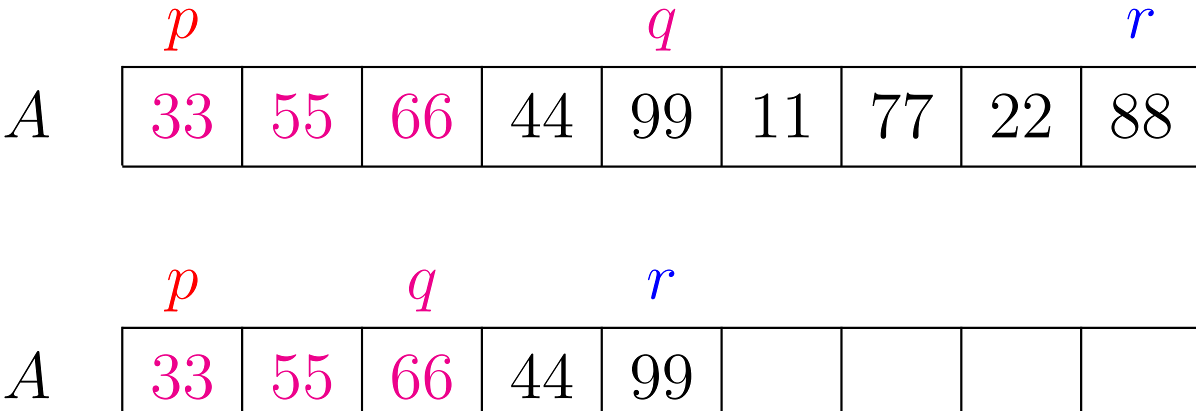
Merge-Sort



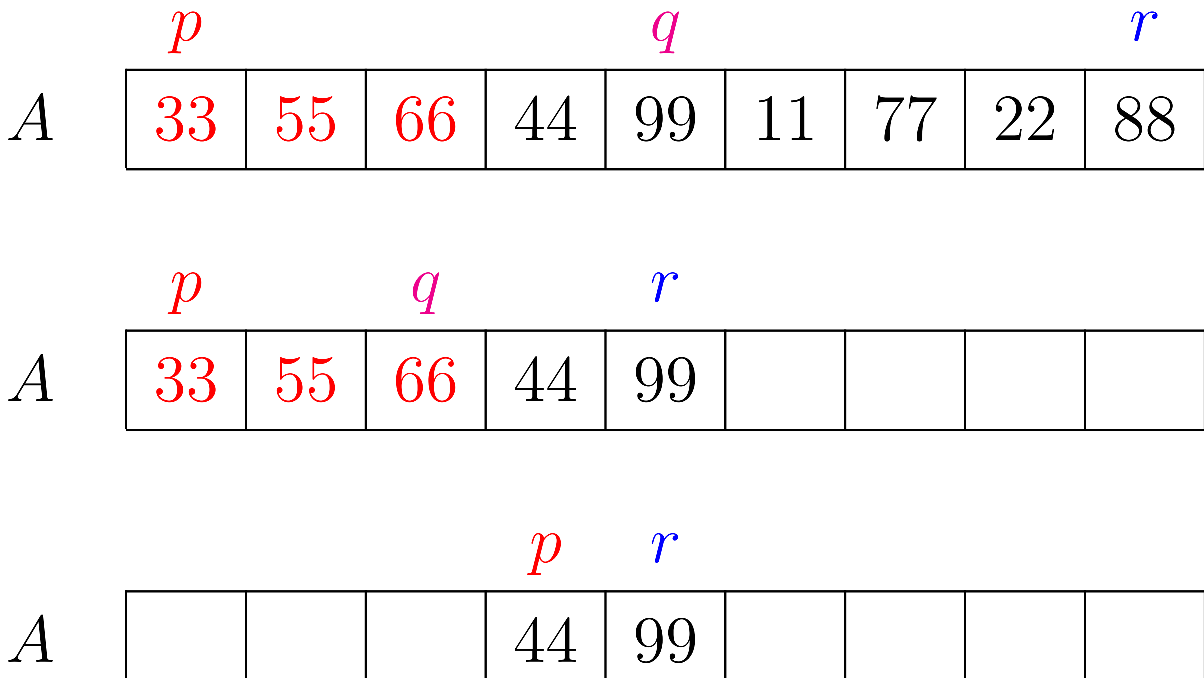
Merge-Sort



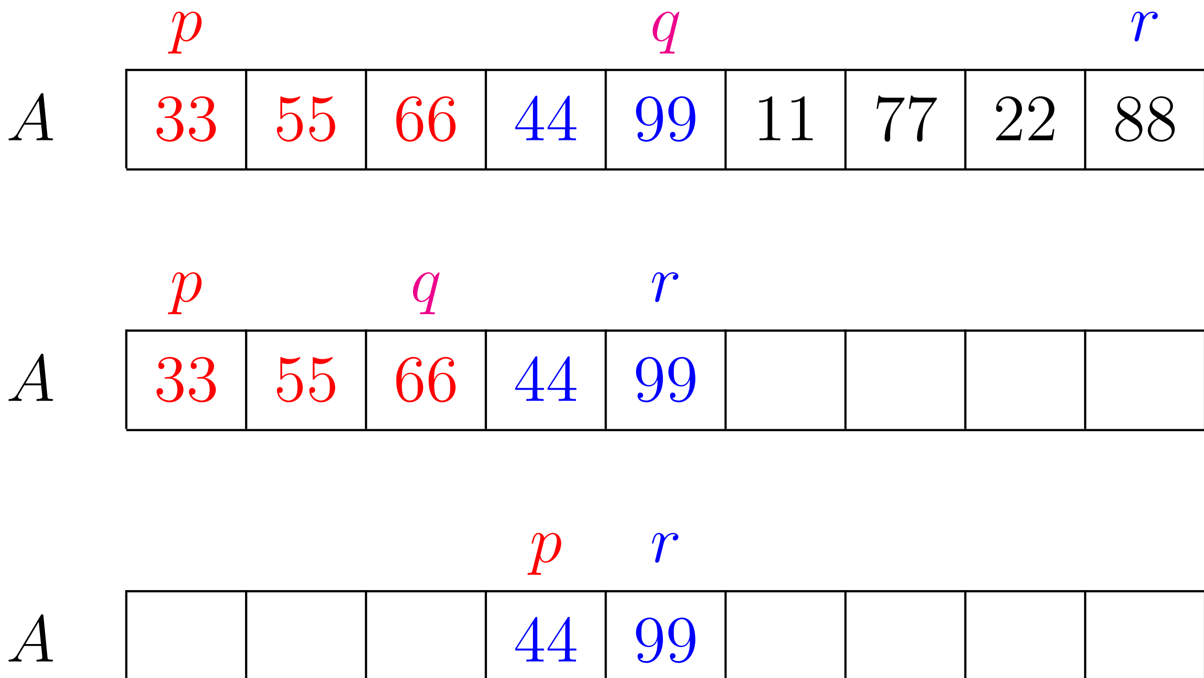
Merge-Sort



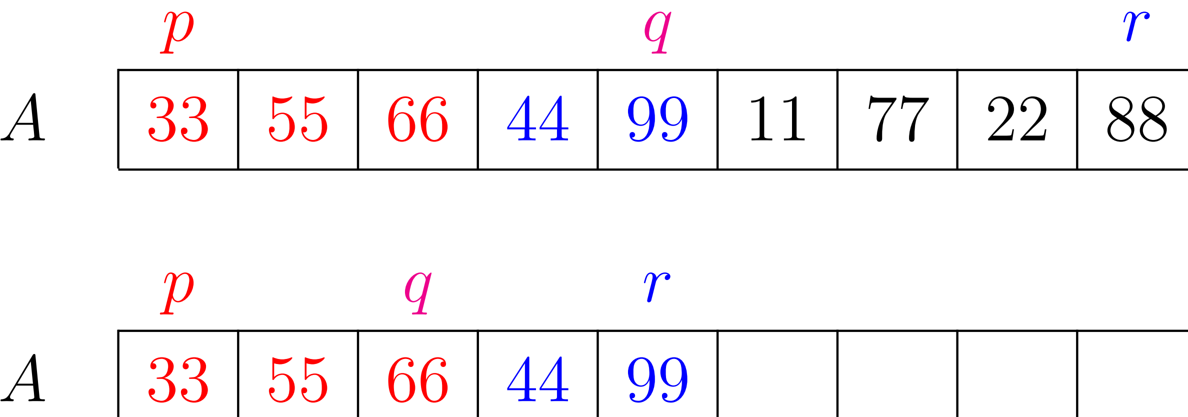
Merge-Sort



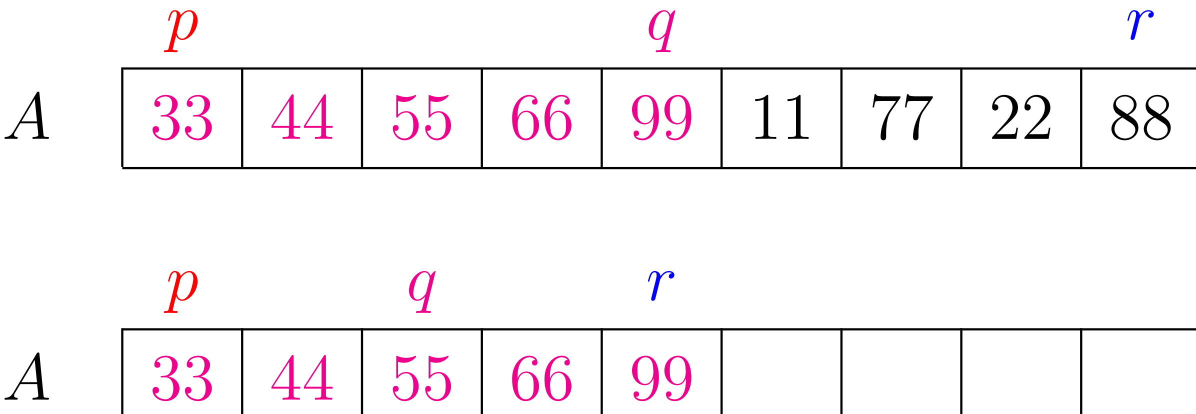
Merge-Sort



Merge-Sort



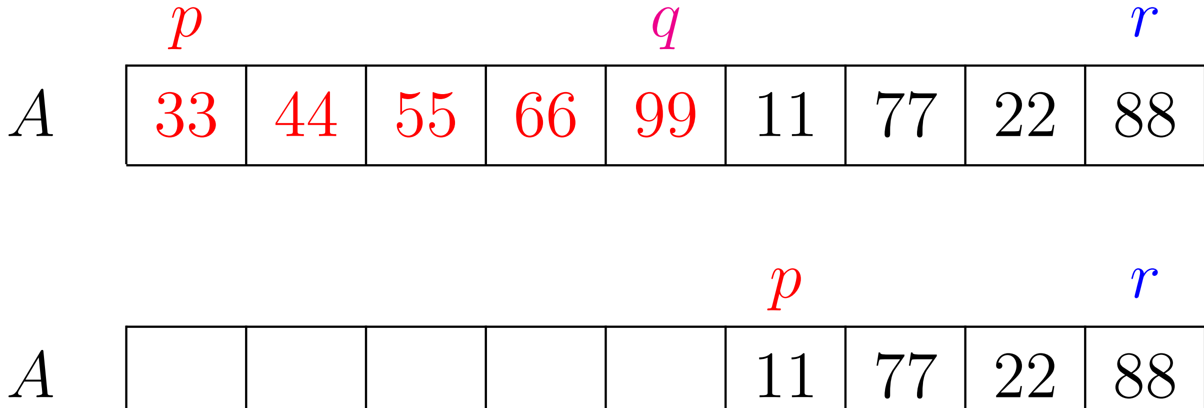
Merge-Sort



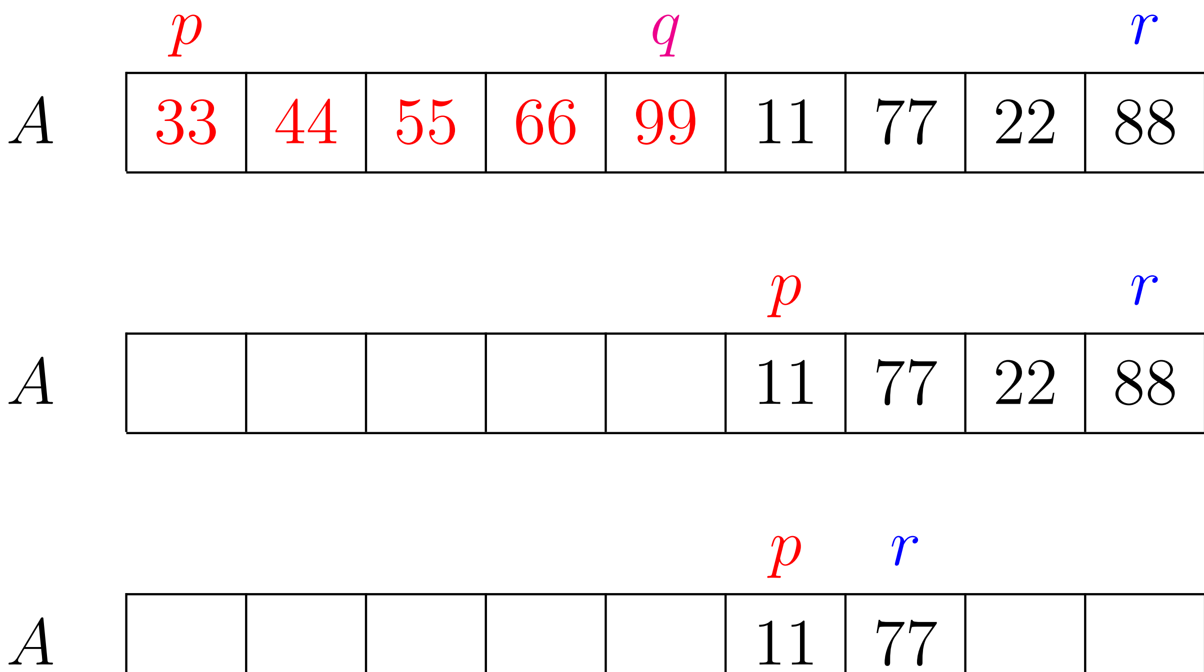
Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

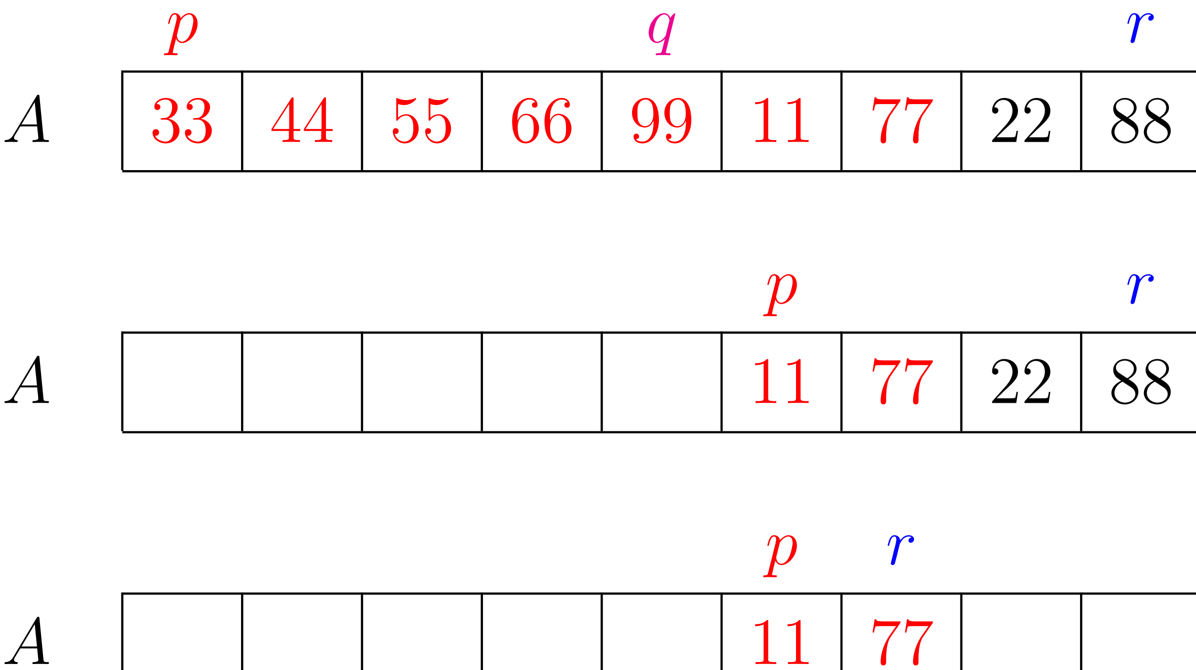
Merge-Sort



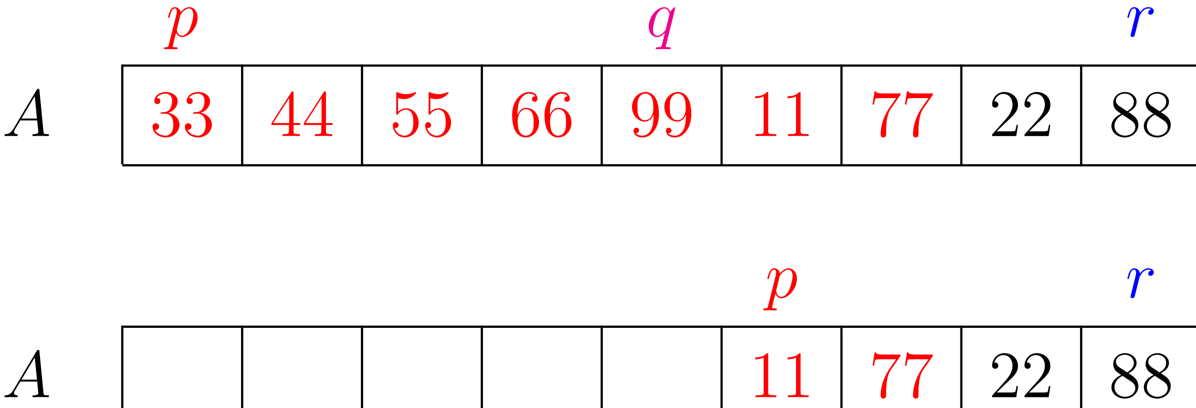
Merge-Sort



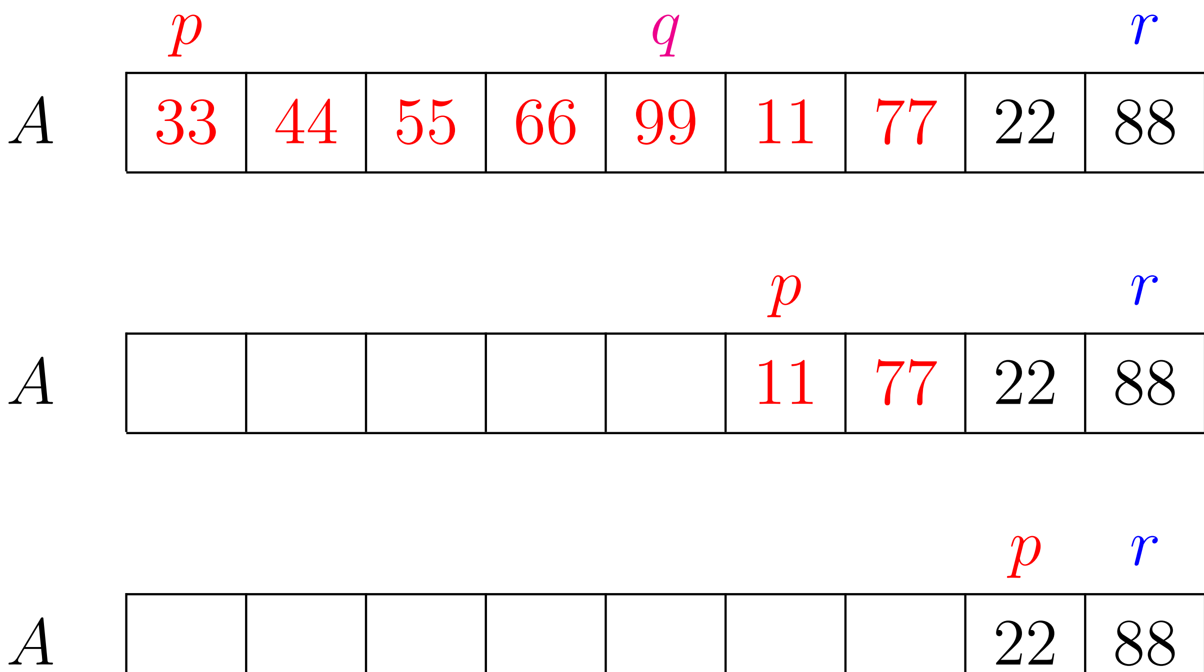
Merge-Sort



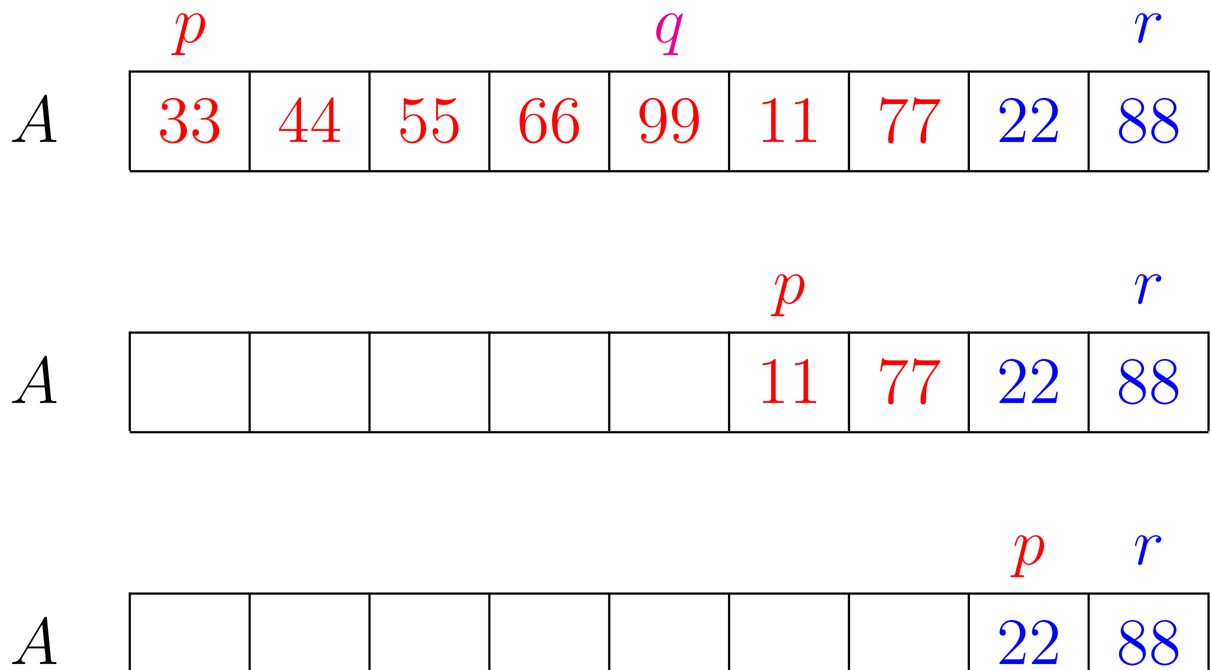
Merge-Sort



Merge-Sort



Merge-Sort

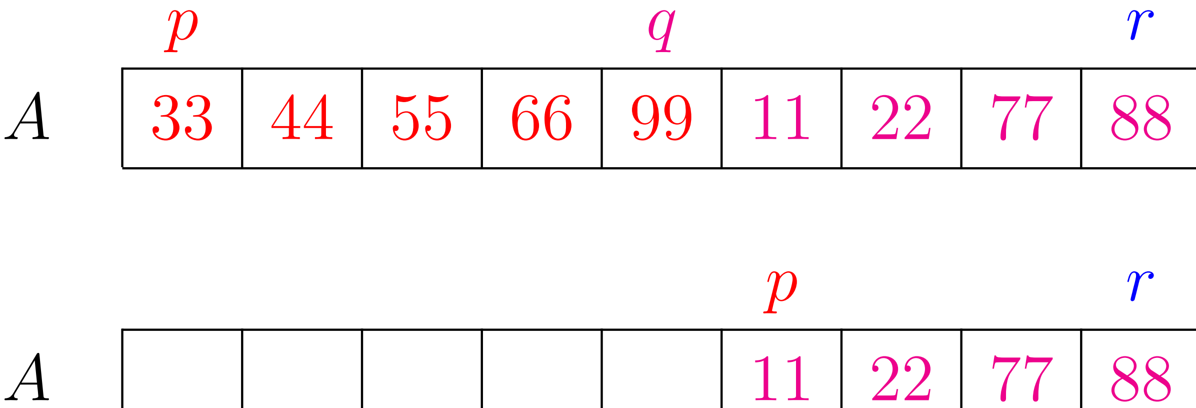


Merge-Sort

	p				q				r
A	33	44	55	66	99	11	77	22	88

						p			r
A						11	77	22	88

Merge-Sort



Merge-Sort

	p				q				r
A	33	44	55	66	99	11	22	77	88

Merge-Sort

A	p				q				r
	11	22	33	44	55	66	77	88	99

Merge-Sort

	p				q				r
A	11	22	33	44	55	66	77	88	99

Merge-Sort

Rearranja $A[p \,..\, r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

	p				q				r
A	55	33	66	44	99	11	77	22	88

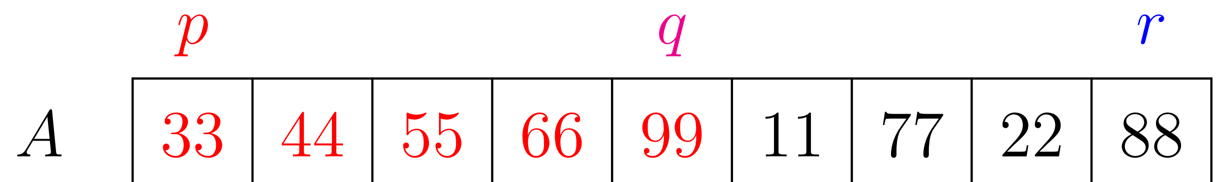
Merge-Sort

Rearranja $A[p \mathinner{.\,.} r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 **MERGE-SORT** (A, p, q)

---

4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)



Merge-Sort

Rearranja $A[p \,..\, r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 **MERGE-SORT** $(A, q + 1, r)$

---

5 INTERCALA (A, p, q, r)

	p				q				r
A	33	44	55	66	99	11	22	77	88

Merge-Sort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p+r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q+1, r)$
5 **INTERCALA** (A, p, q, r)

	p				q				r
A	11	22	33	44	55	66	77	88	99

Merge-Sort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

O algoritmo está correto?

Merge-Sort

Rearranja $A[p \,..\, r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

O algoritmo está correto?
A correção do algoritmo, que se apóia na correção do INTERCALA, pode ser demonstrada por indução em $n := r - p + 1$.

Merge-Sort

Rearranja $A[p \,..\, r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q+1, r)$
5 INTERCALA (A, p, q, r)

Consumo de tempo?

Merge-Sort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

Consumo de tempo?

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

Merge-Sort

```
MERGE-SORT (A, p, r)
1    se p < r
2        então q ← ⌊(p + r)/2⌋
3            MERGE-SORT (A, p, q)
4            MERGE-SORT (A, q + 1, r)
5            INTERCALA (A, p, q, r)
```

linha	consumo na linha
1	?
2	?
3	?
4	?
5	?

$T(n) = ?$

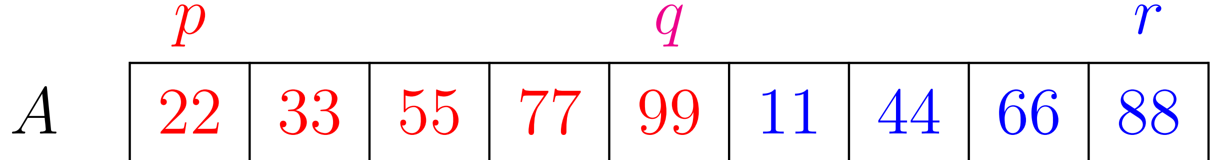
Melhores momentos

AULA 3

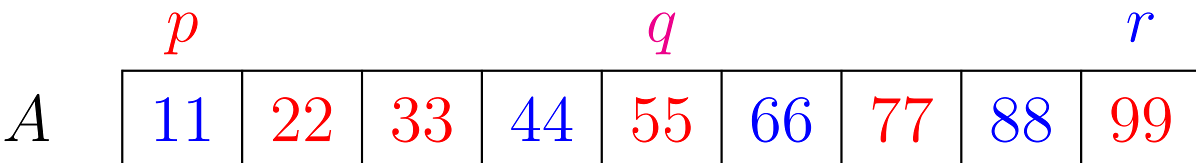
Análise da intercalação

Problema: Dados $A[p \mathinner{..} q]$ e $A[q{+}1 \mathinner{..} r]$ crescentes, rearranjar $A[p \mathinner{..} r]$ de modo que ele fique em ordem crescente.

Entra:



Sai:



Intercalação

```
INTERCALA (A, p, q, r)
 0    ▷ B[p . . r] é um vetor auxiliar
 1    para i ← p até q faça
 2          B[i] ← A[i]
 3    para j ← q + 1 até r faça
 4          B[r + q + 1 − j] ← A[j]
 5    i ← p
 6    j ← r
 7    para k ← p até r faça
 8          se B[i] ≤ B[j]
 9                então A[k] ← B[i]
10                      i ← i + 1
11                senão A[k] ← B[j]
12                      j ← j − 1
```

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é ($n := r - p + 1$):

linha	todas as execuções da linha
1	$= \quad q - p + 2 \;=\; n - r + q + 1$
2	$= \quad q - p + 1 \;=\; n - r + q$
3	$= \quad r - (q + 1) + 2 \;=\; n - q + p$
4	$= \quad r - (q + 1) + 1 \;=\; n - q + p - 1$
5	$= \quad 1$
6	$= \quad 1$
7	$= \quad r - p + 2 \;=\; n + 1$
8	$= \quad r - p + 1 \;=\; n$
9–12	$= \quad 2\,(r - p + 1) \;=\; 2n$

total $= \quad 8n - 2(r - p + 1) + 5 = \; 6n + 5$

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha
1–4	$\mathrm{O}(n)$
5–6	$\mathrm{O}(1)$
7	$n\mathrm{O}(1) = \mathrm{O}(n)$
8	$n\mathrm{O}(1) = \mathrm{O}(n)$
9–12	$n\mathrm{O}(1) = \mathrm{O}(n)$
total	$\mathrm{O}(4n + 1)$ = $\mathrm{O}(n)$

Conclusão:

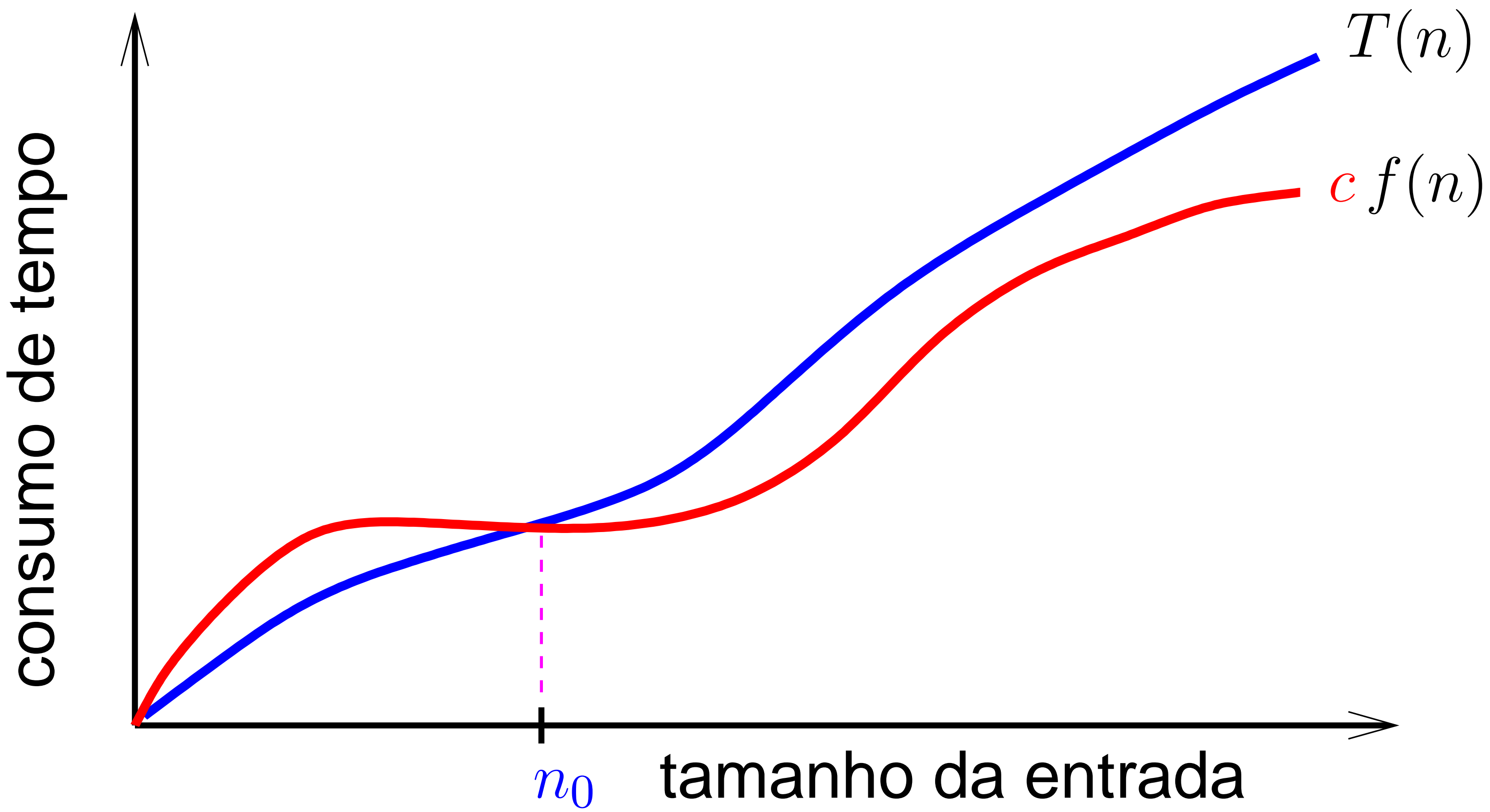
O algoritmo consome $\mathrm{O}(n)$ unidades de tempo.

Definição

Dizemos que $T(n)$ é $\Omega(f(n))$ se existem constantes positivas c e n_0 tais que

$$c\,f(n) \leq T(n)$$

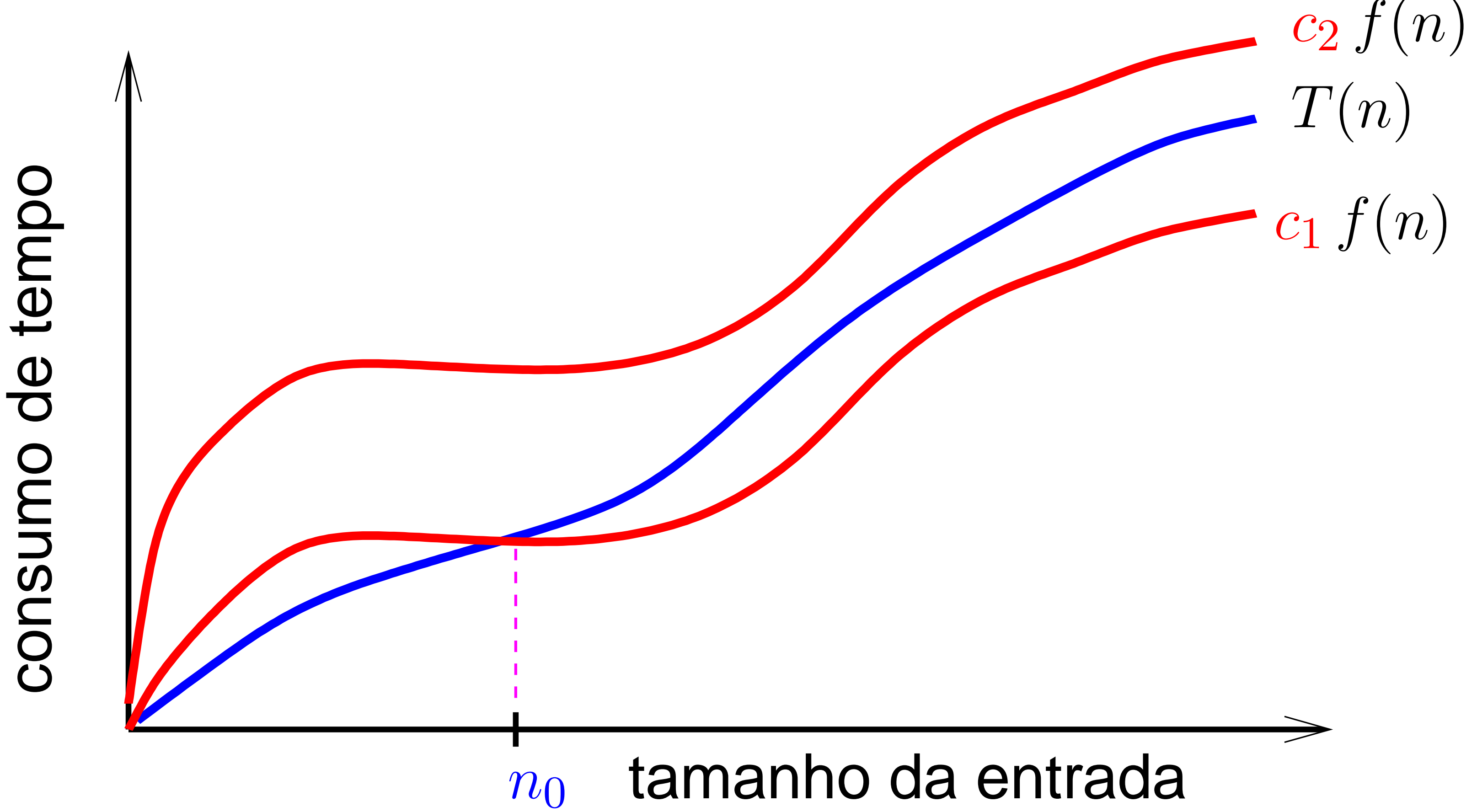
para todo $n \geq n_0$.



Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais. Dizemos que $T(n)$ é $\Theta(f(n))$ se

$$T(n) \text{ é } \mathrm{O}(f(n)) \text{ e } T(n) \text{ é } \Omega(f(n)).$$

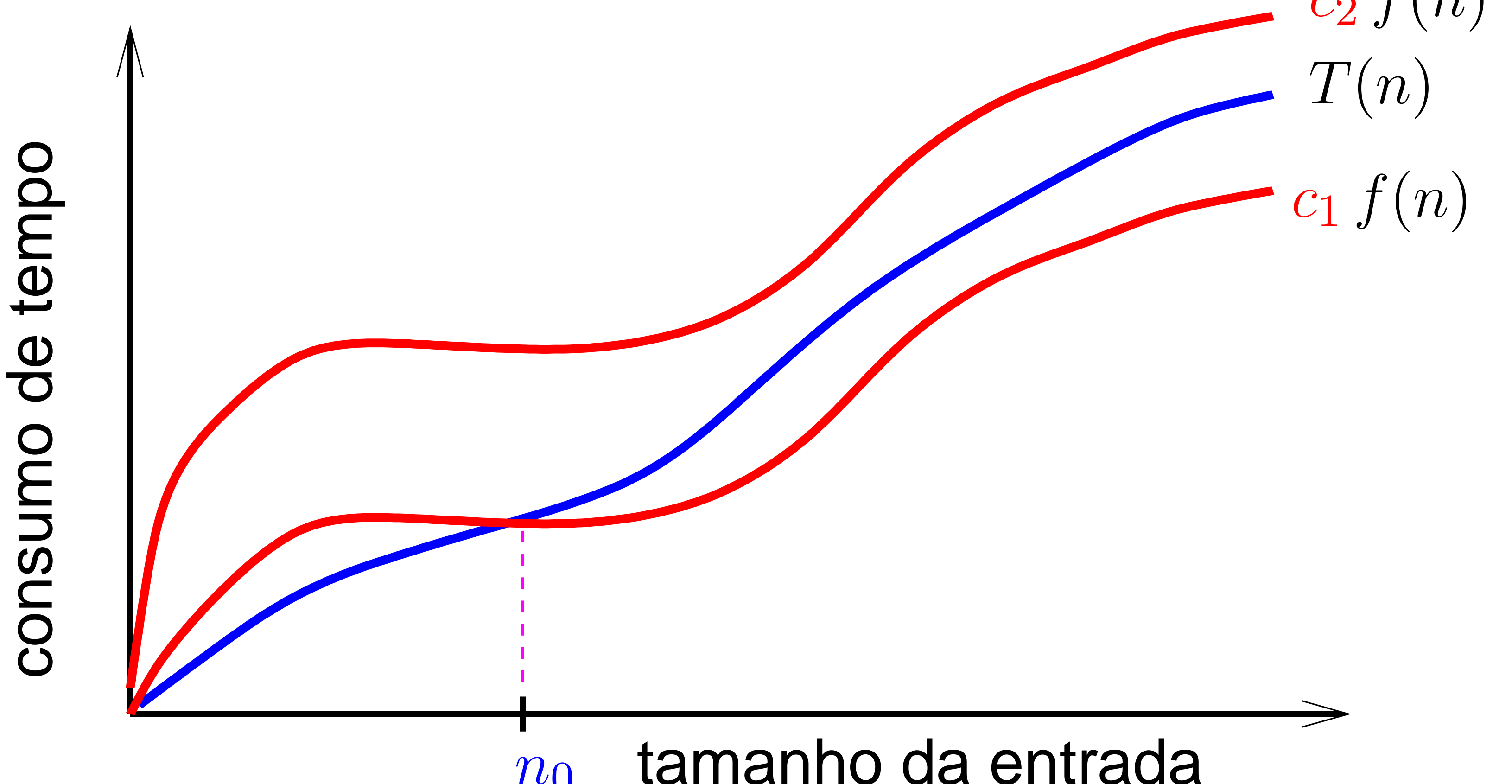


Definição

Dizemos que $T(n)$ é $\Theta(f(n))$ se se existem constantes positivas c_1, c_2 e n_0 tais que

$$c_1\, f(n) \;\leq\; T(n) \;\leq\; c_2\, f(n)$$

para todo $n \geq n_0$.



Exercícios

Exercício 6.A

Já sabemos que ORDENA-POR-INSERÇÃO é $O(n^2)$.
Mostre que o algoritmo é $\Omega(n)$.

Exercício 6.B

Mostre que ORDENA-POR-INSERÇÃO é $\Omega(n^2)$ no pior caso.

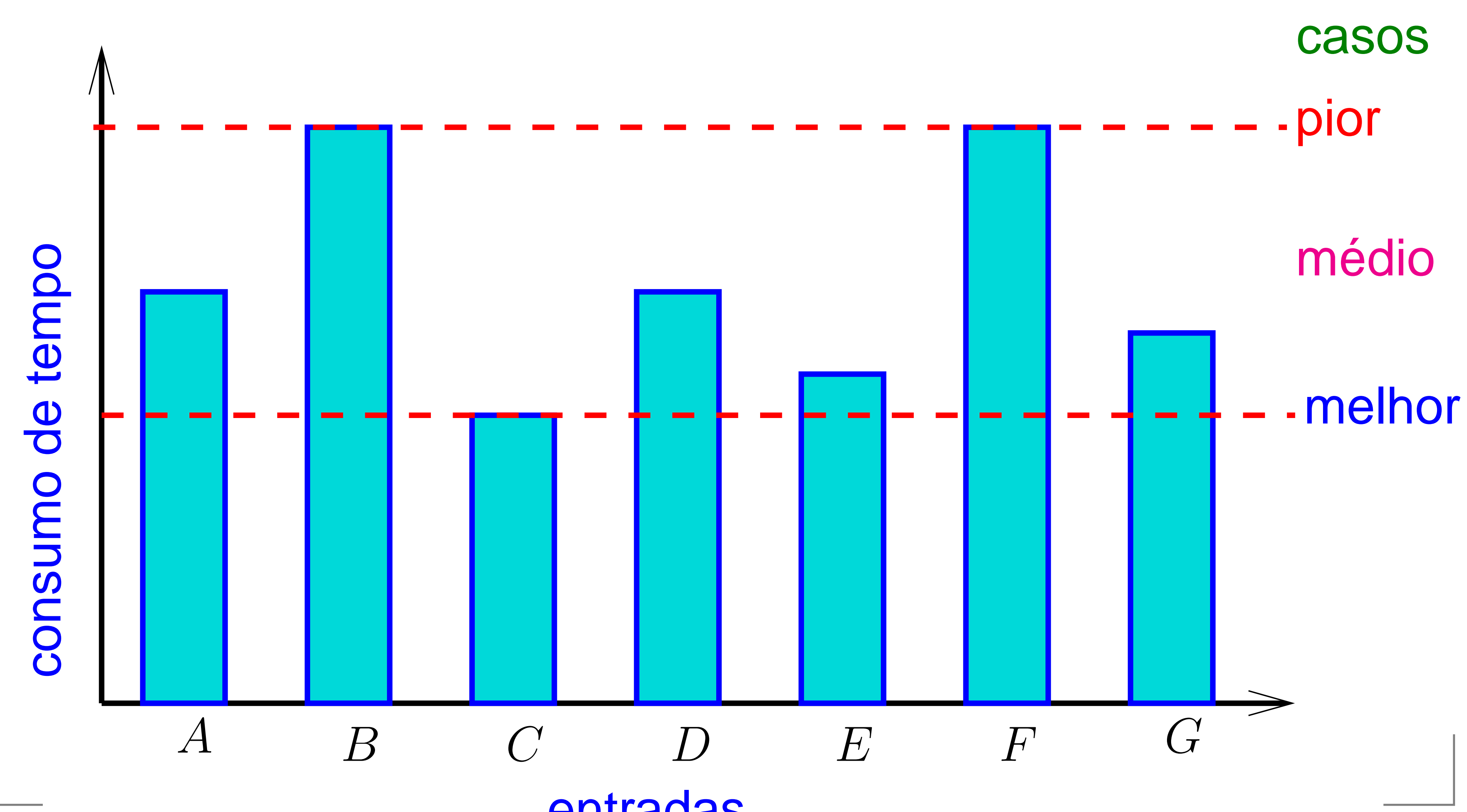
Exercício 6.C

Mostre que ORDENA-POR-INSERÇÃO é $O(n)$ no melhor caso.

Consumo de tempo

Mínimo, médio ou máximo?

Melhor caso, caso médio, pior caso?



AULA 4

Recursão (continuação)

CLRS 2.3
AU 2.6, 2.7, 2.9

"To understand recursion, we must first understand recursion."

Merge-Sort

Rearranja $A[p \,..\, r]$, com $p \leq r$, em ordem crescente.

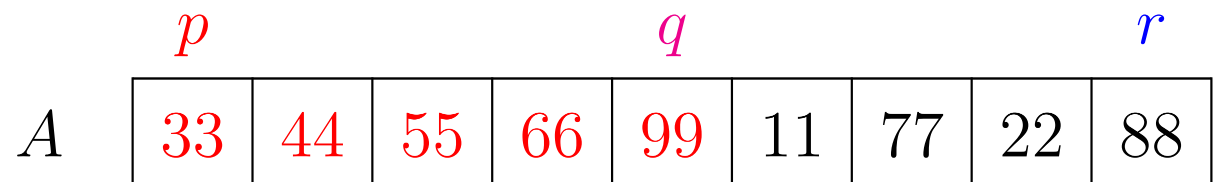
MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

	p				q				r
A	55	33	66	44	99	11	77	22	88

Merge-Sort

Rearranja $A[p \,..\, r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 **MERGE-SORT** (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)



Merge-Sort

Rearranja $A[p \mathinner{.\,.} r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 **MERGE-SORT** $(A, q + 1, r)$

---

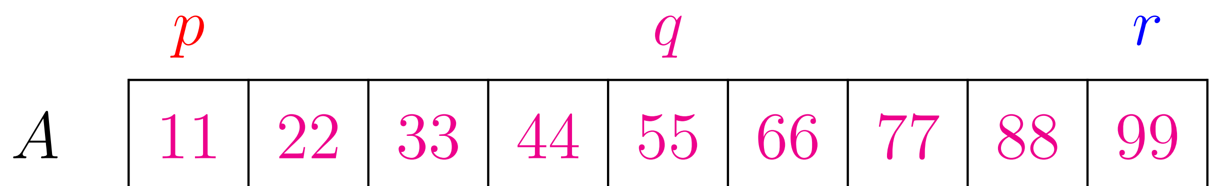
5 INTERCALA (A, p, q, r)

	p				q				r
A	33	44	55	66	99	11	22	77	88

Merge-Sort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 **INTERCALA** (A, p, q, r)



Merge-Sort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p+r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q+1, r)$
5 INTERCALA (A, p, q, r)

O algoritmo está correto?

Merge-Sort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

O algoritmo está correto?
A correção do algoritmo, que se apóia na correção do INTERCALA, pode ser demonstrada por indução em $n := r - p + 1$.

Merge-Sort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

Consumo de tempo?

Merge-Sort

Rearranja $A[p \mathinner{.\,.} r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q + 1, r)$
5 INTERCALA (A, p, q, r)

Consumo de tempo?

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

Merge-Sort

```
MERGE-SORT (A, p, r)
1    se p < r
2         então q ← ⌊(p + r)/2⌋
3              MERGE-SORT (A, p, q)
4              MERGE-SORT (A, q + 1, r)
5              INTERCALA (A, p, q, r)
```

linha	consumo na linha
1	?
2	?
3	?
4	?
5	?

$T(n) = ?$

Merge-Sort

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p+r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q+1, r)$
5 INTERCALA (A, p, q, r)

linha	consumo na linha
1	$\Theta(1)$
2	$\Theta(1)$
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	$\Theta(n)$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n+2)$$

Merge-Sort

$T(n)$:= consumo de tempo máximo quando $n = r - p + 1$

$$T(1) = \Theta(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Merge-Sort

$T(n)$:= consumo de tempo máximo quando $n = r - p + 1$

$$T(1) = \Theta(1)$$
$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Solução: $T(n)$ é $\Theta(???)$.

Demonstração: ...

Merge-Sort

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

$$T(1) = \Theta(1)$$
$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Solução: $T(n)$ é $\Theta(???)$.

Demonstração: ...

Veremos, mas antes estudaremos Recorrências.

Exercícios

Exercício 7.A

Problema: verificar se v é elemento de $A[p\,..\,r]$.
(Para quais valores de p e r faz sentido?)
Escreva um algoritmo recursivo que resolve o problema. O algoritmo deve devolver i tal que $A[i] = v$.

Exercício 7.B

Problema: verificar se v é elemento de vetor crescente $A[p\,..\,r]$. Escreva algoritmo recursivo de busca "linear" e outro de busca "binária".

Exercício 7.C

Escreva versão recursiva da ordenação por inserção. O algoritmo deve rearranjar em ordem crescente qualquer vetor dado $A[p\,..\,r]$.

Mais exercícios

Exercício 7.D (Versão sofisticada de busca)
Problema: verificar se v é elemento de vetor crescente $A[p \mathinner{.\,.} r]$. Escreva um algoritmo que devolva j tal que

$$A[j] \leq v < A[j+1] \; .$$

Quais os possíveis valores de j? Escreva duas versões: uma "linear" e uma "binária". Prove que os seus algoritmos estão corretos.

Exercício 7.E
Escreva uma versão recursiva do algoritmo de ordenação por seleção.

Exercício 7.F
Escreva uma versão iterativa do MERGE-SORT.

Recorrências

CLRS 4.1–4.2

AU 3.9, 3.11

Recorrências

Recorrência =

= “fórmula” que define uma função
em termos d’ela mesma

= algoritmo recursivo que calcula uma função

Exemplo 1

$$T(1) = 1$$
$$T(n) = T(n-1) + 3n + 2 \quad \text{para } n = 2, 3, 4, \ldots$$

Define função T sobre inteiros positivos:

n	1	2	3	4	5	6
$T(n)$	1	9	20	34	51	71

$T(n)$
1 **se** $n = 1$
2 **então devolva** 1
3 **senão devolva** $T(n-1) + 3n + 2$

Resolver uma recorrência

Resolver uma recorrência =

= obter uma “fórmula fechada” para $T(n)$

Método da substituição:

“chute” fórmula e verifique por indução

Exemplo 1 (continuação)

Eu acho que $T(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4$.

Exemplo 1 (continuação)

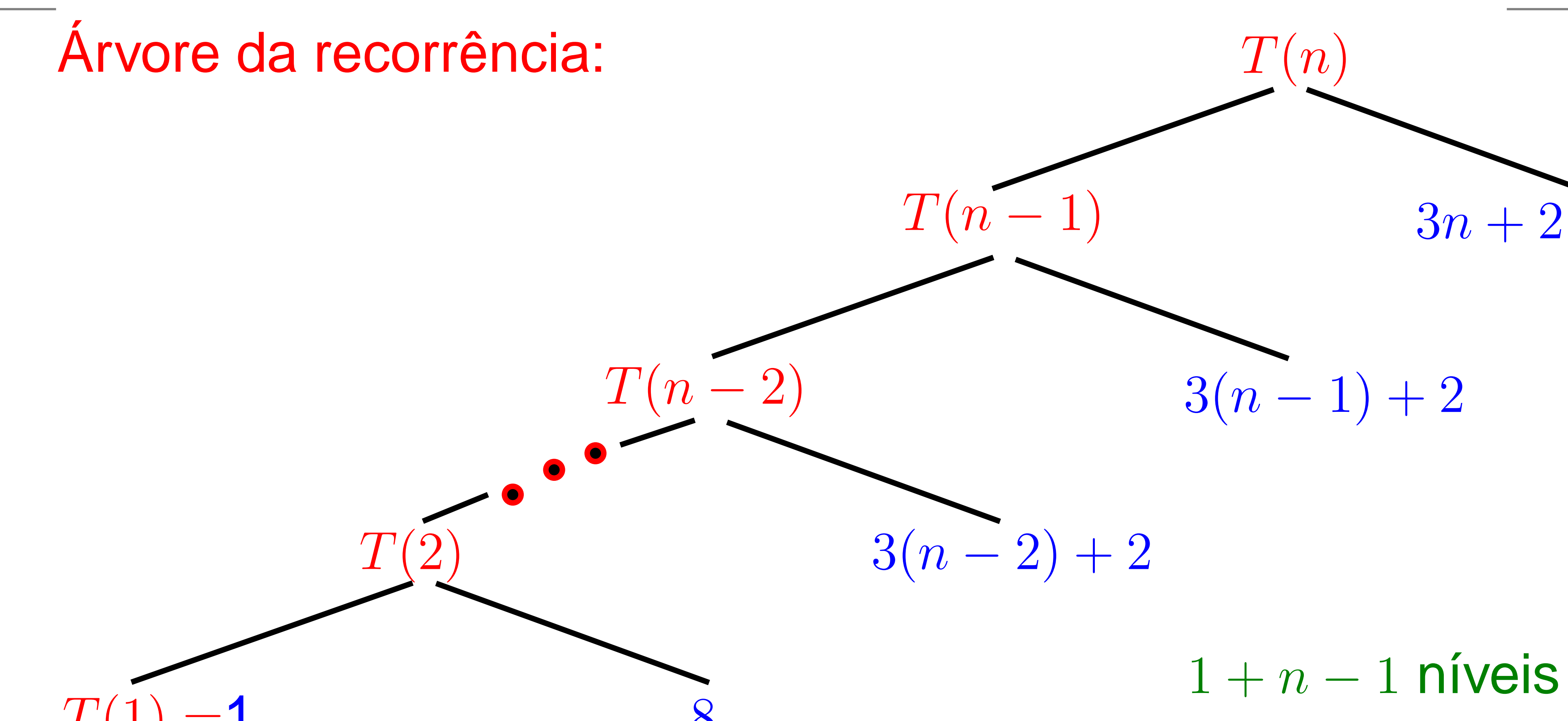
Eu acho que $T(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4$.

Verificação:
Se $n = 1$ então $T(n) = 1 = \frac{3}{2} + \frac{7}{2} - 4$.
Tome $n \geq 2$ e suponha que a fórmula está certa para $n - 1$:

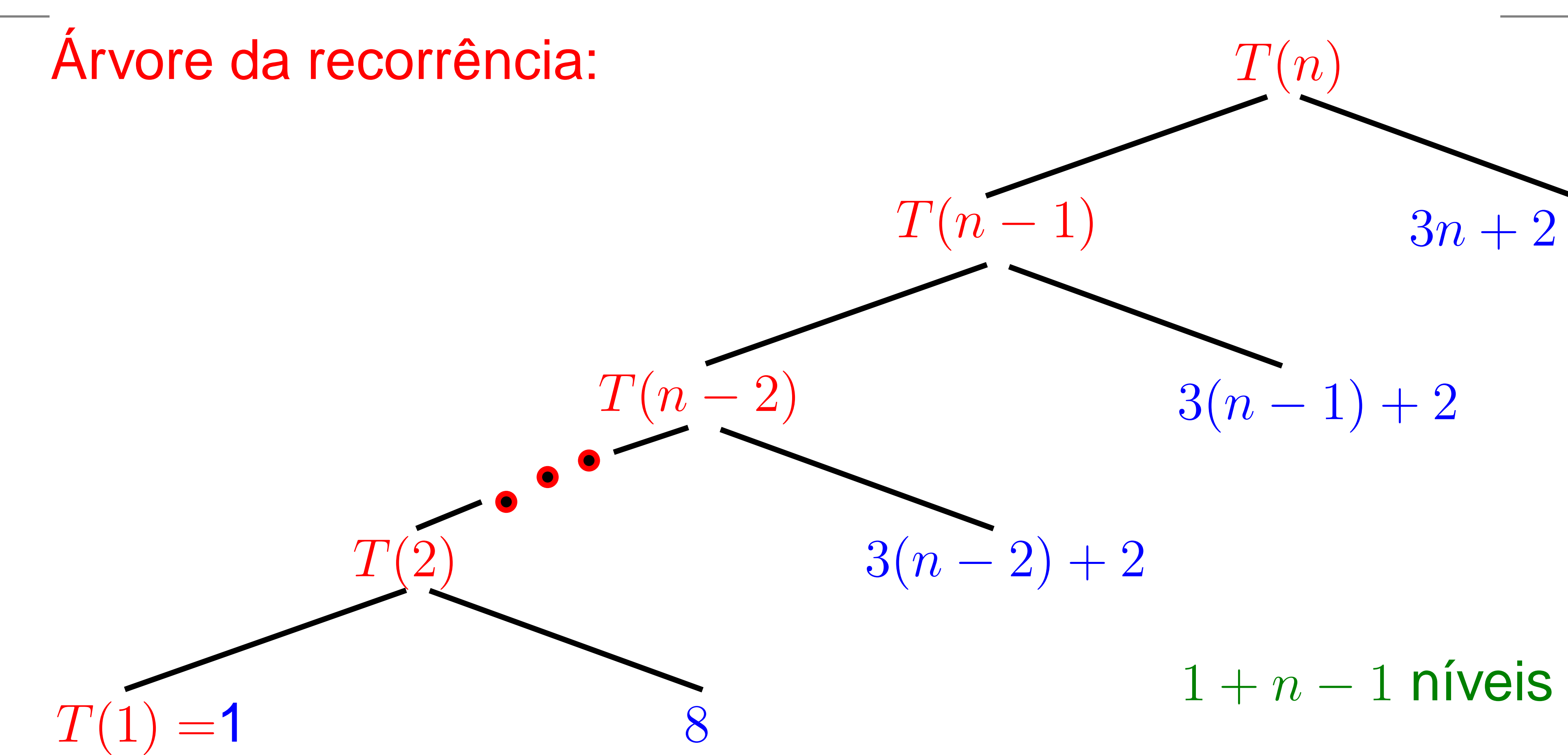
$$
\begin{aligned}
T(n) &= T(n-1) + 3n + 2 \\
&\overset{\text{hi}}{=} \tfrac{3}{2}(n-1)^2 + \tfrac{7}{2}(n-1) - 4 + 3n + 2 \\
&= \tfrac{3}{2}n^2 - 3n + \tfrac{3}{2} + \tfrac{7}{2}n - \tfrac{7}{2} - 4 + 3n + 2 \\
&= \tfrac{3}{2}n^2 + \tfrac{7}{2}n - 4\,.
\end{aligned}
$$

Bingo!

Como adivinhei fórmula fechada?



Como adivinhei fórmula fechada?



$$T(n) = (3n+2) + (3n-1) + \cdots + 8 + 1$$

$$= \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

Exemplo 2

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(n/2) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

Exemplo 2

$$T(1) = 1$$
$$T(n) = 2T(n/2) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \ldots$$

Não é uma recorrência! Não faz sentido!
$T(3)$ depende de $T(3/2)$...

Exemplo 3

$$\begin{aligned} G(1) &= 1 \\ G(n) &= 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16 .., 2^i, .. \end{aligned}$$

n	1	2	4	8	16
$G(n)$	1	18	66	190	494

Exemplo 3

$$
\begin{aligned}
G(1) &= 1 \\
G(n) &= 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16\,.\,., 2^i, \,.\,.
\end{aligned}
$$

n	1	2	4	8	16
$G(n)$	1	18	66	190	494

Fórmula fechada: $G(n) =$???

Hmmmmmm

Acho que $G(n)$ é da forma $n \lg n$ ...

n	$G(n)$	$6n \lg n$	$7n \lg n$	$8n \lg n$	n^2
1	1	0	0	0	1
2	18	12	14	16	4
4	66	48	56	64	16
8	190	144	168	192	64
16	494	384	448	512	256
32	1214	960	1120	1280	1024
64	2878	2304	2688	3072	4096
128	6654	5376	6272	7168	16384
256	15102	12288	14336	16384	65536

Chute

Acho que a fórmula fechada é

$$G(n) = 7n \lg n + 3n - 2$$

para $n = 1, 2, 4, 8, 16, 32 \ldots$

Lá vamos nós outra vez. . .

Chute

$G(n) = 7n\lg n + 3n - 2$ para $n = 1, 2, 4, 8, 16, 32\ldots$

Chute

$G(n) = 7n\lg n + 3n - 2$ para $n = 1, 2, 4, 8, 16, 32\ldots$

Prova: Se $n = 1$ então $G(n) = 1 = 7 \cdot 1\lg 1 + 3 \cdot 1 - 2$.
Se $n \geq 2$ então

$$\begin{aligned} G(n) &= 2G(\tfrac{n}{2}) + 7n + 2 \\ &\overset{\text{hi}}{=} 2\left(7\tfrac{n}{2}\lg\tfrac{n}{2} + 3\,\tfrac{n}{2} - 2\right) + 7n + 2 \\ &= 7n(\lg n - 1) + 3n - 4 + 7n + 2 \\ &= 7n\lg n - 7n + 3n - 2 + 7n \\ &= 7n\lg n + 3n - 2 \end{aligned}$$

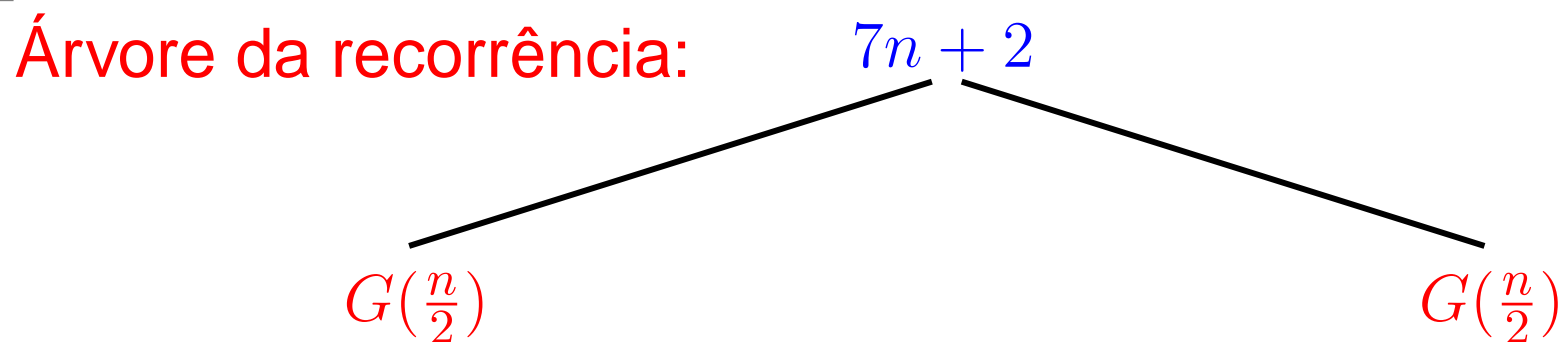
Iiiiiééééssss!

Como adivinhei fórmula fechada?

Árvore da recorrência: $G(n)$

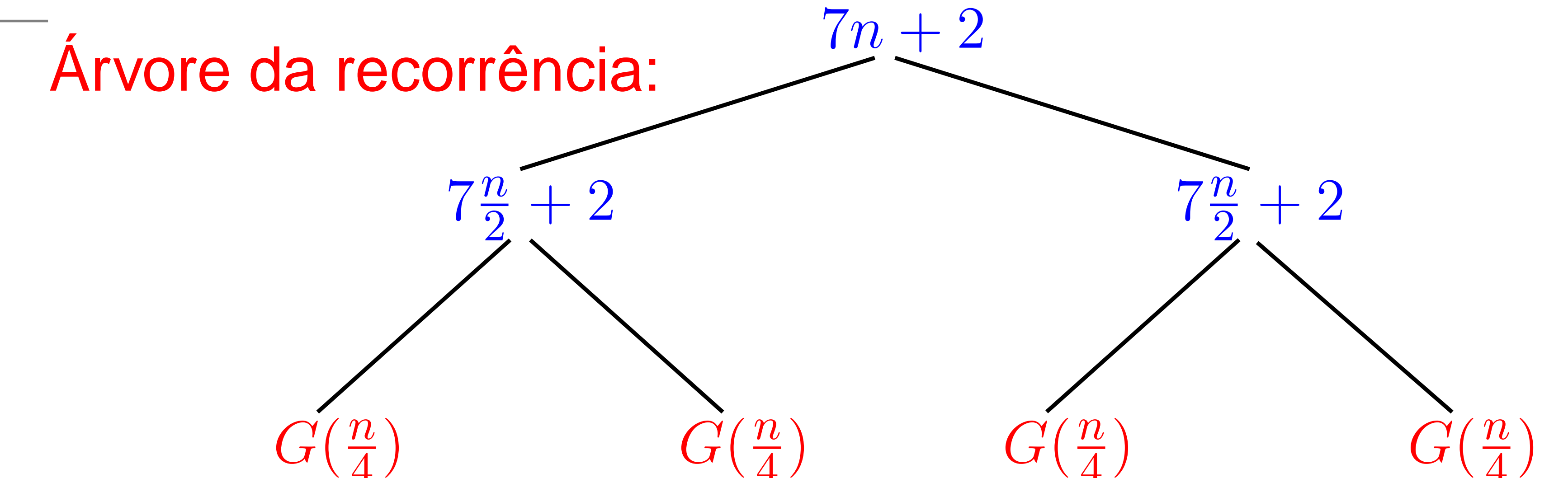
Como adivinhei fórmula fechada?

Árvore da recorrência:



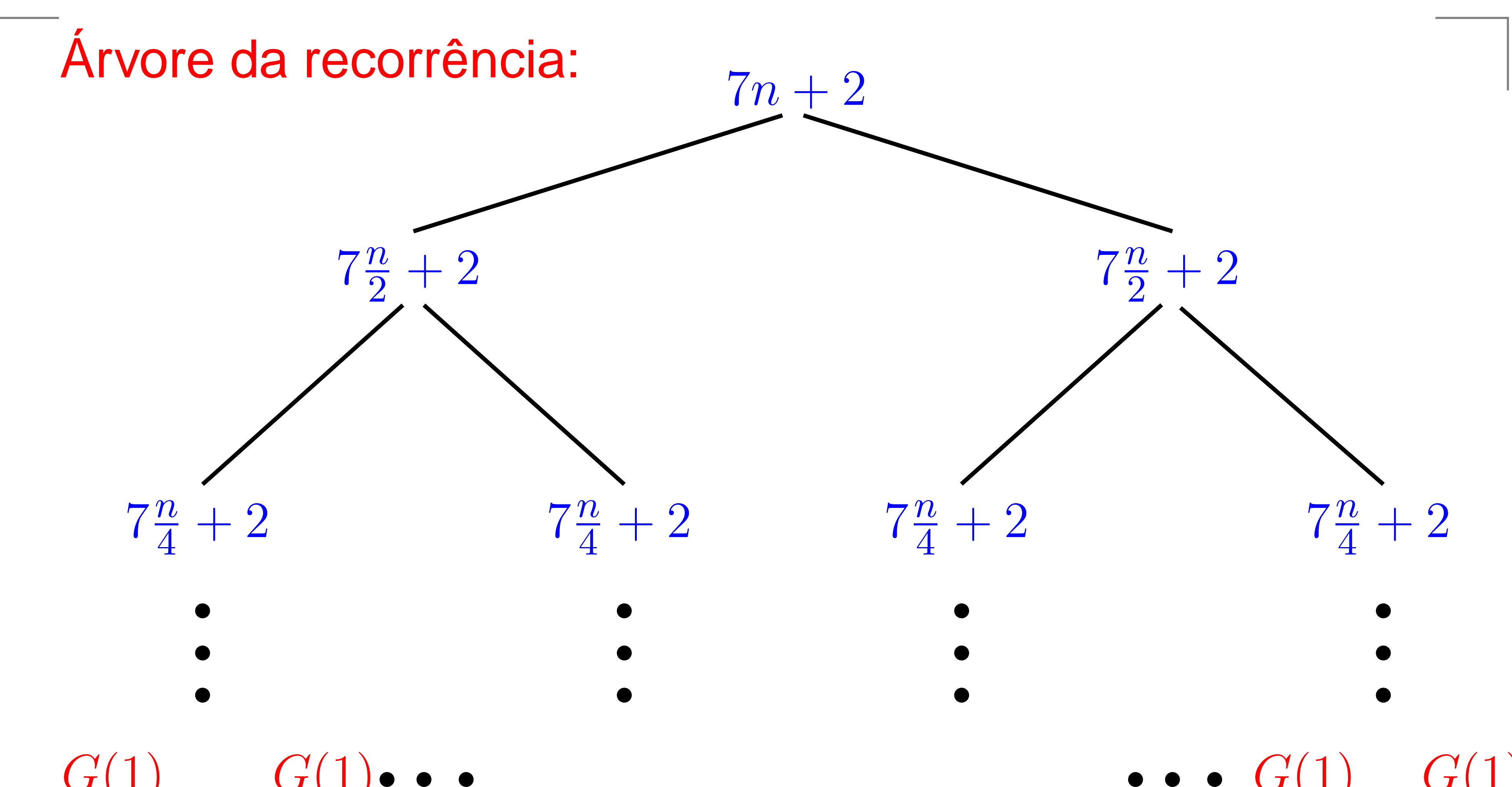
Como adivinhei fórmula fechada?

Árvore da recorrência:



Como adivinhei fórmula fechada?

Árvore da recorrência:



total de $1 + \lg n$ níveis

Contas

nível	0	1	2	$\ldots$	$k-1$	k
soma	$7n+2$	$7n+4$	$7n+8$	$\ldots$	$7n+2^k$	$2^k G(1)$

$n = 2^k \qquad k = \lg n$

Contas

nível	0	1	2	$\ldots$	$k-1$	k
soma	$7n+2$	$7n+4$	$7n+8$	$\ldots$	$7n+2^k$	$2^k G(1)$

$n = 2^k \qquad k = \lg n$

$$
\begin{aligned}
G(n) &= 7n + 2^1 + 7n + 2^2 + \cdots + 7n + 2^k + 2^k G(1) \\
&= 7n\,k + (2 + 4 + \cdots + 2^k) + 2^k \\
&= 7n\,k + 2 \cdot 2^k - 2 + n \\
&= 7n \lg n + 2n - 2 + n \quad (k = \lg n) \\
&= 7n \lg n + 3n - 2
\end{aligned}
$$

Iiiiééééssss

Série geométrica

Para $x \neq 1$, quanto vale $1 + x^1 + x^2 + \cdots + x^{k-1} + x^k$?

(CLRS (A.5), p.1060)

Solução: Seja $S_k := 1 + x^1 + x^2 + \cdots + x^{k-1} + x^k$.
Temos que $xS_k = x^1 + x^2 + x^3 + \cdots + x^k + x^{k+1}$.
Logo,

$$xS_k - S_k = x^{k+1} - 1$$

e

$$S_k = \frac{x^{k+1} - 1}{x - 1}.$$

Conclusão:

$$1 + x^1 + x^2 + \cdots + x^{k-1} + x^k = \frac{x^{k+1}-1}{x-1}.$$

Exemplo 3 (continuação)

É mais fácil mostrar que $G(n)$ é $\mathrm{O}(n \lg n)$.

Vou provar que $G(n) \leq 9n \lg n$ para $n = 2, 4, 8, 16, \ldots, 2^i, \ldots$

Exemplo 3 (continuação)

É mais fácil mostrar que $G(n)$ é $\mathrm{O}(n \lg n)$.

Vou provar que $G(n) \leq 9n \lg n$ para $n = 2, 4, 8, 16, \ldots, 2^i, \ldots$

Prova: Se $n = 2$, $G(n) = 18 = 9 \cdot 2 \cdot \lg 2$.

Se $n \geq 4$,

$$G(n) = 2G(n/2) + 7n + 2$$

hi

$$\leq 2 \cdot 9(n/2) \lg(n/2) + 7n + 2$$

$$= 9n\,(\lg n - 1) + 7n + 2$$

$$= 9n \lg n - 2n + 2$$

$$< 9n \lg n \quad \text{(pois } n > 1\text{)}$$

Da linha 1 para a linha 2, a hipótese de indução vale pois $2 \leq n/2 < n$.

Exercícios

Exercício 8.A

Seja T a função definida pela recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n-1) + 2n - 2 \qquad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

Verifique que a recorrência é honesta, ou seja, de fato define uma função. A partir da árvore da recorrência, adivinhe uma boa delimitação assintótica para $T(n)$; dê a resposta em notação O. Prove a delimitação pelo método da substituição.

Exercício 8.B

Resolva a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n-2) + 2n + 1 \qquad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

Desenhe a árvore da recorrência. Dê a resposta em notação O.

Mais exercícios

Exercício 8.C

Resolva a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(2) &= 2 \\ T(n) &= T(n-2) + 2n + 1 \quad \text{para } n = 3, 4, 5, 6, \ldots \end{aligned}$$

Exercício 8.D

Resolva a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2) + 1 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

Exercício 8.E

Resolva a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lfloor n/2 \rfloor) + 1 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

Mais exercícios ainda

Exercício 8.F

Resolva a recorrência

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= T(\lfloor n/2 \rfloor) + n \qquad \text{para } n = 2, 3, 4, 5, \ldots
\end{aligned}
$$

Exercício 8.G

Resolva a recorrência

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= 2T(\lfloor n/2 \rfloor) + n \qquad \text{para } n = 2, 3, 4, 5, \ldots
\end{aligned}
$$

Exercício 8.H

Resolva a recorrência

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= 2T(\lceil n/2 \rceil) + n \qquad \text{para } n = 2, 3, 4, 5, \ldots
\end{aligned}
$$

Melhores momentos

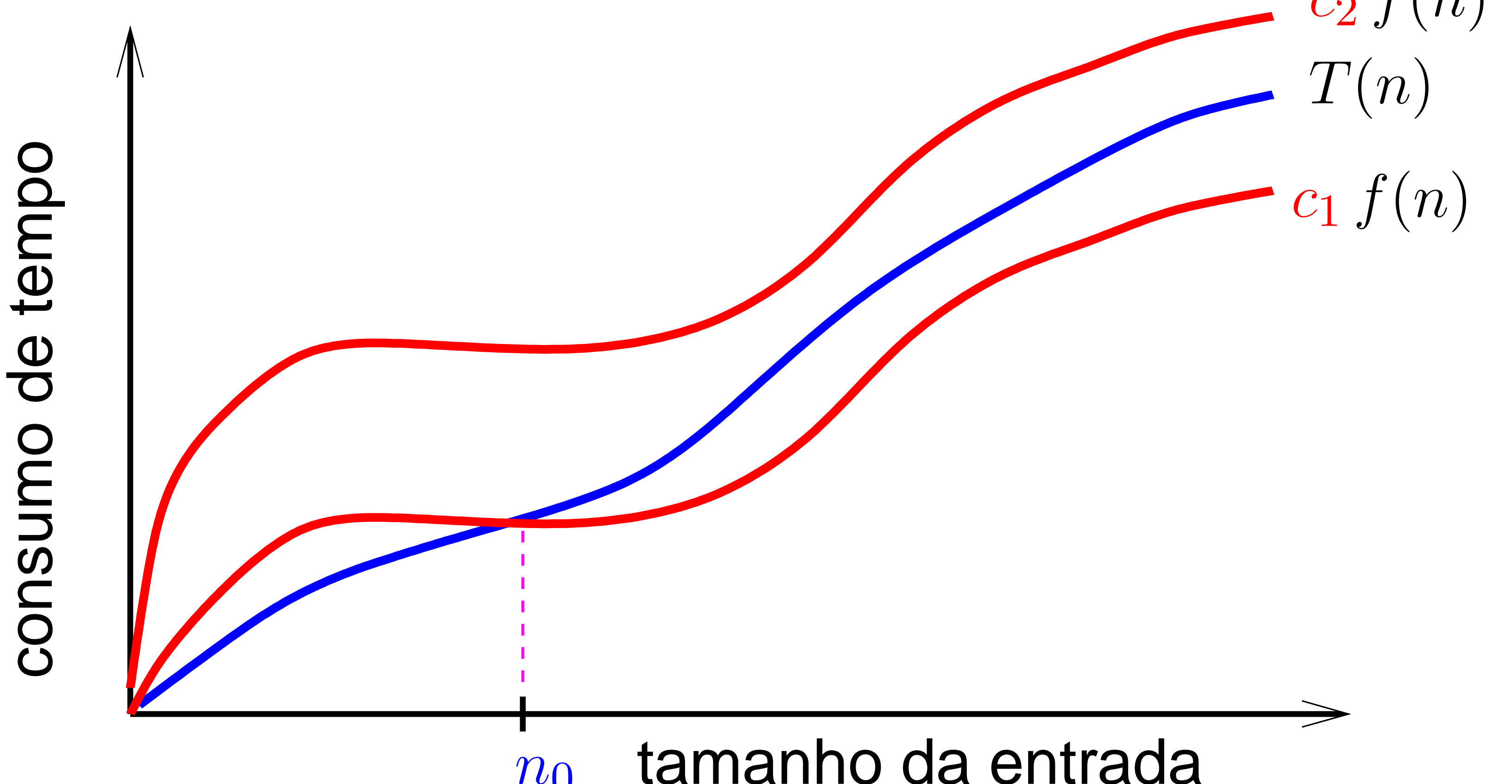
AULA 4

Definição

Dizemos que $T(n)$ é $\Theta(f(n))$ se se existem constantes positivas c_1, c_2 e n_0 tais que

$$c_1 \, f(n) \;\leq\; T(n) \;\leq\; c_2 \, f(n)$$

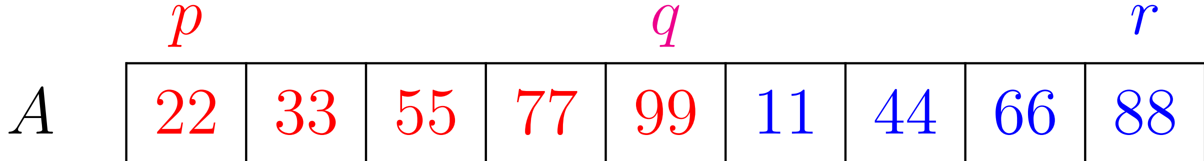
para todo $n \geq n_0$.



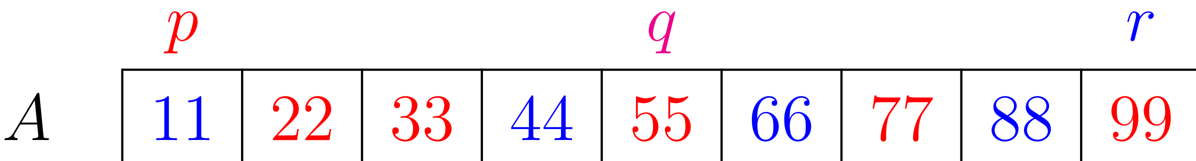
Análise da intercalação

Problema: Dados $A[p \,..\, q]$ e $A[q{+}1 \,..\, r]$ crescentes, rearranjar $A[p \,..\, r]$ de modo que ele fique em ordem crescente.

Entra:



Sai:



Intercalação

```
INTERCALA (A, p, q, r)
 0    ▷ B[p . . r] é um vetor auxiliar
 1    para i ← p até q faça
 2          B[i] ← A[i]
 3    para j ← q + 1 até r faça
 4          B[r + q + 1 − j] ← A[j]
 5    i ← p
 6    j ← r
 7    para k ← p até r faça
 8          se B[i] ≤ B[j]
 9                então A[k] ← B[i]
10                      i ← i + 1
11                senão A[k] ← B[j]
12                      j ← j − 1
```

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é ($n := r - p + 1$):

linha	todas as execuções da linha
1	$= \quad q - p + 2 = n - r + q + 1$
2	$= \quad q - p + 1 = n - r + q$
3	$= \quad r - (q+1) + 2 = n - q + p$
4	$= \quad r - (q+1) + 1 = n - q + p - 1$
5	$= \quad 1$
6	$= \quad 1$
7	$= \quad r - p + 2 = n + 1$
8	$= \quad r - p + 1 = n$
9–12	$= \quad 2\,(r - p + 1) = 2n$
total	$= \quad 8n - 2(r - p + 1) + 5 = 6n + 5$

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha
1–4	$\Theta(n)$
5–6	$\Theta(1)$
7	$n\Theta(1) = \Theta(n)$
8	$n\Theta(1) = \Theta(n)$
9–12	$n\Theta(1) = \Theta(n)$
total	$\Theta(4n + 1)$ = $\Theta(n)$

Conclusão:

O algoritmo consome $\Theta(n)$ unidades de tempo.

Recorrências

$$T(1) = 1$$
$$T(n) = T(n-1) + 3n + 2 \quad \text{para } n = 2, 3, 4, \ldots$$

Define função T sobre inteiros positivos:

n	1	2	3	4	5	6
$T(n)$	1	9	20	34	51	71

$T(n)$
1 **se** $n = 1$
2 **então devolva** 1
3 **senão devolva** $T(n-1) + 3n + 2$

Método da substituição

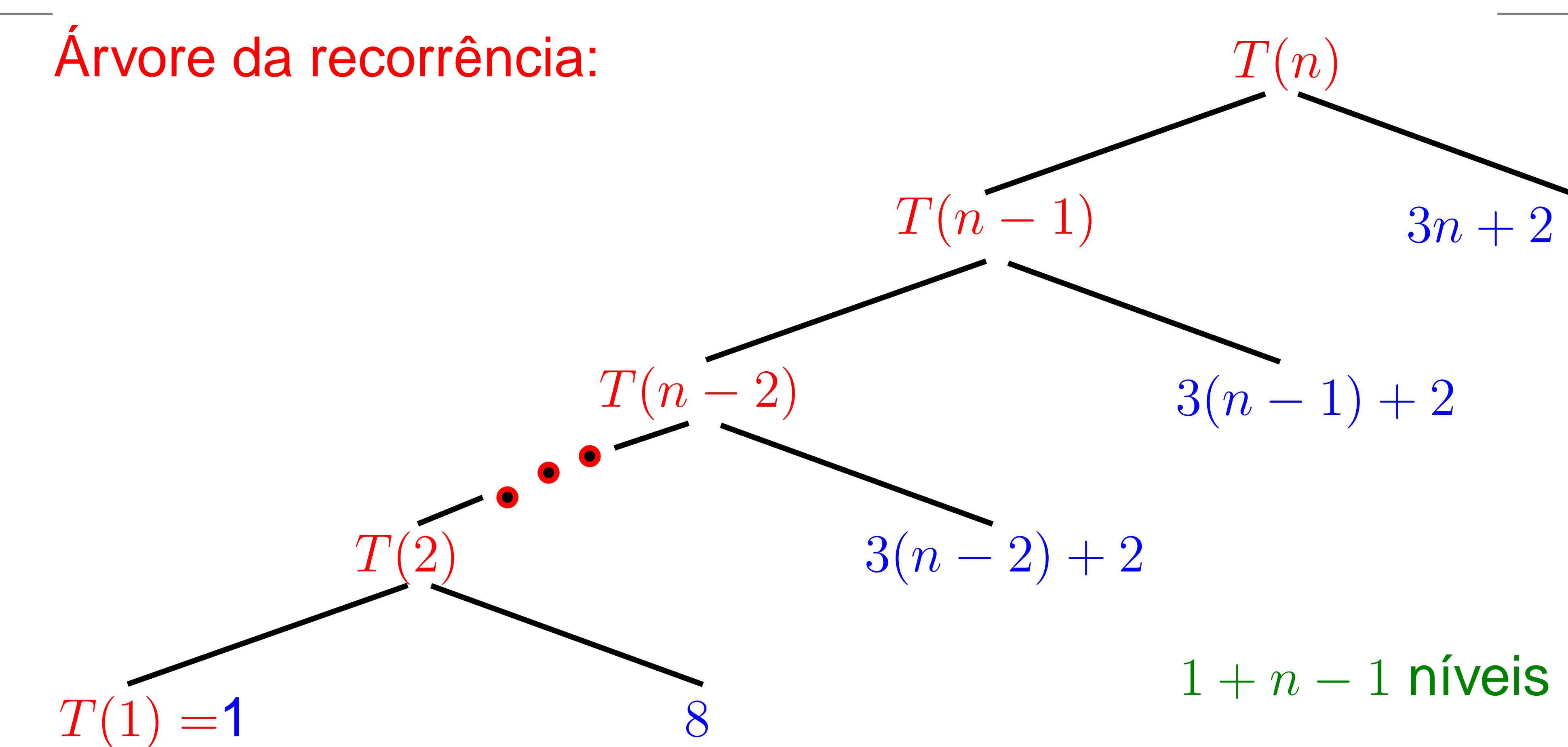
Chute:
Eu acho que $T(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4$.

Verificação por indução:
Se $n = 1$ então $T(n) = 1 = \frac{3}{2} + \frac{7}{2} - 4$.
Tome $n \geq 2$ e suponha que a fórmula está certa para $n - 1$:

$$
\begin{aligned}
T(n) &= T(n-1) + 3n + 2 \\
&\overset{\text{hi}}{=} \frac{3}{2}(n-1)^2 + \frac{7}{2}(n-1) - 4 + 3n + 2 \\
&= \frac{3}{2}n^2 - 3n + \frac{3}{2} + \frac{7}{2}n - \frac{7}{2} - 4 + 3n + 2 \\
&= \frac{3}{2}n^2 + \frac{7}{2}n - 4\,.
\end{aligned}
$$

Bingo!

Como adivinhei fórmula fechada?



$$T(n) = (3n+2) + (3n-1) + \cdots + 8 + 1$$
$$= \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

Exemplo 3

$$
\begin{aligned}
G(1) &= 1 \\
G(n) &= 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16 \ldots, 2^i, \ldots
\end{aligned}
$$

n	1	2	4	8	16
$G(n)$	1	18	66	190	494

Fórmula fechada: $G(n) =$???

Resolvendo a recorrência

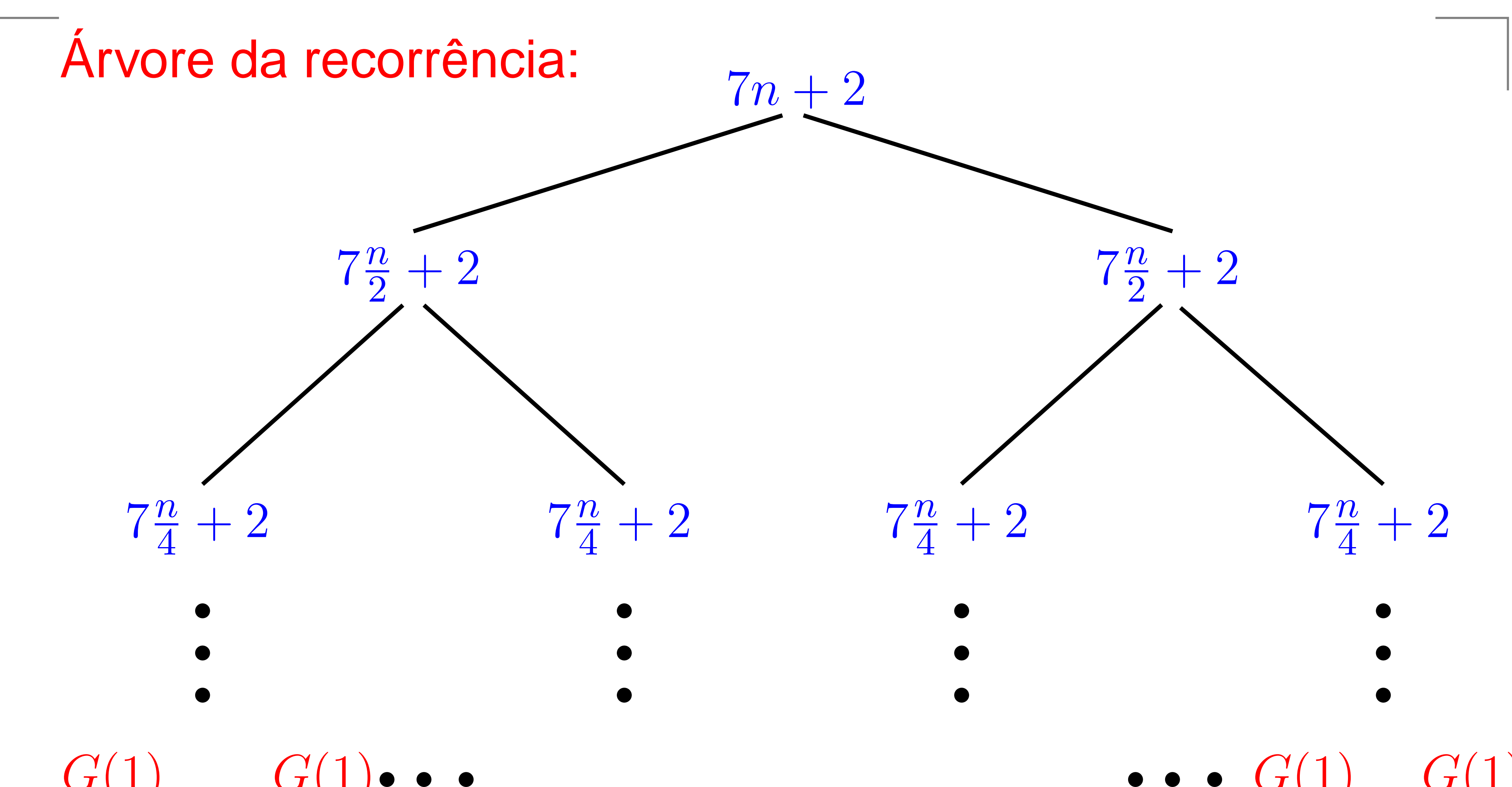
$G(n) = 7n \lg n + 3n - 2$ para $n = 1, 2, 4, 8, 16, 32 \ldots$

Prova: Se $n = 1$ então $G(n) = 1 = 7 \cdot 1 \lg 1 + 3 \cdot 1 - 2$.
Se $n \geq 2$ então

$$
\begin{aligned}
G(n) &= 2G(\tfrac{n}{2}) + 7n + 2 \\
&\overset{\text{hi}}{=} 2\left(7\tfrac{n}{2} \lg \tfrac{n}{2} + 3\,\tfrac{n}{2} - 2\right) + 7n + 2 \\
&= 7n(\lg n - 1) + 3n - 4 + 7n + 2 \\
&= 7n \lg n - 7n + 3n - 2 + 7n \\
&= 7n \lg n + 3n - 2
\end{aligned}
$$

Iiiiiéééésss!

Como adivinhei fórmula fechada?



total de $1+\lg n$ níveis

Contas

nível	0	1	2	...	$k-1$	k
soma	$7n+2$	$7n+4$	$7n+8$	...	$7n+2^k$	$2^k G(1)$

$n = 2^k \qquad k = \lg n$

$$
\begin{aligned}
G(n) &= 7n + 2^1 + 7n + 2^2 + \cdots + 7n + 2^k + 2^k G(1) \\
&= 7n\,k + (2 + 4 + \cdots + 2^k) + 2^k \\
&= 7n\,k + 2 \cdot 2^k - 2 + n \\
&= 7n \lg n + 2n - 2 + n \quad (k = \lg n) \\
&= 7n \lg n + 3n - 2 \qquad \text{(-5)}
\end{aligned}
$$

Iiiiééééssss

Exemplo 3 (continuação)

É mais fácil mostrar que $G(n) = \mathrm{O}(n \lg n)$.
Vou provar que $G(n) \leq 9n \lg n$ para $n = 2, 4, 8, 16, \ldots, 2^i, \ldots$

Prova: Se $n = 2$, $G(n) = 18 = 9 \cdot 2 \cdot \lg 2$.
Se $n \geq 4$,

$$\begin{aligned} G(n) &= 2G(n/2) + 7n + 2 \\ &\overset{\text{hi}}{\leq} 2 \cdot 9(n/2)\lg(n/2) + 7n + 2 \\ &= 9n(\lg n - 1) + 7n + 2 \\ &= 9n \lg n - 2n + 2 \\ &< 9n \lg n \quad \text{(pois } n \geq 2\text{)} \end{aligned}$$

Da linha 1 para a linha 2, a hipótese de indução vale pois $2 \leq n/2 < n$.

Exemplo 3 (novamente)

É mais fácil mostrar que $G(n) = \mathrm{O}(n \lg n)$.
Vou provar que $G(n) \leq 8n \lg n$ para $n = 8, 16, \ldots, 2^i, \ldots$

Prova: Se $n = 8$, $G(n) = 190 < 192 = 8 \cdot 8 \cdot \lg 8 = 64 \cdot 3$.
Se $n \geq 16$,

$$
\begin{aligned}
G(n) &= 2G(n/2) + 7n + 2 \\
&\overset{\text{hi}}{\leq} 2 \cdot 8(n/2)\lg(n/2) + 7n + 2 \\
&= 8n(\lg n - 1) + 7n + 2 \\
&= 8n \lg n - n + 2 \\
&< 8n \lg n \quad \text{(pois } n \geq 16\text{)}
\end{aligned}
$$

Da linha 1 para a linha 2, a hipótese de indução vale pois $2 \leq n/2 < n$.

AULA 5

Recorrências (continuação)

CLRS 4.1–4.2

AU 3.9, 3.11

Exemplo 4

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 6n + 5 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259

Exemplo 4

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 6n + 5 \quad \text{para } n = 2, 3, 4, 5, \ldots
\end{aligned}
$$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259

$T(n) = \mathrm{O}(\text{???})$

Exemplo 4

Vou mostrar que $T(n) \leq 20\,n \lg n$ para $n = 2, 3, 4, \ldots$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259
$20\,n\lfloor \lg n \rfloor$	0	40	60	160	200	240	280	480	540	600

Exemplo 4

Vou mostrar que $T(n) \leq 20\,n \lg n$ para $n = 2, 3, 4, \ldots$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259
$20\,n\lfloor\lg n\rfloor$	0	40	60	160	200	240	280	480	540	600

Prova:
Se $n = 2, 3$, então $T(n) \leq 20\,n\lfloor\lg n\rfloor \leq 20\,n \lg n$, como mostra a tabela.

Note que a base da indução é $n = 2, 3$. Por quê???

Exemplo 4

Prova: (continuação) Se $n \geq 4$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n + 5$$

hi

$$\leq 20 \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + 20 \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$

$$\leq 20 \left\lceil \frac{n}{2} \right\rceil \lg n + 20 \left\lfloor \frac{n}{2} \right\rfloor \lg \frac{n}{2} + 6n + 5$$

$$= 20 \left\lceil \frac{n}{2} \right\rceil \lg n + 20 \left\lfloor \frac{n}{2} \right\rfloor (\lg n - 1) + 6n + 5$$

$$= 20 \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \right) \lg n - 20 \left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$

$$= 20\, n \lg n - 20 \left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$

$$\leq 20\, n \lg n \quad \text{(pois } n \geq 4\text{)}$$

Iiiiééééssss!

Como achei as contantes?

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n + 5 \quad \text{para } n = 2, 3, 4, \ldots
\end{aligned}
$$

Supeito que $T(n) = \mathrm{O}(n \lg n)$.

Como achei as contantes?

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n + 5 \quad \text{para } n = 2, 3, 4, \ldots \end{aligned}$$

Supeito que $T(n) = \mathrm{O}(n \lg n)$.

Vamos tentar mostrar isto!

Precisamos encontrar um número real positivo c e um número inteiro positivo n_0 tais que

$$T(n) \leq c\, n \lg n$$

para todo $n \geq n_0$.

Rascunho

Suponha que existam as tais constantes c e n_0.
Vamos descobrir a "cara" delas.

$$
\begin{aligned}
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n + 5 \\
&\overset{\text{hi}}{\leq} c \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + c \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + 6n + 5 \\
&\leq c \left\lceil \frac{n}{2} \right\rceil \lg n + c \left\lfloor \frac{n}{2} \right\rfloor \lg n + 6n + 5 \\
&= c \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \right) \lg n + 6n + 5 \\
&= c\, n \lg n + 6n + 5 \ldots \quad \text{Hmmm, não deu\ldots}
\end{aligned}
$$

Nova tentativa

Suponha que existam c e n_0.

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n + 5$$

$$\overset{\text{hi}}{\leq} c\left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + c\left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$

$$\leq c\left\lceil \frac{n}{2} \right\rceil \lg n + c\left\lfloor \frac{n}{2} \right\rfloor \lg \frac{n}{2} + 6n + 5$$

$$= c\left\lceil \frac{n}{2} \right\rceil \lg n + c\left\lfloor \frac{n}{2} \right\rfloor (\lg n - 1) + 6n + 5$$

$$= c\left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor\right) \lg n - c\left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$

$$= c\,n \lg n - c\left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$ Agora vai!

Agora vai

Queremos saber "quando" $c\left\lfloor\frac{n}{2}\right\rfloor \geq 6n+5$. Como $c\left\lfloor\frac{n}{2}\right\rfloor \geq c\,\frac{n-1}{2}$, basta que $c\,\frac{n-1}{2} \geq 6n+5$, se e só se

$$c \geq \frac{12n+10}{n-1} = 12 + \frac{22}{n-1},$$

O que é verdade para $c = 20$ e todo $n \geq n_0 = 4$.

Conclusão

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

MERGE-SORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p+r)/2 \rfloor$
3 MERGE-SORT (A, p, q)
4 MERGE-SORT $(A, q+1, r)$
5 INTERCALA (A, p, q, r)

Conclusão:

O MERGE-SORT consome $\mathrm{O}(n \lg n)$ unidades de tempo.

Exemplo 4 (continuação)

Vou mostrar que $T(n) \geq n \lg n$ para $n = 1, 2, 3, 4, \ldots$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259
$n\lceil\lg n\rceil$	0	2	6	8	15	18	21	24	36	40

Exemplo 4 (continuação)

Vou mostrar que $T(n) \geq n \lg n$ para $n = 1, 2, 3, 4, \ldots$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259
$n\lceil \lg n \rceil$	0	2	6	8	15	18	21	24	36	40

Prova:

Se $n = 1$, então $T(1) = 1 > 1 \cdot \lg 1 = 0$.

Exemplo 4

Prova: (continuação) Se $n \geq 2$, então

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 6n + 5$$

$$\stackrel{\text{hi}}{\geq} \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$

$$\geq \left\lceil \frac{n}{2} \right\rceil \lg \frac{n}{2} + \left\lfloor \frac{n}{2} \right\rfloor \lg \frac{n-1}{2} + 6n + 5$$

$$= \left\lceil \frac{n}{2} \right\rceil (\lg n - 1) + \left\lfloor \frac{n}{2} \right\rfloor (\lg(n-1) - 1) + 6n + 5$$

$$\geq \left\lceil \frac{n}{2} \right\rceil (\lg n - 1) + \left\lfloor \frac{n}{2} \right\rfloor (\lg n - 2) + 6n + 5$$

$$= (\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor) \lg n - \left\lceil \frac{n}{2} \right\rceil - 2 \left\lfloor \frac{n}{2} \right\rfloor + 6n + 5$$

$$\geq n \lg n - 3 \left\lceil \frac{n}{2} \right\rceil + 6n + 5 \geq n \lg n.$$

Iiiiééééssss!

Exemplo 4

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 6n + 5 \quad \text{para } n = 2, 3, 4, 5, \ldots
\end{aligned}
$$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259

Conclusão:

$T(n)$ é $\Theta(n \lg n)$.

Conclusão da conclusão

O consumo de tempo do MERGE-SORT é $\Theta(n \lg n)$ no pior caso.

Exercício. Mostre que:

O consumo de tempo do MERGE-SORT é $\Theta(n \lg n)$.

Hmmmm... Qual a diferença entra as duas afirmações?

Recorrências com notação O

CLRS 4.1–4.2

AU 3.9, 3.11

Classe O da solução de uma recorrência

Não faço questão de solução exata: basta solução assintótica (em notação O, se possível Θ)

Classe O da solução de uma recorrência

Não faço questão de solução exata: basta solução assintótica (em notação O, se possível Θ)

Exemplo:

$$\begin{aligned} G(1) &= 1 \\ G(n) &= 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16, \ldots \end{aligned}$$

Solução exata: $G(n) = 7n \lg n + 3n - 2$

Solução assintótica: $G(n) = \mathrm{O}(n \lg n)$ $\quad$ ($G(n) = \Theta(n \lg n)$)

Classe O da solução de uma recorrência

Não faço questão de solução exata: basta solução assintótica (em notação O, se possível Θ)

Exemplo:

$$\begin{aligned} G(1) &= 1 \\ G(n) &= 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16, \ldots \end{aligned}$$

Solução exata: $G(n) = 7n \lg n + 3n - 2$

Solução assintótica: $G(n) = \mathrm{O}(n \lg n)$ $\quad (G(n) = \Theta(n \lg n))$

Em geral, é mais fácil obter e provar solução assintótica que solução exata

Dica prática (sem prova)

A solução da recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(\lfloor n/2 \rfloor) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

está na mesma classe Θ que a solução de

$$\begin{aligned} T'(1) &= 1 \\ T'(n) &= 2T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \ldots \end{aligned}$$

Dica prática (sem prova)

A solução da recorrência

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= 2T(\lfloor n/2 \rfloor) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \ldots
\end{aligned}
$$

está na mesma classe Θ que a solução de

$$
\begin{aligned}
T'(1) &= 1 \\
T'(n) &= 2T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \ldots
\end{aligned}
$$

e na mesma classe Θ que a solução de

$$
\begin{aligned}
T''(4) &= 10 \\
T''(n) &= 2T''(n/2) + n \quad \text{para } n = 2^3, 2^4, 2^5, \ldots
\end{aligned}
$$

Recorrências com O do lado direito

A "recorrência"

$$T(n) = 2T(n/2) + \mathrm{O}(n)$$

representa todas as recorrências da forma $T(n) = 2T(n/2) + f(n)$ em que $f(n)$ é $\mathrm{O}(n)$.

Recorrências com O do lado direito

A "recorrência"

$$T(n) = 2T(n/2) + \mathrm{O}(n)$$

representa todas as recorrências da forma
$T(n) = 2T(n/2) + f(n)$ em que $f(n)$ é $\mathrm{O}(n)$.

Melhor: representa todas as recorrências do tipo

$$\begin{aligned} T'(n) &\leq a \quad \text{para } n = k, k+1, \ldots, 2k-1 \\ T'(n) &\leq 2T'(\lfloor n/2 \rfloor) + b\,n \quad \text{para } n \geq 2k \end{aligned}$$

quaisquer que sejam a, $b > 0$ e $k > 0$
(poderíamos tomar $n_0 = 1$; veja ex ??.I)

Recorrências com O do lado direito

A "recorrência"

$$T(n) = 2T(n/2) + \mathrm{O}(n)$$

também representa todas as recorrências do tipo

$$\begin{aligned} T''(n) &= a \quad \text{para } n = 2^{k-1} \\ T''(n) &\leq 2T''(n/2) + b\,n \quad \text{para } n = 2^k, 2^{k+1}, \ldots \end{aligned}$$

quaisquer que sejam a, $b > 0$ e $k > 0$

Recorrências com O do lado direito

A “recorrência”

$$T(n) = 2T(n/2) + \mathrm{O}(n)$$

também representa todas as recorrências do tipo

$$\begin{aligned} T''(n) &= a \quad \text{para } n = 2^{k-1} \\ T''(n) &\leq 2T''(n/2) + b\,n \quad \text{para } n = 2^{k}, 2^{k+1}, \ldots \end{aligned}$$

quaisquer que sejam a, $b > 0$ e $k > 0$

As soluções exatas vão depender de a, b, k;
mas todas estarão na mesma classe O (no caso, em $\mathrm{O}(n \lg n)$)

Exemplo com O

Recorrência com O do lado direito

Na análise do consumo de tempo do MERGE-SORT tinhamos:

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

$$T(1) = \mathrm{O}(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \mathrm{O}(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Solução assintótica: $T(n)$ é $\mathrm{O}(n \lg n)$.

Exemplo com Θ

Recorrências com Θ do lado direito

Na análise do consumo de tempo do MERGE-SORT poderíamos ter feito:

$T(n) :=$ consumo de tempo quando $n = r - p + 1$

$$T(1) = \Theta(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Solução assintótica: $T(n)$ é $\Theta(n \lg n)$.

Dica prática

Suponha dada uma "recorrência" como

$$T(n) = 2T(n/2) + \mathrm{O}(n).$$

Para ter uma idéia da classe O a que T pertence, resolva a recorrência

$$\begin{aligned} T''(2^0) &= 1 \\ T''(n) &\leq 2T''(n/2) + n \quad \text{para } n = 2^1, 2^2, 2^3, \ldots \end{aligned}$$

Mais dicas práticas

recorrência	condição	solução
$T(n) = T(n-1) + 4n^3$		$\Theta(n^{3+1})$
$T(n) = 6T(n-1) + 4n^3$		$\Theta(6^n)$
$T(n) = aT(n/5) + 4n^3$	$a < 5^3$	$\Theta(n^3)$
$T(n) = aT(n/5) + 4n^3$	$a = 5^3$	$\Theta(n^3 \log n)$
$T(n) = aT(n/5) + 4n^3$	$a > 5^3$	$\Theta(n^{\log_5 a})$

Veja AU, sec 3.11, p.151

Mais dicas práticas (continuação)

No lugar de

- "$n/5$", posso escrever "$\lfloor n/5 \rfloor$" ou "$\lceil n/5 \rceil$"
- "5", posso escrever qualquer $b > 1$
- "4", posso escrever qualquer número real
- "$4n^3$" posso escrever qualquer polinômio de grau 3
- "3", posso escrever qualquer inteiro $k \geq 0$
- "6", posso escrever qualquer número $a > 1$

Mais dicas práticas ainda

A mesma coisa, escrita de maneira um pouco diferente (Master Theorem, CLRS, sec. 4.3, p.73):

Suponha $T(n) = aT(n/5) + f(n)$ para algum $a \geq 1$. Então, em geral,

$$
\begin{array}{lll}
\text{se } f(n) = \mathrm{O}(n^{\log_5 a - \epsilon}) & \text{então} & T(n) = \Theta(n^{\log_5 a}) \\
\text{se } f(n) = \Theta(n^{\log_5 a}) & \text{então} & T(n) = \Theta(n^{\log_5 a} \lg n) \\
\text{se } f(n) = \Omega(n^{\log_5 a + \epsilon}) & \text{então} & T(n) = \Theta(f(n))
\end{array}
$$

para qualquer $\epsilon > 0$

Teorema Bambambã

Teorema Mestre (Master Theorem, CLRS, sec. 4.3, p.73):

Suponha

$$T(n) = a\,T(n/b) + f(n)$$

para algum $a \geq 1$ e $b > 1$ e onde n/b significa $\lceil n/b \rceil$ ou $\lfloor n/b \rfloor$. Então, em geral,

se $f(n) = \mathrm{O}(n^{\log_b a - \epsilon})$ então $T(n) = \Theta(n^{\log_b a})$
se $f(n) = \Theta(n^{\log_b a})$ então $T(n) = \Theta(n^{\log_b a} \lg n)$
se $f(n) = \Omega(n^{\log_b a + \epsilon})$ então $T(n) = \Theta(f(n))$

para qualquer $\epsilon > 0$.

Exercícios

Exercício 9.A
A que classe O pertencem as solução de recorrência do tipo $T(n) = T(n/3) + \mathrm{O}(1)$?

Exercício 9.B
Seja T a função definida pela recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(\lfloor n/2 \rfloor) + n \qquad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

A que ordem Θ pertence T?

Exercício 9.C [CLRS 4.2-1]
Seja T a função definida pela recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T(\lfloor n/2 \rfloor) + n \qquad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

A partir da árvore da recorrência, adivinhe a que classe Θ pertence $T(n)$. Prove a delimitação pelo método da substituição.

Mais exercícios

Exercício 9.D

Resolva a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(\lceil n/2 \rceil) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

Exercício 9.E

Resolva a “recorrência” $T(n) = T(n-2) + \mathrm{O}(n)$.

Exercício 9.F

Resolva a “recorrência” $T(n) = 5T(n-1) + \mathrm{O}(n)$.

Melhores momentos

AULA 5

Recorrências

$$
\begin{aligned}
T(1) &= 1 \\
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 6n + 5 \quad \text{para } n = 2, 3, 4, 5, \ldots
\end{aligned}
$$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	19	43	67	97	127	157	187	223	259

Vimos que:

$T(n)$ é $\Theta(n \lg n)$.

Conclusão

O consumo de tempo do MERGE-SORT é $\Theta(n \lg n)$ no pior caso.

Exercício. Mostre que:

O consumo de tempo do MERGE-SORT é $\Theta(n \lg n)$.

Hmmmm... Qual a diferença entra as duas afirmações?

Classe O da solução de uma recorrência

Não faço questão de solução exata: basta solução aproximada (em notação O; melhor ainda Θ)

Exemplo:

$$\begin{aligned} G(1) &= 1 \\ G(n) &= 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16, \ldots \end{aligned}$$

Solução exata: $G(n) = 7n \lg n + 3n - 2$

Solução aproximada: $G(n) = \mathrm{O}(n \lg n)$ $\quad$ ($G(n) = \Theta(n \lg n)$!)

Em geral, é mais fácil obter e provar solução aproximada que solução exata

Dica prática (sem prova)

A solução da recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(\lfloor n/2 \rfloor) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

está na mesma classe Θ que a solução de

$$\begin{aligned} T'(1) &= 1 \\ T'(n) &= 2T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \ldots \end{aligned}$$

e na mesma classe Θ que a solução de

$$\begin{aligned} T''(4) &= 10 \\ T''(n) &= 2T''(n/2) + n \quad \text{para } n = 2^3, 2^4, 2^5, \ldots \end{aligned}$$

Recorrências com O do lado direito

A "recorrência"

$$T(n) = 2T(n/2) + \mathrm{O}(n)$$

representa todas as recorrências da forma
$T(n) = 2T(n/2) + f(n)$ em que $f(n)$ é $\mathrm{O}(n)$.

Melhor: representa todas as recorrências do tipo

$$\begin{aligned} T'(n) &\leq a \quad \text{para } n = k, k+1, \ldots, 2k-1 \\ T'(n) &\leq 2T'(\lfloor n/2 \rfloor) + b\,n \quad \text{para } n \geq 2k \end{aligned}$$

quaisquer que sejam a, $b > 0$ e $k > 0$
(poderíamos tomar $n_0 = 1$; veja ex ??.I)

Exemplo com O

Recorrência com O do lado direito

Na análise do consumo de tempo do MERGE-SORT tinhamos:

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

$$T(1) = \mathrm{O}(1)$$
$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \mathrm{O}(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Solução aproximada: $T(n)$ é $\mathrm{O}(n \lg n)$.

Exemplo com Θ

Recorrências com Θ do lado direito

Na análise do consumo de tempo do MERGE-SORT tinhamos:

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

$$T(1) = \Theta(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Solução aproximada: $T(n)$ é $\Theta(n \lg n)$.

Teorema Ban-Ban-Ban

Teorema Mestre (Master Theorem, CLRS, sec. 4.3, p.73):

Suponha

$$T(n) = a\,T(n/b) + f(n)$$

para algum $a \geq 1$ e $b > 1$ e onde n/b significa $\lceil n/b \rceil$ ou $\lfloor n/b \rfloor$.
Então, em geral,

se $f(n) = \mathrm{O}(n^{\log_b a - \epsilon})$ então $T(n) = \Theta(n^{\log_b a})$
se $f(n) = \Theta(n^{\log_b a})$ então $T(n) = \Theta(n^{\log_b a} \lg n)$
se $f(n) = \Omega(n^{\log_b a + \epsilon})$ então $T(n) = \Theta(f(n))$

para qualquer $\epsilon > 0$.

AULA 6

Técnicas em projeto de algoritmos

Programming Pearls: Algorithm Design Techniques,
Jon Bentley, Addison-Wesley, 1986

Segmento de soma máxima

Um segmento de um vetor $A[1\,..\,n]$ é qualquer subvetor da forma $A[e\,..\,d]$.

Problema: Dado um vetor $A[1\,..\,n]$ de números inteiros, determinar um segmento $A[e\,..\,d]$ de soma máxima.

Entra:

	1									n
A	31	-41	59	26	-53	58	97	-93	-23	84

Segmento de soma máxima

Um segmento de um vetor $A[1\,..\,n]$ é qualquer subvetor da forma $A[e\,..\,d]$.

Problema: Dado um vetor $A[1\,..\,n]$ de números inteiros, determinar um segmento $A[e\,..\,d]$ de soma máxima.

Entra:

	1									n
A	31	−41	59	26	−53	58	97	−93	−23	84

Sai:

	1		3				7			n
A	31	−41	59	26	−53	58	97	−93	−23	84

$A[e\,..\,d] = A[3\,..\,7]$ é segmento de soma máxima.

$A[3\,..\,7]$ tem soma 187.

Segmento de soma máxima

Problema (versão simplificada): Determinar a soma máxima de um segmento de um dado vetor $A[1\,..\,n]$.

Entra:

	1									n
A	31	−41	59	26	−53	58	97	−93	−23	84

Sai:

	1		3				7			n
A	31	−41	59	26	−53	58	97	−93	−23	84

A soma máxima é 187.

Algoritmo café-com-leite

Algoritmo determina um segmento de soma máxima de $A[1\,.\,.\,n]$.

SEG-MAX-3 (A, n)

```
 1  somamax ← 0
 2  e ← 0    d ← −1    ▷ A[e . . d] é vazio
 3  para i ← 1 até n faça
 4        para f ← i até n faça
 5              soma ← 0
 6              para k ← i até f faça
 7                    soma ← soma + A[k]
 8              se soma > somamax então
 9                    somamax ← soma    e ← i    d ← f
10  devolva e, d e somamax
```

Correção

Relação invariante chave:

(i0) na linha 3 vale que: $A[e\,.\,.\,d]$ é um segmento de soma máxima com $e < i$.

A	31	−41	59	26	−53	58	97	−93	−23	84
	e		i				d			n

Correção

Relação invariante chave:

(i0) na linha 3 vale que: $A[e\,.\,.\,d]$ é um segmento de soma máxima com $e < i$.

	e		i				d			n
A	31	−41	59	26	−53	58	97	−93	−23	84

Mais relações invariantes:

(i1) na linha 3 vale que:

$$somamax = A[e] + A[e+1] + A[e+2] + \cdots + A[d];$$

(i2) na linha 6 vale que:

$$soma = A[i] + A[i+1] + A[i+2] + \cdots + A[k-1].$$

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha		todas as execuções da linha	
1-2	$=$	2	$= \Theta(1)$
3	$=$	$n+1$	$= \Theta(n)$
4	$=$	$(n+1)+n+(n-1)+\cdots+1$	$= \Theta(n^2)$
5	$=$	$n+(n-1)+\cdots+1$	$= \Theta(n^2)$
6	$=$	$(2+\cdots+(n+1))+(2+\cdots+n)+\cdots+2$	$= \Theta(n^3)$
7	$=$	$(1+\cdots+n)+(1+\cdots+(n-1))+\cdots+1$	$= \Theta(n^3)$
8	$=$	$n+(n-1)+(n-2)+\cdots+1$	$= \Theta(n^2)$
9	$\leq$	$n+(n-1)+(n-2)+\cdots+1$	$= \mathrm{O}(n^2)$
10	$=$	1	$= \Theta(1)$
total	$=$	$\Theta(2n^3+3n^2+n+2)+\mathrm{O}(n^2)$	$= \Theta(n^3)$

Algoritmo arroz-com-feijão

Algoritmo determina um segmento de soma máxima de $A[1\,..\,n]$.

SEG-MAX-2 (A, n)

1 $\quad somamax \leftarrow 0$
2 $\quad e \leftarrow 0 \quad d \leftarrow -1 \quad \triangleright\ A[e\,..\,d]$ é vazio
3 $\quad$ **para** $i \leftarrow 1$ **até** n **faça**
4 $\quad\quad soma \leftarrow 0$
5 $\quad\quad$ **para** $f \leftarrow i$ **até** n **faça**
6 $\quad\quad\quad soma \leftarrow soma + A[f]$
7 $\quad\quad\quad$ **se** $soma > somamax$ **então**
8 $\quad\quad\quad\quad somamax \leftarrow soma \quad e \leftarrow i \quad d \leftarrow f$
9 $\quad$ **devolva** e, d **e** $somamax$

Correção

Relação invariante chave:

(i0) na linha 3 vale que: $A[e\,..\,d]$ é um segmento de soma máxima com $e < i$.

	e						d	i		n
A	31	−41	59	26	−53	58	97	−93	−23	84

Mais relações invariante:

(i1) na linha 3 vale que:

$$somamax = A[e] + A[e+1] + A[e+2] + \cdots + A[d];$$

(i2) na linha 5 vale que:

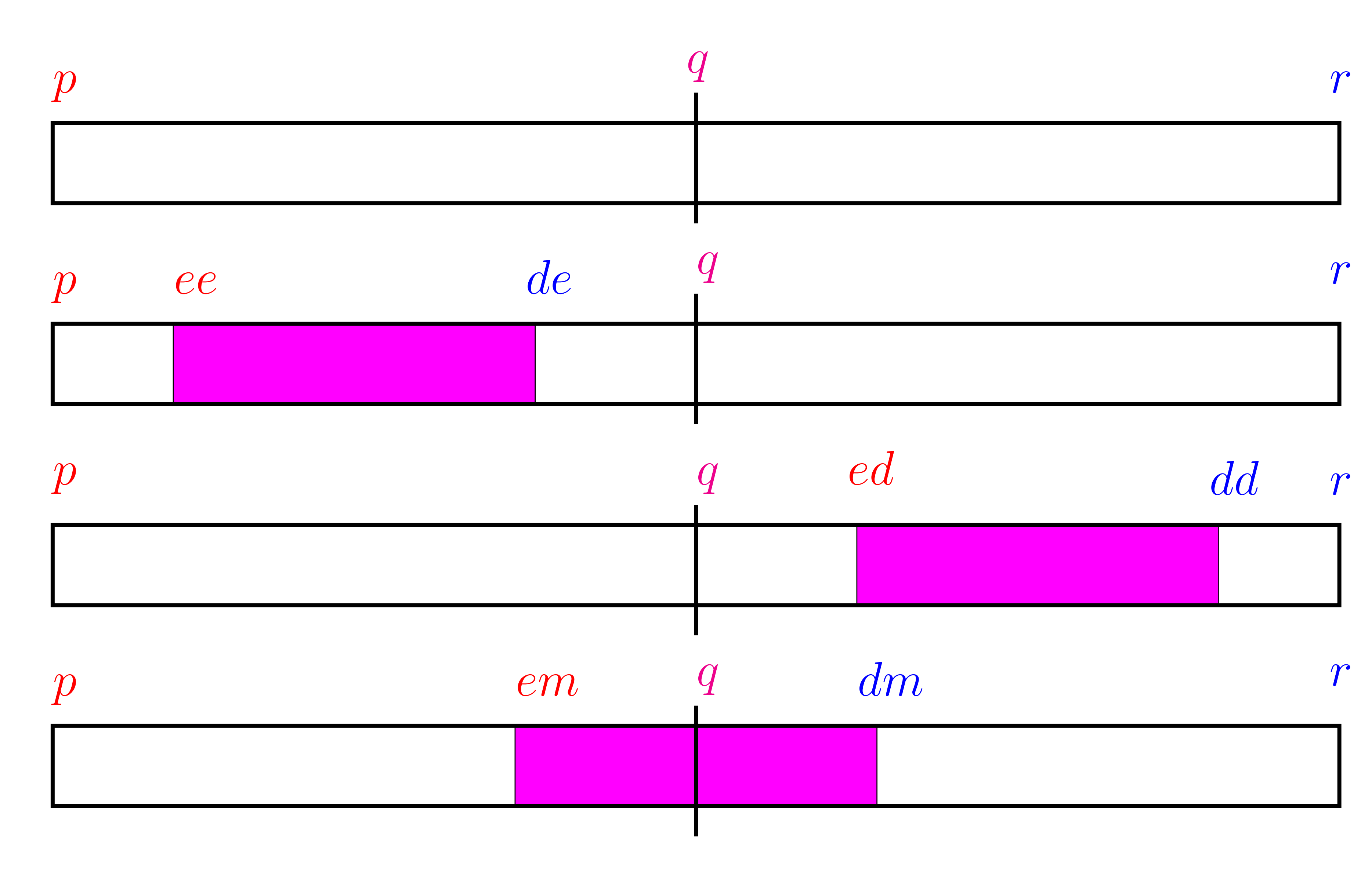
$$soma = A[i] + A[i+1] + A[i+2] + \cdots + A[f-1];$$

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha		todas as execuções da linha	
1-2	$=$	2	$= \Theta(1)$
3	$=$	$n+1$	$= \Theta(n)$
4	$=$	n	$= \Theta(n)$
5	$=$	$(n+1) + n + \cdots + 2$	$= \Theta(n^2)$
6	$=$	$n + (n-1) + \cdots + 1$	$= \Theta(n^2)$
7	$=$	$n + (n-1) + \cdots + 1$	$= \Theta(n^2)$
8	$\leq$	$n + (n-1) + \cdots + 1$	$= \mathrm{O}(n^2)$
9	$=$	1	$= \Theta(1)$
total	$=$	$\Theta(3n^2 + 2n + 2) + \mathrm{O}(n^2)$	$= \Theta(n^2)$

Solução de divisão-e-conquista



Algoritmo de divisão-e-conquista

Algoritmo determina soma máxima de um seg. de $A[p\,.\,.\,r]$.

SEG-MAX-DC (A, p, r)

1 se $p = r$ **então devolva** $\max(0, A[p])$
2 $q \leftarrow \lfloor (p+r)/2 \rfloor$
3 $maxesq \leftarrow$ SEG-MAX-DC(A, p, q)
4 $maxdir \leftarrow$ SEG-MAX-DC$(A, q+1, r)$
5 $max2esq \leftarrow soma \leftarrow A[q]$
6 **para** $i \leftarrow q-1$ **decrescendo até** p **faça**
7 $\quad soma \leftarrow soma + A[i]$
8 $\quad max2esq \leftarrow \max(max2esq, soma)$
9 $max2dir \leftarrow soma \leftarrow A[q+1]$
10 **para** $f \leftarrow q+2$ **até** r **faça**
11 $\quad soma \leftarrow soma + A[f]$
12 $\quad max2dir \leftarrow \max(max2dir, soma)$
13 $maxcruz \leftarrow max2esq + max2dir$
14 **devolva** $\max(maxesq, maxcruz, maxdir)$

Correção

Verifique que:

- $maxesq$ é a soma máxima de um segmento de $A[p\,..\,q]$;
- $maxdir$ é a soma máxima de um segmento de $A[q+1\,..\,r]$; e
- $maxcruz$ é a soma máximo de um segmento da forma $A[i\,..\,f]$ com $i \leq q$ e $q+1 \leq f$.

Conclua que o algoritmo devolve a soma máxima de um segmento de $A[p\,..\,r]$.

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha		todas as execuções da linha	
1-2	$=$	2	$= \Theta(1)$
3	$=$	$T(\lceil \frac{n}{2} \rceil)$	$= T(\lceil \frac{n}{2} \rceil)$
4	$=$	$T(\lfloor \frac{n}{2} \rfloor)$	$= T(\lfloor \frac{n}{2} \rfloor)$
5	$=$	1	$= \Theta(1)$
6	$=$	$\lceil \frac{n}{2} \rceil + 1$	$= \Theta(n)$
7-8	$=$	$\lceil \frac{n}{2} \rceil$	$= \Theta(n)$
9	$=$	1	$= \Theta(1)$
10	$=$	$\lfloor \frac{n}{2} \rfloor + 1$	$= \Theta(n)$
11-12	$=$	$\lfloor \frac{n}{2} \rfloor$	$= \Theta(n)$
13-14	$=$	2	$= \Theta(1)$

total $= T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(4n + 4)$

Consumo de tempo

$T(n) :=$ consumo de tempo quando $n = r - p + 1$

Na análise do consumo de tempo do SEG-MAX-DC chegamos a (já manjada) recorrências com Θ do lado direito:

$$T(1) = \Theta(1)$$
$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Solução assintótica: $T(n)$ é $\Theta(n \lg n)$.

Algoritmo ingênuo

SEG-MAX-INGÊNUO (A, n)

```
1   somamax ← 0   e ← 0   d ← −1   ▷ A[e . . d] é vazio
2   soma ← 0   i ← 1   f ← 0   ▷ A[i . . f] é vazio
3   enquanto f < n faça
4       f ← f + 1
5       soma ← 0   i ← 1   f ← 0   ▷ A[i . . f] é vazio
6       somaf ← 0
7       para k ← f decrescendo até 1 faça
8           somaf ← somaf + A[k]
9           se somaf > soma
10              então soma ← somaf   i ← k
11      se soma > somamax
12          então somamax ← soma   e ← i   d ← f
13  devolva e, d e somamax
```

Correção

A correção do algoritmo se apóia nas relações invariantes a seguir.

Relação invariante chave:

(i0) na linha 3 vale que: $A[e\,..\,d]$ é segmento de soma máxima com $d \leq f$.

	e	d	f					n
A									

A	31	−41	59	26	−53	58	97	−93	−23	84
			e	d	f					n

Mais uma relação invariante:

(i1) na linha 3 vale que:

$$somamax = A[e] + A[e+1] + A[e+2] + \cdots + A[d].$$

Mais relações invariantes

As relações invariantes a seguir passam meio desapercebidas.

Na linha 3 vale que:

(i2) $A[i \mathinner{.\,.} f]$ é segmento de soma máxima com termino em f;

(i3) $soma = A[i] + A[i+1] + A[i+2] + \cdots + A[f]$;

(i4) para $k = i, i+1, \ldots, f$, vale que

$$A[i] + A[i+1] + \cdots + A[k] \geq 0\,;$$

(i5) para $k = 1, 2, \ldots, i-1$, vale que

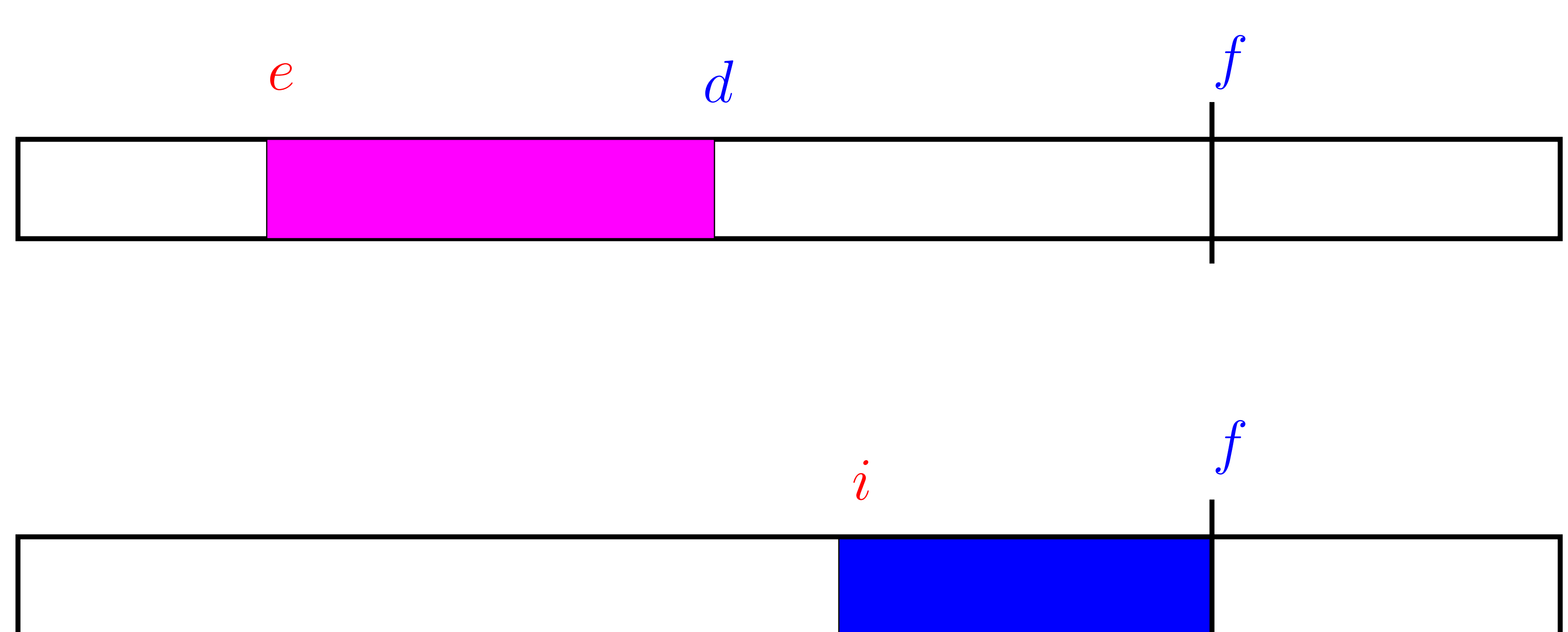
$$A[k] + A[k+1] + \cdots + A[i-1] \leq 0\,.$$

O algoritmo linear se aproveita de dessas relações!

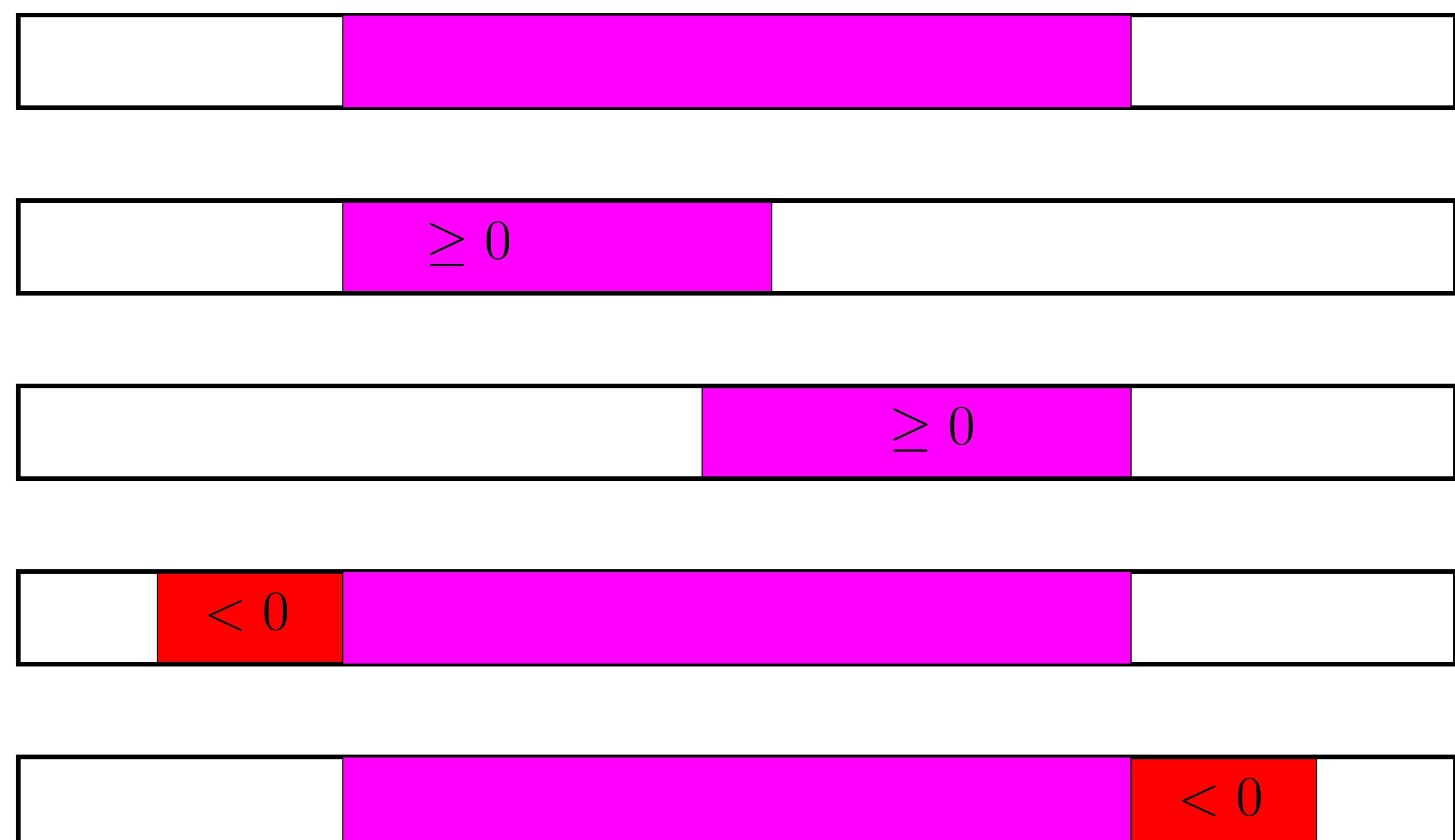
Consumo de tempo

linha		todas as execuções da linha	
1-2	$=$	2	$=\Theta(1)$
3-4	$=$	2	$=\Theta(1)$
5	$=$	$n+1$	$=\Theta(n)$
4-5	$=$	n	$=\Theta(n)$
6	$=$	n	$=\Theta(n)$
7	$=$	$2+3+\ldots+(n+1)$	$=\Theta(n^2)$
8-9	$=$	$1+2+\ldots+n$	$=\Theta(n^2)$
10	$\leq$	$1+2+\ldots+n$	$=\mathrm{O}(n^2)$
11	$=$	n	$=\Theta(n)$
12	$\leq$	n	$=\mathrm{O}(n)$
12	$=$	1	$=\Theta(1)$
total	$=$	$\Theta(2n^2+4n+3)+\mathrm{O}(n^2+n)$	$=\Theta(n^2)$

Invariantes (i0) e (i2)



Cara da solução



Conclusões

Se $A[e\,..\,d]$ é um segmento de soma máxima então:

- para $k = e, e+1, \ldots, d$, vale que

$$A[e] + A[e+1] + \cdots + A[k] \geq 0\,;$$

- para $k = e, e+1, \ldots, d$, vale que

$$A[k] + A[k+1] + \cdots + A[d] \geq 0\,;$$

- para $k = 1, 2, \ldots, e-1$, vale que

$$A[k] + A[k+1] + \cdots + A[e-1] \leq 0\,;$$

- para todo $k = d+1, d+2, \ldots, n$, vale que

$$A[d+1] + A[d+2] + \cdots + A[k] \leq 0\,.$$

Mais conclusões

Se $A[i\,..\,f]$ é um segmento de soma máxima terminando em f então:

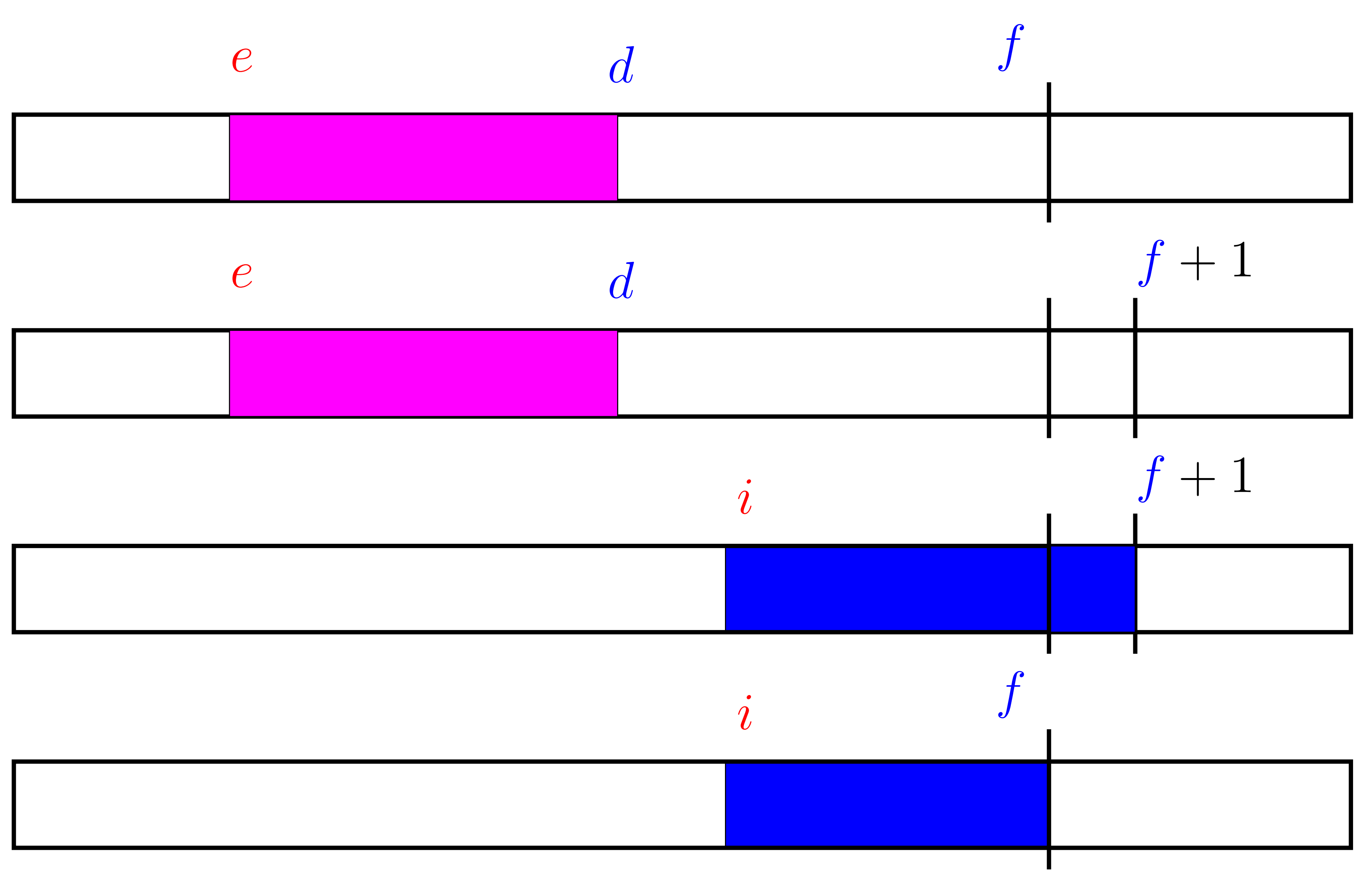
- para $k = i, i+1, \ldots, f$, vale que

$$A[i] + A[i+1] + \cdots + A[k] \geq 0\,;$$

- para $k = 1, 2, \ldots, i-1$, vale que

$$A[k] + A[k+1] + \cdots + A[i-1] \leq 0\,.$$

Solução indutiva



Algoritmo linear

Algoritmo determina um segmento de soma máxima de $A[1\,..\,n]$ (por Jay Kadane).

SEG-MAX-1 (A, n)

1 $somamax \leftarrow 0$
2 $e \leftarrow 0 \quad d \leftarrow -1 \quad \triangleright\ A[e\,..\,d]$ é vazio
3 $soma \leftarrow 0$
4 $i \leftarrow 1 \quad f \leftarrow 0 \quad \triangleright\ A[i\,..\,f]$ é vazio
5 **enquanto** $f < n$ **faça**
6 $\quad f \leftarrow f + 1$
7 $\quad soma \leftarrow soma + A[f]$
8 $\quad$ **se** $soma < 0$
9 $\quad\quad$ **então** $soma \leftarrow 0 \quad i \leftarrow f + 1$
10 $\quad$ **senão se** $soma > somamax$
11 $\quad\quad$ **então** $somamax \leftarrow soma \quad e \leftarrow i \quad d \leftarrow f$
12 **devolva** e, d **e** $somamax$

Correção

Relação invariante chave:

(i0) na linha 5 vale que: $A[e \,..\, d]$ é segmento de soma máxima com $d \leq f$.

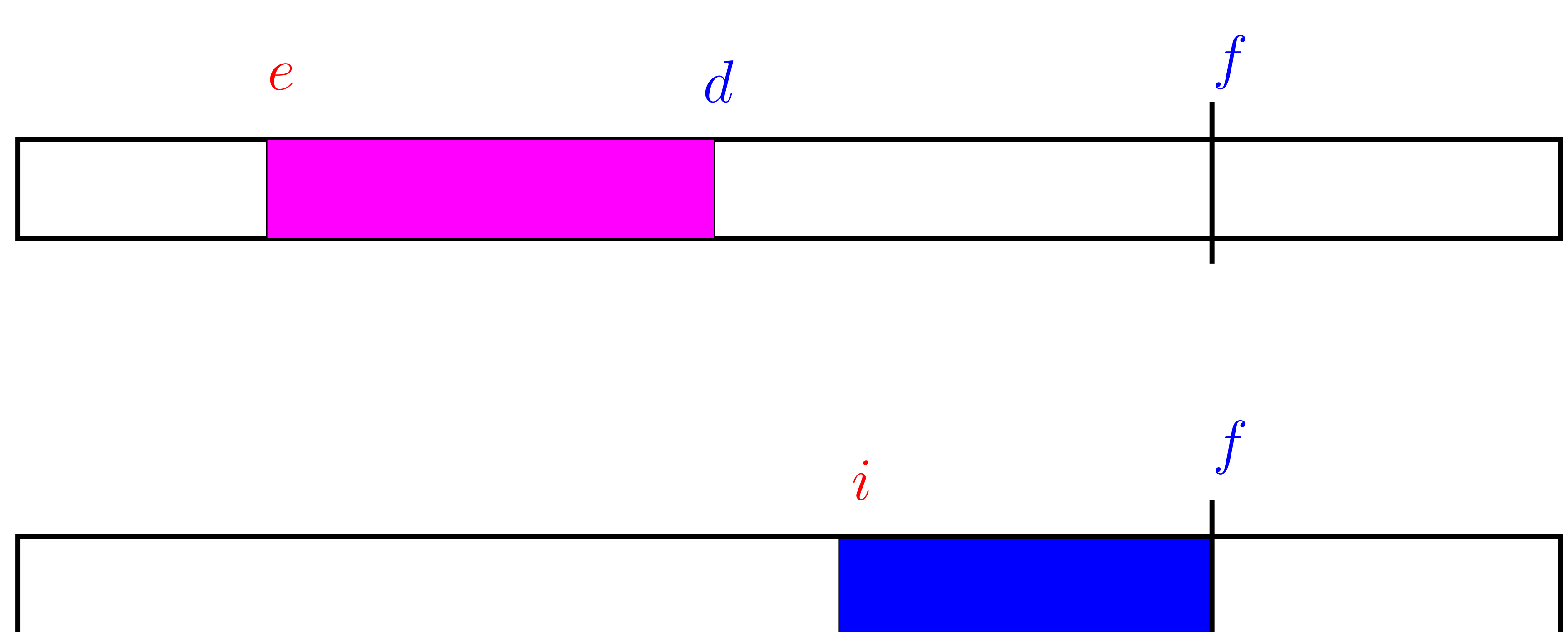
			e	d	f					n
A	31	−41	59	26	−53	58	97	−93	−23	84

Mais uma relação invariante:

(i1) na linha 5 vale que:

$$somamax = A[e] + A[e+1] + A[e+2] + \cdots + A[d].$$

Invariantes (i0) e (i2)



Mais relações invariantes

Na linha 5 vale que:

(i2) $A[i\,..\,f]$ é segmento de soma máxima com termino em f;

(i3) $soma = A[i] + A[i+1] + A[i+2] + \cdots + A[f]$;

(i4) para $k = i, i+1, \ldots, f$, vale que

$$A[i] + A[i+1] + \cdots + A[k] \geq 0\,;$$

Na linha 9 vale que:

(i5) $soma = A[i] + A[i+1] + \cdots + A[f] < 0$

As relações invariantes (i4) e (i5) implicam que na linha 9:

(i6) para $k = 1, 2, 3, \ldots, f$, vale que

$$A[k] + A[k+1] + \cdots + A[f] < 0\,;$$

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha		todas as execuções da linha	
1-2	$=$	2	$= \Theta(1)$
3-4	$=$	2	$= \Theta(1)$
5	$=$	$n+1$	$= \Theta(n)$
6-7	$=$	n	$= \Theta(n)$
8	$=$	n	$= \Theta(n)$
9-10	$=$	n	$= \Theta(n)$
11	$\leq$	n	$= \mathrm{O}(n)$
12	$=$	1	$= \Theta(1)$
total	$=$	$\Theta(4n+3) + \mathrm{O}(n)$	$= \Theta(n)$

Conclusões

O consumo de tempo do algoritmo SEG-MAX-3 é $\Theta(n^3)$.

O consumo de tempo do algoritmo SEG-MAX-2 é $\Theta(n^2)$.

O consumo de tempo do algoritmo SEG-MAX-DC é $\Theta(n \lg n)$.

O consumo de tempo do algoritmo SEG-MAX-1 é $\Theta(n)$.

Técnicas

- Evitar recomputações. Usar espaço para armazenar resultados a fim de evitar recomputá-los (SEG-MAX-2, SEG-MAX-1, programação dinâmica).
- Pré-processar os dados. HEAPSORT pré-processa os dados armazenando-os em uma estrutura de dados a fim de realizar computações futuras mais eficientemente.
- Divisão-e-conquista. MERGE-SORT e SEG-MAX-DC utilizam uma forma conhecidade dessa técnica.
- Algoritmos incrementais/varredura. Solução de um subproblema é estendida a uma solução do problema original (SEG-MAX-1).
- Delimitação inferior. Projetistas de algoritmos só dormem em paz quando sabem que seus algoritmos são o melhor possível (SEG-MAX-1).

Análise experimental de algoritmos

"O interesse em experimentação, é devido ao reconhecimento de que os resultados teóricos, freqüentemente, não trazem informações referentes ao desempenho do algoritmo na prática."

Análise experimental de algoritmos

Segundo D.S. Johnson, pode-se dizer que existem quatro motivos básicos que levam a realizar um trabalho de implementação de um algoritmo:

- usar o código em uma aplicação particular, cujo propósito é descrever o impacto do algoritmo em um certo contexto;
- proporcionar evidências da superioridade de um algoritmo;
- melhor compreensão dos pontos fortes e fracos e do desempenho das operações algorítmicas na prática; e
- produzir conjecturas sobre o comportamento do algoritmo no caso-médio sob distribuições específicas de instâncias onde a análise probabilística direta é muito difícil.

Ambiente experimental

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

Os códigos estão compilados com o `gcc` versão `??` e opção de compilação `??`.

As instâncias são obtidas utilizando-se o gerador `gera.c` disponível em

`...~coelho/mac0338-2007/tarefa-extra/.`

As implementações comparadas neste experimento são SEG-MAX-3, SEG-MAX-2 e SEG-MAX-1.

Ambiente experimental

A estimativa do tempo é calculada utilizando-se:

```
#include <time.h>
[...]
clock_t start, end;
double time;

start = clock();

[...implementação...]

end  = clock();
time = ((double)(end - start))/CLOCKS_PER_SEC;
```

O tempo de entrada/saída é computado separadamente.

Resultados experimentais

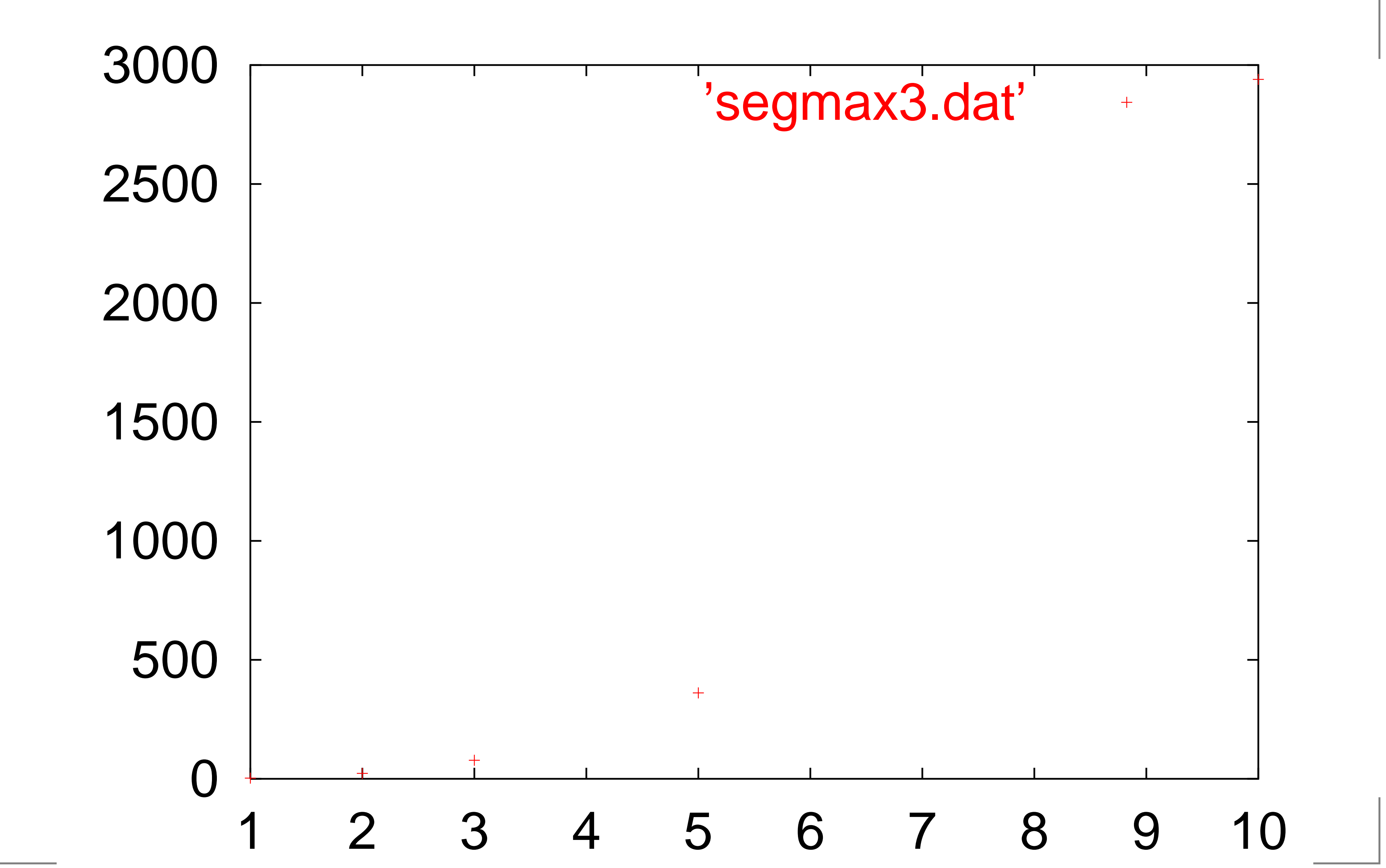
n	SEG-MAX-2		SEG-MAX-3	
	`clock`	`user time`	`clock`	`user time`
1000	0.01	0.01	2.91	2.92
2000	0.03	0.03	23.00	23.04
3000	0.08	0.08	77.58	1m17.57
5000	0.24	0.24	360.78	6m0.45s
10000	1.16	1.62	2940	49m27.32
20000	4.8	4.82	?	?
50000	42.8	42.96	?	?
100000	188.0	3m7.58s	?	?
200000	911.74	15m10.34s	?	?
400000	?	64m22.880s	?	?

Resultados experimentais

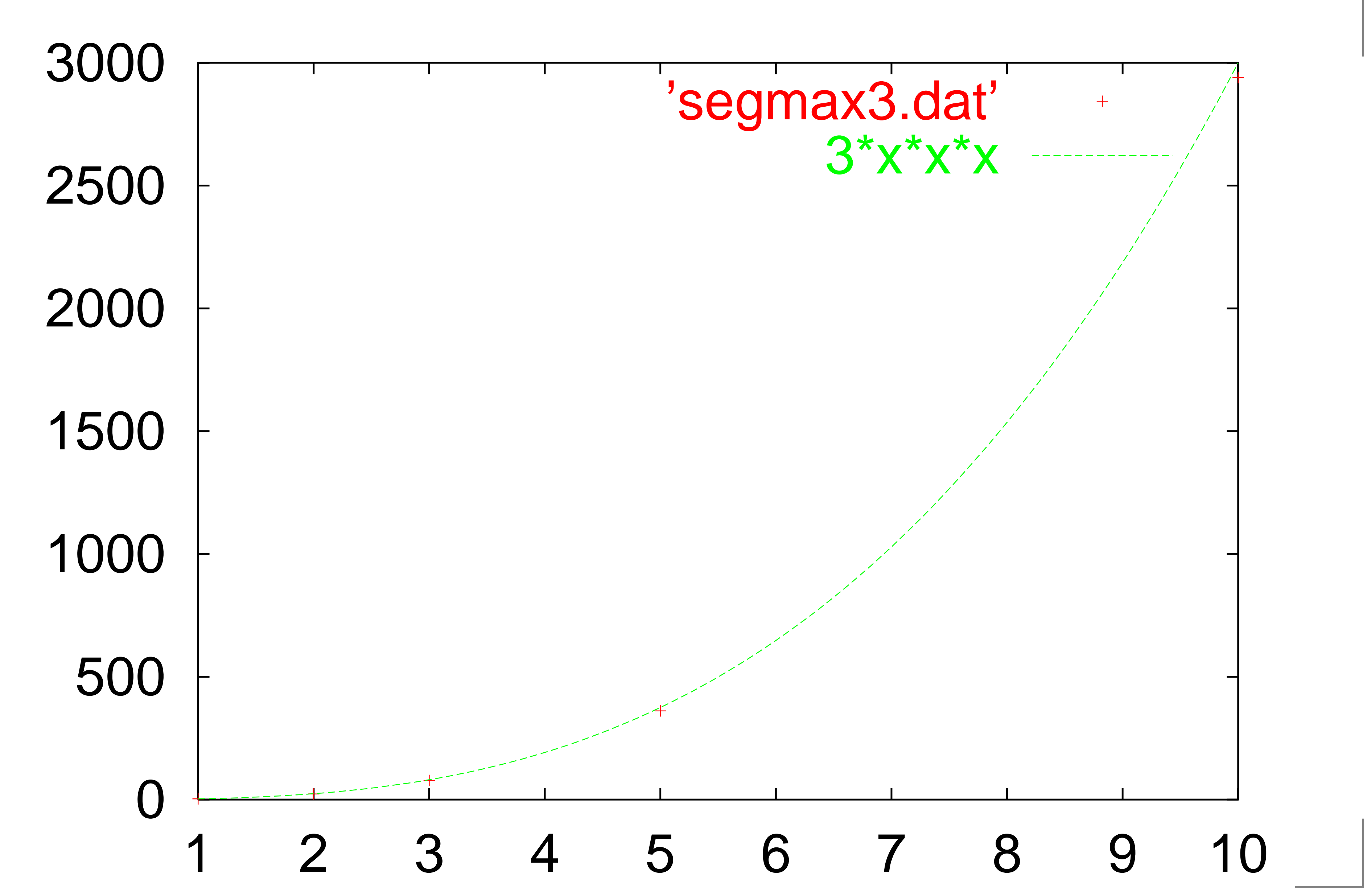
n (M)	SEG-MAX-1 A		SEG-MAX-1 B	
	`clock`	`user time`	`clock`	`user time`
1	0.06	0.11	0.08	0.07
2	0.11	0.23	0.15	0.06
3	0.16	0.28	0.30	0.12
6	0.32	0.63	0.56	0.29
8	0.43	0.80	0.71	0.43
10	0.55	1.03	1.00	0.52
15	0.82	1.55	1.37	0.64
20	1.08	2.03	2.17	0.95
25	1.37	2.52	2.44	1.38
30	1.70	3.13	3.28	1.68

Todos os tempos estão em segundos.

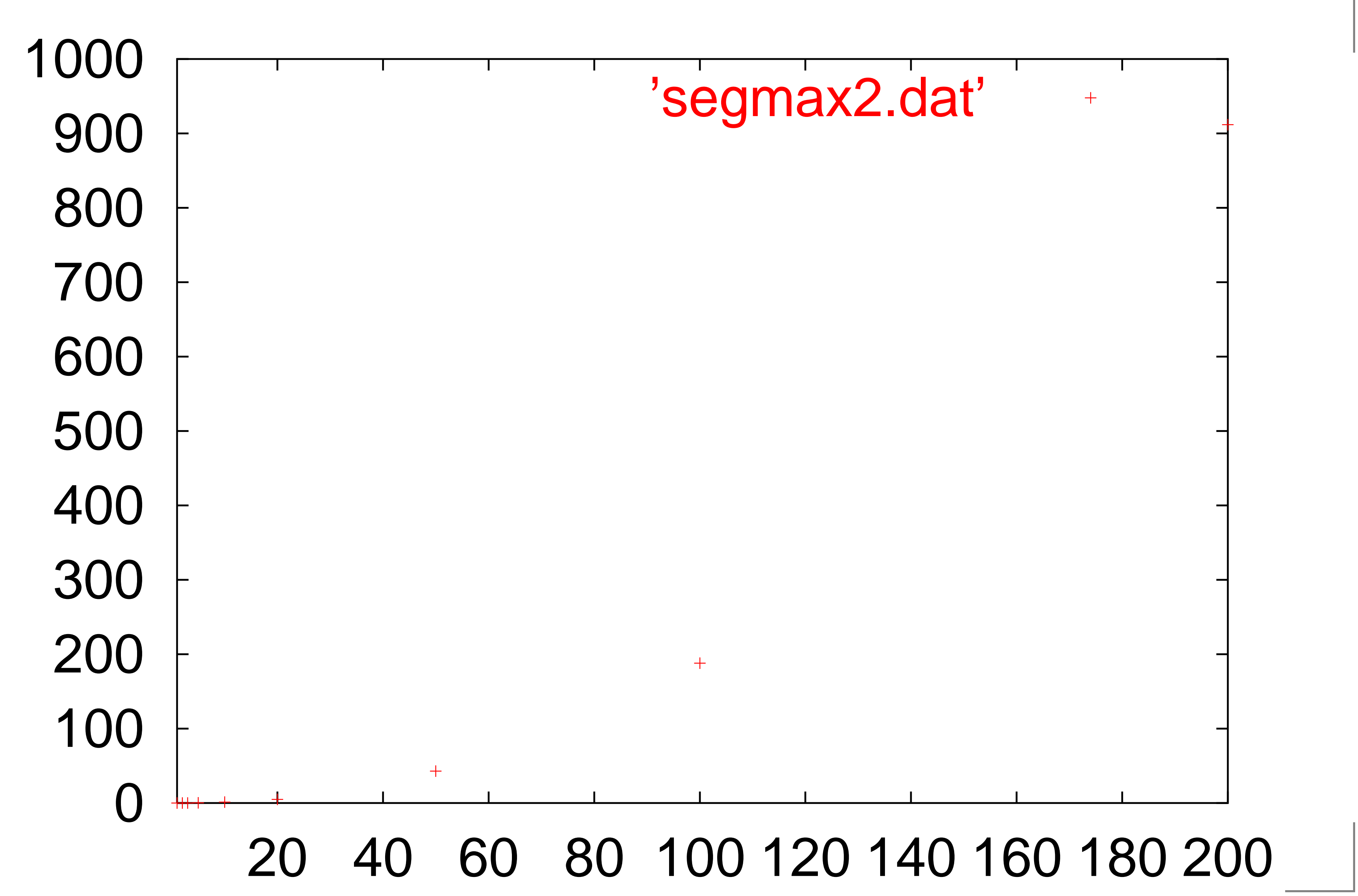
SEG-MAX-3



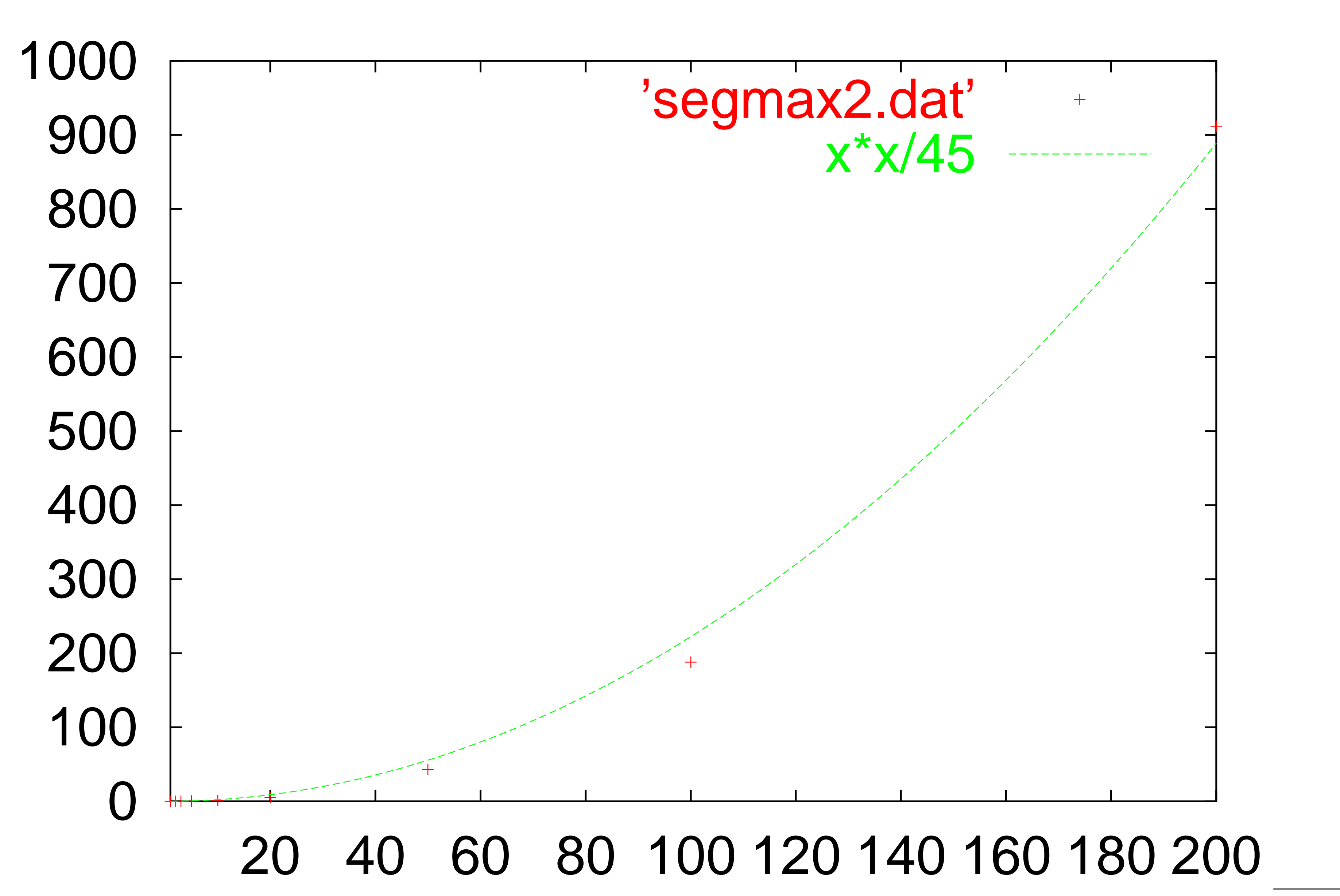
SEG-MAX-3



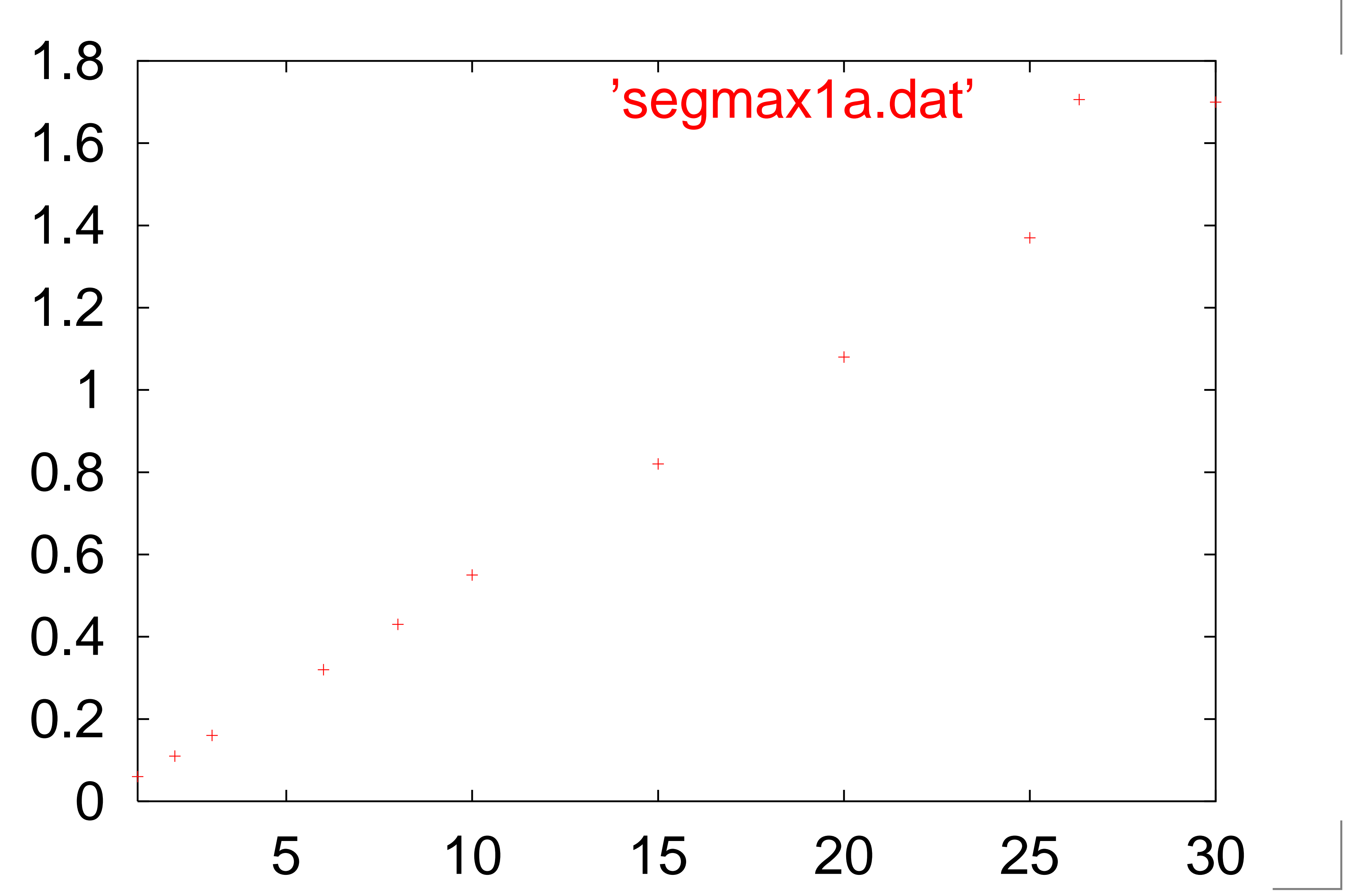
SEG-MAX-2



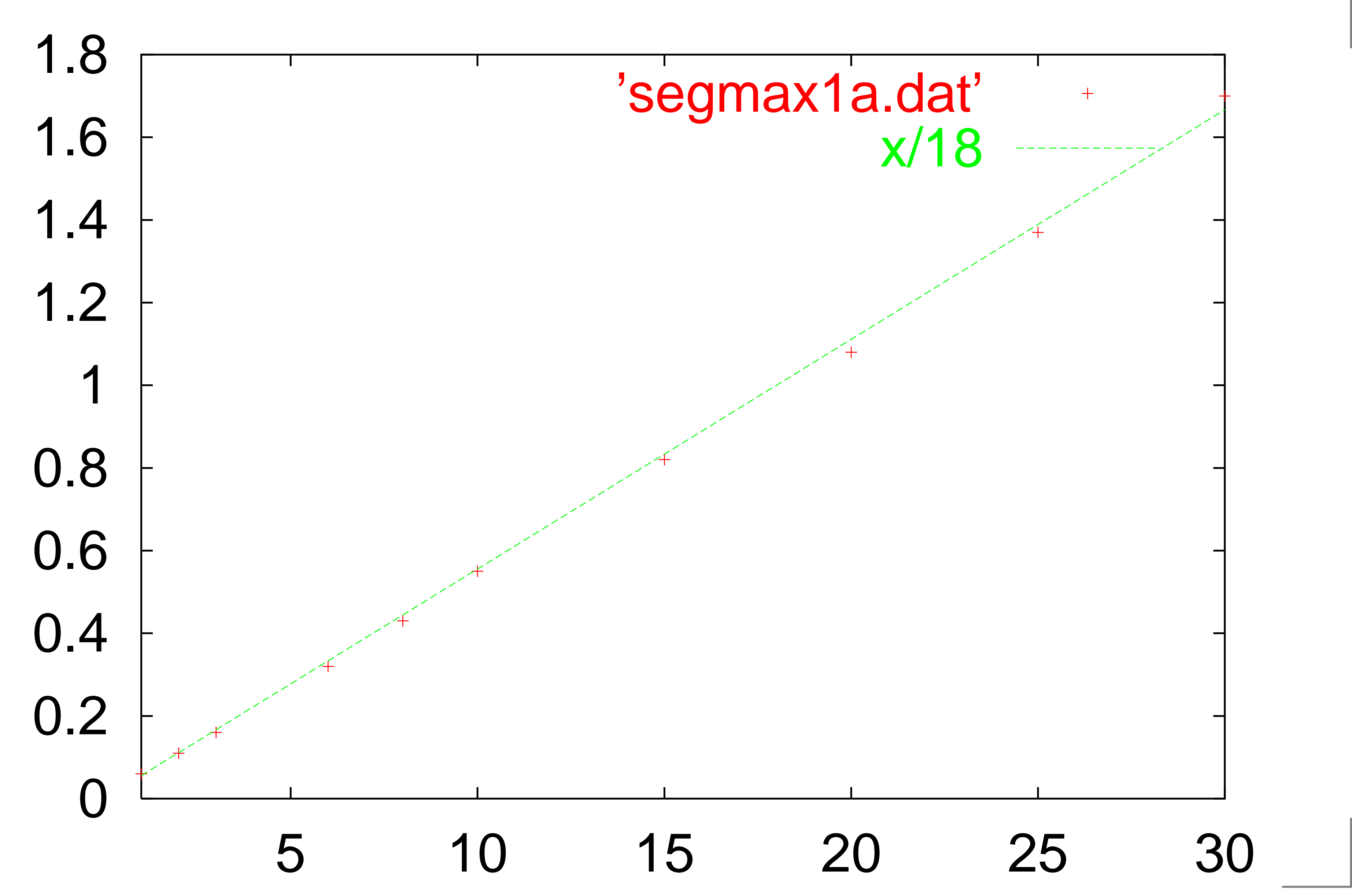
SEG-MAX-2



SEG-MAX-1



SEG-MAX-1



Exercícios

Exercício 10.A

Nos algoritmos convencionamos o segmento vazio como sendo o de soma máxima se $A[1\,..\,n]$ tem apenas números negativos. Suponha que em vez disso tivéssemos decidido que nesse caso o maior elemento de $A[1\,..\,n]$ é o segmento de soma máxima. Como isto alteraria s quatro algoritmos?

Exercício 10.B

Suponha que desejamos determinar um segmento como soma mais próxima de zero, em vez do de soma máxima. Projete um algoritmo para esta tarefa. Qual é o consumo de tempo do seu algoritmo? E se estivéssemos interessados em um algoritmo para encontrar um segmento de soma mais próxima de um dado valor?

Exercício 10.C

No problema do retângulo de soma máxima é dada uma matriz $A[1\,..\,n, 1\,..\,n]$ de números inteiros e deseja-se determinar um retângulo de soma máxima. Projete um algoritmo para este problema. Qual o consumo de tempo do seu algoritmo?

Exercício 10.D

Modifique o algoritmo SEG-MAX-DC para que seu consumo de tempo seja linear.

Mais exercícios

Exercício 10.E
Demonstre que qualquer algoritmo para o problema do segmento de soma máxima deve examinar cada um dos n elementos do vetor. (Algoritmos para alguns problemas podem, corretamente, ignorar alguns dos dados; o algoritmo de Boyer e Moore para busca de padrões é um exemplo.)

Exercício 10.F
Projete um algoritmo que recebe um vetor $A[1\,..\,n]$ de números inteiros e um número inteiro m e devolve um índice i tal que $1 \leq i \leq n-m$ e a soma $A[i] + \cdots + A[i+m]$ seja próxima de zero. Qual o consumo de tempo do seu algoritmo.

Exercício 10.G
Resolva a recorrência

$$T(1) = 0$$
$$T(n) = 2\,T(n/2) + n \quad \text{para } n = 2, 4, 8, \ldots$$

Demonstre por indução que a sua solução está correta.

Melhores momentos

AULAS ANTERIORES

Classe O da solução de uma recorrência

Não faço questão de solução exata: basta solução aproximada (em notação O; melhor ainda Θ)

Exemplo:

$$
\begin{aligned}
G(1) &= 1 \\
G(n) &= 2G(n/2) + 7n + 2 \quad \text{para } n = 2, 4, 8, 16, \ldots
\end{aligned}
$$

Solução exata: $G(n) = 7n \lg n + 3n - 2$

Solução aproximada: $G(n) = \mathrm{O}(n \lg n)$ $\quad$ ($G(n) = \Theta(n \lg n)$!)

Em geral, é mais fácil obter e provar solução aproximada que solução exata

Dica prática (sem prova)

A solução da recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(\lfloor n/2 \rfloor) + 7n + 2 \quad \text{para } n = 2, 3, 4, 5, \ldots \end{aligned}$$

está na mesma classe Θ que a solução de

$$\begin{aligned} T'(1) &= 1 \\ T'(n) &= 2T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \ldots \end{aligned}$$

e na mesma classe Θ que a solução de

$$\begin{aligned} T''(4) &= 10 \\ T''(n) &= 2T''(n/2) + n \quad \text{para } n = 2^3, 2^4, 2^5, \ldots \end{aligned}$$

Recorrências com O do lado direito

A "recorrência"

$$T(n) = 2T(n/2) + \mathrm{O}(n)$$

representa todas as recorrências da forma
$T(n) = 2T(n/2) + f(n)$ em que $f(n)$ é $\mathrm{O}(n)$.

Melhor: representa todas as recorrências do tipo

$$\begin{aligned} T'(n) &\leq a \quad \text{para } n = k, k+1, \ldots, 2k-1 \\ T'(n) &\leq 2T'(\lfloor n/2 \rfloor) + b\,n \quad \text{para } n \geq 2k \end{aligned}$$

quaisquer que sejam a, $b > 0$ e $k > 0$
(poderíamos tomar $n_0 = 1$; veja ex ??.I)

Teorema Ban-Ban-Ban

Teorema Mestre (Master Theorem, CLRS, sec. 4.3, p.73):

Suponha

$$T(n) = a\,T\big(n/b\big) + f(n)$$

para algum $a \geq 1$ e $b > 1$ e onde n/b significa $\lceil n/b \rceil$ ou $\lfloor n/b \rfloor$.
Então, em geral,

$$\begin{array}{lll} \text{se } f(n) = \mathrm{O}(n^{\log_b a - \epsilon}) & \text{então} & T(n) = \Theta(n^{\log_b a}) \\ \text{se } f(n) = \Theta(n^{\log_b a}) & \text{então} & T(n) = \Theta(n^{\log_b a} \lg n) \\ \text{se } f(n) = \Omega(n^{\log_b a + \epsilon}) & \text{então} & T(n) = \Theta(f(n)) \end{array}$$

para qualquer $\epsilon > 0$.

Teorema Ban-Ban-Ban

Teorema Mestre Simplificado:

Suponha

$$T(n) = a\,T(n/b) + c\,n^k$$

para algum $a \geq 1$ e $b > 1$ e onde n/b significa $\lceil n/b \rceil$ ou $\lfloor n/b \rfloor$.
Então, em geral,

se $a > b^k$ então $T(n) = \Theta(n^{\log_b a})$
se $a = b^k$ então $T(n) = \Theta(n^k \lg n)$
se $a < b^k$ então $T(n) = \Theta(n^k)$

Segmento de soma máxima

Um segmento de um vetor $A[1\,..\,n]$ é um qualquer subvetor da forma $A[e\,..\,d]$.

Problema: Dado um vetor $A[1\,..\,n]$ de números inteiros, determinar um segmento $A[e\,..\,d]$ de soma máxima.

Entra:

	1									n
A	31	−41	59	26	−53	58	97	−93	−23	84

Sai:

	1		3				7			n
A	31	−41	59	26	−53	58	97	−93	−23	84

$A[e\,..\,d] = A[3\,..\,7]$ é segmento de soma máxima.
$A[3\,..\,7]$ tem soma 187.

Conclusões

O consumo de tempo do algoritmo SEG-MAX-3 é $\Theta(n^3)$.

O consumo de tempo do algoritmo SEG-MAX-2 é $\Theta(n^2)$.

O consumo de tempo do algoritmo SEG-MAX-DC é $\Theta(n \lg n)$.

O consumo de tempo do algoritmo SEG-MAX-1 é $\Theta(n)$.

Técnicas

- Evitar recomputações. Usar espaço para armazenar resultados a fim de evitar recomputá-los (SEG-MAX-2, SEG-MAX-1, programação dinâmica).
- Pré-processar os dados. O HEAPSORT pré-processa os dados armazenandos em uma entrutura de dados para realizar computações futuras mais eficientemente.
- Divisão-e-conquista. Os algoritmos Mergesort e SegMaxdc utilizam uma forma conhecidade dessa técnica.
- Algoritmos incrementais/varredura. Como estender a solução de um subproblema a uma solução do problema (SegMaxu).
- Delimitação inferior. Projetistas de algoritmos só dormem em paz quando sabem que seus algoritmos são o melhor possível (SegMaxu).

Resultados experimentais

n	SEG-MAX-2		SEG-MAX-3	
	clock	user time	clock	user time
1000	0.01	0.01	2.91	2.92
2000	0.03	0.03	23.00	23.04
3000	0.08	0.08	77.58	1m17.57
5000	0.24	0.24	360.78	6m0.45s
10000	1.16	1.62	2940	49m27.32
20000	4.8	4.82	?	?
50000	42.8	42.96	?	?
100000	188.0	3m7.58s	?	?
200000	911.74	15m10.34s	?	?
400000	?	64m22.880s	?	?

Resultados experimentais

n (M)	SEG-MAX-1 A		SEG-MAX-1 B	
	`clock`	`user time`	`clock`	`user time`
1	0.06	0.11	0.08	0.07
2	0.11	0.23	0.15	0.06
3	0.16	0.28	0.30	0.12
6	0.32	0.63	0.56	0.29
8	0.43	0.80	0.71	0.43
10	0.55	1.03	1.00	0.52
15	0.82	1.55	1.37	0.64
20	1.08	2.03	2.17	0.95
25	1.37	2.52	2.44	1.38
30	1.70	3.13	3.28	1.68

Todos os tempos estão em segundos.

AULA 7

Mais análise de algoritmos recursivos

CLRS 2.1–2.2, 28.2

AU 3.3, 3.6 (muito bom)

Notas de AA de Jeff Edmonds:

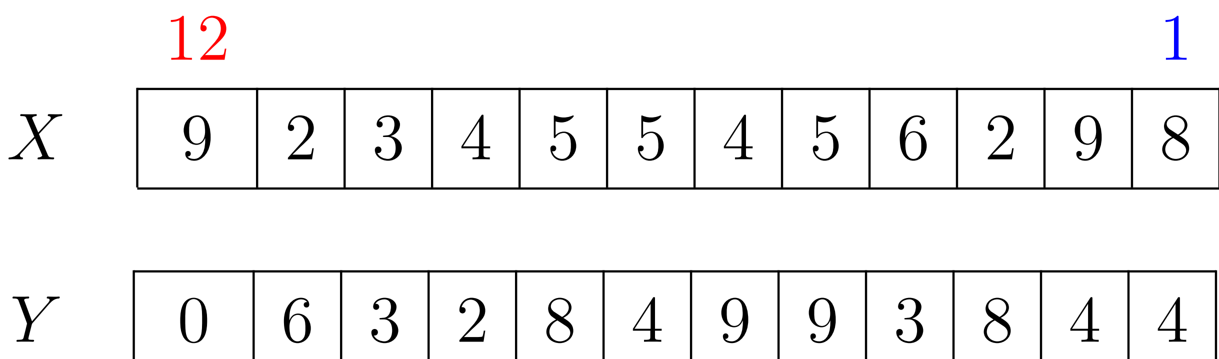
`http://www.cs.yorku.ca/~jeff/notes/3101/`

Multiplicação de inteiros gigantescos

$n :=$ número de algarismos.

Problema: Dados dois números inteiros $X[1\,..\,n]$ e $Y[1\,..\,n]$ calcular o produto $X \cdot Y$.

Entra: Exemplo com $n = 12$

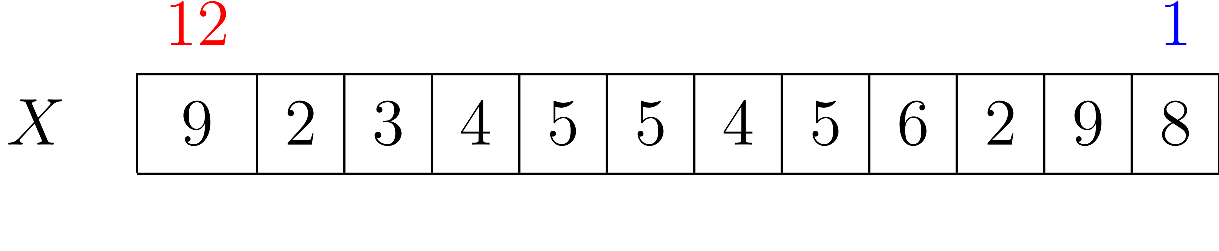


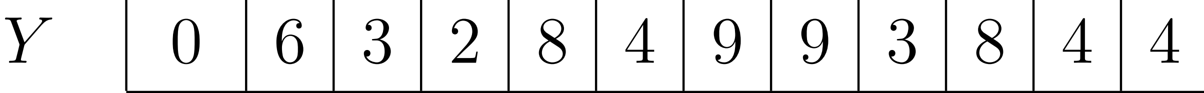
Multiplicação de inteiros gigantescos

$n :=$ número de algarismos.

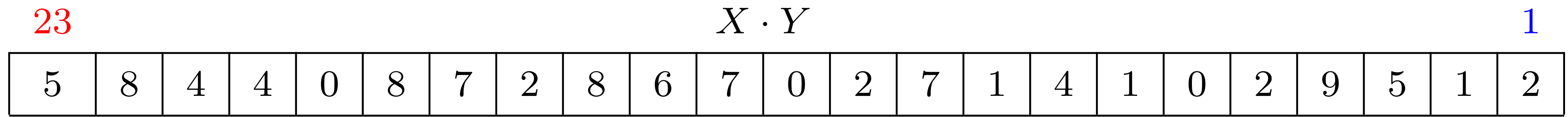
Problema: Dados dois números inteiros $X[1\,..\,n]$ e $Y[1\,..\,n]$ calcular o produto $X \cdot Y$.

Entra: Exemplo com $n = 12$



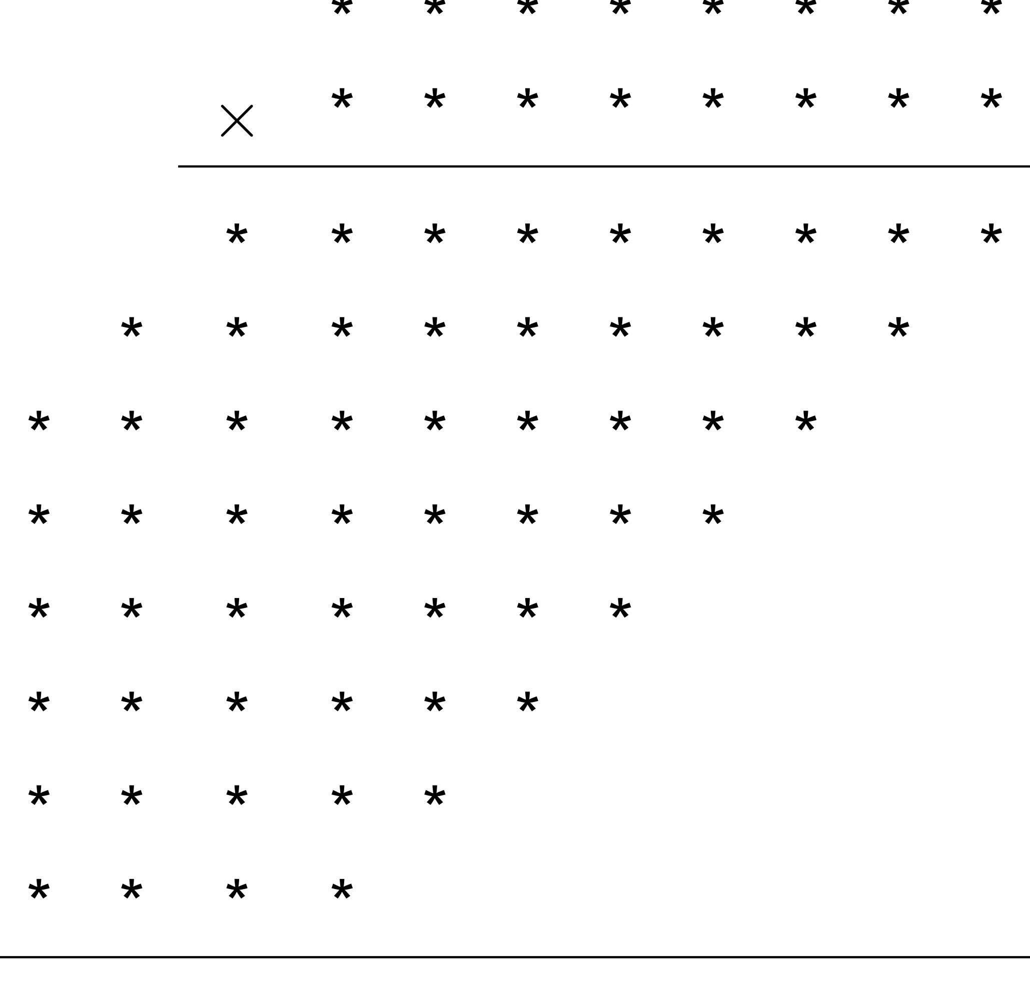


Sai:



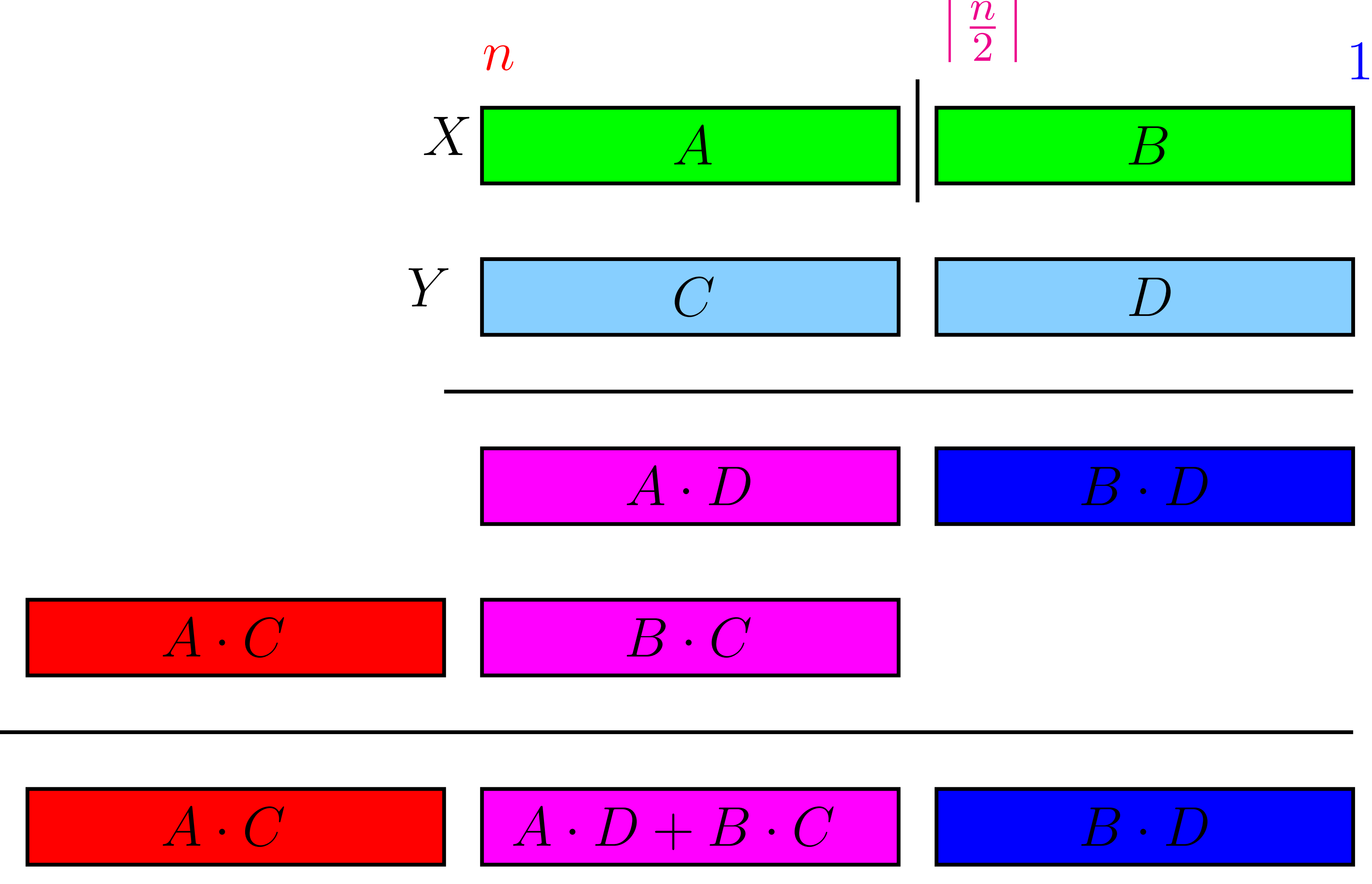
Aplicações em esquemas criptográficos.

Algoritmo do ensino fundamental



O algoritmo do ensino fundamental é $\Theta(n^2)$.

Divisão e conquista



$$X \cdot Y = A \cdot C \times 10^{2\lceil n/2 \rceil} + (A \cdot D + B \cdot C) \times 10^{\lceil n/2 \rceil} + B \cdot D$$

Exemplo

	4			1
X	3	1	4	1

	4			1
Y	5	9	3	6

Exemplo

	4			1		4			1
X	3	1	4	1	Y	5	9	3	6

A	3	1	B	4	1	C	5	9	D	3	6

Exemplo

	4			1
X	3	1	4	1

	4			1
Y	5	9	3	6

A | 3 | 1 | B | 4 | 1 | C | 5 | 9 | D | 3 | 6 |

$$X \cdot Y = A \cdot C \times 10^4 + (A \cdot D + B \cdot C) \times 10^2 + B \cdot D$$

$A \cdot C = 1829$ $\quad (A \cdot D + B \cdot C) = 1116 + 2419 = 3535$

$B \cdot D = 1476$

$A \cdot C$	1	8	2	9	0	0	0	0
$(A \cdot D + B \cdot C)$			3	5	3	5	0	0
$B \cdot D$					1	4	7	6
$X \cdot Y =$	1	8	6	4	4	9	7	6

Algoritmo de Multi-DC

Algoritmo recebe inteiros $X[1\,..\,n]$ e $Y[1\,..\,n]$ e devolve $X \cdot Y$.

MULT (X, Y, n)
1 **se** $n = 1$ **devolva** $X \cdot Y$
2 $q \leftarrow \lceil n/2 \rceil$
3 $A \leftarrow X[q+1\,..\,n]$ $B \leftarrow X[1\,..\,q]$
4 $C \leftarrow Y[q+1\,..\,n]$ $D \leftarrow Y[1\,..\,q]$
5 $E \leftarrow$ MULT$(A, C, \lfloor n/2 \rfloor)$
6 $F \leftarrow$ MULT$(B, D, \lceil n/2 \rceil)$
7 $G \leftarrow$ MULT$(A, D, \lceil n/2 \rceil)$
8 $H \leftarrow$ MULT$(B, C, \lceil n/2 \rceil)$
9 $R \leftarrow E \times 10^{2\lceil n/2 \rceil} + (G + H) \times 10^{\lceil n/2 \rceil} + F$
10 **devolva** R

$T(n) =$ consumo de tempo do algoritmo para multiplicar dois inteiros com n algarismos.

Consumo de tempo

linha	todas as execuções da linha	
1	$=$	$\Theta(1)$
2	$=$	$\Theta(1)$
3	$=$	$\Theta(n)$
4	$=$	$\Theta(n)$
5	$=$	$T(\lfloor n/2 \rfloor)$
6	$=$	$T(\lceil n/2 \rceil)$
7	$=$	$T(\lceil n/2 \rceil)$
8	$=$	$T(\lceil n/2 \rceil)$
9	$=$	$\Theta(n)$
10	$=$	$\Theta(n)$
total	$=$	$T(\lfloor n/2 \rfloor) + 3\,T(\lceil n/2 \rceil) + \Theta(4n+2)$

Consumo de tempo

As dicas no nosso estudo de recorrências sugeri que a solução da recorrência

$$
\begin{aligned}
T(1) &= \Theta(1) \\
T(n) &= T(\lfloor n/2 \rfloor) + 3\,T(\lceil n/2 \rceil) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots
\end{aligned}
$$

está na mesma classe Θ que a solução de

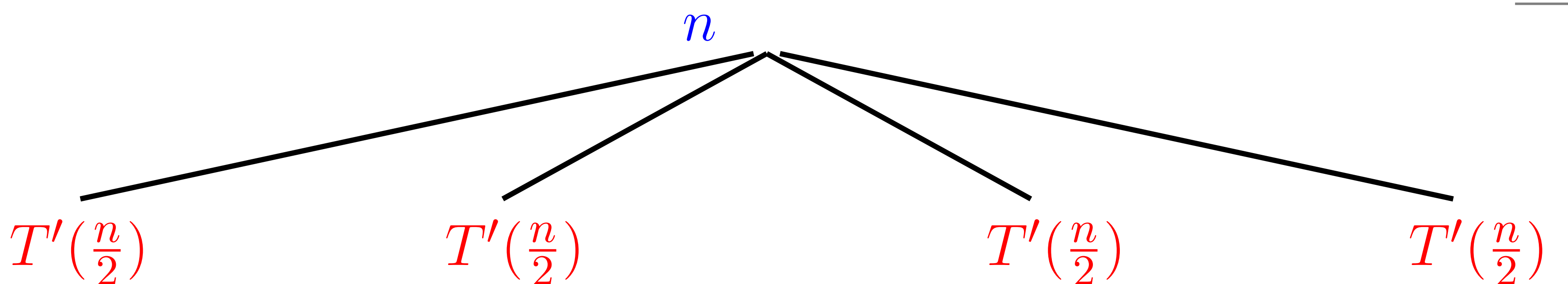
$$
\begin{aligned}
T'(1) &= 1 \\
T'(n) &= 4T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \ldots
\end{aligned}
$$

n	1	2	4	8	16	32	64	128	256	512
$T'(n)$	1	6	28	120	496	2016	8128	32640	130816	523776

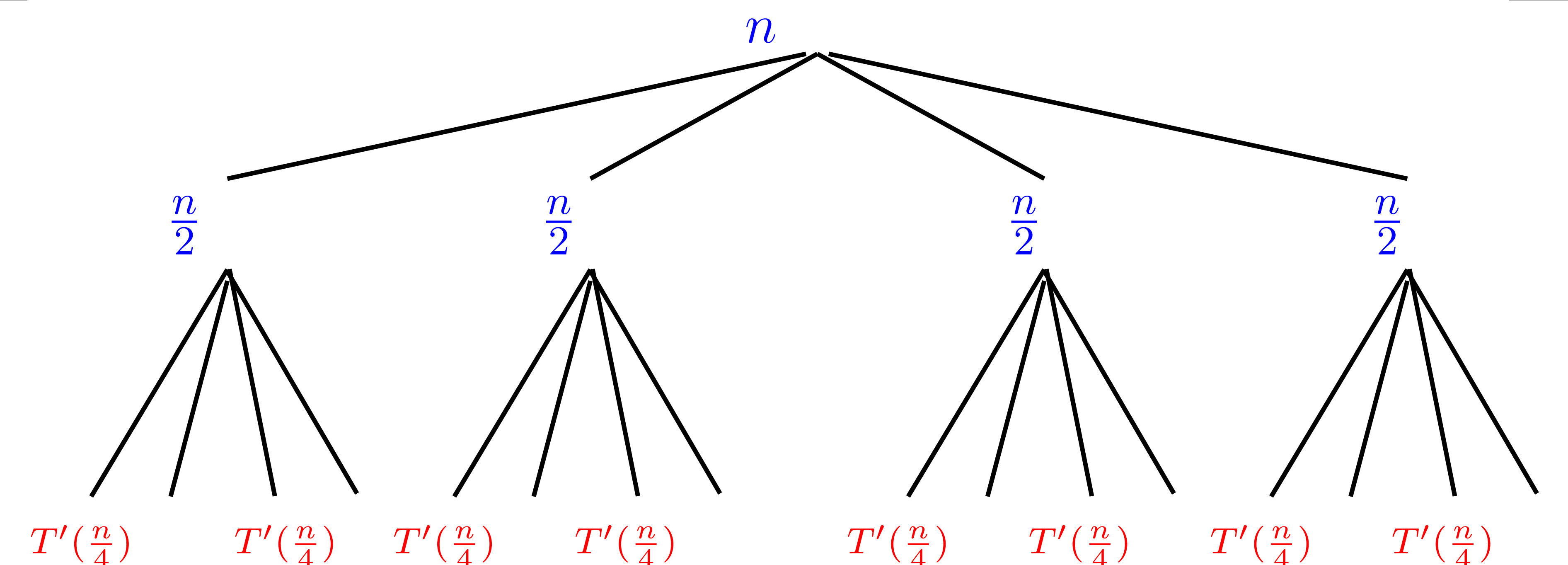
Árvore da recorrência

$$T'(n)$$

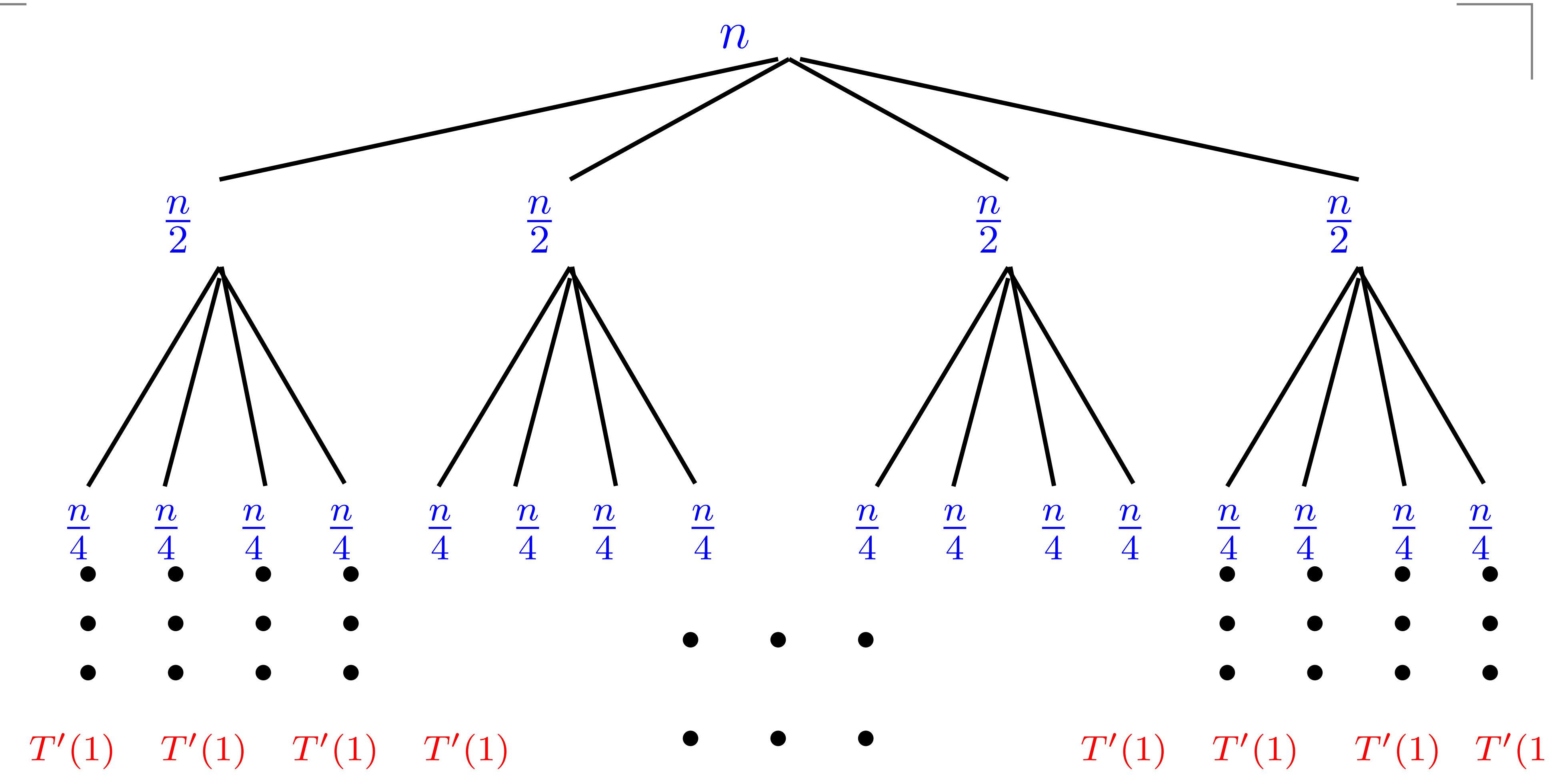
Árvore da recorrência



Árvore da recorrência



Árvore da recorrência



total de $1 + \lg n$ níveis

Contas

nível	0	1	2	$\dots$	$k-1$	k
soma	n	$2n$	$4n$	$\dots$	$2^{k-1}n$	$2^k n$

$n = 2^k$ $\qquad$ $k = \lg n$

Contas

nível	0	1	2	$\dots$	$k-1$	k
soma	n	$2n$	$4n$	$\dots$	$2^{k-1}n$	$2^k n$

$n = 2^k$ $\qquad$ $k = \lg n$

$$\begin{aligned} T'(n) &= n + 2n + \cdots + 2^{k-1}n + 2^k n \\ &= n\,(1 + 2 + 4 + \cdots + 2^k) \\ &= n\,(2^{k+1} - 1) \\ &= n\,(2n - 1) \qquad (k = \lg n) \\ &= 2n^2 - n = \Theta(n^2) \end{aligned}$$

ÊPA, não melhorou ...

Teorema Bambambã

Teorema Mestre Simplificado:

Suponha

$$T(n) = a\,T(n/b) + c\,n^k$$

para algum $a \geq 1$ e $b > 1$ e onde n/b significa $\lceil n/b \rceil$ ou $\lfloor n/b \rfloor$.
Então, em geral,

se $a > b^k$ então $T(n) = \Theta(n^{\log_b a})$
se $a = b^k$ então $T(n) = \Theta(n^k \lg n)$
se $a < b^k$ então $T(n) = \Theta(n^k)$

Conclusões

$T'(n)$ é $\Theta(n^2)$.

$T(n)$ é $\Theta(n^2)$.

O consumo de tempo do algoritmo MULT é $\Theta(n^2)$.

Tanto trabalho por nada ...
Será?!?

Pensar pequeno

Olhar para números com 2 algarismos ($n = 2$).

Suponha $X = a\,b$ e $Y = c\,d$.
Se cada multiplicação custa R$ 1,00 e
cada soma custa R$ 0,01, quanto custa $X \cdot Y$?

Pensar pequeno

Olhar para números com 2 algarismos ($n = 2$).

Suponha $X = a\,b$ e $Y = c\,d$.
Se cada multiplicação custa R$ 1,00 e
cada soma custa R$ 0,01, quanto custa $X \cdot Y$?
Eis $X \cdot Y$ por R$ 4,03:

X		a	b
Y		c	d
		ad	bd
	ac	bc	
$X \cdot Y$	ac	$ad + bc$	bd

$$X \cdot Y = ac \times 10^2 + (ad + bc) \times 10^1 + bd$$

Pensar pequeno

Olhar para números com 2 algarismos ($n = 2$).

Suponha $X = a\,b$ e $Y = c\,d$.
Se cada multiplicação custa R$ 1,00 e
cada soma custa R$ 0,01, quanto custa $X \cdot Y$?
Eis $X \cdot Y$ por R$ 4,03:

$$\begin{array}{cccc} X & & a & b \\ Y & & c & d \\ \hline & & ad & bd \\ & ac & bc & \\ \hline X \cdot Y & ac & ad + bc & bd \end{array}$$

$$X \cdot Y = ac \times 10^2 + (ad + bc) \times 10^1 + bd$$

Solução mais barata?

Pensar pequeno

Olhar para números com 2 algarismos ($n = 2$).

Suponha $X = a\,b$ e $Y = c\,d$.
Se cada multiplicação custa R$ 1,00 e
cada soma custa R$ 0,01, quanto custa $X \cdot Y$?
Eis $X \cdot Y$ por R$ 4,03:

X		a	b
Y		c	d
		ad	bd
	ac	bc	
$X \cdot Y$	ac	$ad + bc$	bd

$$X \cdot Y = ac \times 10^2 + (ad + bc) \times 10^1 + bd$$

Solução mais barata?
Gauss faz por R$ 3,06!

$X \cdot Y$ por apenas R$ 3,06

X		a	b
Y		c	d
		ad	bd
	ac	bc	
$X \cdot Y$	ac	$ad + bc$	bd

$X \cdot Y$ por apenas R$ 3,06

X		a	b
Y		c	d
		ad	bd
	ac	bc	
$X \cdot Y$	ac	$ad + bc$	bd

$$(a+b)(c+d) = ac + ad + bc + bd \Rightarrow$$
$$ad + bc = (a+b)(c+d) - ac - bd$$

$$g = (a+b)(c+d) \qquad e = ac \qquad f = bd \qquad h = g - e - f$$

$$X \cdot Y \ \text{(por R\$ 3,06)} = e \times 10^2 + h \times 10^1 + f$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$

$$ac = ? \qquad bd = ? \qquad (a+b)(c+d) = ?$$

Exemplo

$$X = \quad 2133 \quad Y = \quad 2312 \qquad\qquad X \cdot Y = \quad ?$$

$$ac = \quad ? \qquad bd = \quad ? \qquad (a+b)(c+d) = \quad ?$$

$$X = \quad 21 \quad Y = \quad 23 \qquad\qquad X \cdot Y = \quad ?$$

$$ac = \quad ? \qquad bd = \quad ? \qquad (a+b)(c+d) = \quad ?$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$
$$ac = ? \qquad bd = ? \qquad (a+b)(c+d) = ?$$

$$X = 21 \quad Y = 23 \qquad X \cdot Y = ?$$
$$ac = ? \qquad bd = ? \qquad (a+b)(c+d) = ?$$

$$X = 2 \quad Y = 2 \quad X \cdot Y = 4$$

Exemplo

$$X = \ 2133 \quad Y = \ 2312 \qquad X \cdot Y = \ ?$$

$$ac = \ ? \quad bd = \ ? \qquad (a+b)(c+d) = \ ?$$

$$X = \ 21 \quad Y = \ 23 \qquad X \cdot Y = \ ?$$

$$ac = \ 4 \quad bd = \ ? \qquad (a+b)(c+d) = \ ?$$

Exemplo

$X = \quad 2133 \quad Y = \quad 2312 \qquad X \cdot Y = \quad ?$

$ac = \quad ? \qquad bd = \quad ? \qquad (a+b)(c+d) = \quad ?$

$X = \quad 21 \quad Y = \quad 23 \qquad X \cdot Y = \quad ?$

$ac = \quad 4 \qquad bd = \quad ? \qquad (a+b)(c+d) = \quad ?$

$X = \quad 1 \quad Y = \quad 3 \quad X \cdot Y = \quad 3$

Exemplo

$X = \ 2133 \quad Y = \ 2312 \qquad\qquad X \cdot Y = \ ?$

$ac = \ ? \qquad bd = \ ? \qquad (a+b)(c+d) = \ ?$

$X = \ 21 \quad Y = \ 23 \qquad\qquad X \cdot Y = \ ?$

$ac = \ 4 \qquad bd = \ 3 \qquad (a+b)(c+d) = \ ?$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$

$$ac = ? \qquad bd = ? \qquad (a+b)(c+d) = ?$$

$$X = 21 \quad Y = 23 \qquad X \cdot Y = ?$$

$$ac = 4 \qquad bd = 3 \qquad (a+b)(c+d) = ?$$

$$X = 3 \quad Y = 5 \quad X \cdot Y = 15$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$
$$ac = ? \quad bd = ? \qquad (a+b)(c+d) = ?$$

$$X = 21 \quad Y = 23 \qquad X \cdot Y = 483$$
$$ac = 4 \quad bd = 3 \qquad (a+b)(c+d) = 15$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$

$$ac = 483 \quad bd = ? \qquad (a+b)(c+d) = ?$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$
$$ac = 483 \quad bd = ? \qquad (a+b)(c+d) = ?$$

$$X = 33 \quad Y = 12 \qquad X \cdot Y = ?$$
$$ac = ? \quad bd = ? \qquad (a+b)(c+d) = ?$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$

$$ac = 483 \quad bd = ? \qquad (a+b)(c+d) = ?$$

$$X = 33 \quad Y = 12 \qquad X \cdot Y = 396$$

$$ac = 3 \quad bd = 6 \qquad (a+b)(c+d) = 27$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = \; ?$$

$$ac = 483 \quad bd = 396 \qquad (a+b)(c+d) = \; ?$$

Exemplo

$$X = \; 2133 \quad Y = \; 2312 \qquad X \cdot Y = \; ?$$

$$ac = \; 483 \quad bd = \; 396 \quad (a+b)(c+d) = \; ?$$

$$X = \; 54 \quad Y = \; 35 \qquad X \cdot Y = \; ?$$

$$ac = \; ? \quad bd = \; ? \quad (a+b)(c+d) = \; ?$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = ?$$
$$ac = 483 \quad bd = 396 \quad (a+b)(c+d) = ?$$

$$X = 54 \quad Y = 35 \qquad X \cdot Y = 1890$$
$$ac = 15 \quad bd = 20 \quad (a+b)(c+d) = 72$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = \ ?$$

$$ac = 483 \quad bd = 396 \qquad (a+b)(c+d) = 1890$$

Exemplo

$$X = 2133 \quad Y = 2312 \qquad X \cdot Y = 4931496$$

$$ac = 483 \quad bd = 396 \quad (a+b)(c+d) = 1890$$

Algoritmo Karatsuba

Algoritmo recebe inteiros $X[1\,.\,.\,n]$ e $Y[1\,.\,.\,n]$ e devolve $X \cdot Y$ (Karatsuba e Ofman).

KARATSUBA (X, Y, n)
1 **se** $n \leq 3$ **devolva** $X \cdot Y$
2 $q \leftarrow \lceil n/2 \rceil$
3 $A \leftarrow X[q+1\,.\,.\,n]$ $B \leftarrow X[1\,.\,.\,q]$
4 $C \leftarrow Y[q+1\,.\,.\,n]$ $D \leftarrow Y[1\,.\,.\,q]$
5 $E \leftarrow$ KARATSUBA$(A, C, \lfloor n/2 \rfloor)$
6 $F \leftarrow$ KARATSUBA$(B, D, \lceil n/2 \rceil)$
7 $G \leftarrow$ KARATSUBA$(A+B, C+D, \lceil n/2 \rceil + 1)$
8 $H \leftarrow G - F - E$
9 $R \leftarrow E \times 10^{2\lceil n/2 \rceil} + H \times 10^{\lceil n/2 \rceil} + F$
10 **devolva** R

$T(n) =$ consumo de tempo do algoritmo para multiplicar dois inteiros com n algarismos.

Consumo de tempo

linha		todas as execuções da linha
1	$=$	$\Theta(1)$
2	$=$	$\Theta(1)$
3	$=$	$\Theta(n)$
4	$=$	$\Theta(n)$
5	$=$	$T(\lfloor n/2 \rfloor)$
6	$=$	$T(\lceil n/2 \rceil)$
7	$=$	$T(\lceil n/2 \rceil + 1)$
8	$=$	$\Theta(n)$
9	$=$	$\Theta(n)$
10	$=$	$\Theta(n)$
total	$=$	$T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + \Theta(5n + 2)$

Consumo de tempo

As dicas no nosso estudo de recorrências sugerem que a solução da recorrência

$$\begin{aligned} T(n) &= \Theta(1) \quad \text{para } n = 1,2,3 \\ T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil + 1) + \Theta(n) \quad n \geq 4 \end{aligned}$$

está na mesma classe Θ que a solução de

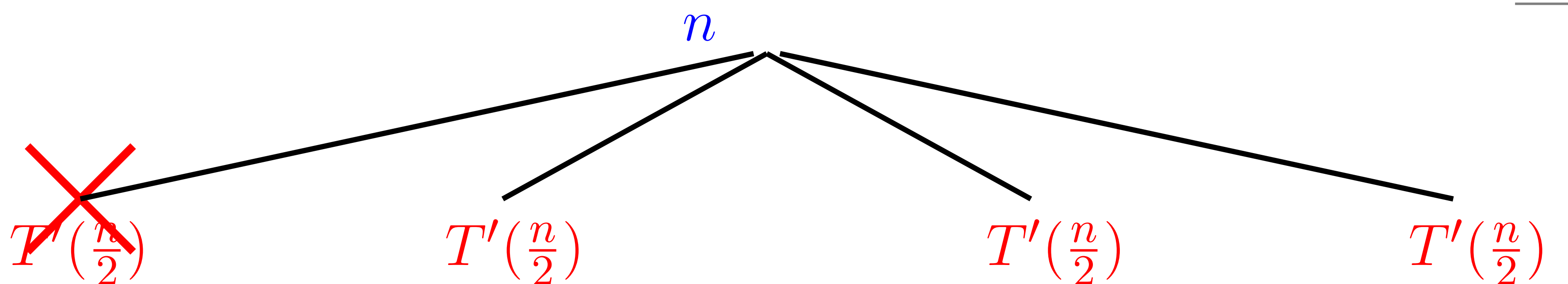
$$\begin{aligned} T'(1) &= 1 \\ T'(n) &= 3T'(n/2) + n \quad \text{para } n = 2, 2^2, 2^3, \ldots \end{aligned}$$

n	1	2	4	8	16	32	64	128	256	512
$T'(n)$	1	5	19	65	211	665	2059	6305	19171	58025

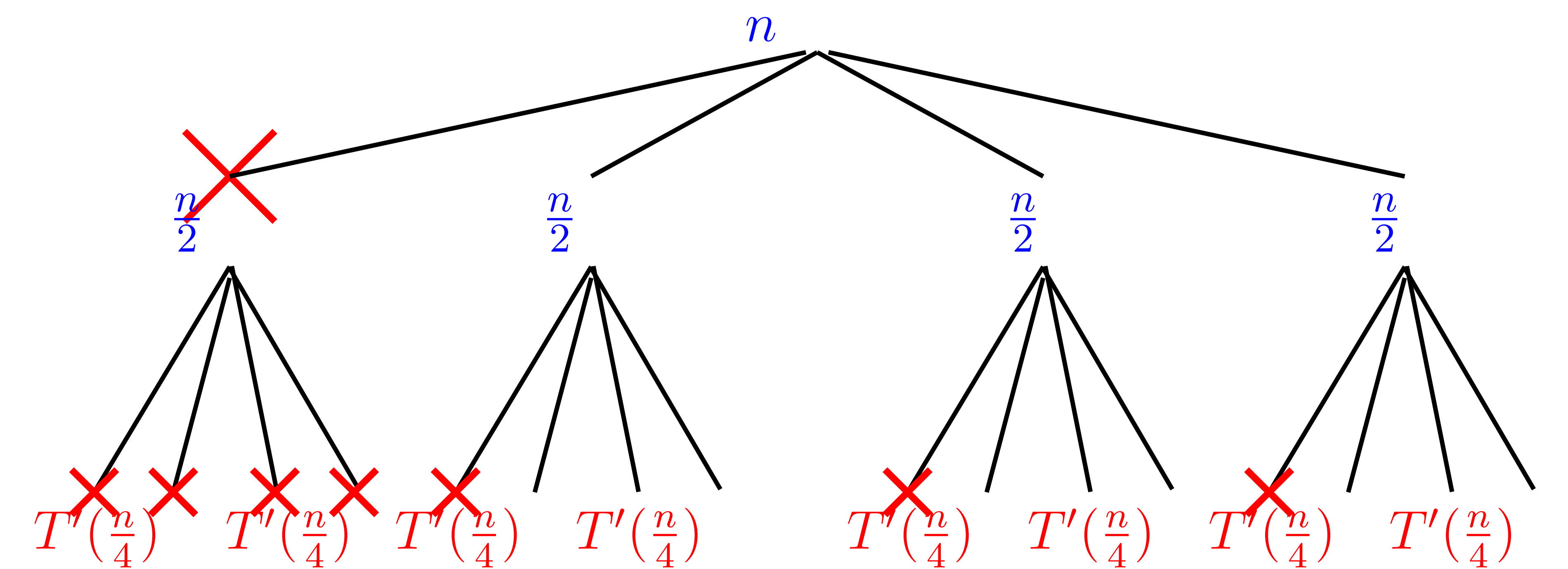
Árvore da recorrência

$$T'(n)$$

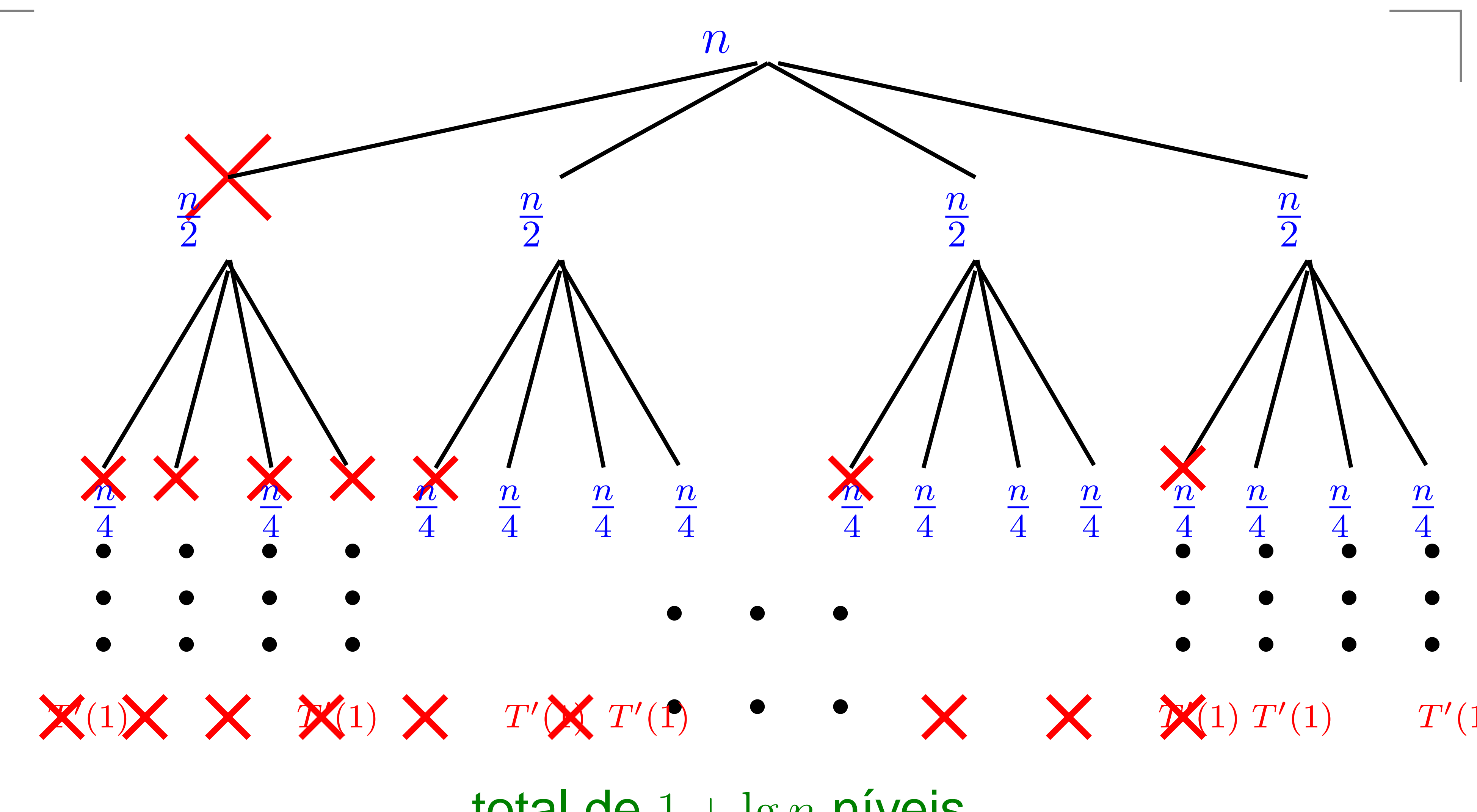
Árvore da recorrência



Árvore da recorrência



Árvore da recorrência



total de $1 + \lg n$ níveis

Contas

nível	0	1	2	$\ldots$	$k-1$	k
soma	n	$(3/2)n$	$(3/2)^2 n$	$\ldots$	$(3/2)^{k-1} n$	$(3/2)^k n$

$n = 2^k \qquad k = \lg n$

Contas

nível	0	1	2	...	$k-1$	k
soma	n	$(3/2)n$	$(3/2)^2 n$	...	$(3/2)^{k-1} n$	$(3/2)^k n$

$n = 2^k \qquad k = \lg n$

$$\begin{aligned} T'(n) &= n + (3/2)n + \cdots + (3/2)^{k-1} n + (3/2)^k n \\ &= n\,(1 + (3/2) + (3/2)^2 + \cdots + (3/2)^k) \\ &= n\,(3(3/2)^k - 2) \\ &= n \left(3 \frac{3^{\lg n}}{2^{\lg n}} - 2\right) \qquad (\text{pois}, k = \lg n) \\ &= 3 \cdot 3^{\lg n} - 2n = 3n^{\lg 3} - 2n = \Theta(n^{\lg 3}) \end{aligned}$$

$1{,}584 < \lg 3 < 1.585$

LEGAL, estamos de volta ao jogo!

Recorrência

Considere a recorrência

$$
\begin{aligned}
R(1) &= 1 \\
R(2) &= 1 \\
R(3) &= 1 \\
R(n) &= 3R(\lceil \tfrac{n}{2} \rceil + 1) + n \quad \text{para } n = 4, 5, 6 \ldots
\end{aligned}
$$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	1	1	7	14	15	29	36	45	53
$R(n)$	1	1	1	7	26	27	85	86	90	91

Recorrência

Considere a recorrência

$$
\begin{aligned}
R(1) &= 1 \\
R(2) &= 1 \\
R(3) &= 1 \\
R(n) &= 3R(\lceil \tfrac{n}{2} \rceil + 1) + n \quad \text{para } n = 4, 5, 6 \ldots
\end{aligned}
$$

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	1	1	7	14	15	29	36	45	53
$R(n)$	1	1	1	7	26	27	85	86	90	91

Vamos mostra que $R(n)$ é $\mathrm{O}(n^{\lg 3})$.
Isto implica que $T(n)$ é $\mathrm{O}(n^{\lg 3})$.

Solução assintótica da recorrência

Vou mostrar que $R(n) \leq 31\,(n-3)^{\lg 3} - 6n$ para $n = 4, 5, 6, \ldots$

n	1	2	3	4	5	6	7	8	9	10
$R(n)$	1	1	1	7	26	27	85	86	90	91
$31(n-3)^{\lg 3} - 6n$	$*$	$*$	$*$	7	63	119	237	324	473	5910

Solução assintótica da recorrência

Vou mostrar que $R(n) \leq 31\,(n-3)^{\lg 3} - 6n$ para $n = 4, 5, 6, \ldots$

n	1	2	3	4	5	6	7	8	9	10
$R(n)$	1	1	1	7	26	27	85	86	90	91
$31(n-3)^{\lg 3} - 6n$	*	*	*	7	63	119	237	324	473	5910

Prova:
Se $n = 4$, então $R(n) = 7 = 31(n-3)^{\lg 3} - 6n$.

Solução assintótica da recorrência

Prova: (continuação) Se $n \geq 5$ vale que

$$R(n) = 3R(\lceil n/2 \rceil + 1) + n$$

hi

$$\leq 3(31(\lceil n/2 \rceil + 1 - 3)^{\lg 3} - 6(\lceil n/2 \rceil + 1)) + n$$

$$\leq 3(31\left(\frac{(n+1)}{2} - 2\right)^{\lg 3} - 6\left(\frac{n}{2} + 1\right)) + n$$

$$= 3(31\left(\frac{(n-3)}{2}\right)^{\lg 3} - 3n - 6) + n$$

$$= 3(31\frac{(n-3)^{\lg 3}}{2^{\lg 3}} - 3n - 6) + n$$

$$= 3 \cdot 31\frac{(n-3)^{\lg 3}}{3} - 9n - 18 + n$$

$$= 31(n-3)^{\lg 3} - 6n - 2n - 18$$

$$< 31(n-3)^{\lg 3} - 6n = \mathrm{O}(n^{\lg 3})$$

Teorema Bambambã

Teorema Mestre Simplificado:

Suponha

$$T(n) = a\,T(n/b) + c\,n^k$$

para algum $a \geq 1$ e $b > 1$ e onde n/b significa $\lceil n/b \rceil$ ou $\lfloor n/b \rfloor$.
Então, em geral,

se $a > b^k$ então $T(n) = \Theta(n^{\log_b a})$
se $a = b^k$ então $T(n) = \Theta(n^k \lg n)$
se $a < b^k$ então $T(n) = \Theta(n^k)$

Conclusões

$R(n)$ é $\Theta(n^{\lg 3})$.

Conclusão anterior$+$Exercício 9.C $\Rightarrow$
$T(n)$ é $\Theta(n^{\lg 3})$.

O consumo de tempo do algoritmo KARATSUBA é $\Theta(n^{\lg 3})$ ($1{,}584 < \lg 3 < 1{,}585$).

Mais conclusões

Consumo de tempo de algoritmos para multiplicação de inteiros:

Jardim de infância	$\Theta(n\,10^n)$
Ensino fundamental	$\Theta(n^2)$
Karatsuba e Ofman	$\Theta(n^{\lg 3}) = \mathrm{O}(n^{1{,}585})$
Schönhage e Strassen (FFT)	$\mathrm{O}(n \lg n \lg\lg n)$

$1{,}584 < \lg 3 < 1{,}585$

Ambiente experimental

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

Os códigos estão compilados com o gcc versão 2.7.2.1 e opção de compilação -O2.

As implementações comparadas neste experimento são as do algoritmo do ensino fundamental e do algoritmo KARATSUBA.

O programa foi escrito por Carl Burch:

`http://www-2.cs.cmu.edu/~cburch/251/karat/.`

Resultados experimentais

n	Ensino Fund.	KARATSUBA
4	0.005662	0.005815
8	0.010141	0.010600
16	0.020406	0.023643
32	0.051744	0.060335
64	0.155788	0.165563
128	0.532198	0.470810
256	1.941748	1.369863
512	7.352941	4.032258

Tempos em 10^{-3} segundos.

Multiplicação de matrizes

Problema: Dadas duas matrizes $X[1\,..\,n, 1\,..\,n]$ e $Y[1\,..\,n, 1\,..\,n]$ calcular o produto $X \cdot Y$.

Os algoritmo tradicional de multiplicação de matrizes consome tempo $\Theta(n^3)$.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

$$r = ae + bg$$

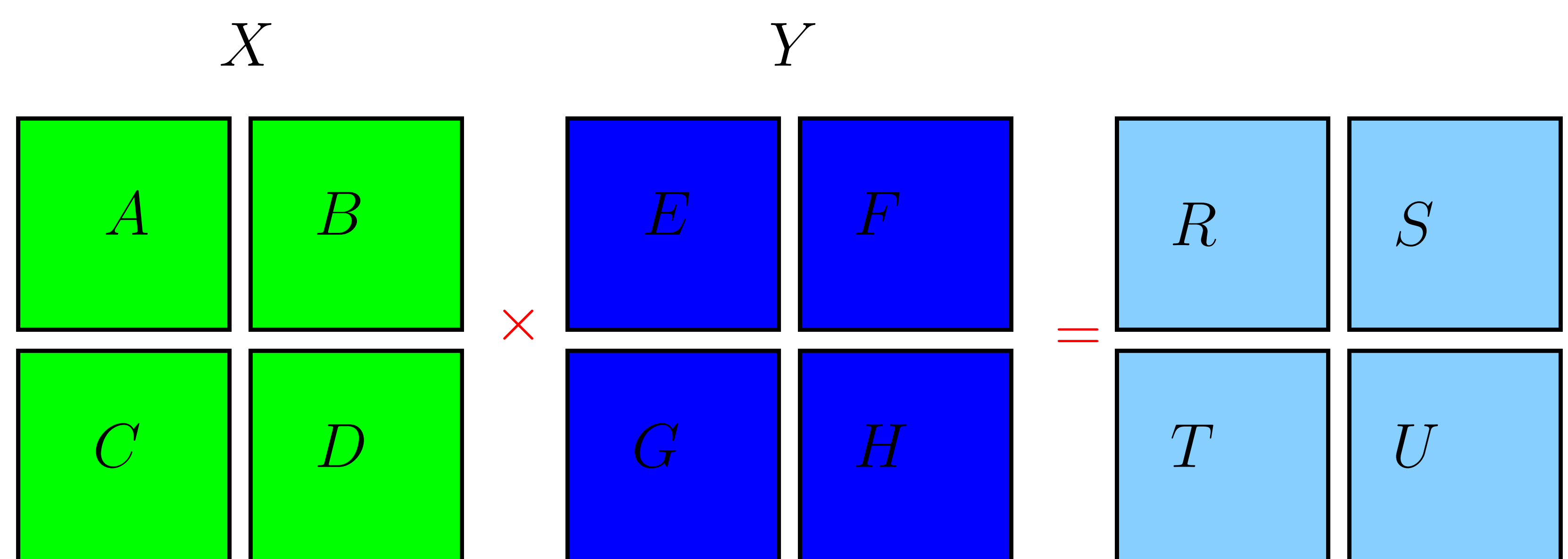
$$s = af + bh$$

$$t = ce + dg$$

$$u = cf + dh$$

Solução custa R$ 8,04

Divisão e conquista



$$R = AE + BG$$

$$S = AF + BH$$

$$T = CE + DG$$

$$U = CF + DH$$

Algoritmo de Multi-Mat

Algoritmo recebe inteiros $X[1\,..\,n]$ e $Y[1\,..\,n]$ e devolve $X \cdot Y$.

MULTI-M (X, Y, n) $\triangleright$ supõe n da forma 2^k

1 **se** $n = 1$ **devolva** $X \cdot Y$
2 $(A, B, C, D) \leftarrow$ PARTICIONE(X, n)
3 $(E, F, G, H) \leftarrow$ PARTICIONE(Y, n)
4 $R \leftarrow$ MULTI-M$(A, E, n/2)$ + MULTI-M$(B, G, n/2)$
5 $S \leftarrow$ MULTI-M$(A, F, n/2)$ + MULTI-M$(B, H, n/2)$
6 $T \leftarrow$ MULTI-M$(C, E, n/2)$ + MULTI-M$(D, G, n/2)$
7 $U \leftarrow$ MULTI-M$(C, F, n/2)$ + MULTI-M$(D, H, n/2)$
8 $P \leftarrow$ CONSTRÓI-MAT(R, S, T, U)
9 **devolva** P

$T(n) =$ consumo de tempo do algoritmo para multiplicar duas matrizes de n linhas e n colunas.

Consumo de tempo

linha	todas as execuções da linha	
1	$=$	$\Theta(1)$
2	$=$	$\Theta(n^2)$
3	$=$	$\Theta(n^2)$
4	$=$	$T(n/2) + T(n/2)$
5	$=$	$T(n/2) + T(n/2)$
6	$=$	$T(n/2) + T(n/2)$
7	$=$	$T(n/2) + T(n/2)$
8	$=$	$\Theta(n^2)$
9	$=$	$\Theta(n^2)$
total	$=$	$8\,T(n/2) + \Theta(4n^2 + 1)$

Consumo de tempo

As dicas no nosso estudo de recorrências sugeri que a solução da recorrência

$$\begin{aligned} T(1) &= \Theta(1) \\ T(n) &= 8\,T(n/2) + \Theta(n^2) \quad \text{para } n = 2, 3, 4, \ldots \end{aligned}$$

está na mesma classe Θ que a solução de

$$\begin{aligned} T'(1) &= 1 \\ T'(n) &= 8T'(n/2) + n^2 \quad \text{para } n = 2, 2^2, 2^3, \ldots \end{aligned}$$

n	1	2	4	8	16	32	64	128	256
$T'(n)$	1	12	112	960	7936	64512	520192	4177920	3348889

Solução assintótica da recorrência

Considere a recorrência

$$
\begin{aligned}
R(1) &= 1 \\
R(n) &= 8\,R(\lceil \tfrac{n}{2} \rceil) + n^2 \quad \text{para } n = 2, 3, 4, \ldots
\end{aligned}
$$

Verifique por indução que $R(n) \leq 20(n-1)^3 - 2n^2$ para $n = 2, 3, 4 \ldots$.

n	1	2	3	4	5	6	7	8
$R(n)$	1	12	105	112	865	876	945	960
$20\,(n-1)^3 - 2n^2$	-2	12	142	508	1230	2428	4222	6732

Conclusões

$R(n)$ é $\Theta(n^3)$.

Conclusão anterior$+$Exercício $\Rightarrow$
$T(n)$ é $\Theta(n^3)$.

O consumo de tempo do algoritmo MULTI-M é $\Theta(n^3)$.

Strassen: $X \cdot Y$ por apenas R$ 7,18

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

Strassen: $X \cdot Y$ por apenas R$ 7,18

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

$$p_1 = a(f - h) = af - ah$$

$$p_2 = (a + b)h = ah + bh$$

$$p_3 = (c + d)e = ce + de$$

$$p_4 = d(g - e) = dg - de$$

$$p_5 = (a + d)(e + h) = ae + ah + de + dh$$

$$p_6 = (b - d)(g + h) = bg + bh - dg - dh$$

$$p_7 = (a - c)(e + f) = ae + af - ce - cf$$

Strassen: $X \cdot Y$ por apenas R$ 7,18

$$p_1 = a(f-h) = af - ah$$
$$p_2 = (a+b)h = ah + bh$$
$$p_3 = (c+d)e = ce + de$$
$$p_4 = d(g-e) = dg - de$$
$$p_5 = (a+d)(e+h) = ae + ah + de + dh$$
$$p_6 = (b-d)(g+h) = bg + bh - dg - dh$$
$$p_7 = (a-c)(e+f) = ae + af - ce - cf$$

$$r = p_5 + p_4 - p_2 + p_6 = ae + bg$$
$$s = p_1 + p_2 = af + bh$$
$$t = p_3 + p_4 = ce + dg$$
$$u = p_5 + p_1 - p_3 - p_7 = cf + dh$$

Algoritmo de Strassen

STRASSEN (X, Y, n) ▷ supõe n da forma 2^k

1 **se** $n = 1$ **devolva** $X \cdot Y$
2 $(A, B, C, D) \leftarrow$ PARTICIONE(X, n)
3 $(E, F, G, H) \leftarrow$ PARTICIONE(Y, n)
4 $P_1 \leftarrow$ STRASSEN$(A, F - H, n/2)$
5 $P_2 \leftarrow$ STRASSEN$(A + B, H, n/2)$
6 $P_3 \leftarrow$ STRASSEN$(C + D, E, n/2)$
7 $P_4 \leftarrow$ STRASSEN$(D, G - E, n/2)$
8 $P_5 \leftarrow$ STRASSEN$(A + D, E + H, n/2)$
9 $P_6 \leftarrow$ STRASSEN$(B - D, G + H, n/2)$
10 $P_7 \leftarrow$ STRASSEN$(A - C, E + F, n/2)$
11 $R \leftarrow P_5 + P_4 - P_2 + P_6$
12 $S \leftarrow P_1 + P_2$
13 $T \leftarrow P_3 + P_4$
14 $U \leftarrow P_5 + P_1 - P_3 - P_7$
15 **devolva** $P \leftarrow$ CONSTRÓI-MAT(R, S, T, U)

Consumo de tempo

linha		todas as execuções da linha
1	$=$	$\Theta(1)$
2-3	$=$	$\Theta(n^2)$
4-10	$=$	$7, T(n/2) + \Theta(n^2)$
11-14	$=$	$\Theta(n^2)$
15	$=$	$\Theta(n^2)$
total	$=$	$7\,T(n/2) + \Theta(4n^2 + 1)$

Consumo de tempo

As dicas no nosso estudo de recorrências sugeri que a solução da recorrência

$$
\begin{aligned}
T(1) &= \Theta(1) \\
T(n) &= 7\,T(n/2) + \Theta(n^2) \quad \text{para } n = 2, 3, 4, \ldots
\end{aligned}
$$

está na mesma classe Θ que a solução de

$$
\begin{aligned}
T'(1) &= 1 \\
T'(n) &= 7T'(n/2) + n^2 \quad \text{para } n = 2, 2^2, 2^3, \ldots
\end{aligned}
$$

n	1	2	4	8	16	32	64	128	256
$T'(n)$	1	11	93	715	5261	37851	269053	1899755	13363821

Solução assintótica da recorrência

Considere a recorrência

$$\begin{aligned} R(1) &= 1 \\ R(n) &= 7\,R(\lceil \tfrac{n}{2} \rceil) + n^2 \quad \text{para } n = 2, 3, 4, \ldots \end{aligned}$$

Verifique por indução que $R(n) \leq 19(n-1)^{\lg 7} - 2n^2$ para $n = 2, 3, 4 \ldots$.

$2{,}80 < \lg 7 < 2{,}81$

n	1	2	3	4	5	6	7	8
$R(n)$	1	11	86	93	627	638	700	715
$19\,(n-1)^{\lg 7} - 2n^2$	-1	11	115	327	881	1657	2790	4337

Teorema Bambambã

Teorema Mestre Simplificado:

Suponha

$$T(n) = a\,T(n/b) + c\,n^k$$

para algum $a \geq 1$ e $b > 1$ e onde n/b significa $\lceil n/b \rceil$ ou $\lfloor n/b \rfloor$.
Então, em geral,

se $a > b^k$ então $T(n) = \Theta(n^{\log_b a})$
se $a = b^k$ então $T(n) = \Theta(n^k \lg n)$
se $a < b^k$ então $T(n) = \Theta(n^k)$

Conclusões

$R(n)$ é $\Theta(n^{\lg 7})$.

$T(n)$ é $\Theta(n^{\lg 7})$.

O consumo de tempo do algoritmo STRASSEN é $\Theta(n^{\lg 7})$ ($2{,}80 < \lg 7 < 2{,}81$).

Mais conclusões

Consumo de tempo de algoritmos para multiplicação de matrizes:

Ensino fundamental	$\Theta(n^3)$
Strassen	$\Theta(n^{\lg 7}) = \mathrm{O}(n^{2.81})$
...	...
Coppersmith e Winograd	$\mathrm{O}(n^{2.38})$

Exercícios

Exercício 11.A

O algoritmo abaixo promete rearranjar $A[p \mathinner{.\,.} r]$, com $p \leq r$, em ordem crescente.

ORDENA-POR-INS (A, p, r)

1 se $p < r$
2 então ORDENA-POR-INS $(A, p, r-1)$
3 $chave \leftarrow A[r]$
4 $i \leftarrow r - 1$
5 enquanto $i \geq p$ e $A[i] > chave$ faça
6 $A[i+1] \leftarrow A[i]$
7 $i \leftarrow i - 1$
8 $A[i+1] \leftarrow chave$

O algoritmo está correto? Escreva uma recorrência que expresse o consumo de tempo. Qual o consumo de tempo do algoritmo?

Exercício 11.B

Que acontece se trocarmos $\lfloor (p+r)/2 \rfloor$ por $\lceil (p+r)/2 \rceil$ na linha 2 do MERGE-SORT?

Mais exercicios

Exercício 11.C

Quantas vezes a comparação "$A[r] \neq 0$" é executada? Defina esse número por meio de um recorrência.

```
LIMPA (A, p, r)
1   se p = r
2       então devolva r
3       senão q ← LIMPA (A, p, r − 1)
4           se A[r] ≠ 0
5               então q ← q + 1
6                     A[q] ← A[r]
7           devolva q
```

Dê uma fórmula exata para a função definida pela recorrência. Em que classe Θ está a função definida pela recorrência?

AULA 8

Heapsort

CLRS 6

(veja tb CLRS B.5.3)

Merge sort ilustrou uma técnica (ou paradigma, ou estratégia) de concepção de algoritmos eficientes: a divisão e conquista.

Heapsort ilustra o uso de estruturas de dados no projeto de algoritmos eficientes.

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

Entra:

Ordenação

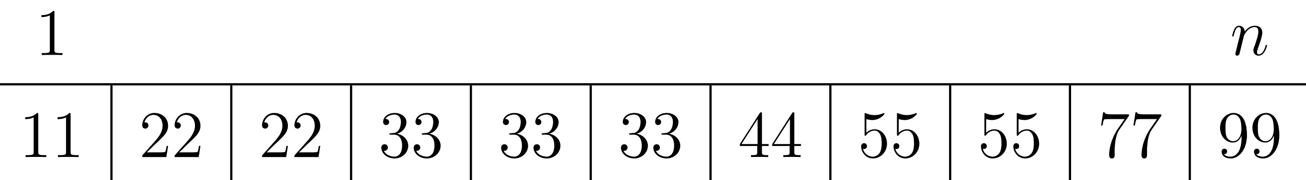
$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

Entra:

1										n
33	55	33	44	33	22	11	99	22	55	77

Sai:



1										n
11	22	22	33	33	33	44	55	55	77	99

Ordenação por seleção

$m = 5$

1				max						n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

1	j		max							n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

1	j		max							n
38	50	20	44	10	50	55	60	75	85	99

j	max									n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

$m = 5$

1			j	max						n
38	50	20	44	10	50	55	60	75	85	99

1		j	max							n
38	50	20	44	10	50	55	60	75	85	99

1	j		max							n
38	50	20	44	10	50	55	60	75	85	99

j	max									n
38	50	20	44	10	50	55	60	75	85	99

1	max									n
38	50	20	44	10	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
38	10	20	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
20	10	38	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
20	10	38	44	50	50	55	60	75	85	99

1	m									n
20	10	38	44	50	50	55	60	75	85	99

Ordenação por seleção

1			m							n
38	10	20	44	50	50	55	60	75	85	99

1		m								n
20	10	38	44	50	50	55	60	75	85	99

1	m									n
10	20	38	44	50	50	55	60	75	85	99

Ordenação por seleção

1										n
			m							
38	10	20	44	50	50	55	60	75	85	99

1										n
		m								
20	10	38	44	50	50	55	60	75	85	99

1										n
	m									
10	20	38	44	50	50	55	60	75	85	99

1										n
10	20	38	44	50	50	55	60	75	85	99

Ordenação por seleção

Algoritmo rearranja $A[1\,..\,n]$ em ordem crescente

ORDENA-POR-SELEÇÃO (A, n)

1 **para** $m \leftarrow n$ **decrescendo até** 2 **faça**

2 $max \leftarrow m$

3 **para** $j \leftarrow m - 1$ **decrescendo até** 1 **faça**

4 **se** $A[max] < A[j]$

5 **então** $max \leftarrow j$

6 $A[m] \leftrightarrow A[max]$

Invariantes

Relação invariante chave:

(i0) na linha 1 vale que: $A[m \mathinner{.\,.} n]$ é crescente.

1				m						n
38	50	20	44	10	50	55	60	75	85	99

Invariantes

Relação invariante chave:

(i0) na linha 1 vale que: $A[m \mathinner{.\,.} n]$ é crescente.

1				m						n
38	50	20	44	10	50	55	60	75	85	99

Supondo que a invariante vale. Correção do algoritmo é evidente.

No início da última iteração das linhas 1–6 tem-se que $m = 1$. Da invariante conclui-se que $A[1 \mathinner{.\,.} n]$ é crescente.

Mais invariantes

Na linha 1 vale que:

(i1) $A[1\,..\,m] \leq A[m+1]$;

Na linha 3 vale que:

(i2) $A[j+1\,..\,m] \leq A[max]$

1	j		max		m					n
38	50	20	44	10	25	55	60	75	85	99

Mais invariantes

Na linha 1 vale que:

(i1) $A[1\,.\,.\,m] \leq A[m+1]$;

Na linha 3 vale que:

(i2) $A[j+1\,.\,.\,m] \leq A[max]$

1	j		max		m					n
38	50	20	44	10	25	55	60	75	85	99

invariantes (i1),(i2)
$+$ condição de parada do **para** da linha 3
$+$ troca linha 6 $\Rightarrow$ validade (i0)

Verifique!

Consumo de tempo

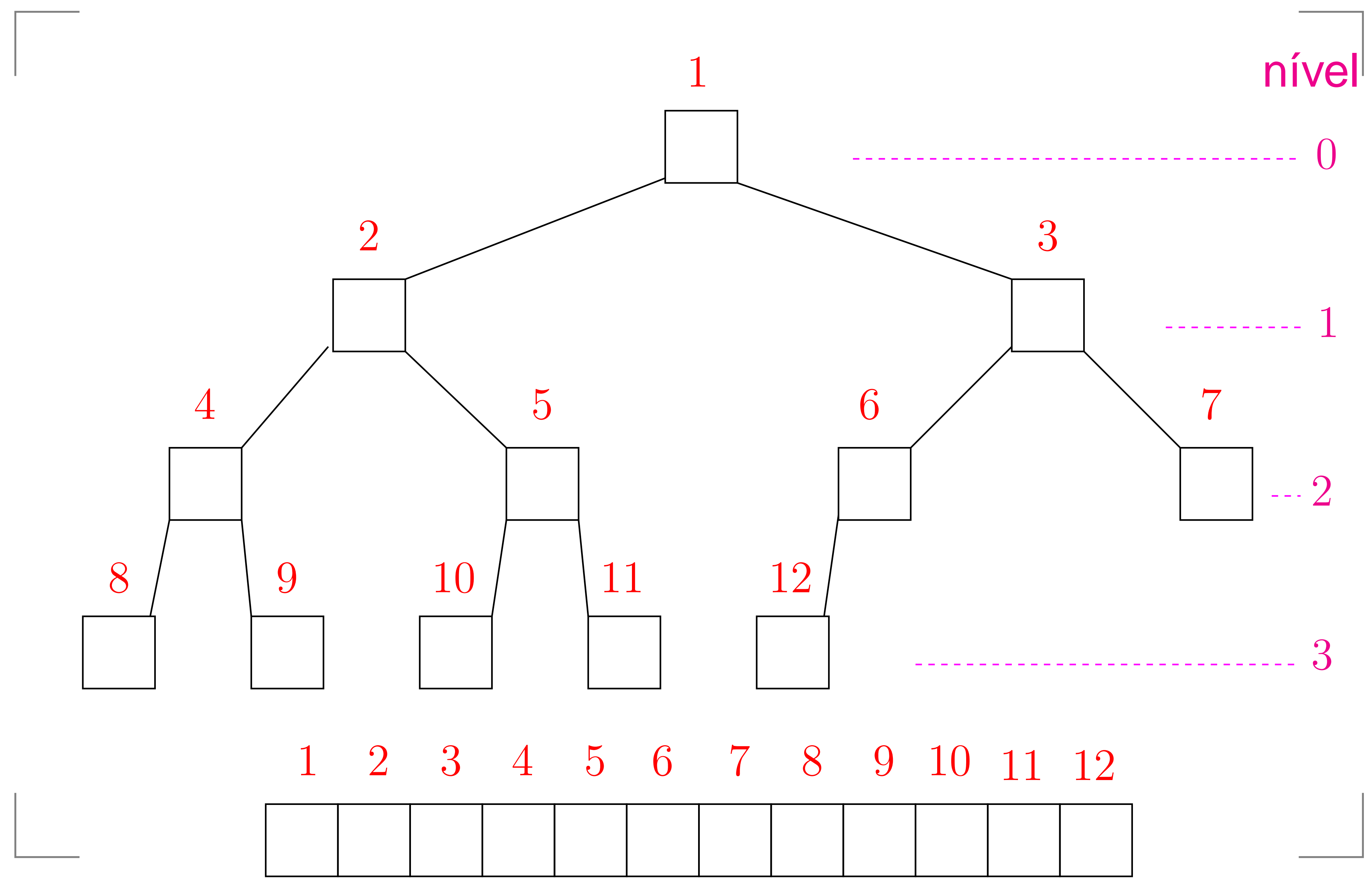
linha	todas as execuções da linha
1	$= \Theta(n)$
2	$= \Theta(n)$
3	$= \Theta(n^2)$
4	$= \Theta(n^2)$
5	$= \mathrm{O}(n^2)$
6	$= \Theta(n)$

total $= \Theta(2n^2 + 3n) + \mathrm{O}(n^2) = \Theta(n^2)$

Conclusão

O consumo de tempo do algoritmo ORDENA-POR-SELEÇÃO é $\Theta(n^2)$.

Representação de árvores em vetores



Pais e filhos

$A[1\,..\,m]$ é um vetor representando uma árvore.
Diremos que para qualquer índice ou nó i,

- $\lfloor i/2 \rfloor$ é o pai de i;
- $2\,i$ é o filho esquerdo de i;
- $2\,i + 1$ é o filho direito.

O nó 1 não tem pai e é chamado de raiz.

Um nó i só tem filho esquerdo se $2\,i \le m$.

Um nó i só tem filho direito se $2\,i + 1 \le m$.

Um nó i é um folha se não tem filhos, ou seja $2\,i > m$.

Todo nó i é raiz da subárvore formada por

$$A[i, 2i, 2i+1, 4i, 4i+1, 4i+2, 4i+3, 8i, \ldots, 8i+7, \ldots]$$

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

$$2^p, 2^p + 1, 2^p + 2, \ldots, 2^{p+1} - 1.$$

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

$$2^p, 2^p + 1, 2^p + 2, \ldots, 2^{p+1} - 1.$$

O nó i pertence ao nível ???.

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

$$2^p, 2^p + 1, 2^p + 2, \ldots, 2^{p+1} - 1.$$

O nó i pertence ao nível $\lfloor \lg i \rfloor$.

Prova: Se p é o nível do nó i, então

$$\begin{array}{rcccll} 2^p & \leq & i & < & 2^{p+1} & \Rightarrow \\ \lg 2^p & \leq & \lg i & < & \lg 2^{p+1} & \Rightarrow \\ p & \leq & \lg i & < & p+1 & \end{array}$$

Logo, $p = \lfloor \lg i \rfloor$.

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

$$2^p, 2^p + 1, 2^p + 2, \ldots, 2^{p+1} - 1.$$

O nó i pertence ao nível $\lfloor \lg i \rfloor$.

Prova: Se p é o nível do nó i, então

$$\begin{array}{rcccll} 2^p & \leq & i & < & 2^{p+1} & \Rightarrow \\ \lg 2^p & \leq & \lg i & < & \lg 2^{p+1} & \Rightarrow \\ p & \leq & \lg i & < & p+1 & \end{array}$$

Logo, $p = \lfloor \lg i \rfloor$.

Portanto o número total de níveis é ???.

Níveis

Cada nível p, exceto talvez o último, tem exatamente 2^p nós e esses são

$$2^p, 2^p + 1, 2^p + 2, \ldots, 2^{p+1} - 1.$$

O nó i pertence ao nível $\lfloor \lg i \rfloor$.

Prova: Se p é o nível do nó i, então

$$\begin{array}{ccccccc} 2^p & \leq & i & < & 2^{p+1} & \Rightarrow \\ \lg 2^p & \leq & \lg i & < & \lg 2^{p+1} & \Rightarrow \\ p & \leq & \lg i & < & p+1 & \end{array}$$

Logo, $p = \lfloor \lg i \rfloor$.

Portanto, o número total de níveis é $1 + \lfloor \lg m \rfloor$.

Altura

A altura de um nó i é o maior comprimento de um caminho de i a uma folha.

Em outras palavras, a altura de um nó i é o maior comprimento de uma seqüência da forma

$$\langle \text{filho}(i), \text{filho}(\text{filho}(i)), \text{filho}(\text{filho}(\text{filho}(i))), \ldots \rangle$$

onde $\text{filho}(i)$ vale $2i$ ou $2i + 1$.

Os nós que têm altura zero são as folhas.

Altura

A altura de um nó i é o maior comprimento de um caminho de i a uma folha.

Em outras palavras, a altura de um nó i é o maior comprimento de uma seqüência da forma

$$\langle \mathrm{filho}(i), \mathrm{filho}(\mathrm{filho}(i)), \mathrm{filho}(\mathrm{filho}(\mathrm{filho}(i))), \ldots \rangle$$

onde $\mathrm{filho}(i)$ vale $2i$ ou $2i + 1$.

Os nós que têm altura zero são as folhas.

A altura de um nó i é ????.

Exercício 12.A

A altura de um nó i é o comprimento da seqüência

$$\langle 2\,i, 2^2\,i, 2^3\,i, \ldots, 2^h\,i \rangle$$

onde $2^h\,i \le m < 2^{(h+1)}\,i$. Assim,

$$\begin{array}{rcccl} 2^h\,i & \le & m & < & 2^{h+1}i & \Rightarrow \\ 2^h & \le & m/i & < & 2^{h+1} & \Rightarrow \\ \lg 2^h & \le & \lg(m/i) & < & \lg 2^{h+1} & \Rightarrow \\ h & \le & \lg(m/i) & < & h+1 & \end{array}$$

Portanto, a altura de i é $h = \lfloor \lg(m/i) \rfloor$.

Exercício 12.B

Mostre que $A[1\,..\,m]$ tem no máximo $\lceil m/2^{h+1}\rceil$ nós com altura h.

Exemplo: N_h = número de nós à altura h

m	$\lfloor m/2^{0+1}\rfloor$	N_0	$\lceil m/2^{0+1}\rceil$	$\lfloor m/2^{1+1}\rfloor$	N_1	$\lceil m/2^{1+1}\rceil$
16	8	8	8	4	4	4
17	8	9	9	4	4	5
18	9	9	9	4	5	5
19	9	10	10	4	5	5
20	10	10	10	5	5	5
21	10	11	11	5	5	6
22	11	11	11	5	6	6
23	11	12	12	5	6	6
24	12	12	12	6	6	6

Solução

Prova: Exercício 12.B

Um nó i tem altura $h = \lfloor \lg(m/i) \rfloor$. Logo,

$$
\begin{array}{rcccl}
2^h & \leq & m/i & < & 2^{h+1} \\
m/2^{h+1} & < & i & \leq & m/2^h \\
\lfloor m/2^{h+1} \rfloor + 1 & \leq & i & \leq & \lfloor m/2^h \rfloor
\end{array}
$$

O número de possíveis valores para i é

$$
\begin{array}{cl}
= & \lfloor m/2^h \rfloor - \lfloor m/2^{h+1} \rfloor \\
= & \lfloor m/2^h \rfloor - \lfloor \lfloor m/2^h \rfloor /2 \rfloor \\
= & \lceil \lfloor m/2^h \rfloor /2 \rceil \\
\leq & \lceil \lceil m/2^h \rceil /2 \rceil = \lceil m/2^{h+1} \rceil .
\end{array}
$$

Resumão

Considere uma árvore representada em um vetor $A[1\,.\,.\,m]$.

filho esquerdo de i:	$2\,i$
filho direito de i:	$2\,i+1$
pai de i:	$\lfloor i/2 \rfloor$
nível da raiz:	0
nível de i:	$\lfloor \lg i \rfloor$
altura da raiz:	$\lfloor \lg m \rfloor$
altura da árvore:	$\lfloor \lg m \rfloor$
altura de i:	$\lfloor \lg(m/i) \rfloor$
altura de uma folha:	0
total de nós de altura h	$\leq \lceil m/2^{h+1} \rceil$ (Exercício 12.B)

Heap

Um vetor $A[1\,.\,.\,m]$ é um max-heap se

$$A[\lfloor i/2 \rfloor] \geq A[i]$$

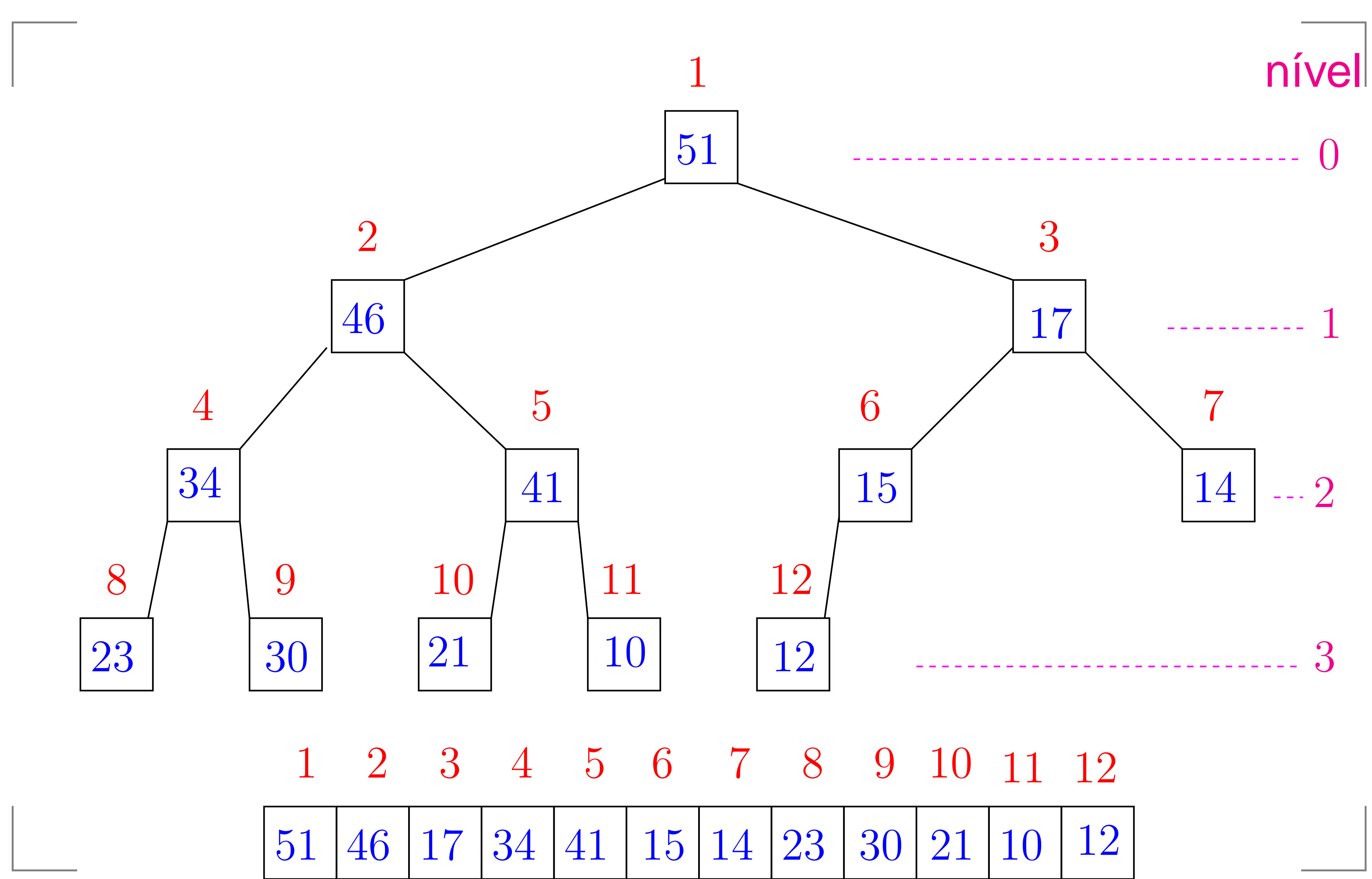
para todo $i = 2, 3, \ldots, m$.

De uma forma mais geral, $A[j\,.\,.\,m]$ é um max-heap se

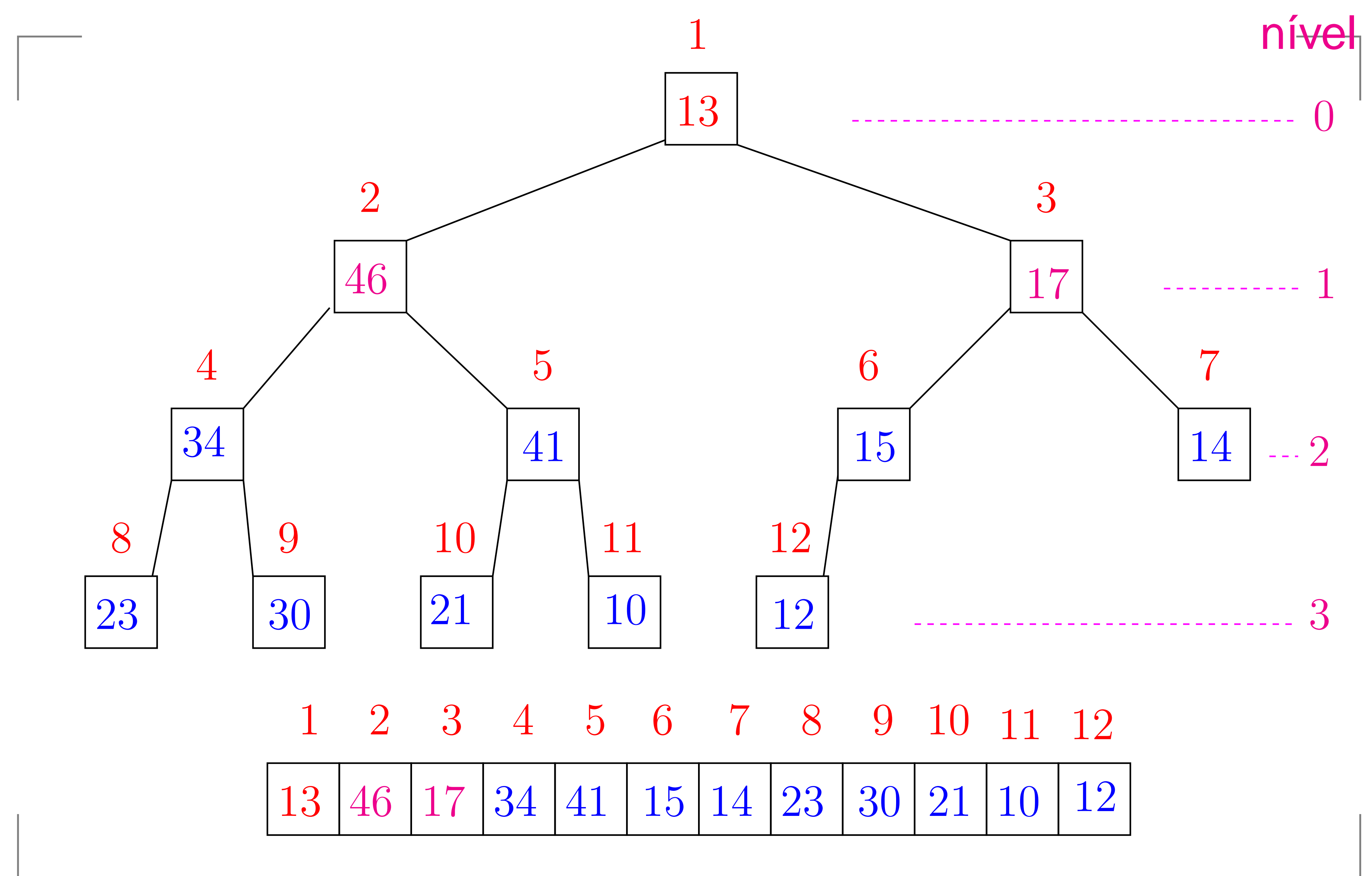
$$A[\lfloor i/2 \rfloor] \geq A[i]$$

para todo $i = 2j, 2j+1, 4j, \ldots, 4j+3, 8j, \ldots, 8j+7, \ldots$.
Neste caso também diremos que a subárvore com raiz j é um max-heap.

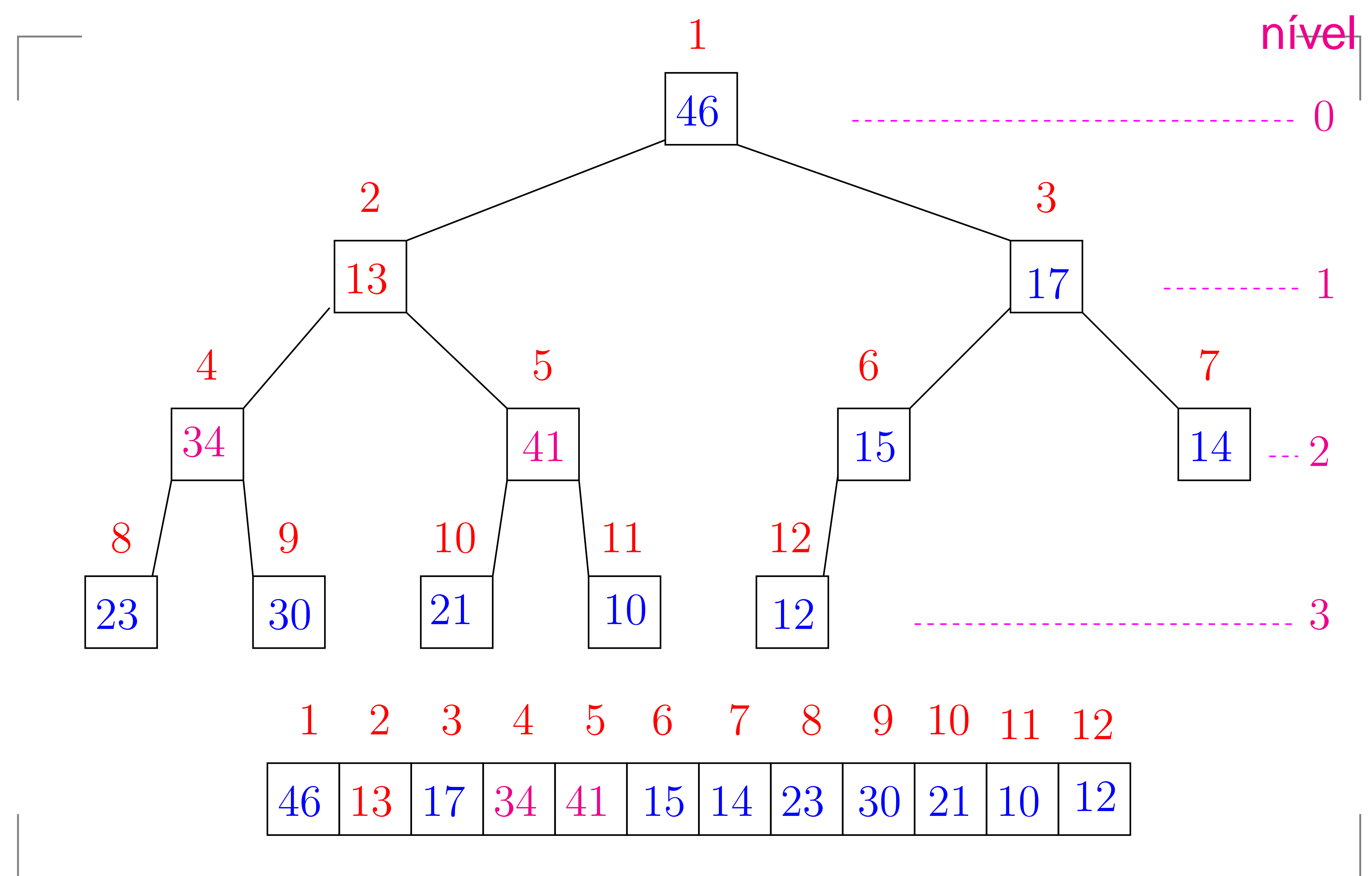
Max-heap



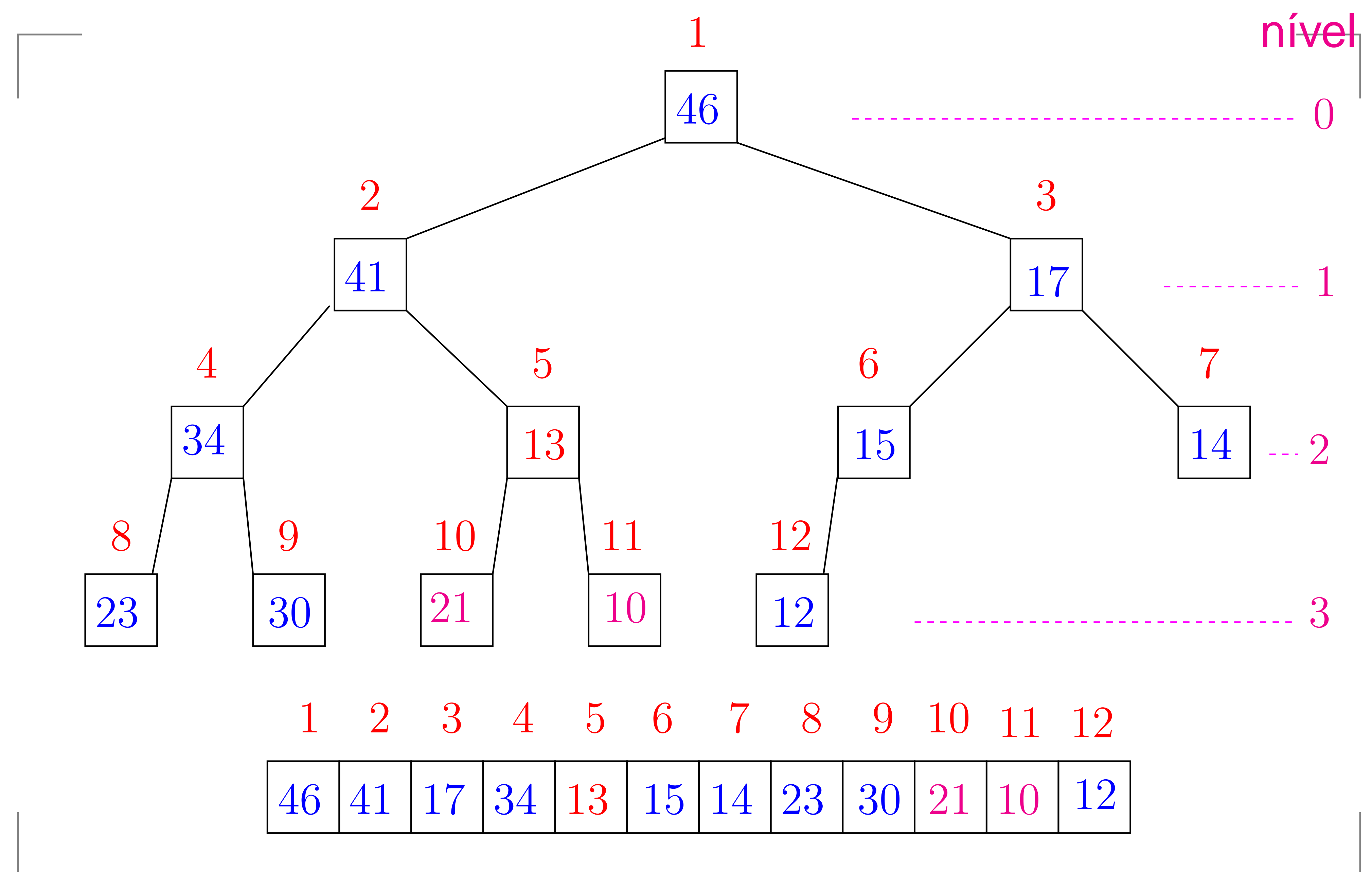
Rotina básica de manipulação de max-heap



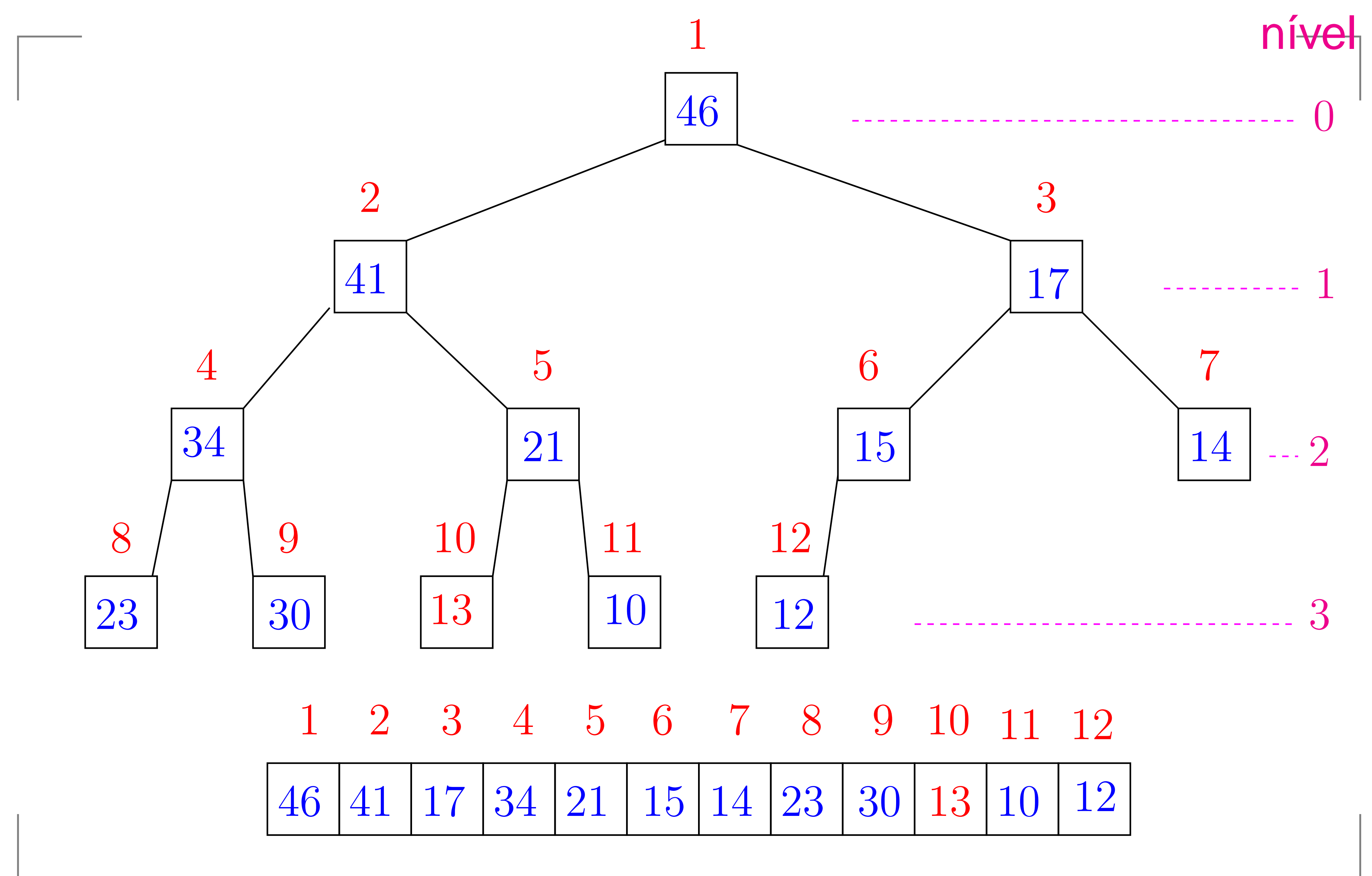
Rotina básica de manipulação de max-heap



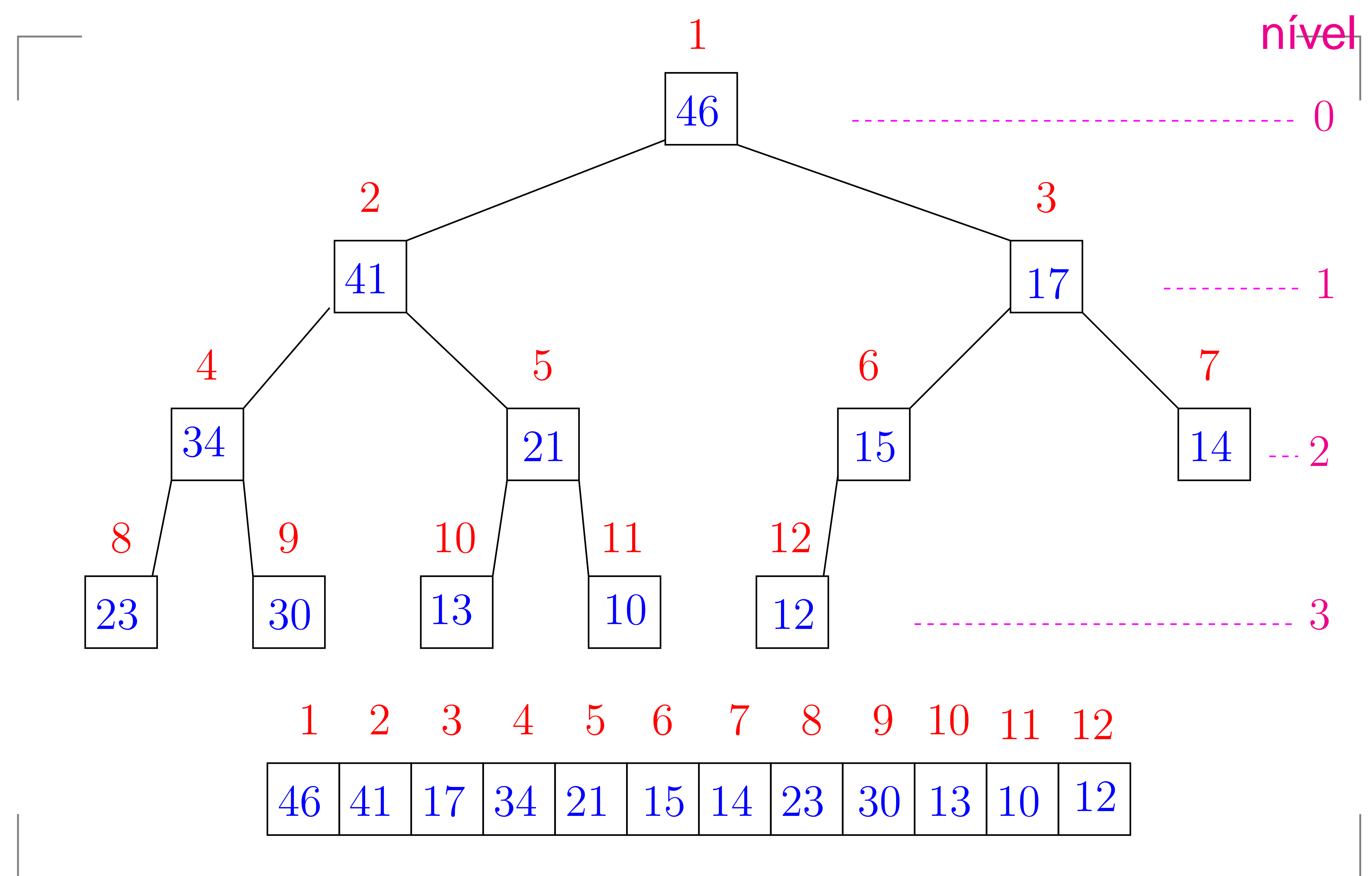
Rotina básica de manipulação de max-heap



Rotina básica de manipulação de max-heap



Rotina básica de manipulação de max-heap



Rotina básica de manipulação de max-heap

Recebe $A[1\,..\,m]$ e $i \geq 1$ tais que subárvores com raiz $2i$ e $2i+1$ são max-heaps e rearranja A de modo que subárvore com raiz i seja max-heap.

MAX-HEAPIFY (A, m, i)
1 $e \leftarrow 2i$
2 $d \leftarrow 2i+1$
3 **se** $e \leq m$ e $A[e] > A[i]$
4 **então** $maior \leftarrow e$
5 **senão** $maior \leftarrow i$
6 **se** $d \leq m$ e $A[d] > A[maior]$
7 **então** $maior \leftarrow d$
8 **se** $maior \neq i$
9 **então** $A[i] \leftrightarrow A[maior]$
10 MAX-HEAPIFY $(A, m, maior)$

Consumo de tempo

h := altura de i = $\lfloor \lg \frac{m}{i} \rfloor$
$T(h)$:= consumo de tempo no pior caso

linha		todas as execuções da linha
1-3	$=$	$3\,\Theta(1)$
4-5	$=$	$2\,\mathrm{O}(1)$
6	$=$	$\Theta(1)$
7	$=$	$\mathrm{O}(1)$
8	$=$	$\Theta(1)$
9	$=$	$\mathrm{O}(1)$
10	$\leq$	$T(h-1)$
total	$\leq$	$T(h-1) + \Theta(5) + \mathrm{O}(2)$

Consumo de tempo

h := altura de i = $\lfloor \lg \frac{m}{i} \rfloor$
$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

$$T(h) \leq T(h-1) + \Theta(1),$$

pois altura de $maior$ é $h-1$.

Consumo de tempo

h := altura de i = $\lfloor \lg \frac{m}{i} \rfloor$
$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

$$T(h) \leq T(h-1) + \Theta(1),$$

pois altura de $maior$ é $h-1$.

Solução assintótica: $T(h)$ é ???.

Consumo de tempo

h := altura de i = $\lfloor \lg \frac{m}{i} \rfloor$

$T(h)$:= consumo de tempo no pior caso

Recorrência associada:

$$T(h) \leq T(h-1) + \Theta(1),$$

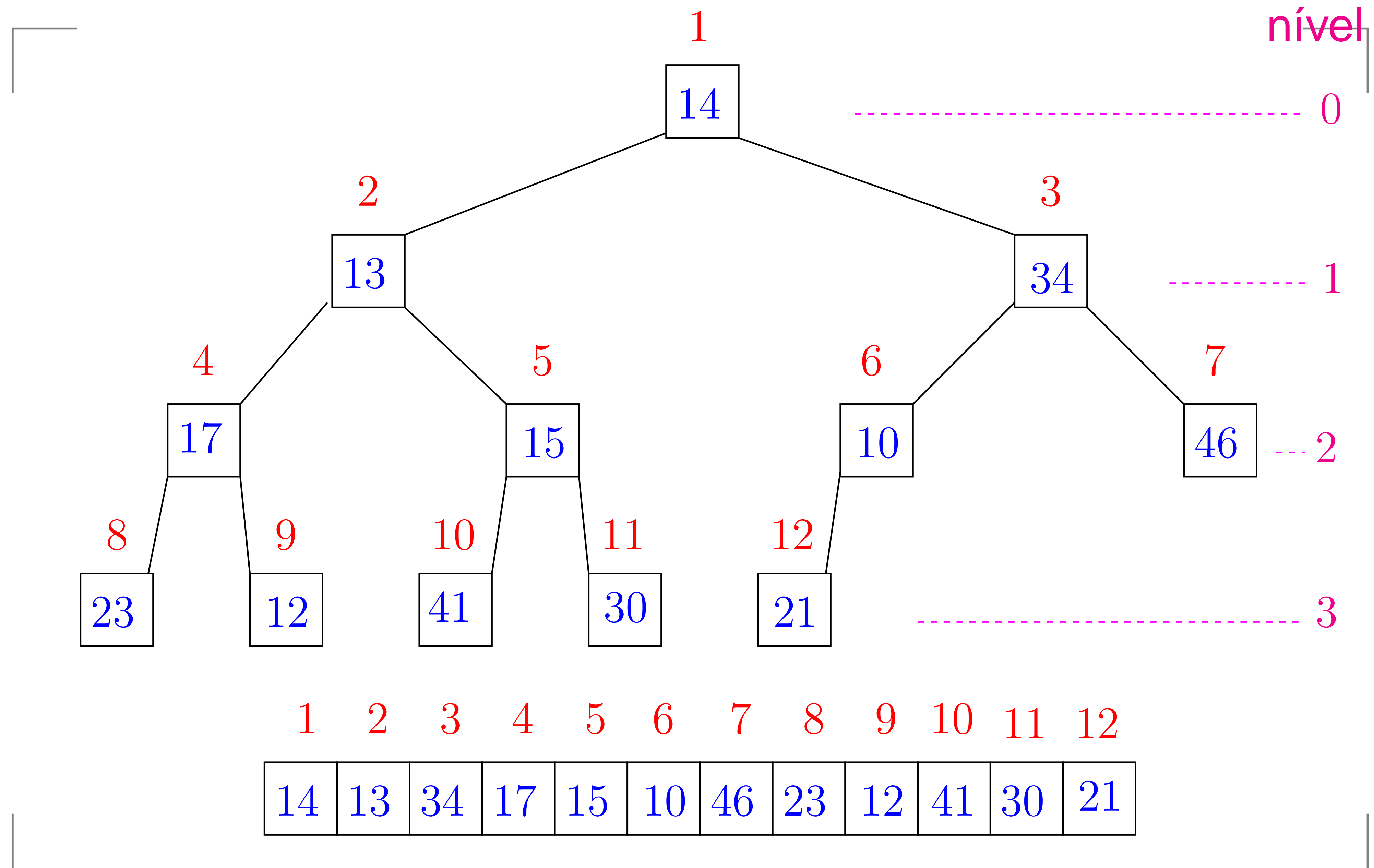
pois altura de $maior$ é $h-1$.

Solução assintótica: $T(h)$ é $\mathrm{O}(h)$.

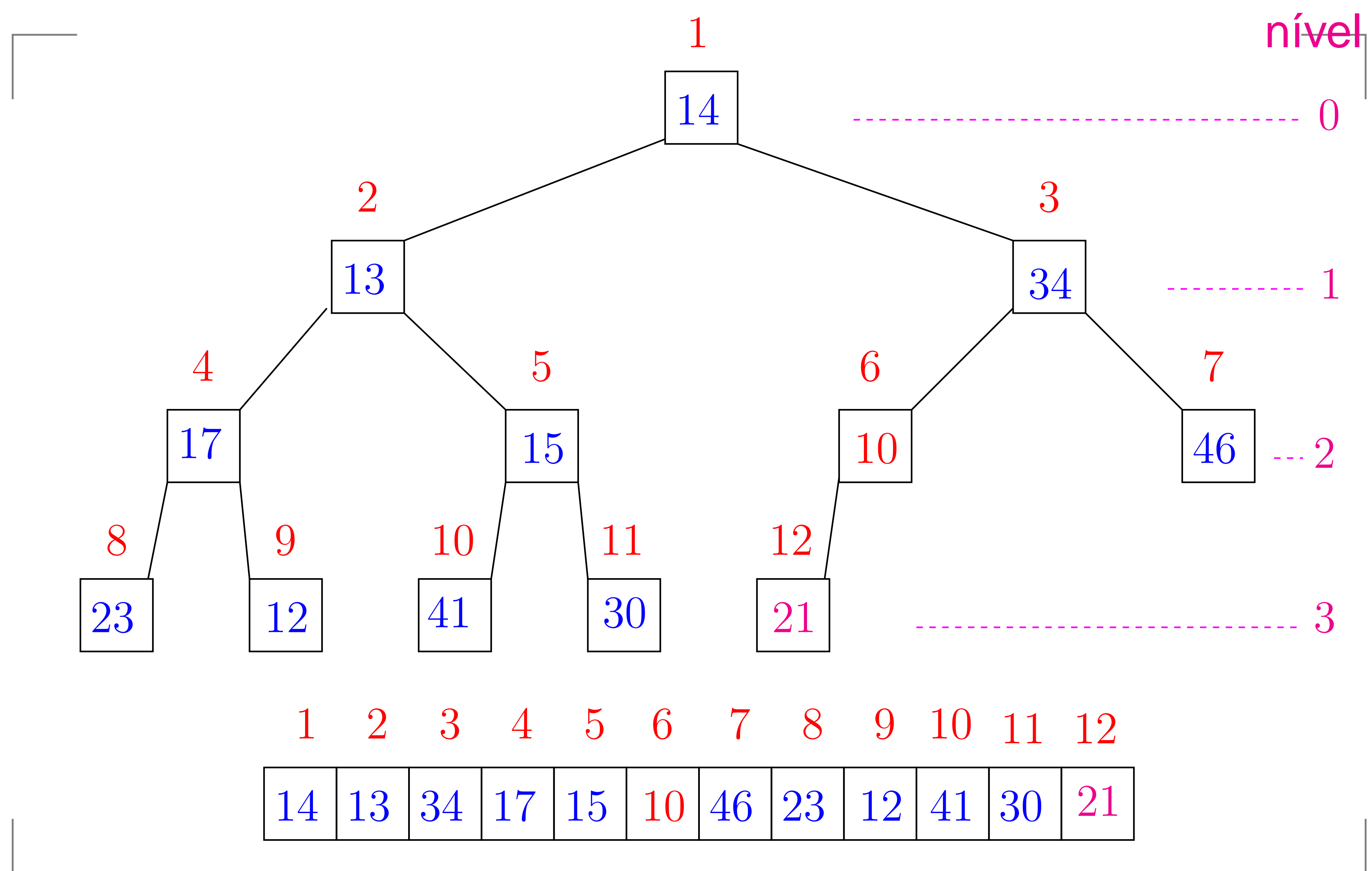
Como $h \leq \lg m$, podemos dizer que:

> O consumo de tempo do algoritmo MAX-HEAPIFY é $\mathrm{O}(\lg m)$ (ou melhor ainda, $\mathrm{O}(\lg \frac{m}{i})$).

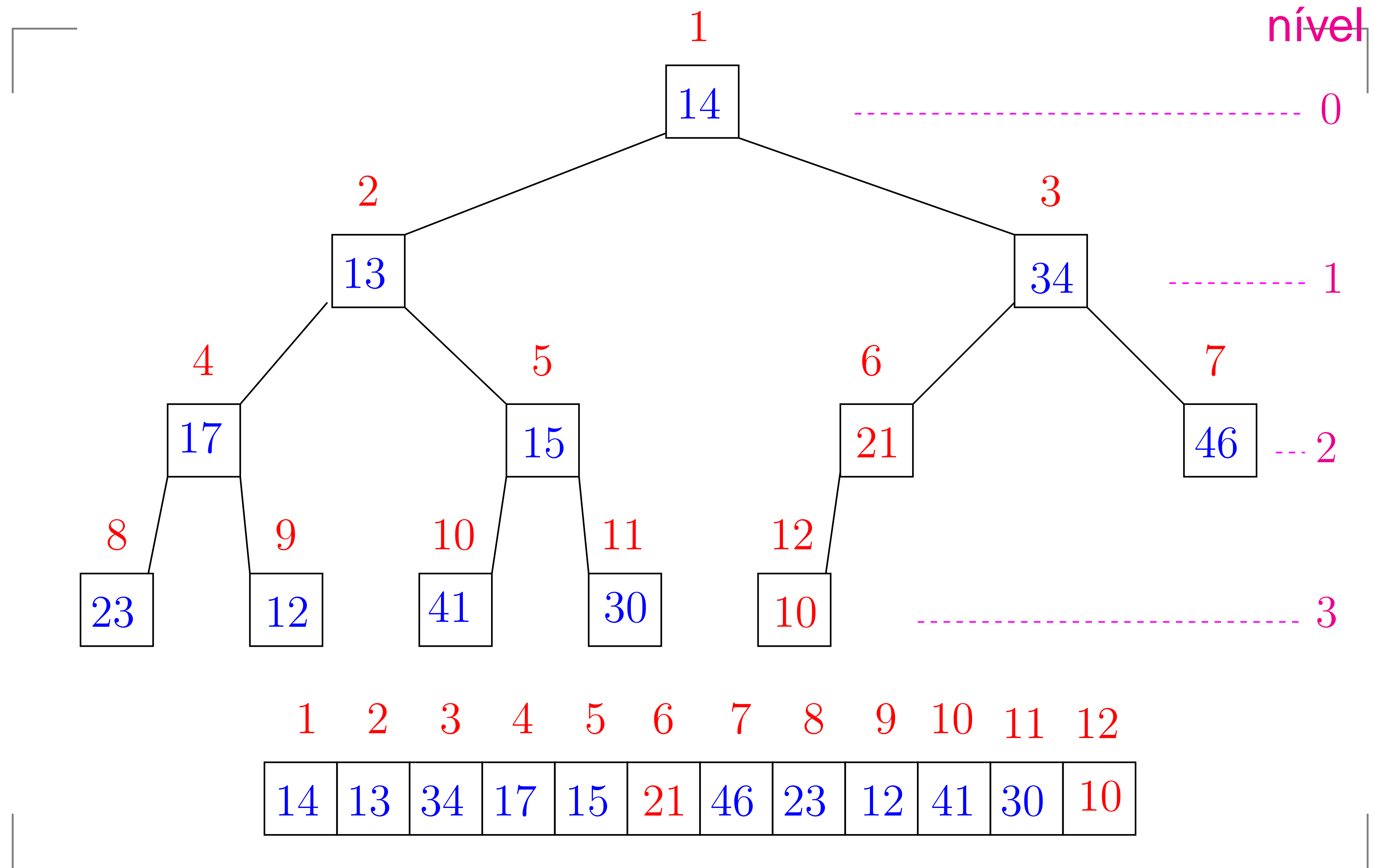
Construção de um max-heap



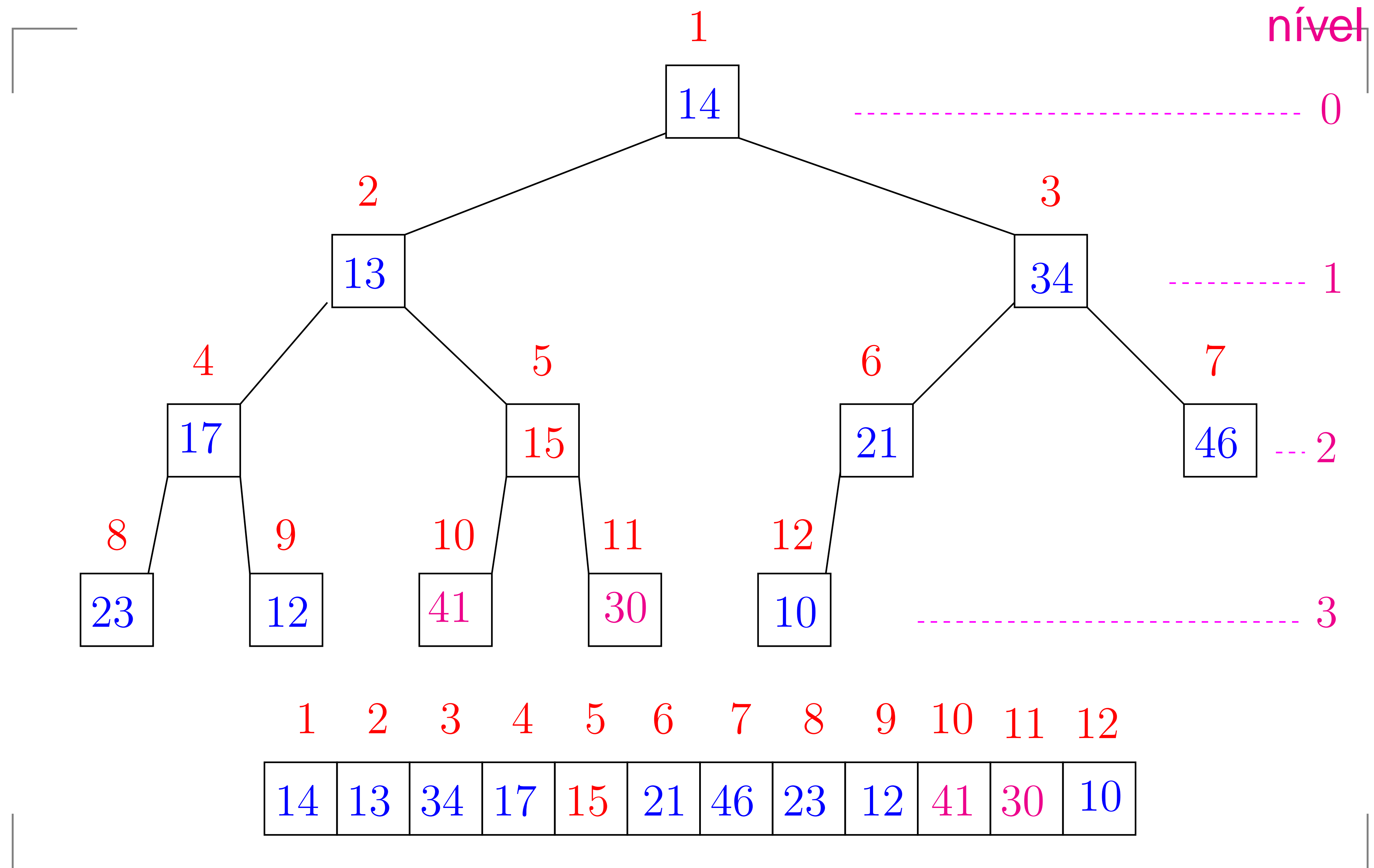
Construção de um max-heap



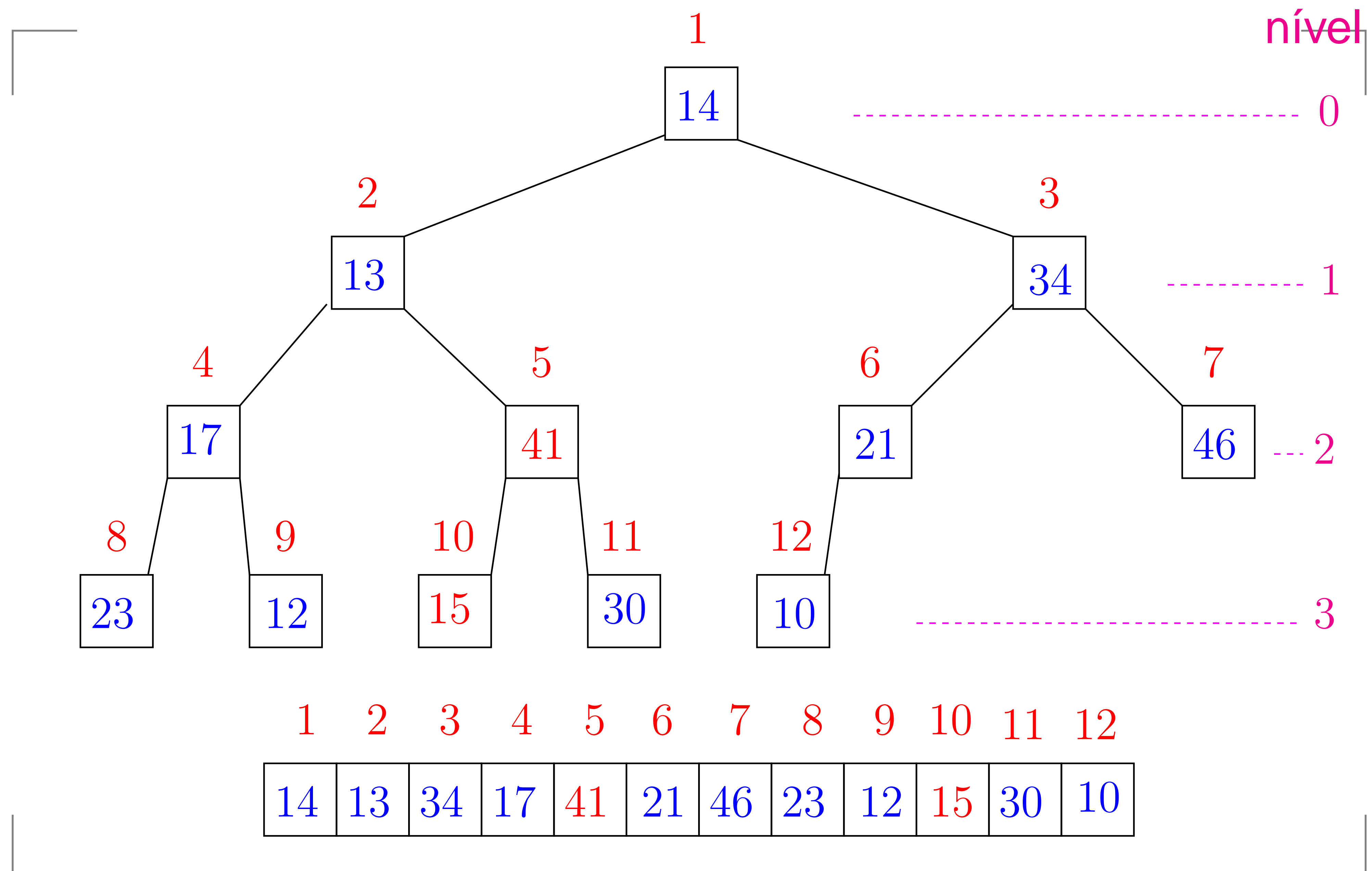
Construção de um max-heap



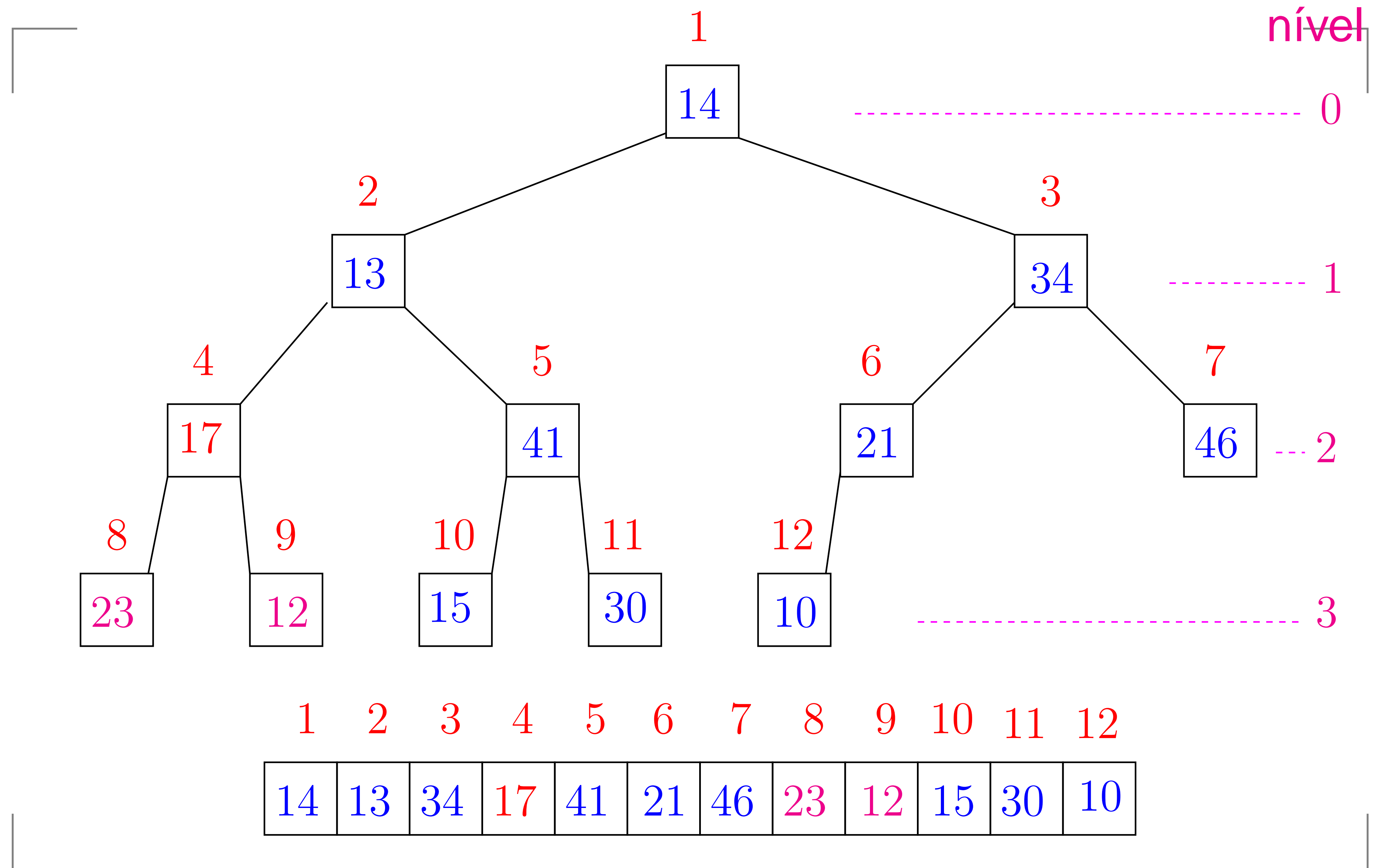
Construção de um max-heap



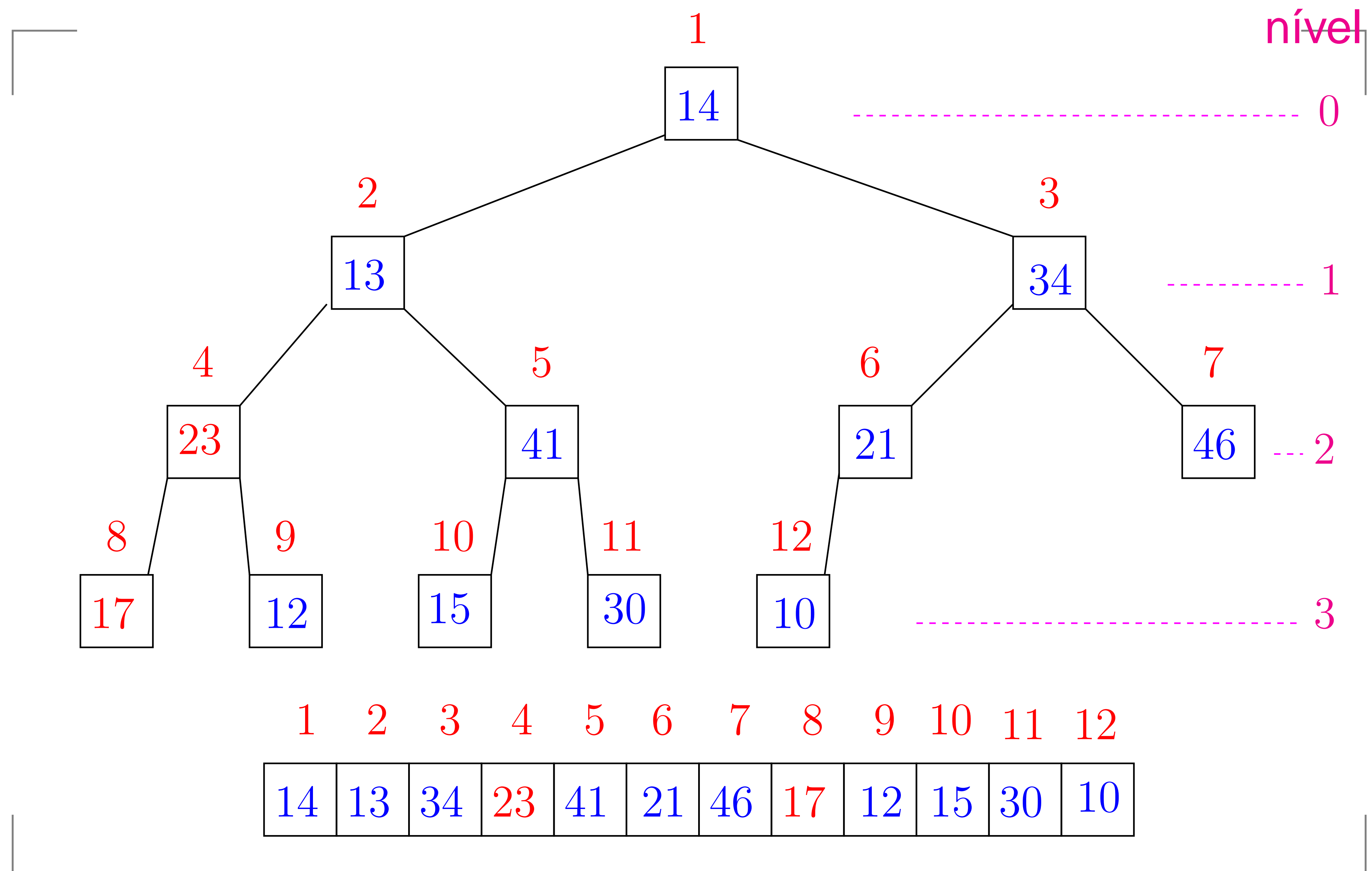
Construção de um max-heap



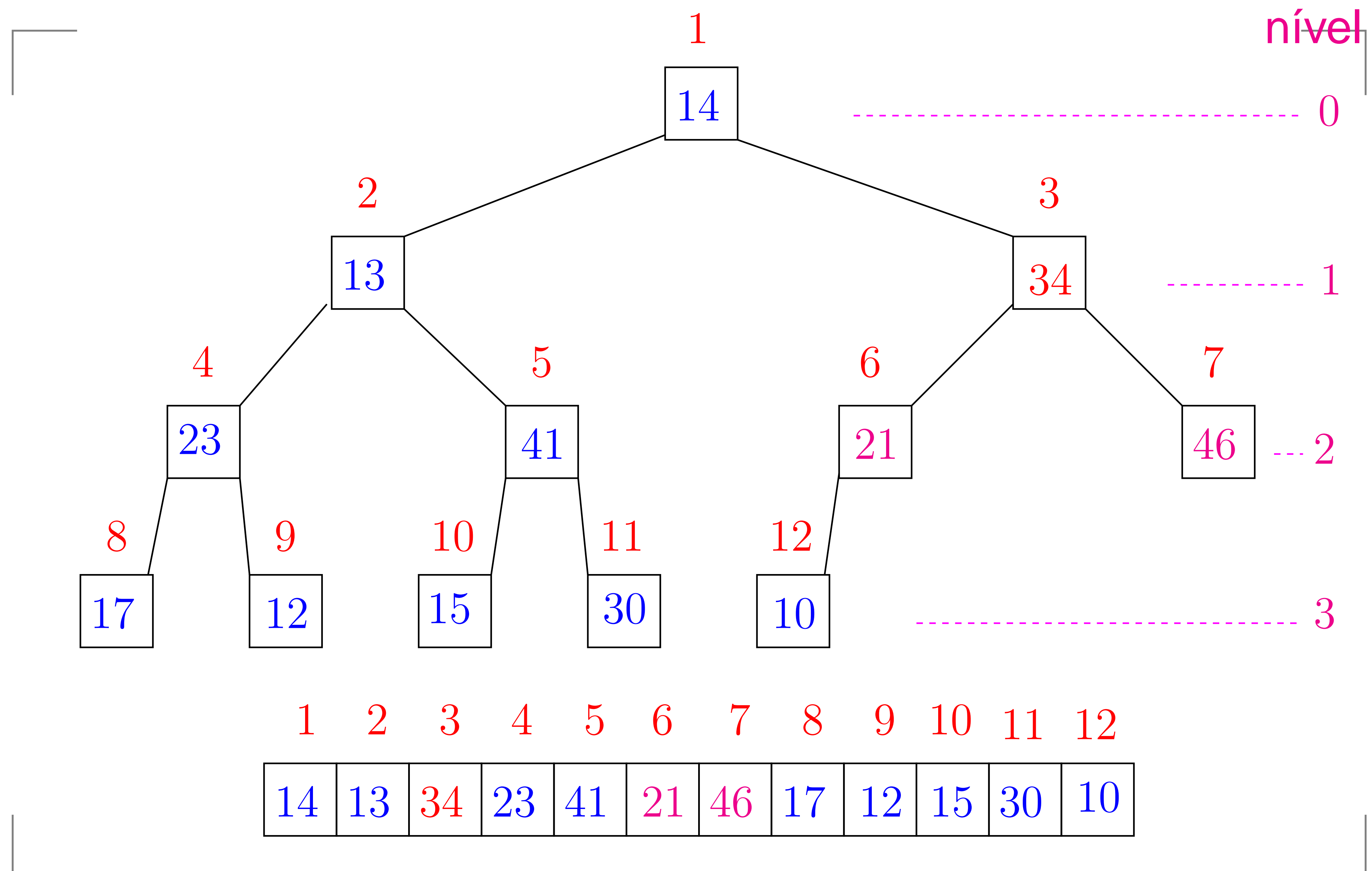
Construção de um max-heap



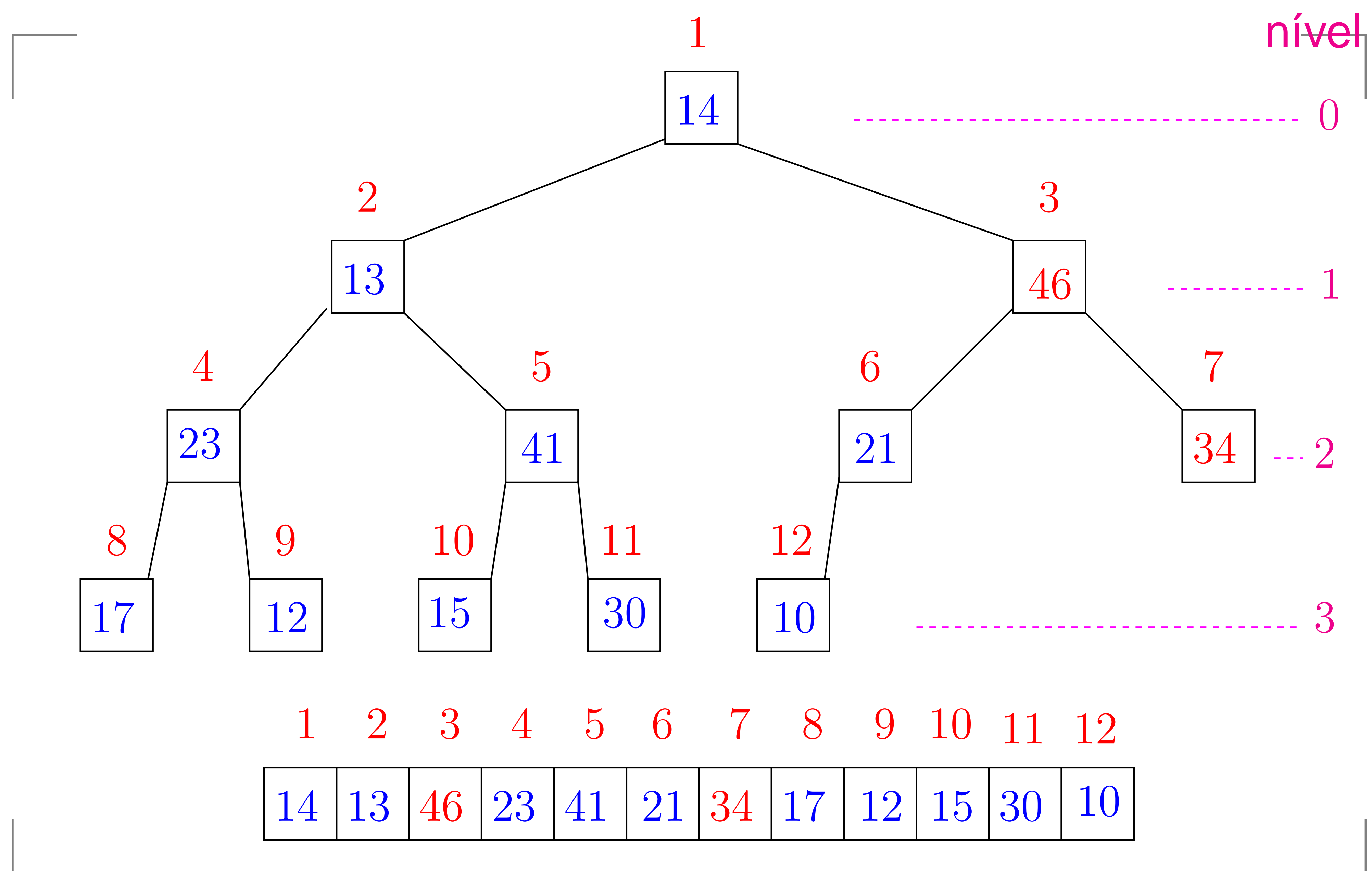
Construção de um max-heap



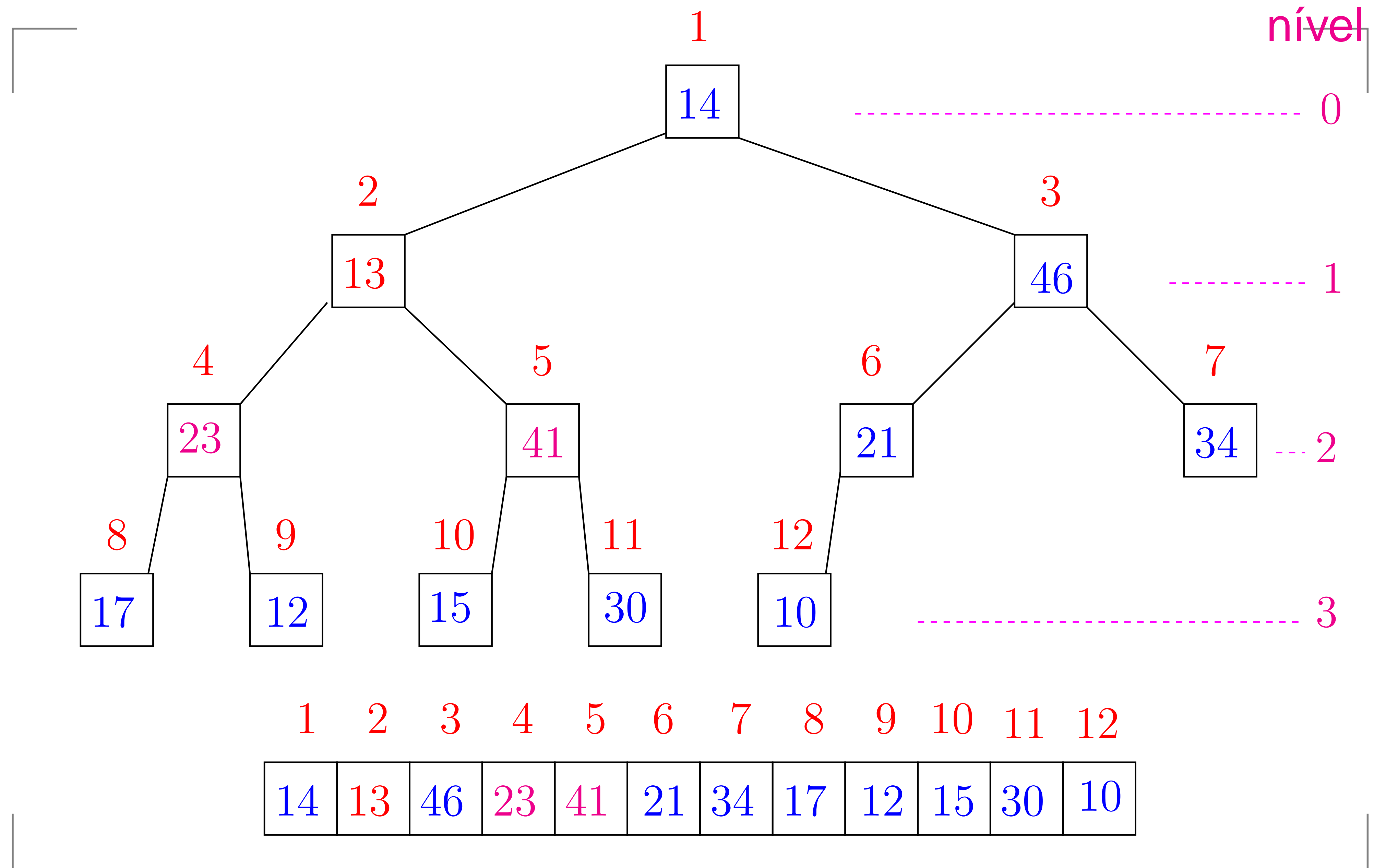
Construção de um max-heap



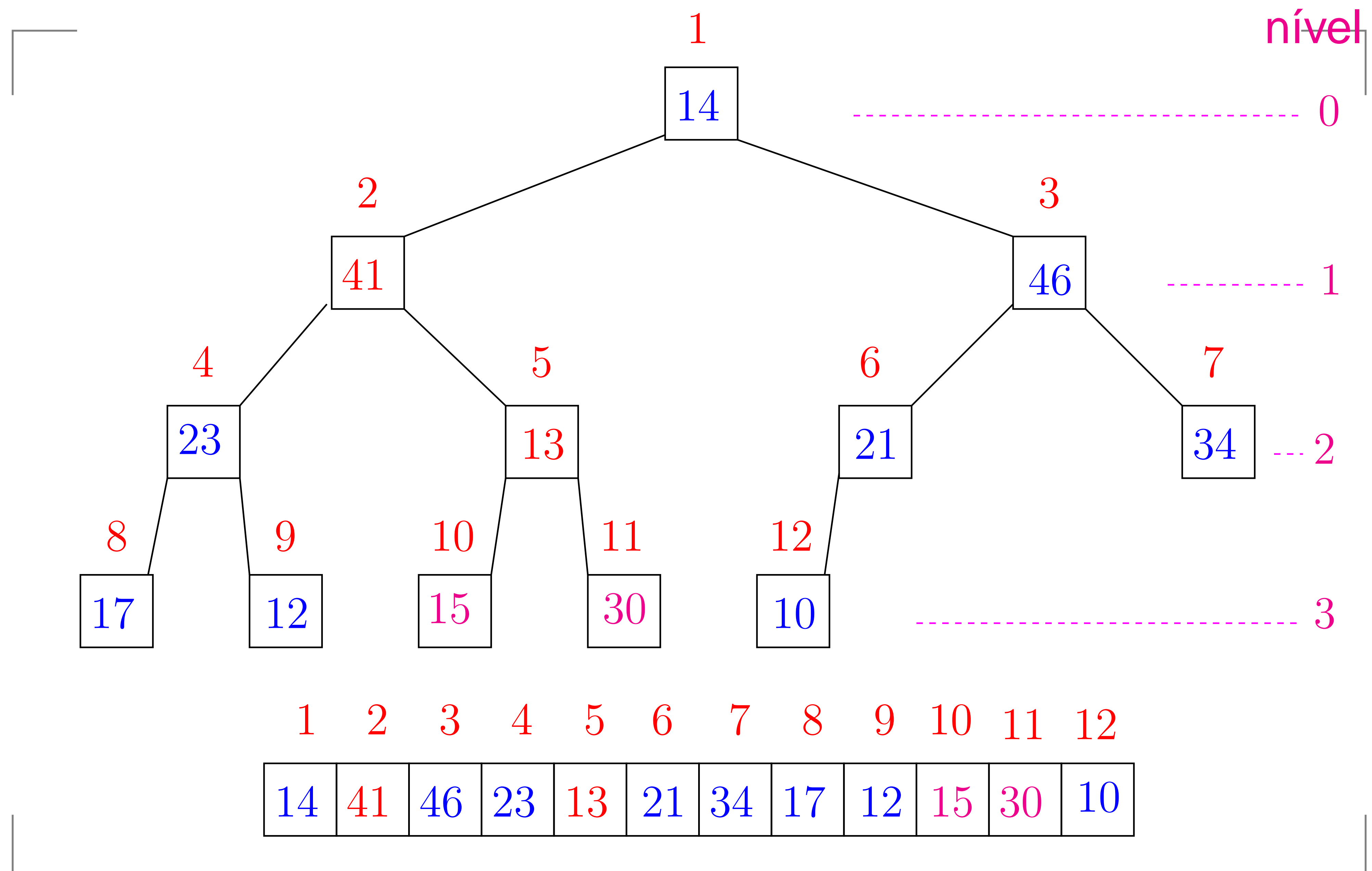
Construção de um max-heap



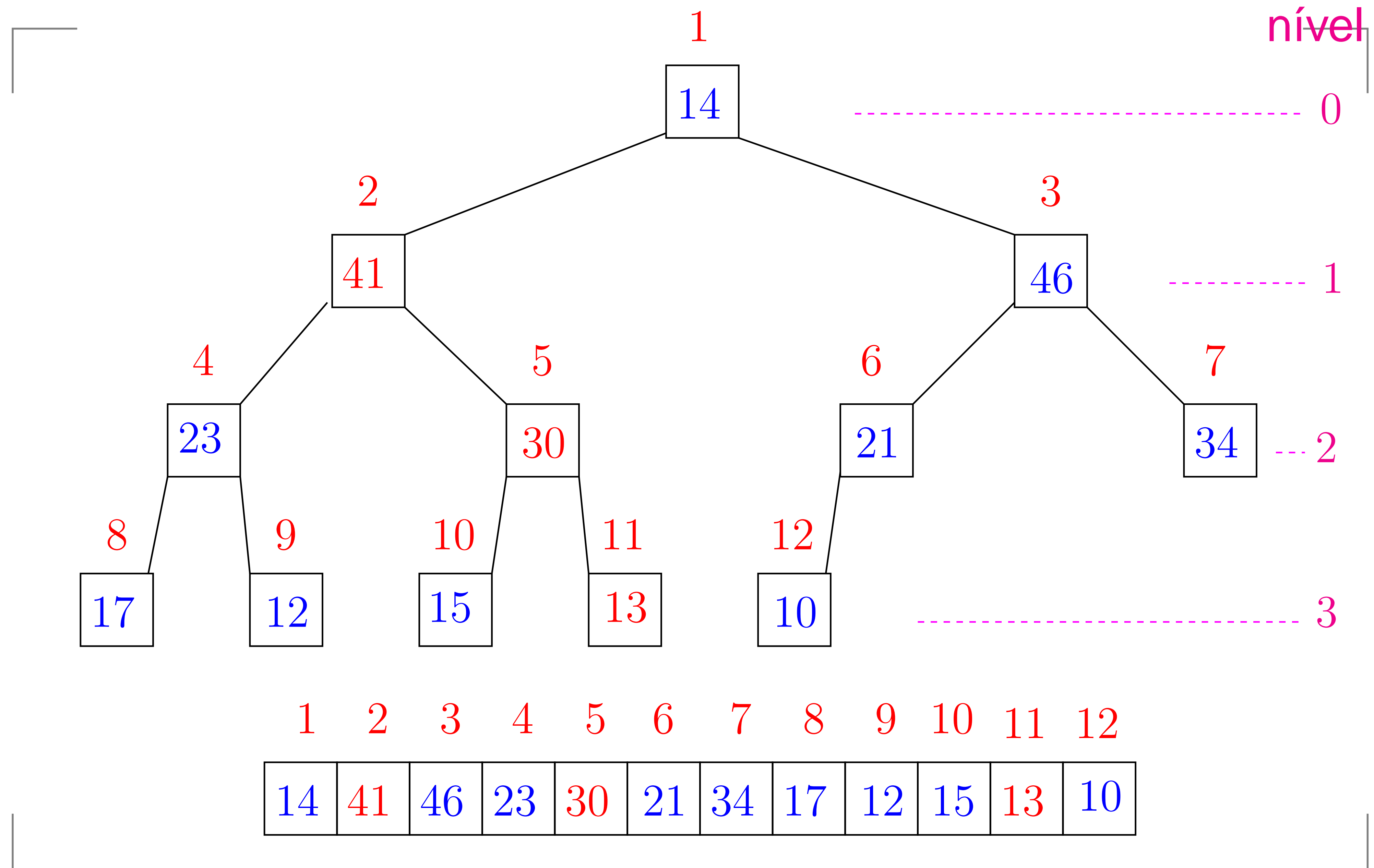
Construção de um max-heap



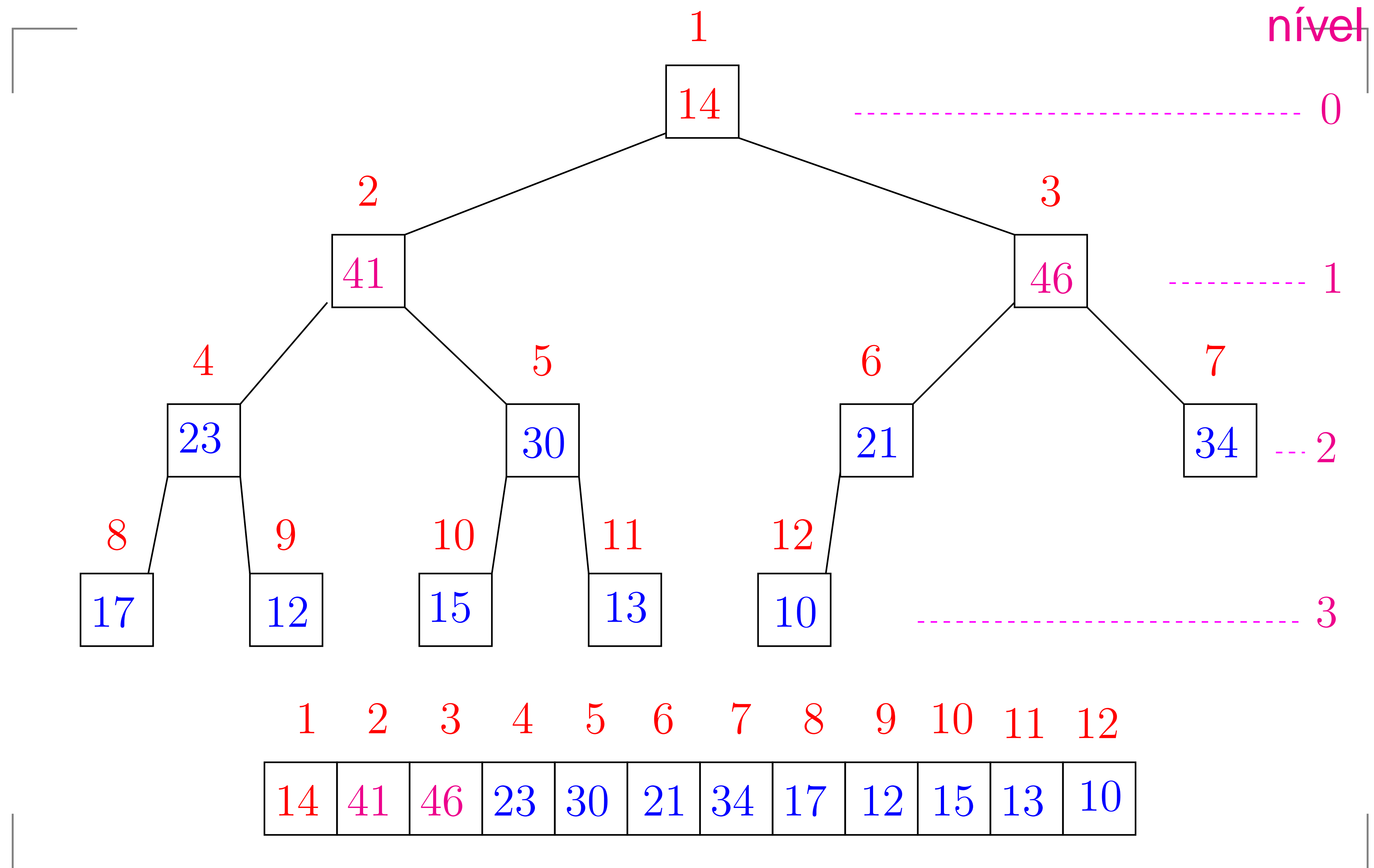
Construção de um max-heap



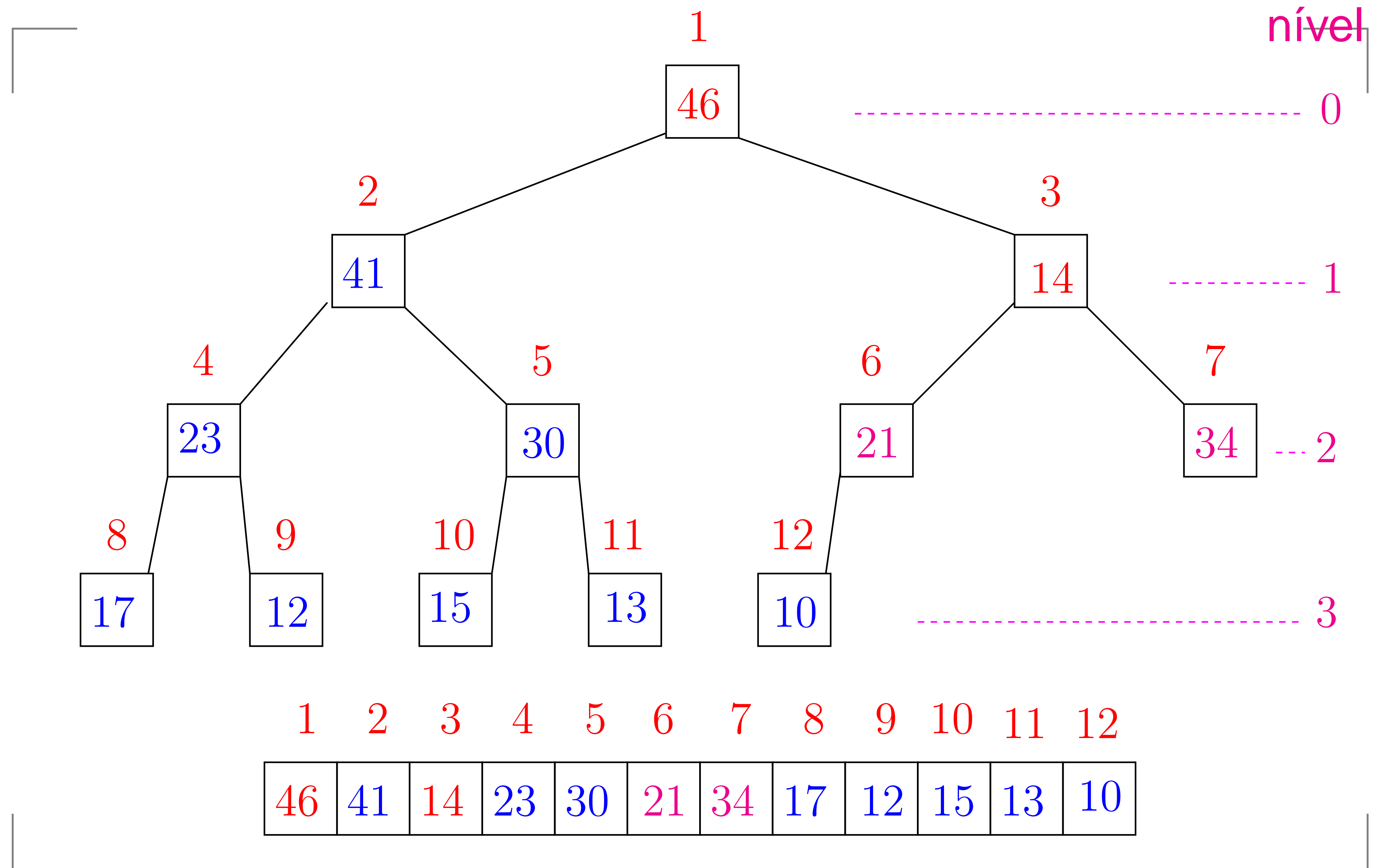
Construção de um max-heap



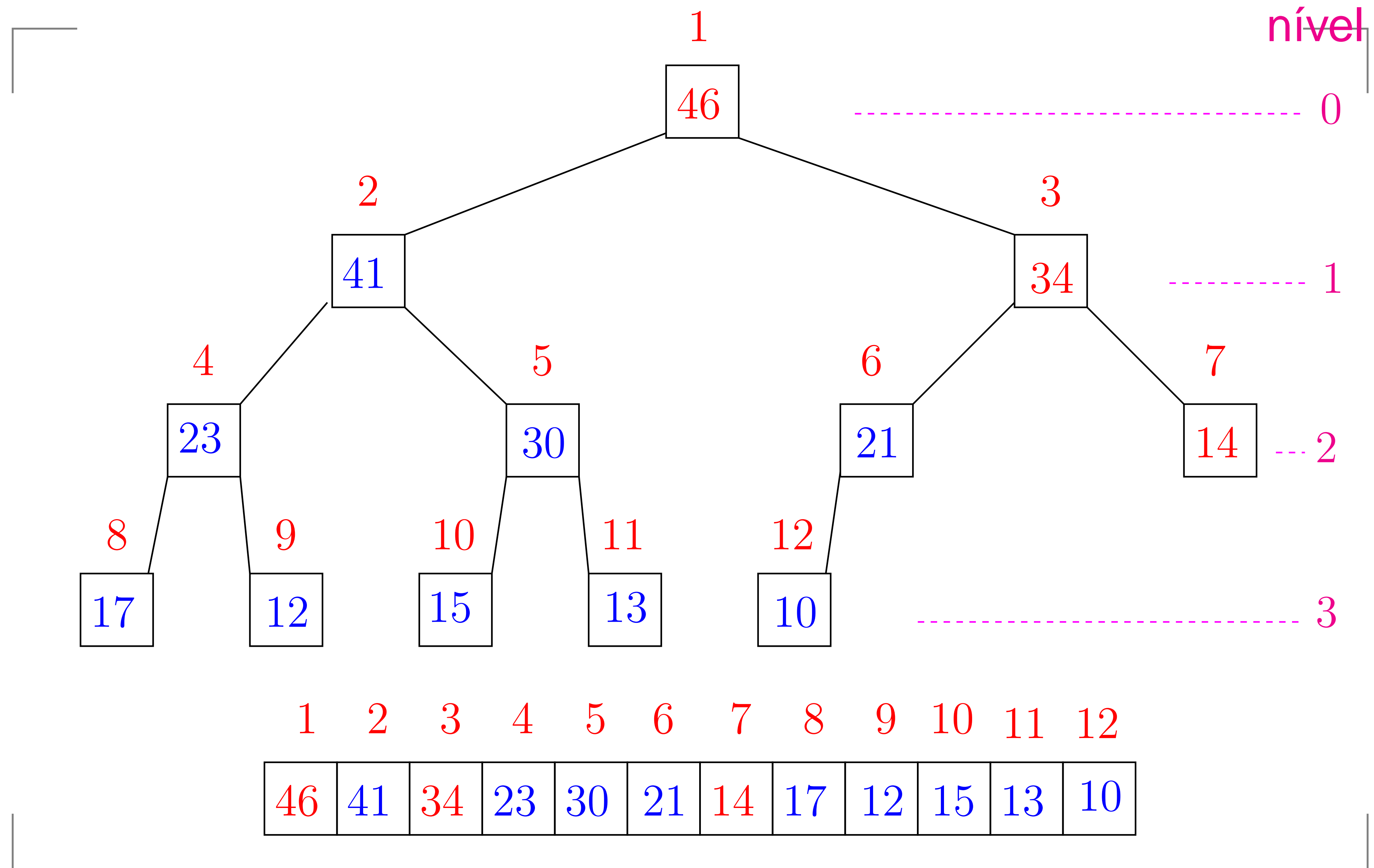
Construção de um max-heap



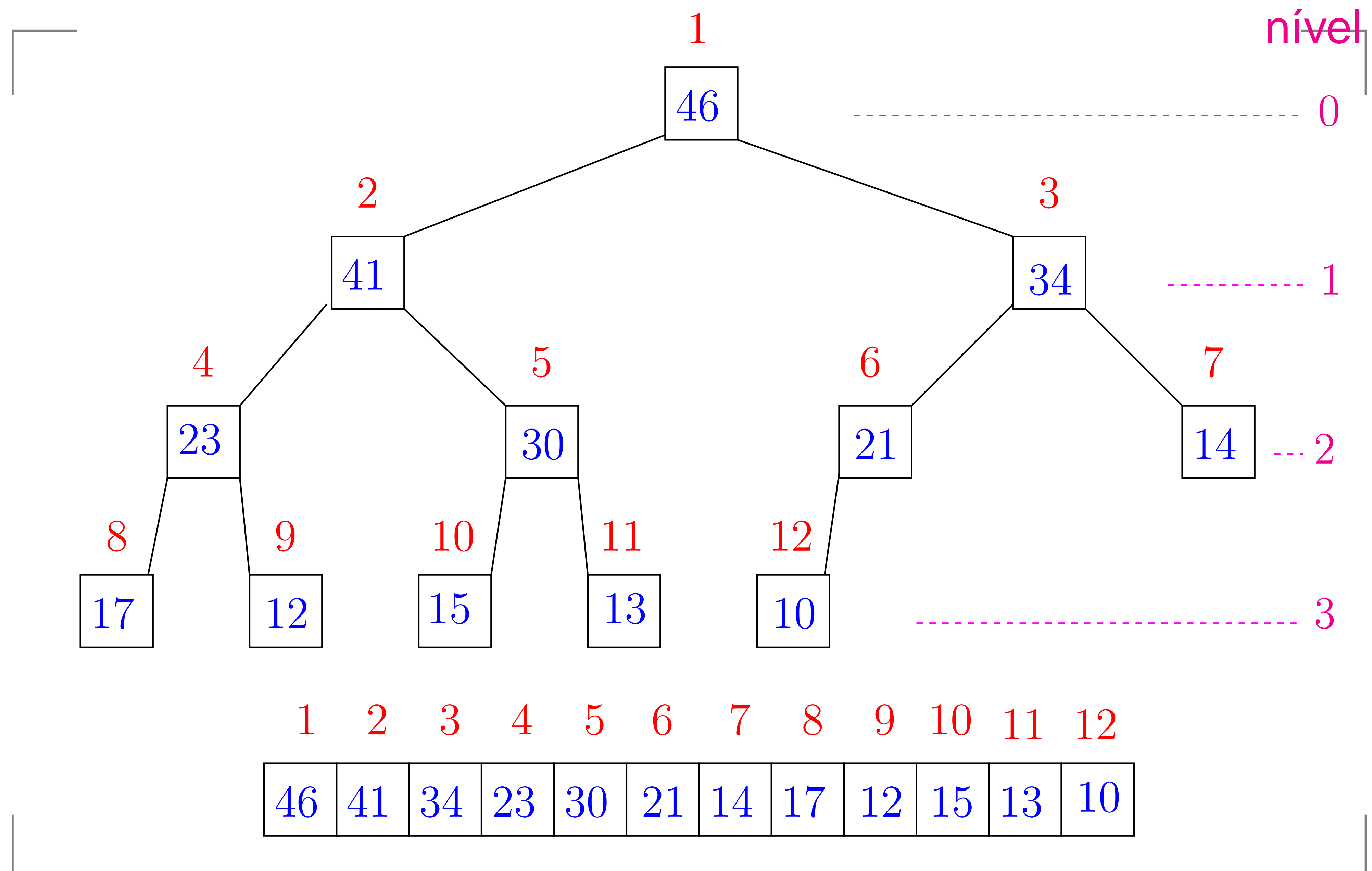
Construção de um max-heap



Construção de um max-heap



Construção de um max-heap



Construção de um max-heap

Recebe um vetor $A[1 \mathinner{.\,.} n]$ e rearranja A para que seja max-heap.

BUILD-MAX-HEAP (A, n)
2 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
3 MAX-HEAPIFY (A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i+1, \ldots, n$ são raízes de max-heaps.

$T(n)$:= consumo de tempo no pior caso

Construção de um max-heap

Recebe um vetor $A[1\,.\,.\,n]$ e rearranja A para que seja max-heap.

BUILD-MAX-HEAP (A, n)
2 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
3 MAX-HEAPIFY (A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i+1, \ldots, n$ são raízes de max-heaps.

$T(n)$:= consumo de tempo no pior caso

Análise grosseira: $T(n)$ é $\frac{n}{2}\,\mathrm{O}(\lg n) = \mathrm{O}(n \lg n)$.

Análise mais cuidadosa: $T(n)$ é ????.

$T(n)$ é $\mathrm{O}(n)$

Prova: O consumo de MAX-HEAPIFY (A, n, i) é proporcional a $h = \lfloor \lg \frac{n}{i} \rfloor$. Logo,

$$
\begin{aligned}
T(n) &\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil h \\
&\leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h} h \quad \text{(Exercício 12.C)} \\
&= n \left(\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{\lfloor \lg n \rfloor}{2^{\lfloor \lg n \rfloor}} \right) \\
&< n \frac{1/2}{(1 - 1/2)^2} \\
&= 2n.
\end{aligned}
$$

$T(n)$ é $\mathrm{O}(n)$

Prova: O consumo de tempo de MAX-HEAPIFY (A, n, i) é a $\mathrm{O}(h) = \mathrm{O}(\lfloor \lg \frac{n}{i} \rfloor)$. Logo,

$$
\begin{aligned}
T(n) &= \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \mathrm{O}(h) \\
&= \mathrm{O}\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) \qquad \text{(Exercício 12.C)} \\
&= \mathrm{O}\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) \\
&= \mathrm{O}\left(n \frac{1/2}{(1 - 1/2)^2} \right) \\
&= \mathrm{O}(2n) = \mathrm{O}(n)
\end{aligned}
$$

Algumas séries

Para todo número real x, $|x| < 1$, temos que $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$.

Para todo número real x, $|x| < 1$, temos que

$$\sum_{i=1}^{\infty} i\, x^i = \frac{x}{(1-x)^2}$$

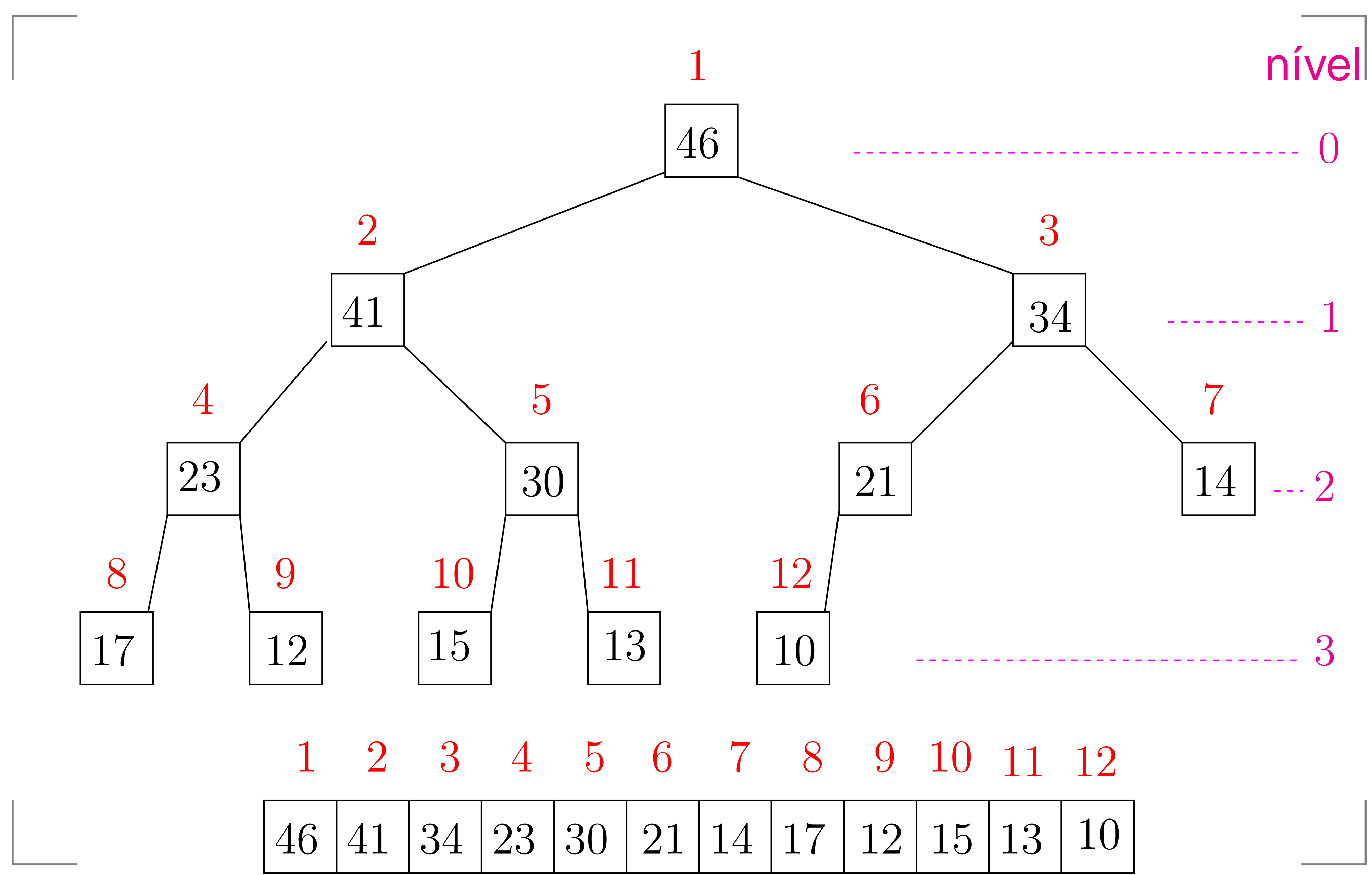
Prova:

$$\begin{aligned}
\sum_{i=1}^{\infty} i\, x^i &= \sum_{i=1}^{\infty} x^i + \sum_{i=2}^{\infty} x^i + \cdots + \sum_{i=k}^{\infty} x^i + \cdots \\
&= \frac{x}{1-x} + \frac{x^2}{1-x} + \cdots + \frac{x^k}{1-x} + \cdots \\
&= \frac{x}{1-x}(x^0 + x^1 + x^2 + \cdots + x^k + \cdots) = \frac{x}{(1-x)^2}.
\end{aligned}$$

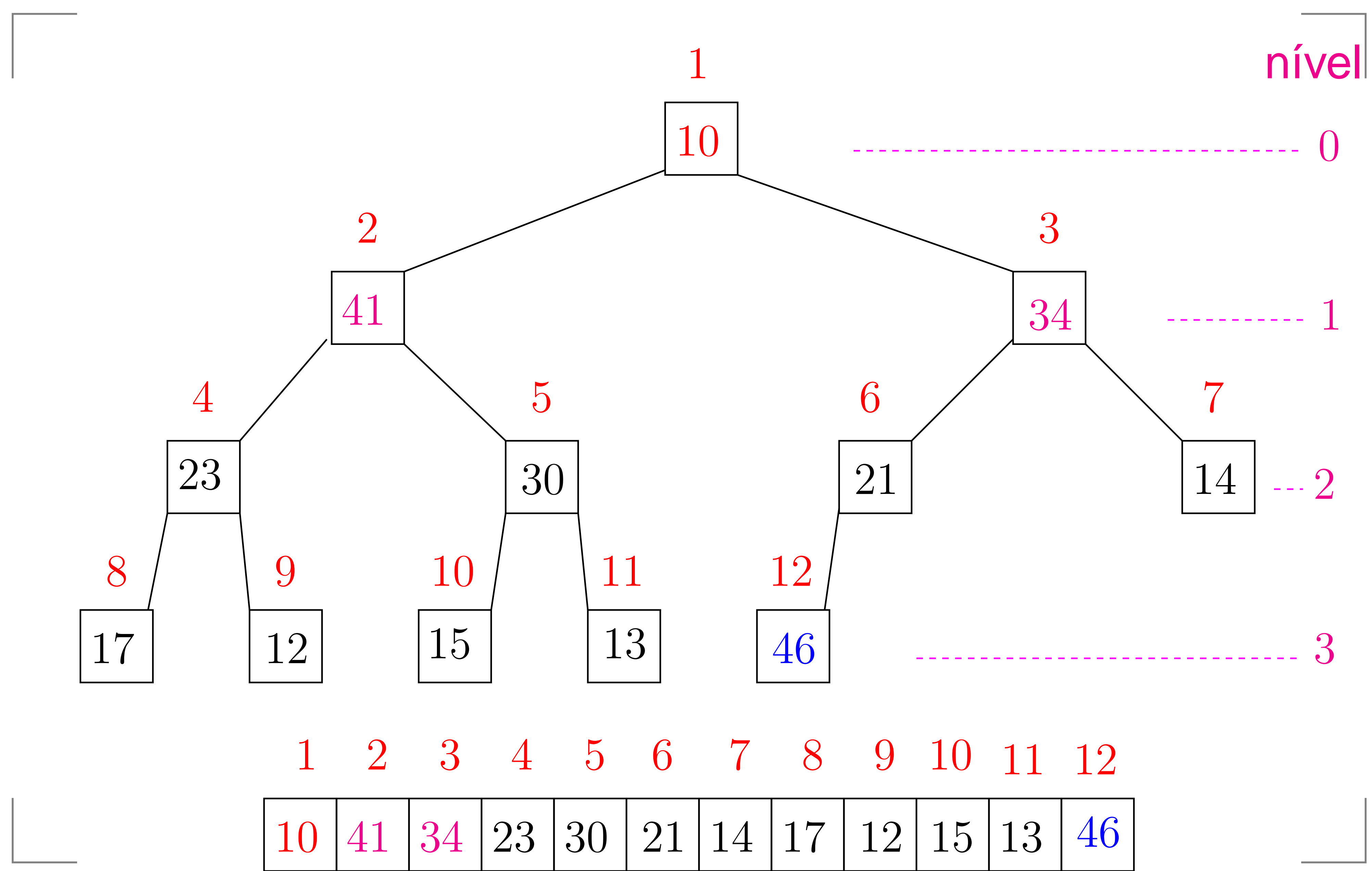
Conclusão

O consumo de tempo do algoritmo BUILD-MAX-HEAP é $\Theta(n)$.

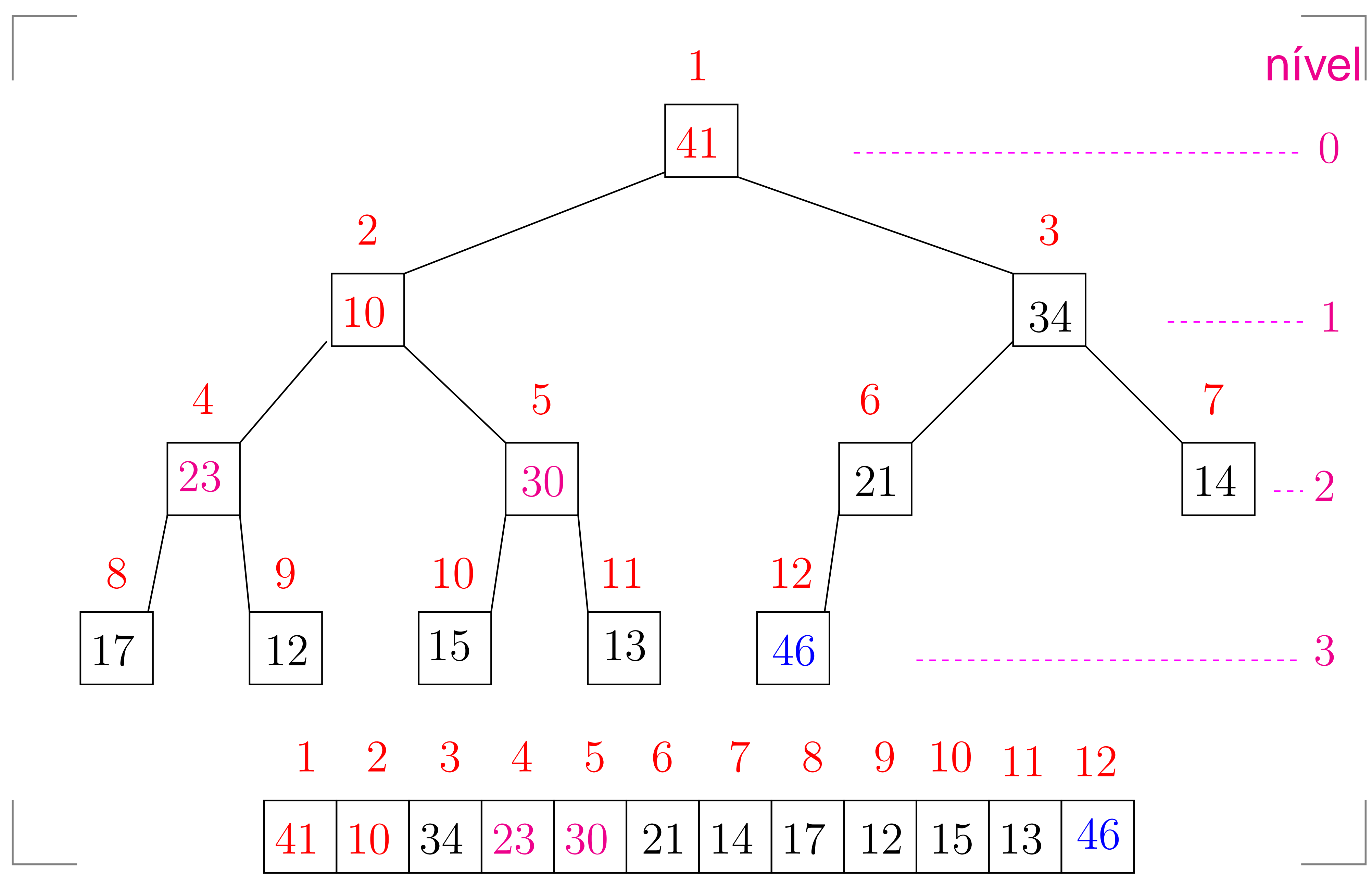
Heapsort



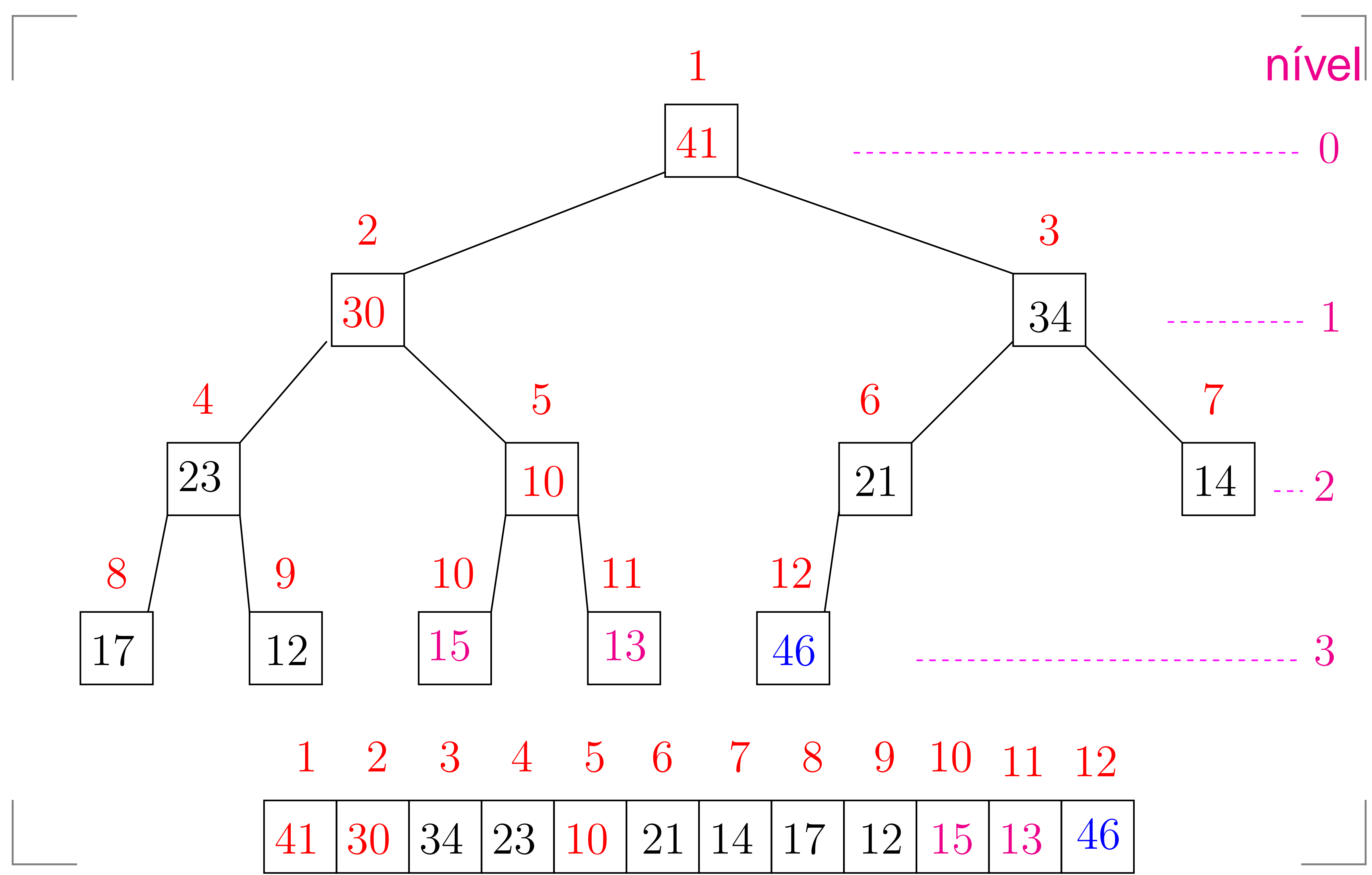
Heapsort



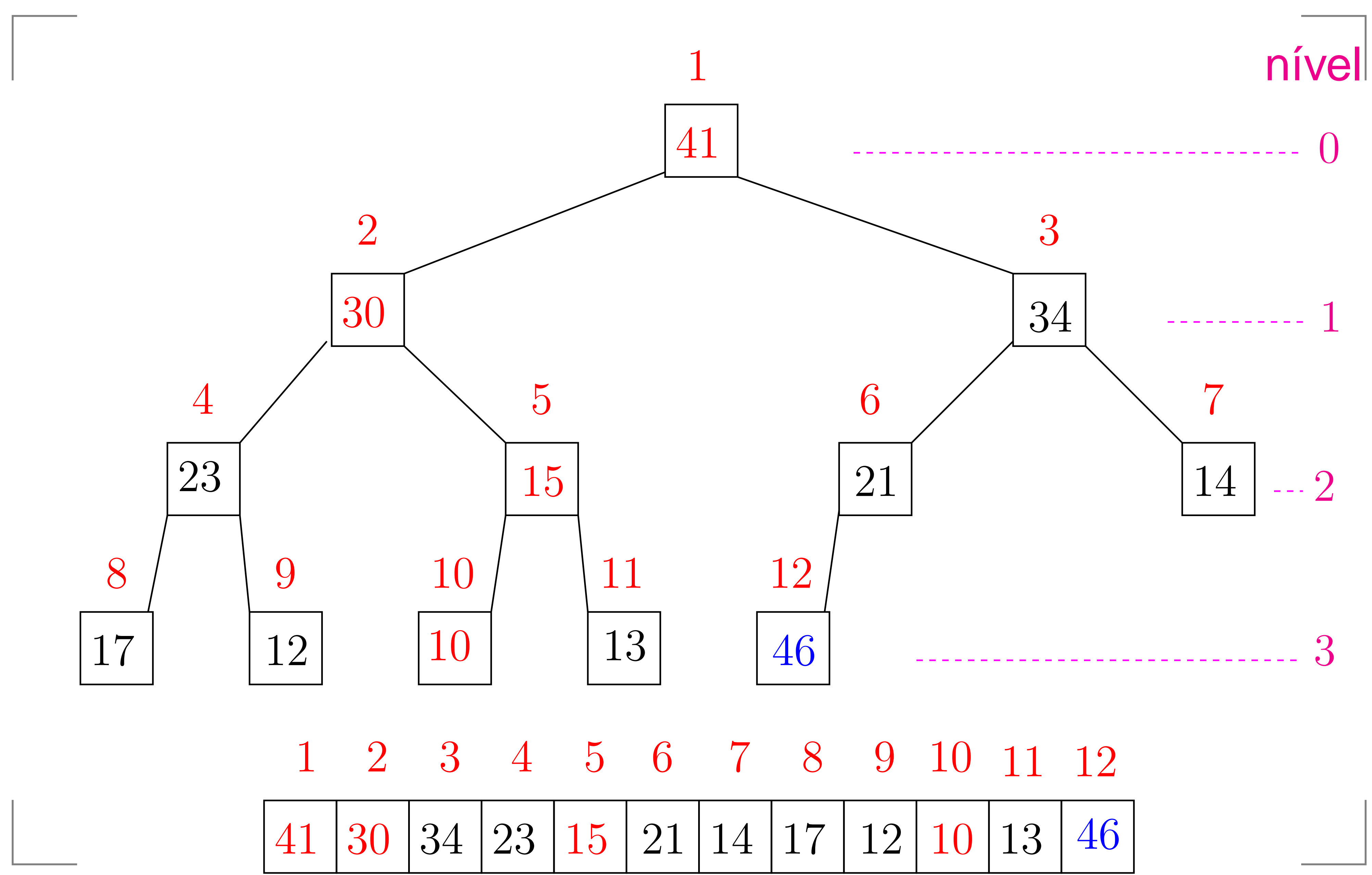
Heapsort



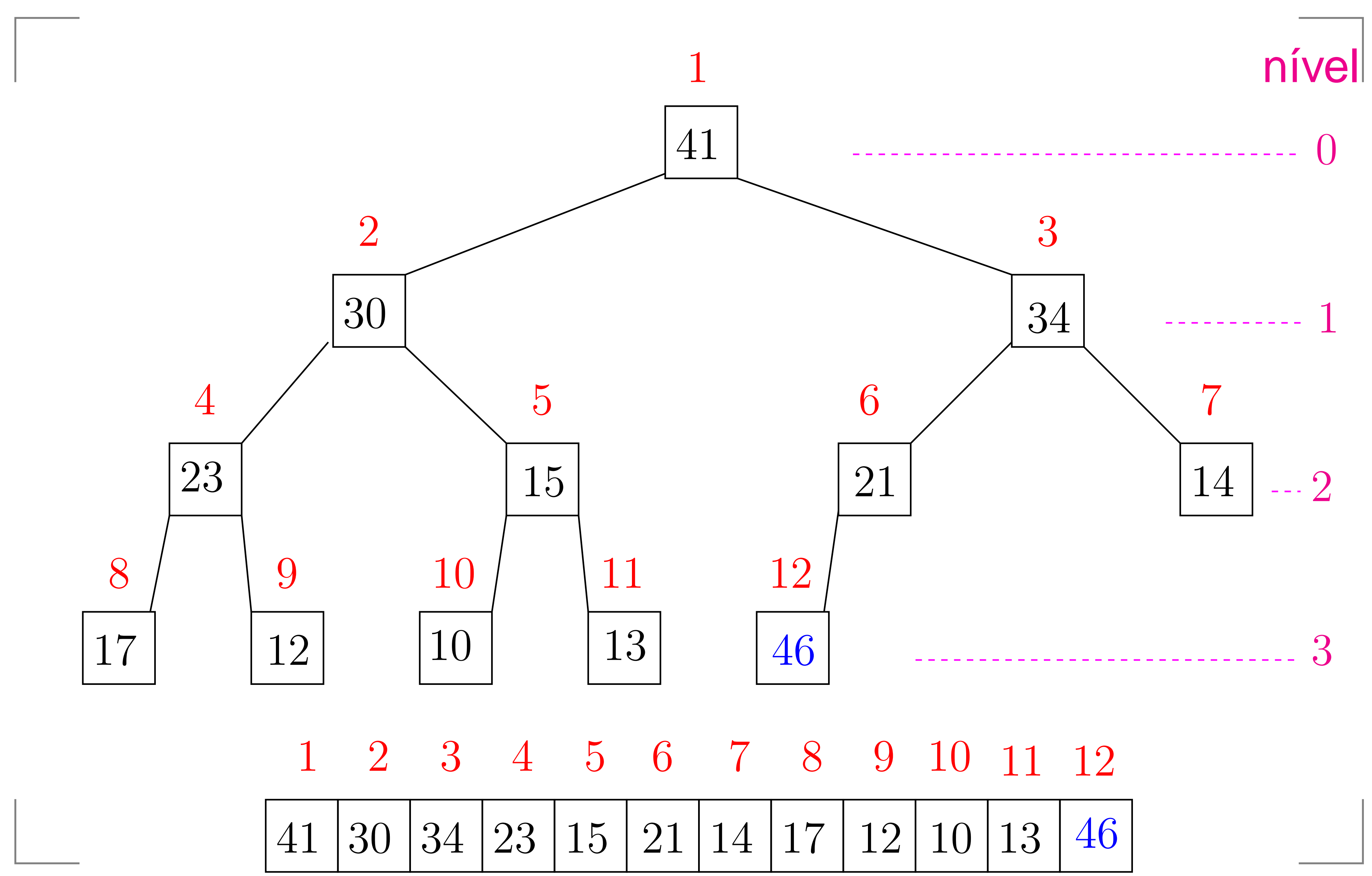
Heapsort



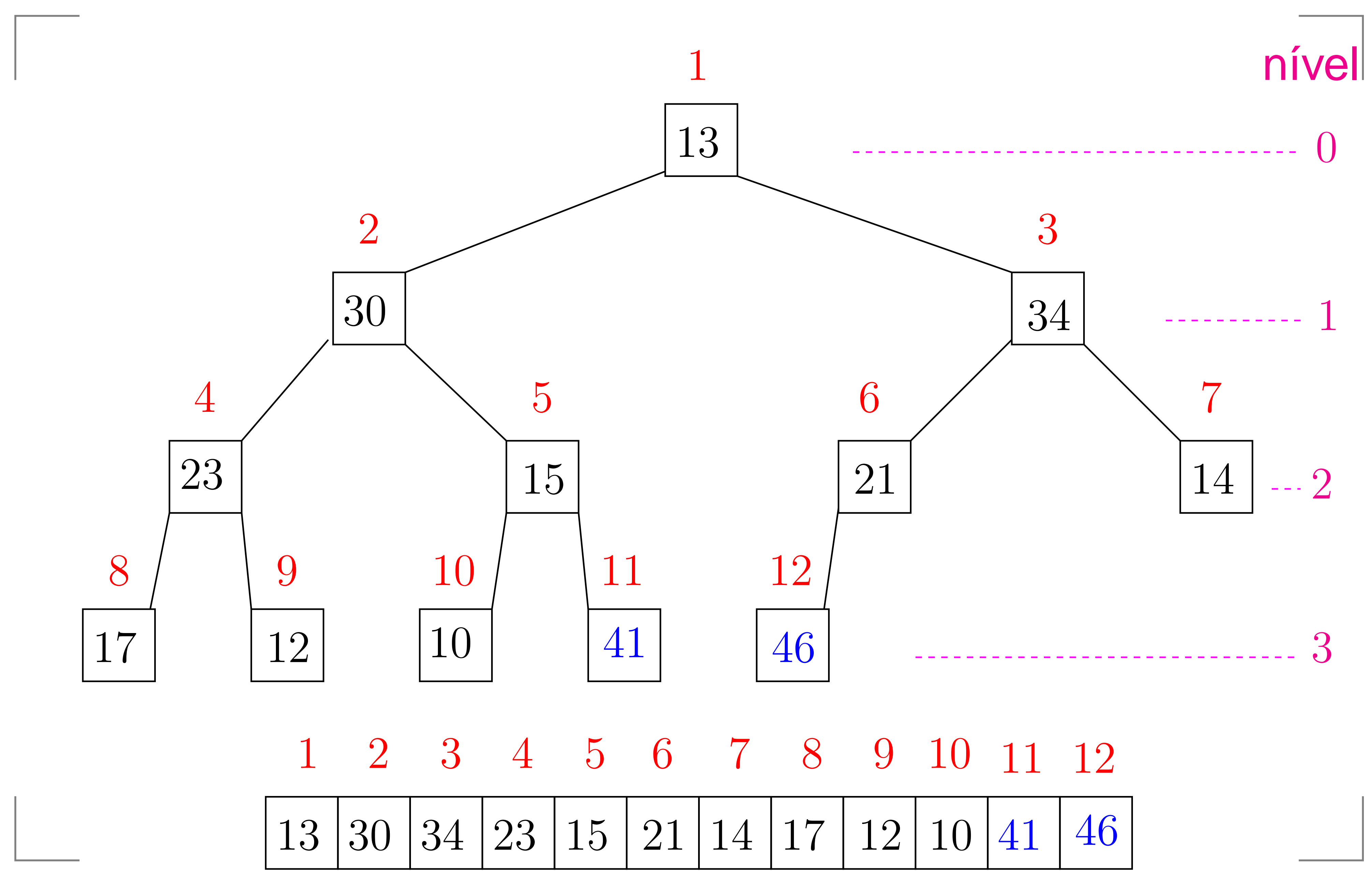
Heapsort



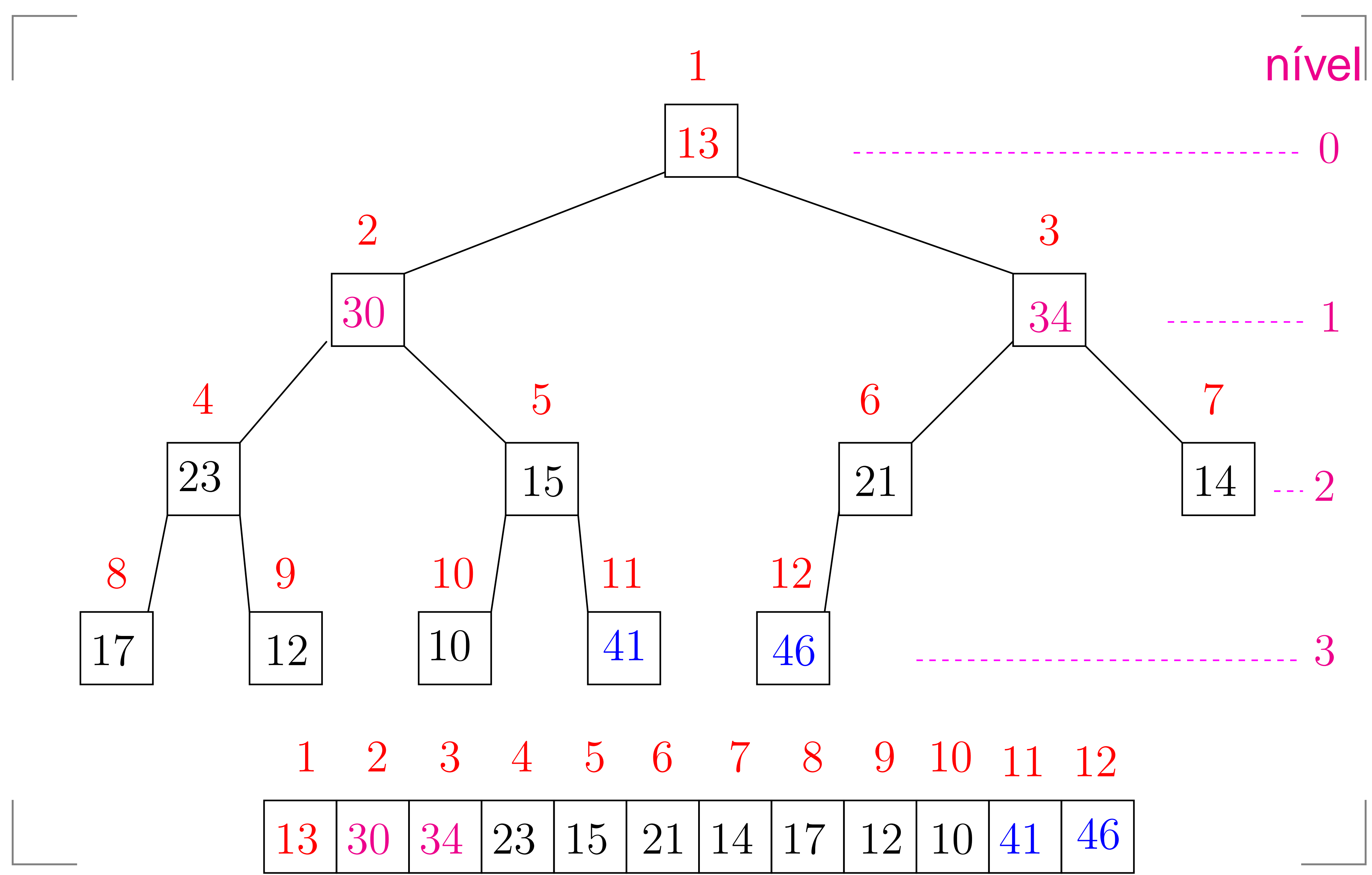
Heapsort



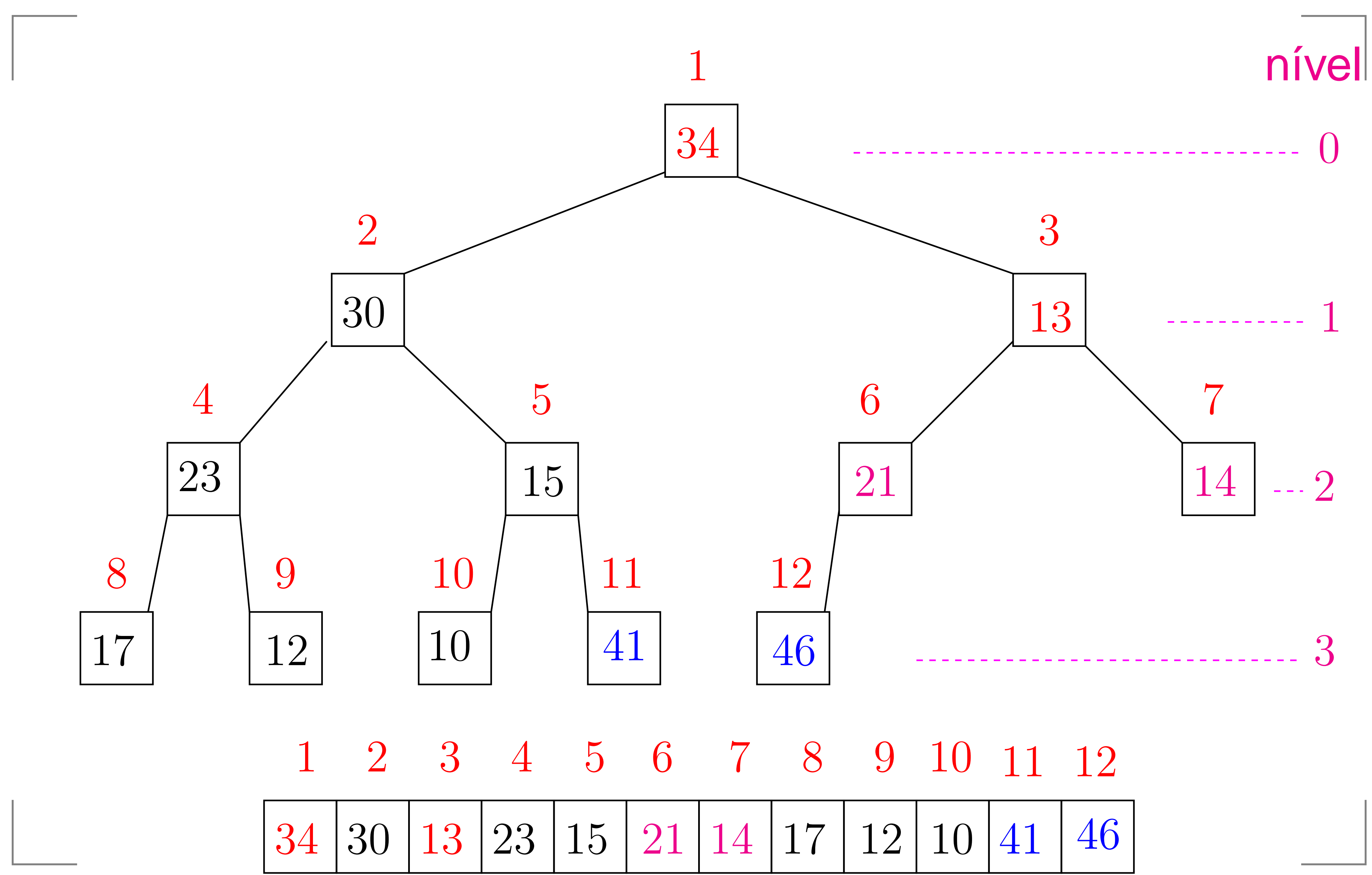
Heapsort



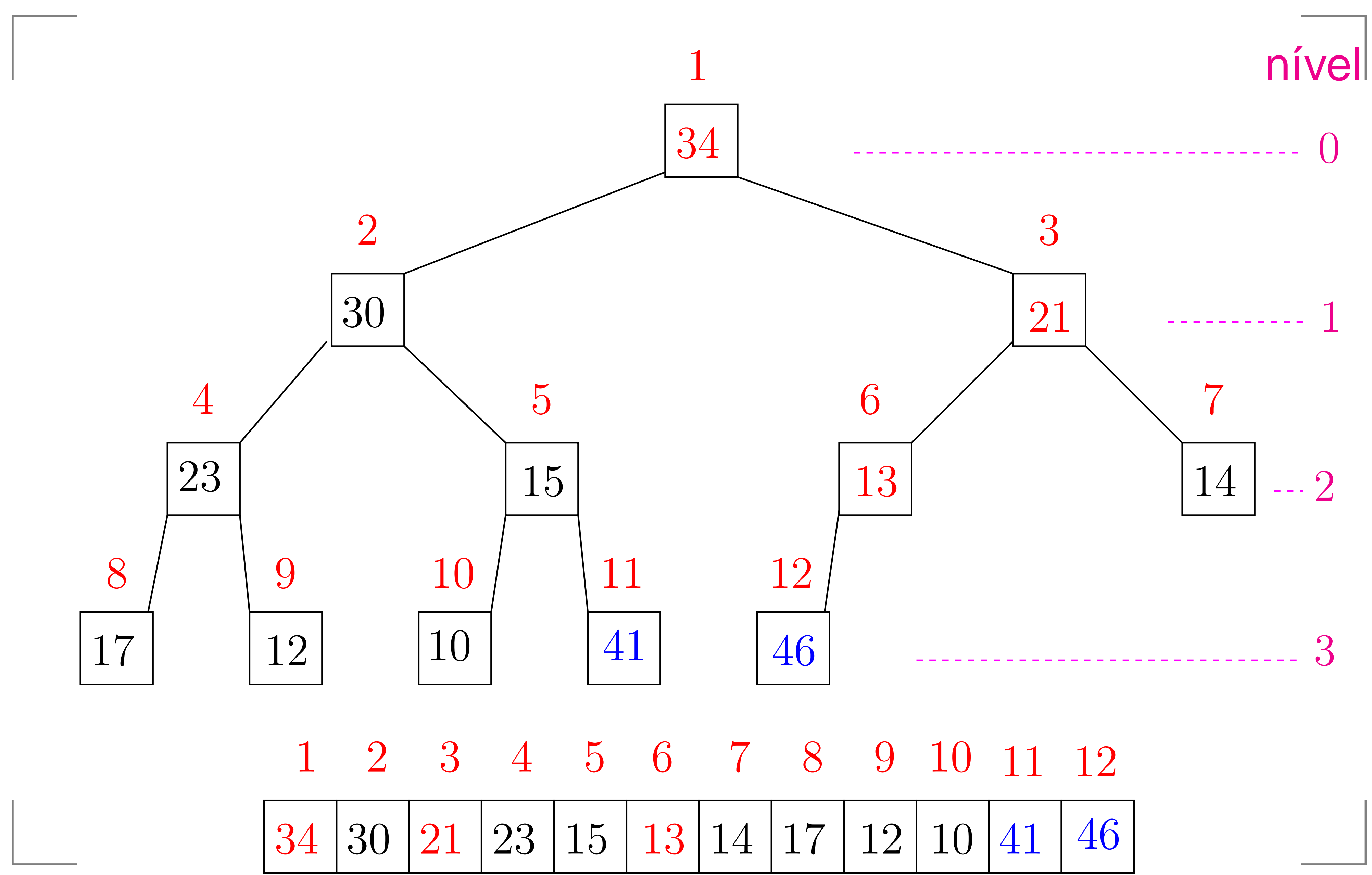
Heapsort



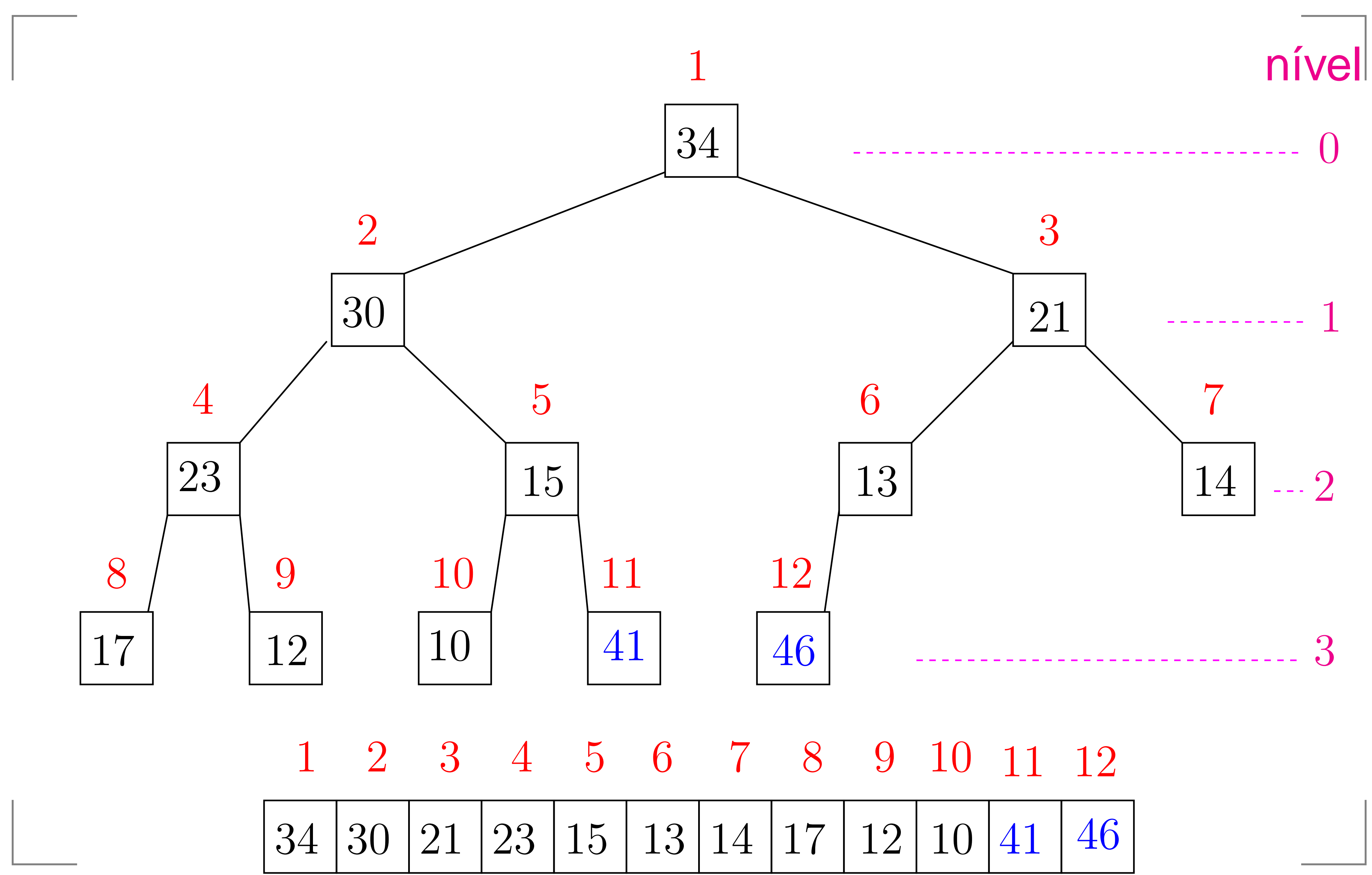
Heapsort



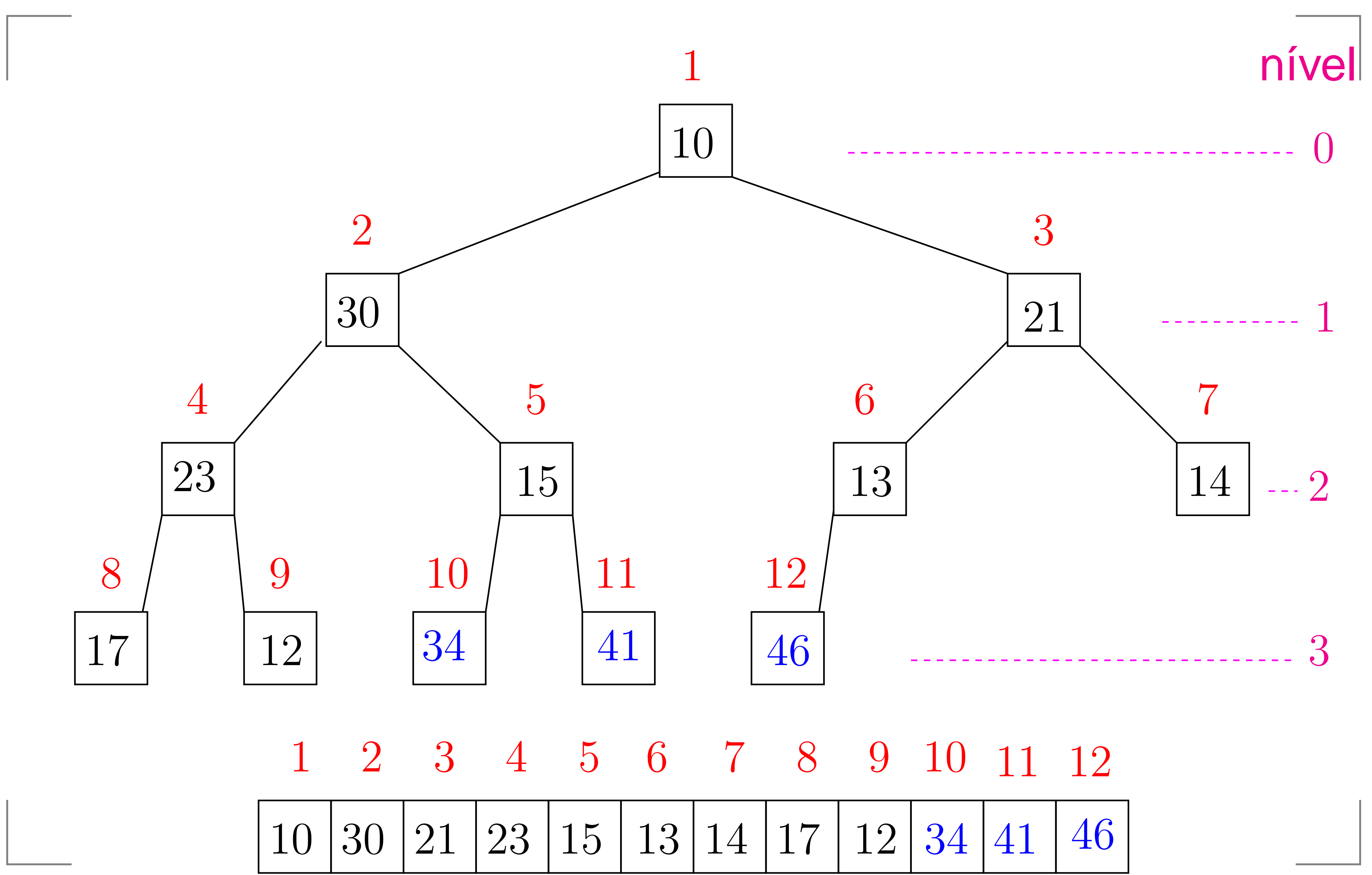
Heapsort



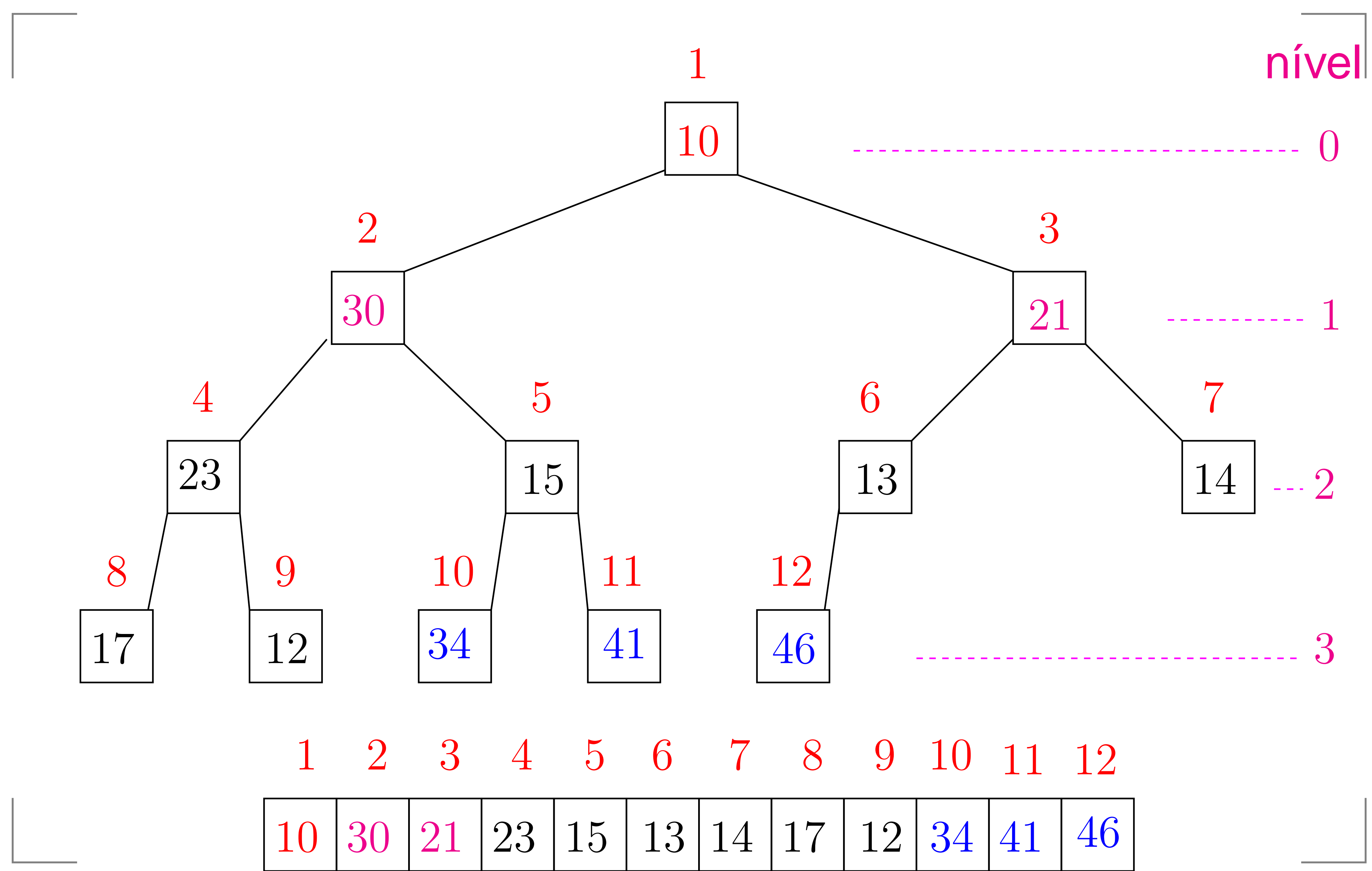
Heapsort



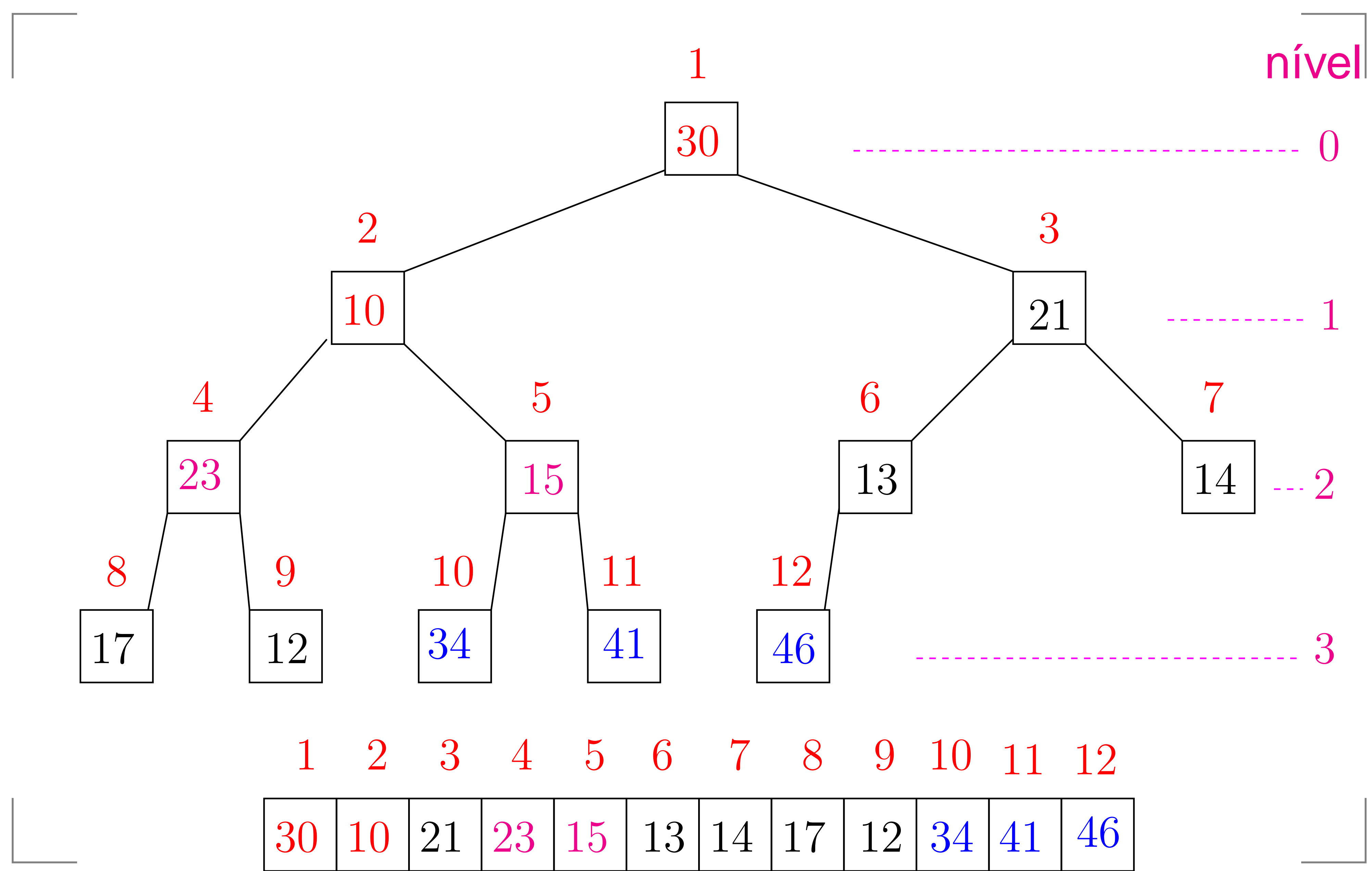
Heapsort



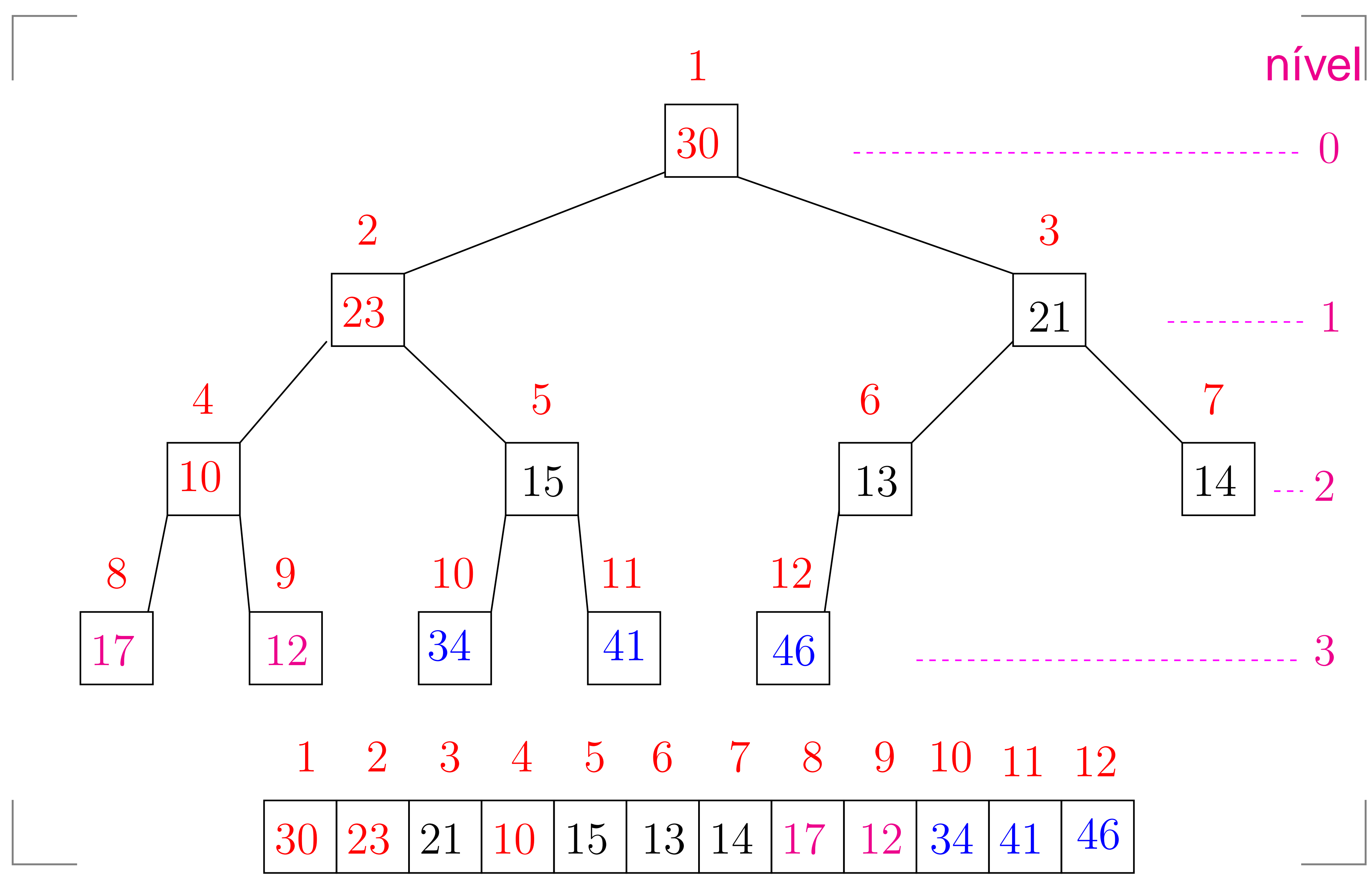
Heapsort



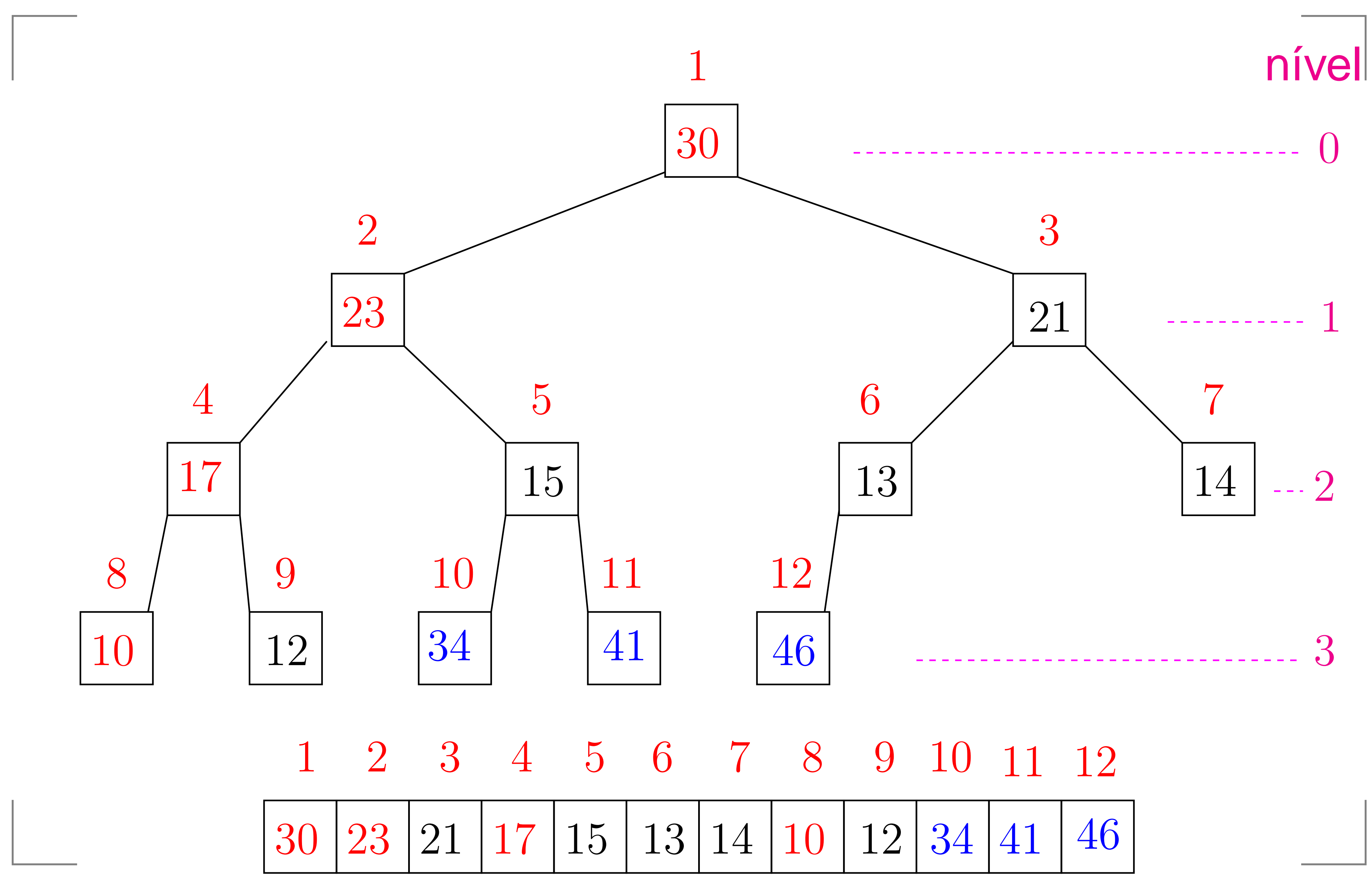
Heapsort



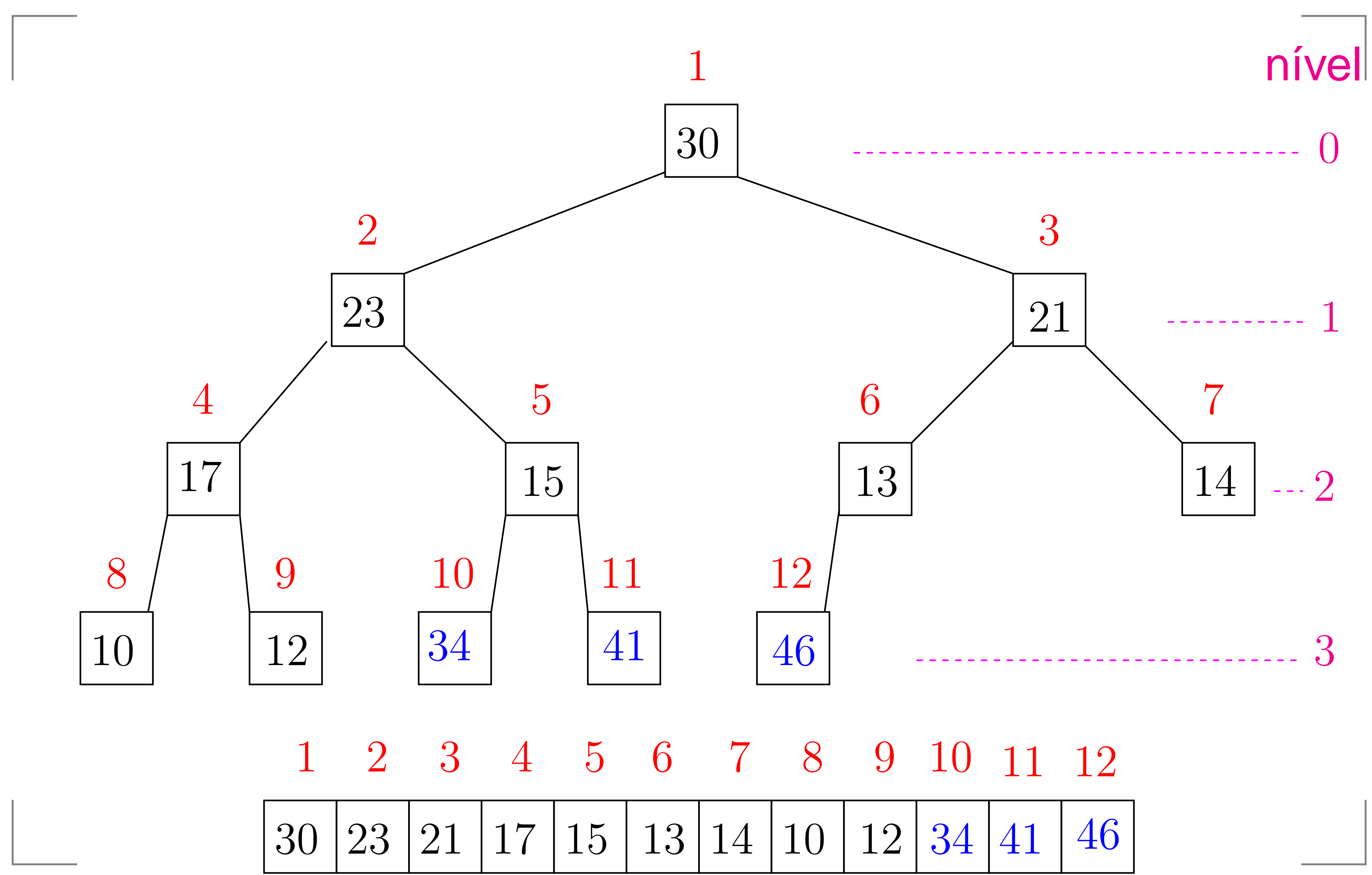
Heapsort



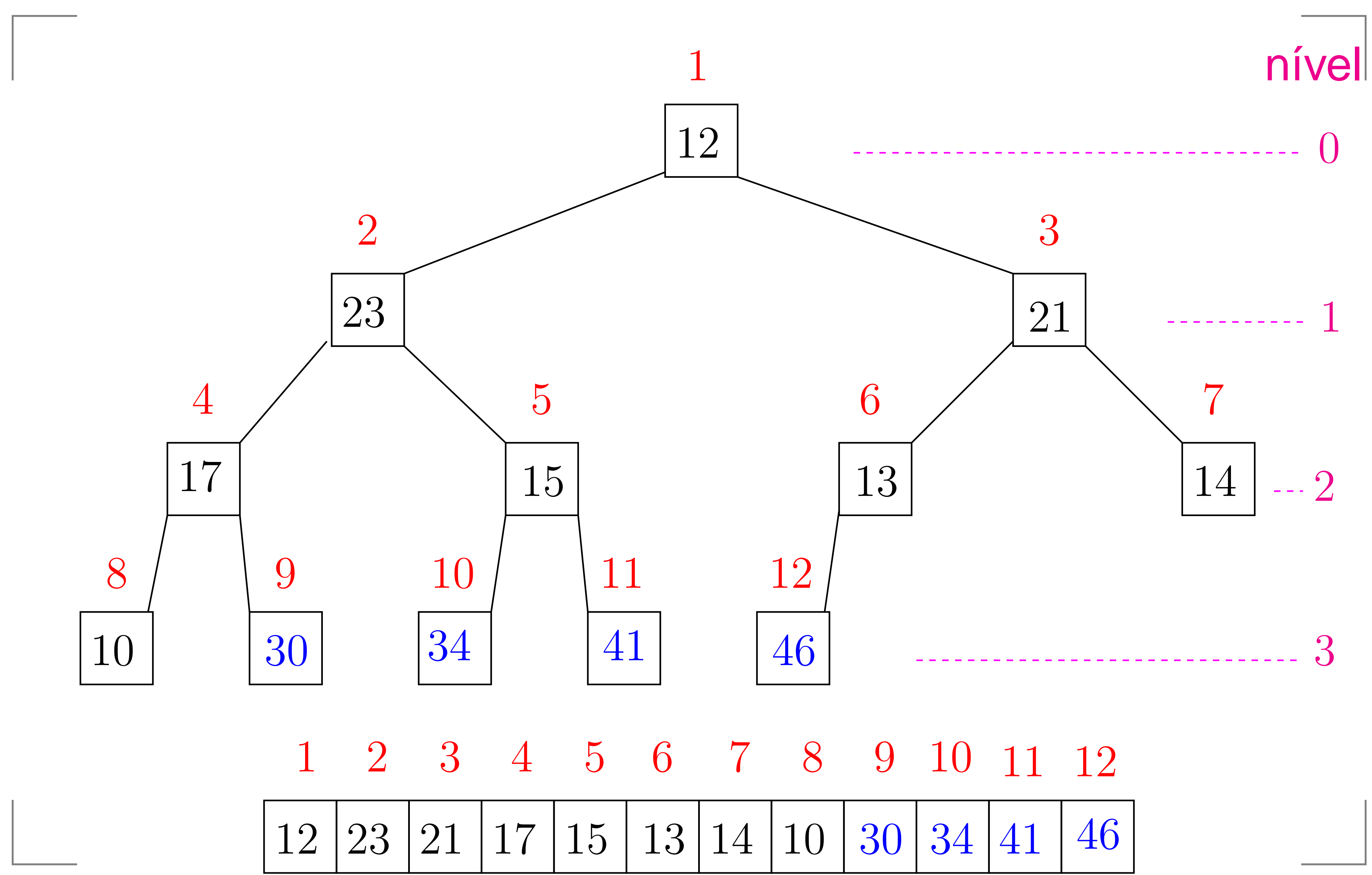
Heapsort



Heapsort



Heapsort



Heapsort

Algoritmo rearranja $A[1\,..\,n]$ em ordem crescente.

HEAPSORT (A, n)
0 BUILD-MAX-HEAP (A, n) $\triangleright$ pré-processamento
1 $m \leftarrow n$
2 **para** $i \leftarrow n$ **decrescendo até** 2 **faça**
3 $A[1] \leftrightarrow A[i]$
4 $m \leftarrow m - 1$
5 MAX-HEAPIFY $(A, m, 1)$

Relações invariantes: Na linha 2 vale que:

(i0) $A[m\,..\,n]$ é crescente;

(i1) $A[1\,..\,m] \leq A[m+1]$;

(i2) $A[1\,..\,m]$ é um max-heap.

Consumo de tempo

linha	todas as execuções da linha
0	$= \Theta(n)$
1	$= \Theta(1)$
2	$= \Theta(n)$
3	$= \Theta(n)$
4	$= \Theta(n)$
6	$= n\mathrm{O}(\lg n)$

total $= n\mathrm{O}(\lg n) + \Theta(4n+1) = \mathrm{O}(n \lg n)$

Conclusão

O consumo de tempo do algoritmo HEAPSORT é $O(n \lg n)$.

Um pouco de experimentação

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

Os códigos estão compilados com o gcc versão 2.7.2.1 e opção de compilação -O2.

Algoritmos implementados:

bub	`bubblesort`
bub2	`bubblesort_2`
shkr	`shakersort`
sele	ORDENA-POR-SELEÇÃO
ins	ORDENA-POR-INSERÇÃO
insS	`inserção Sedgewick`
insB	`inserção binária`
shell	`shellsort`

Crescente

n	bub	bub2	shkr	sele	ins	insS	insB	she
256	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.
512	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.
1024	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.
2048	0.04	0.00	0.00	0.04	0.00	0.00	0.00	0.
4096	0.18	0.00	0.00	0.14	0.00	0.00	0.00	0.
8192	0.90	0.00	0.00	0.71	0.00	0.00	0.01	0.
16384	3.81	0.00	0.00	3.04	0.00	0.01	0.00	0.
32768	15.48	0.00	0.00	12.32	0.00	0.00	0.01	0.
65536	62.43	0.01	0.00	49.44	0.00	0.01	0.04	0.
31072	266.90	0.01	0.00	214.86	0.01	0.01	0.09	0.

Decrescente

n	bub	bub2	shkr	sele	ins	insS	in
256	0.00	0.01	0.00	0.00	0.00	0.00	0.
512	0.00	0.01	0.00	0.01	0.00	0.00	0.
1024	0.01	0.02	0.02	0.01	0.00	0.01	0.
2048	0.06	0.08	0.07	0.03	0.03	0.02	0.
4096	0.28	0.29	0.28	0.15	0.11	0.07	0.
8192	1.30	1.25	1.20	0.71	0.75	0.56	0.
16384	5.41	5.15	4.98	3.01	3.45	2.57	3.
32768	23.59	22.19	20.56	13.28	14.56	11.21	14.
65536	91.07	85.88	84.66	51.16	67.98	55.89	59.
31072	361.67	339.09	341.04	204.23	244.55	207.27	222.

Aleatório: média de 10

n	bub	bub2	shkr	sele	ins	insS	in
256	0.00	0.00	0.00	0.00	0.00	0.00	0.
512	0.01	0.01	0.01	0.00	0.00	0.00	0.
1024	0.03	0.02	0.02	0.01	0.00	0.01	0.
2048	0.09	0.11	0.07	0.03	0.02	0.01	0.
4096	0.37	0.41	0.27	0.14	0.06	0.04	0.
8192	1.70	1.77	1.12	0.71	0.29	0.22	0.
16384	7.08	7.33	4.75	3.05	1.55	1.25	1.
32768	28.54	29.32	19.30	12.33	6.88	5.61	7.
65536	113.55	116.95	77.64	49.17	28.37	23.17	28.
31072	493.08	506.67	323.34	224.09	121.01	105.44	115.

Mais experimentação

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

Os códigos estão compilados com o gcc versão 2.7.2.1 e opção de compilação -O2.

Algoritmos implementados:

`shell`	`shellsort` origina
`merge_r`	MERGE-SORT recursivo
`merge_i`	MERGE-SORT iterativo
`heap`	HEAPSORT

Crescente

n	shell	merge_r	merge_i	heap
4096	0.01	0.00	0.01	0.00
8192	0.00	0.01	0.01	0.01
16384	0.01	0.02	0.02	0.01
32768	0.02	0.05	0.04	0.03
65536	0.05	0.10	0.10	0.07
131072	0.12	0.23	0.28	0.16

Decrescente

n	shell	merge_r	merge_i	heap
4096	0.00	0.01	0.00	0.00
8192	0.00	0.01	0.01	0.01
16384	0.02	0.02	0.02	0.01
32768	0.03	0.04	0.04	0.03
65536	0.07	0.09	0.09	0.07
131072	0.15	0.21	0.28	0.16

Aleatório: média de 10

n	shell	merge_r	merge_i	heap
4096	0.01	0.01	0.00	0.01
8192	0.01	0.01	0.01	0.01
16384	0.03	0.02	0.02	0.02
32768	0.07	0.05	0.05	0.04
65536	0.16	0.11	0.11	0.10
131072	0.38	0.25	0.33	0.23
262144	0.92	0.54	0.73	0.54
524288	2.13	1.18	1.55	1.31
1048576	4.97	2.52	3.32	3.06
2097152	11.38	5.42	6.96	7.07
4194304	26.50	11.40	14.46	15.84
8388608	61.68	24.35	29.76	35.85

Exercícios

Exercício 12.A

A altura de i em $A[1\,..\,m]$ é o comprimento da mais longa seqüência da forma

$$\langle \text{filho}(i), \text{filho}(\text{filho}(i)), \text{filho}(\text{filho}(\text{filho}(i))), \ldots \rangle$$

onde $\text{filho}(i)$ vale $2i$ ou $2i+1$. Mostre que a altura de i é $\lfloor \lg \frac{m}{i} \rfloor$.
É verdade que $\lfloor \lg \frac{m}{i} \rfloor = \lfloor \lg m \rfloor - \lfloor \lg i \rfloor$?

Exercício 12.B

Mostre que um heap $A[1\,..\,m]$ tem no máximo $\lceil m/2^{h+1} \rceil$ nós com altura h.

Exercício 12.C

Mostre que $\lceil m/2^{h+1} \rceil \leq m/2^h$ quando $h \leq \lfloor \lg m \rfloor$.

Exercício 12.D

Mostre que um heap $A[1\,..\,m]$ tem no mínimo $\lfloor m/2^{h+1} \rfloor$ nós com altura h.

Exercício 12.E

Considere um heap $A[1\,..\,m]$; a raiz do heap é o elemento de índice 1. Seja m' o número de elementos do “sub-heap esquerdo”, cuja raiz é o elemento de índice 2. Seja m'' o número de elementos do “sub-heap direito”, cuja raiz é o elemento de índice 3. Mostre que

$$m'' \leq m' < 2m/3\,.$$

Mais execícios

Exercício 12.F

Mostre que a solução da recorrência

$$\begin{aligned} T(1) &= 1 \\ T(k) &\leq T(\lfloor 2k/3 \rfloor) + 5 \quad \text{para } k \geq 2 \end{aligned}$$

é $\mathrm{O}(\log k)$. Mais geral: mostre que se $T(k) = T(2k/3) + \mathrm{O}(1)$ então $T(k)$ é $\mathrm{O}(\log k)$. (Curiosidade: Essa é a recorrência do MAX-HEAPIFY (A, m, i) se interpretarmos k como sendo o número de nós na subárvore com raiz i).

Exercício 12.G

Escreva uma versão iterativa do algoritmo MAX-HEAPIFY. Faça uma análise do consumo de tempo do algoritmo.

Mais exercícios ainda

Exercício 12.H

Discuta a seguinte variante do algoritmo MAX-HEAPIFY:

M-H (A, m, i)
1 $e \leftarrow 2i$
2 $d \leftarrow 2i + 1$
3 **se** $e \leq m$ e $A[e] > A[i]$
4 **então** $A[i] \leftrightarrow A[e]$
5 M-H (A, m, e)
6 **se** $d \leq m$ e $A[d] > A[i]$
7 **então** $A[i] \leftrightarrow A[d]$
8 M-H (A, m, d)

Filas com prioridades

CLRS 6.5 e 19

Filas com prioridades

Uma fila com prioridades é um tipo abstrato de dados que consiste de uma coleção S de itens, cada um com um valor ou prioridade associada.

Algumas operações típicas em uma fila com prioridades são:

MAXIMUM(S): devolve o elemento de S com a maior prioridade;

EXTRACT-MAX(S): remove e devolve o elemento em S com a maior prioridade;

INCREASE-KEY(S, s, p): aumenta o valor da prioridade do elemento s para p; e

INSERT(S, s, p): insere o elemento s em S com prioridade p.

Implementação com max-heap

HEAP-MAX (A, m)
1 **devolva** $A[1]$

Consome tempo $\Theta(1)$.

HEAP-EXTRACT-MAX (A, m) $\triangleright$ $m \geq 1$
1 $max \leftarrow A[1]$
2 $A[1] \leftarrow A[m]$
3 $m \leftarrow m - 1$
4 MAX-HEAPIFY $(A, m, 1)$
5 **devolva** max

Consome tempo $\mathrm{O}(\lg m)$.

Implementação com max-heap

HEAP-INCREASE-KEY $(A, i, prior)$ $\triangleright$ $prior \geq A[i]$

1 $A[i] \leftarrow prior$

2 **enquanto** $i > 1$ e $A[\lfloor i/2 \rfloor] < A[i]$ **faça**

3 $A[i] \leftrightarrow A[\lfloor i/2 \rfloor]$

4 $i \leftarrow \lfloor i/2 \rfloor$

Consome tempo $\mathrm{O}(\lg m)$.

MAX-HEAP-INSERT $(A, m, prior)$

1 $m \leftarrow m + 1$

2 $A[m] \leftarrow -\infty$

3 HEAP-INCREASE-KEY $(A, m, prior)$

Consome tempo $\mathrm{O}(\lg m)$.

Outras implementações

Operação	max-heap (pior caso)	heap binomial (pior caso)	fibonacci heap (amortizado)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg m)$	$\mathrm{O}(\lg m)$	$\Theta(1)$
MAXIMUM	$\Theta(1)$	$\mathrm{O}(\lg m)$	$\Theta(1)$
EXTRACT-MAX	$\Theta(\lg m)$	$\Theta(\lg m)$	$\mathrm{O}(\lg m)$
UNION	$\Theta(m)$	$\mathrm{O}(\lg m)$	$\Theta(1)$
INCREASE-KEY	$\Theta(\lg m)$	$\Theta(\lg m)$	$\Theta(1)$
DELETE	$\Theta(\lg m)$	$\Theta(\lg m)$	$\mathrm{O}(\lg m)$

MAKE-HEAP(): cria um heap vazio.

Árvores binárias de busca podem ser usadas para implementar filas com prioridades. Consumo de tempo ???.

Exercícios

Exercício 13.A [CLRS 6.5-7]
Escreva uma implementação eficiente da operação MAX-HEAP-DELETE(A, m, i). Ela deve remover o nó i do max-heap $A[1\,..\,m]$ e armazenar os elementos restantes, em forma de max-heap, no vetor $A[1\,..\,m-1]$.

Exercício 13.B [CLRS 6-1, p.142]
O algoritmo abaixo faz a mesma coisa que o BUILD-HEAP?

B-H (A, n)
1 **para** $m \leftarrow 2$ **até** n **faça**
2 $\quad i \leftarrow m$
3 $\quad$ **enquanto** $i > 1$ e $A[\lfloor i/2 \rfloor] < A[i]$ **faça**
4 $\quad\quad A[\lfloor i/2 \rfloor] \leftrightarrow A[i]$ $\quad \triangleright$ troca
5 $\quad\quad i \leftarrow \lfloor i/2 \rfloor$

Qual a relação invariante no início de cada iteração do bloco de linhas 2–5? Qual o consumo de tempo do algoritmo?

Exercício 13.C [CLRS 6.5-5]
Prove que HEAP-I NCREASE-KEY está correto. Use o seguinte invariante: no início de cada iteração, $A[1\,..\,m]$ é um max-heap exceto talvez pela violação da relação $A[\lfloor i/2 \rfloor] \geq A[i]$.

AULA 9

Quicksort

CLRS 7

Partição

Problema: Rearranjar um dado vetor $A[p\mathinner{.\,.}r]$ e devolver um índice q, $p \le q \le r$, tais que

$$A[p\mathinner{.\,.}q-1] \le A[q] < A[q+1\mathinner{.\,.}r]$$

Entra:

	p									r
A	99	33	55	77	11	22	88	66	33	44

Partição

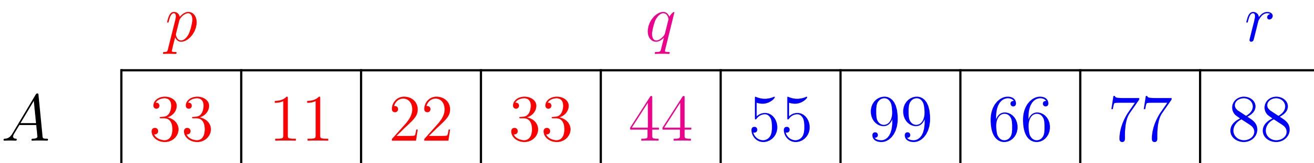
Problema: Rearranjar um dado vetor $A[p\,..\,r]$ e devolver um índice q, $p \leq q \leq r$, tais que

$$A[p\,..\,q{-}1] \leq A[q] < A[q{+}1\,..\,r]$$

Entra:

	p									r
A	99	33	55	77	11	22	88	66	33	44

Sai:



Particione

	p									r
A	99	33	55	77	11	22	88	66	33	44

Particione

	i	j									x
A		99	33	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i			j						x
A	33	99	55	77	11	22	88	66	33	44

Particione

	i		j								x
		A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i				j					x
A	33	99	55	77	11	22	88	66	33	44

Particione

	i	j								x
A		99	33	55	77	11	22	88	66	33	44

	i	j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

		i		j						x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

Particione

	i		j							x
A		99	33	55	77	11	22	88	66	33	44

	i	j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

Particione

	i	j								x
A		99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i						j	x
A	33	11	22	77	99	55	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i	j							x
A	33	99	55	77	11	22	88	66	33	44

Positions: i above 33 (1st), j above 55 (3rd), x above 44.

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

Particione

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

```
PARTICIONE (A, p, r)    ▷ supõe p ≤ r
1     x ← A[r]          ▷ x é o "pivô"
2     i ← p−1
3     para j ← p até r − 1 faça
4          se A[j] ≤ x
5               então i ← i + 1
6                     A[i] ↔ A[j]
7     A[i+1] ↔ A[r]
8     devolva i + 1
```

Invariantes: no começo de cada iteração de 3–6,

(i0) $A[p\,..\,i] \le x$ (i1) $A[i{+}1\,..\,j{-}1] > x$ (i2) $A[r] = x$

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha
1-2	$= 2\,\Theta(1)$
3	$= \Theta(n)$
4	$= \Theta(n)$
5-6	$= 2\,\mathrm{O}(n)$
7-8	$= 2\,\Theta(1)$
total	$= \Theta(2n+4) + \mathrm{O}(2n) \quad = \Theta(n)$

Conclusão

O algoritmo PARTICIONE consome tempo $\Theta(n)$.

Quicksort

Rearranja $A[p \mathinner{.\,.} r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q - 1)$
4 QUICKSORT $(A, q + 1, r)$

	p									r
A	99	33	55	77	11	22	88	66	33	44

Quicksort

Rearranja $A[p\,..\,r]$ em ordem crescente.

```
QUICKSORT (A, p, r)
1    se p < r
2         então q ← PARTICIONE (A, p, r)
3              QUICKSORT (A, p, q − 1)
4              QUICKSORT (A, q + 1, r)
```

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

No começo da linha 3,

$$A[p\,..\,q-1] \leq A[q] < A[q+1\,..\,r]$$

Quicksort

Rearranja $A[p \,..\, r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$

---

4 QUICKSORT $(A, q+1, r)$

	p				q					r
A	11	22	33	33	44	55	88	66	77	99

Quicksort

Rearranja $A[p \,..\, r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$
4 QUICKSORT $(A, q+1, r)$

	p				q					r
A	11	22	33	33	44	55	66	77	88	99

Quicksort

Rearranja $A[p \mathinner{.\,.} r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$
4 QUICKSORT $(A, q+1, r)$

No começo da linha 3,

$$A[p \mathinner{.\,.} q{-}1] \leq A[q] < A[q{+}1 \mathinner{.\,.} r]$$

Consumo de tempo?

Quicksort

Rearranja $A[p \mathinner{.\,.} r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$
4 QUICKSORT $(A, q+1, r)$

No começo da linha 3,

$$A[p \mathinner{.\,.} q{-}1] \leq A[q] < A[q{+}1 \mathinner{.\,.} r]$$

Consumo de tempo?

$T(n) :=$ consumo de tempo no pior caso sendo

$$n := r - p + 1$$

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha		consumo de todas as execuções da linha
1	$=$	?
2	$=$	?
3	$=$	?
4	$=$	?
total	$=$	????

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha		consumo de todas as execuções da linha
1	$=$	$\Theta(1)$
2	$=$	$\Theta(n)$
3	$=$	$T(k)$
4	$=$	$T(n-k-1)$
total	$=$	$T(k) + T(n-k-1) + \Theta(n+1)$

$$0 \leq k := q - p \leq n - 1$$

Recorrência

$T(n)$:= consumo de tempo máximo quando $n = r - p + 1$

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

$$T(n) = T(k) + T(n - k - 1) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Recorrência

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = T(k) + T(n - k - 1) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Recorrência grosseira:

$$T(n) = T(0) + T(n - 1) + \Theta(n)$$

$T(n)$ é $\Theta(???)$.

Recorrência

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = T(k) + T(n-k-1) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

Recorrência grosseira:

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

$T(n)$ é $\Theta(n^2)$.

Demonstração: ...

Recorrência cuidadosa

$T(n) :=$ consumo de tempo máximo quando $n = r - p + 1$

$T(0) = \Theta(1)$

$T(1) = \Theta(1)$

$T(n) = \max_{0 \leq k \leq n-1} \{T(k) + T(n-k-1)\} + \Theta(n)$ para $n = 2, 3, 4, \ldots$

Exemplo concreto

$S(0) = 1$

$S(1) = 1$

$S(n) = \max_{0 \le k \le n-1} \{S(k) + S(n-k-1)\} + n$ para $n = 2, 3, 4, \ldots$

n	0	1	2	3	4	5
$S(n)$	1	1	$2+2$	$5+3$	$9+4$	$14+5$

Exemplo concreto

$S(0) = 1$

$S(1) = 1$

$S(n) = \max_{0 \leq k \leq n-1} \{S(k) + S(n-k-1)\} + n$ para $n = 2, 3, 4, \ldots$

n	0	1	2	3	4	5
$S(n)$	1	1	$2+2$	$5+3$	$9+4$	$14+5$

Vou mostrar que $S(n) \leq n^2 + 1$ para $n \geq 0$.

Demonstração

Prova: Trivial para $n \leq 1$. Se $n \geq 2$ então

$$
\begin{aligned}
S(n) &= \max_{0 \leq k \leq n-1} \left\{ S(k) + S(n-k-1) \right\} + n \\
&\overset{\text{hi}}{\leq} \max_{0 \leq k \leq n-1} \left\{ k^2 + 1 + (n-k-1)^2 + 1 \right\} + n \\
&= (n-1)^2 + 2 + n \qquad \triangleright \text{ exerc 14.E} \\
&= n^2 - n + 3 \\
&\leq n^2 + 1 \,.
\end{aligned}
$$

Prove que $S(n) \geq \frac{1}{2}\, n^2$ para $n \geq 1$.

Algumas conclusões

$T(n)$ é $\Theta(n^2)$.

O consumo de tempo do QUICKSORT no pior caso é $\mathrm{O}(n^2)$.

O consumo de tempo do QUICKSORT é $\mathrm{O}(n^2)$.

Quicksort no melhor caso

$M(n) :=$ consumo de tempo mínimo quando $n = r - p + 1$

$$M(0) = \Theta(1)$$

$$M(1) = \Theta(1)$$

$$M(n) = \min_{0 \le k \le n-1} \{M(k) + M(n-k-1)\} + \Theta(n) \quad \text{para } n = 3, 4, \ldots$$

Quicksort no melhor caso

$M(n) :=$ consumo de tempo mínimo quando $n = r - p + 1$

$$M(0) = \Theta(1)$$

$$M(1) = \Theta(1)$$

$$M(n) = \min_{0 \le k \le n-1} \{M(k) + M(n-k-1)\} + \Theta(n) \quad \text{para } n = 3, 4, \ldots$$

Mostre que, para $n \ge 1$,

$$M(n) \ge \frac{(n-1)}{2} \lg \frac{n-1}{2}.$$

Isto implica que no melhor caso o QUICKSORT é $\Omega(n \lg n)$.

Que é o mesmo que dizer que o QUICKSORT é $\Omega(n \lg n)$.

Quicksort no melhor caso

No melhor caso k é aproximadamente $(n-1)/2$.

$$R(n) = R(\left\lfloor \frac{n-1}{2} \right\rfloor) + R(\left\lceil \frac{n-1}{2} \right\rceil) + \Theta(n)$$

Solução: $R(n)$ é $\Theta(n \lg n)$.

Mais algumas conclusões

$M(n)$ é $\Theta(n \lg n)$.

O consumo de tempo do QUICKSORT no melhor caso é $\Omega(n \log n)$.

Na verdade ...

O consumo de tempo do QUICKSORT no melhor caso é $\Theta(n \log n)$.

Discussão geral

Pior caso, melhor caso, todos os casos?!?!

Dado um algoritmo $\mathcal{A}$ o que significam as expressões:

- $\mathcal{A}$ é $\mathrm{O}(n^2)$ no pior caso.
- $\mathcal{A}$ é $\mathrm{O}(n^2)$ no melhor caso.
- $\mathcal{A}$ é $\mathrm{O}(n^2)$.
- $\mathcal{A}$ é $\Omega(n^2)$ no melhor caso.
- $\mathcal{A}$ é $\Omega(n^2)$ no pior caso.
- $\mathcal{A}$ é $\Omega(n^2)$

Análise experimental

Algoritmos implementados:

`shell`	`shellsort` original (Knuth).
`merge_r`	MERGE-SORT recursivo.
`merge_i`	MERGE-SORT iterativo.
`heap`	HEAPSORT.
`quick`	QUICKSORT recursivo.
`qsort`	`quicksort` da biblioteca do C.

Compilador: `gcc -O2`.

Computador: Pentium II, 233Mhz, 128Mb.

Estudo empírico (aleatório)

n	shell	merge_r	merge_i	heap	quick	qsort
4096	0.01	0.01	0.00	0.01	0.00	0.01
8192	0.01	0.01	0.01	0.01	0.01	0.03
16384	0.03	0.02	0.02	0.02	0.02	0.06
32768	0.07	0.05	0.05	0.04	0.04	0.12
65536	0.16	0.11	0.11	0.10	0.07	0.25
131072	0.38	0.25	0.33	0.23	0.16	0.54
262144	0.92	0.54	0.73	0.54	0.34	1.15
524288	2.13	1.18	1.55	1.31	0.74	2.44
1048576	4.97	2.52	3.32	3.06	1.60	5.20
2097152	11.38	5.42	6.96	7.07	3.38	10.88
4194304	26.50	11.40	14.46	15.84	7.16	22.78
8388608	61.68	24.35	29.76	35.85	15.24	47.85

Estudo empírico (decrescente)

n	shell	merge_r	merge_i	heap	quick	qsort
4096	0.00	0.01	0.00	0.00	0.22	0.01
8192	0.00	0.01	0.01	0.01	0.90	0.02
16384	0.02	0.02	0.02	0.01	3.70	0.04
32768	0.03	0.04	0.04	0.03	15.29	0.08
65536	0.07	0.09	0.09	0.07	62.23	0.18
131072	0.15	0.21	0.28	0.16	261.26	0.40

Estudo empírico (crescente)

n	shell	merge_r	merge_i	heap	quick	qsort
4096	0.01	0.00	0.01	0.00	0.29	0.00
8192	0.00	0.01	0.01	0.01	1.16	0.02
16384	0.01	0.02	0.02	0.01	4.63	0.03
32768	0.02	0.05	0.04	0.03	18.47	0.06
65536	0.05	0.10	0.10	0.07	74.16	0.11
131072	0.12	0.23	0.28	0.16	372.45	0.23

Exercícios

Exercício 14.A
Submeta ao algoritmo PARTICIONE um vetor com n elementos iguais. Como o algoritmo permuta o vetor recebido? Quantas trocas faz (linhas 6 e 7) entre elementos do vetor?

Exercício 14.B [CLRS 7.2-2]
Qual o consumo de tempo do QUICKSORT quando aplicado a um vetor com n elementos iguais?

Exercício 14.C [CLRS 7.2-3]
Mostre que o consumo de tempo do QUICKSORT é $\Omega(n^2)$ quando aplicado a um vetor crescente com n elementos distintos.

Exercício 14.D [CLRS 7.4-1, modificado]
Seja S a função definida sobre os inteiros positivos pela seguinte recorrência:
$S(0) = S(1) = 1$ e

$$S(n) = \max_{0 \leq k \leq n-1} \left\{S(k) + S(n-k-1)\right\} + n$$

quando $n \geq 2$. Mostre que $S(n) \geq \frac{1}{2}n^2$ para $n \geq 1$.

Mais exercícios

Exercício 14.E [CLRS 7.4-3]
Mostre que $k^2 + (n-k-1)^2$ atinge o máximo para $0 \leq k \leq n-1$ quando $k = 0$ ou $k = n-1$.

Exercício 14.F
É verdade que $\lceil 2\lceil 2n/3 \rceil/3 \rceil = \lceil 4n/9 \rceil$? É verdade que $\lfloor 2\lfloor 2n/3 \rfloor/3 \rfloor = \lfloor 4n/9 \rfloor$?

Exercício 14.G [CLRS 7-4]
Considere a seguinte variante do algoritmo Quicksort:

QUICKSORT′ (A, p, r)
  enquanto $p < r$ faça
    $q \leftarrow$ PARTICIONE (A, p, r)
    QUICKSORT′ $(A, p, q-1)$
    $p \leftarrow q + 1$

Mostre que a pilha de recursão pode atingir altura proporcional a n, onde $n := r - p + 1$.
Modifique o código de modo que a pilha de recursão tenha altura $\mathrm{O}(\lg n)$. (Veja enunciado completo em CLRS p.162.)

AULA 11

Análise probabilística

CLRS 5.1, C.2, C.3

Máximo

Problema: Encontrar o elemento máximo de um vetor $A[1\,..\,n]$ de números inteiros positivos distintos.

MAX (A, n)
1 $\quad max \leftarrow 0$
2 $\quad$ **para** $i \leftarrow 1$ **até** n **faça**
3 $\quad\quad$ **se** $A[i] > max$
4 $\quad\quad\quad$ **então** $max \leftarrow A[i]$
5 $\quad$ **devolva** max

Quantas vezes linha 4 é executada?

Máximo

Problema: Encontrar o elemento máximo de um vetor $A[1\,..\,n]$ de números inteiros positivos distintos.

MAX (A, n)
1 $\quad max \leftarrow 0$
2 $\quad$ **para** $i \leftarrow 1$ **até** n **faça**
3 $\quad\quad$ **se** $A[i] > max$
4 $\quad\quad\quad$ **então** $max \leftarrow A[i]$

---

5 $\quad$ **devolva** max

Quantas vezes linha 4 é executada?
Melhor caso, pior caso, caso médio?

Máximo

Problema: Encontrar o elemento máximo de um vetor $A[1\,..\,n]$ de números inteiros positivos distintos.

```
MAX (A, n)
1    max ← 0
2    para i ← 1 até n faça
3         se A[i] > max
4              então max ← A[i]
5    devolva max
```

Quantas vezes linha 4 é executada?
Melhor caso, pior caso, caso médio?

Suponha que $A[1\,..\,n]$ é permutação aleatória uniforme de $1, \ldots, n$

Cada permutação tem probabilidade $1/n!$.

Um pouco de probabilidade

$(S, \Pr)$ espaço de probabilidade
$S =$ conjunto finito (eventos elementares)
$\Pr\{\} =$ (distribuição de probabilidades) função de S em $[0, 1]$ tal que

p1. $\Pr\{s\} \geq 0$;

p2. $\Pr\{S\} = 1$; e

p3. $R, T \subseteq S, R \cap T = \emptyset \Rightarrow \Pr\{R \cup T\} = \Pr\{R\} + \Pr\{T\}$.

$$\Pr\{U\} \text{ é abreviação de } \sum_{u \in U} \Pr\{u\}.$$

Um pouco de probabilidade

(S, Pr) espaço de probabilidade
$S =$ conjunto finito (eventos elementares)
$\mathrm{Pr}\{\} =$ (distribuição de probabilidades) função de S em $[0, 1]$ tal que

p1. $\mathrm{Pr}\{s\} \geq 0$;

p2. $\mathrm{Pr}\{S\} = 1$; e

p3. $R, T \subseteq S, R \cap T = \emptyset \Rightarrow \mathrm{Pr}\{R \cup T\} = \mathrm{Pr}\{R\} + \mathrm{Pr}\{T\}$.

$$\mathrm{Pr}\{U\} \text{ é abreviação de } \sum_{u \in U} \mathrm{Pr}\{u\}.$$

No problema do máximo:

- S é o conjunto das permutações dos números em $A[1 \mathinner{..} n]$;
- na distribuição uniforme, para cada $s \in S$, $\mathrm{Pr}\{s\} = 1/n!$.

Eventos

Um evento é um subconjunto de S.

Eventos

Um evento é um subconjunto de S.

No problema do máximo, eventos são subconjuntos de permutações de $A[1\,..\,n]$.

Exemplo.

$$U := \{\text{permutações de } A[1\,..\,n] \text{ em que } A[n] \text{ é máximo}\}$$

é um evento de S.

Eventos

Um evento é um subconjunto de S.

No problema do máximo, eventos são subconjuntos de permutações de $A[1\,..\,n]$.

Exemplo.

$$U := \{\text{permutações de } A[1\,..\,n] \text{ em que } A[n] \text{ é máximo}\}$$

é um evento de S.

Se $\Pr\{\}$ é distribuição uniforme, então

$$\Pr\{U\} = ???.$$

Eventos

Um evento é um subconjunto de S.

No problema do máximo, eventos são subconjuntos de permutações de $A[1\,..\,n]$.

Exemplo.

$$U := \{\text{permutações de } A[1\,..\,n] \text{ em que } A[n] \text{ é máximo}\}$$

é um evento de S.

Se $\Pr\{\}$ é distribuição uniforme, então

$$\Pr\{U\} = 1/n.$$

Probabilidade condicional

A probabilidade condicional de ocorrer um evento E ocorrer dado que um evento F ocorreu é

$$\Pr\{E \mid F\} = \frac{\Pr\{E \cap F\}}{\Pr\{F\}}.$$

A expressão acima só faz sentido quando $\Pr\{F\} \neq 0$.

$\Pr\{E \mid F\}$ lê-se "A probabilidade de E dado F".

Intuitivamente, estamos interessados na probabilidade de ocorrer $E \cap F$ dentro de F. Como F define a nossa restrição do espaço, normalizamos as probabilidades dividindo por $\Pr\{F\}$.

Rescrevendo, temos que

$$\Pr\{E \cap F\} = \Pr\{F\} \times \Pr\{E \mid F\}.$$

Probabilidade condicional (exemplo)

$E :=$ {permutações de $A[1\,..\,4]$ em que $A[1] = 1$ e $A[2] = 2$}

$F :=$ {permutações de $A[1\,..\,4]$ em que $A[1] = 1$}

$\Pr\{E \cap F\} = \Pr\{E\} = 2!/4! = 1/12$

$\Pr\{F\} = 3!/4! = 1/4$

Logo,

$$\Pr\{E \mid F\} = \frac{\Pr\{E \cap F\}}{\Pr\{F\}} = \frac{\frac{1}{12}}{\frac{1}{4}} = \frac{1}{3}.$$

Verifique que $1/3$ das permutações de $A[1\,..\,4]$ com $A[1] = 1$ têm $A[2] = 2$.

Variáveis aleatórias e esperança

Uma variável aleatória é uma função númerica definida sobre os eventos elementares.

Variáveis aleatórias e esperança

Uma variável aleatória é uma função númerica definida sobre os eventos elementares.

Exemplo de variável aleatória

$X(A) :=$ número de execuções da linha 4 em MAX(A, n)

Variáveis aleatórias e esperança

Uma variável aleatória é uma função númerica definida sobre os eventos elementares.

Exemplo de variável aleatória

$X(A) :=$ número de execuções da linha 4 em MAX(A, n)

"$X = k$" é uma abreviação de $\{s \in S : X(s) = k\}$

Esperança $\mathrm{E}[X]$ de uma variável aleatória X

$$\mathrm{E}[X] = \sum_{k \in X(S)} k \cdot \Pr\{X = k\} = \sum_{s \in S} X(s) \cdot \Pr\{s\}$$

Variáveis aleatórias e esperança

Uma variável aleatória é uma função númerica definida sobre os eventos elementares.

Exemplo de variável aleatória

$X(A) :=$ número de execuções da linha 4 em MAX(A, n)

"$X = k$" é uma abreviação de $\{s \in S : X(s) = k\}$

Esperança $\mathrm{E}[X]$ de uma variável aleatória X

$$\mathrm{E}[X] = \sum_{k \in X(S)} k \cdot \Pr\{X = k\} = \sum_{s \in S} X(s) \cdot \Pr\{s\}$$

Linearidade da esperança: $\mathrm{E}[\alpha X + Y] = \alpha \mathrm{E}[X] + \mathrm{E}[Y]$

De volta ao máximo

Problema: Encontrar o elemento máximo de um vetor $A[1\,.\,.\,n]$ de números inteiros positivos distintos.

```
MAX (A, n)
1    max ← 0
2    para i ← 1 até n faça
3        se A[i] > max
4            então max ← A[i]
5    devolva max
```

Quantas vezes linha 4 é executada no caso médio?

Suponha que $A[1\,.\,.\,n]$ é permutação aleatória uniforme de $1, \ldots, n$

Cada permutação tem probabilidade $1/n!$.

Exemplos

$A[1\,..\,2]$	linha 4
1,2	2
2,1	1
$\mathrm{E}[X]$	3/2

$A[1\,..\,3]$	linha 4
1,2,3	3
1,3,2	2
2,1,3	2
2,3,1	2
3,1,2	1
3,2,1	1
$\mathrm{E}[X]$	11/6

Mais um exemplo

$A[1\,..\,4]$	linha 4
1,2,3,4	4
1,2,4,3	3
1,3,2,4	3
1,3,4,2	3
1,4,2,3	2
1,4,3,2	2
2,1,3,4	3
2,1,4,3	2
2,3,1,4	3
2,3,4,1	3
2,4,1,3	2
2,4,3,1	2

$A[1\,..\,4]$	linha 4
3,1,2,4	2
3,1,4,2	2
3,2,1,4	2
3,2,4,1	2
3,4,1,2	2
3,4,2,1	2
4,1,2,3	1
4,1,3,2	1
4,2,1,3	1
4,2,3,1	1
4,3,1,2	1
4,3,2,1	1
$\mathrm{E}[X]$	50/24

Variáveis aleatórias indicadoras

X = número total de execuções da linha 4

Variáveis aleatórias indicadoras

$$X = \text{número total de execuções da linha 4}$$

$$X_i = \begin{cases} 1 \text{ se "}max \leftarrow A[i]\text{" é executado} \\ 0 \text{ caso contrário} \end{cases}$$

$$\begin{aligned} X &= \text{número total de execuções da linha 4} \\ &= X_1 + \cdots + X_n \end{aligned}$$

Variáveis aleatórias indicadoras

$$X = \text{número total de execuções da linha 4}$$

$$X_i = \begin{cases} 1 \text{ se "} max \leftarrow A[i] \text{" é executado} \\ 0 \text{ caso contrário} \end{cases}$$

$$\begin{aligned} X &= \text{número total de execuções da linha 4} \\ &= X_1 + \cdots + X_n \end{aligned}$$

Esperanças:

$$\begin{aligned} \mathrm{E}[X_i] &= 0 \times \Pr\{X_i = 0\} + 1 \times \Pr\{X_i = 1\} \\ &= \Pr\{X_i = 1\} \\ &= \text{probabilidade de que } A[i] \text{ seja} \\ &\quad\ \text{máximo em } A[1 \mathinner{.\,.} i] \\ &= 1/i \end{aligned}$$

Esperança

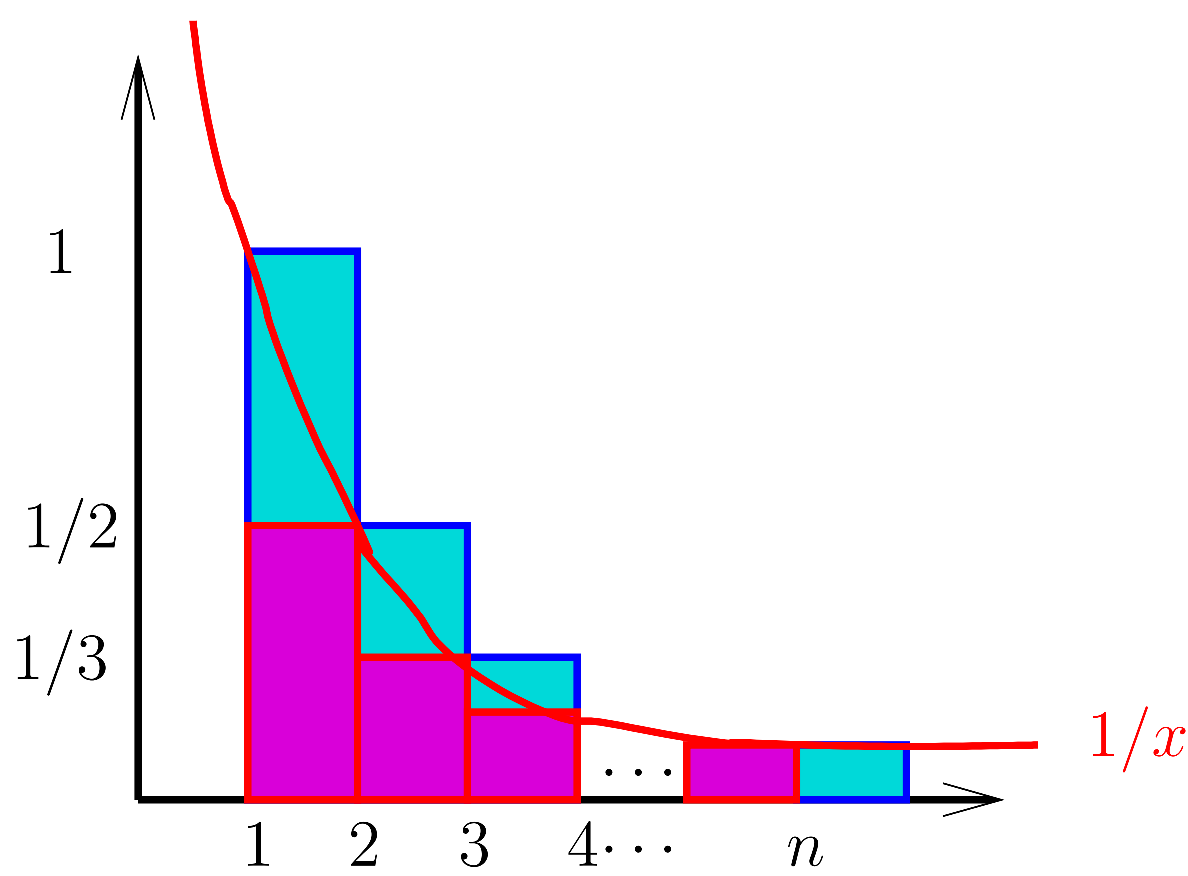
$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}[X_1 + \cdots + X_n] \\ &= \mathrm{E}[X_1] + \cdots + \mathrm{E}[X_n] \\ &= 1/1 + \cdots + 1/n \\ &< 1 + \ln n \\ &= \mathrm{O}(\lg n) \end{aligned}$$

$$2.92 < \frac{1}{1} + \cdots + \frac{1}{10} < 2.93 < 3.30 < 1 + \ln 10$$

$$5.18 < \frac{1}{1} + \cdots + \frac{1}{100} < 5.19 < 5.60 < 1 + \ln 100$$

$$9.78 < \frac{1}{1} + \cdots + \frac{1}{10000} < 9.79 < 10.21 < 1 + \ln 10000$$

Série harmônica



$$\ln n = \int_1^n \frac{dx}{x} < H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} < 1 + \int_1^n \frac{dx}{x} = 1 + \ln n$$

Experimentos

Para cada valor de $n = 252, 512, 1024, \ldots$ foram geradas 10, 100 ou 200 amostras de seqüencias de inteiros através do trecho de código

```
for (i = 0; i < n; i++){
  v[i]=(int)((double)INT_MAX*rand()/(RAND_MAX+1
}
```

onde `rand()` é a função geradora de números (pseudo-)aleatórios da biblioteca do `C`.

A coluna $\mathrm{E}[\hat{X}]$ nas tabelas a seguir mostra o número médio de vezes que a linha 4 do algoritmo MAX foi executada para cada valor de n e cada amostra de seqüências.

Experimentos (10)

n	$E[\hat{X}]$	$1 + \ln n$
256	7.20	6.55
512	6.90	7.24
1024	7.30	7.93
2048	7.10	8.62
4096	10.20	9.32
8192	9.00	10.01
16384	10.80	10.70
32768	11.00	11.40
65536	12.50	12.09
131072	12.60	12.78
262144	13.20	13.48
524288	13.20	14.17
1048576	12.80	14.86
2097152	13.90	15.56
4194304	14.90	16.25
8388608	17.90	16.94

Experimentos (100)

n	$E[\hat{X}]$	$1 + \ln n$
256	5.92	6.55
512	6.98	7.24
1024	7.55	7.93
2048	8.39	8.62
4096	8.97	9.32
8192	9.26	10.01
16384	10.44	10.70
32768	11.32	11.40
65536	11.66	12.09
131072	12.38	12.78
262144	13.17	13.48
524288	13.56	14.17
1048576	14.54	14.86
2097152	15.10	15.56
4194304	15.61	16.25
8388608	16.56	16.94

Experimentos (200)

n	$E[\hat{X}]$	$1 + \ln n$
256	6.12	6.55
512	6.86	7.24
1024	7.38	7.93
2048	7.96	8.62
4096	8.87	9.32
8192	9.41	10.01
16384	10.28	10.70
32768	10.92	11.40
65536	11.31	12.09
131072	12.37	12.78
262144	12.92	13.48
524288	13.98	14.17
1048576	14.19	14.86
2097152	15.62	15.56
4194304	15.74	16.25
8388608	17.06	16.94

Resumo de probabilidade

$(S, \Pr)$ espaço de probabilidade, $S =$ conjunto finito (eventos elementares)
$\Pr\{\} =$ (distribuição de probabilidades) função de S em $[0, 1]$ tal que

p1. $\Pr\{s\} \geq 0$;

p2. $\Pr\{S\} = 1$; e

p3. $A, B \subseteq S, A \cap B = \emptyset \Rightarrow \Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\}$.

Um evento é um subconjunto de S.

Uma variável aleatória é uma função númerica definida sobre os eventos elementares.

"$X = k$" é uma abreviação de $\{s \in S : X(s) = k\}$

$E[X] = \sum_{k \in X(S)} k \cdot \Pr\{X = k\} = \sum_{s \in S} X(s) \cdot \Pr\{s\}$

Algoritmos aleatorizados ou probabilísticos

CLRS 5.2, 5.3, C.2, C.3

Algoritmos aleatorizados ou probabilísticos

Um algoritmo é probabilístico ou aleatorizado se o seu comportamento não é determinado apenas pela entrada, mas também depende dos valores produzidos por um gerador de números aleatórios.

RANDOM(a, b): devolve um inteiro i, $a \leq i \leq b$.

Algoritmos aleatorizados ou probabilísticos

Um algoritmo é probabilístico ou aleatorizado se o seu comportamento não é determinado apenas pela entrada, mas também depende dos valores produzidos por um gerador de números aleatórios.

RANDOM(a, b): devolve um inteiro i, $a \leq i \leq b$.

Todos os inteiros devolvido por RANDOM são independentes dos inteiros previamente devolvidos com probabilidade uniforme, em outras palavras:

$$\Pr\{\text{RANDOM}(a, b) = i\} = ???.$$

Algoritmos aleatorizados ou probabilísticos

Um algoritmo é probabilístico ou aleatorizado se o seu comportamento não é determinado apenas pela entrada, mas também depende dos valores produzidos por um gerador de números aleatórios.

RANDOM(a,b): devolve um inteiro i, $a \leq i \leq b$.

Todos os inteiros devolvido por RANDOM são independentes dos inteiros previamente devolvidos com probabilidade uniforme, em outras palavras:

$$\Pr\{\text{RANDOM}(a,b) = i\} = \frac{1}{b-a+1}.$$

Existem implementações aproximadas de RANDOM. Veja

D.E. Knuth, *Seminumerical algorithms*, volume 2, *The Art of Computer Programming*.

Permutação aleatória uniforme

Problema: Obter uma permutação aleatória uniforme de um dado vetor $A[1\,..\,n]$ de números inteiros positivos distintos.

Permutação *in-place*

Devolve uma permutação aleatória uniforme de $A[1\,.\,.\,n]$.

PERMUTE-IN-PLACE(A, n)
1 para $i \leftarrow 1$ até $n - 1$ faça
2 $\quad j \leftarrow$ RANDOM(i, n)
3 $\quad A[i] \leftrightarrow A[j]$

Não é óbvio que isso produz permutação aleatória uniforme de $A[1\,.\,.\,n]$.

Consumo de tempo: $\Theta(n)$, supondo que RANDOM é $\Theta(1)$.

Uniformidade

Considere uma permutação $B[1\,..\,n]$ de $A[1\,..\,n]$.

Qual é a probabilidade de PERMUTE-IN-PLACE(A, n) produzir $B[1\,..\,n]$?

Uniformidade

Considere uma permutação $B[1\,..\,n]$ de $A[1\,..\,n]$.

Qual é a probabilidade de PERMUTE-IN-PLACE(A, n) produzir $B[1\,..\,n]$?

Relação invariante: Na linha 1 vale que

(i0) $\Pr\{B[1\,..\,i-1] = A[1\,..\,i-1]\} = (n-i+1)!/n!$.

Uniformidade

Considere uma permutação $B[1\,..\,n]$ de $A[1\,..\,n]$.

Qual é a probabilidade de PERMUTE-IN-PLACE(A, n) produzir $B[1\,..\,n]$?

Relação invariante: Na linha 1 vale que

(i0) $\Pr\{B[1\,..\,i-1] = A[1\,..\,i-1]\} = (n-i+1)!/n!$.

Na última iteração $i = n$ e da invariante (i0) temos que

$$\Pr\{B[1\,..\,n] = A[1\,..\,n]\} = (n-i+1)!/n! = 1/n!.$$

A invariante vale no início da primeira iteração já que

$$\Pr\{B[1\,..\,0] = A[1\,..\,0]\} = 1.$$

Uniformidade (cont.)

Considere uma iteração que não seja a última. Da invariante (i0) sabemos que

$$\Pr\{B[1\,..\,i-1] = A[1\,..\,i-1]\} = (n-i+1)!/n!$$

Se $B[1\,..\,i-1] = A[1\,..\,i-1]$, então, após a execução da linha 2 temos que

$$\Pr\{B[i] = A[j]\} = \frac{1}{n-i+1}\ .$$

Logo, após a execução da linha 3 temos que

$$\Pr\{B[1\,..\,i] = A[1\,..\,i]\} = \frac{(n-i+1)!}{n!} \times \frac{1}{n-i+1} = \frac{(n-i)!}{n!}.$$

Portanto, (i0) vale no início da próxima iteração.

Uniformidade (cont.)

Reescrevendo . . . Considere uma iteração que não seja . . .

$E :=$ {após a execução da linha 2 $B[i] = A[j]$}

$F :=$ {no início da iteração $B[1\,..\,i-1] = A[1\,..\,i-1]$}

Após a execução da linha 3, ao final da iteração, vale que

$$\begin{aligned}
\Pr\{B[1\,..\,i] = A[1\,..\,i]\} &= \Pr\{E \cap F\} \\
&= \Pr\{F\} \times \Pr\{E \mid F\} \\
&= \frac{(n-i+1)!}{n!} \times \frac{1}{n-i+1} \\
&= \frac{(n-i)!}{n!}
\end{aligned}$$

Portanto, (i0) vale no início da próxima iteração.

Permutação por ordenação

Devolve uma permutação aleatória uniforme de $A[1\,..\,n]$.

PERMUTE-POR-ORDENAÇÃO (A, n)
1 para $i \leftarrow 1$ até n faça
2 $P[i] \leftarrow$ RANDOM$(1, n^3)$
3 ordene $A[1\,..\,n]$ com chaves $P[1\,..\,n]$

No começo da linha 3, com grande probabilidade, $P[1\,..\,n]$ não tem elementos repetidos (Exercício 5.3-5 do CLRS).

Linha 3 faz permutação $A[j_1\,..\,j_n]$ de $A[1\,..\,n]$ tal que

$$P[j_1] \leq \cdots \leq P[j_n].$$

Consumo de tempo: $\Theta(n \lg n)$ se RANDOM for $\Theta(1)$

Max aleatorizado

Devolve o elemento máximo de um vetor $A[1\,..\,n]$.

MAX-ALEATORIZADO(A, n)
0 PERMUTE(A, n)
1 $max \leftarrow 0$
2 **para** $i \leftarrow 1$ **até** n **faça**
3   **se** $A[i] > max$
4   **então** $max \leftarrow A[i]$
5 **devolva** max

Linha 0 faz uma permutação aleatória uniforme dos elementos de A.

Qual é a entrada que dá o pior caso?

Exercícios

Exercício 15.A [CLRS 5.3-3]
Que acontece se trocarmos "RANDOM(i, n)" por "RANDOM$(1, n)$" no algoritmo PERMUTE-IN-PLACE? O algoritmo continua produzindo permutação aleatória uniforme?

Exercício 15.B [CLRS C.3-2]
Um vetor $A[1\,..\,n]$ contém n números distintos aleatoriamente ordenados. Suponha que cada permutação dos n números é igualmente provável. Qual é a esperança do índice do maior elemento do vetor? Qual é a esperança do índice do menor elemento do vetor?

AULA 12

Quicksort aleatorizado

CLRS 7.4

Relembrar ...

Antes de mais nada, vamos relembrar rapidamente os melhores momentos da aula 9:

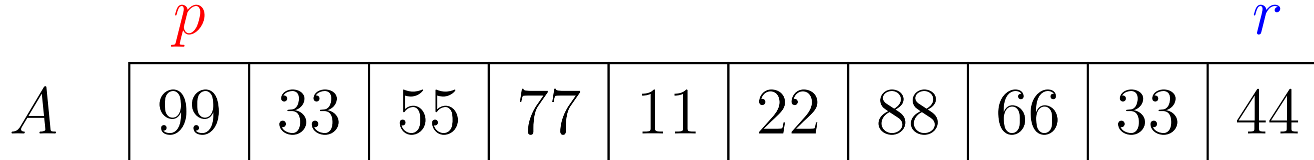
- PARTICIONE;
- consumo de tempo do PARTICIONE;
- QUICKSORT; e
- consumo de tempo do QUICKSORT.

Partição

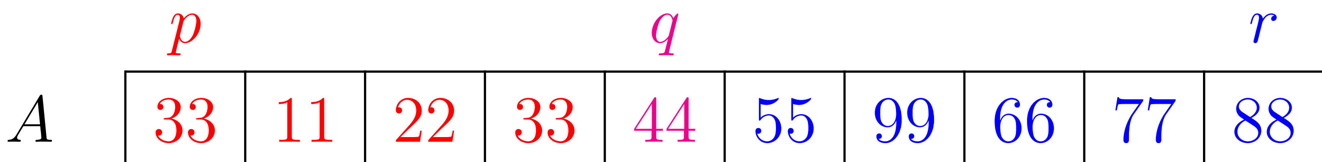
Problema: Rearranjar um dado vetor $A[p\,..\,r]$ e devolver um índice q, $p \le q \le r$, tais que

$$A[p\,..\,q-1] \le A[q] < A[q+1\,..\,r]$$

Entra:



Sai:



Particione

	p									r
A	99	33	55	77	11	22	88	66	33	44

Particione

	i	j									x
A		99	33	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i			j						x
A	33	99	55	77	11	22	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i	j							x
A	33	99	55	77	11	22	88	66	33	44

	i				j				x
A	33	99	55	77	11	22	88	66	33	44

Particione

i (before position 1), j (position 1), x (position 10)

A	99	33	55	77	11	22	88	66	33	44

i (position 1), j (position 3), x (position 10)

A	33	99	55	77	11	22	88	66	33	44

i (position 2), j (position 6), x (position 10)

A	33	11	55	77	99	22	88	66	33	44

Particione

	i	j								x
A		99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

Particione

	i	j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i						j	x
A	33	11	22	77	99	55	88	66	33	44

Particione

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

Particione

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

PARTICIONE (A, p, r)
1 $x \leftarrow A[r]$ $\triangleright$ x é o "pivô"
2 $i \leftarrow p{-}1$
3 **para** $j \leftarrow p$ **até** $r - 1$ **faça**
4 **se** $A[j] \le x$
5 **então** $i \leftarrow i + 1$
6 $A[i] \leftrightarrow A[j]$
7 $A[i{+}1] \leftrightarrow A[r]$
8 **devolva** $i + 1$

O algoritmo PARTICIONE consome tempo $\Theta(n)$.

Quicksort

Rearranja $A[p \,.\,.\, r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$
4 QUICKSORT $(A, q+1, r)$

	p									r
A	99	33	55	77	11	22	88	66	33	44

Quicksort

Rearranja $A[p \mathinner{.\,.} r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)

---

3 QUICKSORT $(A, p, q-1)$
4 QUICKSORT $(A, q+1, r)$

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

No começo da linha 3,

$$A[p \mathinner{.\,.} q-1] \leq A[q] \leq A[q+1 \mathinner{.\,.} r]$$

Quicksort

Rearranja $A[p \,..\, r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$

---

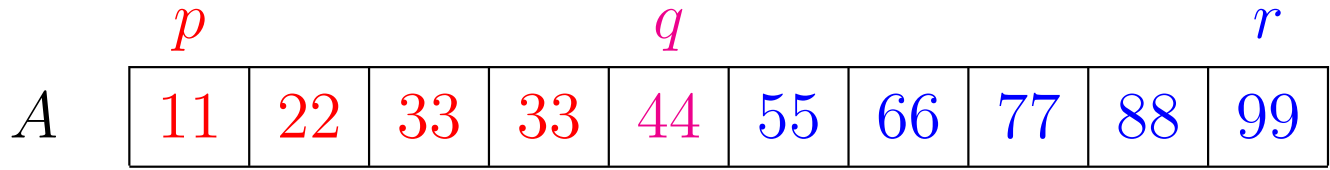
4 QUICKSORT $(A, q+1, r)$

	p				q					r
A	11	22	33	33	44	55	88	66	77	99

Quicksort

Rearranja $A[p \mathinner{.\,.} r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$
4 QUICKSORT $(A, q+1, r)$



O consumo de tempo é proporcional ao número de execuções da linha 4 do PARTICIONE.

Resumo

O consumo de tempo do QUICKSORT é $O(n^2)$.

O consumo de tempo do QUICKSORT no melhor caso é $\Theta(n \log n)$.

Caso médio

O consumo de tempo do QUICKSORT no caso médio é $???$.

Partição $\frac{1}{3}$ para $\frac{2}{3}$:

$$R(n) = R(\lfloor \frac{n-1}{3} \rfloor) + R(\lceil \frac{2n-2}{3} \rceil) + \Theta(n)$$

Solução: $R(n)$ é $\Theta(n \lg n)$. veja exercício a seguir

Exercício

Considere a recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n$$

para $n = 2, 3, 4, \ldots$

Solução assintótica: $T(n)$ é $\mathrm{O}(???)$, $T(n)$ é $\Theta(???)$

Exercício

Considere a recorrência

$$T(1) = 1$$

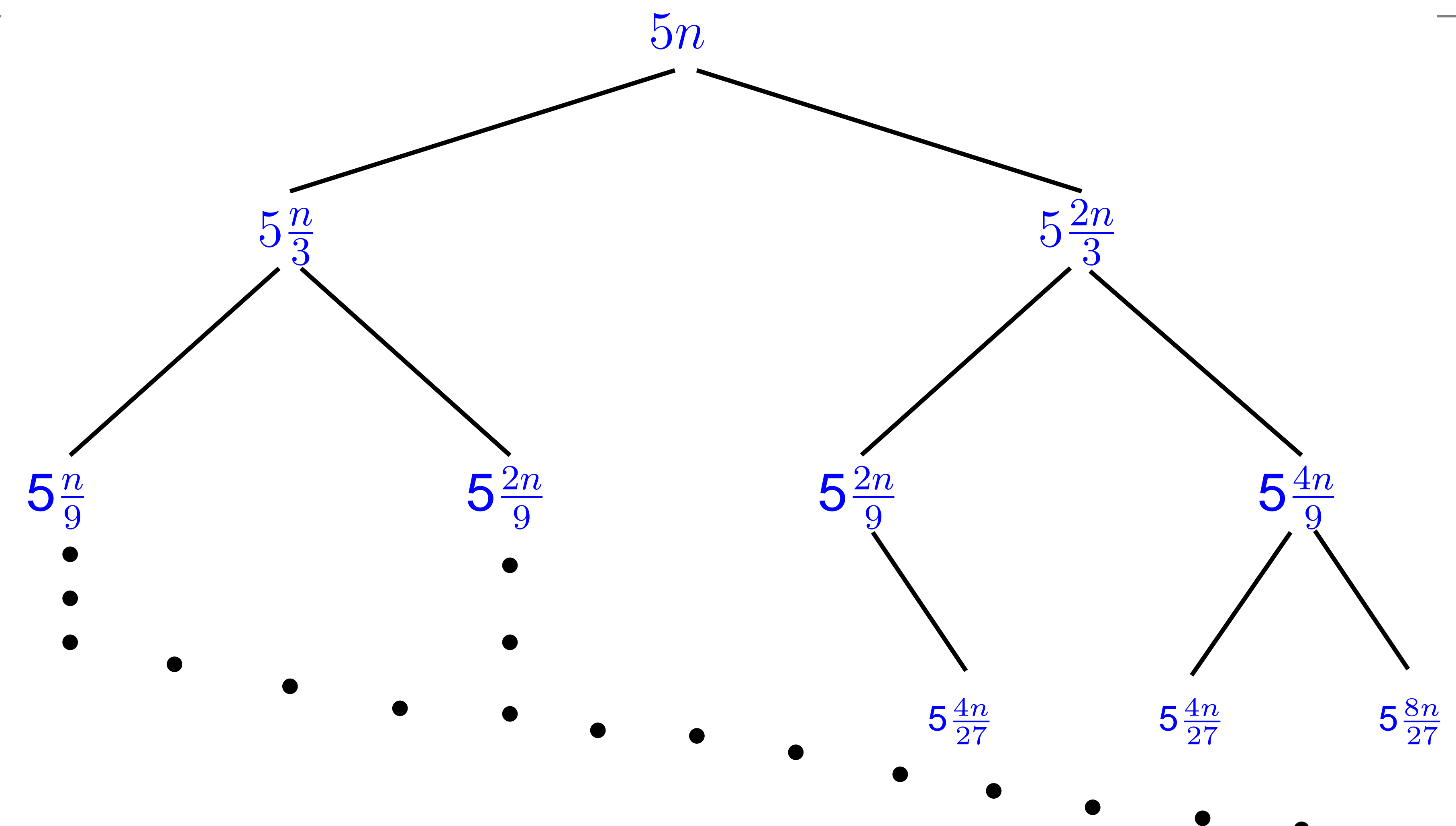
$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n$$

para $n = 2, 3, 4, \ldots$

Solução assintótica: $T(n)$ é $\mathrm{O}(???)$, $T(n)$ é $\Theta(???)$

Vamos olhar a árvore da recorrência.

Arvoré da recorrência



total de níveis $\leq \log_{3/2} n$

Arvoré da recorrência

soma em cada horizontal $\leq 5n$

número de "níveis" $\leq \log_{3/2} n$

$T(n)$ = a soma de tudo

$$T(n) \leq 5n \log_{3/2} n + \underbrace{1 + \cdots + 1}_{\log_{3/2} n}$$

$T(n)$ é $O(n \lg n)$.

De volta a recorrência

$$T(1) = 1$$

$$T(n) = T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + 5n$$

para $n = 2, 3, 4, \ldots$

n	$T(n)$
1	1
2	$1 + 1 + 5 \cdot 2 = 12$
3	$1 + 12 + 5 \cdot 3 = 28$
4	$12 + 12 + 5 \cdot 4 = 44$

Vamos mostrar que $T(n) \leq 100\, n \lg n$ para $n = 2, 3, 4, 5, 6, \ldots$
Para $n = 2$ temos $T(2) = 12 < 100 \cdot 2 \cdot \lg 2$.
Para $n = 3$ temos $T(3) = 28 < 100 \cdot 3 \cdot \lg 3$.

Suponha agora que $n > 3$. Então

Continuação da prova

$$
\begin{aligned}
T(n) &= T\left(\left\lceil \frac{n}{3} \right\rceil\right) + T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + 5n \\
&\overset{\text{hi}}{\leq} 100 \left\lceil \frac{n}{3} \right\rceil \lg \left\lceil \frac{n}{3} \right\rceil + 100 \left\lfloor \frac{2n}{3} \right\rfloor \lg \left\lfloor \frac{2n}{3} \right\rfloor + 5n \\
&\leq 100 \frac{n+2}{3} \left\lceil \lg \frac{n}{3} \right\rceil + 100 \frac{2n}{3} \lg \frac{2n}{3} + 5n \\
&< 100 \frac{n+2}{3} \left(\lg \frac{n}{3} + 1\right) + 100 \frac{2n}{3} \lg \frac{2n}{3} + 5n \\
&= 100 \frac{n+2}{3} \lg \frac{2n}{3} + 100 \frac{2n}{3} \lg \frac{2n}{3} + 5n \\
&= 100 \frac{n}{3} \lg \frac{2n}{3} + 100 \frac{2}{3} \lg \frac{2n}{3} + 100 \frac{2n}{3} \lg \frac{2n}{3} + 5n
\end{aligned}
$$

Continuação da continuação da prova

$$
\begin{aligned}
&< 100n \lg \frac{2n}{3} + 67 \lg \frac{2n}{3} + 5n \\
&= 100n \lg n + 100n \lg \frac{2}{3} + 67 \lg n + 67 \lg \frac{2}{3} + 5n \\
&< 100n \lg n + 100n(-0.58) + 67 \lg n + 67(-0.58) + 5n \\
&< 100n \lg n - 58n + 67 \lg n - 38 + 5n \\
&= 100n \lg n - 53n + 67 \lg n - 38 \\
&< 100n \lg n
\end{aligned}
$$

Iiiéééééssss!

De volta ao caso médio

O consumo de tempo do QUICKSORT no caso médio é ???.

Partição $\frac{1}{10}$ para $\frac{9}{10}$:

$$R(n) = R(\lfloor \frac{n-1}{10} \rfloor) + R(\lceil \frac{9n-9}{10} \rceil) + \Theta(n)$$

De volta ao caso médio

O consumo de tempo do QUICKSORT no caso médio é $???$.

Partição $\frac{1}{10}$ para $\frac{9}{10}$:

$$R(n) = R(\lfloor \tfrac{n-1}{10} \rfloor) + R(\lceil \tfrac{9n-9}{10} \rceil) + \Theta(n)$$

Solução: $R(n)$ é $\Theta(n \lg n)$

De volta ao caso médio

O consumo de tempo do QUICKSORT no caso médio é $???$.

Partição $\frac{1}{10}$ para $\frac{9}{10}$:

$$R(n) = R(\lfloor \tfrac{n-1}{10} \rfloor) + R(\lceil \tfrac{9n-9}{10} \rceil) + \Theta(n)$$

Solução: $R(n)$ é $\Theta(n \lg n)$

Isso sugere que consumo médio é $\Theta(n \lg n)$.

Confirmação?

Exemplos

Número médio de execuções da linha 4 do PARTICIONE.

Suponha que $A[p\,..\,r]$ é permutação de $1\,..\,n$.

$A[p\,..\,r]$	execs
1,2	1
2,1	1
média	1

$A[p\,..\,r]$	execs
1,2,3	2+1
2,1,3	2+1
1,3,2	2+0
3,1,2	2+0
2,3,1	2+1
3,2,1	2+1
média	16/6

Mais um exemplo

$A[p\,.\,.\,r]$	execs
1,2,3,4	3+3
2,1,3,4	3+3
1,3,2,4	3+2
3,1,2,4	3+2
2,3,1,4	3+3
3,2,1,4	3+3
1,2,4,3	3+1
2,1,4,3	3+1
1,4,2,3	3+1
4,1,2,3	3+1
2,4,1,3	3+1
4,2,1,3	3+1

$A[p\,.\,.\,r]$	execs
1,3,4,2	3+1
3,1,4,2	3+1
1,4,3,2	3+1
4,1,3,2	3+1
3,4,1,2	3+1
4,3,1,2	3+1
2,3,4,1	3+3
3,2,4,1	3+3
2,4,3,1	3+2
4,2,3,1	3+2
3,4,2,1	3+3
4,3,2,1	3+3
média	116/24

Quicksort caso médio

Rearranja $A[p \mathinner{..} r]$ em ordem crescente.

QUICKSORT (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE (A, p, r)
3 QUICKSORT $(A, p, q-1)$
4 QUICKSORT $(A, q+1, r)$

Análise do consumo médio?

Basta contar o número esperado de comparações na linha 4 do PARTICIONE

Suponha que $A[1 \mathinner{..} n]$ é permutação aleatória uniforme de $1, \ldots, n$

Cada permutação tem probabilidade $1/n!$.

Consumo de tempo médio

O número esperado $C(n)$ de comparações na linha 4 do PARTICIONE satisfaz a recorrência:

$$C(0) = 0$$

$$C(n) = \frac{1}{n}\Big(\sum_{k=0}^{n-1} C(k) + C(n-k-1)\Big) + n - 1 \quad \text{para } n = 1, 2, 3, \ldots$$

Rescrevendo …

$$C(0) = 0$$

$$C(n) = \frac{2}{n}\sum_{k=0}^{n-1} C(k) + n - 1 \quad \text{para } n = 1, 2, 3, \ldots$$

Consumo de tempo médio (cont.)

O número esperado $C(n)$ de comparações na linha 4 do PARTICIONE satisfaz a recorrência:

$$C(0) = 0$$

$$C(n) = \frac{2}{n} \sum_{k=0}^{n-1} C(k) + n - 1 \quad \text{para } n = 1, 2, 3, \ldots$$

Para nos livrarmos da divisão, multiplicamos ambos os lados por n

$$nC(n) = 2 \sum_{k=0}^{n-1} C(k) + n^2 - n$$

Consumo de tempo médio (cont.)

Agora, para nos livramos do somatório, subtraímos a equação anterior de

$$(n-1)C(n-1) = 2\sum_{k=0}^{n-2} C(k) + (n-1)^2 - (n-1)$$

e obtemos

$$nC(n) - (n-1)C(n-1) = 2\,C(n-1) + 2n - 2.$$

Agora temos que resolver uma recorrência 'mais simples':

$$C(0) = 0$$

$$nC(n) = (n+1)C(n-1) + 2n - 2 \quad \text{para } n = 1, 2, 3, \ldots$$

Consumo de tempo médio (cont.)

Multiplicando, ambos os lados por $\frac{1}{n(n+1)}$ obtemos

$$\frac{1}{n+1}C(n) = \frac{1}{n}C(n-1) + \frac{2}{n+1} - \frac{2}{n(n+1)}\ .$$

Fazendo $S(n) := \frac{1}{n+1}C(n)$ chegamos a recorrência

$$S(0) = 0$$

$$S(n) = S(n-1) + \frac{2}{n+1} - \frac{2}{n(n+1)} \quad \text{para } n = 1, 2, 3, \ldots$$

que tem como solução

$$S(n) = 2\Big(H_{n+1} - 2 - \frac{1}{n+1}\Big)\ .$$

Consumo de tempo médio (cont.)

Concluímos que

$$C(n) = 2(n+1)H_{n+1} - 4(n+1) + 2 \text{ para } n = 0, 1, 2, \ldots$$

$$\ln(n+1) < H_{n+1} < 1 + \ln(n+1)$$

Verifiquemos o valor de $C(n)$ para valores pequenos

n	0	1	2	3	4
$C(n)$	0	1	1	$16/6$	$116/24$

Hmmmmm, parece que está certo.

Conclusões

$C(n)$ é $\Theta(n \log n)$.

O consumo de tempo esperado do algoritmo QUICKSORT é $\Theta(n \log n)$.

Quicksort aleatorizado

PARTICIONE-ALEA(A, p, r)
1 $i \leftarrow$ RANDOM (p, r)
2 $A[i] \leftrightarrow A[r]$
3 **devolva** PARTICIONE (A, p, r)

QUICKSORT-ALE (A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow$ PARTICIONE-ALEA (A, p, r)
3 QUICKSORT-ALE $(A, p, q - 1)$
4 QUICKSORT-ALE $(A, q + 1, r)$

Análise do consumo médio?

Basta contar o número esperado de comparações na linha 4 do PARTICIONE

Exemplo

1	3	6	2	5	7	4

1	3	2	4	5	7	6

1	2	3	4	5	6	7

	1	2	3	4	5	6	7
1		1	0	1	0	0	0
2	1		1	1	0	0	0
3	0	1		1	0	0	0
4	1	1	1		1	1	1
5	0	0	0	1		1	0
6	0	0	0	1	1		1
7	0	0	0	1	0	1	

Consumo de tempo esperado

Suponha $A[p \mathrel{..} r]$ permutação de $1 \mathrel{..} n$.

X_{ab} = número de comparações entre a e b
na linha 4 de PARTICIONE

Queremos calcular

$$
\begin{aligned}
X &= \text{total de comparações "}A[j] \leq x\text{"} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{ab}
\end{aligned}
$$

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô em } \{a, \ldots, b\} \text{ é } a \text{ ou } b \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô em } \{a, \ldots, b\} \text{ é } a \text{ ou } b \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

$$\text{Pr}\{X_{ab}{=}1\} \;=\; \frac{1}{b-a+1} + \frac{1}{b-a+1} \;=\; \mathrm{E}[X_{ab}]$$

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô em } \{a, \ldots, b\} \text{ é } a \text{ ou } b \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

$$\text{Pr}\,\{X_{ab}{=}1\} \;=\; \frac{1}{b-a+1} + \frac{1}{b-a+1} \;=\; \mathrm{E}[X_{ab}]$$

$$X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{ab}$$

$\mathrm{E}[X] =$????

Consumo de tempo esperado

$$
\begin{aligned}
\mathrm{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \mathrm{E}[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \Pr\{X_{ab}=1\} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b-a+1} \\
&= \sum_{a=1}^{n-1} \sum_{k=1}^{n-a} \frac{2}{k+1} \\
&< \sum_{a=1}^{n-1} 2\left(\frac{1}{1}+\frac{1}{2}+\cdots+\frac{1}{n}\right) \\
&< 2n\left(\frac{1}{1}+\frac{1}{2}+\cdots+\frac{1}{n}\right) < 2n\,(1+\ln n)
\end{aligned}
$$

CLRS (A.7), p.1060

Conclusões

O consumo de tempo esperado do algoritmo QUICKSORT-ALE é $O(n \lg n)$.

Do exercício 7.4-4 do CLRS temos que

O consumo de tempo esperado do algoritmo QUICKSORT-ALE é $\Theta(n \log n)$.

Limites inferiores

CLRS 8.1

Máximo: limite inferior

Problema: Encontrar o maior elemento de um vetor $A[1 \,.\,.\, n]$.

Existe um algoritmo que faz o serviço com $n-1$ comparações.

MAX (A, n)
1 $\quad max \leftarrow A[1]$
2 $\quad$ **para** $i \leftarrow 2$ **até** n **faça**
3 $\qquad$ **se** $A[i] > max$

---

4 $\qquad\quad$ **então** $max \leftarrow A[i]$
5 $\quad$ **devolva** max

Máximo: limite inferior

Existe um algoritmo que faz menos comparações?

Máximo: limite inferior

Existe um algoritmo que faz menos comparações?

Não, se o algoritmo é baseado em comparações:

> dados dois número $A[i]$ e $A[j]$ podemos apenas comparar-los a fim de encontrar o maior elemento.

Suponha dado um algoritmo baseado em comparações que resolve o problema.

Algoritmo baseado em comparações

O algoritmo consiste, no fundo, na determinação de uma coleção $\mathcal{A}$ de pares ou arcos $\langle i, j \rangle$ de elementos distintos em $\{1, \ldots, n\}$ tais que $A[i] < A[j]$ e existe um "sorvedouro".

Eis o paradigma de todo algoritmo baseado em comparações:

MAX (A, n)
1 $\mathcal{A} \leftarrow \emptyset$
2 **enquanto** $\mathcal{A}$ "não possui sorvedouro" **faça**
3 **escolha** índice i e j em $\{1, \ldots, n\}$
4 **se** $A[i] < A[j]$
5 **então** $\mathcal{A} \leftarrow \mathcal{A} \cup \langle i, j \rangle$
6 **senão** $\mathcal{A} \leftarrow \mathcal{A} \cup \langle j, i \rangle$
7 devolva $\mathcal{A}$

Conclusão

Qualquer conjunto $\mathcal{A}$ devolvido pelo método contém uma "árvore enraizada" e portanto contém pelo menos $n-1$ arcos.

Qualquer algoritmo baseado em comparações que encontra o maior elemento de um vetor $A[1\,.\,.\,n]$ faz pelo menos $n-1$ comparações.

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

NÃO, se o algoritmo é baseado em comparações.

Prova?

Ordenação: limite inferior

Problema: Rearranjar um vetor $A[1..n]$ de modo que ele fique em ordem crescente.

Existem algoritmos que consomem tempo $O(n \lg n)$.

Existe algoritmo assintoticamente melhor?

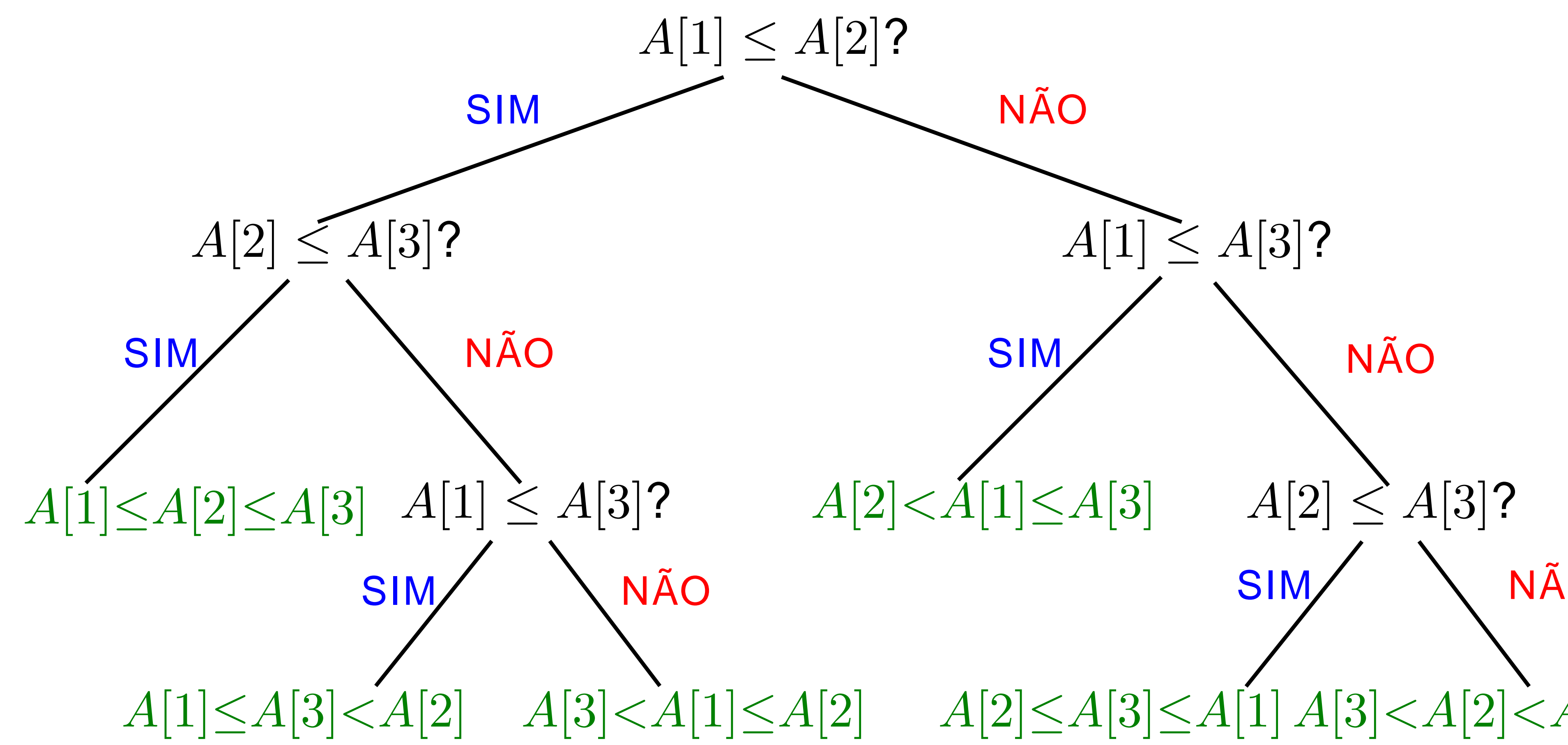
NÃO, se o algoritmo é baseado em comparações.

Prova?

Qualquer algoritmo baseado em comparações é uma "árvore de decisão".

Árvore de decisão

ORDENA-POR-INSERÇÃO $(A[1\mathinner{.\,.}3])$:



Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Qual é o número de comparações, no pior caso?

Limite inferior

Considere uma árvore de decisão para $A[1 \,..\, n]$.

Qual é o número de comparações, no pior caso?

Resposta: altura, h, da árvore de decisão.

Limite inferior

Considere uma árvore de decisão para $A[1 \,.\,.\, n]$.

Qual é o número de comparações, no pior caso?

Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Qual é o número de comparações, no pior caso?

Resposta: altura, h, da árvore de decisão.

Todas as $n!$ permutações de $1, \ldots, n$ devem ser folhas.

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: ...

Limite inferior

Toda árvore binária de altura h tem no máximo 2^h folhas.

Prova: Por indução em h.

A afirmação vale para $h = 0$

Suponha que a afirmação vale para toda árvore binária de altura menor que h, $h \geq 1$.

O número de folhas de uma árvore de altura h é a soma do número de folhas de suas sub-árvores; que têm altura $\leq h - 1$.

Logo, o número de folhas de uma árvore de altura h é não superior a

$$2 \times 2^{h-1} = 2^h.$$

Limite inferior

Logo, devemos ter $2^h \geq n!$, donde $h \geq \lg(n!)$.

$$(n!)^2 = \prod_{i=0}^{n-1} (n-i)(i+1) \geq \prod_{i=1}^{n} n = n^n$$

Portanto,

$$h \geq \lg(n!) \geq \lg n^{\frac{n}{2}} = \frac{1}{2} n \lg n.$$

Conclusão

Todo algoritmo de ordenação baseado em comparações faz

$$\Omega(n \lg n)$$

comparações no pior caso.

Exercícios

Exercício 16.A
Desenhe a árvore de decisão para o SELECTION-SORT aplicado a $A[1\,..\,3]$ com todos os elementos distintos.

Exercício 16.B [CLRS 8.1-1]
Qual o menor profundidade (= menor nível) que uma folha pode ter em uma árvore de decisão que descreve um algoritmo de ordenação baseado em comparações?

Exercício 16.C [CLRS 8.1-2]
Mostre que $\lg(n!) = \Omega(n \lg n)$ sem usar a fórmula de Stirling. Sugestão: Calcule $\sum_{k=1}^{n} \lg k$. Use as técnicas de CLRS A.2.

AULA 13

Ordenação em tempo linear

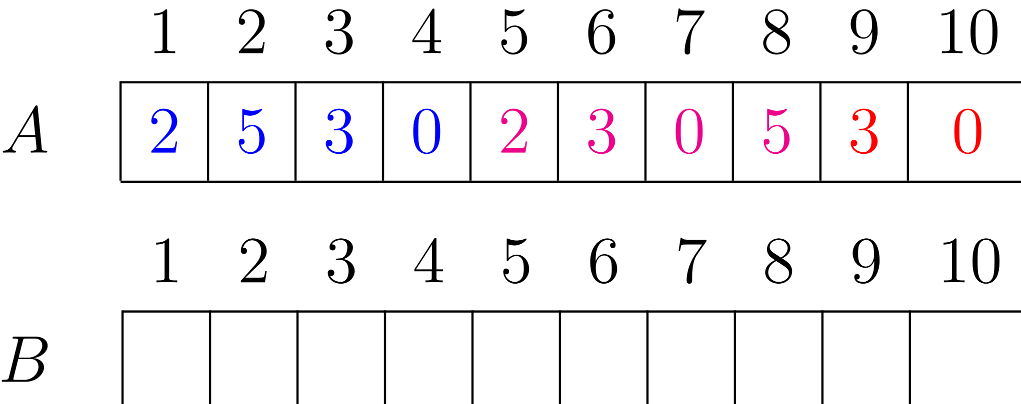
CLRS 8.2–8.3

Ordenação por contagem

Recebe vetores $A[1\,..\,n]$ e $B[1\,..\,n]$ e devolve no vetor $B[1\,..\,n]$ os elementos de $A[1\,..\,n]$ em ordem crescente.

Cada $A[i]$ está em $\{0, \ldots, k\}$.

Entra:

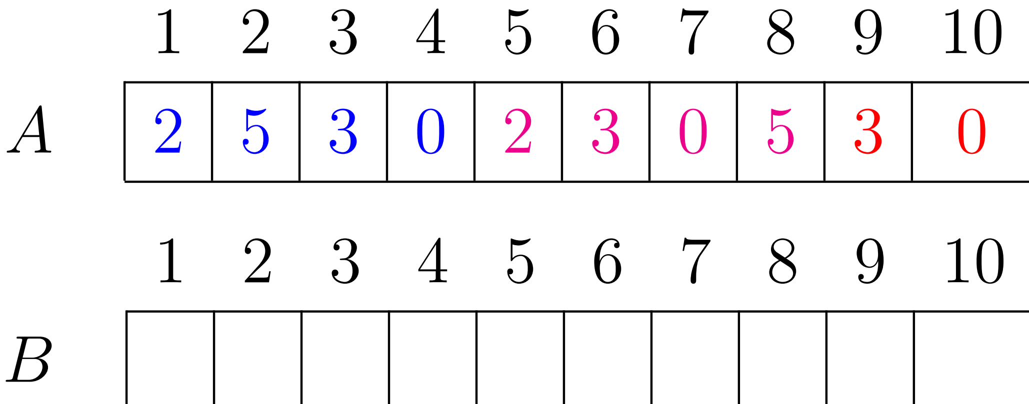


Ordenação por contagem

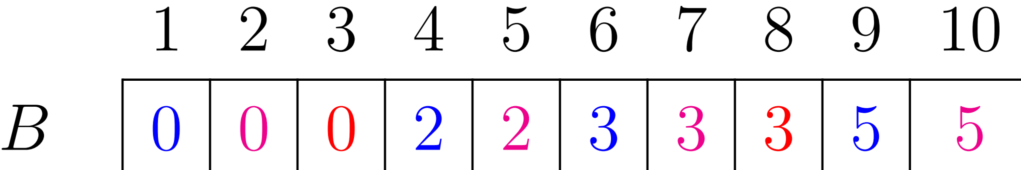
Recebe vetores $A[1\,.\,.\,n]$ e $B[1\,.\,.\,n]$ e devolve no vetor $B[1\,.\,.\,n]$ os elementos de $A[1\,.\,.\,n]$ em ordem crescente.

Cada $A[i]$ está em $\{0, \ldots, k\}$.

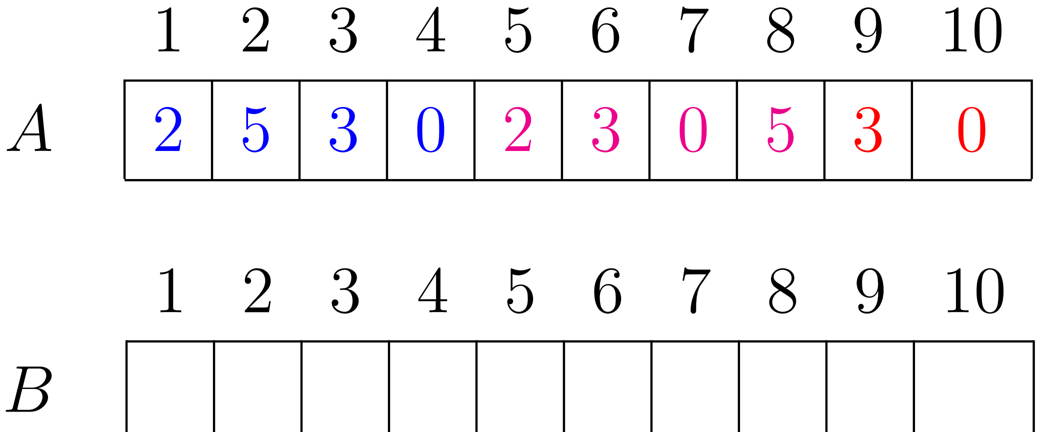
Entra:



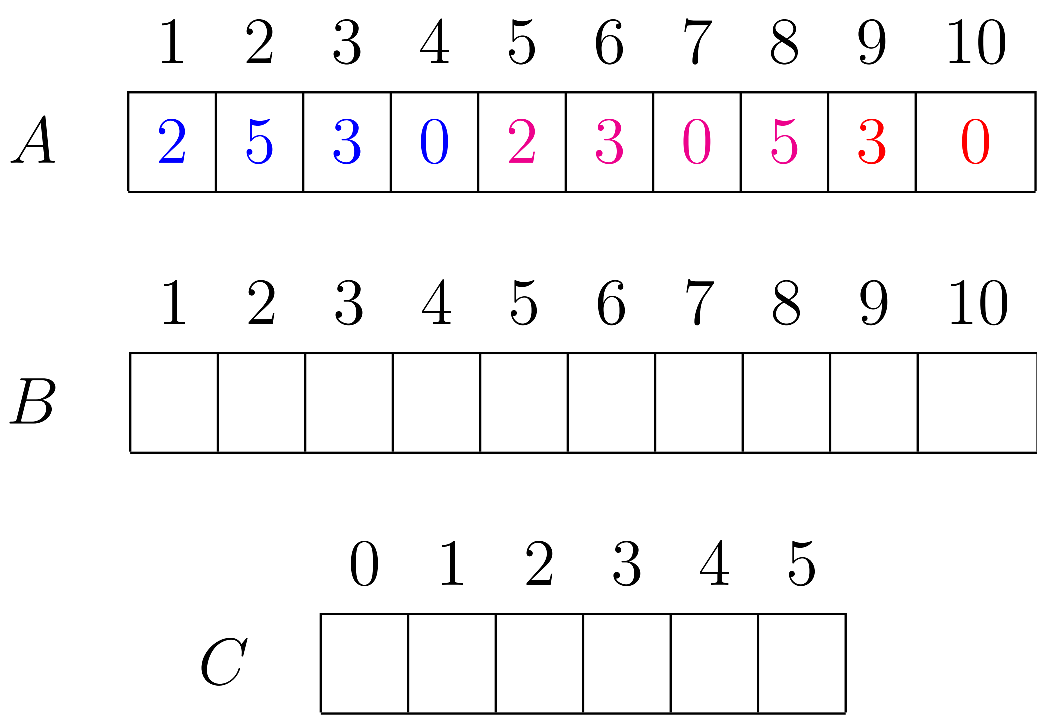
Sai:



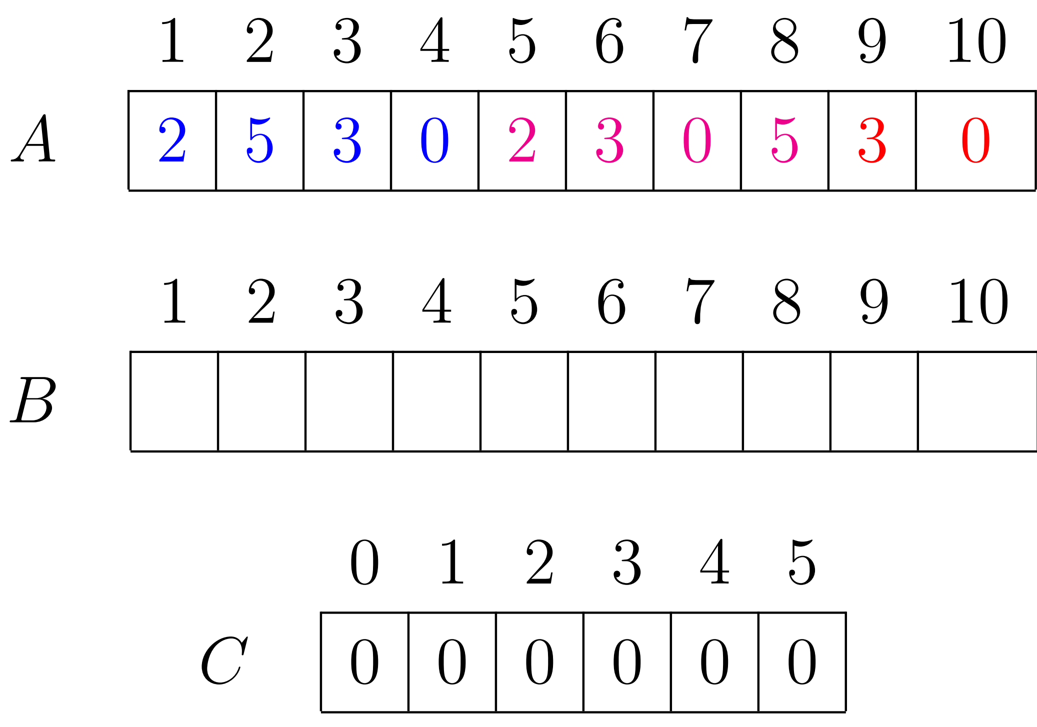
Ordenação por contagem



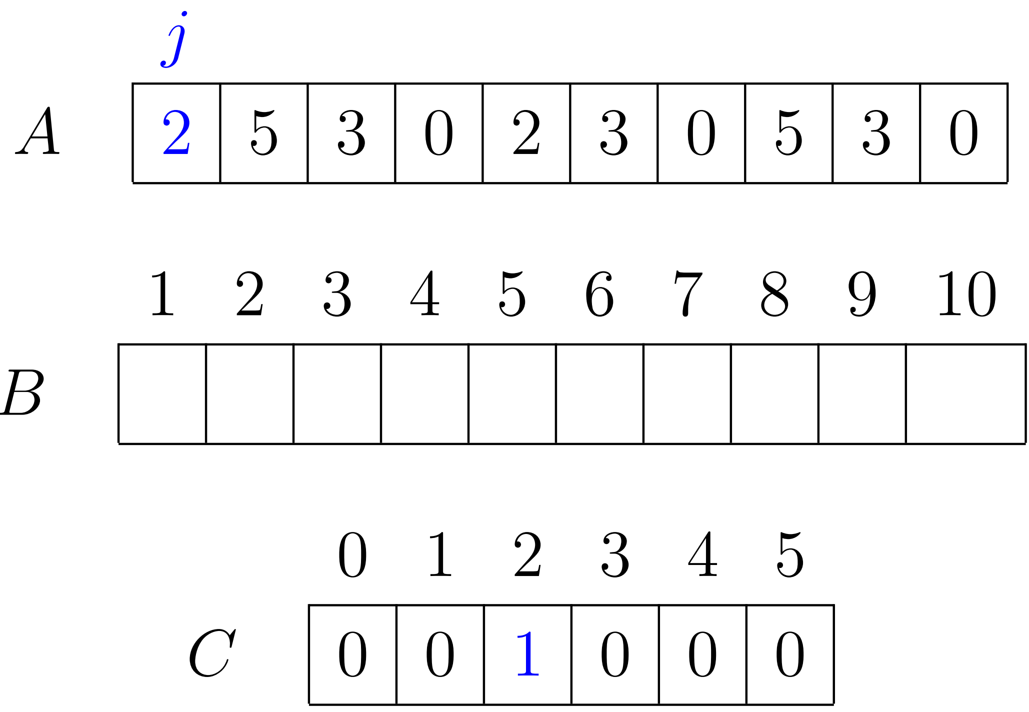
Ordenação por contagem



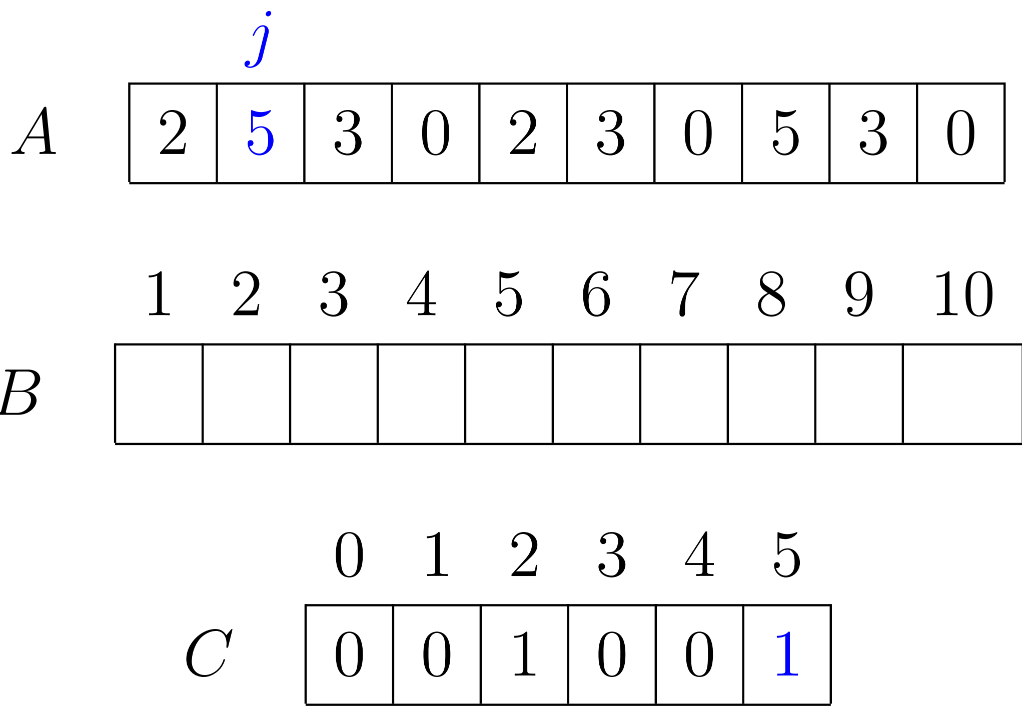
Ordenação por contagem



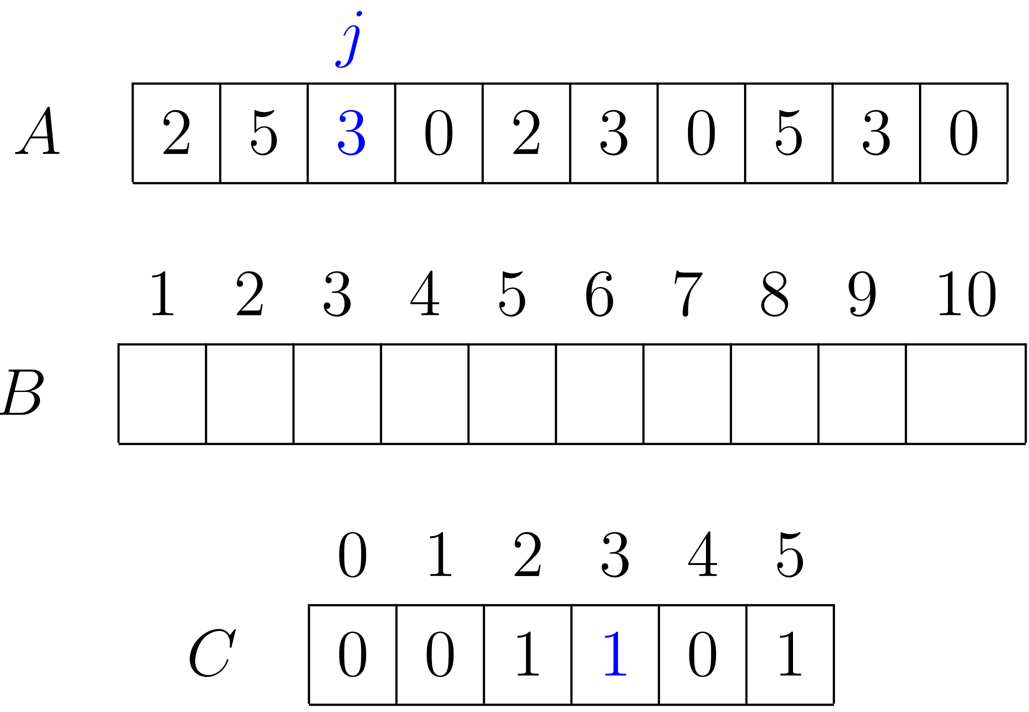
Ordenação por contagem



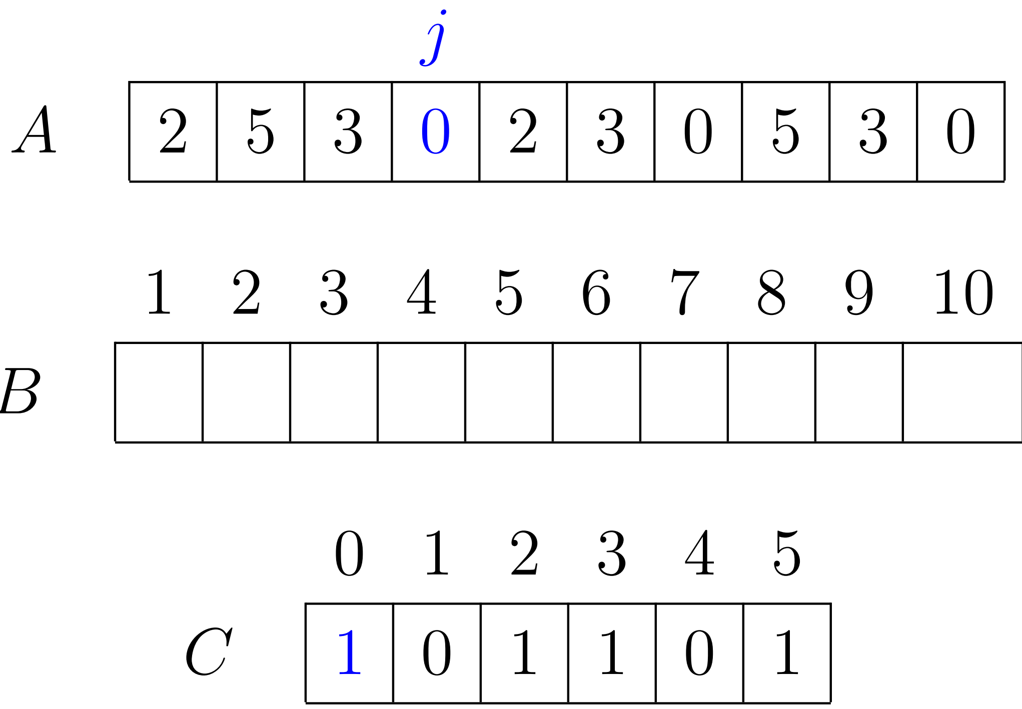
Ordenação por contagem



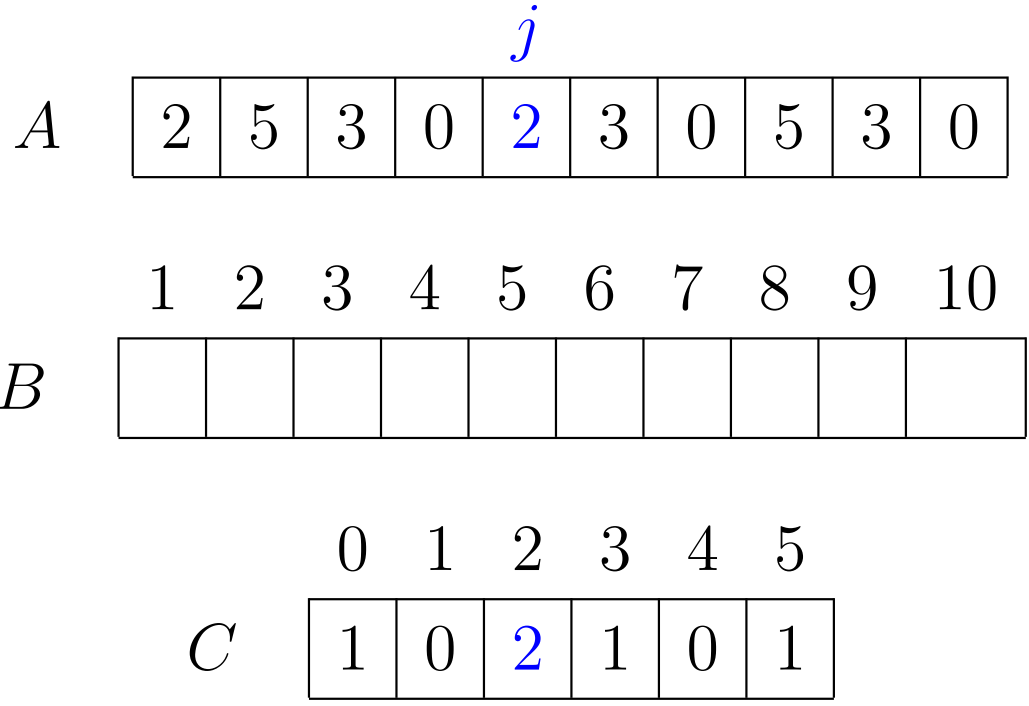
Ordenação por contagem



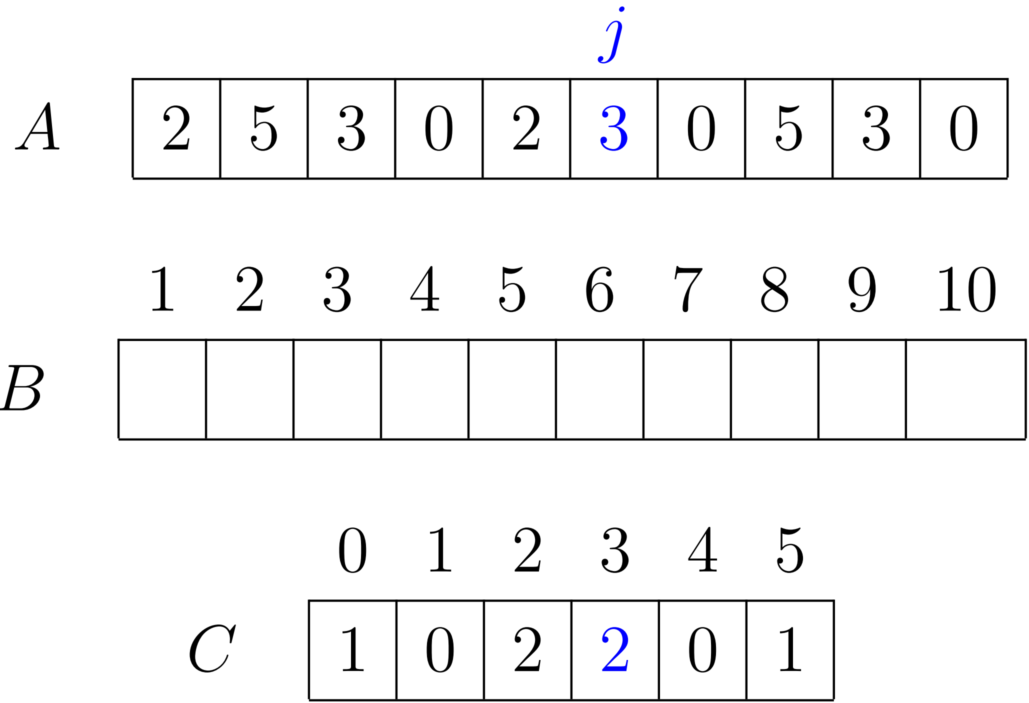
Ordenação por contagem



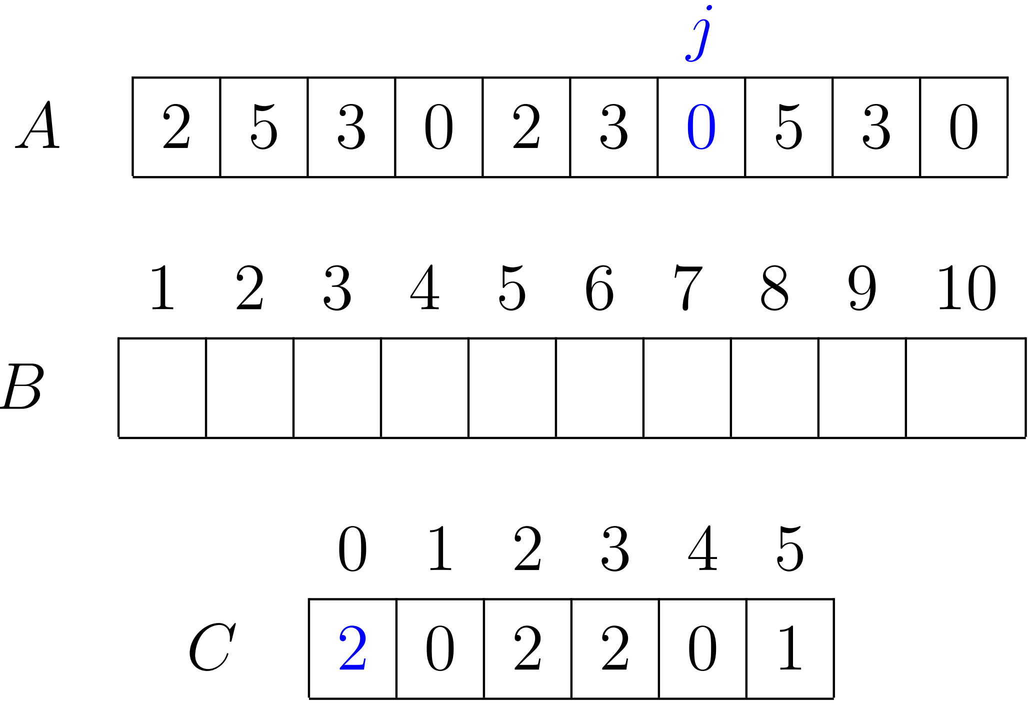
Ordenação por contagem



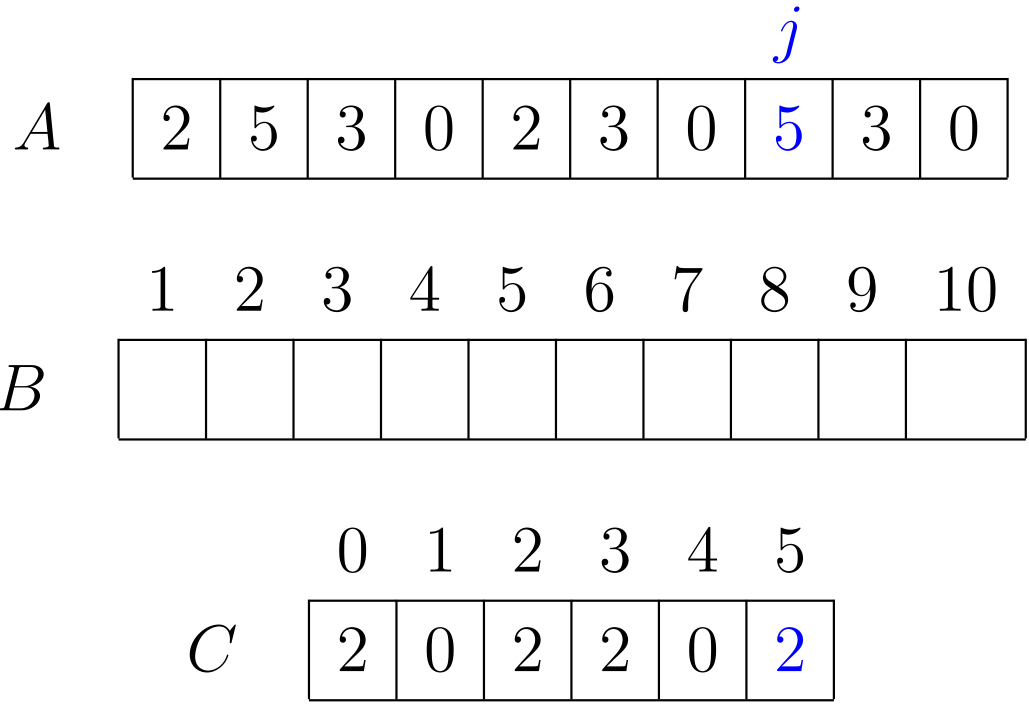
Ordenação por contagem



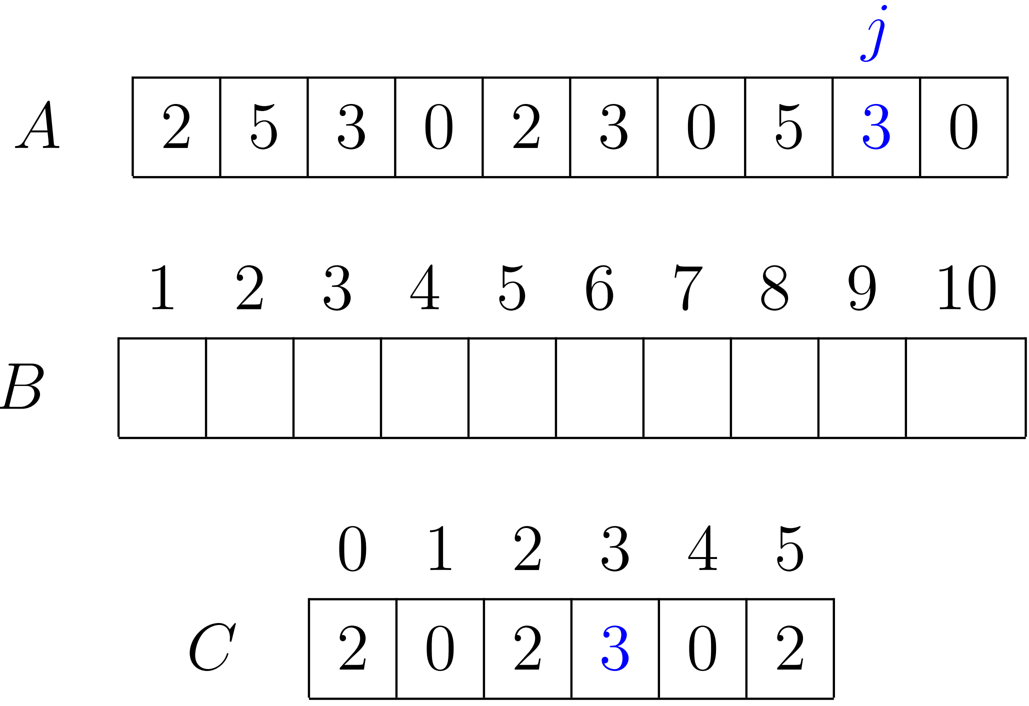
Ordenação por contagem



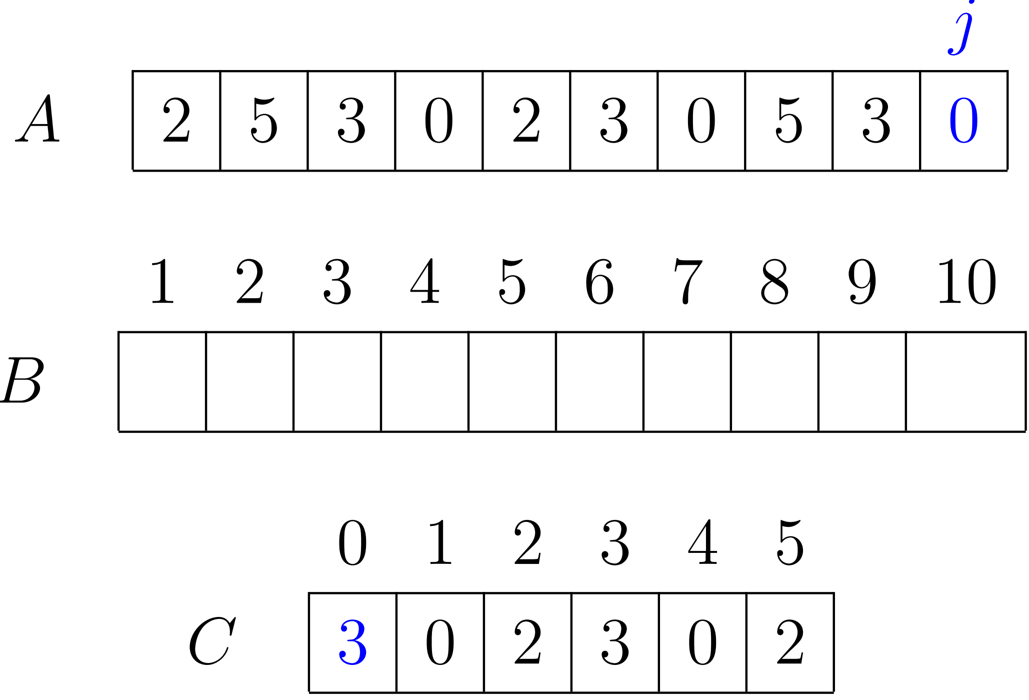
Ordenação por contagem



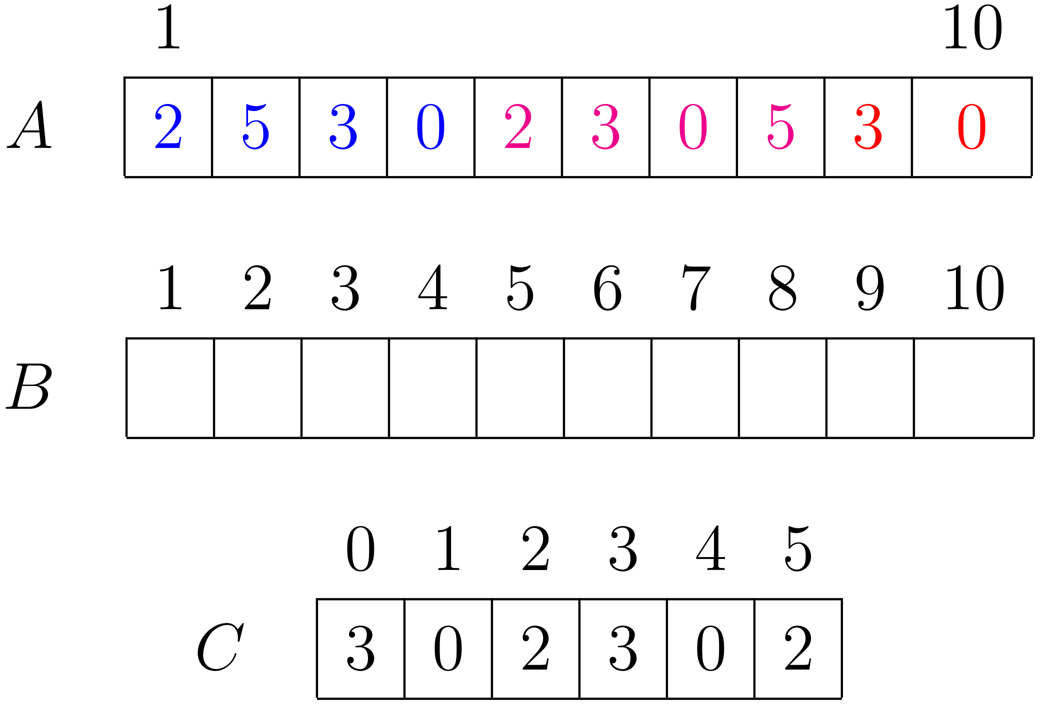
Ordenação por contagem



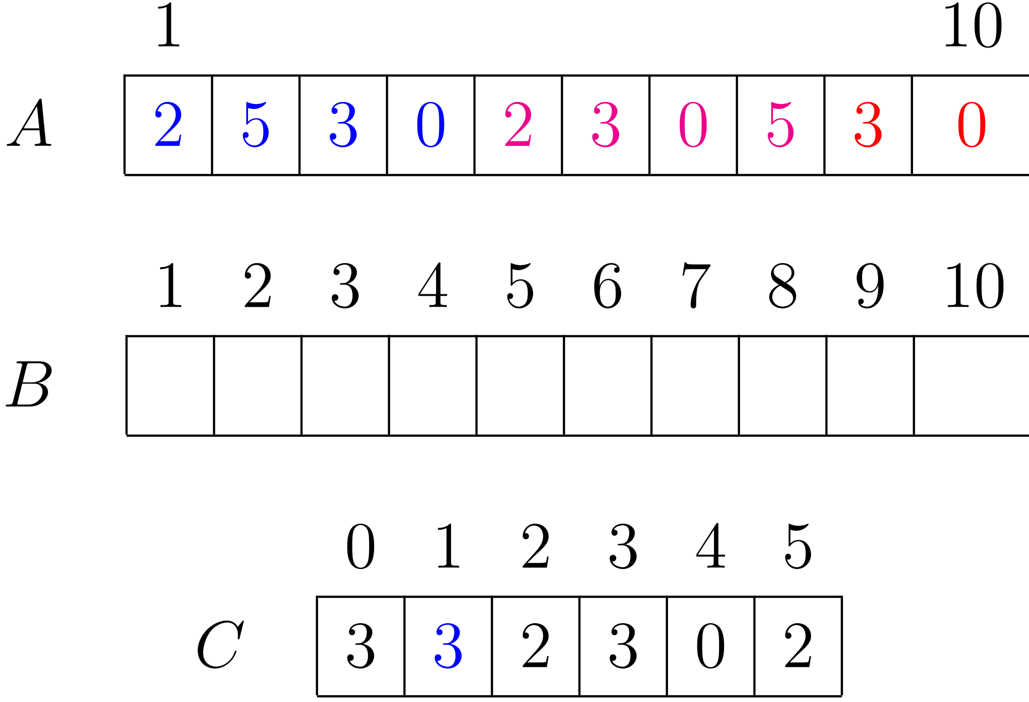
Ordenação por contagem



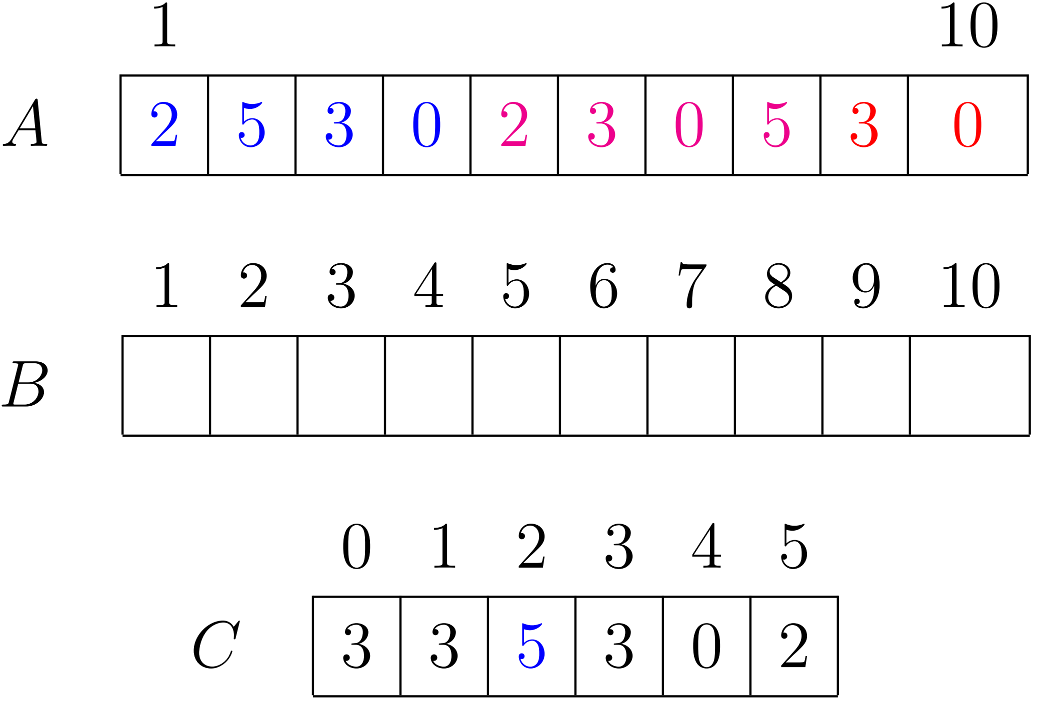
Ordenação por contagem



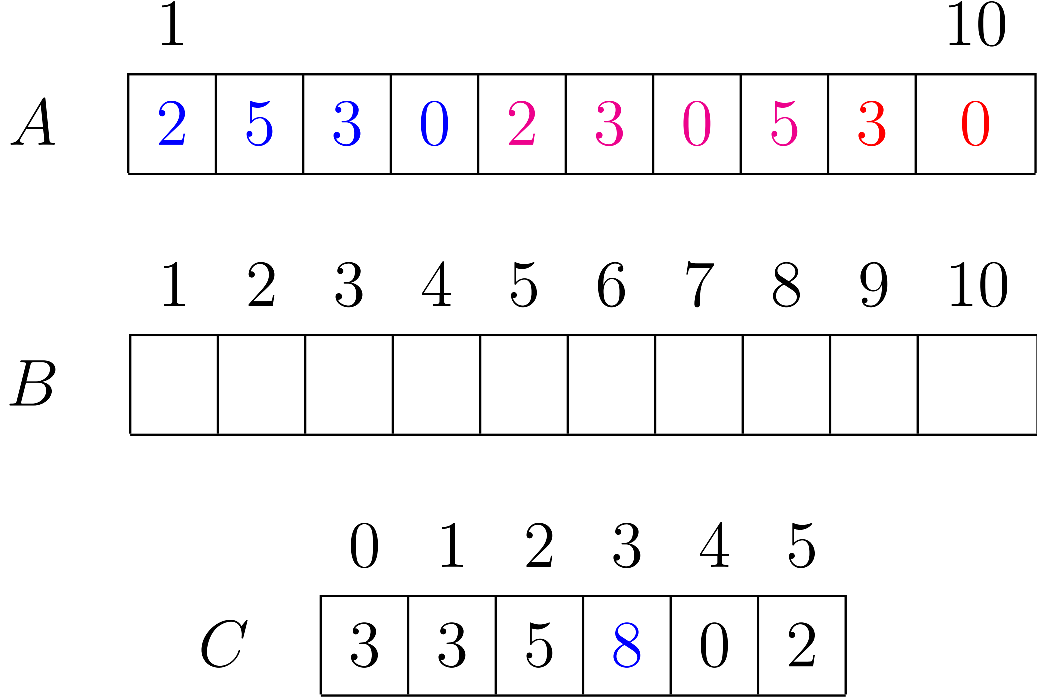
Ordenação por contagem



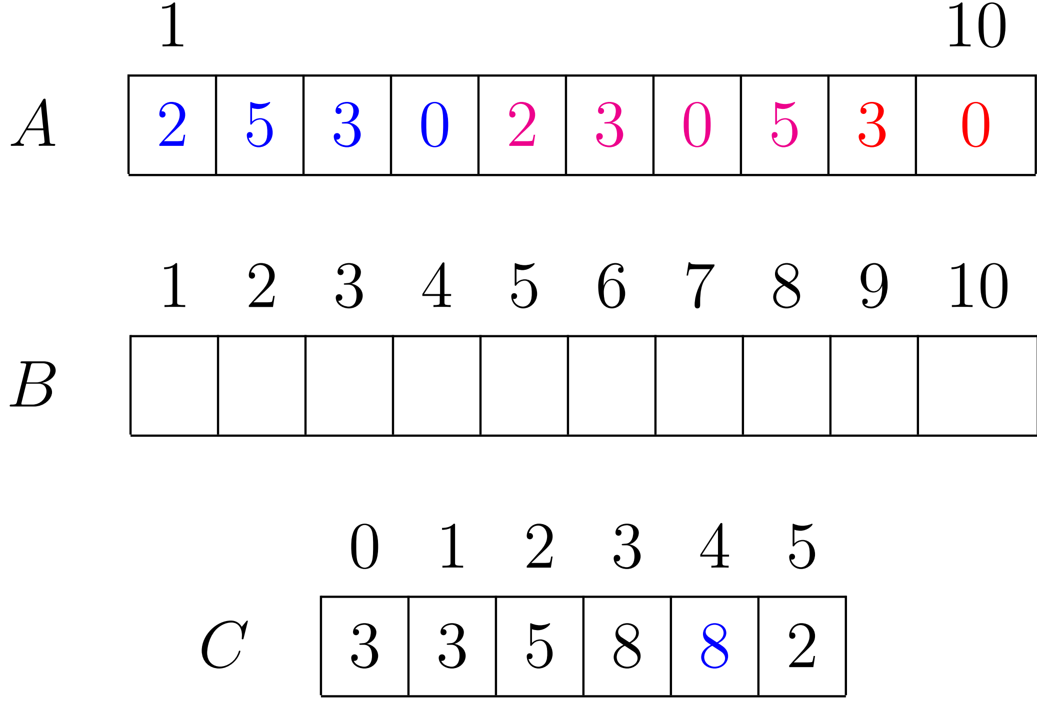
Ordenação por contagem



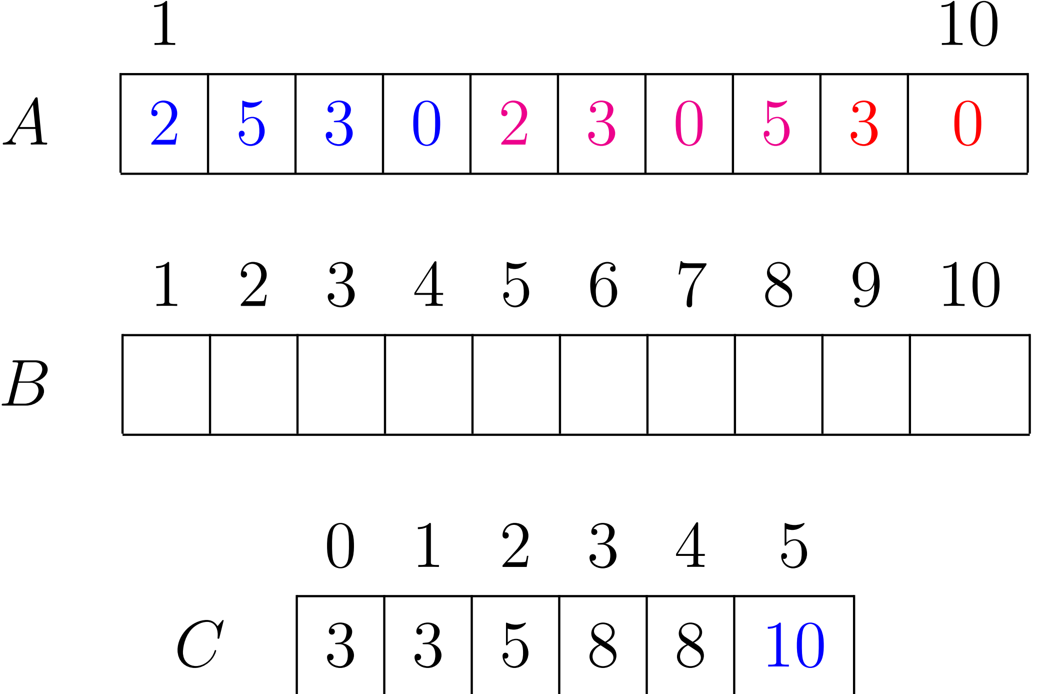
Ordenação por contagem



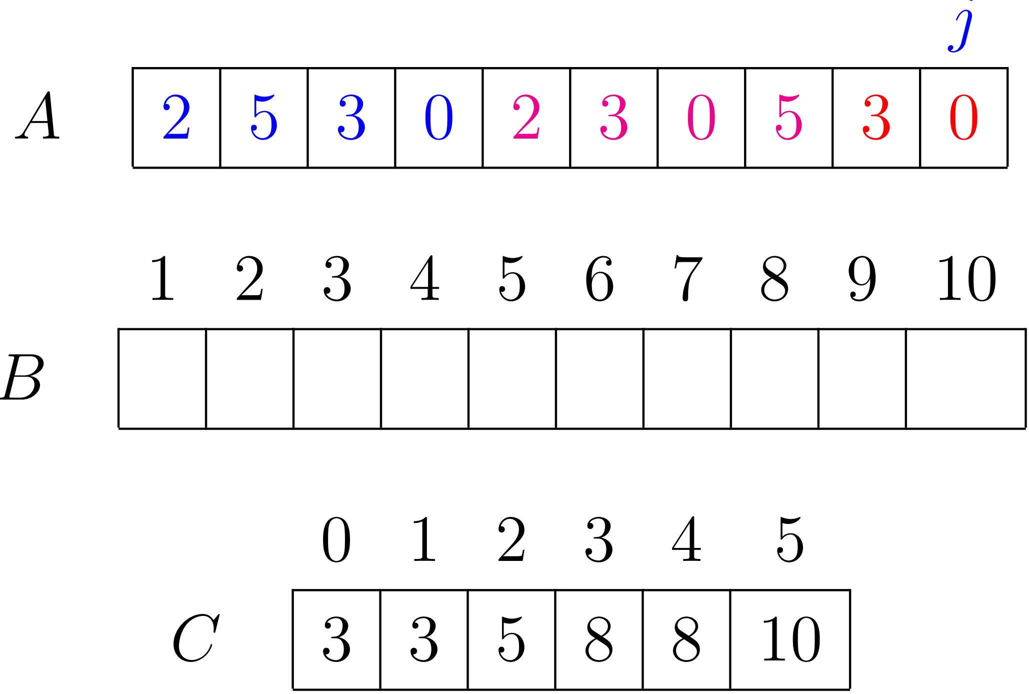
Ordenação por contagem



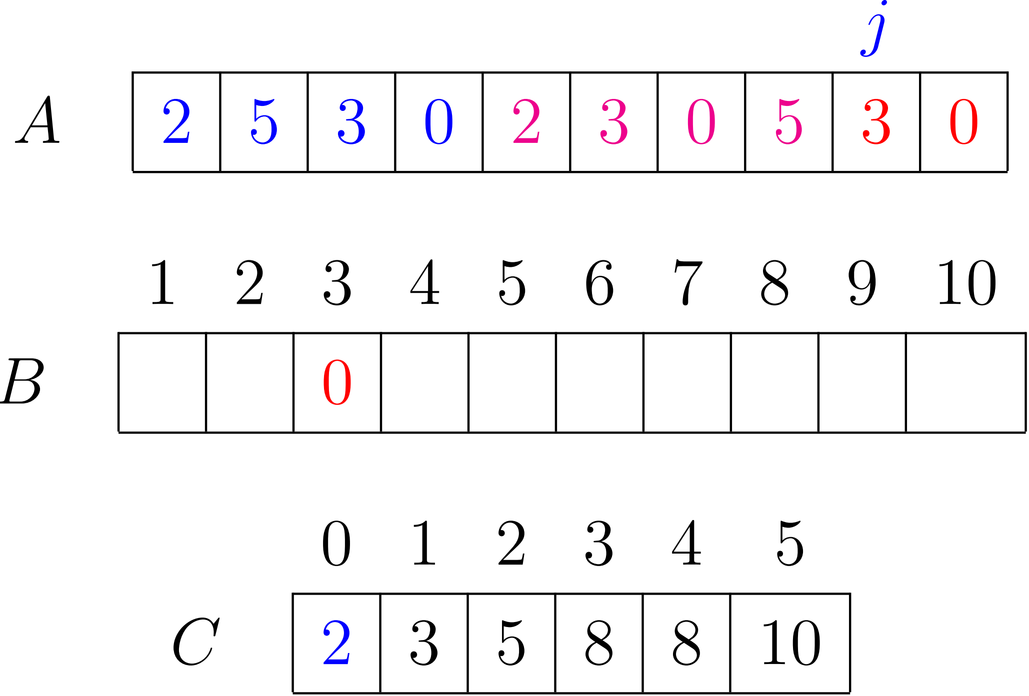
Ordenação por contagem



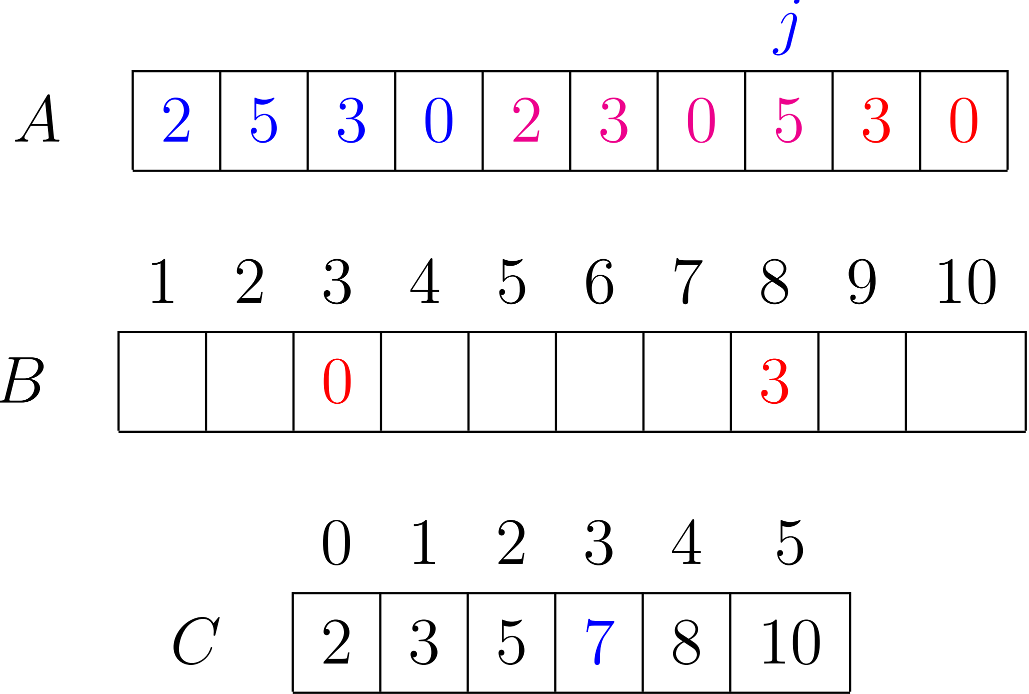
Ordenação por contagem



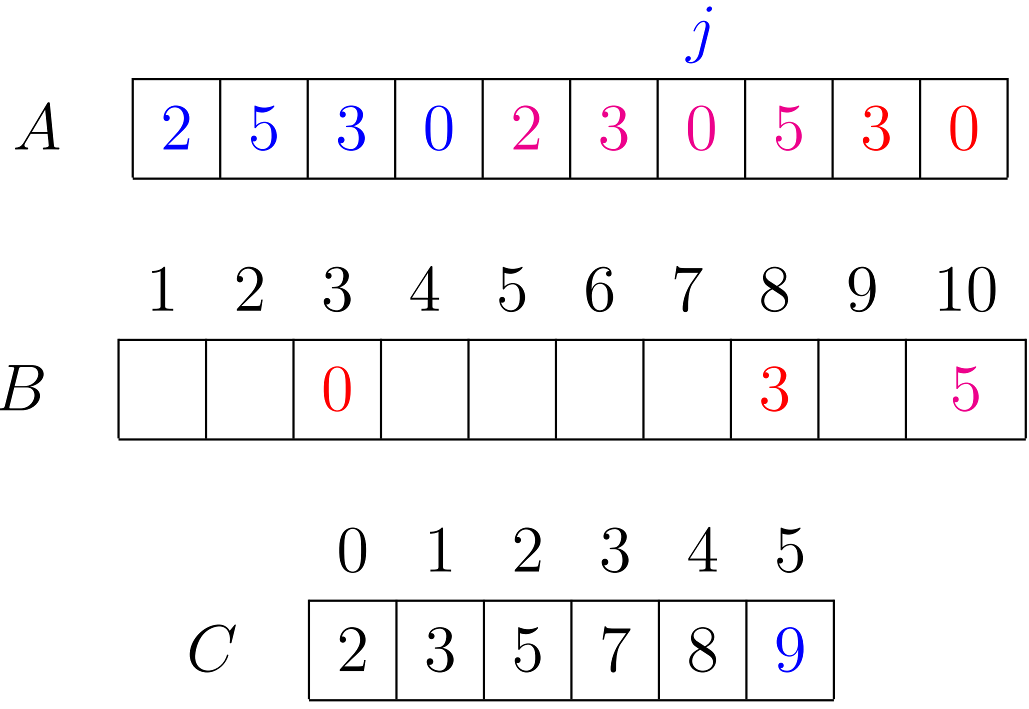
Ordenação por contagem



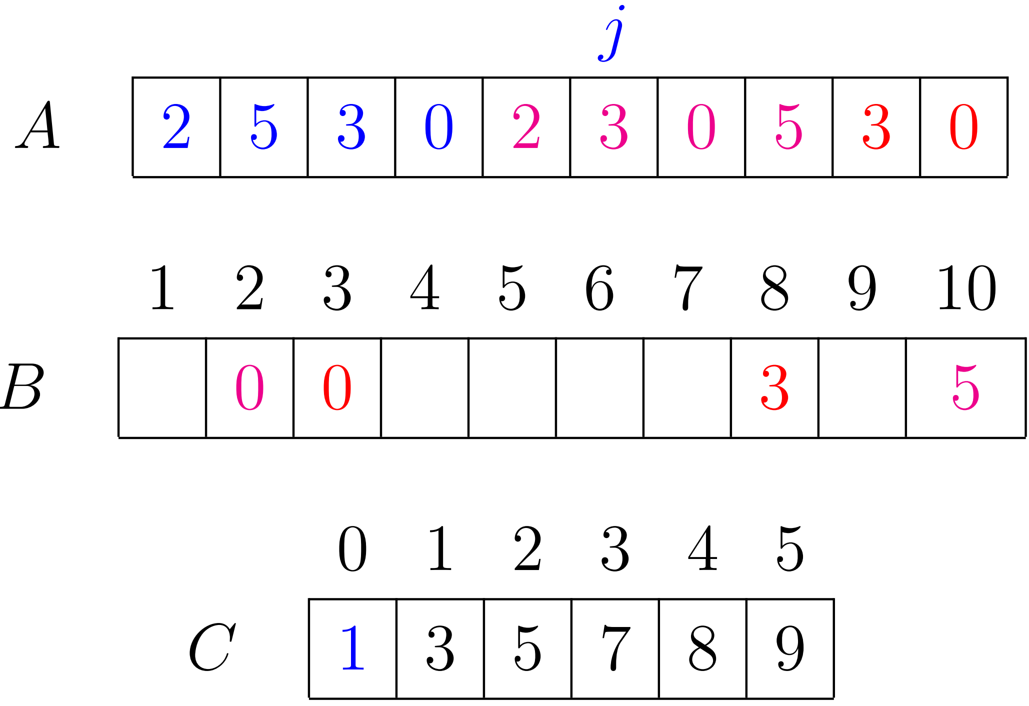
Ordenação por contagem



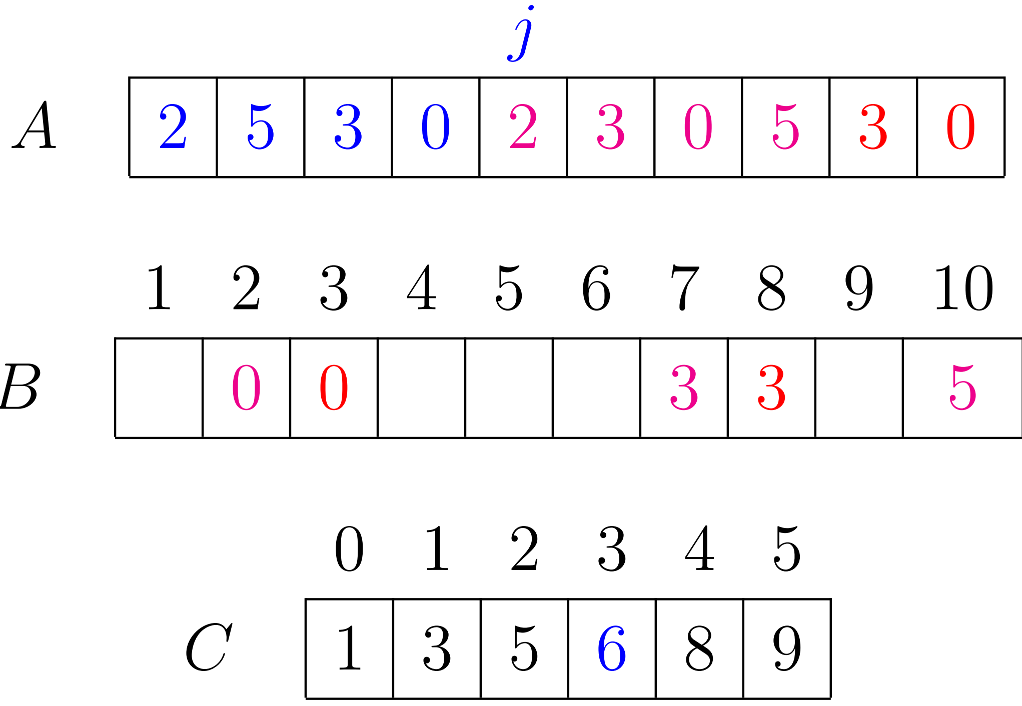
Ordenação por contagem



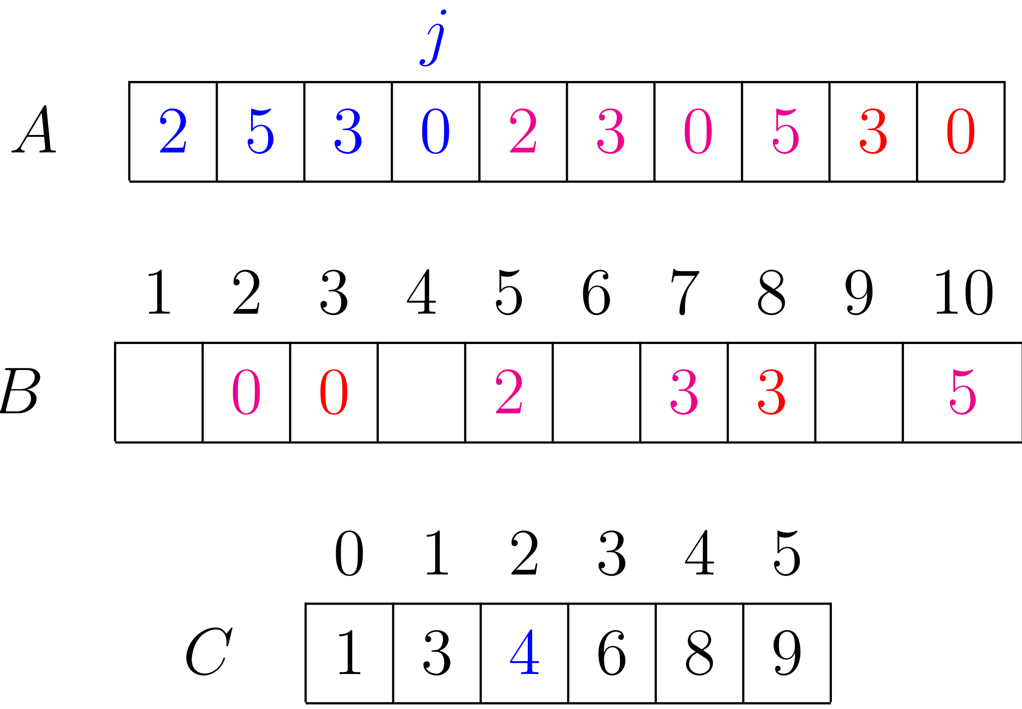
Ordenação por contagem



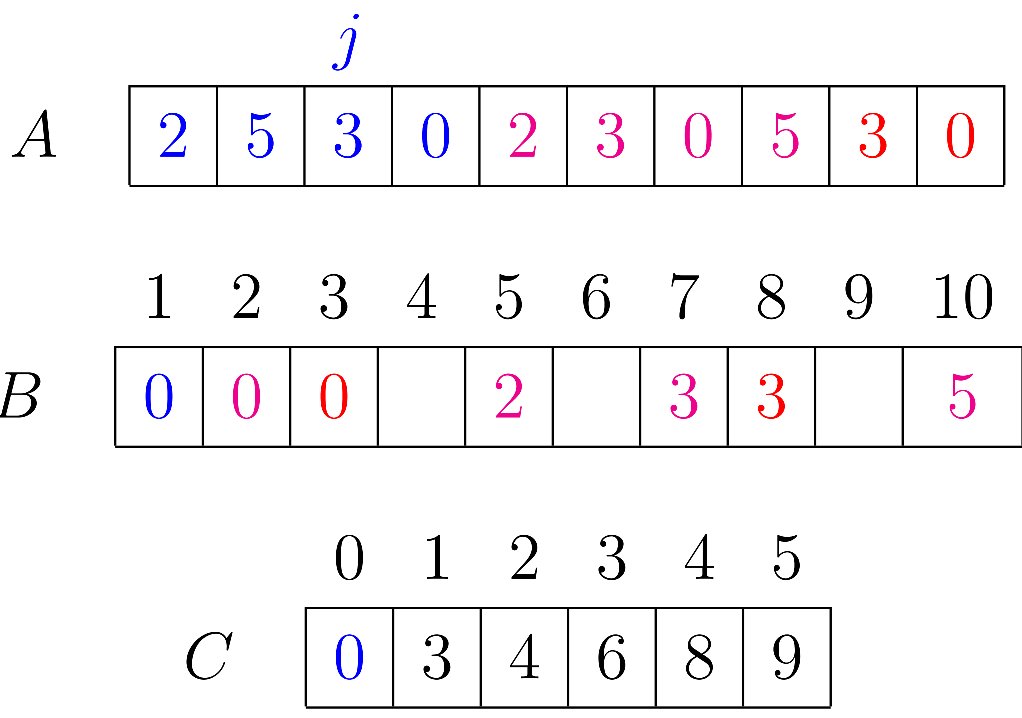
Ordenação por contagem



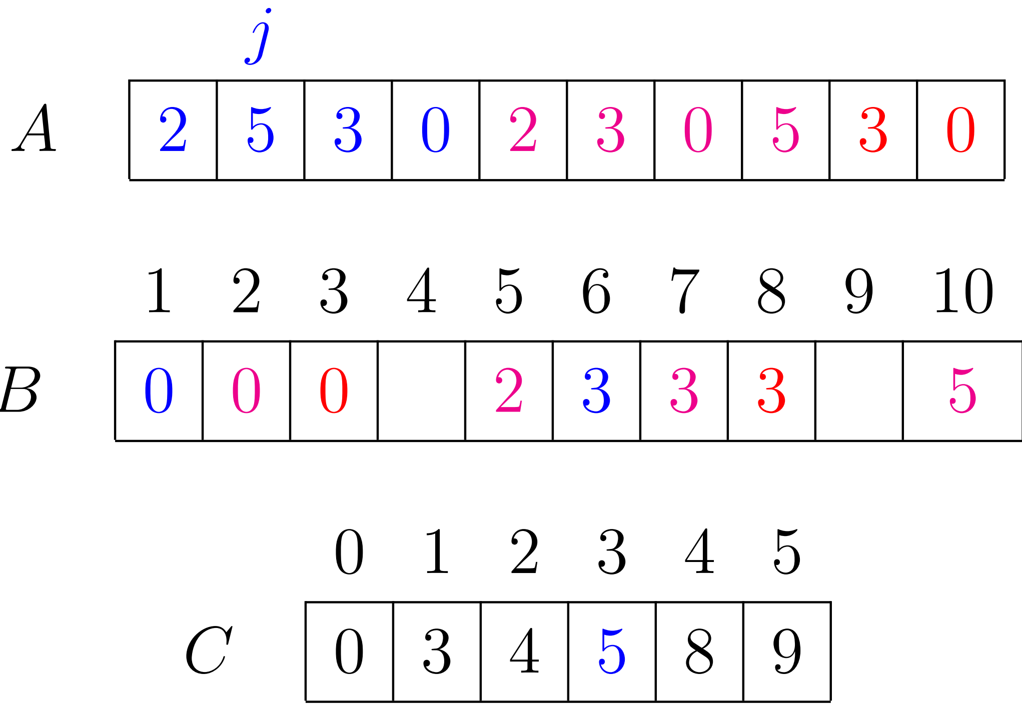
Ordenação por contagem



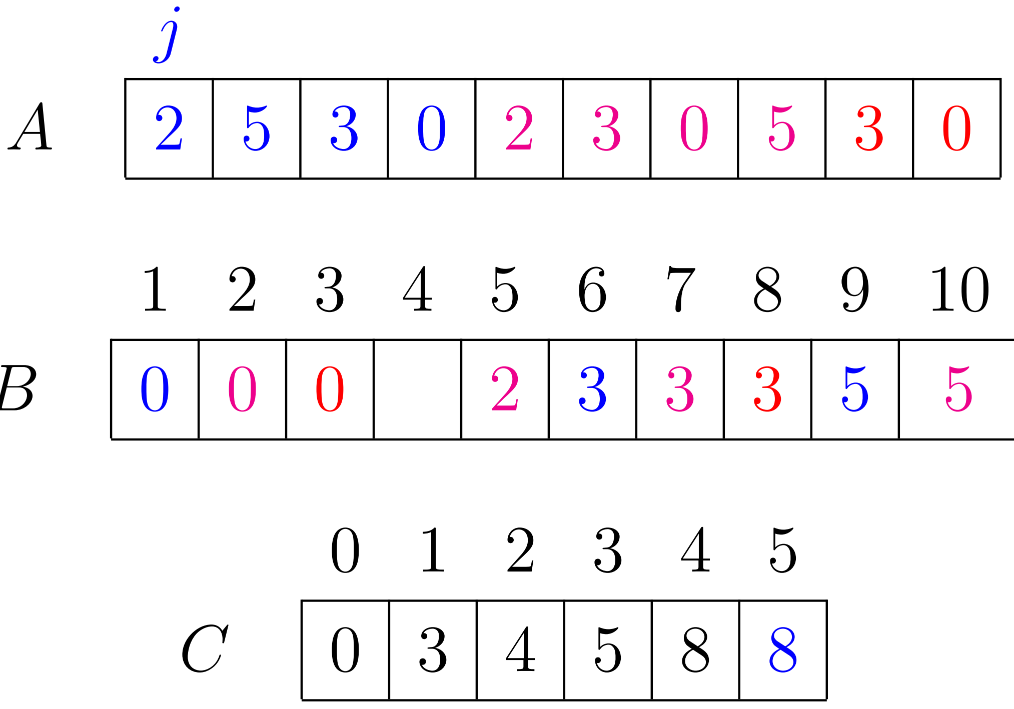
Ordenação por contagem



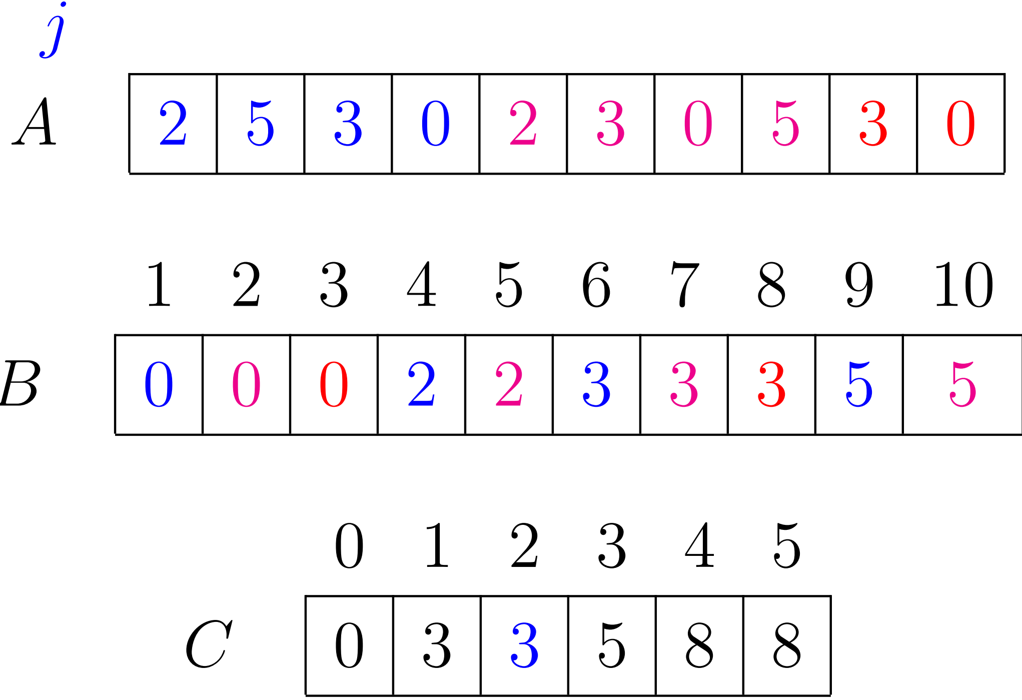
Ordenação por contagem



Ordenação por contagem



Ordenação por contagem



Ordenação por contagem

COUNTING-SORT (A, B, n, k)

1 **para** $i \leftarrow 0$ **até** k **faça**
2 $\quad C[i] \leftarrow 0$

3 **para** $j \leftarrow 1$ **até** n **faça**
4 $\quad C[A[j]] \leftarrow C[A[j]] + 1$
$\triangleright$ $C[i]$ é o número de js tais que $A[j] = i$

5 **para** $i \leftarrow 1$ **até** k **faça**
6 $\quad C[i] \leftarrow C[i] + C[i-1]$
$\triangleright$ $C[i]$ é o número de js tais que $A[j] \leq i$

7 **para** $j \leftarrow n$ **decrescendo até** 1 **faça**
8 $\quad B[C[A[j]]] \leftarrow A[j]$
9 $\quad C[A[j]] \leftarrow C[A[j]] - 1$

Consumo de tempo

linha	consumo na linha
$1-2$	$\Theta(k)$
$3-4$	$\Theta(n)$
$5-6$	$\Theta(k)$
$7-9$	$\Theta(n)$

Consumo total: $\Theta(n + k)$

Conclusões

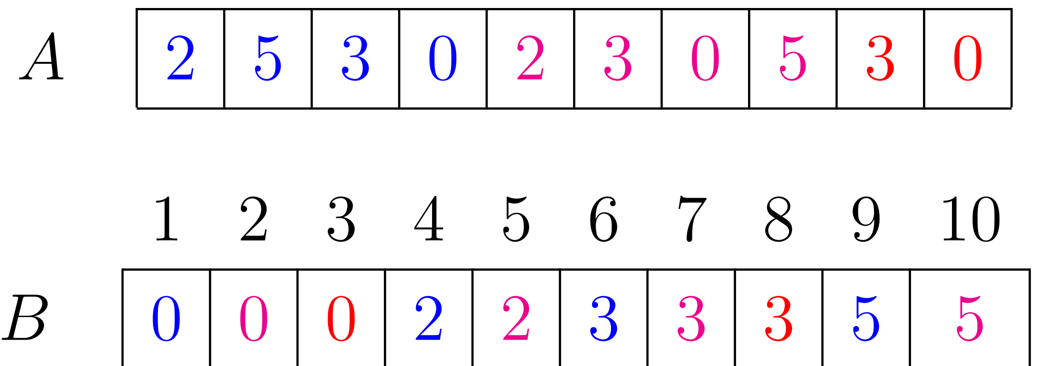
O consumo de tempo do COUNTING-SORT é
$\Theta(n+k)$.

- se $k \leq n$ então consumo é $\Theta(n)$
- se $k \leq 10n$ então consumo é $\Theta(n)$
- se $k = \mathrm{O}(n)$ então consumo é $\Theta(n)$
- se $k \geq n^2$ então consumo é $\Theta(k)$
- se $k = \Omega(n)$ então consumo é $\Theta(k)$

Estabilidade

A propósito: COUNTING-SORT é estável:

na saída, chaves com mesmo valor estão na mesma ordem que apareciam na entrada.



Ordenação digital (=radix sort)

Exemplo:

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Cada $A[j]$ têm d dígitos decimais:

$A[j] = a_d\, 10^{d-1} + \cdots + a_2\, 10^1 + a_1\, 10^0$

Exemplo com $d = 3$: $\quad 3 \cdot 10^2 + 2 \cdot 10 + 9$

Ordenação digital

RADIX-SORT (A, n, d)
1 para $i \leftarrow 1$ até d faça
2 $\triangleright$ 1 até d e não o contrário!
3 ordene $A[1\,..\,n]$ pelo dígito i

Linha 3:

- faz ordenação $A[j_1\,..\,j_n]$ de $A[1\,..\,n]$ tal que

$$A[j_1]_i \leq \cdots \leq A[j_n]_i;$$

- ordenação deve ser estável; e
- use COUNTING-SORT.

Exemplos

- dígitos decimais: $\Theta(dn)$
- dígitos em $0 \,.\,.\, k$: $\Theta(d\,(n+k))$.

Exemplo com $d = 5$ e $k = 128$:

$$a_5 128^4 + a_4 128^3 + a_3 128^2 + a_2 128 + a_1$$

sendo $0 \leq a_i \leq 127$

Conclusão

Dados n números com b bits e um inteiro $r \leq b$, RADIX-SORT ordena esses números em tempo

$$\Theta\left(\frac{b}{r}(n+2^r)\right).$$

Mais experimentação

Os programas foram executados na

SunOS rebutosa 5.7 Generic_106541-04 sun4d sparc.

Os códigos foram compilados com o

gcc version 2.95.2 19991024 (release)

e opção de compilação

-Wall -ansi -pedantic -O2.

Algoritmos implementados:

`merger`	MERGE-SORT recursivo
`mergei`	MERGE-SORT iterativo
`heap`	HEAPSORT
`quick`	QUICKSORT recursivo.
`qCLR`	QUICKSORT do CLR.
`radix`	RADIX-SORT

Resultados

n	merger	mergei	heap	quick	qCLR	radix
4096	0.02	0.03	0.01	0.01	0.01	0.01
8192	0.05	0.06	0.03	0.02	0.02	0.03
16384	0.12	0.13	0.07	0.06	0.06	0.06
32768	0.24	0.27	0.15	0.11	0.11	0.11
65536	0.52	0.58	0.33	0.25	0.23	0.22
131072	1.10	1.25	0.75	0.58	0.48	0.49
262144	2.37	2.64	1.71	1.39	0.98	0.98
524288	5.10	5.57	5.07	3.91	1.93	2.06
1048576	10.86	11.77	14.07	10.12	4.32	4.44
2097152	22.71	24.45	37.48	26.61	9.05	10.61
4194304	47.63	52.23	90.99	75.76	19.19	23.98
8388608	99.90	108.73	214.78	231.30	39.91	44.70

Código

```
#define BITSWORD    32
#define BITSBYTE    8
#define K           256
#define BYTESWORD   4
#define R           (1 << BITSBYTE)
#define digit(k,d)
     (((k)>>(BITSWORD-(d)*BITSBYTE))&(R-1))
void
radix_sort(int *v, int l, int r)
{
  int i; /* digito */

  for (i = BYTEWORD-1; i >= 0; i--)
    {
      count_sort(v, l, r+1, i);
    }
}
```

Exercícios

Exercício 17.A

O seguinte algoritmo promete rearranjar o vetor $A[1\,..\,n]$ em ordem crescente supondo que cada $A[i]$ está em $\{0,\dots,k\}$. O algoritmo está correto?

```
C-SORT (A, n, k)
    para i ← 0 até k faça
        C[i] ← 0
    para j ← 1 até n faça
        C[A[j]] ← C[A[j]] + 1
    j ← 1
    para i ← 0 até k faça
        enquanto C[i] > 0 faça
            A[j] ← i
            j ← j + 1
            C[i] ← C[i] − 1
```

Qual o consumo de tempo do algoritmo?

Mais exercícios

Exercício 17.B

O seguinte algoritmo promete rearranjar o vetor $A[1\,..\,n]$ em ordem crescente supondo que cada $A[j]$ está em $\{1,\ldots,k\}$. O algoritmo está correto? Estime, em notação O, o consumo de tempo do algoritmo.

VITO-SORT (A, n, k)

```
1  i ← 1
2  para a ← 1 até k − 1 faça
3      para j ← i até n faça
4          se A[j] = a
5              então A[j] ↔ A[i]     ▷ troca
6                  i ← i + 1
```

Exercício 17.C

Suponha que os components do vetor $A[1\,..\,n]$ estão todos em $\{0,1\}$. Prove que $n-1$ comparações são suficientes para rearranjar o vetor em ordem crescente.

Exercício 17.D

Qual a principal invariante do algoritmo RADIX-SORT?

i-ésimo menor elemento

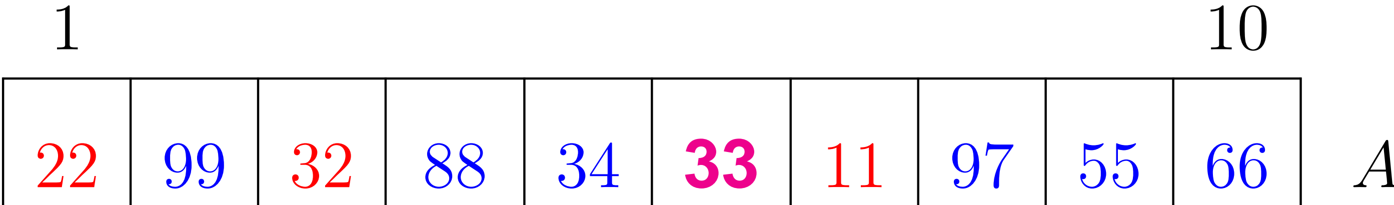
CLRS 9

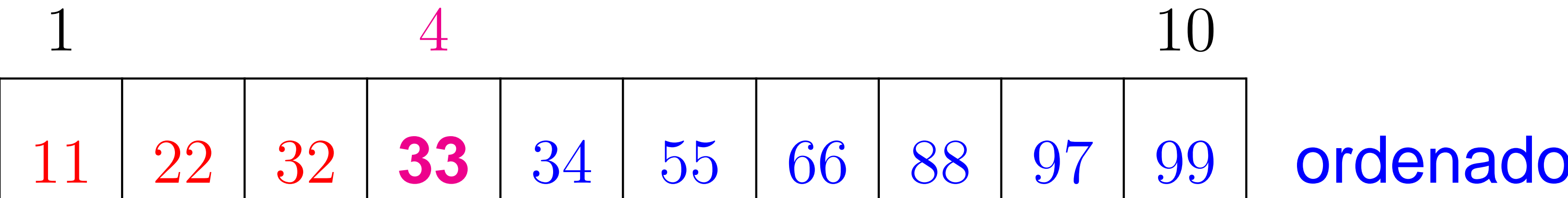
i-ésimo menor

Problema: Encontrar o i-ésimo menor elemento de $A[1\,..\,n]$

Suponha $A[1\,..\,n]$ sem elementos repetidos.

Exemplo: 33 é o 4o. menor elemento de:





Mediana

Mediana é o $\lfloor \frac{n+1}{2} \rfloor$-ésimo menor ou o $\lceil \frac{n+1}{2} \rceil$-ésimo menor elemento

Exemplo: a mediana é 34 ou 55:

1									10	
22	99	32	88	**34**	33	11	97	**55**	66	A

1				5	6				10	
11	22	32	33	**34**	**55**	66	88	97	99	ordenado

Menor

Recebe um vetor $A[1 \mathinner{.\,.} n]$ e devolve o valor do menor elemento.

MENOR (A, n)
1 $\quad$ $\mathrm{menor} \leftarrow A[1]$
2 $\quad$ **para** $k \leftarrow 2$ **até** n **faça**
3 $\quad\quad$ **se** $A[k] < \mathrm{menor}$
4 $\quad\quad\quad$ **então** $\mathrm{menor} \leftarrow A[k]$
5 $\quad$ **devolva** menor

O consumo de tempo do algoritmo MENOR é $\Theta(n)$.

Segundo menor

Recebe um vetor $A[1\,..\,n]$ e devolve o valor do segundo menor elemento, supondo $n \geq 2$.

SEG-MENOR (A, n)

1 $\quad$ $\text{menor} \leftarrow \min\{A[1], A[2]\}$ $\quad$ $\text{segmenor} \leftarrow \max\{A[1], A[2]\}$
2 $\quad$ **para** $k \leftarrow 3$ **até** n **faça**
3 $\quad\quad$ **se** $A[k] < \text{menor}$
4 $\quad\quad\quad$ **então** $\text{segmenor} \leftarrow \text{menor}$
5 $\quad\quad\quad\quad$ $\text{menor} \leftarrow A[k]$
6 $\quad\quad$ **senão se** $A[k] < \text{segmenor}$
7 $\quad\quad\quad$ **então** $\text{segmenor} \leftarrow A[k]$
8 $\quad$ **devolva** segmenor

O consumo de tempo do algoritmo SEG-MENOR é $\Theta(n)$.

i-ésimo menor

Recebe $A[1\,..\,n]$ e i tal que $1 \leq i \leq n$
e devolve valor do i-ésimo menor elemento de $A[1\,..\,n]$

SELECT-ORD (A, n, i)
1 ORDENE (A, n)
2 **devolva** $A[i]$

O consumo de tempo do algoritmo SELECT-ORD é $\Theta(n \lg n)$.

Particione

Rearranja $A[p\,.\,.\,r]$ de modo que $p \le q \le r$ e
$A[p\,.\,.\,q{-}1] \le A[q] < A[q{+}1\,.\,.\,r]$

PARTICIONE (A, p, r)
1 $x \leftarrow A[r]$ $\triangleright$ x é o "pivô"
2 $i \leftarrow p{-}1$
3 **para** $j \leftarrow p$ **até** $r - 1$ **faça**
4 **se** $A[j] \le x$

---

5 **então** $i \leftarrow i + 1$
6 $A[i] \leftrightarrow A[j]$
7 $A[i{+}1] \leftrightarrow A[r]$
8 **devolva** $i + 1$

	p									r
A	99	33	55	77	11	22	88	66	33	44

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \leq q \leq r$ e
$A[p\,..\,q{-}1] \leq A[q] < A[q{+}1\,..\,r]$

PARTICIONE (A, p, r)
1 $x \leftarrow A[r]$ $\triangleright$ x é o "pivô"
2 $i \leftarrow p{-}1$
3 **para** $j \leftarrow p$ **até** $r-1$ **faça**
4 **se** $A[j] \leq x$
5 **então** $i \leftarrow i+1$
6 $A[i] \leftrightarrow A[j]$
7 $A[i{+}1] \leftrightarrow A[r]$
8 **devolva** $i+1$

	p				q					r
A	11	22	33	33	44	55	88	66	77	99

Particione

Rearranja $A[p \,..\, r]$ de modo que $p \le q \le r$ e
$A[p \,..\, q{-}1] \le A[q] < A[q{+}1 \,..\, r]$

PARTICIONE (A, p, r)
1 $\quad x \leftarrow A[r]$ $\qquad \rhd$ x é o "pivô"
2 $\quad i \leftarrow p{-}1$
3 $\quad$ **para** $j \leftarrow p$ **até** $r - 1$ **faça**
4 $\quad\quad$ **se** $A[j] \le x$

---

5 $\quad\quad\quad$ **então** $i \leftarrow i + 1$
6 $\quad\quad\quad\quad A[i] \leftrightarrow A[j]$
7 $\quad A[i{+}1] \leftrightarrow A[r]$
8 $\quad$ **devolva** $i + 1$

O algoritmo PARTICIONE consome tempo $\Theta(n)$.

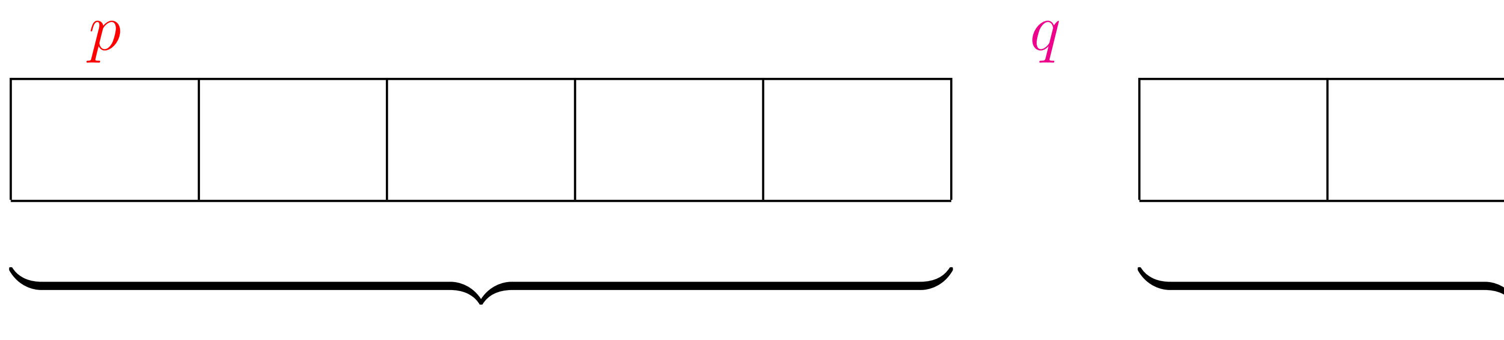
Algoritmo Select

Recebe $A[p\,..\,r]$ e i tal que $1 \leq i \leq r{-}p{+}1$
e devolve valor do i-ésimo menor elemento de $A[p\,..\,r]$

SELECT(A, p, r, i)
1 **se** $p = r$
2 **então devolva** $A[p]$
3 $q \leftarrow$ PARTICIONE (p, r)
4 $k \leftarrow q - p + 1$
5 **se** $k = i$
6 **então devolva** $A[q]$
7 **se** $k > i$
8 **então devolva** SELECT $(A, p, q-1, i)$
9 **senão devolva** SELECT $(A, q+1, r, i-k)$

Algoritmo SELECT

```
SELECT(A, p, r, i)
1    se p = r
2        então devolva A[p]
3    q ← PARTICIONE (A, p, r)
4    k ← q − p + 1
5    se k = i
6        então devolva A[q]
7    se k > i
8        então devolva SELECT (A, p, q − 1, i)
9        senão devolva SELECT (A, q + 1, r, i − k)
```



Consumo de tempo

$T(n)$ = consumo de tempo **máximo** quando $n = r - p + 1$

linha	consumo de todas as execuções da linha
1-2	$= 2\,\Theta(1)$
3	$= \Theta(n)$
4-7	$= 4\,\Theta(1)$
8	$= T(k-1)$
9	$= T(n-k)$

$$T(n) = \Theta(n+6) + \max\{T(k-1), T(n-k)\}$$

$$= \Theta(n) + \max\{T(k-1), T(n-k)\}$$

Consumo de tempo

$T(n)$ pertence a mesma classe Θ que:

$$T'(1) = 1$$
$$T'(n) = T'(n-1) + n \quad \text{para } n = 2, 3, \ldots$$

Solução assintótica:

$$T'(n) \text{ é } \Theta(n^2).$$

Solução exata:

$$T'(n) = \frac{n^2}{2} + \frac{n}{2}.$$

Algumas conclusões

No melhor caso o consumo de tempo do algoritmo SELECT é $\Theta(n)$.

No pior caso o consumo de tempo do algoritmo SELECT é $\Theta(n^2)$.

Consumo médio?
$\mathrm{E}[T(n)] = ???$

Exemplos

Número médio de comparações sobre todas as permutações de $A[p\,..\,r]$ (supondo que nas linhas 8 e 9 o algoritmo sempre escolhe o lado maior):

$A[p\,..\,r]$	comps
1,2	1+0
2,1	1+0
média	2/2

$A[p\,..\,r]$	comps
1,2,3	2+1
2,1,3	2+1
1,3,2	2+0
3,1,2	2+0
2,3,1	2+1
3,2,1	2+1
média	16/6

Mais exemplos

$A[p\,..\,r]$	comps	$A[p\,..\,r]$	comps
1,2,3,4	3+3	1,3,4,2	3+1
2,1,3,4	3+3	3,1,4,2	3+1
1,3,2,4	3+2	1,4,3,2	3+1
3,1,2,4	3+2	4,1,3,2	3+1
2,3,1,4	3+3	3,4,1,2	3+1
3,2,1,4	3+3	4,3,1,2	3+1
1,2,4,3	3+1	2,3,4,1	3+3
2,1,4,3	3+1	3,2,4,1	3+3
1,4,2,3	3+1	2,4,3,1	3+2
4,1,2,3	3+1	4,2,3,1	3+2
2,4,1,3	3+1	3,4,2,1	3+3
4,2,1,3	3+1	4,3,2,1	3+3
		média	116/24

Ainda exemplos

No caso $r - p + 1 = 5$, a média é $864/120$.

n	$\mathrm{E}[T(n)]$	
1	0	0
2	$2/2$	1
3	$16/6$	2.7
4	$116/24$	4.8
5	$864/120$	7.2

Número de comparações

O consumo de tempo assintótico é proporcional a

$C(n) =$ número de comparações entre elementos de A
quando $n = r - p + 1$

linha	número de comparações na linha
1-2	$= 0$
3	$= n - 1$
4-7	$= 0$
8	$= C(k-1)$
9	$= C(n-k)$

total $\leq \max\{C(k-1), C(n-k)\} + n - 1$

Número de comparações

No pior caso $C(n)$ pertence a mesma classe Θ que:

$$C'(1) = 0$$

$$C'(n) = C'(n-1) + n - 1 \quad \text{para } n = 3, 4, \ldots$$

Solução assintótica:

$$C'(n) \text{ é } \Theta(n^2)$$

Solução exata:

$$C'(n) = \frac{n^2}{2} - \frac{n}{2}.$$

Particione aleatorizado

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e $A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

PARTICIONE-ALEA(A, p, r)
1 $i \leftarrow$ RANDOM (p, r)
2 $A[i] \leftrightarrow A[r]$
3 **devolva** PARTICIONE (A, p, r)

O algoritmo PARTICIONE-ALEA consome tempo $\Theta(n)$.

SELECT-ALEATORIZADO (= randomized select)

Recebe $A[p \,.\,.\, r]$ e i tal que $1 \le i \le r-p+1$
e devolve valor do i-ésimo menor elemento de $A[p \,.\,.\, r]$

SELECT-ALEA(A, p, r, i)
1 **se** $p = r$
2 **então devolva** $A[p]$
3 $q \leftarrow$ PARTICIONE-ALEA (A, p, r)
4 $k \leftarrow q - p + 1$
5 **se** $k = i$
6 **então devolva** $A[q]$
7 **se** $k > i$
8 **então devolva** SELECT-ALEA $(A, p, q-1, i)$
9 **senão devolva** SELECT-ALEA $(A, q+1, r, i-k)$

Consumo de tempo

O consumo de tempo é proporcional a

$T(n) =$ número de comparações entre elementos de A
quando $n = r - p + 1$

linha	número de comparações na linha
1-2	$= 0$
3	$= n - 1$
4-7	$= 0$
8	$= T(k-1)$
9	$= T(n-k)$
total	$\leq \max\{T(k-1), T(n-k)\} + n - 1$

$T(n)$ é uma variável aleatória.

Consumo de tempo

$$T(1) = 0$$

$$T(n) \leq \sum_{h=1}^{n-1} X_h T(h) + n - 1 \quad \text{para } n = 2, 3, \ldots$$

onde

$$X_h = \begin{cases} 1 & \text{se } \max\{k-1, n-k\} = h \\ 0 & \text{caso contrário} \end{cases}$$

$\Pr\{X_h = 1\} = \mathrm{E}[X_h]$

$$X_h = \begin{cases} 1 & \text{se } \max\{k-1, n-k\} = h \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_h valer 1?

$\Pr\{X_h = 1\} = \mathrm{E}[X_h]$

$$X_h = \begin{cases} 1 & \text{se } \max\{k-1, n-k\} = h \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_h valer 1?

Para $h = 1, \ldots, \lfloor n/2 \rfloor - 1$, $\Pr\{X_h = 1\} = 0 = \mathrm{E}[X_h]$.

Para $h = \lceil n/2 \rceil, \ldots, n$,

$$\text{Pr}\,\{X_h{=}1\} = \frac{1}{n} + \frac{1}{n} = \frac{2}{n} = \mathrm{E}[X_h]$$

Se n é impar e $h = \lfloor n/2 \rfloor$, então

$$\Pr\{X_h = 1\} = \frac{1}{n} = \mathrm{E}[X_h]$$

Consumo de tempo esperado

$$\mathrm{E}[T(1)] = 0$$

$$\mathrm{E}[T(n)] \leq \sum_{h=1}^{n-1} \mathrm{E}[X_h T(h)] + n - 1$$

$$= \sum_{h=1}^{n-1} \mathrm{E}[X_h]\, \mathrm{E}[T(h)] + n - 1 \quad \text{(CLRS 9.2-2)}$$

$$\leq \frac{2}{n} \sum_{h=a}^{n-1} \mathrm{E}[T(h)] + n - 1 \quad \text{para } n = 2, 3, \ldots$$

onde $a = \lfloor n/2 \rfloor$.

Solução: $\mathrm{E}[T(n)] = \mathrm{O}(n)$.

Consumo de tempo esperado

$\mathrm{E}[T(n)]$ pertence a mesma classe O que:

$$S(1) = 0$$

$$S(n) \le \frac{2}{n} \sum_{h=a}^{n-1} S(h) + n - 1 \quad \text{para } n = 2, 3, \ldots$$

onde $a = \lfloor n/2 \rfloor$.

n	1	2	3	4	5	6	7	8	9	10
$S(n)$	0.0	1.0	2.7	4.8	7.4	10.0	13.1	15.8	19.4	22.1
$4n$	4	8	12	16	20	24	28	32	36	40

Vamos verificar que $S(n) < 4\,n$ para $n = 1, 2, 3, 4, \ldots$

Recorrência

Prova: Se $n = 1$, então $S(n) = 0 < 4 = 4 \cdot 1 = 4n$. Se $n \geq 2$,

$$
\begin{aligned}
S(n) &\leq \frac{2}{n} \sum_{h=a}^{n-1} S(h) + n - 1 \\
&\overset{\text{hi}}{<} \frac{2}{n} \sum_{h=a}^{n-1} 4\,h + n - 1 \\
&= \frac{8}{n} \Big(\sum_{h=1}^{n-1} h - \sum_{h=1}^{a-1} h \Big) + n - 1 \\
&\leq \frac{4}{n} \Big(n^2 - n - \frac{(n-1)(n-3)}{4} \Big) + n - 1 \\
&= \frac{4}{n} \Big(\frac{3n^2}{4} - \frac{3}{4} \Big) + n - 1 \\
&= 3n - \frac{3}{n} + n - 1 = 4n - \frac{3}{n} - 1 < 4n.
\end{aligned}
$$

Conclusão

O consumo de tempo esperado do algoritmo SELECT-ALEA é $O(n)$.

Exercícios

Exercício 18.A [CLRS 9.1-1] [muito bom!]
Mostre que o segundo menor elemento de um vetor $A[1\,..\,n]$ pode ser encontrado com não mais que $n + \lceil \lg n \rceil - 2$ comparações.

Exercício 18.B
Prove que o algoritmo Select Aleatorizado (= Randomized Select) funciona corretamente.

Exercício 18.C [CLRS 9.2-3]
Escreva uma versão iterativa do algoritmo Select Aleatorizado (= Randomized Select).

Melhores momentos

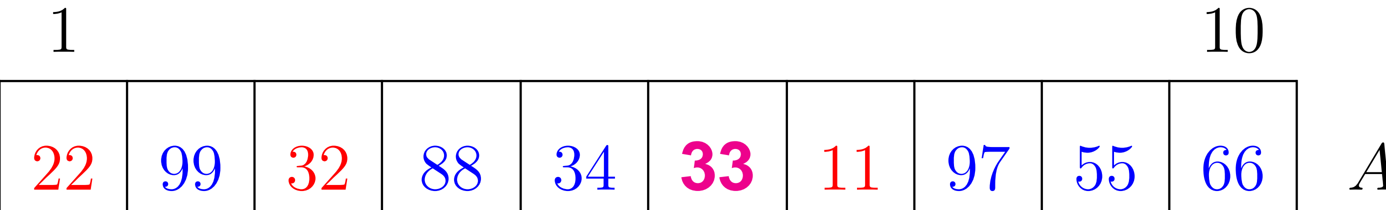
AULA 13

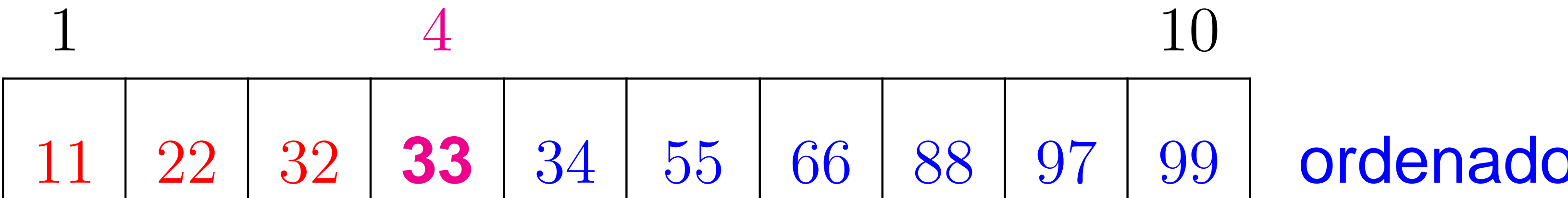
i-ésimo menor

Problema: Encontrar o i-ésimo menor elemento de $A[1\,..\,n]$

Suponha $A[1\,..\,n]$ sem elementos repetidos.

Exemplo: 33 é o 4o. menor elemento de:





Particione

Rearranja $A[p\,..\,r]$ de modo que $p \leq q \leq r$ e
$A[p\,..\,q{-}1] \leq A[q] < A[q{+}1\,..\,r]$

PARTICIONE (A, p, r)
1 $x \leftarrow A[r]$ $\triangleright$ x é o "pivô"
2 $i \leftarrow p{-}1$
3 **para** $j \leftarrow p$ **até** $r-1$ **faça**
4 **se** $A[j] \leq x$
5 **então** $i \leftarrow i+1$
6 $A[i] \leftrightarrow A[j]$
7 $A[i{+}1] \leftrightarrow A[r]$
8 **devolva** $i+1$

	p									r
A	99	33	55	77	11	22	88	66	33	44

Particione

Rearranja $A[p \mathinner{.\,.} r]$ de modo que $p \le q \le r$ e
$A[p \mathinner{.\,.} q{-}1] \le A[q] < A[q{+}1 \mathinner{.\,.} r]$

PARTICIONE (A, p, r)
1 $\quad x \leftarrow A[r]$ $\qquad \triangleright$ x é o "pivô"
2 $\quad i \leftarrow p{-}1$
3 $\quad$ **para** $j \leftarrow p$ **até** $r-1$ **faça**
4 $\qquad$ **se** $A[j] \le x$
5 $\qquad\quad$ **então** $i \leftarrow i+1$
6 $\qquad\qquad A[i] \leftrightarrow A[j]$
7 $\quad A[i{+}1] \leftrightarrow A[r]$
8 $\quad$ **devolva** $i+1$

	p				q					r
A	11	22	33	33	44	55	88	66	77	99

Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

PARTICIONE (A, p, r)
1 $x \leftarrow A[r]$ $\triangleright$ x é o "pivô"
2 $i \leftarrow p{-}1$
3 **para** $j \leftarrow p$ **até** $r - 1$ **faça**
4 **se** $A[j] \le x$

5 **então** $i \leftarrow i + 1$
6 $A[i] \leftrightarrow A[j]$
7 $A[i{+}1] \leftrightarrow A[r]$
8 **devolva** $i + 1$

O algoritmo PARTICIONE consome tempo $\Theta(n)$.

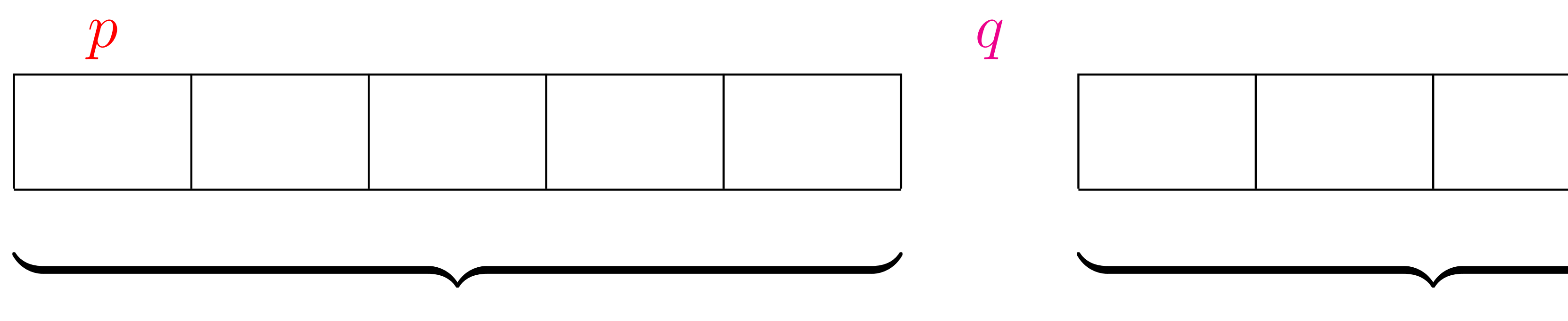
Algoritmo Select

Recebe $A[p\,..\,r]$ e i tal que $1 \leq i \leq r-p+1$
e devolve um índice q tal que $A[q]$ é o i-ésimo menor
elemento de $A[p\,..\,r]$

SELECT(A, p, r, i)
1 **se** $p = r$
2 **então devolva** p $\triangleright$ p e não $A[p]$
3 $q \leftarrow$ PARTICIONE (p, r)
4 $k \leftarrow q - p + 1$
5 **se** $k = i$
6 **então devolva** q $\triangleright$ q e não $A[q]$
7 **se** $k > i$
8 **então devolva** SELECT $(A, p, q-1, i)$
9 **senão devolva** SELECT $(A, q+1, r, i-k)$

Algoritmo SELECT

SELECT(A, p, r, i)
1 **se** $p = r$
2 **então devolva** p $\triangleright$ p e não $A[p]$
3 $q \leftarrow$ PARTICIONE (A, p, r)
4 $k \leftarrow q - p + 1$
5 **se** $k = i$
6 **então devolva** q $\triangleright$ q e não $A[q]$
7 **se** $k > i$
8 **então devolva** SELECT $(A, p, q-1, i)$
9 **senão devolva** SELECT $(A, q+1, r, i-k)$



Algumas conclusões

No melhor caso o consumo de tempo do algoritmo SELECT é $\Theta(n)$.

No pior caso o consumo de tempo do algoritmo SELECT é $\Theta(n^2)$.

Consumo de tempo

$T(n)$ = consumo de tempo máximo quando $n = r - p + 1$

linha	consumo de todas as execuções da linha
1-2	$= 2\,\Theta(1)$
3	$= \Theta(n)$
4-7	$= 4\,\Theta(1)$
8	$= T(k-1)$
9	$= T(n-k)$

$$T(n) = \Theta(n+6) + \max\{T(k-1), T(n-k)\}$$

$$= \Theta(n) + \max\{T(k-1), T(n-k)\}$$

Particione aleatorizado

Rearranja $A[p \,.\,.\, r]$ de modo que $p \leq q \leq r$ e
$A[p \,.\,.\, q{-}1] \leq A[q] < A[q{+}1 \,.\,.\, r]$

PARTICIONE-ALEA(A, p, r)
1 $\quad i \leftarrow$ RANDOM (p, r)
2 $\quad A[i] \leftrightarrow A[r]$
3 $\quad$ **devolva** PARTICIONE (A, p, r)

O algoritmo PARTICIONE-ALEA consome tempo $\Theta(n)$.

SELECT-ALEATORIZADO (= randomized select)

Recebe $A[p \mathinner{.\,.} r]$ e i tal que $1 \leq i \leq r - p + 1$
e devolve um índice q tal que $A[q]$ é o i-ésimo menor
elemento de $A[p \mathinner{.\,.} r]$

SELECT-ALEA(A, p, r, i)
1 **se** $p = r$
2 **então devolva** p $\triangleright$ p e não $A[p]$
3 $q \leftarrow$ PARTICIONE-ALEA (A, p, r)
4 $k \leftarrow q - p + 1$
5 **se** $k = i$
6 **então devolva** q $\triangleright$ q e não $A[q]$
7 **se** $k > i$
8 **então devolva** SELECT-ALEA $(A, p, q - 1, i)$
9 **senão devolva** SELECT-ALEA $(A, q + 1, r, i - k)$

Conclusão

O consumo de tempo esperado do algoritmo SELECT-ALEA é $O(n)$.

AULA 14

Seleção em tempo linear

CLRS 9.3

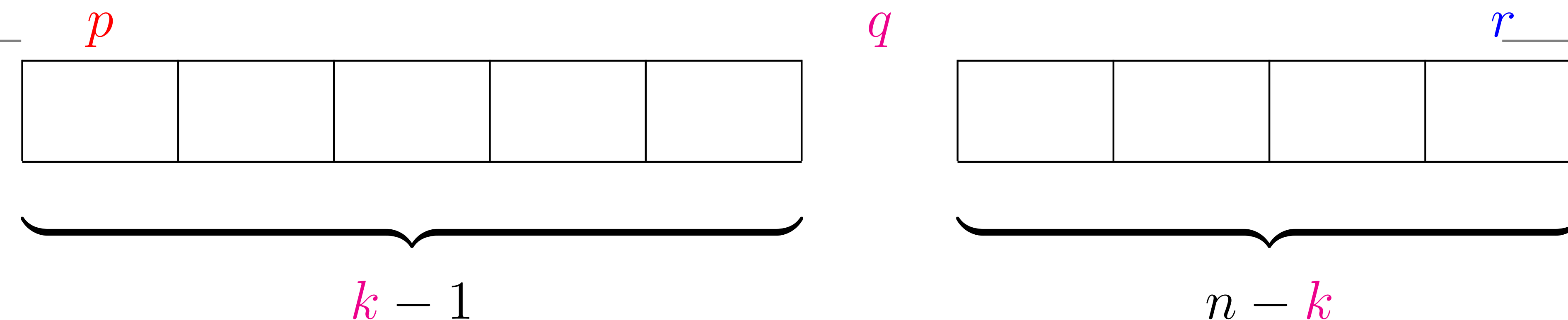
BFPRT = Blum, Floyd, Pratt, Rivest e Tarjan

Select-BFPRT

Recebe $A[p\,.\,.\,r]$ e i tal que $1 \leq i \leq r - p + 1$
e devolve um índice q tal que $A[q]$ é o i-ésimo menor
elemento de $A[p\,.\,.\,r]$

SELECT-BFPRT(A, p, r, i)
1 **se** $p = r$
2 **então devolva** p $\triangleright$ p e não $A[p]$
3 $q \leftarrow$ PARTICIONE-BFPRT (A, p, r)
4 $k \leftarrow q - p + 1$
5 **se** $k = i$
6 **então devolva** q $\triangleright$ q e não $A[q]$
7 **se** $k > i$
8 **então devolva** SELECT-BFPRT $(A, p, q - 1, i)$
9 **senão devolva** SELECT-BFPRT $(A, q + 1, r, i - k)$

Particione-BFPRT



Rearranja $A[p \mathinner{..} r]$ e devolve um índice q, $p \leq q \leq r$, tal que $A[p \mathinner{..} q-1] \leq A[q] < A[q+1 \mathinner{..} r]$ **e**

$$\max\{k-1, n-k\} \leq \left\lfloor \frac{7n}{10} \right\rfloor + 6,$$

onde $n = r - p + 1$ e $k = q - p + 1$.

Suponha que

$P(n) :=$ consumo de tempo máximo do algoritmo PARTICIONE-BFPRT quando $n = r - p + 1$

Consumo de tempo

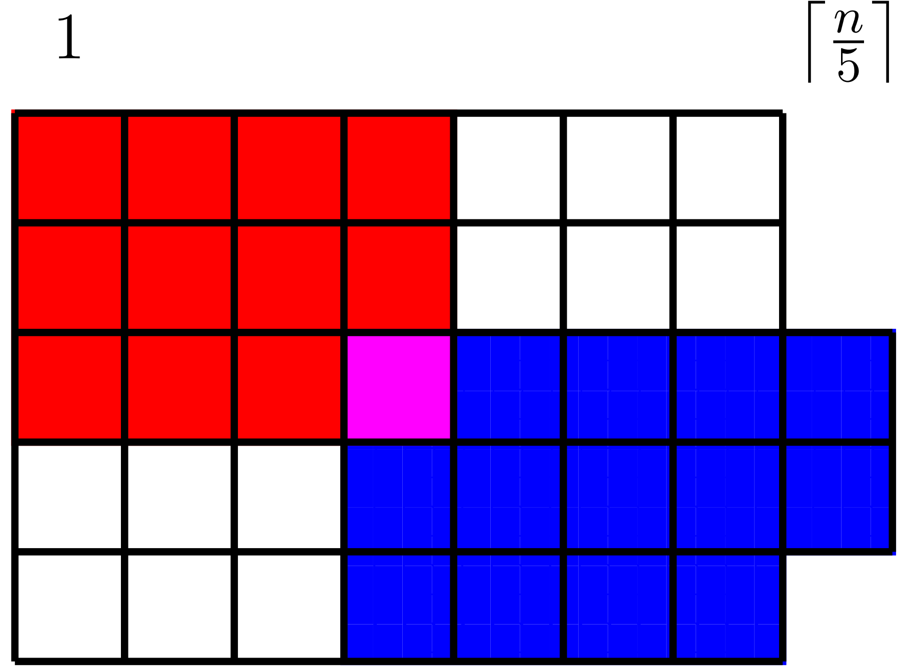
$T(n) :=$ consumo de tempo máximo do algoritmo SELECT-BFPRT quando $n = r - p + 1$

linha	consumo de todas as execuções da linha
1-2	$= 2\,\Theta(1)$
3	$= P(n)$
4-7	$= 4\,\Theta(1)$
8	$= T(k-1)$
9	$= T(n-k)$

$$T(n) = 6\,\Theta(1) + P(n) + \max\{T(k-1), T(n-k)\}$$

$$\leq \Theta(1) + P(n) + T(\lfloor \tfrac{7n}{10} \rfloor + 6)$$

Particione-BFPRT



$$\max\{k-1, n-k\} \le n - 3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right)$$

$$\le n - \left(\frac{3n}{10} - 6\right) = \frac{7n}{10} + 6$$

Particione-BFPRT

$n := r - p + 1$

PARTICIONE-BFPRT (A, p, r)

1 **para** $j \leftarrow p, p{+}5, p{+}5 \cdot 2, \ldots$ **até** $p{+}5(\lceil n/5 \rceil{-}1)$ **faça**

2 ORDENE $(A, j, j{+}4)$

3 ORDENE $(A, p{+}5\lfloor n/5 \rfloor, n)$

4 **para** $j \leftarrow 1$ **até** $\lceil n/5 \rceil{-}1$ **faça**

5 $B[j] \leftarrow A[p{+}5j{-}3]$

6 $B[\lceil n/5 \rceil] \leftarrow A[\lfloor (p{+}5\lfloor n/5 \rfloor{+}n)/2 \rfloor]$

7 $k \leftarrow$ SELECT-BFPRT$(B, 1, \lceil n/5 \rceil, \lfloor (\lceil n/5 \rceil{+}1)/2 \rfloor)$

8 $A[k] \leftrightarrow A[r]$

9 **devolva** PARTICIONE (A, p, r)

Particione-BFPRT

$n := r - p + 1$

PARTICIONE-BFPRT (A, p, r)
1 **para** $j \leftarrow p, p{+}5, p{+}5 \cdot 2, \ldots$ **até** $p{+}5(\lceil n/5 \rceil{-}1)$ **faça**
2 ORDENE $(A, j, j{+}4)$
3 ORDENE $(A, p{+}5\lfloor n/5 \rfloor, n)$

4 **para** $j \leftarrow 1$ **até** $\lceil n/5 \rceil{-}1$ **faça**
5 $A[j] \leftrightarrow A[p{+}5j{-}3]$
6 $A[\lceil n/5 \rceil] \leftrightarrow A[\lfloor (p{+}5\lfloor n/5 \rfloor{+}n)/2 \rfloor]$

7 $k \leftarrow$ SELECT-BFPRT$(A, p, p{+}\lceil n/5 \rceil{-}1, \lfloor (\lceil n/5 \rceil{+}1)/2 \rfloor)$

8 $A[k] \leftrightarrow A[r]$
9 **devolva** PARTICIONE (A, p, r)

Consumo de tempo do Particione-BFPRT

$P(n) :=$ consumo de tempo máximo do algoritmo PARTICIONE-BFPRT quando $n = r - p + 1$

linha	consumo de todas as execuções da linha
1-3	$= \lceil n/5 \rceil \, \Theta(1)$
4-6	$= \lceil n/5 \rceil \, \Theta(1)$
7	$= T(\lceil n/5 \rceil)$
8	$= \Theta(1)$
9	$= \Theta(n)$

$$P(n) = \Theta(2\lceil n/5 \rceil + n + 1) + T(\lceil n/5 \rceil)$$

$$= \Theta(n) + T(\lceil n/5 \rceil)$$

Consumo de tempo do Select-BFPRT

$T(n) :=$ consumo de tempo máximo do algoritmo SELECT-BFPRT quando $n = r - p + 1$

Temos que

$$T(1) = \Theta(1) \quad \text{para } n < 30$$

$$\begin{aligned} T(n) &\leq \Theta(1) + P(n) + T\left(\left\lfloor \frac{7n}{10} \right\rfloor + 6\right) \\ &\leq \Theta(1) + \Theta(n) + T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lfloor \frac{7n}{10} \right\rfloor + 6\right) \\ &= \Theta(n) + T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lfloor \frac{7n}{10} \right\rfloor + 6\right) \end{aligned}$$

para $n = 30, 31, \ldots$,

Consumo de tempo do Select-BFPRT

$T(n)$ pertence a mesma classe O que:

$$S(n) = 1 \quad \text{para } n < 30$$

$$S(n) \leq S\left(\left\lceil \frac{n}{5} \right\rceil\right) + S\left(\left\lfloor \frac{7n}{10} \right\rfloor + 6\right) + n \quad \text{para } n \geq 30$$

n	30	60	90	120	150	180	210	240	270	300
$S(n)$	32	185	330	451	572	732	902	1040	1224	1439

Vamos verificar que $S(n) < 80\,n$ para $n = 1, 2, 3, 4, \ldots$

Prova: Se $n = 1, \ldots, 29$, então $S(n) = 1 < 80 < 80n$.
Se $n = 30, \ldots, 99$, então

$$S(n) < S(120) = 451 < 80 \times 30 \leq 80n.$$

Recorrência

Se $n \geq 100$, então

$$
\begin{aligned}
S(n) &\leq S\left(\left\lceil \frac{n}{5} \right\rceil\right) + S\left(\left\lfloor \frac{7n}{10} \right\rfloor + 6\right) + n \\
&\overset{\text{hi}}{<} 80 \left\lceil \frac{n}{5} \right\rceil + 80 \left(\left\lfloor \frac{7n}{10} \right\rfloor + 6\right) + n \\
&\leq 80 \left(\frac{n}{5} + 1\right) + 80 \left(\frac{7n}{10} + 6\right) + n \\
&= 80\frac{n}{5} + 80 + 80\frac{7n}{10} + 480 + n \\
&= 16n + 56n + n + 560 \\
&= 73n + 560 \\
&< 80n \quad \text{(pois } n \geq 100\text{).}
\end{aligned}
$$

Logo, $T(n)$ é $\mathrm{O}(n)$.

Conclusão

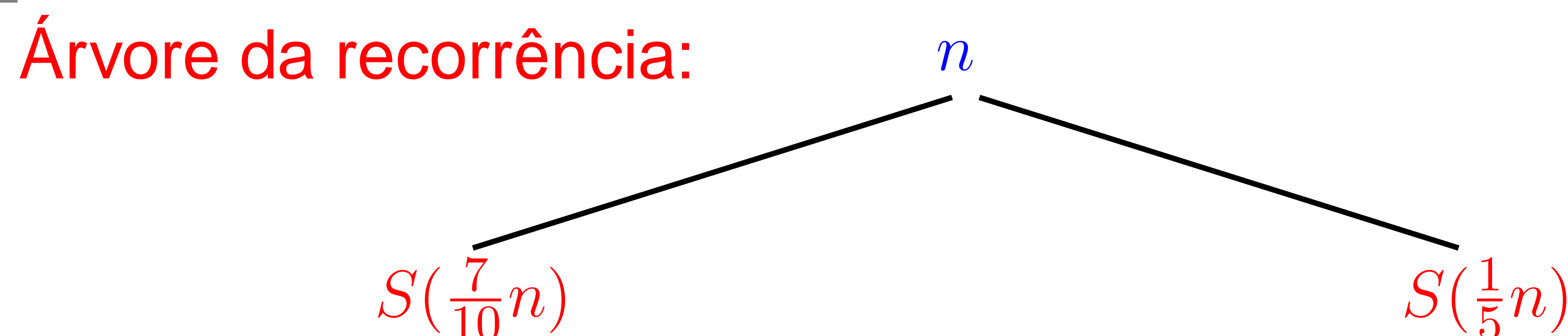
O consumo de tempo do algoritmo SELECT-BFPRT é $O(n)$.

Como adivinhei classe O?

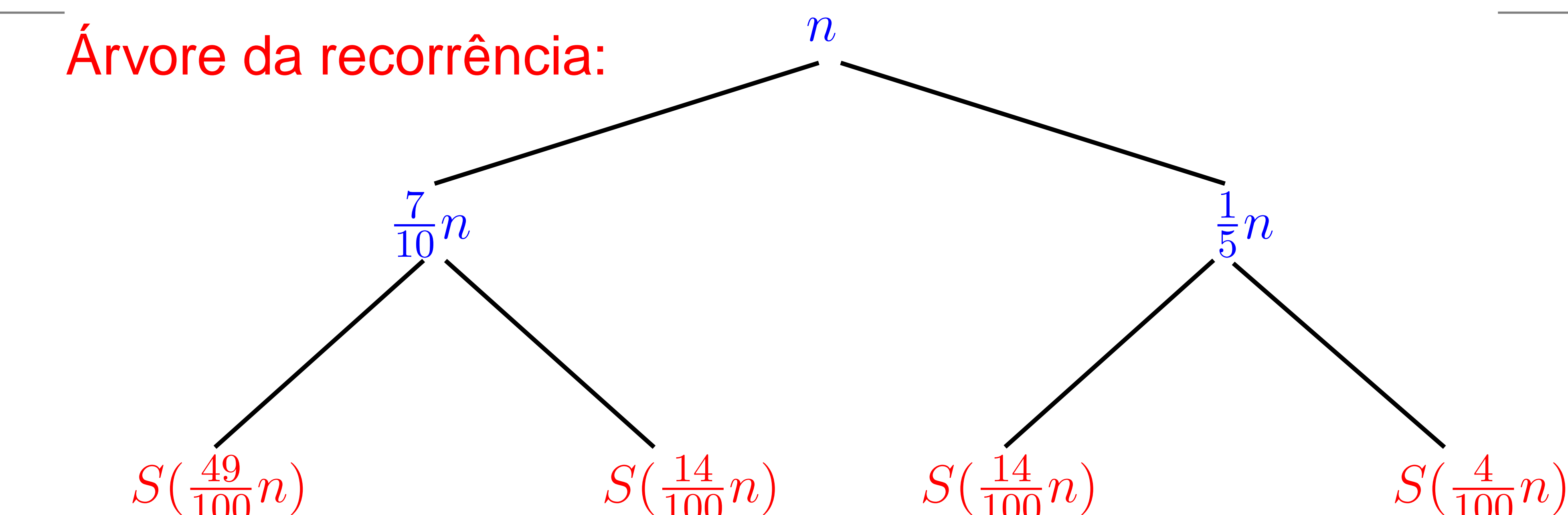
Árvore da recorrência: $S(n)$

Como adivinhei classe O?

Árvore da recorrência:

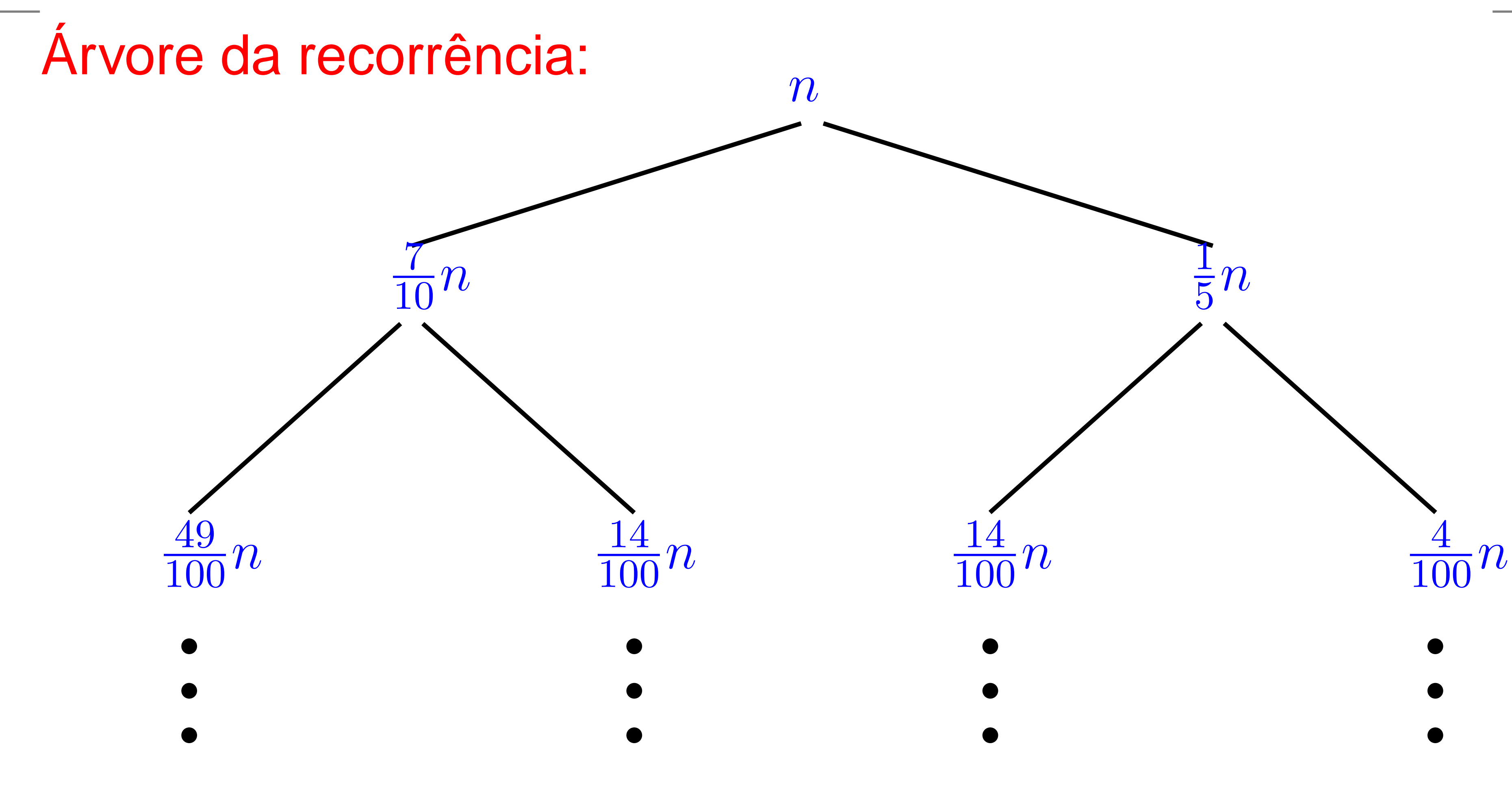


Como adivinhei classe O?



Como adivinhei classe O?

Árvore da recorrência:



Contabilidade

nível	0	1	2	$\ldots$	$k-1$	k
soma	n	$\frac{9}{10}n$	$\frac{9^2}{10^2}n$	$\ldots$	$\frac{9^{k-1}}{10^{k-1}}n$	$\frac{9^k}{10^k}n$

$$\frac{10^{k-1}}{9^{k-1}} < n \leq \frac{10^k}{9^k} \Rightarrow k = \lceil \log_{\frac{10}{9}} n \rceil$$

$$\begin{aligned}
S(n) &= n + \frac{9}{10}n + \cdots + \frac{9^{k-1}}{10^{k-1}}n + \frac{9^k}{10^k}n \\
&= (1 + \frac{9}{10} + \cdots + \frac{9^k}{10^k})n \\
&= 10(1 - \frac{9^{k+1}}{10^{k+1}})n \\
&< 10n
\end{aligned}$$

AULA 15

Programação dinâmica

CLRS 15.1–15.3

= "recursão–com–tabela"

= transformação inteligente de recursão em iteração

Programação dinâmica

"Dynamic programming is a fancy name for divide-and-conquer with a table. Instead of solving subproblems recursively, solve them sequentially and store their solutions in a table. The trick is to solve them in the right order so that whenever the solution to a subproblem is needed, it is already available in the table. Dynamic programming is particularly useful on problems for which divide-and-conquer appears to yield an exponential number of subproblems, but there are really only a small number of subproblems repeated exponentially often. In this case, it makes sense to compute each solution the first time and store it away in a table for later use, instead of recomputing it recursively every time it is needed."

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Números de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Números de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Algoritmo recursivo para F_n:

FIBO-REC (n)
1 **se** $n \leq 1$
2 **então devolva** n
3 **senão** $a \leftarrow$ FIBO-REC $(n-1)$
4 $b \leftarrow$ FIBO-REC $(n-2)$
5 **devolva** $a + b$

Consumo de tempo

```
FIBO-REC (n)
1    se n ≤ 1
2        então devolva n
3        senão a ← FIBO-REC (n − 1)
4              b ← FIBO-REC (n − 2)
5              devolva a + b
```

n	16	32	40	41	42	43	44	45	47
tempo	0.002	0.06	2.91	4.71	7.62	12.37	19.94	32.37	84.50

tempo em segundos.

$F_{47} = 2971215073$

Consumo de tempo

$T(n) :=$ número de somas feitas por FIBO-REC (n)

linha	número de somas
1-2	$= 0$
3	$= T(n-1)$
4	$= T(n-2)$
5	$= 1$

$$T(n) = T(n-1) + T(n-2) + 1$$

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \text{ para } n = 2, 3, \ldots$$

A que classe Ω pertence $T(n)$?
A que classe O pertence $T(n)$?

Recorrência

$$T(0) = 0$$

$$T(1) = 0$$

$$T(n) = T(n-1) + T(n-2) + 1 \text{ para } n = 2, 3, \ldots$$

A que classe Ω pertence $T(n)$?
A que classe O pertence $T(n)$?

Solução: $T(n) > (3/2)^n$ para $n \geq 6$.

n	0	1	2	3	4	5	6	7	8	9
T_n	0	0	1	2	4	7	12	20	33	54
$(3/2)^n$	1	1.5	2.25	3.38	5.06	7.59	11.39	17.09	25.63	38.4

Recorrência

Prova: $T(6) = 12 > 11.40 > (3/2)^6$ e $T(7) = 20 > 18 > (3/2)^7$.
Se $n \geq 8$, então

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&\stackrel{\text{hi}}{>} (3/2)^{n-1} + (3/2)^{n-2} + 1 \\
&= (3/2 + 1)\,(3/2)^{n-2} + 1 \\
&> (5/2)\,(3/2)^{n-2} \\
&> (9/4)\,(3/2)^{n-2} \\
&= (3/2)^2 (3/2)^{n-2} \\
&= (3/2)^n\,.
\end{aligned}
$$

Logo, $T(n)$ é $\Omega((3/2)^n)$. Verifique que $T(n)$ é $\mathrm{O}(2^n)$.

Exercícios

Prove que

$$T(n) = \frac{\phi^{n+1} - \hat{\phi}^{n+1}}{\sqrt{5}} - 1 \quad \text{para } n = 0, 1, 2, \ldots$$

onde

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}61803 \quad \text{e} \quad \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0{,}61803.$$

Prove que $1 + \phi = \phi^2$.

Prove que $1 + \hat{\phi} = \hat{\phi}^2$.

Mais exercícios

[CLRS 3.2-6] Prove que

$$F_n = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n) \quad \text{para } n = 0, 1, 2, \ldots$$

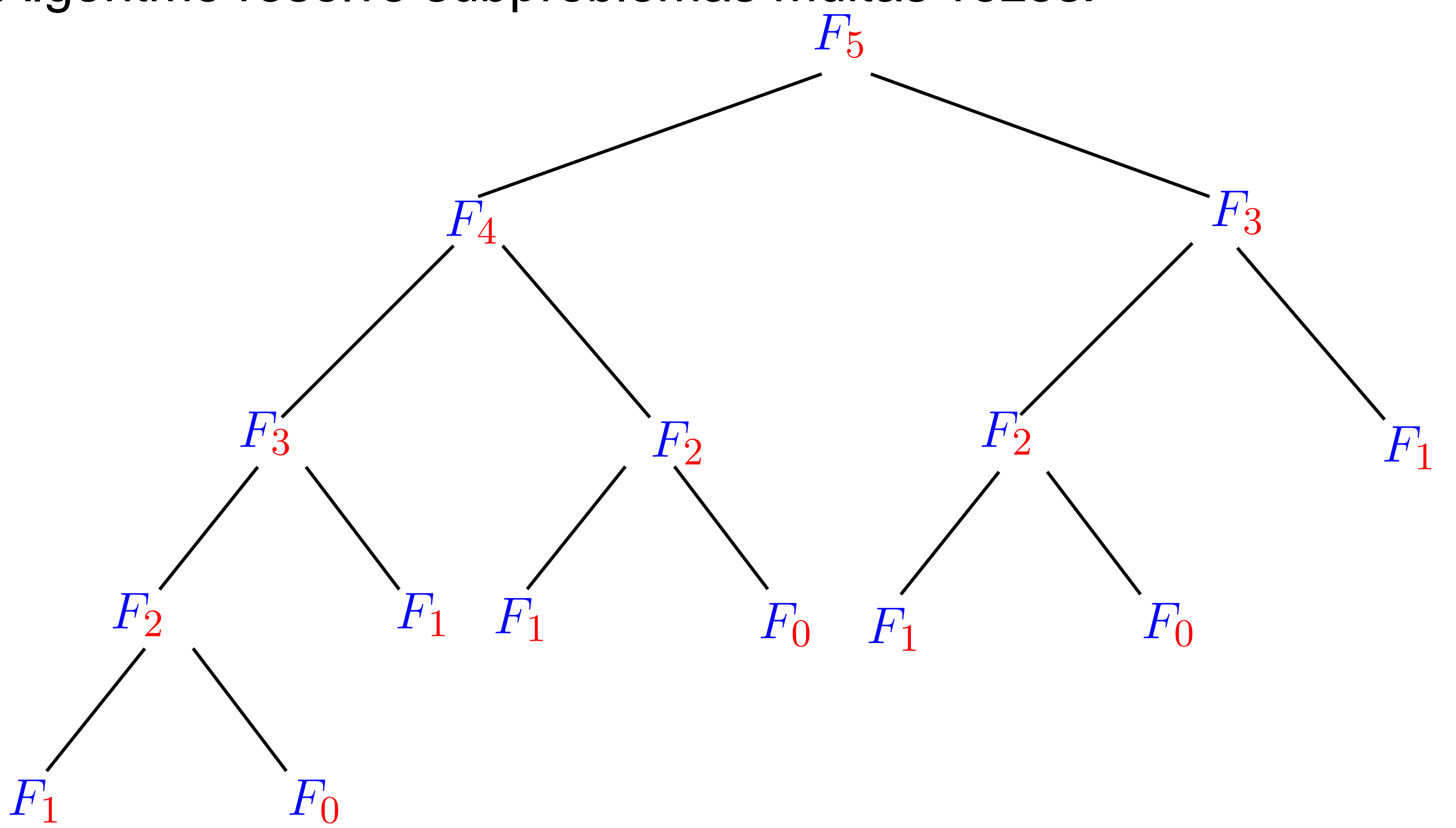
[CLRS 3.2-7] Prove que $F_{i+2} \geq \phi^i$ para $i = 2, 3, \ldots$

Veja o exercício [CLRS 4.5].

Consumo de tempo

Consumo de tempo é exponencial.

Algoritmo resolve subproblemas muitas vezes.



Resolve subproblemas muitas vezes

```
FIBO-REC(5)
  FIBO-REC(4)
    FIBO-REC(3)
      FIBO-REC(2)
        FIBO-REC(1)
        FIBO-REC(0)
      FIBO-REC(1)
    FIBO-REC(2)
      FIBO-REC(1)
      FIBO-REC(0)
  FIBO-REC(3)
    FIBO-REC(2)
      FIBO-REC(1)
      FIBO-REC(0)
    FIBO-REC(1)
```

FIBO-REC(5) = 5

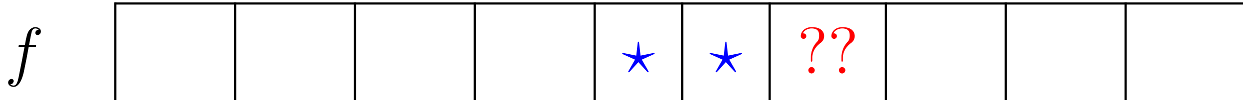
Resolve subproblemas muitas vezes

```
FIBO-REC(8)
  FIBO-REC(7)
    FIBO-REC(6)
      FIBO-REC(5)
        FIBO-REC(4)
          FIBO-REC(3)
            FIBO-REC(2)
              FIBO-REC(1)
              FIBO-REC(0)
            FIBO-REC(1)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
    FIBO-REC(5)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
      FIBO-REC(3)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
        FIBO-REC(1)
  FIBO-REC(6)
    FIBO-REC(5)
      FIBO-REC(4)
        FIBO-REC(3)
            FIBO-REC(2)
              FIBO-REC(1)
              FIBO-REC(0)
            FIBO-REC(1)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
      FIBO-REC(4)
        FIBO-REC(3)
          FIBO-REC(2)
            FIBO-REC(1)
            FIBO-REC(0)
          FIBO-REC(1)
        FIBO-REC(2)
          FIBO-REC(1)
          FIBO-REC(0)
```

Algoritmo de programação dinâmica

FIBO (n)
1 $f[0] \leftarrow 0$
2 $f[1] \leftarrow 1$
3 **para** $i \leftarrow 2$ **até** n **faça**
4 $f[i] \leftarrow f[i-1] + f[i-2]$
5 **devolva** $f[n]$

Note a tabela $f[0\,..\,n-1]$.



Consumo de tempo é $\Theta(n)$.

Algoritmo de programação dinâmica

Versão com economia de espaço.

```
FIBO (n)
0    se n = 0 então devolva 0
1    f_ant ← 0
2    f_atual ← 1
3    para i ← 2 até n faça
4        f_prox ← f_atual + f_ant
5        f_ant ← f_atual
6        f_atual ← f_prox
7    devolva f_atual
```

Versão recursiva eficiente

MEMOIZED-FIBO (f, n)

1 **para** $i \leftarrow 0$ **até** n **faça**

2 $f[i] \leftarrow -1$

3 **devolva** LOOKUP-FIBO (f, n)

LOOKUP-FIBO (f, n)

1 **se** $f[n] \geq 0$

2 **então devolva** $f[n]$

3 **se** $n \leq 1$

4 **então** $f[n] \leftarrow n$

5 **senão** $f[n] \leftarrow$ LOOKUP-FIBO$(f, n-1)$ $+$ LOOKUP-FIBO$(f, n-2)$

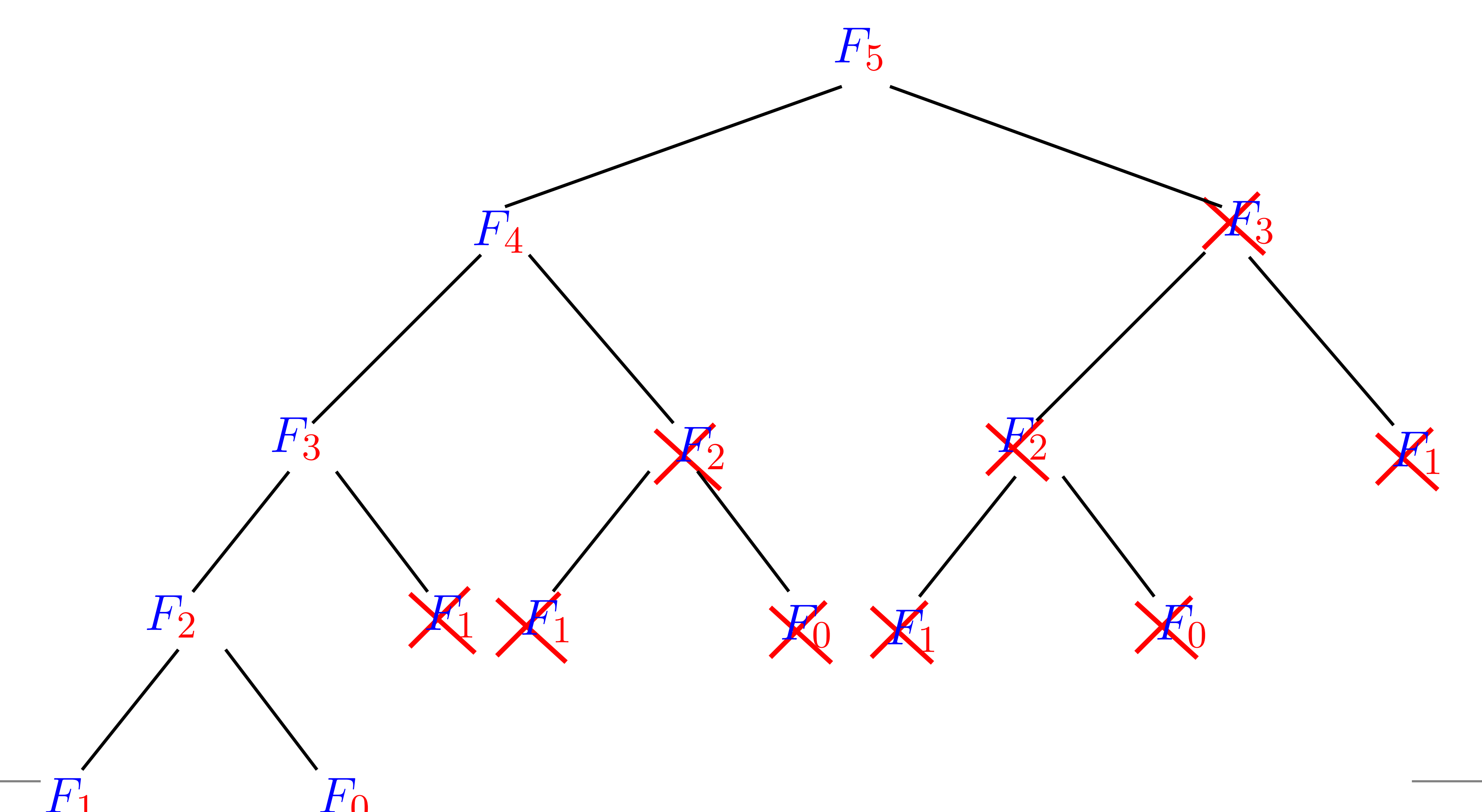
6 **devolva** $f[n]$

Não recalcula valores de f.

Consumo de tempo

Consumo de tempo é $\Theta(n)$.

Algoritmo resolve cada subproblemas apenas uma vezes.



Multiplicação iterada de matrizes

Se A é $p \times q$ e B é $q \times r$ então AB é $p \times r$.

$(AB)[i,j] = \sum_k A[i,k]\, B[k,j]$

MULT-MAT (p, A, q, B, r)

1 **para** $i \leftarrow 1$ **até** p **faça**
2 **para** $j \leftarrow 1$ **até** r **faça**
3 $AB[i,j] \leftarrow 0$
4 **para** $k \leftarrow 1$ **até** q **faça**
5 $AB[i,j] \leftarrow AB[i,j] + A[i,k] \cdot B[k,j]$

Número de multiplicações escalares = $p \cdot q \cdot r$

Multiplicação iterada

Problema: Encontrar número mínimo de multiplicações escalares necessário para calcular produto $A_1 A_2 \cdots A_n$.

$$
\begin{array}{ccccccccccc}
p[0] & & p[1] & & p[2] & \ldots & p[n-1] & & p[n] \\
& A_1 & & A_2 & & \ldots & & A_n &
\end{array}
$$

cada A_i é $p[i-1] \times p[i]$ ($A_i[1\,..\,p[i-1], 1\,..\,p[i]]$)

Exemplo: $A_1 \cdot A_2 \cdot A_3$

$$
\begin{array}{ccccccc}
10 & A_1 & 100 & A_2 & 5 & A_3 & 50
\end{array}
$$

$((A_1\ A_2)\ A_3)$	7500	multiplicações escalares
$(A_1\ (A_2\ A_3))$	75000	multiplicações escalares

Soluções ótimas contêm soluções ótimas

Se

$$(A_1 A_2)\,(A_3((A_4 A_5) A_6))$$

é ordem ótima de multiplicação então

$$(A_1 A_2) \quad \text{e} \quad (A_3((A_4 A_5) A_6))$$

também são ordens ótimas.

Soluções ótimas contêm soluções ótimas

Se

$$(A_1 A_2)\,(A_3((A_4 A_5) A_6))$$

é ordem ótima de multiplicação então

$$(A_1 A_2) \quad \text{e} \quad (A_3((A_4 A_5) A_6))$$

também são ordens ótimas.

Decomposição: $(A_i \cdots A_k)\,(A_{k+1} \cdots A_j)$

Decomposição sugere um algoritmo recursivo.

Algoritmo recursivo

Recebe $p[i-1\,..\,j]$ e devolve o número mínimo de multiplicações escalares para calcular $A_i \cdots A_j$.

REC-MAT-CHAIN (p, i, j)

```
1   se i = j
2       então devolva 0
3   m ← ∞
4   para k ← i até j − 1 faça
5       q1 ← REC-MAT-CHAIN (p, i, k)
6       q2 ← REC-MAT-CHAIN (p, k + 1, j)
7       q ← q1 + p[i − 1]p[k]p[j] + q2
8       se q < m
9           então m ← q
10  devolva m
```

Consumo de tempo?

Consumo de tempo

A plataforma utilizada nos experimentos é um PC rodando Linux Debian ?.? com um processador Pentium II de 233 MHz e 128MB de memória RAM .

O programa foi compilado com o gcc versão ?? e opção de compilação "-O2".

n	3	6	10	20	25
tempo	$0.0s$	$0.0s$	$0.01s$	$201s$	$567m$

Consumo de tempo

$T(n) =$ número comparações entre q e m
na linha 8 quando $n := j - i + 1$

$$T(1) = 0$$

$$T(n) = \sum_{h=1}^{n-1} (T(h) + T(n-h) + 1)$$

$$= 2\sum_{h=2}^{n-1} T(h) + (n-1)$$

$$= 2(T(2) + \cdots + T(n-1)) + (n-1) \text{ para } n \geq 2$$

Consumo de tempo

$T(n) =$ número comparações entre q e m na linha 8 quando $n := j - i + 1$

$$
\begin{aligned}
T(1) &= 0 \\
T(n) &= \sum_{h=1}^{n-1} (T(h) + T(n-h) + 1) \\
&= 2\sum_{h=2}^{n-1} T(h) \;+\; (n-1) \\
&= 2(T(2) + \cdots + T(n-1)) \;+\; (n-1) \quad \text{para } n \geq 2
\end{aligned}
$$

Fácil verificar: $T(n) \geq 2^{n-2}$ para $n \geq 2$

Recorrência

n	1	2	3	4	5	6	7	8
$T(n)$	0	1	4	13	40	121	364	1093
2^{n-2}	0.5	1	2	8	16	32	64	128

Prova: Para $n = 2$, $T(2) = 1 = 2^{2-2}$.
Para $n \geq 3$,

$$\begin{aligned} T(n) &= 2(T(2) + \cdots + T(n-1)) + n - 1 \\ &\overset{\text{hi}}{\geq} 2(2^0 + \cdots + 2^{n-3}) + n - 1 \\ &> 2^0 + \cdots + 2^{n-3} + n - 1 \\ &= 2^{n-2} - 1 + n - 1 \\ &> 2^{n-2} \quad \text{(pois } n \geq 3\text{).} \end{aligned}$$

Conclusão

O consumo de tempo do algoritmo
REC-MAT-CHAIN é $\Omega(2^n)$.

Fórmula fechada

Temos que

$$T(n) = 2(T(2) + \cdots + T(n-2) + T(n-1)) + n - 1$$
$$T(n-1) = 2(T(2) + \cdots + T(n-2)) \qquad\qquad + n - 2$$

Logo,

$$T(n) - T(n-1) = 2T(n-1) + 1,$$

ou seja

$$T(n) = 3T(n-1) + 1\ .$$

Portanto,

$$T(n) = \frac{3^{n-1} - 1}{2}\ .$$

Nova conclusão

O consumo de tempo do algoritmo
REC-MAT-CHAIN é $\Omega(3^n)$.

Resolve subproblemas muitas vezes

$p[0] = 10 \quad p[1] = 100 \quad p[2] = 5 \quad p[3] = 50$

```
REC-MAT-CHAIN(p, 1, 3)
  REC-MAT-CHAIN(p, 1, 1)
  REC-MAT-CHAIN(p, 2, 3)
    REC-MAT-CHAIN(p, 2, 2)
    REC-MAT-CHAIN(p, 3, 3)
  REC-MAT-CHAIN(p, 1, 2)
    REC-MAT-CHAIN(p, 1, 1)
    REC-MAT-CHAIN(p, 2, 2)
  REC-MAT-CHAIN(p, 3, 3)
```

Número mínimo de mults = 7500

Resolve subproblemas muitas vezes

```
REC-MAT-CHAIN(p, 1, 5)
  REC-MAT-CHAIN(p, 1, 1)
  REC-MAT-CHAIN(p, 2, 5)
    REC-MAT-CHAIN(p, 2, 2)
    REC-MAT-CHAIN(p, 3, 5)
      REC-MAT-CHAIN(p, 3, 3)
      REC-MAT-CHAIN(p, 4, 5)
        REC-MAT-CHAIN(p, 4, 4)
        REC-MAT-CHAIN(p, 5, 5)
      REC-MAT-CHAIN(p, 3, 4)
        REC-MAT-CHAIN(p, 3, 3)
        REC-MAT-CHAIN(p, 4, 4)
      REC-MAT-CHAIN(p, 5, 5)
    REC-MAT-CHAIN(p, 2, 3)
      REC-MAT-CHAIN(p, 2, 2)
      REC-MAT-CHAIN(p, 3, 3)
    REC-MAT-CHAIN(p, 4, 5)
      REC-MAT-CHAIN(p, 4, 4)
      REC-MAT-CHAIN(p, 5, 5)
    REC-MAT-CHAIN(p, 2, 4)
      REC-MAT-CHAIN(p, 2, 2)
      REC-MAT-CHAIN(p, 3, 4)
        REC-MAT-CHAIN(p, 3, 3)
      REC-MAT-CHAIN(p, 4, 4)
    REC-MAT-CHAIN(p, 5, 5)
  REC-MAT-CHAIN(p, 1, 2)
    REC-MAT-CHAIN(p, 1, 1)
    REC-MAT-CHAIN(p, 2, 2)
  REC-MAT-CHAIN(p, 3, 5)
    REC-MAT-CHAIN(p, 3, 3)
    REC-MAT-CHAIN(p, 4, 5)
      REC-MAT-CHAIN(p, 4, 4)
      REC-MAT-CHAIN(p, 5, 5)
    REC-MAT-CHAIN(p, 3, 4)
      REC-MAT-CHAIN(p, 3, 3)
      REC-MAT-CHAIN(p, 4, 4)
    REC-MAT-CHAIN(p, 5, 5)
  REC-MAT-CHAIN(p, 1, 3)
    REC-MAT-CHAIN(p, 1, 1)
    REC-MAT-CHAIN(p, 2, 3)
      REC-MAT-CHAIN(p, 2, 2)
      REC-MAT-CHAIN(p, 3, 3)
    REC-MAT-CHAIN(p, 1, 2)
      REC-MAT-CHAIN(p, 1, 1)
      REC-MAT-CHAIN(p, 2, 2)
    REC-MAT-CHAIN(p, 3, 3)
REC-MAT-CHAIN(p, 1, 1)
REC-MAT-CHAIN(p, 2, 4)
  REC-MAT-CHAIN(p, 2,
  REC-MAT-CHAIN(p, 3,
    REC-MAT-CHAIN(p, 3
    REC-MAT-CHAIN(p, 4
  REC-MAT-CHAIN(p, 2,
    REC-MAT-CHAIN(p, 2
    REC-MAT-CHAIN(p, 3
  REC-MAT-CHAIN(p, 4,
REC-MAT-CHAIN(p, 1, 2)
  REC-MAT-CHAIN(p, 1,
  REC-MAT-CHAIN(p, 2,
REC-MAT-CHAIN(p, 3, 4)
  REC-MAT-CHAIN(p, 3,
  REC-MAT-CHAIN(p, 4,
REC-MAT-CHAIN(p, 1, 3)
  REC-MAT-CHAIN(p, 1,
  REC-MAT-CHAIN(p, 2,
    REC-MAT-CHAIN(p, 2
    REC-MAT-CHAIN(p, 3
  REC-MAT-CHAIN(p, 1,
    REC-MAT-CHAIN(p, 1
```

Programação dinâmica

$m[i,j] =$ número mínimo de multiplicações escalares para calcular $A_i \cdots A_j$

se $i = j$ então $m[i,j] = 0$

se $i < j$ então

$$m[i,j] = \min_{i \le k < j} \{ m[i,k] + p[i-1]p[k]p[j] + m[k{+}1,j] \}$$

Exemplo:

$$m[3,7] = \min_{3 \le k < 7} \{ m[3,k] + p[2]p[k]p[7] + m[k{+}1,7] \}$$

Programação dinâmica

Cada subproblema

$$A_i \cdots A_j$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela m?

Para calcular $m[2, 6]$ preciso de ...

Programação dinâmica

Cada subproblema

$$A_i \cdots A_j$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela m?

Para calcular $m[2,6]$ preciso de . . .

$m[2,2]$, $m[2,3]$, $m[2,4]$, $m[2,5]$ e de
$m[3,6]$, $m[4,6]$, $m[5,6]$, $m[6,6]$.

Programação dinâmica

Cada subproblema

$$A_i \cdots A_j$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela m?

Para calcular $m[2,6]$ preciso de ...

$m[2,2]$, $m[2,3]$, $m[2,4]$, $m[2,5]$ e de
$m[3,6]$, $m[4,6]$, $m[5,6]$, $m[6,6]$.

Calcule todos os $m[i,j]$ com $j-i+1=2$,
depois todos com $j-i+1=3$,
depois todos com $j-i+1=4$,
etc.

Programação dinâmica

	1	2	3	4	5	6	7	8	j
1	0								
2		0	$\star$	$\star$	$\star$	??			
3			0			$\star$			
4				0		$\star$			
5					0	$\star$			
6						0			
7							0		
8								0	
i									

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	??					
2		0					
3			0				
4				0			
5					0		
6						0	
i							

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0					
3			0				
4				0			
5					0		
6						0	

$$m[1,1] + p[1-1]p[1]p[2] + m[1+1,2] = 0 + 2000 + 0 = 2000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000				
2		0	??			
3			0			
4				0		
5					0	
6						0

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0	6000				
3			0				
4				0			
5					0		
6						0	

$$m[2,2] + p[2{-}1]p[2]p[3] + m[2{+}1,3]{=}0{+}6000{+}0{=}6000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000				
2		0	6000			
3			0	??		
4				0		
5					0	
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6
1	0	2000				
2		0	6000			
3			0	6000		
4				0		
5					0	
6						0

j

$$m[3,3] + p[3{-}1]p[3]p[4] + m[3{+}1,4]{=}0{+}6000{+}0{=}6000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000					
2		0	6000				
3			0	6000			
4				0	??		
5					0		
6						0	
i							

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000				
2		0	6000			
3			0	6000		
4				0	4500	
5					0	
6						0

$$m[4,4] + p[4{-}1]p[4]p[5] + m[4{+}1,5]{=}0{+}4500{+}0{=}4500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

$i \backslash j$	1	2	3	4	5	6
1	0	2000				
2		0	6000			
3			0	6000		
4				0	4500	
5					0	??
6						0

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000				
2		0	6000			
3			0	6000		
4				0	4500	
5					0	4500
6						0

$$m[5,5] + p[5{-}1]p[5]p[6] + m[5{+}1,6]{=}0{+}4500{+}0{=}4500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	??				
2		0	6000				
3			0	6000			
4				0	4500		
5					0	4500	
6						0	
i							

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000	9000			
2		0	6000			
3			0	6000		
4				0	4500	
5					0	4500
6						0

$$m[1,1] + p[1-1]p[1]p[3] + m[1+1,3]{=}0{+}3000{+}6000{=}9000$$

Simulação

$p[0]=10$ $p[1]=10$ $p[2]=20$ $p[3]=30$ $p[4]=10$ $p[5]=15$ $p[6]=30$

	1	2	3	4	5	6 j
1	0	2000	8000			
2		0	6000			
3			0	6000		
4				0	4500	
5					0	4500
6						0

$$m[1,2] + p[1-1]p[2]p[3] + m[2+1,3] = 2000 + 6000 + 0 = 8000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000			
2		0	6000	??		
3			0	6000		
4				0	4500	
5					0	4500
6						0

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000			
4				0	4500		
5					0	4500	
6						0	

$$m[2,2] + p[2{-}1]p[2]p[4] + m[2{+}1,4]{=}0{+}2000{+}6000{=}8000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000			
4				0	4500		
5					0	4500	
6						0	

$m[2,3] + p[2{-}1]p[3]p[4] + m[3{+}1,4]{=}6000{+}3000{+}0{=}9000$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	??		
4				0	4500		
5					0	4500	
6						0	
i							

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	13500		
4				0	4500		
5					0	4500	
6						0	

$$m[3,3] + p[3{-}1]p[3]p[5] + m[3{+}1,5] = 0 + 9000 + 4500 = 13500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	9000		
4				0	4500		
5					0	4500	
6						0	

$m[3,4] + p[3{-}1]p[4]p[5] + m[4{+}1,5]{=}6000{+}3000{+}0{=}9000$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000			
2		0	6000	8000		
3			0	6000	9000	
4				0	4500	??
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[4,4] + p[4-1]p[4]p[6] + m[4+1,6] = 0+9000+4500=13500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000				
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[4,5] + p[4{-}1]p[5]p[6] + m[5{+}1,6]{=}4500{+}13500{+}0{=}18000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	??		
2		0	6000	8000		
3			0	6000	9000	
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,1] + p[1{-}1]p[1]p[4] + m[1{+}1,4]{=}0{+}1000{+}8000{=}9000$$

Simulação

$p[0]=10 \quad p[1]=10 \quad p[2]=20 \quad p[3]=30 \quad p[4]=10 \quad p[5]=15 \quad p[6]=30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,2] + p[1-1]p[2]p[4] + m[2+1,4] = 2000+2000+6000 = 10000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000			
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,3] + p[1{-}1]p[3]p[4] + m[3{+}1,4]{=}8000{+}3000{+}0{=}11000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	9000		
2		0	6000	8000	??	
3			0	6000	9000	
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	12000		
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,2] + p[2-1]p[2]p[5] + m[2+1,5] = 0 + 3000 + 9000 = 12000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	12000		
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,3] + p[2{-}1]p[3]p[5] + m[3{+}1,5]{=}6000{+}4500{+}4500{=}15000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	9500		
3			0	6000	9000		
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,4] + p[2{-}1]p[4]p[5] + m[4{+}1,5]{=}8000{+}1500{+}0{=}9500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	9000		
2		0	6000	8000	9500	
3			0	6000	9000	??
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10\ \ p[1]{=}10\ \ p[2]{=}20\ \ p[3]{=}30\ \ p[4]{=}10\ \ p[5]{=}15\ \ p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	9500		
3			0	6000	9000	31500	
4				0	4500	13500	
5					0	4500	
6						0	

$m[3,3] + p[3{-}1]p[3]p[6] + m[3{+}1,6]{=}0{+}18000{+}13500{=}31500$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000	8000	9000		
2		0	6000	8000	9500	
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

$m[3,4] + p[3{-}1]p[4]p[6] + m[4{+}1,6] = 6000{+}6000{+}4500 = 16500$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000			
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[3,5] + p[3{-}1]p[5]p[6] + m[5{+}1,6]{=}9000{+}9000{+}0{=}18000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	9000	??	
2		0	6000	8000	9500	
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	11000		
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,1] + p[1{-}1]p[1]p[5] + m[1{+}1,5] = 0 + 1500 + 9500 = 11000$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	11000		
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,2] + p[1{-}1]p[2]p[5] + m[2{+}1,5]{=}2000{+}3000{+}9000{=}14000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6
1	0	2000	8000	9000	11000	
2		0	6000	8000	9500	
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

j

$$m[1,3] + p[1-1]p[3]p[5] + m[3+1,5] = 8000 + 4500 + 4500 = 17000$$

Simulação

$p[0]{=}10\quad p[1]{=}10\quad p[2]{=}20\quad p[3]{=}30\quad p[4]{=}10\quad p[5]{=}15\quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500		
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,4] + p[1{-}1]p[4]p[5] + m[4{+}1,5]{=}9000{+}1500{+}0{=}10500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	9000	10500	
2		0	6000	8000	9500	??
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500	22500	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,2] + p[2{-}1]p[2]p[6] + m[2{+}1,6]{=}0{+}6000{+}16500{=}22500$$

Simulação

$p[0]{=}10$ $p[1]{=}10$ $p[2]{=}20$ $p[3]{=}30$ $p[4]{=}10$ $p[5]{=}15$ $p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500	22500	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[2,3] + p[2{-}1]p[3]p[6] + m[3{+}1,6]{=}6000{+}9000{+}13500{=}28500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000	8000	9000	10500	
2		0	6000	8000	9500	15500
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

$$m[2,4] + p[2-1]p[4]p[6] + m[4+1,6] = 8000 + 3000 + 4500 = 15500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500		
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$m[2,5] + p[2{-}1]p[5]p[6] + m[5{+}1,6]{=}9500{+}4500{+}0{=}14000$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	9000	10500	??
2		0	6000	8000	9500	14000
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	17000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,1] + p[1{-}1]p[1]p[6] + m[1{+}1,6] = 0 + 3000 + 14000 = 17000$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6 j
1	0	2000	8000	9000	10500	17000
2		0	6000	8000	9500	14000
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

$$m[1,2] + p[1-1]p[2]p[6] + m[2+1,6] = 2000 + 6000 + 16500 = 24500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	17000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,3] + p[1{-}1]p[3]p[6] + m[3{+}1,6] {=} 8000{+}9000{+}13500{=}30500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	16500	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,4] + p[1{-}1]p[4]p[6] + m[4{+}1,6]{=}9000{+}3000{+}4500{=}16500$$

Simulação

$p[0]{=}10 \quad p[1]{=}10 \quad p[2]{=}20 \quad p[3]{=}30 \quad p[4]{=}10 \quad p[5]{=}15 \quad p[6]{=}30$

	1	2	3	4	5	6	j
1	0	2000	8000	9000	10500	15000	
2		0	6000	8000	9500	14000	
3			0	6000	9000	16500	
4				0	4500	13500	
5					0	4500	
6						0	

$$m[1,5] + p[1-1]p[5]p[6] + m[5+1,6] = 10500 + 4500 + 0 = 15000$$

Simulação

$p[0]=10$ $p[1]=10$ $p[2]=20$ $p[3]=30$ $p[4]=10$ $p[5]=15$ $p[6]=30$

i \ j	1	2	3	4	5	6
1	0	2000	8000	9000	10500	15000
2		0	6000	8000	9500	14000
3			0	6000	9000	16500
4				0	4500	13500
5					0	4500
6						0

Algoritmo de programação dinâmica

Recebe $p[0\,..\,n]$ e devolve $m[1,n]$.

MATRIX-CHAIN-ORDER (p,n)

```
 1   para i ← 1 até n faça
 2       m[i,i] ← 0
 3   para l ← 2 até n faça
 4       para i ← 1 até n − l + 1 faça
 5           j ← i + l − 1
 6           m[i,j] ← ∞
 7           para k ← i até j − 1 faça
 8               q ← m[i,k] + p[i − 1]p[k]p[j] + m[k+1,j]
 9               se q < m[i,j]
10                   então m[i,j] ← q
11   devolva m[1,n]
```

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: ???

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: $\mathrm{O}(n^3)$ (três loops encaixados)

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: $\mathrm{O}(n^3)$ (três loops encaixados)

Curioso verificar que consumo de tempo é $\Omega(n^3)$:
Número de execuções da linha 8:

Correção e consumo de tempo

Linhas 3–10: tratam das subcadeias $A_i \cdots A_j$ de comprimento l

Consumo de tempo: $\mathrm{O}(n^3)$ (três loops encaixados)

Curioso verificar que consumo de tempo é $\Omega(n^3)$:
Número de execuções da linha 8:

l	i	execs linha 8
2	$1,\ldots,n-1$	$(n-1)\cdot 1$
3	$1,\ldots,n-2$	$(n-2)\cdot 2$
4	$1,\ldots,n-3$	$(n-3)\cdot 3$
$\ldots$	$\ldots$	$\ldots$
$n-1$	$1,2$	$2\cdot(n-2)$
n	1	$1\cdot(n-1)$
	total	$\sum_{h=1}^{n-1} h(n-h)$

Consumo de tempo

Para $n \geq 6$, $\sum_{h=1}^{n-1} h(n-h) \;=$

$$= \; n\sum_{h=1}^{n-1} h \; - \; \sum_{h=1}^{n-1} h^2$$

$$= \; n\tfrac{1}{2}n(n-1) - \tfrac{1}{6}(n-1)n(2n-1) \quad \text{(CLRS p.1060)}$$

$$\geq \; \tfrac{1}{2}n^2(n-1) - \tfrac{1}{6}2n^3$$

$$\geq \; \tfrac{1}{2}n^2\tfrac{5n}{6} - \tfrac{1}{3}n^3$$

$$= \; \tfrac{5}{12}n^3 - \tfrac{1}{3}n^3$$

$$= \; \tfrac{1}{12}n^3$$

Consumo de tempo é $\Omega(n^3)$

Conclusão

O consumo de tempo do algoritmo
MATRIX-CHAIN-ORDER é $\Theta(n^3)$.

Versão recursiva eficiente

MEMOIZED-MATRIX-CHAIN-ORDER (p, n)

1 **para** $i \leftarrow 1$ **até** n **faça**
2   **para** $j \leftarrow 1$ **até** n **faça**
3         $m[i, j] \leftarrow \infty$
4 **devolva** LOOKUP-CHAIN $(p, 1, n)$

Versão recursiva eficiente

LOOKUP-CHAIN (p, i, j)

1 **se** $m[i, j] < \infty$
2   **então devolva** $m[i, j]$
3 **se** $i = j$
4   **então** $m[i, j] \leftarrow 0$
5   **senão para** $k \leftarrow i$ **até** $j - 1$ **faça**
6         $q_1 \leftarrow$ LOOKUP-CHAIN (p, i, k)
7         $q_2 \leftarrow$ LOOKUP-CHAIN $(p, k{+}1, j)$
8         $q \leftarrow q_1 + p[i{-}1]p[k]p[j] + q_2$
9   **se** $q < m[i, j]$
10   **então** $m[i, j] \leftarrow q$
11 **devolva** $m[1, n]$

Ingredientes de programação dinâmica

- Subestrutura ótima: soluções ótimas contém soluções ótimas de subproblemas.
- Subestrutura: decomponha o problema em subproblemas menores e, com sorte, mais simples.
- *Bottom-up*: combine as soluções dos problemas menores para obter soluções dos maiores.
- Tabela: armazene as soluções dos subproblemas em uma tabela, pois soluções dos subproblemas são consultadas várias vezes.
- Número de subproblemas: para a eficiência do algoritmo é importante que o número de subproblemas resolvidos seja 'pequeno'.
- Memoized: versão *top-down*, recursão com tabela.

Exercícios

Exercício 19.A [CLRS 15.2-1]
Encontre a maneira ótima de fazer a multiplicação iterada das matrizes cujas dimensões são $(5, 10, 3, 12, 5, 50, 6)$.

Exercício 19.B [CLRS 15.2-5]
Mostre que são necessários exatamente $n - 1$ pares de parênteses para especificar exatamente a ordem de multiplicação de $A_1 \cdot A_2 \cdots A_n$.

Exercício 19.C [CLRS 15.3-2]
Desenhe a árvore de recursão para o algoritmo MERGE-SORT aplicado a um vetor de 16 elementos. Por que a técnica de programação dinâmica não é capaz de acelerar o algoritmo?

Exercício 19.D [CLRS 15.3-5 expandido]
Considere o seguinte algoritmo para determinar a ordem de multiplicação de uma cadeia de matrizes $A_1, A_2, \ldots, A_n$ de dimensões $p_0, p_1, \ldots, p_n$: primeiro, escolha $k \in \{1, \ldots, n-1\}$ que minimize p_k; depois, determine recursivamente as ordens de multiplicação de $A_1, \ldots, A_k$ e $A_{k+1}, \ldots, A_n$. Esse algoritmo produz uma ordem que minimiza o número total de multiplicações escalares? E se $k \in \{1, \ldots, n-1\}$ for escolhido de modo a maximizar p_k?

Mais exercícios

Exercício 19.E
Prove que o número de execuções da linha 9 em MATRIX-CHAIN-ORDER é $\mathrm{O}(n^3)$.

Exercício 19.F [Subset-sum. CLRS 16.2-2 simplificado]
Escreva um algoritmo de programação dinâmica para o seguinte problema: dados números inteiros não-negativos $w_1, \ldots, w_n$ e W, encontrar um subconjunto K de $\{1, \ldots, n\}$ que satisfaça $\sum_{k \in K} w_k \leq W$ e maximize $\sum_{k \in K} w_k$. (Imagine que $w_1, \ldots, w_n$ são os tamanhos de arquivos digitais que você deseja armazenar em um disquete de capacidade W.)

Exercício 19.G [Mochila 0-1. CLRS 16.2-2]
O problema da mochila 0-1 consiste no seguinte: dados números inteiros não-negativos $v_1, \ldots, v_n, w_1, \ldots, w_n$ e W, queremos encontrar um subconjunto K de $\{1, \ldots, n\}$ que

satisfaça $\sum_{k \in K} w_k \leq W$ e maximize $\sum_{k \in K} v_k$.

(Imagine que w_i é o *peso* e v_i é o *valor* do objeto i.) Resolva o problema usando programação dinâmica.

Mais um exercício

Exercício 19.H [Partição equilibrada]
Seja S o conjunto das raízes raízes quadradas dos números $1, 2, \ldots, 500$. Escreva e teste um programa que determine uma partição (A, B) de S tal que a soma dos números em A seja tão próxima quanto possível da soma dos números em B. Seu algoritmo resolve o problema? ou só dá uma solução "aproximada"?
Uma vez calculados A e B, seu programa deve imprimir a diferença entre a soma de A e a soma de B e depois imprimir a lista dos quadrados dos números em um dos conjuntos.

AULA 16

Mais programação dinâmica

CLRS 15.4

= “recursão–com–tabela”

= transformação inteligente de recursão em iteração

Subseqüências

$\langle z_1, \ldots, z_k \rangle$ é subseqüência de $\langle x_1, \ldots, x_m \rangle$ se existem índices $i_1 < \cdots < i_k$ tais que

$$z_1 = x_{i_1} \quad \ldots \quad z_k = x_{i_k}$$

EXEMPLOS:

$\langle 5, 9, 2, 7 \rangle$ é subseqüência de $\langle 9, 5, 6, 9, 6, 2, 7, 3 \rangle$

⟨A,A, D, A, A⟩ é subseqüência de
⟨A, B, R, A, C, A, D, A, B, R, A⟩

```
A           A           D   A           A
|           |           |   |           |
A   B   R   A   C   A   D   A   B   R   A
```

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

SUB-SEQ- (Z, m, X, n)
1 $i \leftarrow m$
2 $j \leftarrow n$
3 **enquanto** $i \geq 1$ **e** $j \geq 1$ **faça**
4 **se** $Z[i] = X[j]$
5 **então** $i \leftarrow i - 1$
6 $j \leftarrow j - 1$
7 **se** $i \geq 1$
8 **então devolva** “`não é subseqüência`”
9 **senão devolva** “`é subseqüência`”

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

SUB-SEQ- (Z, m, X, n)
1 $\quad i \leftarrow m$
2 $\quad j \leftarrow n$
3 $\quad$ **enquanto** $i \geq 1$ **e** $j \geq 1$ **faça**
4 $\qquad$ **se** $Z[i] = X[j]$
5 $\qquad\quad$ **então** $i \leftarrow i - 1$
6 $\qquad j \leftarrow j - 1$
7 $\quad$ **se** $i \geq 1$
8 $\qquad$ **então devolva** "`não é subseqüência`"
9 $\qquad$ **senão devolva** "`é subseqüência`"

Consumo de tempo é $\mathrm{O}(m + n)$ e $\Omega(\min\{m, n\})$.

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

SUB-SEQ- (Z, m, X, n)
1 $\quad i \leftarrow m$
2 $\quad j \leftarrow n$
3 $\quad$ **enquanto** $i \geq 1$ **e** $j \geq 1$ **faça**
4 $\qquad$ **se** $Z[i] = X[j]$
5 $\qquad\quad$ **então** $i \leftarrow i - 1$
6 $\qquad j \leftarrow j - 1$
7 $\quad$ **se** $i \geq 1$
8 $\qquad$ **então devolva** "`não é subseqüência`"
9 $\qquad$ **senão devolva** "`é subseqüência`"

Invariantes:
(i0) $Z[i{+}1\,..\,m]$ é subseqüência de $X[j{+}1\,..\,n]$
(i1) $Z[i\,..\,m]$ **não** é subseqüência de $X[j{+}1\,..\,n]$

Subseqüência comum máxima

Z é subseq comum de X e Y

se Z é subseqüência comum de X e de Y

ssco $=$ subseqüência comum

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

Outra ssco = B D A B

Problema

Problema: Encontrar uma ssco máxima de X e Y.

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

ssco maximal = A B A

ssco máxima = B C A B

Outra ssco máxima = B D A B

LCS = Longest Common Subsequence

diff

```
> more abracadabra
A
B
R
A
C
A
D
A
B
R
A
```

```
> more yabbadabbadoo
Y
A
B
B
A
D
A
B
B
A
D
O
O
```

diff -u abracadabra yabbadabbadoo

```
+Y
 A
 B
-R
-A
-C
+B
 A
 D
 A
 B
-R
+B
 A
+D
+O
+O
```

Subestrutura ótima

Suponha que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

- Se $X[m] = Y[n]$, então $Z[k] = X[m] = Y[n]$ e $Z[1\,..\,k{-}1]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n{-}1]$.
- Se $X[m] \neq Y[n]$, então $Z[k] \neq X[m]$ implica que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n]$.
- Se $X[m] \neq Y[n]$, então $Z[k] \neq Y[n]$ implica que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n{-}1]$.

Algoritmo recursivo

Devolve o comprimento de uma ssco máxima de $X[1\,.\,.\,i]$ e $Y[1\,.\,.\,j]$.

REC-LCS-LENGTH (X, i, Y, j)

1. **se** $i = 0$ **ou** $j = 0$
2. **então devolva** 0
3. **se** $X[i] = Y[j]$
4. **então** $c \leftarrow$ REC-LCS-LENGTH $(X, i-1, Y, j-1) + 1$
5. **senão** $q_1 \leftarrow$ REC-LCS-LENGTH $(X, i-1, Y, j)$
6. $q_2 \leftarrow$ REC-LCS-LENGTH $(X, i, Y, j-1)$
7. **se** $q_1 \geq q2$
8. **então** $c \leftarrow q_1$
9. **senão** $c \leftarrow q_2$
10. **devolva** c

Consumo de tempo

$T(m, n) :=$ número máximo de comparações feitas por REC-LCS-LENGTH (X, m, Y, n)

Recorrência

$$T(0, n) = 0$$

$$T(m, 0) = 0$$

$$T(m, n) = T(m-1, n) + T(m, n-1) + 1 \quad \text{para } n \geq 0 \text{ e } m \geq 0$$

A que classe Ω pertence $T(m, n)$?

Recorrência

Note que $T(m, n) = T(n, m)$ para $n = 0, 1, \ldots$ e $m = 0, 1, \ldots$.

Seja $k := \min\{m, n\}$. Temos que

$$T(m, n) \geq T(k, k) \geq S(k),$$

onde

$$S(0) = 0$$
$$S(k) = 2S(k-1) + 1 \quad \text{para } k = 1, 2, \ldots$$

$S(k)$ é $\Theta(2^k) \Rightarrow T(m, n)$ é $\Omega(2^{\min\{m,n\}})$

$T(m, n)$ é exponencial

Conclusão

O consumo de tempo do algoritmo REC-LCS-LENGTH é $\Omega(2^{\min\{m,n\}})$.

Fórmula fechada

Prove que

$$T(m, n) = \binom{m+n}{m} - 1 \ .$$

Logo,

$$T(m, m) = \binom{2m}{m} - 1$$

$$> \frac{4^m}{2m+1} - 1.$$

Portanto, $T(m, m)$ é $\Omega(4^m/m)$.

Programação dinâmica

Problema: encontrar o comprimento de uma ssco máxima.

$c[i,j] =$ comprimento de uma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$

Recorrência:

$c[0,j] = c[i,0] = 0$

$c[i,j] = c[i-1,j-1] + 1$ se $X[i] = Y[j]$

$c[i,j] = \max\,(c[i,j-1], c[i-1,j])$ se $X[i] \neq Y[j]$

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,.\,.\,i] \quad \text{e} \quad Y[1\,.\,.\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[3,5]$ preciso de ...

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,.\,.\,i] \quad \text{e} \quad Y[1\,.\,.\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[3,5]$ preciso de ...
$c[3,4]$, $c[2,5]$ e de $c[2,4]$.

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,.\,.\,i] \quad \text{e} \quad Y[1\,.\,.\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[3,5]$ preciso de ...

$c[3,4]$, $c[2,5]$ e de $c[2,4]$.

Calcule todos os $c[i,j]$ com $i=1, j=0,1,\ldots,n$,
depois todos com $i=2, j=0,1,\ldots,n$,
depois todos com $i=3, j=0,1,\ldots,n$,
etc.

Programação dinâmica

	0	1	2	3	4	5	6	7	j
0	0	0	0	0	0	0	0	0	
1	0								
2	0				$\star$	$\star$			
3	0				$\star$	??			
4	0								
5	0								
6	0								
7	0								
i									

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	??						
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	??					
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	??				
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	??			
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	??		
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	??	
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	??						
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	??					
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

Y			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	??				
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	??			
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	??		
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	??	
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	??						
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	??					
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	??				
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	??			
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	??		
B	4	0							
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	??	
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	??						
D	5	0							
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A
X		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	??				
D	5	0						
A	6	0						
B	7	0						

j

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	??				
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	??			
D	5	0							
A	6	0							
B	7	0							

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	??		
D	5	0							
A	6	0							
B	7	0							

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	??	
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	??						
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	??					
A	6	0							
B	7	0							

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	??				
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	??			
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	??		
A	6	0							
B	7	0							
	i								

Simulação

X \ Y			B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	??	
A	6	0							
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	??						
B	7	0							

i

Simulação

Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	??				
B	7	0						

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	??				
B	7	0							
	i								

Simulação

X	Y		B	D	C	A	B	A
	i \ j	0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1	1
B	2	0	1	1	1	1	2	2
C	3	0	1	1	2	2	2	2
B	4	0	1	1	2	2	3	3
D	5	0	1	2	2	2	3	3
A	6	0	1	2	2	??		
B	7	0						

Simulação

Y			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	??		
B	7	0							
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	??	
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	??						

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	??					
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	??				

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	??			
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	??		
	i								

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	??	

i

Simulação

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	4	

i

Algoritmo de programação dinâmica

Devolve o comprimento de uma ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

LCS-LENGTH (X, m, Y, n)

1 **para** $i \leftarrow 0$ **até** m **faça**
2 $c[i, 0] \leftarrow 0$
3 **para** $j \leftarrow 1$ **até** n **faça**
4 $c[0, j] \leftarrow 0$
5 **para** $i \leftarrow 1$ **até** m **faça**
6 **para** $j \leftarrow 1$ **até** n **faça**
7   **se** $X[i] = Y[j]$
8   **então** $c[i, j] \leftarrow c[i-1, j-1] + 1$
9   **senão se** $c[i-1, j] \geq c[i, j-1]$
10   **então** $c[i, j] \leftarrow c[i-1, j]$
11   **senão** $c[i, j] \leftarrow c[i, j-1]$
12 **devolva** $c[m, n]$

Conclusão

O consumo de tempo do algoritmo LCS-LENGTH é $\Theta(mn)$.

Subseqüência comum máxima

X	Y		B	D	C	A	B	A	
		0	1	2	3	4	5	6	j
	0	⋆	⋆	⋆	⋆	⋆	⋆	⋆	
A	1	⋆	↑	↑	↑	↖	←	↖	
B	2	⋆	↖	←	←	↑	↖	←	
C	3	⋆	↑	↑	↖	←	↑	↑	
B	4	⋆	↖	↑	↑	↑	↖	←	
D	5	⋆	↑	↖	↑	↑	↑	↑	
A	6	⋆	↑	↑	↑	↖	↑	↖	
B	7	⋆	↖	↑	↑	↑	↖	↑	

i

Algoritmo de programação dinâmica

```
LCS-LENGTH (X, m, Y, n)
 1   para i ← 0 até m faça
 2       c[i, 0] ← 0
 3   para j ← 1 até n faça
 4       c[0, j] ← 0
 5   para i ← 1 até m faça
 6       para j ← 1 até n faça
 7           se X[i] = Y[j]
 8               então c[i, j] ← c[i − 1, j − 1] + 1
 8                     b[i, j] ← "↖"
 9               senão se c[i − 1, j] ≥ c[i, j − 1]
10                   então c[i, j] ← c[i − 1, j]
10                         b[i, j] ← "↑"
11                   senão c[i, j] ← c[i, j − 1]
11                         b[i, j] ← "←"
12   devolva c e b
```

Get-LCS

GET-LCS $(X, m, n, b,$ máxcomp$)$

1 $\quad k \leftarrow$ máxcomp
2 $\quad i \leftarrow m$
2 $\quad j \leftarrow n$
3 $\quad$ **enquanto** $i > 0$ **e** $j > 0$ **faça**
4 $\qquad$ **se** $b[i, j] =$ "$\nwarrow$"
5 $\qquad\quad$ **então** $Z[k] \leftarrow X[i]$
6 $\qquad\qquad k \leftarrow k - 1 \quad i \leftarrow i - 1 \quad j \leftarrow j - 1$
9 $\qquad\quad$ **senão se** $b[i, j] =$ "$\leftarrow$"
10 $\qquad\qquad$ **então** $j \leftarrow j - 1$
11 $\qquad\qquad$ **senão** $i \leftarrow i - 1$
12 $\quad$ **devolva** Z

Consumo de tempo é $\mathrm{O}(m + n)$ e $\Omega(\min\{m, n\})$.

Versão recursiva eficiente

MEMOIZED-LCS-LENGTH (X, m, Y, n)

1 **para** $i \leftarrow 0$ **até** m **faça**

2 **para** $j \leftarrow 1$ **até** n **faça**

3 $c[i, j] \leftarrow \infty$

4 **devolva** LOOKUP-LCS (c, m, n)

Versão recursiva eficiente

LOOKUP-LCS (c, i, j)

```
1   se c[i, j] < ∞
2       então devolva c[i, j]
3   se i = 0 ou j = 0 então c[i, j] ← 0
4   senão se X[i] = Y[j]
5           então c[i, j] ←LOOKUP-LCS (c, i−1, j−1)
                    +1
6           senão q1 ← LOOKUP-LCS (c, i−1, j)
7                 q2 ← LOOKUP-LCS (c, i, j−1)
8                 se q1 ≥ q2
9                     então c[i, j] ← q1
10                    senão c[i, j] ← q2
11  devolva c[i, j]
```

Exercícios

Exercício 20.A
Escreva um algoritmo para decidir se $\langle z_1, \ldots, z_k \rangle$ é subseqüência de $\langle x_1, \ldots, x_m \rangle$. Prove rigorosamente que o seu algoritmo está correto.

Exercício 20.B
Suponha que os elementos de uma seqüência $\langle a_1, \ldots, a_n \rangle$ são distintos dois a dois. Quantas subseqüências tem a seqüência?

Exercício 20.C
Uma subseqüência crescente Z de uma seqüência X e é *máxima* se não existe outra subseqüência crescente mais longa. A subseqüência $\langle 5, 6, 9 \rangle$ de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$ é máxima? Dê uma seqüência crescente máxima de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$. Mostre que o algoritmo "guloso" óbvio não é capaz, em geral, de encontrar uma subseqüência crescente máxima de uma seqüência dada. (Algoritmo guloso óbvio: escolha o menor elemento de X; a partir daí, escolha sempre o próximo elemento de X que seja maior ao último escolhido.)

Exercício 20.D
Escreva um algoritmo de programação dinâmica para resolver o problema da subseqüência crescente máxima.

Mais exercícios

Exercício 20.E [CLRS 15.4-5]
Mostre como o algoritmo da subseqüência comum máxima pode ser usado para resolver o problema da subseqüência crescente máxima de uma seqüência numérica. Dê uma delimitação justa, em notação Θ, do consumo de tempo de sua solução.

Exercício 20.F [Printing neatly. CLRS 15-2]
Considere a seqüência $P_1, P_2, \ldots, P_n$ de palavras que constitui um parágrafo de texto. A palavra P_i tem l_i caracteres. Queremos imprimir as palavras em linhas, na ordem dada, de modo que cada linha tenha no máximo M caracteres, $|P_i| <= M$, $i = 1, \ldots, n$. Se uma determinada linha contém as palavras $P_i, P_{i+1}, \ldots, P_j$ (com $i \leq j$) e há exatamente um espaço entre cada par de palavras consecutivas, o número de espaços no fim da linha é

$$M - (l_i + 1 + l_{i+1} + 1 + \cdots + 1 + l_j)\,.$$

É claro que não devemos permitir que esse número seja negativo. Queremos minimizar, com relação a todas as linhas exceto a última, a soma dos cubos dos números de espaços no fim de cada linha. (Assim, se temos linhas $1, 2, \ldots, L$ e b_p espaços no fim da linha p, queremos minimizar $b_1^3 + b_2^3 + \cdots + b_{L-1}^3$).
Dê um exemplo para mostrar que algoritmos inocentes não resolvem o problema. Dê um algoritmo de programação dinâmica que resolva o problema. Qual a “optimal substructure property” para esse problema? Faça uma análise do consumo de tempo do algoritmo.

AULA 17

Mochila

Dados dois vetores $w[1\,..\,n]$ e $x[1\,..\,n]$ e , denotamos por $w \cdot x$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores de inteiros não-negativos $w[1\,..\,n]$ e $v[1\,..\,n]$.

Uma mochila é qualquer vetor $x[1\,..\,n]$ tal que

$$w \cdot x \leq W \quad \text{e} \quad 0 \leq x[i] \leq 1 \text{ para todo } i$$

O valor de uma mochila é o número $v \cdot x$.

Uma mochila é ótima se tem valor máximo.

Problema booleano da mochila

Um mochila $x[1\,.\,.\,n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita booleana.

Problema (Knapsack Problem): Dados (w, v, n, W), encontrar uma mochila boolena ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila boolena ótima para o problema (w, v, n, W).

Se $x[n] = 1$

então $x[1\,..\,n{-}1]$ é mochila boolena ótima para
$$(w, v, n-1, W - w[n])$$

senão $x[1\,..\,n]$ é mochila boolena ótima para
$$(w, v, n-1, W)$$

NOTA. Não há nada de especial acerca do índice n. Uma afirmação semelhante vale para qualquer índice i.

Solução recursiva

Devolve o valor de uma mochila ótima para (w, v, n, W).

REC-MOCHILA (w, v, i, Y)
1 **se** $i = 0$ **ou** $Y = 0$
2 **então devolva** 0
3 **se** $w[i] > Y$
4 **então devolva** REC-MOCHILA $(w, v, i-1, Y)$
5 $a \leftarrow$ REC-MOCHILA $(w, v, i-1, Y)$
6 $b \leftarrow$ REC-MOCHILA $(w, v, i-1, Y-w[n]) + v[n]$
7 **devolva** $\max\{a, b\}$

Consumo de tempo no pior caso é $\Omega(2^n)$

Por que demora tanto?
O mesmo subproblema é resolvido muitas vezes.

Exemplo

Suponha $w[i] = 1$ para todo i e $W \geq n$.

$T(i, Y) =$ número de execuções da linha 7 na chamada para (w, v, i, Y)

$T(0, Y) = 0$ para todo Y

$T(i, 0) = 0$ para todo i

$T(i, Y) = T(i-1, Y) + T(i-1, Y-1) + 1$ se $i > 0$ e $Y > 0$

Verifique que $T(n, W) = 2^n - 1$ para $W \geq n$.

$T(i, Y)$

i \ Y	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1
2	0	2	3	3	3	3	3	3
3	0	3	6	7	7	7	7	7
4	0	4	10	14	15	15	15	15
5	0	5	15	25	30	31	31	31
6	0	6	21	40	46	62	63	63
7	0	7	28	62	87	109	126	127

Programação dinâmica

Problema: encontrar o valor de uma mochila booleana ótima.

$t[i, Y] =$ valor de uma mochila booleana ótima para (w, v, i, Y)

$=$ valor da expressão $v \cdot x$ sujeito à restrição

$$w \cdot x \leq Y,$$

onde x é uma mochila booleana ótima

Possíveis valores de Y: $0, 1, 2, \ldots, W$

Recorrência

$t[i, Y]$ = valor máximo da expressão $v \cdot x$ sujeito à restrição

$$w \cdot x \leq Y$$

onde x é um vetor booleano

$t[0, Y] = 0$ para todo Y

$t[i, 0] = 0$ para todo i

$t[i, Y] = t[i-1, Y]$ se $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\}$ se $w[i] \leq Y$

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Para calcular $t[4, 6]$ preciso de ...

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Para calcular $t[4, 6]$ preciso de ...

$t[4-1, 6]$, $t[4-1, 5]$, $t[4-1, 4]$
$t[4-1, 3]$, $t[4-1, 2]$, $t[4-1, 1]$ e de $t[4-1, 0]$.

Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela t?

Para calcular $t[4,6]$ preciso de ...

$t[4{-}1,6]$, $t[4{-}1,5]$, $t[4{-}1,4]$
$t[4{-}1,3]$, $t[4{-}1,2]$, $t[4{-}1,1]$ e de $t[4{-}1,0]$.

Calcule todos os $t[i,Y]$ com $Y = 1, i = 0,1,\ldots,n$,
depois todos com $Y = 2, i = 0,1,\ldots,n$,
depois todos com $Y = 3, i = 0,1,\ldots,n$,
etc.

Programação dinâmica

	1	2	3	4	5	6	7	8	Y
1	0	0	0	0	0	0	0	0	
2	0								
3	0	$\star$	$\star$	$\star$	$\star$	$\star$			
4	0					??			
5	0								
6	0								
7	0								
8	0								
i									

Exemplo

$W = 5$ e $n = 4$

	1	2	3	4
w	4	2	1	3
v	500	400	300	450

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	400	700	800
4	0	300	400	450	710	850

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

MOCHILA-BOOLEANA (w, v, n, W)
1 **para** $Y \leftarrow 0$ **até** W **faça**
2 $\quad t[0, Y] \leftarrow 0$
3 $\quad$ **para** $i \leftarrow 1$ **até** n **faça**
4 $\quad\quad a \leftarrow t[i{-}1, Y]$
5 $\quad\quad$ **se** $w[i] > Y$
6 $\quad\quad\quad$ **então** $b \leftarrow 0$
7 $\quad\quad\quad$ **senão** $b \leftarrow t[i{-}1, Y{-}w[i]] + v[i]$
8 $\quad\quad t[i, Y] \leftarrow \max\{a, b\}$
9 **devolva** $t[n, W]$

Consumo de tempo é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo
MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Se W é $\Omega(2^n)$ o consumo de tempo é $\Omega(n2^n)$,
tão lento quanto o algoritmo força bruta!

Obtenção da mochila

MOCHILA (w, n, W, t)

```
1    Y ← W
2    para i ← n decrescendo até 1 faça
3         se t[i, Y] = t[i−1, Y]
4              então x[i] ← 0
5              senão x[i] ← 1
6                        Y ← Y − w[i]
7    devolva x
```

Consumo de tempo é $\Theta(n)$.

Versão recursiva

MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)

1 **para** $i \leftarrow 0$ **até** n **faça**

2 **para** $Y \leftarrow 0$ **até** W **faça**

3 $t[i, Y] \leftarrow \infty$

4 **devolva** LOOKUP-MOC (w, v, n, W)

Versão recursiva

LOOKUP-MOC (w, v, i, Y)

```
1  se t[i, Y] < ∞
2      então devolva t[i, Y]
3  se i = 0 ou Y = 0 então t[i, Y] ← 0
   senão
4      se w[i] > Y
           então
5              t[i, Y] ← LOOKUP-MOC (w, v, i−1, Y)
           senão
6              a ← LOOKUP-MOC (w, v, i−1, Y)
7              b ← LOOKUP-MOC (w, v, i−1, Y−w[i]) + v[i]
8              t[i, Y] ← max {a, b}
9  devolva t[i, Y]
```

Algoritmos gulosos (*greedy*)

CLRS 16.1–16.3

Algoritmos gulosos

"A *greedy algorithm* starts with a solution to a very small subproblem and augments it successively to a solution for the big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, without regard to whether it will lead to a good long-term or global solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

"One thing you will notice about greedy algorithms is that they are usually easy to design, easy to implement, easy to analyse, and they are very fast, but they are *almost always difficult to prove correct*."

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Problema fracionário da mochila

Problema: Dados (w, v, n, W), encontrar uma mochila ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$
x	1	$1/3$	0	0	valor $= 1040$

A propósito ...

O problema fracionário da mochila é um problema de programação linear (PL): encontrar um vetor x que

$$
\begin{array}{rrll}
\text{maximize} & v \cdot x & & \\
\text{sob as restrições} & w \cdot x & \leq W & \\
 & x[i] & \geq 0 & \text{para } i = 1, \ldots, n \\
 & x[i] & \leq 1 & \text{para } i = 1, \ldots, n
\end{array}
$$

PL's podem ser resolvidos por

SIMPLEX: no pior caso consome tempo exponencial
na prática é muito rápido

ELIPSÓIDES: consome tempo polinomial
na prática é lento

PONTOS-INTERIORES: consome tempo polinomial
na prática é rápido

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila ótima para o problema (w, v, n, W).

Se $x[n] = \delta$

então $x[1\,..\,n{-}1]$ é mochila ótima para

$$(w, v, n-1, W - \delta\, w[n])$$

NOTA. Não há nada de especial acerca do índice n. Uma afirmação semelhante vale para qualquer índice i.

Escolha gulosa

Suponha $w[i] \neq 0$ para todo i.

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i

então EXISTE uma mochila ótima $x[1\,..\,n]$ tal que

$$x[n] = \min\left\{1, \frac{W}{w[n]}\right\}$$

Algoritmo guloso

Esta propriedade da escolha gulosa sugere um algoritmo que atribui os valores de $x[1\,..\,n]$ supondo que os dados estejam em ordem descrescente de "valor especifico" :

$$\frac{v[1]}{w[1]} \leq \frac{v[2]}{w[2]} \leq \cdots \leq \frac{v[n]}{w[n]}$$

É nessa ordem "mágica" que está o segredo do funcionamento do algoritmo.

Algoritmo guloso

Devolve uma mochila ótima para (w, v, n, W).

MOCHILA-FRACIONÁRIA (w, v, n, W)

0 ordene w e v de tal forma que
$$v[1]/w[1] \leq v[2]/w[2] \leq \cdots \leq v[n]/w[n]$$
1 **para** $i \leftarrow n$ **decrescendo até** 1 **faça**
2 **se** $w[i] \leq W$
3 **então** $x[i] \leftarrow 1$
4 $W \leftarrow W - w[i]$
5 **senão** $x[i] \leftarrow W/w[i]$
6 $W \leftarrow 0$
7 **devolva** x

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.
Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Correção

No início de cada execução da linha 1 vale que

(i0) $x' = x[i{+}1 \,..\, n]$ é mochila ótima para

$$(w', v', n', W)$$

onde

$$w' = w[i{+}1 \,..\, n]$$
$$v' = v[i{+}1 \,..\, n]$$
$$n' = n - i$$

Na última iteração $i = 0$ e portanto $x[1 \,..\, n]$ é mochila ótima para (w, v, n, W).

Conclusão

O consumo de tempo do algoritmo MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$.

Escolha gulosa

Precisamos mostrar que se $x[1 \mathinner{.\,.} n]$ é uma mochila ótima, então podemos supor que

$$x[n] = \alpha := \min\left\{1, \frac{W}{w[n]}\right\}$$

Depois de mostrar isto, indução faz o resto do serviço.

Técnica: transformar um solução ótima em uma solução ótima 'gulosa'.

Esta transformação é semelhante ao processo de pivotacão feita pelo algoritmo SIMPLEX para programação linear.

Escolha gulosa

Seja $x[1\,..\,n]$ uma mochila ótima para (w, v, n, W) tal que $x[n]$ é máximo. Se $x[n] = \alpha$, não há o que mostrar.

Suponha $x[n] < \alpha$.

Seja i em $[1\,..\,n{-}1]$ tal que $x[i] > 0$.
(Quem garante que existe um tal i ?).

Seja

$$\delta := \min\left\{ x[i], (\alpha - x[n])\frac{w[n]}{w[i]} \right\}$$

e

$$\beta := \delta\,\frac{w[i]}{w[n]}\,.$$

Note que $\delta > 0$ e $\beta > 0$.

Mais escolha gulosa

Seja $x'[1\,..\,n]$ tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Verifique que $0 \leq x[j] \leq 1$ para todo j.
Além disso, temos que

$$\begin{aligned} w \cdot x' &= w[1]x'[1] + \cdots + w[i]x'[i] + \cdots + w[n]x'[n] \\ &= w[1]x[1] + \cdots + w[i](x[i] - \delta) + \cdots + w[n](x[n] + \beta) \\ &= w[1]x[1] + \cdots + w[i](x[i] - \delta) + \cdots + w[n]\left(x[n] + \delta\,\frac{w[i]}{w[n]}\right) \\ &= w[1]x[1] + \cdots + w[i]x[i] - \delta w[i] + \cdots + w[n]x[n] + \delta w[i] \\ &\leq W\,. \end{aligned}$$

Mais escolha gulosa ainda

Temos ainda que

$$
\begin{aligned}
v \cdot x' &= v[1]x'[1] + \cdots + v[i]x'[i] + \cdots + v[n]x'[n] \\
&= v[1]x[1] + \cdots + v[i](x[i] - \delta) + \cdots + v[n](x[n] + \beta) \\
&= v[1]x[1] + \cdots + v[i](x[i] - \delta) + \cdots + v[n]\left(x[n] + \delta\,\frac{w[i]}{w[n]}\right) \\
&= v[1]x[1] + \cdots + v[i]x[i] - \delta v[i] + \cdots + v[n]x[n] + \delta w[i]\frac{v[n]}{w[n]} \\
&= x \cdot v + \delta\left(w[i]\frac{v[n]}{w[n]} - v[i]\right) \\
&\geq x \cdot v + \delta(w[i]\frac{v[i]}{w[i]} - v[i]) \quad (\text{devido a escolha gulosa!}) \\
&= x \cdot v.
\end{aligned}
$$

Escolha gulosa: epílogo

Assim, x' é uma mochila tal que $x' \cdot v \geq x \cdot v$.
Como x é mochila ótima, concluímos que $x' \cdot v = x \cdot v$ e que x' é uma mochila ótima que contradiz a nossa escolha de x, já que

$$x'[n] = x[n] + \beta > x[n].$$

Conclusão

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i
e x é uma mochila ótima para (w, v, n, W) com $x[n]$
máximo, então

$$x[n] = \min\left\{1, \frac{W}{w[n]}\right\}$$

Máximo e maximal

$\mathcal{S}$ = coleção de subconjuntos de $\{1, \ldots, n\}$

Elemento X de $\mathcal{S}$ é máximo se
não existe Y em $\mathcal{S}$ tal que $|Y| > |X|$.

Elemento X de $\mathcal{S}$ é maximal se
não existe Y em $\mathcal{S}$ tal que $Y \supset X$.

Exemplo:

$\{\,\{1,2\},\{2,3\},\{4,5\},\{1,2,3\},\{2,3,4,5\},\{1,3,4,5\}\,\}$

$\{1,2\}$ não é maximal

$\{1,2,3\}$ é maximal

$\{2,3,4,5\}$ é maximal e máximo

Problemas

Problema 1: Encontrar elemento máximo de $\mathcal{S}$.
Usualmente difícil.

Problema 2: Encontrar elemento maximal de $\mathcal{S}$.
Muito fácil: aplique algoritmo "guloso".

$\mathcal{S}$ "tem estrutura gulosa" se todo maximal é máximo.
Se $\mathcal{S}$ tem estrutura gulosa, Problema 1 é fácil.

Algoritmos gulosos

Algoritmo guloso

- procura maximal e acaba obtendo máximo
- procura ótimo local e acaba obtendo ótimo global

costuma ser

- muito simples e intuitivo
- muito eficiente
- difícil provar que está correto

Problema precisa ter

- subestrutura ótima (como na programação dinâmica)
- propriedade da escolha gulosa (*greedy-choice property*)

Exercícios

Exercício 21.A
O problema da soma de subconjunto do exercício 19.F pode ser resolvido por um algoritmo guloso? O problema tem a propridade da escolha gulosa?

Exercício 21.B
O problema da mochila booleana pode ser resolvido por um algoritmo guloso?

Exercício 21.C [Mochila fracionária. CLRS 16.2-1]
O problema da mochila fracionária consiste no seguinte: dados números inteiros não-negativos $v_1, \ldots, v_n$ e $w_1, \ldots, w_n, W$, encontrar racionais $x_1, \ldots, x_n$ no intervalo $[0, 1]$ tais que

$$x_1 w_1 + \cdots + x_n w_n \leq W \text{ e } x_1 v_1 + \cdots + x_n v_n \text{ é máxima.}$$

(Imagine que w_i é o *peso* e v_i é o *valor* do objeto i.) Escreva um algorito guloso para resolver o problema.

Para provar que o seu algoritmo está correto, verifique as seguintes propriedades.
Propriedades da subestrutura ótima: se $x_1, \ldots, x_n$ é solução ótima do problema (w, v, n, W) então $x_1, \ldots, x_{n-1}$ é solução ótima do problema $(w, v, n-1, W - x_n w_n)$.
Propriedade da escolha gulosa: se $v_n/w_n \geq v_i/w_i$ para todo i então existe uma solução ótima x tal que $x_n = \min(1, W/w_n)$.

AULA 18

Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma coleção máxima de intervalos disjuntos dois a dois.

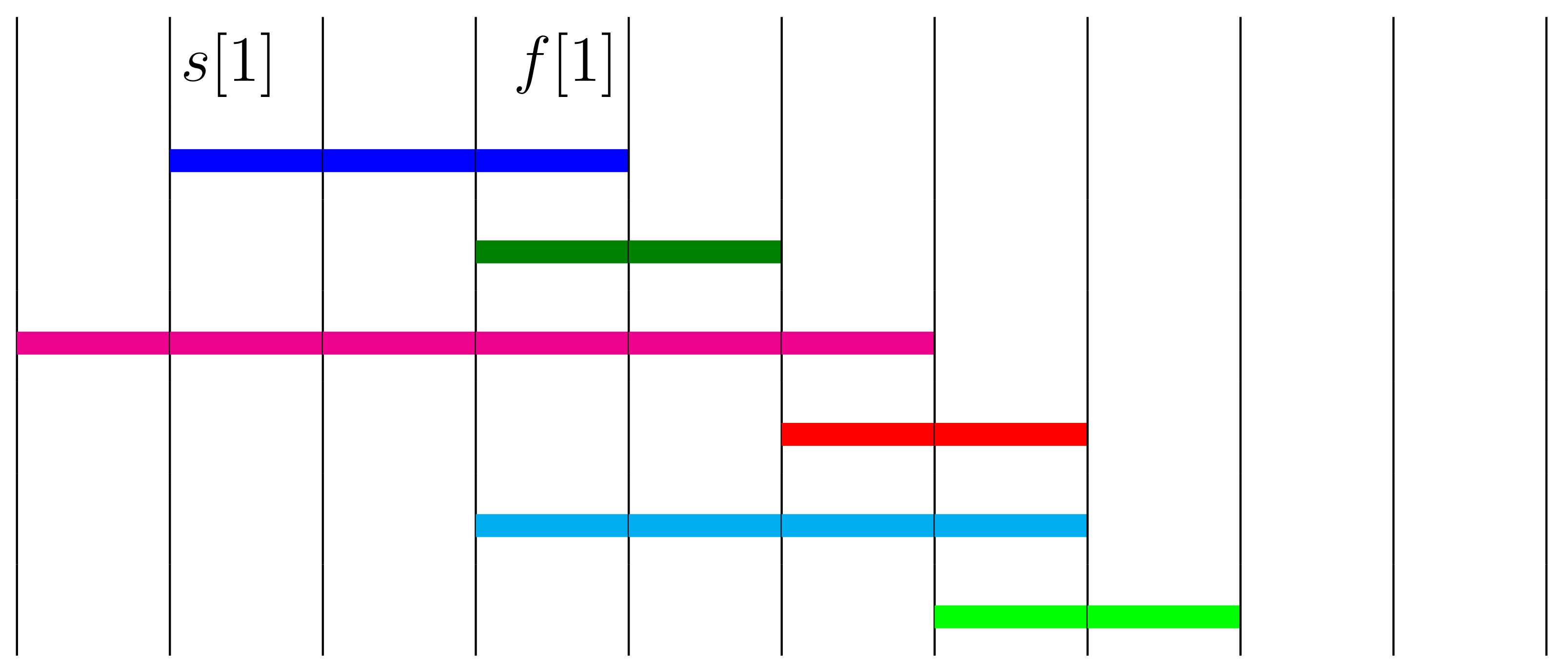
Solução é um subconjunto A de $\{1, \ldots, n\}$.

Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma coleção máxima de intervalos disjuntos dois a dois.

Solução é um subconjunto A de $\{1, \ldots, n\}$.

Exemplo:

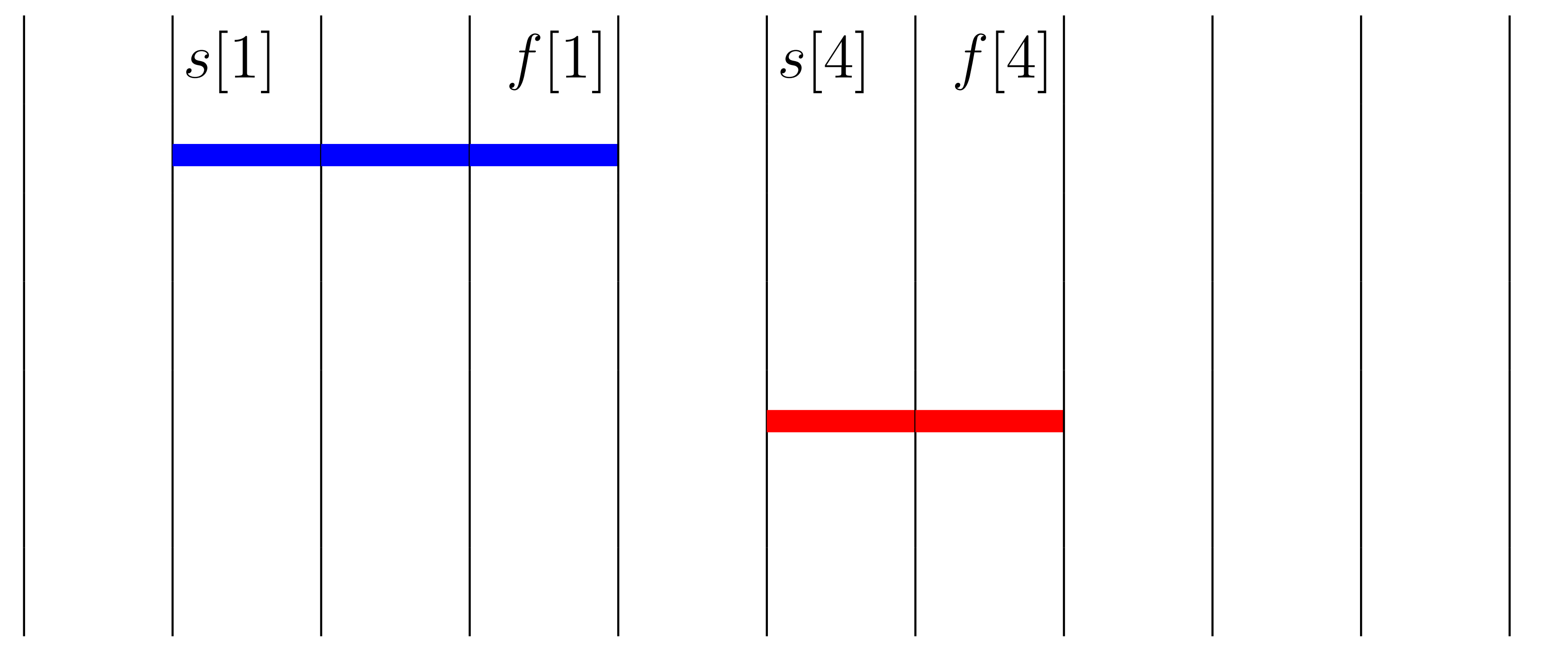


Problema dos intervalos disjuntos

Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma coleção máxima de intervalos disjuntos dois a dois.

Solução é um subconjunto A de $\{1, \ldots, n\}$.

Solução:

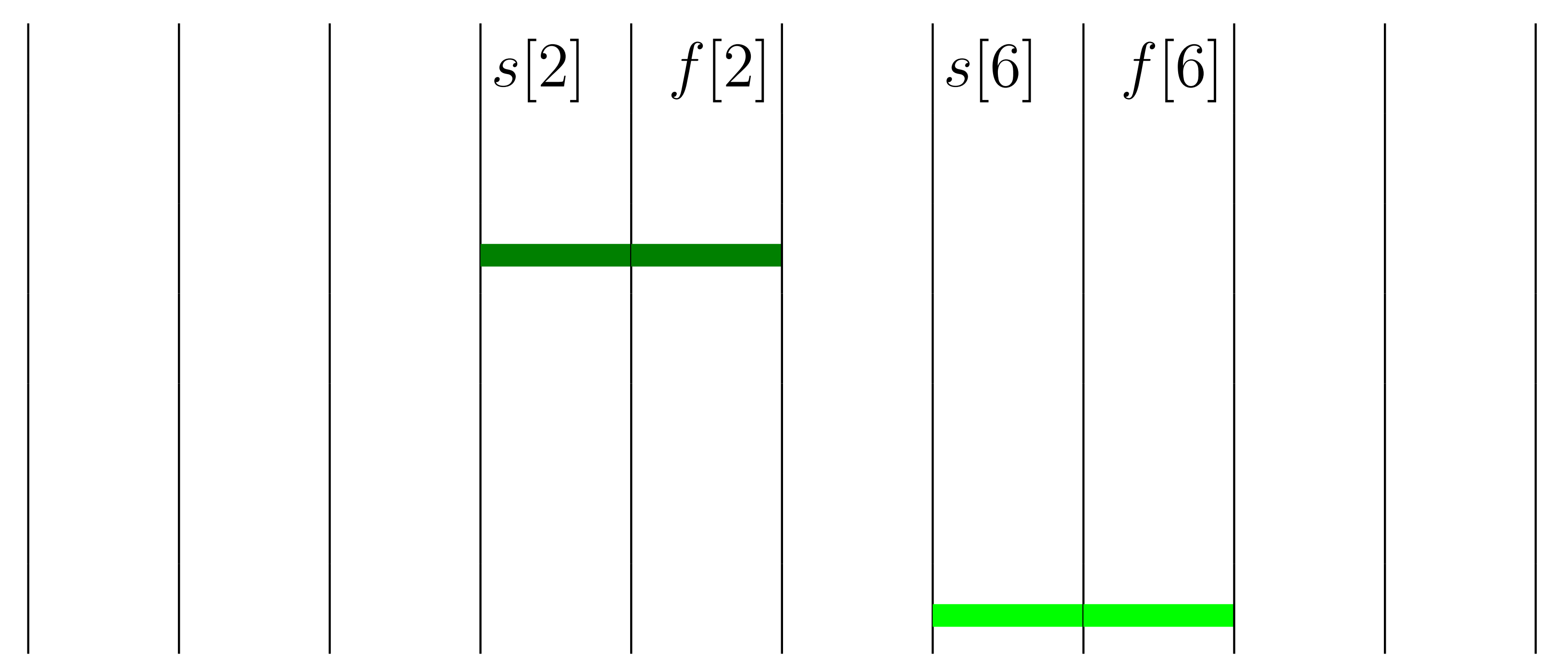


Problema dos intervalos disjuntos

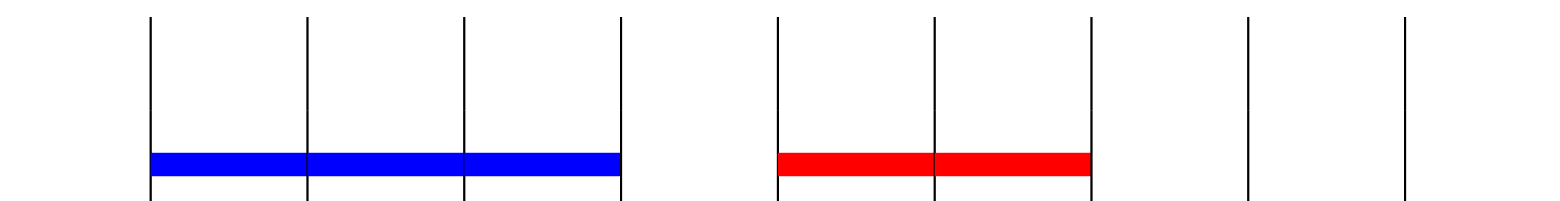
Problema: Dados intervalos $[s[1], f[1]), \ldots, [s[n], f[n])$, encontrar uma coleção máxima de intervalos disjuntos dois a dois.

Solução é um subconjunto A de $\{1, \ldots, n\}$.

Solução:



Motivação



Se cada intervalo é uma “atividade”, queremos coleção disjunta máxima de atividades compatíveis
(i e j são compatíveis se $f[i] \leq s[j]$)

Nome no CLRS: Activity Selection Problem

Subestrutura ótima

Intervalos $S := \{1, \ldots, n\}$

Suponha que A é coleção máxima de intervalos de S disjuntos dois a dois.

Se $i \in A$

então $A - \{i\}$ é coleção máxima de intervalos disjuntos de $S - \{k : [s[k], f[k]) \cap [s[i], f[i]) \neq \emptyset\}$.

senão A é coleção máxima de intervalos disjuntos de $S - \{i\}$.

Demonstre a propriedade.

Subestrutura ótima II

Intervalos $S := \{1, \ldots, n\}$

Suponha que A é coleção máxima de intervalos de S disjuntos dois a dois.

Se $i \in A$ é tal que $f[i]$ é mínimo

então $A - \{i\}$ é coleção máxima de intervalos disjuntos de $\{k : s[k] \geq f[i]\}$.

$\{k : s[k] \geq f[i]\} =$ todos intervalos "à direita" de "i".

Demonstre a propriedade.

Algoritmo de programação dinâmica

Suponha $f[1] \leq f[2] \leq \cdots \leq f[n]$

$t[i] =$ tamanho de uma subcoleção disjunta máxima de $\{i, \ldots, n\}$

$$t[n] = 1$$

$$t[i] = \max\{t[i+1], 1 + t[k]\} \quad \text{para } i = 1, \ldots, n-1,$$

onde k é o menor índice tal que $s[k] \geq f[i]$.

Algoritmo de programação dinâmica

DYNAMIC-ACTIVITY-SELECTOR (s, f, n)

0 ordene s e f de tal forma que

$$f[1] \leq f[2] \leq \cdots \leq f[n]$$

1 $A[n+1] \leftarrow \emptyset$

2 **para** $i \leftarrow n$ **decrescendo até** 1 **faça**

3 $A[i] \leftarrow A[i+1]$

4 $k \leftarrow i+1$

5 **enquanto** $k \leq n$ e $s[k] < f[i]$ **faça**

6 $k \leftarrow k+1$

7 **se** $|A[i]| < 1 + |A[k]|$

8 **então** $A[i] \leftarrow \{i\} \cup A[k]$

9 **devolva** $A[1]$

Consumo de tempo é $\Theta(n^2)$.

Conclusão

Invariante: na linha 2 vale que

(i0) $A[j]$ é coleção disjunta máxima de $\{j, \ldots, n\}$ para $j = i + 1, \ldots, n$.

O consumo de tempo do algoritmo DYNAMIC-ACTIVITY-SELECTOR é $\Theta(n^2)$.

Escolha gulosa

Intervalos $S := \{1, \ldots, n\}$

Se $f[i]$ é mínimo em S,

então EXISTE uma solução ótima A tal que $i \in A$.

Demonstre a propriedade.

Algoritmo guloso

Devolve uma coleção máxima de intervalos disjuntos dois a dois.

INTERVALOS-DISJUNTOS (s, f, n)

0 ordene s e f de tal forma que

$$f[1] \leq f[2] \leq \cdots \leq f[n]$$

1 $A \leftarrow \{1\}$
2 $i \leftarrow 1$
3 **para** $j \leftarrow 2$ **até** n **faça**
4 **se** $s[j] \geq f[i]$
5 **então** $A \leftarrow A \cup \{j\}$
6 $i \leftarrow j$
7 **devolva** A

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.
Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Conclusão

Na linha 3 vale que

(i0) A é uma coleção máxima de intervalos disjuntos de $(s, f, j-1)$

O consumo de tempo do algoritmo INTERVALOS-DISJUNTOS é $\Theta(n \lg n)$.

Algoritmos gulosos

Algoritmo guloso

- procura maximal e acaba obtendo máximo
- procura ótimo local e acaba obtendo ótimo global

costuma ser

- muito simples e intuitivo
- muito eficiente
- difícil provar que está correto

Problema precisa ter

- subestrutura ótima (como na programação dinâmica)
- propriedade da escolha gulosa (*greedy-choice property*)

Exercícios

Exercício 22.A [CLRS 16.1-1]
Escreva um algoritmo de programação dinâmica para resolver o problema dos intervalos disjuntos. (Versão simplificadda do exercício: basta determinar o *tamanho* de uma coleção disjunta máxima.) Qual o consumo de tempo do seu algoritmo?

Exercício 22.B
Prove que o algoritmo guloso para o problema dos intervalos disjuntos está correto. (Ou seja, prove a propriedade da subestrutura ótima e a propriedade da escolha gulosa.)

Exercício 22.C [CLRS 16.1-2]
Mostre que a seguinte idéia também produz um algoritmo guloso correto para o problema da coleção disjunta máxima de intervalos: dentre os intervalos disjuntos dos já escolhidos, escolha um que tenha instante de início máximo. (Em outras palavras, suponha que os intervalos estão em ordem decrescente de início.)

Exercício 22.D [CLRS 16.1-4]
Nem todo algoritmo guloso resolve o problema da coleção disjunta máxima de intervalos. Mostre que nenhuma das três idéias a seguir resolve o problema. Idéia 1: Escolha o intervalo de menor duração dentre os que são disjuntos dos intervalos já escolhidos. Idéia 2: Escolha um intervalo seja disjunto dos já escolhidos e intercepte o menor número possível de intervalos ainda não escolhidos. Idéia 3: Escolha o intervalo disjunto dos já selecionados que tenha o menor instante de início.

Mais exercícios

Exercício 22.E [Coloração de intervalos. CLRS 16.1-3]
Queremos distribuir um conjunto de atividades no menor número possivel de salas. Cada atividade a_i ocupa um certo intervalo de tempo $[s_i, f_i)$; duas atividades podem ser programadas para a mesma sala somente se os correspondentes intervalos são disjuntos. Descreva um algoritmo guloso que resolve o problema. (Represente cada sala por uma cor; use o menor número possível de cores para pintar todos os intervalos.) Prove que o número de salas dado pelo algoritmo é, de fato, mínimo.

Exercício 22.F [pares de livros]
Suponha dado um conjunto de livros numerados de 1 a n. Suponha que o livro i tem peso $p[i]$ e que $0 < p[i] < 1$ para cada i. Problema: acondicionar os livros no menor número possível de envelopes de modo que cada envelope tenha no máximo 2 livros e o peso do conteúdo de cada envelope seja no máximo 1. Escreva um algoritmo guloso que calcule o número mínimo de envelopes. O consumo de tempo do seu algoritmo deve ser $\mathrm{O}(n \lg n)$. Mostre que seu algoritmo está correto (ou seja, prove a “greedy-choice property” e a “optimal substructure” apropriadas). Estime o consumo de tempo do seu algoritmo.

Mais exercícios ainda

Exercício 22.G [Bin-packing]
São dados objetos $1, \ldots, n$ e um número ilimitado de "latas". Cada objeto i tem "peso" w_i e cada lata tem "capacidade" 1: a soma dos pesos dos objetos colocados em uma lata não pode passar de 1. Problema: Distribuir os objetos pelo menor número possível de latas. Programe e teste as seguinte heurísticas. Heurística 1: examine os objetos na ordem dada; tente colocar cada objeto em uma lata já parcialmente ocupada que tiver mais "espaço" livre sobrando; se isso for impossível, pegue uma nova lata. Heurística 2: rearranje os objetos em ordem decrescente de peso; em seguida, aplique a heurística 1. Essas herísticas resolvem o problema? Compare com o exercício 22.F.
Para testar seu programa, sugiro escrever uma rotina que receba $n \leq 100000$ e $u \leq 1$ e gere $w_1, \ldots, w_n$ aleatoriamente, todos no intervalo $(0, u)$.

Exercício 22.H [parte de CLRS 16-4, modificado]
Seja $1, \ldots, n$ um conjunto de *tarefas*. Cada tarefa consome um dia de trabalho; durante um dia de trabalho somente uma das tarefas pode ser executada. Os dias de trabalho são numerados de 1 a n. A cada tarefa t está associado um *prazo* p_t: a tarefa deveria ser executada em algum dia do intervalo $1 \,.\,. \, p_t$. A cada tarefa t está associada uma *multa* não-negativa m_t. Se uma dada tarefa t é executada depois do prazo p_t, sou obrigado a pagar a multa m_t (mas a multa não depende do número de dias de atraso). Problema: Programar as tarefas (ou seja, estabelecer uma bijeção entre as tarefas e os dias de trabalho) de modo a minimizar a multa total. Escreva um algoritmo guloso para resolver o problema. Prove que seu algoritmo está correto (ou seja, prove a "greedy-choice property" e

Análise amortizada

CLR 18 ou CLRS 17

Contador binário

Incrementa de 1 o número binário representado por $A[1 \mathrel{.\,.} k]$.

INCREMENT (A, k)
1 $\quad i \leftarrow 0$
2 $\quad$ **enquanto** $i < k$ **e** $A[i] = 1$ **faça**
3 $\qquad A[i] \leftarrow 0$
4 $\qquad i \leftarrow i + 1$
5 $\quad$ **se** $i < k$
6 $\qquad$ **então** $A[i] \leftarrow 1$

Contador binário

Incrementa de 1 o número binário representado por $A[1\,..\,k]$.

INCREMENT (A, k)
1 $i \leftarrow 0$
2 **enquanto** $i < k$ **e** $A[i] = 1$ **faça**
3 $A[i] \leftarrow 0$
4 $i \leftarrow i + 1$
5 **se** $i < k$
6 **então** $A[i] \leftarrow 1$

Entrada:

$k-1$		3	2	1	0	
0	1	0	1	1	1	A

Contador binário

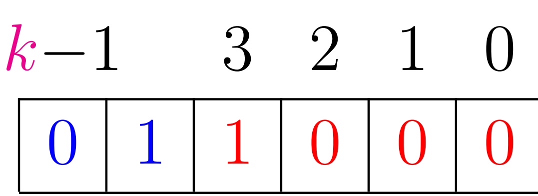
Incrementa de 1 o número binário representado por $A[1\,.\,.\,k]$.

INCREMENT (A, k)
1 $\quad i \leftarrow 0$
2 $\quad$ **enquanto** $i < k$ **e** $A[i] = 1$ **faça**
3 $\qquad A[i] \leftarrow 0$
4 $\qquad i \leftarrow i + 1$
5 $\quad$ **se** $i < k$
6 $\qquad$ **então** $A[i] \leftarrow 1$

Entrada:

$k-1$		3	2	1	0	
0	1	0	1	1	1	A

Saída:



Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\Theta(1)$
2	$O(k)$
3	$O(k)$
4	$O(k)$
5	$\Theta(1)$
6	$O(1)$
total	$O(k) + \Theta(1) = O(k)$

"Custo" =

consumo de tempo = número de bits alterados = $O(k)$

Seqüência de n chamadas

$$\underbrace{\text{INCR} \quad \text{INCR} \quad \cdots \quad \text{INCR} \quad \text{INCR} \quad \text{INCR}}_{n}$$

Consumo de tempo é $\mathrm{O}(nk)$

Seqüência de n chamadas

$$\underbrace{\text{INCR} \quad \text{INCR} \quad \cdots \quad \text{INCR} \quad \text{INCR} \quad \quad \text{INCR}}_{n}$$

Consumo de tempo é $\mathrm{O}(nk)$

EXAGERO!

Exemplo

$n = 16$ $k = 6$

A	A
5 4 3 2 1 0	5 4 3 2 1 0
0 0 0 0 0 0	0 0 1 0 0 0
0 0 0 0 0 1	0 0 1 0 0 1
0 0 0 0 1 0	0 0 1 0 1 0
0 0 0 0 1 1	0 0 1 0 1 1
0 0 0 1 0 0	0 0 1 1 0 0
0 0 0 1 0 1	0 0 1 1 0 1
0 0 0 1 1 0	0 0 1 1 1 0
0 0 0 1 1 1	0 0 1 1 1 1
	0 1 0 0 0 0

Exemplo

$n = 16$ $k = 6$

A	A
5 4 3 2 1 0	5 4 3 2 1 0
0 0 0 0 0 0	0 0 1 0 0 0
0 0 0 0 0 1	0 0 1 0 0 1
0 0 0 0 1 0	0 0 1 0 1 0
0 0 0 0 1 1	0 0 1 0 1 1
0 0 0 1 0 0	0 0 1 1 0 0
0 0 0 1 0 1	0 0 1 1 0 1
0 0 0 1 1 0	0 0 1 1 1 0
0 0 0 1 1 1	0 0 1 1 1 1
	0 1 0 0 0 0

$A[0]$	muda	n	vezes
$A[1]$	"	$\lfloor n/2 \rfloor$	"
$A[2]$	"	$\lfloor n/4 \rfloor$	"
$A[3]$	"	$\lfloor n/8 \rfloor$	"

Custo amortizado

Custo total:

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor \; < \; n \sum_{i=0}^{\infty} \frac{1}{2^i} \; = \; 2n \; = \; \Theta(n)$$

Custo amortizado (= custo fictício = custo faz-de-conta) de uma operação:

$$\frac{2n}{n} \; = \; \Theta(1)$$

Este foi o método agregado de análise: soma os custos de todas as operações para determinar o custo amortizado de cada operação

Conclusões

O consumo de tempo de uma seqüência de n execuções do algoritmo INCREMENT é $\Theta(n)$.

O consumo de tempo amortizado do algoritmo INCREMENT é $\Theta(1)$.

Método de análise contábil

Pague $2 para mudar $0 \to 1$

$0 para mudar $1 \to 0$

Custo amortizado por chamada de INCREMENT: $\leq$ $2

Seqüência de n chamadas de INCREMENT.

Como $ armazenado nunca é negativo,

$$
\begin{aligned}
\text{soma custos reais} &\leq \\
&\leq \text{soma custos amortizados} \\
&= 2n \\
&= \mathrm{O}(n)
\end{aligned}
$$

Método de análise potencial

$$A_0 \xrightarrow{1^a \text{ op}} A_1 \xrightarrow{2^a \text{ op}} A_2 \longrightarrow \cdots \xrightarrow{n^a \text{ op}} A_n$$

A_i = estado de A depois da i^a operação

Custo real da i^a operação: $c_i = t_i + 1$
onde t_i = número de $1 \to 0$ na i^a operação

Energia potencial de A_i:

$$\begin{aligned} \Phi(A_i) &= \text{número de bits "1"} \\ &= \Phi(A_{i-1}) - t_i + 1 \\ &\geq 0 \quad \text{Importante!} \end{aligned}$$

Custo amortizado

Custo amortizado da i^{a} operação:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(A_i) - \Phi(A_{i-1}) \\ &= c_i + \Phi(A_{i-1}) - t_i + 1 - \Phi(A_{i-1}) \\ &= c_i - t_i + 1 \\ &= (t_i + 1) - t_i + 1 \\ &= 2\end{aligned}$$

Custo amortizado

Soma dos custos amortizados limita a soma dos custos reais pois $\Phi \geq 0$:

$$\begin{aligned}
\sum_{i=1}^{n} \hat{c}_i &= \sum (c_i + \Phi(A_i) - \Phi(A_{i-1})) \\
&= \sum c_i + \Phi(A_n) - \Phi(A_0) \\
&= \sum c_i + \Phi(A_n) - 0 \\
&= \sum c_i + \Phi(A_n) \\
&\geq \sum c_i \,, \qquad \text{(pois } \Phi(A_n) \geq 0\text{)} \\
&\geq \sum c_i \\
&= \text{soma dos custos reais}
\end{aligned}$$

Conclusões

$$\sum c_i \le \sum \hat{c}_i = 2n = \mathrm{O}(n)\,.$$

Resumo da relação entre c, $\hat{c}$ e Φ:

$$\hat{c}_i - c_i = \Phi_i - \Phi_{i-1}$$

Pilhas

PUSH(S, x) empilha o elemento x na pilha S

POP(S) desempilha e devolve o elemento no topo de S

STACK-EMPTY(S) devolve VERDADE se a pilha S está vazia e FALSO em caso contrário.

MULTIPOP (S, k)

1 **enquanto** STACK-EMPTY$(S) =$ FALSO **e** $k \neq 0$ **faça**
2 POP(S)
3 $k \leftarrow k - 1$

Consumo de tempo de MULTIPOP é $\mathrm{O}(\min\{|S|, k\})$.

Seqüência de n chamadas

$$\underbrace{\text{PUSH} \quad \text{PUSH} \quad \cdots \quad \text{POP} \quad \text{PUSH} \quad \text{MULTIPOP}}_{n}$$

Consumo de tempo é $O(n^2)$

Seqüência de n chamadas

$$\underbrace{\text{PUSH} \quad \text{PUSH} \quad \cdots \quad \text{POP} \quad \text{PUSH} \quad \text{MULTIPOP}}_{n}$$

Consumo de tempo é $O(n^2)$

EXAGERO!

Método agregado

Cada elemento pode ser desempilhado apenas 1 vez.

Logo, número de POP's em uma pilha não vazia $\leq$ número de PUSH's.

Custo total:

$=$ custo PUSH's $+$ custo POP's $+$ custo MULTIPOP's

$\leq n +$ custo POP's pilha ñ vazia

$\leq 2\,n = \mathrm{O}(n)$

Custo amortizado de cada operação:

$$\frac{2n}{n} = \Theta(1)$$

Conclusões

O consumo de tempo de uma seqüência de n execuções dos algoritmos POP, PUSH e MULTIPOP é $\Theta(n)$.

O consumo de tempo amortizado de cada algoritmo é $\Theta(1)$.

Método de análise contábil

Custos reais:

PUSH	\$1
POP	\$1
MULTIPOP	\$ $\min\{|S|, k\}$

Créditos:

Pague \$2 por um PUSH

\$1 por um POP

\$1 por um MULTIPOP

Custo amortizado por chamada de POP, PUSH e MULTIPOP: $\leq$ \$2

Método de análise contabil

Como $ armazenado nunca é negativo,

$$
\begin{aligned}
\text{soma custos reais} &\leq \\
&\leq \text{soma custos amortizados} \\
&\leq 2n \\
&= \mathrm{O}(n)
\end{aligned}
$$

Método de análise potencial

$$S_0 \xrightarrow{\text{1}^{\text{a}}\text{ op}} S_1 \xrightarrow{\text{2}^{\text{a}}\text{ op}} S_2 \longrightarrow \cdots \xrightarrow{n^{\text{a}}\text{ op}} S_n$$

S_i = estado de S depois da i^{a} operação

Energia potencial de S_i:

$$\begin{aligned} \Phi(S_i) &= \text{número de objetos na pilha} \\ &\geq 0 \quad \text{(Importante!)} \end{aligned}$$

Custo amortizado PUSH

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se a i^{a} operação é PUSH, então $c_i = 1$ e

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(S_i) - \Phi(S_{i-1}) \\
&= c_i + 1 \\
&= 1 + 1 \\
&= 2
\end{aligned}$$

Custo amortizado Pop

c_i = custo real da $i^{\rm a}$ operação.

Custo amortizado da $i^{\rm a}$ operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se a $i^{\rm a}$ operação é POP, então $c_i = 1$ e

$$\begin{aligned} \hat{c}_i &= c_i + \Phi(S_i) - \Phi(S_{i-1}) \\ &= c_i - 1 \\ &= 1 - 1 \\ &= 0 \end{aligned}$$

Custo amortizado MULTIPOP

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se a i^{a} operação é MULTIPOP e k' é o número de elementos desempilhados, então $c_i = k'$ e

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(S_i) - \Phi(S_{i-1}) \\
&= c_i - k' \\
&= k' - k' \\
&= 0
\end{aligned}$$

Custo amortizado

Soma dos custos amortizados limita a soma dos custos reais pois $\Phi \geq 0$:

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &= \sum (\hat{c}_i - \Phi(S_i) + \Phi(S_{i-1})) \\
&= \sum \hat{c}_i \;-\; \Phi(S_n) + \Phi(S_0) \\
&= \sum \hat{c}_i \;-\; \Phi(S_n) + 0 \\
&= \sum \hat{c}_i \;-\; \Phi(S_n) \\
&\leq \sum \hat{c}_i \,, \qquad \text{(pois } \Phi(S_n) \geq 0\text{)} \\
&\leq \sum 2 \\
&= 2\,n
\end{aligned}
$$

AULA 20

Mais análise amortizada

CLR 18 ou CLRS 17

Análise amortizada

Análise amortizada $=$ análise do consumo de tempo de uma seqüência de operações

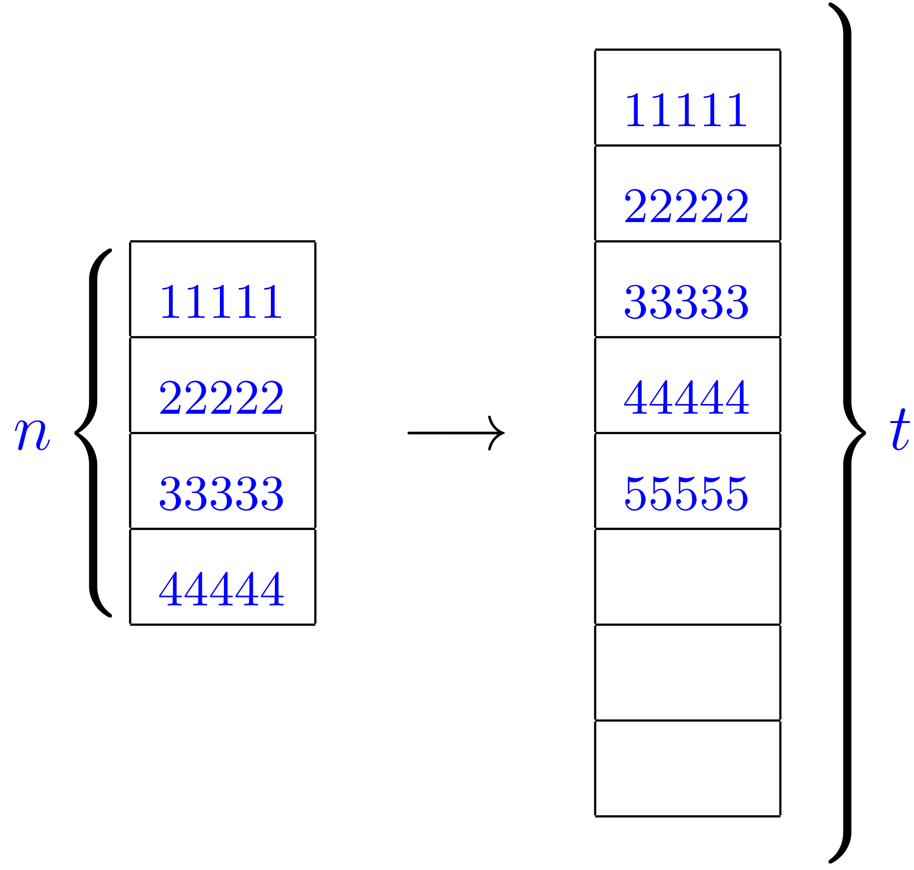
Usada nos casos em que

o consumo de tempo no pior caso de n operações é menor que n vezes o consumo de tempo no pior caso de uma operação

Mais análise amortizada

CLR 18 ou CLRS 17

Tabelas dinâmicas



$n[T]$= número de itens $\quad$ $t[T]$= tamanho de T

Inicialmente $n[T] = t[T] = 0$

Inserção

Insere um elemento x na tabela T

TABLE-INSERT (T, x)

1 **se** $t[T] = 0$
2 **então** aloque $tabela[T]$ com 1 posição
3 $t[T] \leftarrow 1$
4 **se** $n[T] = t[T]$
5 **então** aloque $nova\text{-}tabela$ com $2\,t[T]$ posições
6 insira itens da $tabela[T]$ na $nova\text{-}tabela$

---

7 $t[nova\text{-}tabela] \leftarrow 2\,t[T]$
8 libere $tabela[T]$
9 $tabela[T] \leftarrow nova\text{-}tabela$
10 insira x na $tabela[T]$

---

11 $n[T] \leftarrow n[T] + 1$

Custo $=$ número de inserções elementares (linhas 6 e 10)

Seqüência de m TABLE-INSERTS

$$T_0 \xrightarrow{1^{\text{a}}\text{ op}} T_1 \xrightarrow{2^{\text{a}}\text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\text{a}}\text{ op}} T_m$$

T_i = estado de T depois da i^{a} operação

Custo real da i^{a} operação:

$$c_i = \begin{cases} 1 & \text{se há espaço} \\ n_i & \text{se tabela cheia} \end{cases}$$

onde n_i = valor de $n[T]$ depois da i^{a} operação
= i

Custo de uma operação = $\mathrm{O}(m)$

Custo das m operações = $\mathrm{O}(m^2)$ Exagero!

Exemplo

operação ($n[T]$)	$t[T]$	custo
1	1	1
2	2	$1+1$
3	4	$1+2$
4	4	1
5	8	$1+4$
6	8	1
7	8	1
8	8	1
9	16	$1+8$
10	16	1
16	16	1
17	32	$1+16$
33	64	$1+32$

11111	$\longrightarrow$	11111
		22222

11111	$\longrightarrow$	11111
22222		22222
		33333
		44444

11111	$\longrightarrow$	11111
22222		22222
33333		$\vdots$
44444		88888

Custo amortizado

Custo total:

$$\sum_{i=1}^{n} c_i \;=\; m + \sum_{i=0}^{k} 2^i \;=\; n + 2^{k+1} - 1 \;<\; n + 2m - 1 \;<\; 3m$$

onde $k = \lfloor \lg(m-1) \rfloor$

Custo amortizado:

$$\frac{3m}{m} \;=\; 3 \;=\; \Theta(1)$$

Conclusões

O custo de uma seqüência de m execuções do algoritmo TABLE-INSERT é $\Theta(m)$.

O custo amortizado do algoritmo TABLE-INSERT é $\Theta(1)$.

Método de análise agregada

- m operações consomem tempo $T(m)$
- custo médio de cada operação é $T(m)/m$
- custo amortizado de cada operação é $T(m)/m$
- defeito: no caso de mais de um tipo de operação, o custo de cada tipo não é determinado separadamente

Método de análise contábil

TABLE-INSERT (T, x)

$credito \leftarrow credito + 3$

1 **se** $t[T] = 0$

2 **então** aloque $tabela[T]$ com 1 posição

3 $t[T] \leftarrow 1$

4 **se** $n[T] = t[T]$

5 **então** aloque $nova\text{-}tabela$ com $2\,t[T]$ posições

6 insira itens da $tabela[T]$ na $nova\text{-}tabela$

$custo \leftarrow custo + n[T]$

7 libere $tabela[T]$

8 $tabela[T] \leftarrow nova\text{-}tabela$

9 $t[T] \leftarrow 2\,t[T]$

10 insira x na $tabela[T]$

---

11 $n[T] \leftarrow n[T] + 1$

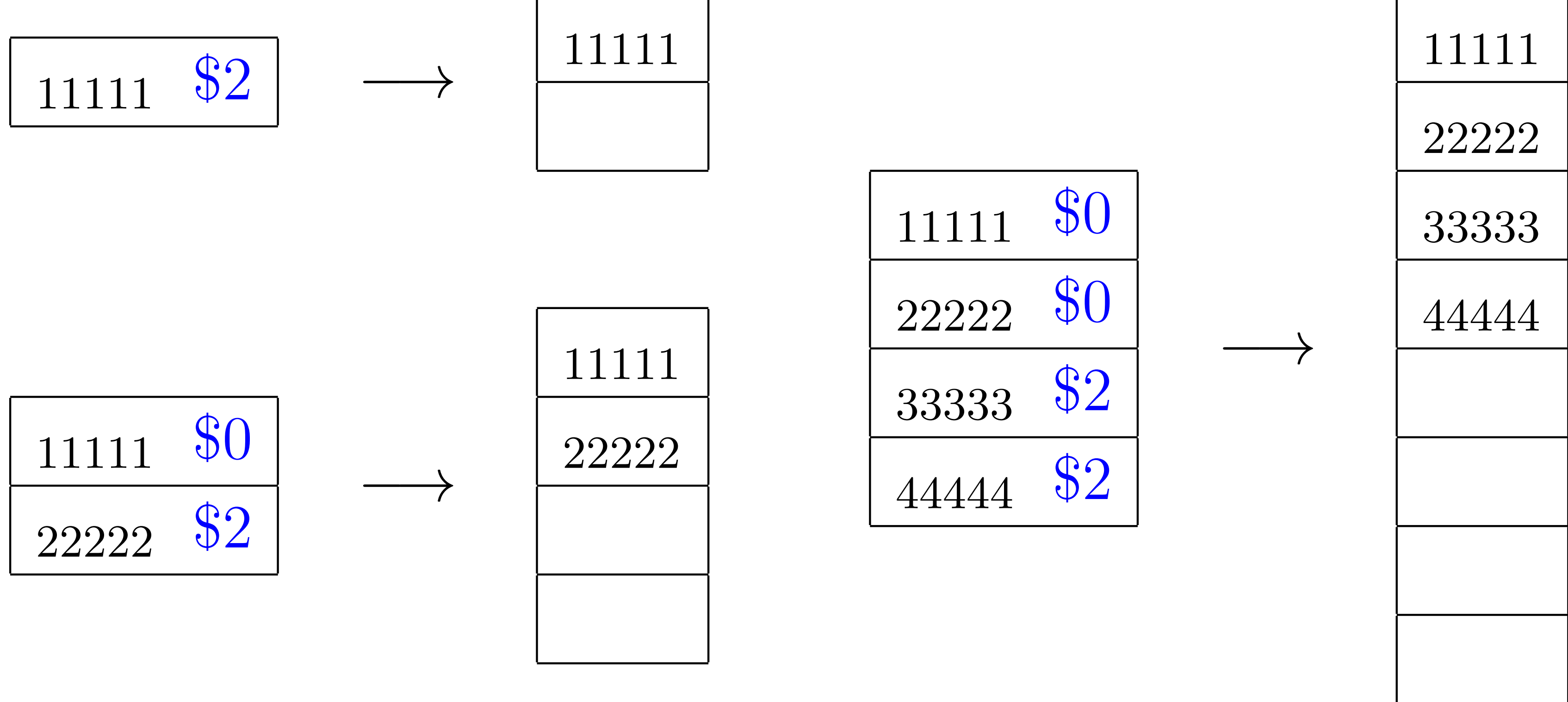
$custo \leftarrow custo + 1$

Método de análise contábil

Invariante: soma créditos $\geq$ soma custos reais

$n[T]$	$t[T]$	custo	crédito	saldo
1	1	1	3	2
2	2	$1+1$	3	3
3	4	$1+2$	3	3
4	4	1	3	5
5	8	$1+4$	3	3
6	8	1	3	5
7	8	1	3	7
8	8	1	3	9
9	16	$1+8$	3	3
10	16	1	3	5
16	16	1	3	17
17	32	$1+16$	3	3

Método de análise contábil



Método de análise contábil

Pague $1 para inserir um novo elemento
guarde $1 para eventualmente mover o novo elemento
guarde $1 para mover um elemento que já está na tabela

Custo amortizado por chamada de TABLE-INSERT: $\leq$ $3
Seqüência de m chamadas de TABLE-INSERT.
Como $ armazenado nunca é negativo,

$$\begin{aligned} \text{soma custos reais} &\leq \text{soma custos amortizados} \\ &= 3m \\ &= \mathrm{O}(m) \end{aligned}$$

Método de análise contábil

- cada operação paga seu custo real
- cada operação recebe um certo número de créditos (chute de custo amortizado)
- balanço nunca pode ser negativo

  soma créditos $\geq$ soma custos reais

  créditos não usados são guardados para pagar operações futuras.
- custo amortizado da operação $\leq$ número médio de créditos recebidos
- custo amortizado de cada tipo de operação pode ser determinado separadamente

Método de análise potencial

Função potencial: $\Phi(T) := 2\,n[T] - t[T]$

$n[T]$	$t[T]$	custo	$\Phi(T)$	$\Delta\Phi$	custo$+\Delta\Phi$
1	1	1	1	$+1$	2
2	2	$1+1$	2	$+1$	3
3	4	$1+2$	2	0	3
4	4	1	4	$+2$	3
5	8	$1+4$	2	-2	3
6	8	1	4	$+2$	3
7	8	1	6	$+2$	3
8	8	1	8	$+2$	3
9	16	$1+8$	2	-6	3
10	16	1	4	$+2$	3
16	16	1	16	$+2$	3
17	32	$1+16$	2	-14	3

Método de análise potencial

Função potencial: $\Phi(T) := 2\,n[T] - t[T]$

Note que $0 \;\leq\; \Phi(T) \;\leq\; t[T]$

Cálculo do custo amortizado $\hat{c}_i$:

Se i^{a} operação não causa expansão então

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2n_i - t_i) - (2n_{i-1} - t_{i-1}) \\
&= 1 + (2n_i - t_i) - (2(n_i - 1) - t_i) \\
&= 3
\end{aligned}$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Método de análise potencial

Se i^{a} operação causa expansão então

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= n_i + (2n_i - t_i) - (2n_{i-1} - t_{i-1}) \\
&= n_i + (2n_i - 2n_{i-1}) - (2n_{i-1} - n_{i-1}) \\
&= n_i + 2n_i - 3n_{i-1} \\
&= n_i + 2n_i - 3(n_i - 1) \\
&= 3
\end{aligned}
$$

Conclusão: $\hat{c}_i = 3$ para qualquer $i \geq 2$

Método de análise potencial

O custo real das n operações é limitado pelo custo amortizado pois $\Phi \geq 0$:

$$
\begin{aligned}
\sum_{i=1}^{m} c_i &= \sum \hat{c}_i - \Phi_m + \Phi_0 \\
&= \sum \hat{c}_i - \Phi_m \\
&\leq \sum \hat{c}_i \\
&= 3m \\
&= \mathrm{O}(m)
\end{aligned}
$$

Conclusões

O custo de uma seqüência de m execuções do algoritmos TABLE-INSERT é $\Theta(m)$.

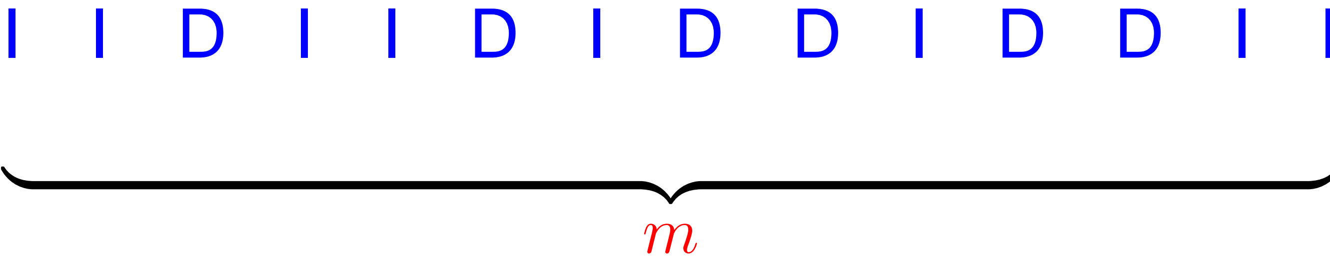
O custo amortizado do algoritmo TABLE-INSERT é $\Theta(1)$.

Método de análise potencial

- método contábil visto como energia potencial
- potencial associado à estrutura de dados

Seqüencia de INSERT e DELETE

Seqüência de operações TABLE-INSERT e TABLE-DELETE



Custo total de uma seqüência de TABLE-INSERT e TABLE-DELETE?

Remoção

Remove um elemento x da tabela T

TABLE-DELETE (T, x) $\triangleright$ supõe x na $tabela[T]$

1 remova x da $tabela[T]$

---

2 $n[T] \leftarrow n[T] - 1$
3 **se** $n[T] < t[T]/\mathbf{2}$ $\triangleright$ tabela está "vazia"?
4 **então** aloque $nova\text{-}tabela$ com $t[T]/\mathbf{2}$ posições
5 insira itens da $tabela[T]$ na $nova\text{-}tabela$

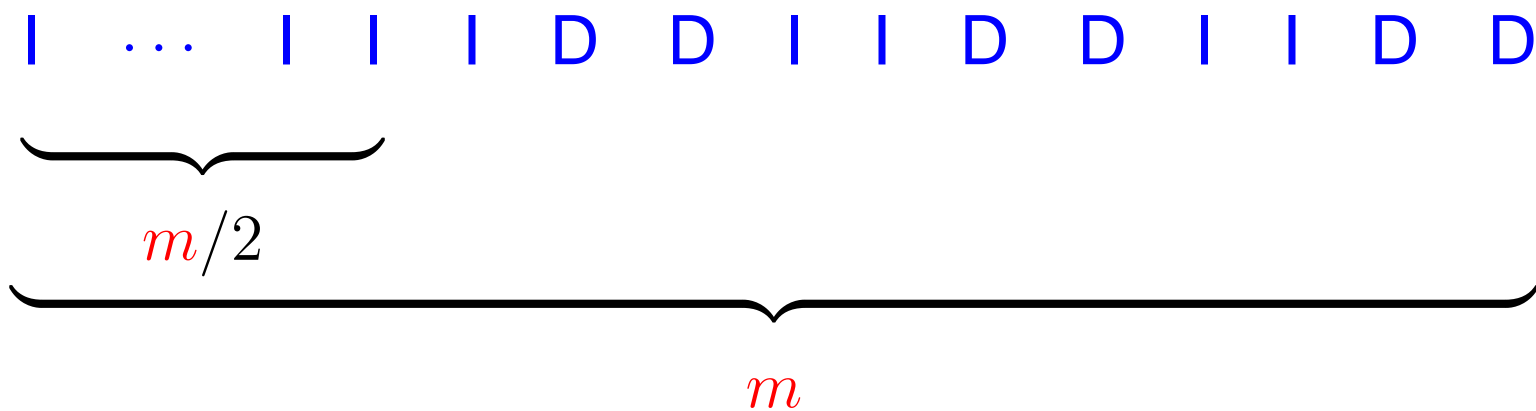
---

6 $t[nova\text{-}tabela] \leftarrow t[T]/2$
7 $n[nova\text{-}tabela] \leftarrow n[T]$
8 libere $tabela[T]$
9 $tabela[T] \leftarrow nova\text{-}tabela$

Custo $=$ número de remoções e inserções elementares (linhas 1 e 5)

Seqüência de INSERT e DELETE

Seqüência de operações TABLE-INSERT e TABLE-DELETE



Se $m = 4\,k$, então custo total da seqüência:

$$\Theta(2\,k) + k\,\Theta(2\,k) = \Theta\left(\frac{m}{2}\right) + \frac{m}{4}\,\Theta\left(\frac{m}{2}\right) = \Theta(m^2)$$

Remoção

Remove um elemento x da tabela T

TABLE-DELETE (T, x) ▷ supõe x na $tabela[T]$

1 remova x da $tabela[T]$

---

2 $n[T] \leftarrow n[T] - 1$

3 **se** $n[T] < t[T]/\mathbf{4}$ ▷ tabela está “vazia”?

4 **então** aloque $nova\text{-}tabela$ com $t[T]/\mathbf{2}$ posições

5 insira itens da $tabela[T]$ na $nova\text{-}tabela$

---

6 $t[nova\text{-}tabela] \leftarrow t[T]/\mathbf{2}$

7 $n[nova\text{-}tabela] \leftarrow n[T]$

8 libere $tabela[T]$

9 $tabela[T] \leftarrow nova\text{-}tabela$

Custo $=$ número de remoções e inserções elementares (linhas 1 e 5)

Seqüência de m operações

$$T_0 \xrightarrow{1^{\text{a}}\text{ op}} T_1 \xrightarrow{2^{\text{a}}\text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\text{a}}\text{ op}} T_m$$

T_i = estado de T depois da i^{a} operação

Custo real da i^{a} operação se for TABLE-INSERT:

$$c_i = \begin{cases} 1 & \text{se há espaço} \\ n_i & \text{se tabela cheia} \end{cases}$$

onde n_i = valor de $n[T]$ depois da i^{a} operação

Custo de uma operação = $\mathrm{O}(m)$

Seqüência de m operações

$$T_0 \xrightarrow{1^{\text{a}} \text{ op}} T_1 \xrightarrow{2^{\text{a}} \text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\text{a}} \text{ op}} T_m$$

T_i = estado de T depois da i^{a} operação

Custo real da i^{a} operação se for TABLE-DELETE:

$$c_i = \begin{cases} 1 & \text{se } n_{i-1} > t_{i-1}/4 \\ 1 + n_i & \text{se } n_{i-1} = t_{i-1}/4 \end{cases}$$

onde n_i = valor de $n[T]$ depois da i^{a} operação

e t_i = valor de $t[T]$ depois da i^{a} operação

Custo de uma operação = $\mathrm{O}(m)$

Custo das m operações = $\mathrm{O}(m^2)$ Exagero!

Método de análise potencial

$$T_0 \xrightarrow{1^{\text{a}}\text{ op}} T_1 \xrightarrow{2^{\text{a}}\text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\text{a}}\text{ op}} T_m$$

T_i = estado de T depois da i^{a} operação

Energia potencial de T_i:

$$\Phi(T_i) = \begin{cases} 2\,n_i - t_i, & \text{se } n_i \geq t_i/\mathbf{2} \\ t_i/2 - n_i, & \text{se } n_i < t_i/\mathbf{2} \end{cases}$$

Note que $0 \leq \Phi(T_i) \leq t[T_i]$

n_i, t_i, Φ_i = $n[T_i], t[T_i], \Phi(T_i)$ depois da i^{a} operação

Custo amortizado TABLE-INSERT (1)

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se i^{a} operação é TABLE-INSERT, $n_{i-1} \geq t_{i-1}/\mathbf{2}$ e não houve expansão, então

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2n_i - t_i) - (2n_{i-1} - t_{i-1}) \\
&= 1 + (2n_i - t_i) - (2(n_i - 1) - t_i) \\
&= 3
\end{aligned}$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-INSERT (2)

Se i^{a} operação é TABLE-INSERT, $n_{i-1} \geq t_{i-1}/\mathbf{2}$ e houve expansão, então

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= n_i + (2n_i - t_i) - (2n_{i-1} - t_{i-1}) \\
&= n_i + (2n_i - 2n_{i-1}) - (2n_{i-1} - n_{i-1}) \\
&= n_i + 2n_i - 3n_{i-1} \\
&= n_i + 2n_i - 3(n_i - 1) \\
&= 3
\end{aligned}
$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-INSERT (3)

Se i^{a} operação é TABLE-INSERT, $n_{i-1} < t_{i-1}/\mathbf{2}$ e $n_i < t_i/\mathbf{2}$, então não houve expansão e

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (t_i/2 - n_i) - (t_{i-1}/2 - n_{i-1}) \\
&= 1 + (t_i/2 - t_i/2) - (n_i - (n_i - 1)) \\
&= 0\,.
\end{aligned}$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-INSERT (4)

Se i^{a} operação é TABLE-INSERT e $n_{i-1} < t_{i-1}/\mathbf{2}$ e $n_i \geq t_i/\mathbf{2}$, então não houve expansão e

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2n_i - t_i) - (t_{i-1}/2 - n_{i-1}) \\
&= 1 + (2(n_{i-1} + 1) - t_{i-1}) - (t_{i-1}/2 - n_{i-1})) \\
&= 3\,n_{i-1} - 3\,(t_{i-1}/2) + 3 \\
&< 3\,n_{i-1} - 3\,n_{i-1} + 3 \\
&= 3\,.
\end{aligned}$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-DELETE (1)

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se i^{a} operação é TABLE-DELETE e $n_{i-1} < t_{i-1}/\mathbf{2}$ e não houve contração, então

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (t_i/2 - n_i) - (t_{i-1}/2 - n_{i-1}) \\ &= 1 + (t_i/2 - t_i/2) - ((n_i - 1) - n_i)) \\ &= 2\,. \end{aligned}$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-DELETE (2)

Se i^{a} operação é TABLE-DELETE e $n_{i-1} < t_{i-1}/\mathbf{2}$ e houve contração, então

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + n_i + (t_i/2 - n_i) - (t_{i-1}/2 - n_{i-1}) \\
&= 1 + n_i + ((n_i + 1) - n_i) - ((2\,n_i + 2) - (n_i + 1)) \\
&= 1\,.
\end{aligned}$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Quando $n_{i-1} \geq t_{i-1}/\mathbf{2}$, o custo amortizado da operação é TABLE-DELETE também é uma constante [CLRS 17.4-2].

Conclusões

O custo de uma seqüência de m execuções dos algoritmos TABLE-INSERT e TABLE-DELETE é $\Theta(m)$.

O custo amortizado dos algoritmos TABLE-INSERT e TABLE-DELETE é $\Theta(1)$.

Class ArrayList

O Paulo (peas) me mostrou o trecho que foi copiado de
`http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html`

"... Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost. ..."

O Paulo também disse que na documentação do STL do `C++` tem algo semelhante.

Exercícios

Exercício 23.A [CLR 18.1-3 18.2-2 18.3-2, CLRS 17.1-3 17.2-2 17.3-2] Uma seqüência de n operações é executada sobre uma certa estrutura de dados. Suponha que a i-a operação custa

$$\begin{array}{ll} i & \text{se } i \text{ é uma potência de } 2, \\ 1 & \text{em caso contrário.} \end{array}$$

Mostre que o custo amortizado de cada operação é $O(1)$. Use o método da "análise agregada"; depois, repita tudo usando o método da função potencial.

Exercício 23.B [Importante]
Mostre que a função $\Phi(T) = t[T] - n[T]$ não é uma boa função potencial para a análise da tabela dinâmica T sob a operação TABLE-INSERT. Mostre que $\Phi(T) = t[T]$ também não é um bom potencial para a análise da tabela dinâmica.

Exercício 23.C [CLR 18.3-3, CLRS 17.3-3]
Considere a estrutura de dados min-heap munida das operações INSERT e EXTRACT-MIN. Cada operação consome tempo $O(\lg n)$, onde n é o número de elementos na estrutura. Dê uma função potencial Φ tal que o custo amortizado de INSERT seja $O(\lg n)$ e o custo amortizado de EXTRACT-MIN seja $O(1)$. Prove que sua função potencial de fato tem essas propriedades.

Outro exercício

Exercício 23.D [CLR 18-2, CLRS 17-2, Busca binária dinâmica]
Busca binária em um vetor ordenado consome tempo logarítmico, mas o tempo necessário para inserir um novo elemento é linear no tamanho do vetor. Isso pode ser melhorado se mantivermos diversos vetores ordenados (em lugar de um só). Suponha que queremos implementar as operações BUSCA e INSERÇÃO em um conjunto de n elementos. Seja $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$ a representação binária de n, onde $k = \lceil \lg(n+1) \rceil$. Temos vetores crescentes $A_0, A_1, \ldots, A_{k-1}$, sendo que o comprimento de cada A_i é 2^i. Um vetor A_i é utilizado se $n_i = 1$ e inutilizado se $n_i = 0$. O número total de elementos utilizados nos k vetores é, portanto,

$$\sum_{i=0}^{k-1} n_i \, 2^i = n \, .$$

Cada vetor é crescente, mas não há qualquer relação entre os valores dos elementos em dois vetores diferentes.

a. Dê um algoritmo para a operação BUSCA. Dê uma delimitação superior para o consumo de tempo do algoritmo.

b. Dê um algoritmo para a operação INSERÇÃO. Dê uma delimitação superior para o consumo de tempo do algoritmo. Calcule o consumo de tempo *amortizado*.

c. Discuta uma implementação da operação REMOÇÃO.

Mais um exercício

Exercício 23.E
Descreva um algoritmo que receba um inteiro positivo n e calcule n^n fazendo não mais que $2 \lg n$ multiplicações de números inteiros.

Busca de palavras (string matching)

CLRS 32

Busca de palavras em um texto

Dizemos que um vetor $P[1\,..\,m]$ ocorre em um vetor $T[1\,..\,n]$ se

$$P[1\,..\,m] = T[s+1\,..\,s+m]$$

para algum s em $[0\,..\,n-m]$.

Exemplo:

	1	2	3	4	5	6	7	8	9	10
T	x	c	b	a	b	b	c	b	a	x

	1	2	3	4
P	b	c	b	a

$P[1..4]$ ocorre em $T[1..10]$ com deslocamento 5.

O valor s é um descolamento válido

Busca de palavras em um texto

Problema: Dados $P[1\,..\,m]$ e $T[1\,..\,n]$, encontrar todos os deslocamentos válidos.

NAIVE-STRING-MATCHER (P, m, T, n)

1 **para** $i \leftarrow 1$ **até** $n - m$ **faça**
2 $\quad q \leftarrow 0$
3 $\quad$ **enquanto** $q < m$ **e** $P[q+1] = T[i+q]$ **faça**
4 $\quad\quad q \leftarrow q + 1$
5 $\quad$ **se** $q = m$
6 $\quad\quad$ **então** "P ocorre com deslocamento $i - 1$"

Relação invariante: na linha 3 vale que

(i0) $P[1\,..\,q] = T[i\,..\,i{+}q{-}1]$

Simulação

$P =$ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	T
1	a	b	a	b																				
2		a																						
3			a	b																				
4				a	b	a	b	b																
5					a																			
6						a	b	a	b	b	a	b	a	b	b									
7							a																	
8								a	b	a														
9									a															
10										a														
11											a	b	a	b										
12												a												
13													a	b	a	b	b	a	b	a	b	b	a	

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\Theta(n - m + 1)$
2	$\Theta(n - m)$
3	$O((n - m)(m + 1)) = O((n - m)m)$
4	$O((n - m)m)$
5	$\Theta(n - m)$
6	$O(n - m)$

total $\Theta(3(n - m) + 1) + O(2(n - m)m + n - m)$

$= O((n - m + 1)m)$

Conclusões

O consumo de tempo do algoritmo NAIVE-STRING-MATCHER é $O((n - m + 1)m)$.

No pior caso, o consumo de tempo do algoritmo NAIVE-STRING-MATCHER é $\Theta((n - m + 1)m)$.

Busca de palavras em um texto

NAIVE-STRING-MATCHER (P, m, T, n)

1 $i \leftarrow 1 \quad q \leftarrow 0$
2 **enquanto** $i \leq n - m + 1$ **faça**
3 **se** $q = m$ **então** ▷ caso 1
4 "P ocorre com deslocamento $i - 1$"
5 $q \leftarrow 0$
6 $i \leftarrow i + 1$
7 **senão se** $P[q{+}1] = T[i{+}q]$ **então** ▷ caso 2
8 $q \leftarrow q + 1$
9 **senão se** $P[q{+}1] \neq T[i{+}q]$ **então** ▷ caso 3
10 $q \leftarrow 0$
11 $i \leftarrow i + 1$

Relação invariante: na linha 2 vale que

(i0) $P[1\,..\,q] = T[i\,..\,i{+}q{-}1]$

Consumo de tempo

Cada linha do algoritmo consome tempo $\Theta(1)$.

Caso 1 e caso 3 ocorrem $n - m + 1$ vezes (total).

Para cada valor de i o número de ocorrências do caso 2 é $\leq m$.

Logo, o número total de iterações das linhas 2–13 é $\leq (n - m + 1)m$.

Portanto, o consumo de tempo algoritmo é $\Theta((n - m + 1)m)$.

Mas, isto já sabíamos...

Simulação

$P =$ a b a b b a b a b b a

T

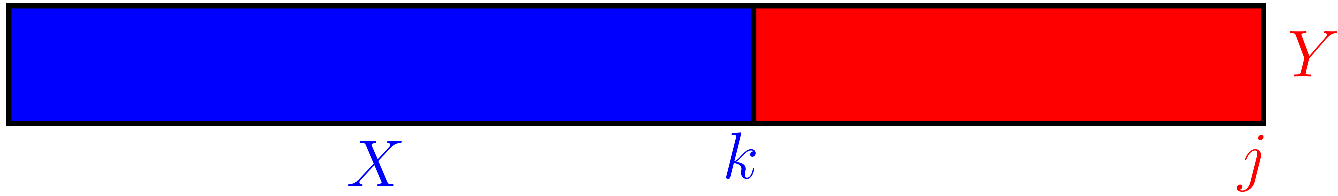
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	a	b	a	b																			
2		a																					
3			a	b																			
4				a	b	a	b	b															
5					a																		
6						a	b	a	b	b	a	b	a	b	b								
7							a																
8								a	b	a													
9									a														
10										a													
11											a	b	a	b									
12												a											
13													a	b	a	b	b	a	b	a	b	b	a

Prefixos e sufixos

$X[1\,..\,k]$ é prefixo de $Y[1\,..\,j]$ se

$$k \leq j \quad \text{e} \quad X[1\,..\,k] = Y[1\,..\,k].$$

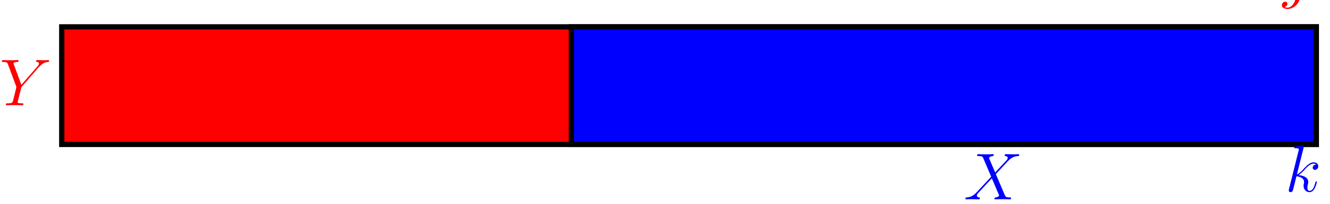
Se $k < j$, então $X[1\,..\,k]$ é prefixo próprio.



$X[1\,..\,k]$ é sufixo de $Y[1\,..\,j]$ se

$$k \leq j \quad \text{e} \quad X[1\,..\,k] = Y[j{-}k{+}1\,..\,j].$$

Se $k < j$, então $X[1\,..\,k]$ é sufixo próprio.



Exemplos

	1	2	3	4	5	6	7	8	9	10	11
P	a	b	a	b	b	a	b	a	b	b	a

$P[1\,..\,1]$ é prefixo próprio de $P[1\,..\,11]$

$P[1\,..\,3]$ é prefixo próprio de $P[1\,..\,7]$

$P[1\,..\,3]$ é sufixo próprio de $P[1\,..\,8]$

$P[1\,..\,5]$ é sufixo próprio de $P[1\,..\,10]$

$P[1\,..\,1]$ é sufixo próprio de $P[1\,..\,11]$

AULA 21

Busca de palavras (string matching)

CLRS 32

Busca de palavras em um texto

Dizemos que um vetor $P[1\,..\,m]$ ocorre em um vetor $T[1\,..\,n]$ se

$$P[1\,..\,m] = T[s+1\,..\,s+m]$$

para algum s em $[0\,..\,n{-}m]$.

Exemplo:

	1	2	3	4	5	6	7	8	9	10
T	x	c	b	a	b	b	c	b	a	x

	1	2	3	4
P	b	c	b	a

$P[1\,..\,4]$ ocorre em $T[1\,..\,10]$ com deslocamento 5.

O valor s é um descolamento válido.

Busca de palavras em um texto

Problema: Dados $P[1\,..\,m]$ e $T[1\,..\,n]$, encontrar todos os deslocamentos válidos.

Exemplo:

Para $n = 10$, $m = 4$, e

	1	2	3	4	5	6	7	8	9	10
T	b	b	a	b	a	b	a	c	b	a

	1	2	3	4
P	b	a	b	a

Os deslocamentos válidos são 1 e 3.

Simulação

$P =$ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	T
1	a	b	a	b																				
2		a																						
3			a	b																				
4				a	b	a	b	b																
5					a																			
6						a	b	a	b	b	a	b	a	b	b									
7							a																	
8								a	b	a														
9									a															
10										a														
11											a	b	a	b										
12												a												
13													a	b	a	b	b	a	b	a	b	b	a	

Naive-String-Matcher

Recebe $P[1\,..\,m]$ e $T[1\,..\,n]$, e devolve todos os deslocamentos válidos.

NAIVE-STRING-MATCHER (P, m, T, n)

1 **para** $i \leftarrow 1$ **até** $n - m$ **faça**
2 $\quad q \leftarrow 0$
3 $\quad$ **enquanto** $q < m$ **e** $P[q+1] = T[i+q]$ **faça**
4 $\quad\quad q \leftarrow q + 1$
5 $\quad$ **se** $q = m$
6 $\quad\quad$ **então** "P ocorre com deslocamento $i - 1$"

Relação invariante: na linha 3 vale que

(i0) $P[1\,..\,q] = T[i\,..\,i{+}q{-}1]$

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\Theta(n - m + 1)$
2	$\Theta(n - m)$
3	$\mathrm{O}((n - m)(m + 1)) = \mathrm{O}((n - m)m)$
4	$\mathrm{O}((n - m)m)$
5	$\Theta(n - m)$
6	$\mathrm{O}(n - m)$

total $\Theta(3(n - m) + 1) + \mathrm{O}(2(n - m)m + n - m)$

$$= \mathrm{O}((n - m + 1)m)$$

Conclusões

O consumo de tempo do algoritmo NAIVE-STRING-MATCHER é $O((n - m + 1)m)$.

No pior caso, o consumo de tempo do algoritmo NAIVE-STRING-MATCHER é $\Theta((n - m + 1)m)$.

Naive-String-Matcher

NAIVE-STRING-MATCHER (P, m, T, n)

1 $i \leftarrow 1 \quad q \leftarrow 0$
2 **enquanto** $i \leq n - m + 1$ **faça**
3 **se** $q = m$ **então** ▷ caso 1
4 "P ocorre com deslocamento $i - 1$"
5 $q \leftarrow 0$
6 $i \leftarrow i + 1$
7 **senão se** $P[q+1] = T[i+q]$ **então** ▷ caso 2
8 $q \leftarrow q + 1$
9 **senão se** $P[q+1] \neq T[i+q]$ **então** ▷ caso 3
10 $q \leftarrow 0$
11 $i \leftarrow i + 1$

Relação invariante: na linha 2 vale que

(i0) $P[1\,..\,q] = T[i\,..\,i+q-1]$

Consumo de tempo

Cada linha do algoritmo consome tempo $\Theta(1)$.

Caso 1 e caso 3 ocorrem $n - m + 1$ vezes (total).

Para cada valor de i o número de ocorrências do caso 2
é $\leq m$.

Logo, o número total de iterações das linhas 2–13
é $\leq (n - m + 1)m$.

Portanto, o consumo de tempo algoritmo é $\mathrm{O}((n - m + 1)m)$.

Mas, isto já sabíamos...

Simulação

$P =$ a b a b b a b a b b a

T

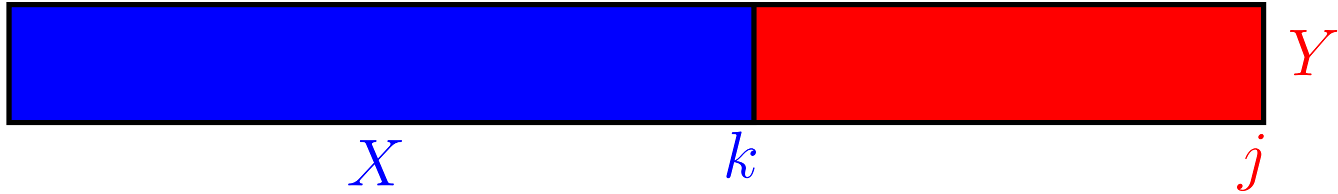
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	a	b	a	b																			
2		a																					
3			a	b																			
4				a	b	a	b	b															
5					a																		
6						a	b	a	b	b	a	b	a	b	b								
7							a																
8								a	b	a													
9									a														
10										a													
11											a	b	a	b									
12												a											
13													a	b	a	b	b	a	b	a	b	b	a

Prefixos e sulfixos

$X[1\,..\,k]$ é prefixo de $Y[1\,..\,j]$ se

$$k \leq j \quad \text{e} \quad X[1\,..\,k] = Y[1\,..\,k].$$

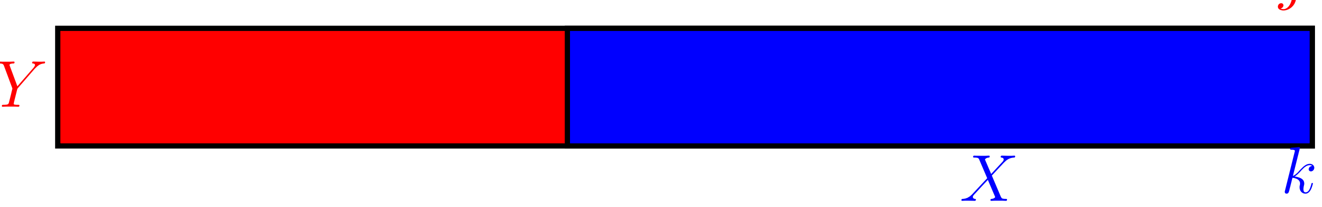
Se $k < j$, então $X[1\,..\,k]$ é prefixo próprio.



$X[1\,..\,k]$ é sulfixo de $Y[1\,..\,j]$ se

$$k \leq j \quad \text{e} \quad X[1\,..\,k] = Y[j-k+1\,..\,j].$$

Se $k < j$, então $X[1\,..\,k]$ é sulfixo próprio.



Exemplos

	1	2	3	4	5	6	7	8	9	10	11
P	a	b	a	b	b	a	b	a	b	b	a

$P[1\,..\,1]$ é prefixo próprio de $P[1\,..\,11]$

$P[1\,..\,3]$ é prefixo próprio de $P[1\,..\,7]$

$P[1\,..\,3]$ é sulfixo próprio de $P[1\,..\,8]$

$P[1\,..\,5]$ é sulfixo próprio de $P[1\,..\,10]$

$P[1\,..\,1]$ é sulfixo próprio de $P[1\,..\,11]$

Algoritmo do autômato finito

Versão grosseira:

FINITE-AUTOMATON-MATCHER (P, m, T, n)

1 $q \leftarrow 0$
2 **para** $i \leftarrow 1$ **até** n **faça**
3 $q \leftarrow$ maior k tal que $P[1\,..\,k]$ é sulfixo de $T[1\,..\,i]$
4 **se** $q = m$
5 **então** "P ocorre com deslocamento $i - m$"

Relação invariante: no final da linha 2 vale que:
q é o maior índice tal que

(i0) $P[1\,..\,q]$ é sulfixo de $T[1\,..\,i{-}1]$

O segredo está em saber executar a linha 3 eficientemente.

Simulação

$P =$ a b a b c a b a b b a

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

? ? T

8 a b a b c

18 a b a b c a b a b b

Subestrutura ótima

Suponha que:

- q é o maior índice tal que $P[1\,..\,q]$ é sulfixo de $T[1\,..\,i-1]$; e
- q' é o maior índice tal que $P[1\,..\,q']$ é sulfixo de $T[1\,..\,i] = T[1\,..\,i-1]a$.

Subestrutura ótima

Suponha que:

- q é o maior índice tal que $P[1\,..\,q]$ é sulfixo de $T[1\,..\,i-1]$; e
- q' é o maior índice tal que $P[1\,..\,q']$ é sulfixo de $T[1\,..\,i] = T[1\,..\,i-1]a$.

Portanto,

- $P[q'] = a$ e
- $P[1\,..\,q'-1]$ é prefixo de $P[1\,..\,q]$.

Logo, $P[1\,..\,q']$ é sulfixo de $P[1\,..\,q]a$.

Subestrutura ótima

Suponha que:

- q é o maior índice tal que $P[1 \,..\, q]$ é sulfixo de $T[1 \,..\, i-1]$; e
- q' é o maior índice tal que $P[1 \,..\, q']$ é sulfixo de $T[1 \,..\, i] = T[1 \,..\, i-1]a$.

Portanto,

- $P[q'] = a$ e
- $P[1 \,..\, q'-1]$ é prefixo de $P[1 \,..\, q]$.

Logo, $P[1 \,..\, q']$ é sulfixo de $P[1 \,..\, q]a$.

Conclusão: q' é o maior k tal que $P[1 \,..\, k]$ é sulfixo de $P[1 \,..\, q]a$

Subestrutura ótima

Suponha que:

- q é o maior índice tal que $P[1\,..\,q]$ é sulfixo de $T[1\,..\,i-1]$; e
- q' é o maior índice tal que $P[1\,..\,q']$ é sulfixo de $T[1\,..\,i] = T[1\,..\,i-1]a$.

Portanto,

- $P[q'] = a$ e
- $P[1\,..\,q'-1]$ é prefixo de $P[1\,..\,q]$.

Logo, $P[1\,..\,q']$ é sulfixo de $P[1\,..\,q]a$.

Conclusão: q' é o maior k tal que $P[1\,..\,k]$ é sulfixo de $P[1\,..\,q]a$

Outra conclusão: o valor de k depende apenas de $P[1\,..\,q]$ e do símbolo a.

Algoritmo do autômato finito

Versão melhorzinha:

FINITE-AUTOMATON-MATCHER (P, m, T, n)

1 $q \leftarrow 0$
2 **para** $i \leftarrow 1$ **até** n **faça**
3 $q' \leftarrow$ maior k tq $P[1\,..\,k]$ é sulfixo de $P[1\,..\,q]T[i]$
3 $q \leftarrow q'$
4 **se** $q = m$
5 **então** "P ocorre com deslocamento $i - m$"

Relação invariante: no final da linha 2 vale que:
q é o maior índice tal que

(i0) $P[1\,..\,q]$ é sulfixo de $T[1\,..\,i{-}1]$

Pré-processamento

O segredo está em saber executar a linha 3 eficiente.

O valor de q' na linha 3 depende apenas de $P[1\,..\,q]$ e do alfabeto!

Alfabeto $\Sigma =$ conjunto dos possíveis símbolos de P e T.

Um pré-processamento de $P[1\,..\,m]$ produz uma tabela δ (conhecida com autômato) definida da seguinte maneira: para cada q em $0\,..\,m$ e a em Σ

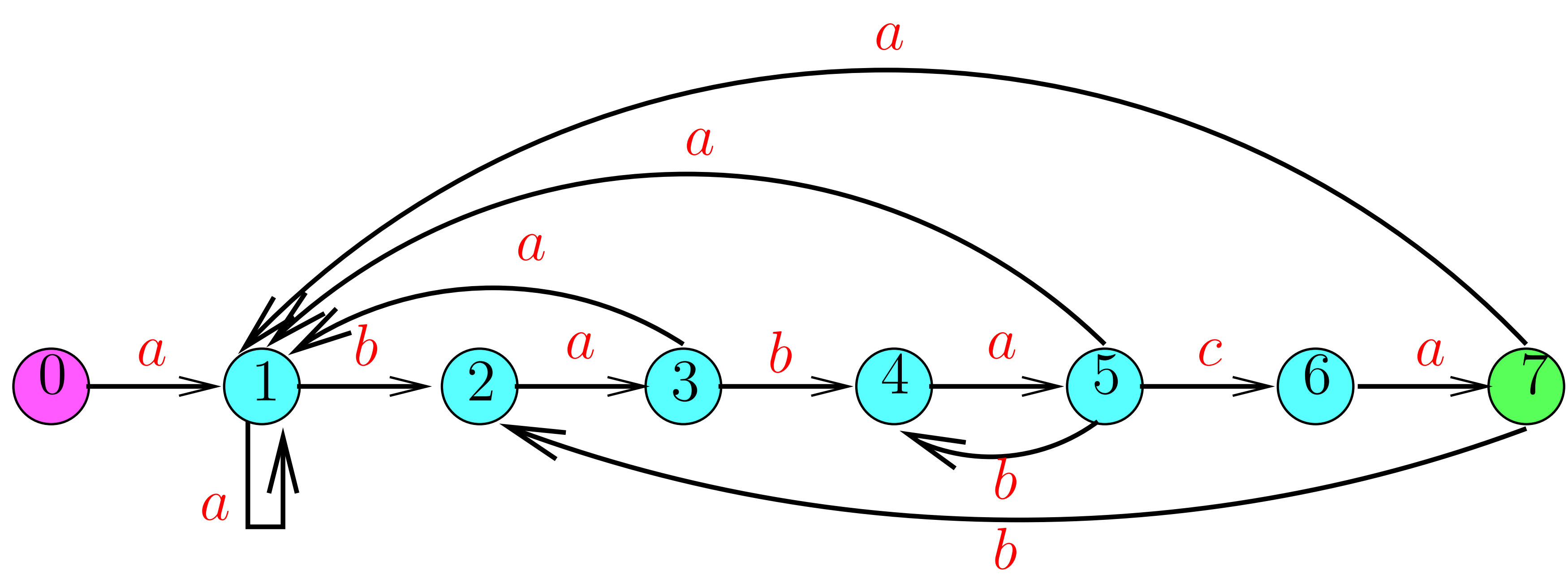
$$\delta[q,a] = \max\{k : P[1\,..\,k] \text{ é sulfixo de } P[1\,..\,q]a\}.$$

Autômato

$P =$ a b a b a c a $\Sigma = \{$a, b, c$\}$

q	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

Diagrama



$0..7 =$ conjunto de estados

$\Sigma = \{$a, b, c$\} =$ alfabeto

δ = função de transição

0 é estado incial e 7 é estado final

Algoritmo do autômato finito

FINITE-AUTOMATON-MATCHER (P, m, T, n)

0 $\delta \leftarrow$ COMPUTE-TRANSITION-FUNCTION(P, m, Σ)
1 $q \leftarrow 0$
2 **para** $i \leftarrow 1$ **até** n **faça**
3 $\quad q \leftarrow \delta[q, T[i]]$
4 $\quad$ **se** $q = m$
5 $\quad\quad$ **então** "P ocorre com deslocamento $i - m$"

Relação invariante: no final da linha 2 vale que:
q é o maior índice tal que

(i0) $P[1 \mathinner{..} q]$ é sulfixo de $T[1 \mathinner{..} i-1]$

Consumo de tempo das linhas 1-5 é $\Theta(n)$.

Simulação

$P =$ a b a b a c a

i		1	2	3	4	5	6	7	8	9	10	11
T		a	b	a	b	a	b	a	c	a	b	a
q	0	1	2	3	4	5	4	5	6	7	2	3

Pré-processamento

COMPUTE-TRANSITION-FUNCTION(P, m, Σ)

```
1   para q ← 0 até m
2       para cada a em Σ faça
3           k ← min{m + 1, q + 2}
4           repita
5               k ← k − 1
6           até que P[1 . . k] seja sulfixo de P[1 . . q]a
7           δ[q, a] ← k
8   devolva δ
```

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\Theta(m)$
2	$\Theta(m\|\Sigma\|)$
3	$\Theta(m\|\Sigma\|)$
4-5	$\Theta(m^2\|\Sigma\|)$
6	$\Theta(m^3\|\Sigma\|)$
7	$\Theta(m\|\Sigma\|)$
8	$\Theta(m\|\Sigma\|)$

total $\Theta(m + 3m|\Sigma| + m^2|\Sigma| + m^3|\Sigma|)$

$$= \Theta(m^3|\Sigma|)$$

Conclusões

O consumo de tempo do algoritmo COMPUTE-TRANSITION-FUNCTION é $\Theta(m^3|\Sigma|)$.

O consumo de tempo do algoritmo FINITE-AUTOMATON-MATCHER é $\Theta(n + m^3|\Sigma|)$.

Nova simulação

$P =$ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	T
1	a	b	a	b																				
4			a	b																				
4				a	b	a	b	b																
8						a	b	a	b	b	a	b	a	b	b									
15											a	b	a	b	b									
15													a	b	a	b	b	a	b	a	b	b	a	
i																								
	1	1	1	3	1	1	1	2	1	1	1	1	1	1	3	1	1	1	1	1	1	1	1	

Função prefixo

$\pi[q] :=$ maior comprimento de um prefixo próprio de $P[1\,.\,.\,q]$ que é sulfixo de $P[1\,.\,.\,q]$

$:= \max\{k : k < q$ e $P[1\,.\,.\,k]$ é sulfixo de $P[1\,.\,.\,q]\}$

q	1	2	3	4	5	6	7	8	9	10	11
P	a	b	a	b	b	a	b	a	b	b	a
π	0	0	1	2	0	1	2	3	4	5	6

q	1	2	3	4	5	6	7	8	9	10
P	a	b	a	b	a	b	a	b	c	a
π	0	0	1	2	3	4	5	6	0	1

KMP-Matcher

KMP-MATCHER (P, m, T, n)

```
0   π ← COMPUTE-PREFIX-FUNCTION(P, m)
1   i ← 1    q ← 0
2   enquanto i ≤ n + 1 faça
3       se q = m então ▷ caso 1
4           "P ocorre com deslocamento i − m"
5           q ← π[q]
6       senão se P[q+1] = T[i] então ▷ caso 2
7           q ← q + 1
8           i ← i + 1
9       senão se P[q+1] ≠ T[i] e q > 0 então ▷ caso 3
10          q ← π[q]
11      senão se P[q+1] ≠ T[i] e q = 0 então ▷ caso 4
12          i ← i + 1
```

Suponha $T[n+1]$ = "símbolo que não ocorre em $P[1\,..\,m]$"

Correção

Relação invariante: na linha 2 vale que

(i0) $P[1\,..\,q]$ é sufixo de $T[1\,..\,i{-}1]$

Na linha 2 vale ainda que

(i1) $P[1\,..\,k]$ não é sulfixo de $T[1\,..\,i]$, para $k \geq q+2$.

Invariante (i1) implica que

$$i-m, i-m+1, \ldots, i-q-1$$

não são deslocamentos válidos, ou seja

$$P[1\,..\,m] \neq T[\,..\,i+1], P[1\,..\,m] \neq T[\,..\,i+2], \ldots$$
$$P[1\,..\,m] \neq T[\,..\,i+m-q-1].$$

Consumo de tempo

Exceto a linha 0, cada linha do algoritmo consome tempo $\Theta(1)$.

Caso 2 e caso 4 ocorrem $n + 1$ vezes (total).

Para cada valor de i, número de ocorrências do caso 1 e caso 3 é $\leq m$.

Logo, o número total de iterações das linhas 2–12
é $\leq n + 1 + (n + 1)m = (n + 1)(m + 1)$.

Portanto, o consumo de tempo total das linhas 1–12
é $\mathrm{O}(nm)$.

Consumo de tempo

Exceto a linha 0, cada linha do algoritmo consome tempo $\Theta(1)$.

Caso 2 e caso 4 ocorrem $n+1$ vezes (total).

Para cada valor de i, número de ocorrências do caso 1 e caso 3 é $\leq m$.

Logo, o número total de iterações das linhas 2–12
é $\leq n+1+(n+1)m = (n+1)(m+1)$.

Portanto, o consumo de tempo total das linhas 1–12
é $O(nm)$.

EXAGERO!

Consumo de tempo

Exceto a linha 0, cada linha do algoritmo consome tempo $\Theta(1)$.

Caso 2 e caso 4 ocorrem $n + 1$ vezes (total).

O número de ocorrências do caso 1 e caso 3 é $\leq$ número de ocorrências do caso 2, pois

- q nunca é negativo;
- q é incrementado no caso 2;
- q é decrementado no caso 1 e no caso 3.

Logo, o número total de iterações é $\leq n + 1 + n + 1 = 2n + 2$.

Portanto, o consumo de tempo das linhas 1–12 é $\Theta(n)$.

KMP-Matcher

KMP-MATCHER (P, m, T, n)

0 $\pi \leftarrow$ COMPUTE-PREFIX-FUNCTION(P, m)
1 $i \leftarrow 1 \quad q \leftarrow 0$
2 **enquanto** $i \leq n$ **faça**
3 **enquanto** $q > 0$ **e** $P[q{+}1] \neq T[i]$ **faça**
4 $q \leftarrow \pi[q]$
5 **se** $P[q{+}1] = T[i]$ **então**
6 $q \leftarrow q + 1$
7 **se** $q = m$ **então**
8 "P ocorre com deslocamento $i - m$"
9 $q \leftarrow \pi[q]$

Compute-Prefix-Function (grosseiro)

COMPUTE-PREFIX-FUNCTION (P, m)

```
0    π[1] ← 0
1    q ← 2
2    enquanto q ≤ m faça
3        k ← q − 1
4        repita
5            i ← k
6            j ← q
7            enquanto i > 0 e P[i] = P[j] faça
8                i ← i − 1    j ← j − 1
9            se i > 0
10               então k ← k − 1
11       até que i = 0
12       π[q] ← k
13   devolva π
```

Consumo de tempo

linha	consumo de todas as execuções da linha
0-1	$\Theta(1)$
2	$\Theta(m)$
3	$\Theta(m-1)$
4-6	$\mathrm{O}(m^2)$
7-8	$\mathrm{O}(m^3)$
9-11	$\mathrm{O}(m^2)$
12	$\Theta(m-1)$
13	$\mathrm{O}(m)$

total $\Theta(3m-1) + \mathrm{O}(m^3 + 2m^2 + m)$

$= \mathrm{O}(m^3)$

Subestrutura ótima

Suponha que $P[1\,..\,k+1]$ é o maior prefixo próprio de $P[1\,..\,q]$ que é sulfixo de $P[1\,..\,q]$ ($\pi[q] = k+1$).

Subestrutura ótima

Suponha que $P[1\,..\,k+1]$ é o maior prefixo próprio de $P[1\,..\,q]$ que é sulfixo de $P[1\,..\,q]$ ($\pi[q] = k+1$).

Portanto,

- $P[k+1] = P[q]$ e
- $P[1\,..\,k]$ é prefixo próprio e sulfixo de $P[1\,..\,q-1]$.

Logo, $P[1\,..\,k]$ é prefixo e sulfixo de $P[1\,..\,\pi[q-1]]$.

Subestrutura ótima

Suponha que $P[1\,..\,k+1]$ é o maior prefixo próprio de $P[1\,..\,q]$ que é sulfixo de $P[1\,..\,q]$ ($\pi[q] = k+1$).

Portanto,

- $P[k+1] = P[q]$ e
- $P[1\,..\,k]$ é prefixo próprio e sulfixo de $P[1\,..\,q-1]$.

Logo, $P[1\,..\,k]$ é prefixo e sulfixo de $P[1\,..\,\pi[q-1]]$.

Se $k \neq \pi[q-1]$, então

Subestrutura ótima

Suponha que $P[1\,..\,k+1]$ é o maior prefixo próprio de $P[1\,..\,q]$ que é sulfixo de $P[1\,..\,q]$ ($\pi[q] = k+1$).

Portanto,

- $P[k+1] = P[q]$ e
- $P[1\,..\,k]$ é prefixo próprio e sulfixo de $P[1\,..\,q-1]$.

Logo, $P[1\,..\,k]$ é prefixo e sulfixo de $P[1\,..\,\pi[q-1]]$.

Se $k \neq \pi[q-1]$, então

$P[1\,..\,k]$ é prefixo próprio e sulfixo de $P[1\,..\,\pi[q-1]]$.

Logo, $P[1\,..\,k]$ é prefixo e sulfixo de

$$P[1\,..\,\pi[\pi[q-1]]] = P[1\,..\,\pi^2[q-1]].$$

Subestrutura ótima (cont.)

Se $k \neq \pi^2[q-1]$, então

Subestrutura ótima (cont.)

Se $k \neq \pi^2[q-1]$, então

$$P[1\,..\,k] \text{ é prefixo próprio e sulfixo de } P[1\,..\,\pi^2[q-1]].$$

Logo, $P[1\,..\,k]$ é sulfixo de

$$P[1\,..\,\pi[\pi[\pi[q-1]]]] = P[1\,..\,\pi^3[q-1]].$$

Subestrutura ótima (cont.)

Se $k \neq \pi^2[q-1]$, então

$P[1\,..\,k]$ é prefixo próprio e sulfixo de $P[1\,..\,\pi^2[q-1]]$.

Logo, $P[1\,..\,k]$ é sulfixo de

$$P[1\,..\,\pi[\pi[\pi[q-1]]]] = P[1\,..\,\pi^3[q-1]].$$

Se $k \neq \pi^3[q-1]$, então

$P[1\,..\,k-1]$ é prefixo próprio e sulfixo
de $P[1\,..\,\pi^3[q-1]]$.

Logo, $P[1\,..\,k]$ é prefixo e sulfixo de $P[1\,..\,\pi^4[q-1]]$.

Se $k \neq \pi^4[q-1]$, então ...

Recorrência

$\pi[q] :=$ maior comprimento de um prefixo próprio de $P[1\,.\,.\,q]$ que é sulfixo de $P[1\,.\,.\,q]$

$:= \max\{k : k < q$ e $P[1\,.\,.\,k]$ é sulfixo de $P[1\,.\,.\,q]\}$

q	0	1	2	3	4	5	6	7	8	9	10	11
P	'?'	a	b	a	b	b	a	b	a	b	b	a
π	-1	0	0	1	2	0	1	2	3	4	5	6

$\pi[0] = -1$

$\pi[1] = 0$

$\pi[q] = \max\{\pi^j[q-1]+1 : P[\pi^j[q-1]+1] = P[q]\}$

Suponha $P[0] =$ "coringa".

Compute-Prefix-Function (grosseiro)

```
COMPUTE-PREFIX-FUNCTION (P, m)
 0   π[1] ← 0
 1   q ← 2
 2   enquanto q ≤ m faça
 3       k ← q − 1
 4       repita
 5           i ← k
 6           j ← q
 7           enquanto i > 0 e P[i] = P[j] faça
 8               i ← i − 1    j ← j − 1
 9           se i > 0
10               então k ← k − 1
11       até que i = 0
12       π[q] ← k
13   devolva π
```

Compute-Prefix-Function (melhorado)

COMPUTE-PREFIX-FUNCTION (P, m)

```
0   π[1] ← 0
1   q ← 2
2   enquanto q ≤ m faça
3         k ← π[q − 1]
4         repita
5               i ← k + 1
6               j ← q
7               enquanto i > 0 e P[i + 1] = P[j] faça
8                     i ← i − 1    j ← j − 1
9               se i > 0
10                    então k ← π[k]
11        até que i = 0
12        π[q] ← k + 1
13  devolva π
```

Compute-Prefix-Function (melhor ainda)

COMPUTE-PREFIX-FUNCTION (P, m)

```
0    π[1] ← 0
1    q ← 2
2    enquanto q ≤ m faça
3         k ← π[q − 1]
4         repita
5              i ← k + 1
6              j ← q
7              se P[i] = P[j]
8                   então i ← 0
10                  senão k ← π[k]
11        até que i = 0
12        π[q] ← k + 1
13   devolva π
```

Compute-Prefix-Function (limpo)

COMPUTE-PREFIX-FUNCTION (P, m)

```
0  π[1] ← 0
1  q ← 2    k ← 0
2  enquanto q ≤ m faça
3        enquanto k > 0 e P[k+1] ≠ P[q] faça
4              k ← π[k]
5        se P[k+1] = P[q]
6              então k ← k+1
7        π[q] ← k
8  devolva π
```

Compute-Prefix-Function (limpo)

COMPUTE-PREFIX-FUNCTION (P, m)

```
0   π[1] ← 0
1   q ← 2   k ← 0
2   enquanto q ≤ m faça
3       se P[k+1] = P[q] então ▷ caso 1
4           π[q] ← k + 1
5           q ← q + 1
6           k ← k + 1
7       senão se P[k+1] ≠ P[q] e k > 0 então ▷ caso 2
8           k ← π[k]
9       senão se P[k+1] ≠ P[q] e k = 0 então ▷ caso 3
10          π[q] ← 0
11          q ← q + 1
12  devolva π
```

Correção

Relações invariantes: na linha 2 vale que

(i0) $P[1\,..\,k]$ é sulfixo de $P[1\,..\,q{-}1]$

(i1) $P[1\,..\,j]$ não é sulfixo de $P[1\,..\,q]$, para $j > k+1$.

Consumo de tempo

Cada linha do algoritmo consome tempo $\Theta(1)$, exceto a linha 12 que pode consumir tempo $\mathrm{O}(m)$.

Caso 1 e caso 3 ocorrem m vezes (total).

O número de ocorrências do caso 2 é $\leq$ número de ocorrências do caso 1, pois

- k nunca é negativo;
- k é incrementado no caso 1; e
- k é decrementado no caso 2.

Logo, o número total de iterações é $\leq m + m = 2m$.

O consumo de tempo do algoritmo é $\Theta(m)$.

Conclusões

O consumo de tempo do algoritmo COMPUTE-PREFIX-FUNCTION é $\Theta(m)$.

O consumo de tempo do algoritmo KMP-MATCHER é $\Theta(n + m)$.

AULA 22

Conjuntos disjuntos dinâmicos

CLR 22 CLRS 21

Conjuntos disjuntos

Seja $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo $i \neq j$.

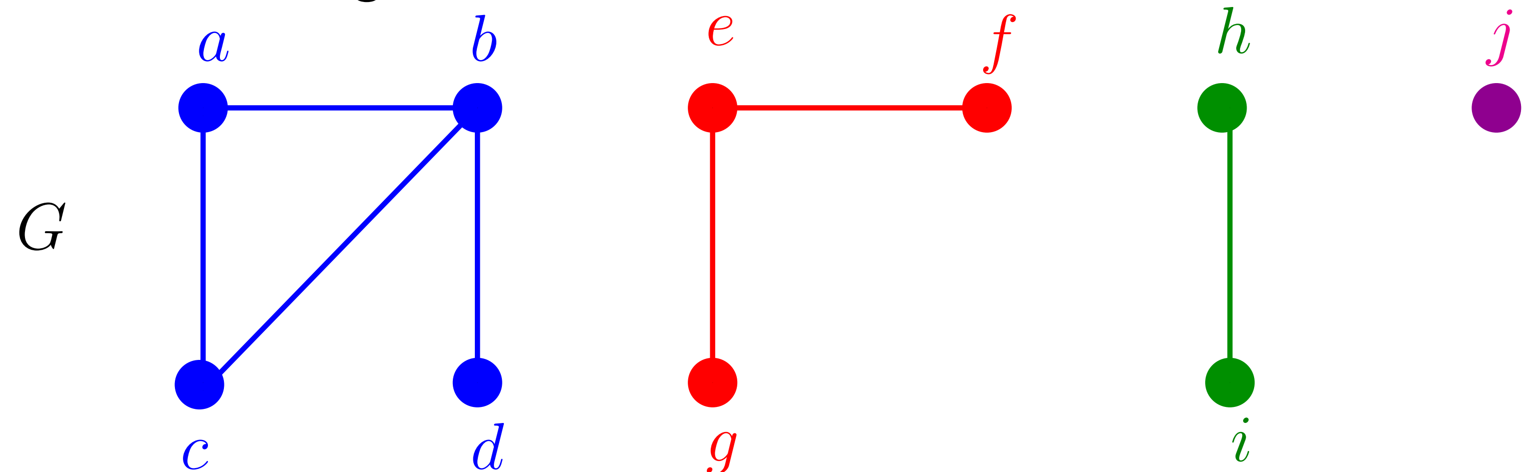
Conjuntos disjuntos

Seja $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo $i \neq j$.

Exemplo de coleção disjunta de conjuntos: componentes conexos de um grafo



componentes = conjuntos disjuntos de vértices

$\{a, b, c, d\}$ $\{e, f, g\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

a b e f h j

G

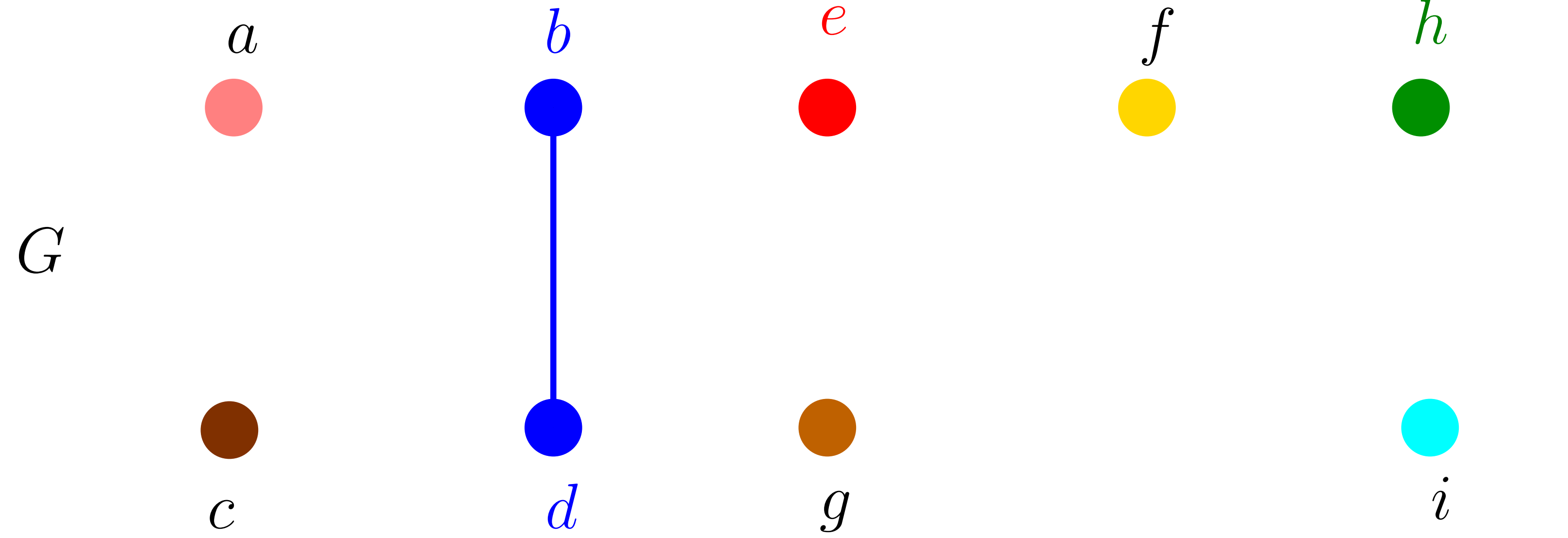
c d g i

aresta	componentes
	$\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

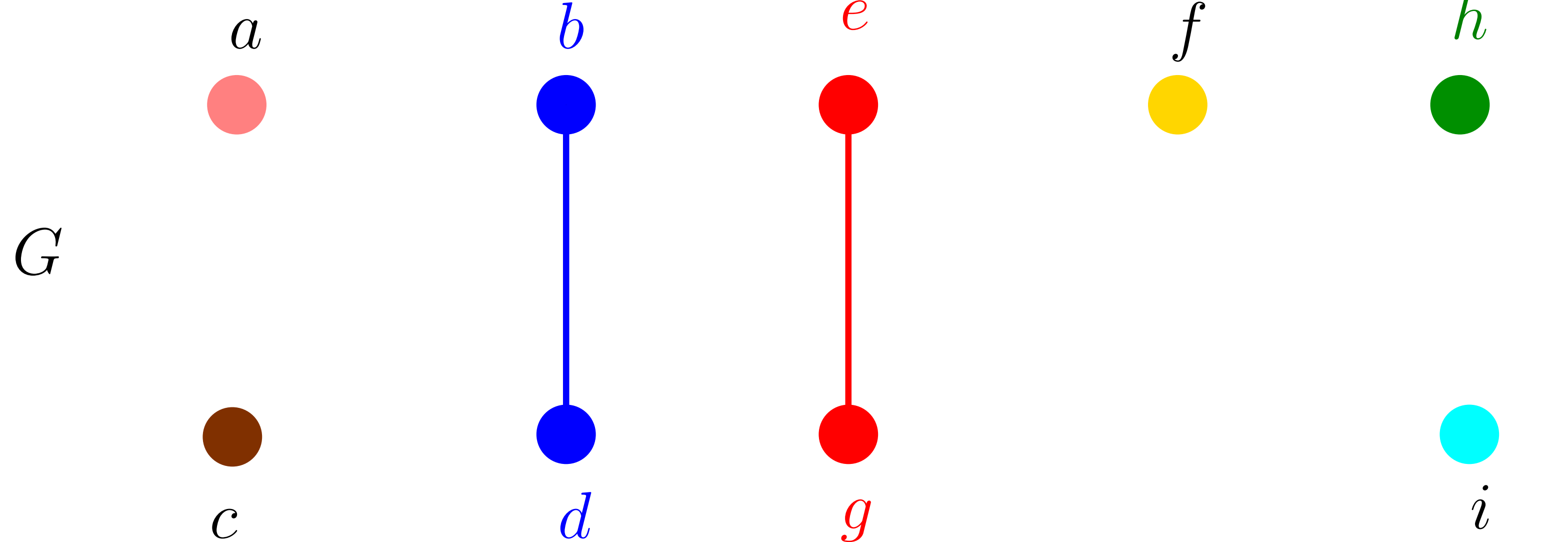


aresta	componentes
(b, d)	$\{a\}$ $\{b, d\}$ $\{c\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

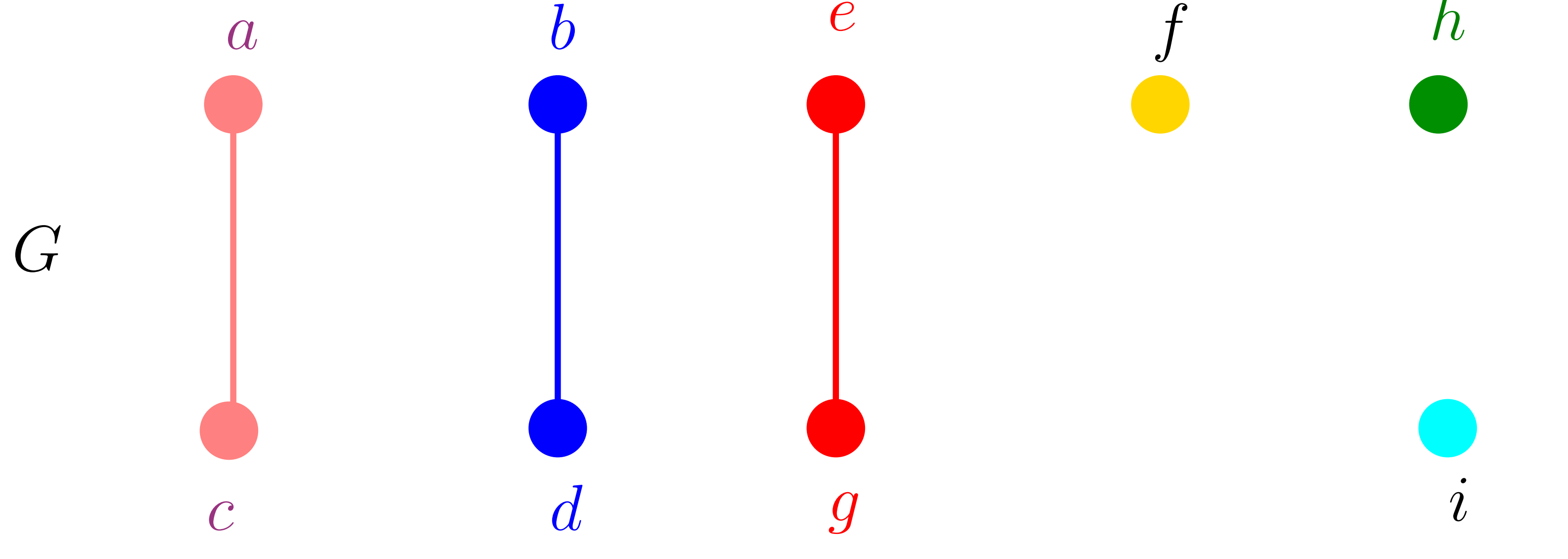


aresta	componentes
(e, g)	$\{a\}$ $\{b, d\}$ $\{c\}$ $\{e, g\}$ $\{f\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

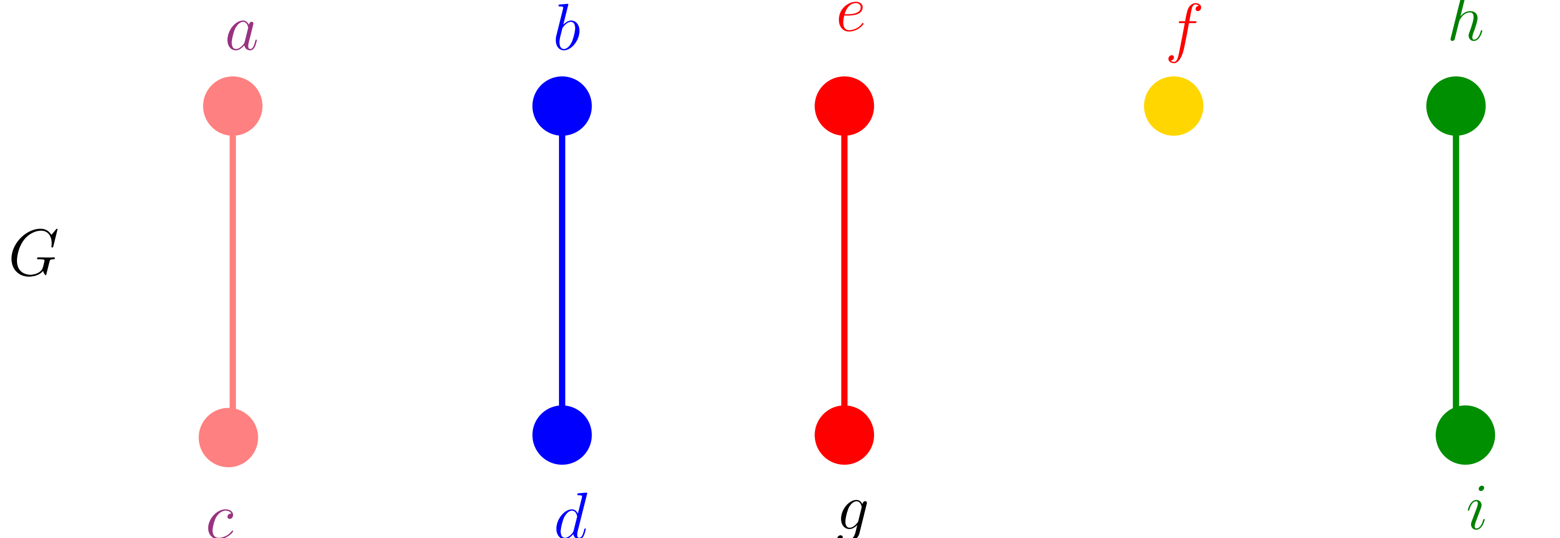


aresta	componentes
(a, c)	$\{a, c\}$ $\{b, d\}$ $\{e, g\}$ $\{f\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

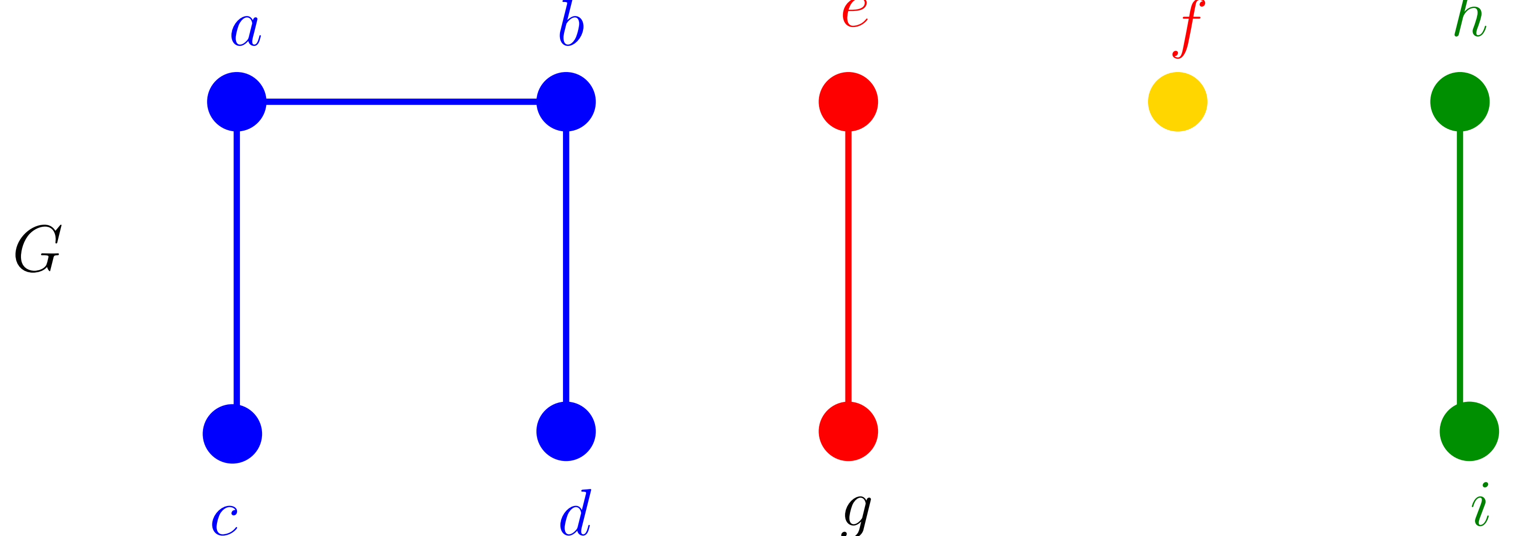


aresta	componentes
(h,i)	$\{a,c\}$ $\{b,d\}$ $\{e,g\}$ $\{f\}$ $\{h,i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

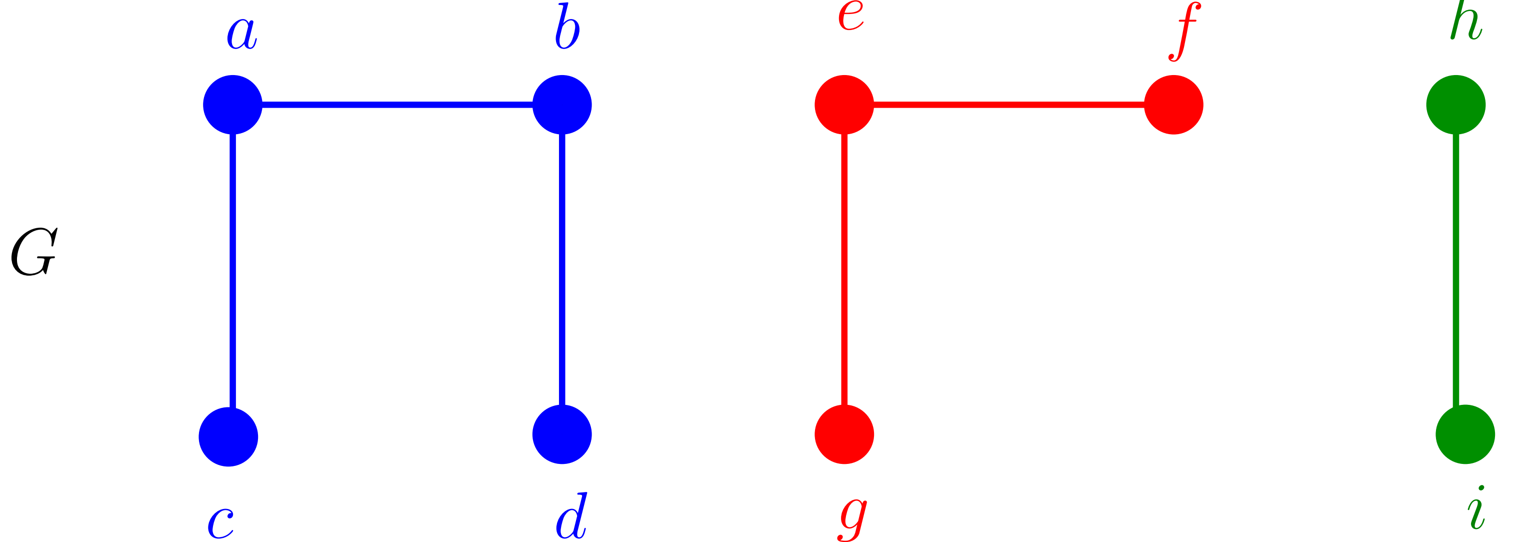


aresta	componentes
(a, b)	$\{a, b, c, d\}$ $\{e, g\}$ $\{f\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

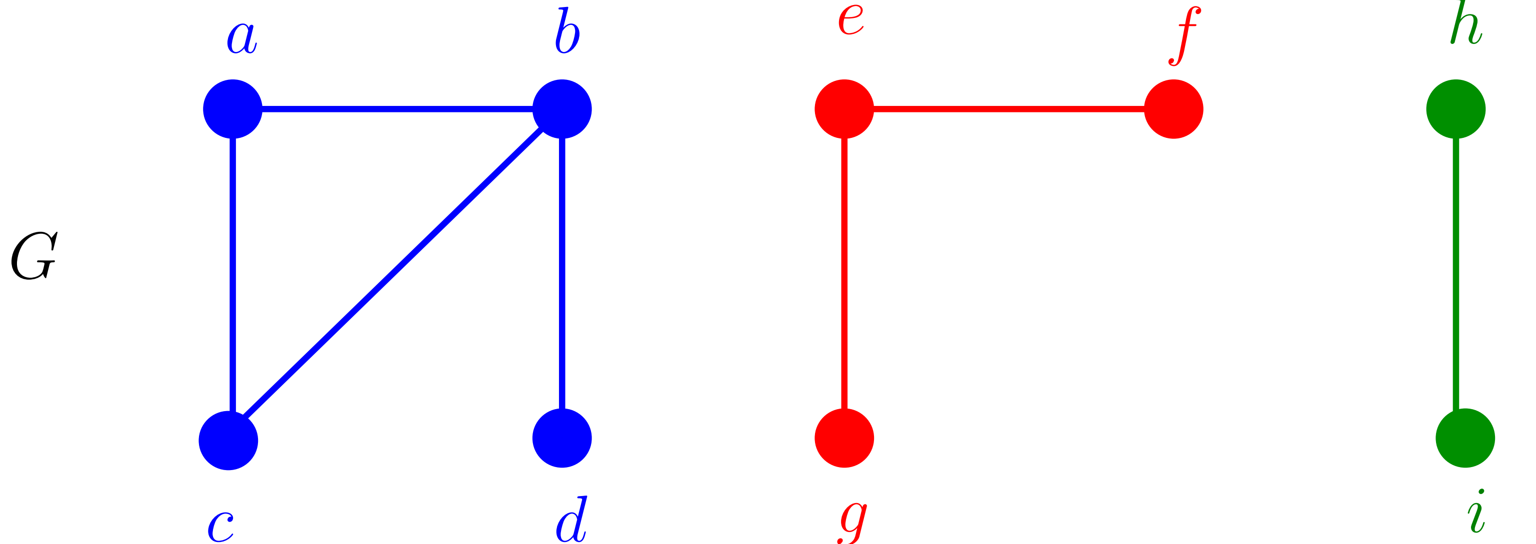


aresta	componentes
(e, f)	$\{a, b, c, d\}$ $\{e, f, g\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta	componentes
(b, c)	$\{a, b, c, d\}$ $\{e, f, g\}$ $\{h, i\}$ $\{j\}$

Operações básicas

$\mathcal{S}$ coleção de conjuntos disjuntos.

Cada conjunto tem um representante.

MAKESET (x): x é elemento novo
$\mathcal{S} \leftarrow \mathcal{S} \cup \{\{x\}\}$

UNION (x, y): x e y em conjuntos diferentes
$\mathcal{S} \leftarrow \mathcal{S} - \{S_x, S_y\} \cup \{S_x \cup S_y\}$
x está em S_x e y está em Sy

FINDSET (x): devolve representante do conjunto que contém x

Connected-Componets

Recebe um grafo G e contrói uma representação dos componentes conexos.

CONNECTED-COMPONENTS (G)

1 **para cada** vértice v de G **faça**
2 MAKESET (v)
3 **para cada** aresta (u, v) de G **faça**
4 **se** FINDSET $(u) \neq$ FINDSET (v)
5 **então** UNION (u, v)

Consumo de tempo

$n :=$ número de vértices do grafo
$m :=$ número de arestas do grafo

linha	consumo de todas as execuções da linha
1	$= \Theta(n)$
2	$= n \times$ consumo de tempo MAKESET
3	$= \Theta(m)$
4	$= 2m \times$ consumo de tempo FINDSET
5	$\leq n \times$ consumo de tempo UNION

total $\leq \Theta(n + m) + n \times$ consumo de tempo MAKESET
$+2m \times$ consumo de tempo FINDSET
$+n \times$ consumo de tempo UNION

Same-Component

Decide se u e v estão no mesmo componente:

SAME-COMPONENT (u, v)

1 **se** FINDSET $(u) =$ FINDSET (v)
2 **então devolva** SIM
3 **senão devolva** NÃO

Algoritmo de Kruskal

Encontra uma árvore geradora mínima (CLRS 23).

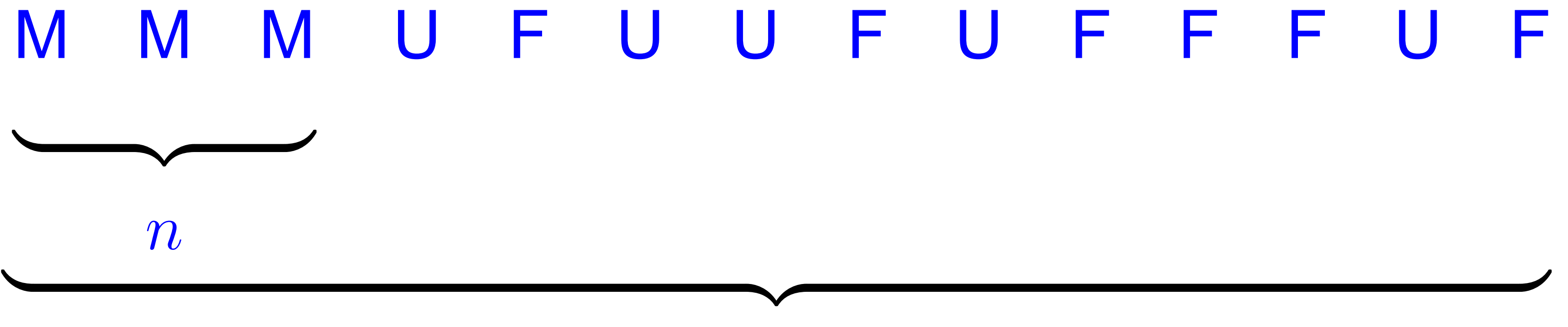
MST-KRUSKAL (G, w) $\triangleright$ G conexo

-1 coloque arestas em ordem crescente de w
0 $A \leftarrow \emptyset$
1 **para cada** vértice v **faça**
2 MAKESET (v)
3 **para cada** aresta uv em ordem crescente de w **faça**
4 **se** FINDSET $(u) \neq$ FINDSET (v)
5 **então** UNION (u, v)
8 $A \leftarrow A \cup \{uv\}$
9 **devolva** A

“Avô” de todos os algoritmos gulosos.

Conjuntos disjuntos dinâmicos

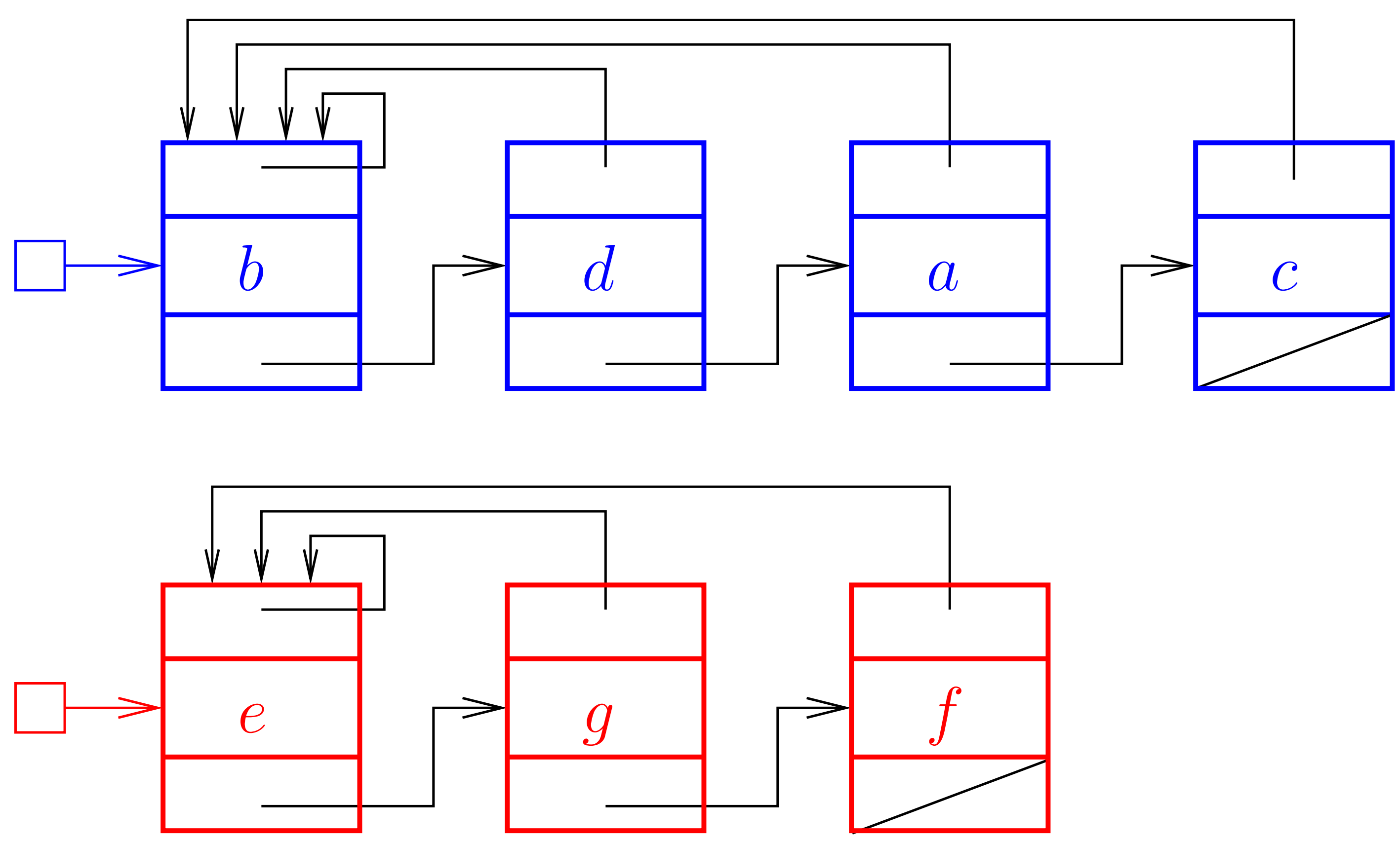
Seqüência de operações MAKESET, UNION, FINDSET



Que estrutura de dados usar?

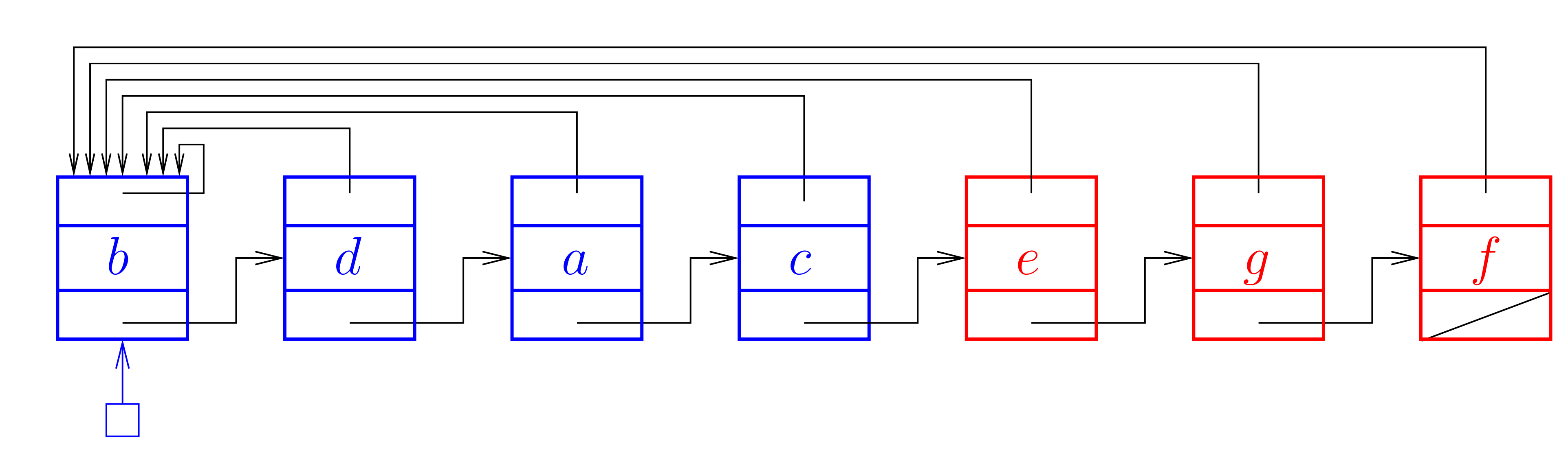
Compromissos (*trade-offs*)

Estrutura de listas ligadas



- cada conjunto tem um representante (início da lista)
- cada nó x tem um campo $repr$
- $repr[x]$ é o representante do conjunto que contém x

Estrutura de listas ligadas



UNION (a, e) : atualiza apontador para o representante.

Consumo de tempo

Operação	número de objetos atualizados
MAKESET(x_1)	1
MAKESET(x_2)	1
$\vdots$	1
MAKESET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
$\vdots$	$\vdots$
UNION(x_{n-1}, x_n)	$n - 1$
total	$= \Theta(n^2) = \Theta(m^2)$

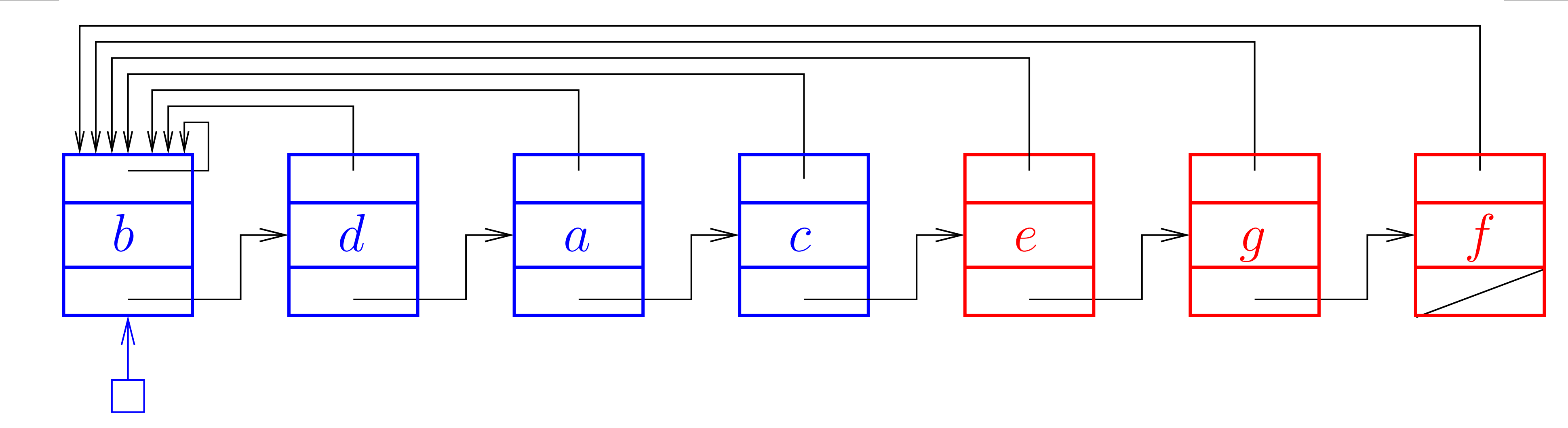
Consumo de tempo

MAKESET	$\Theta(1)$
UNION	$\mathrm{O}(n)$
FINDSET	$\Theta(1)$

Uma seqüência de m operações pode consumir tempo $\Theta(m^2)$ no pior caso.

Consumo de tempo amortizado de cada operação é $\mathrm{O}(m)$.

Melhoramento: *weighted-union*



- cada representante armazena o comprimento da lista
- a lista menor é concatenada com a maior

Cada objeto x é atualizado $\leq \lg n$:

cada vez que x é atualizado o tamanho da lista dobra.

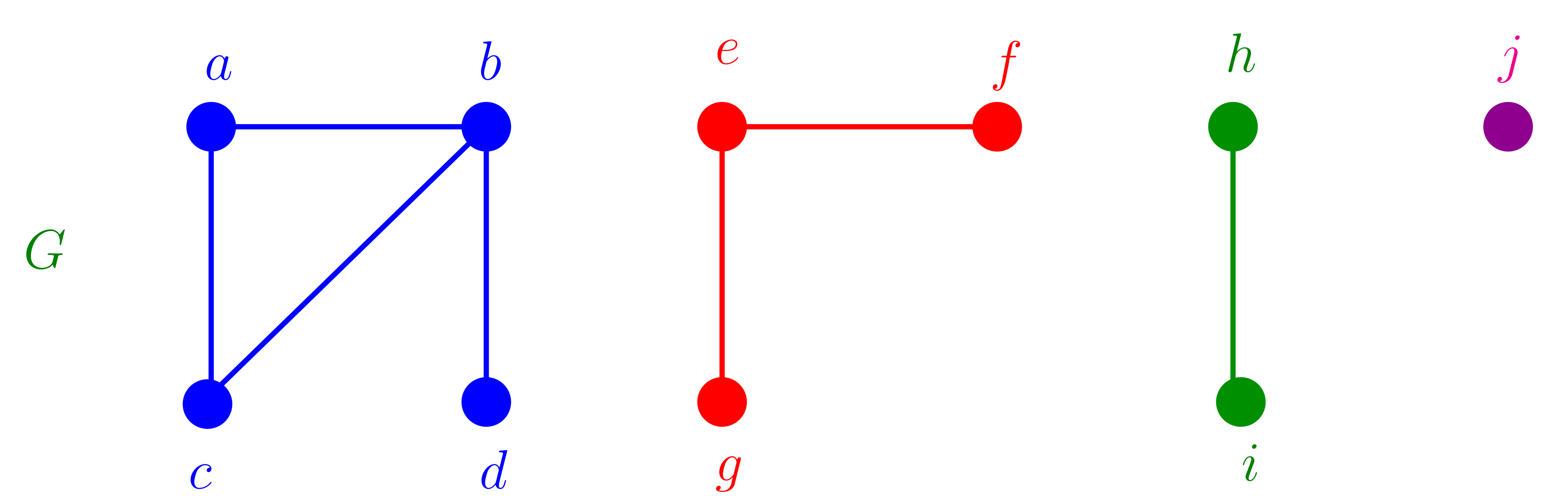
Conclusões

Se conjuntos disjuntos são representados através de listas ligadas e *weighted-union* é utilizada, então uma seqüência de m operações MAKESET, UNION e FINDSET, sendo que n são MAKESET, consome tempo $O(m + n \lg n)$.

Se conjuntos disjuntos são representados através de listas ligadas e *weighted-union* é utilizada, então o algoritmos CONNECTED-COMPONENTS consome tempo $O(m + n \lg n)$ e o algoritmo MST-KRUSKAL consome tempo $O((n + m) \lg n)$.

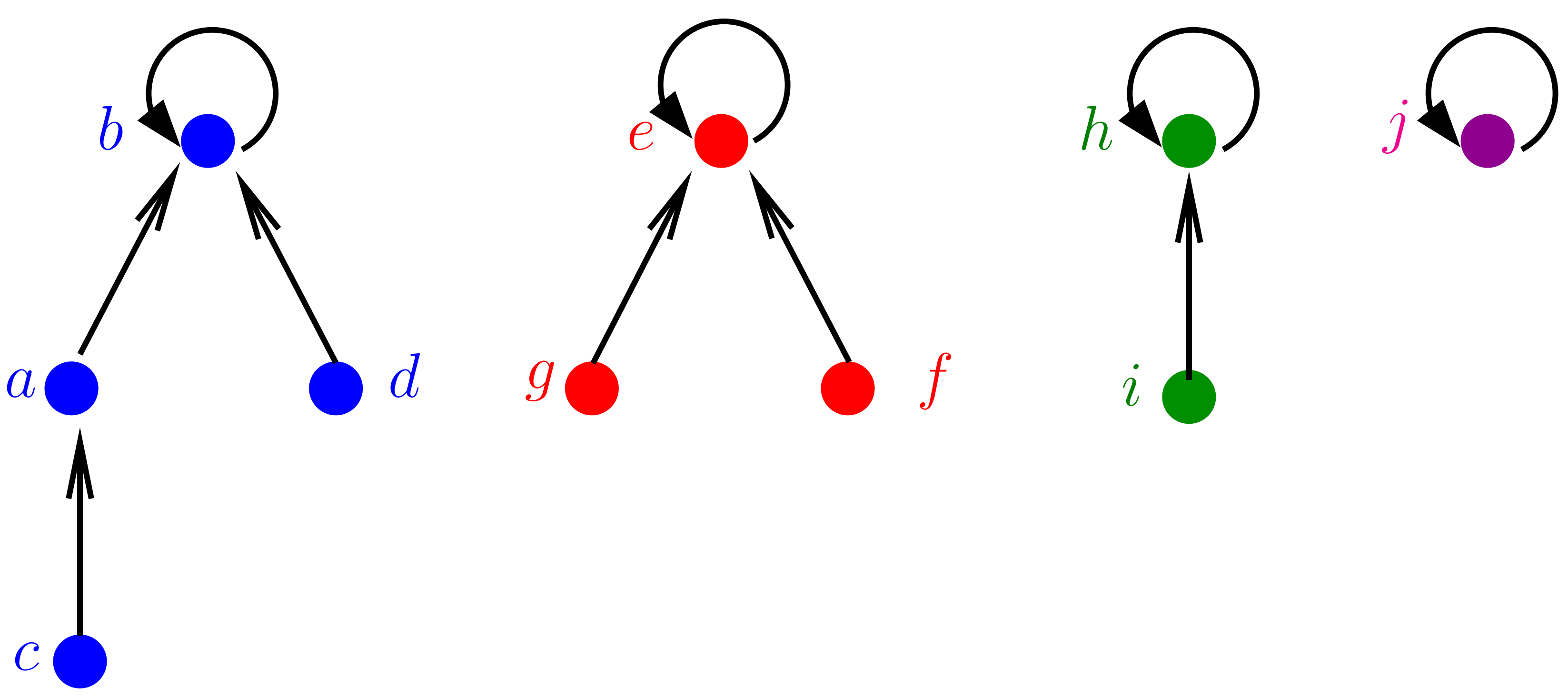
No que se refere ao algoritmo MST-KRUSKAL, estamos supondo que $m = O(n^2)$.

Estrutura *disjoint-set forest*



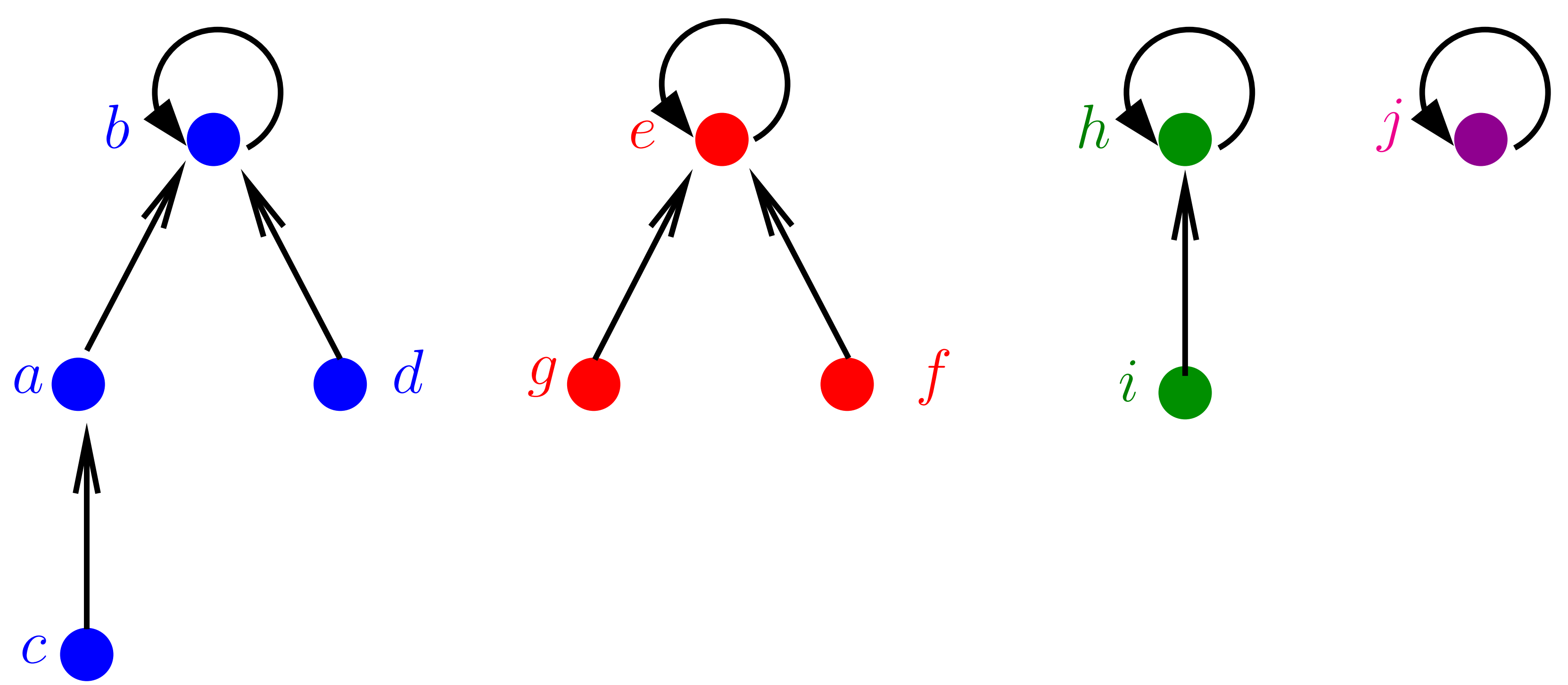
- cada conjunto tem uma *raiz*, que é o seu representate
- cada nó x tem um *pai*
- $pai[x] = x$ se e só se x é uma raiz

Estrutura *disjoint-set forest*



- cada conjunto tem uma $raiz$
- cada nó x tem um pai
- $pai[x] = x$ se e só se x é uma raiz

MakeSet$_0$ e FindSet$_0$



MAKESET$_0$ (x)

1 $\quad pai[x] \leftarrow x$

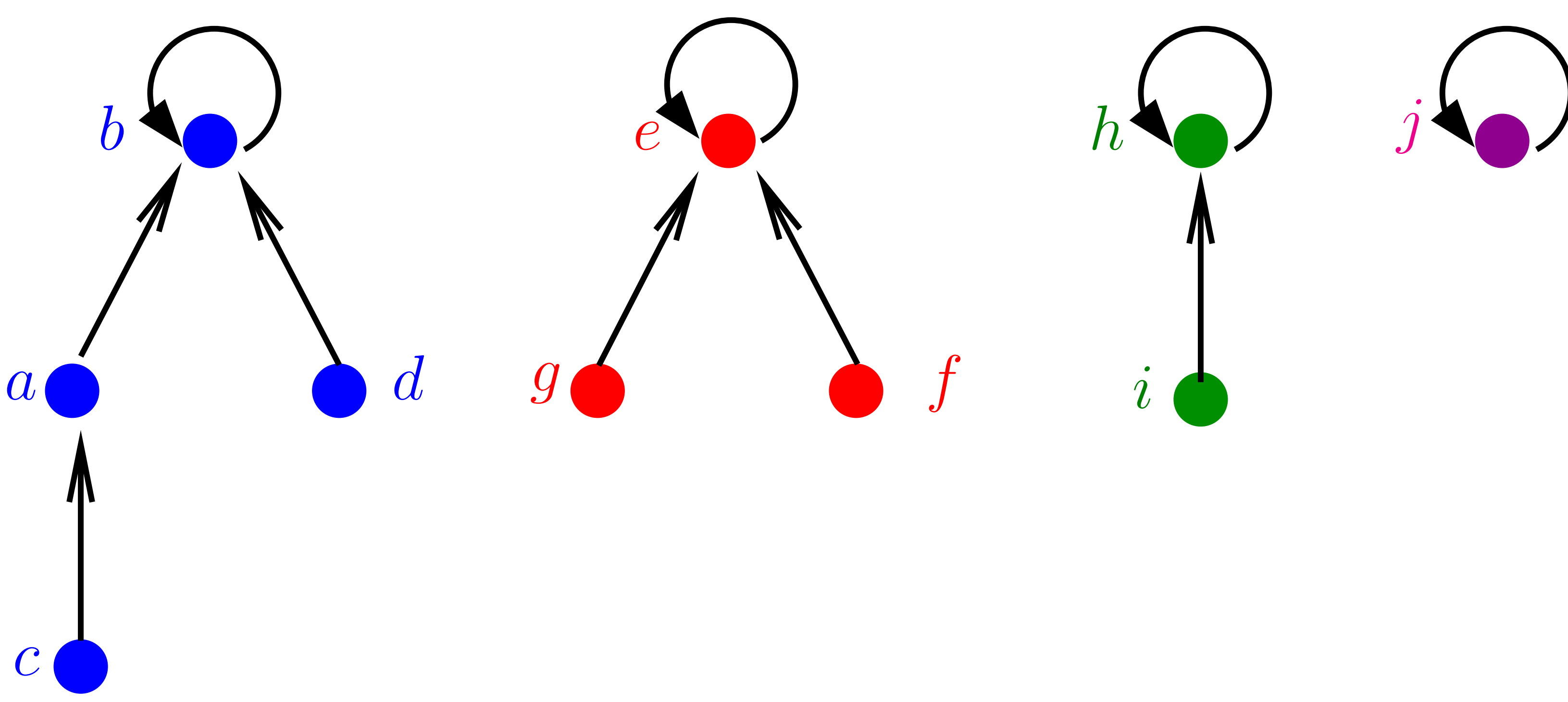
FINDSET$_0$ (x)

1 **enquanto** $pai[x] \neq x$ **faça**

2 $\quad\quad x \leftarrow pai[x]$

3 **devolva** x

FindSet$_1$



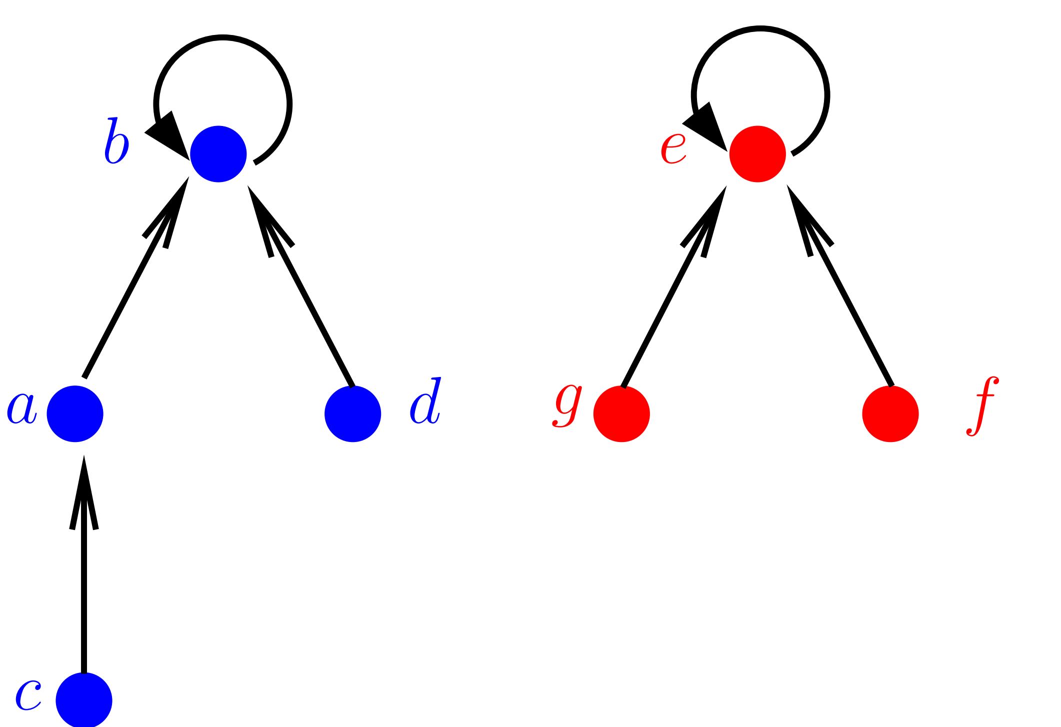
FINDSET$_1$ (x)

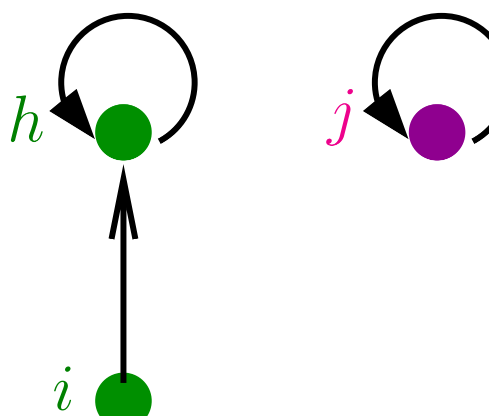
1 **se** $pai[x] = x$

2 **então devolva** x

3 **senão devolva** FINDSET$_1$ $(pai[x])$

Union$_0$





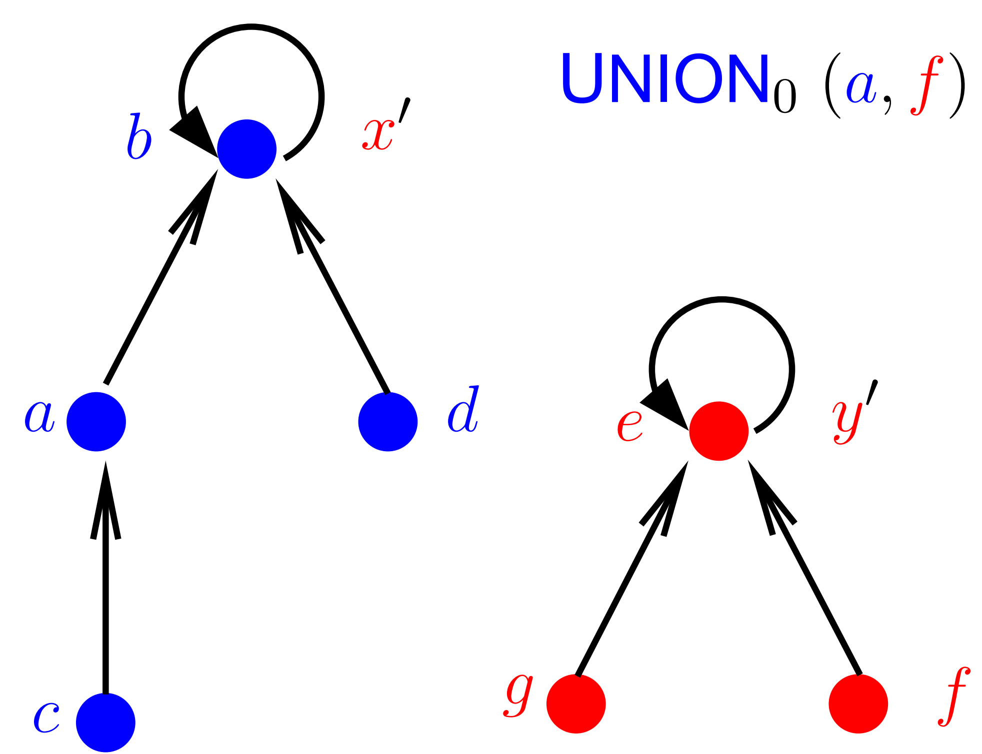
UNION$_0$ (x, y)

1 $x' \leftarrow$ FINDSET$_0$ (x)

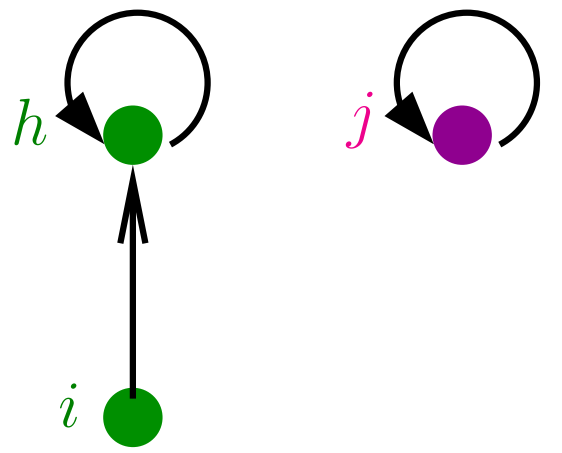
2 $y' \leftarrow$ FINDSET$_0$ (y)

3 $pai[y'] \leftarrow x'$

Union$_0$



UNION$_0$ (a, f)



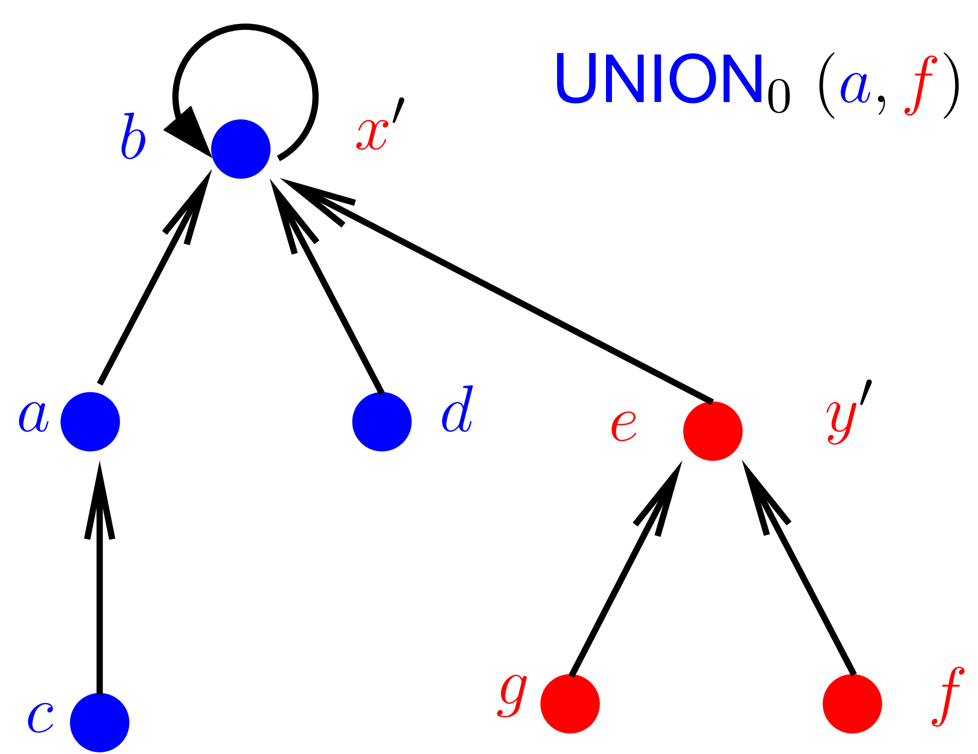
UNION$_0$ (x, y)

1 $x' \leftarrow$ FINDSET$_0$ (x)

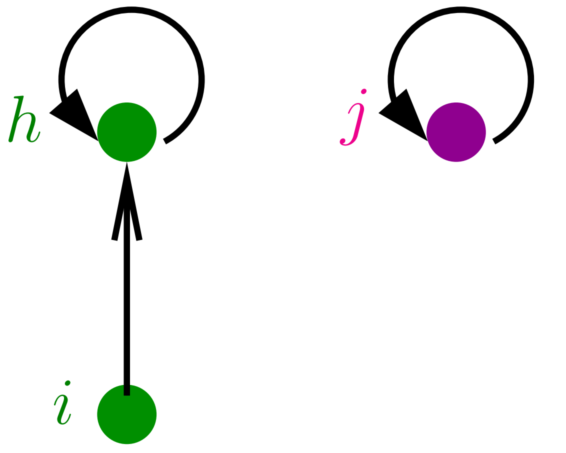
2 $y' \leftarrow$ FINDSET$_0$ (y)

3 $pai[y'] \leftarrow x'$

Union$_0$



UNION$_0$ (a, f)



UNION$_0$ (x, y)

1 $x' \leftarrow$ FINDSET$_0$ (x)

2 $y' \leftarrow$ FINDSET$_0$ (y)

3 $pai[y'] \leftarrow x'$

MakeSet$_0$, Union$_0$ e FindSet$_1$

MAKESET$_0$ (x)

1 $\quad pai[x] \leftarrow x$

UNION$_0$ (x, y)

1 $\quad x' \leftarrow$ FINDSET$_0$ (x)

2 $\quad y' \leftarrow$ FINDSET$_0$ (y)

3 $\quad pai[y'] \leftarrow x'$

FINDSET$_1$ (x)

1 $\quad$ **se** $pai[x] = x$

2 $\qquad$ **então devolva** x

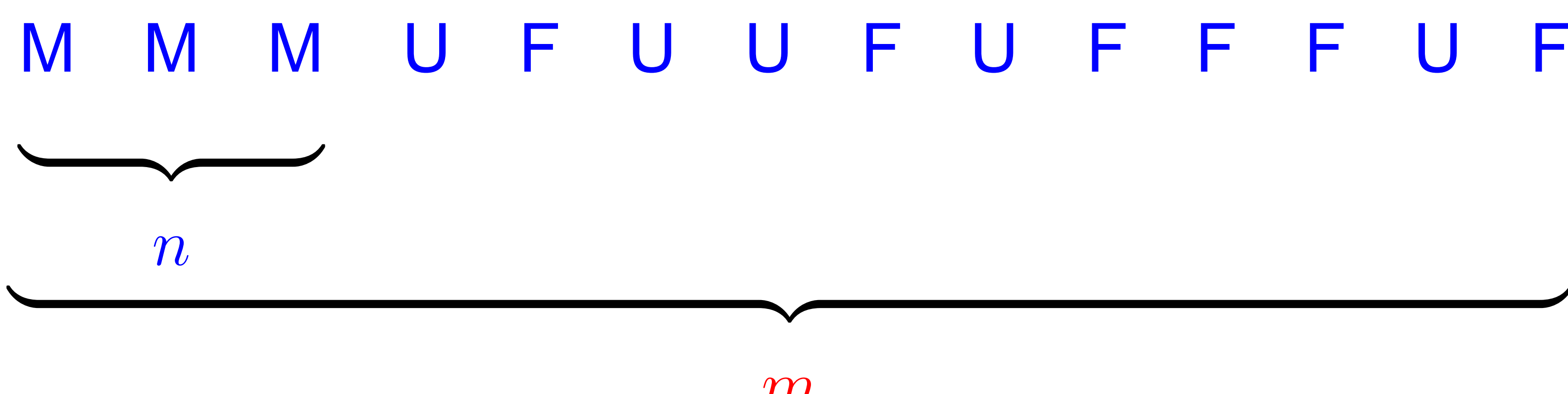
3 $\qquad$ **senão devolva** FINDSET$_1$ $(pai[x])$

Consumo de tempo

MAKESET$_0$ $\Theta(1)$

UNION$_0$ $O(n)$

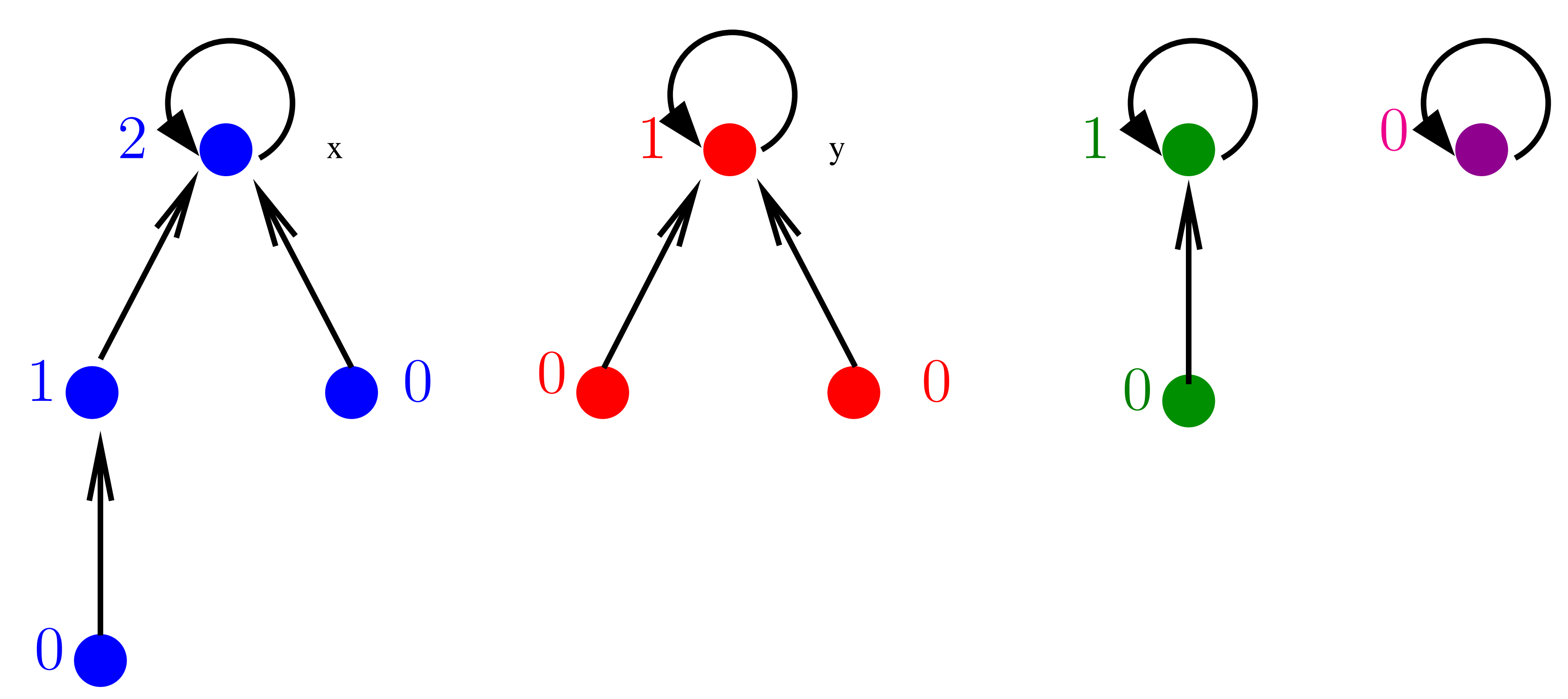
FINDSET$_0$ $O(n)$



Custo total da seqüência:

$$n\,\Theta(1) + n\,O(n) + m\,O(n) = O(mn)$$

Melhoramento 1: *union by rank*



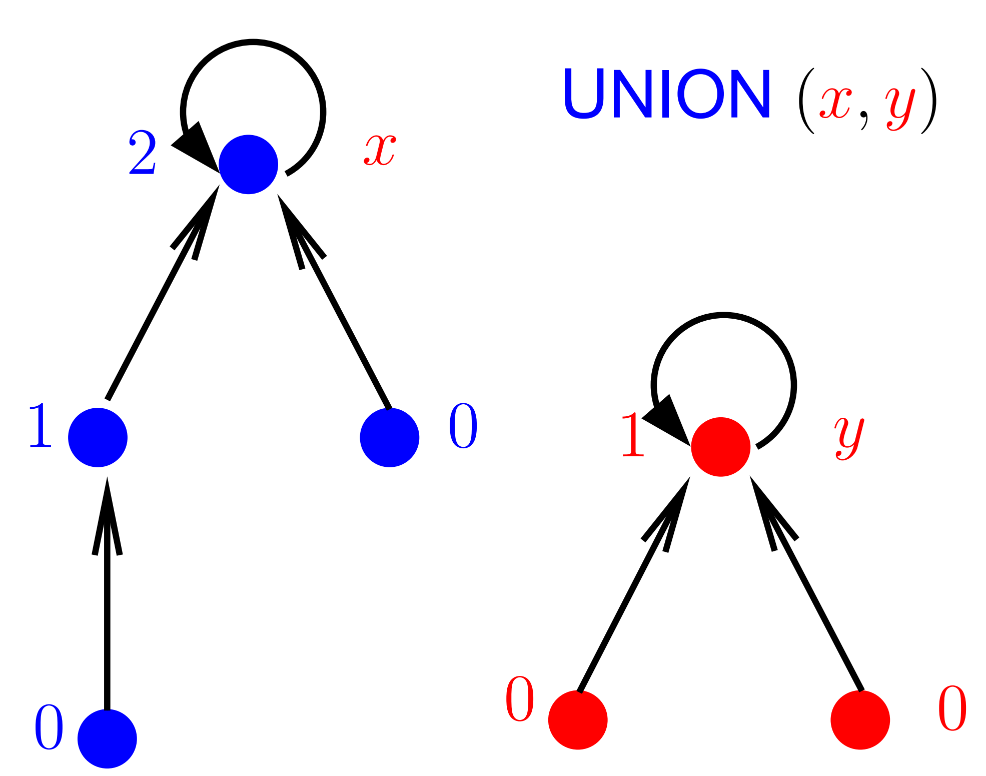
$rank[x]$ = posto do nó x

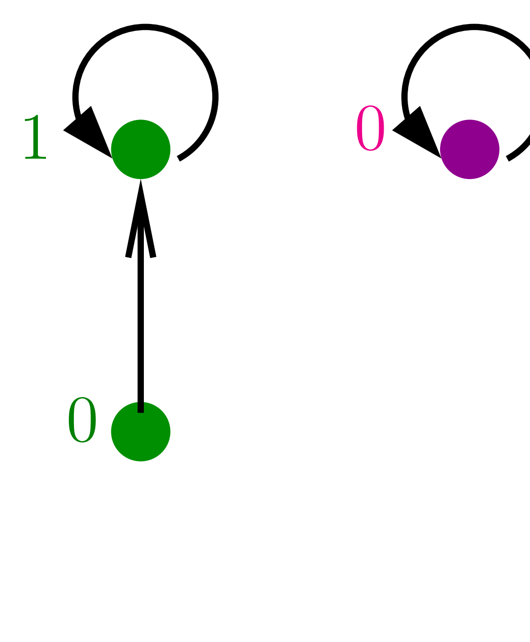
MAKESET (x)

1 $pai[x] \leftarrow x$

2 $rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*





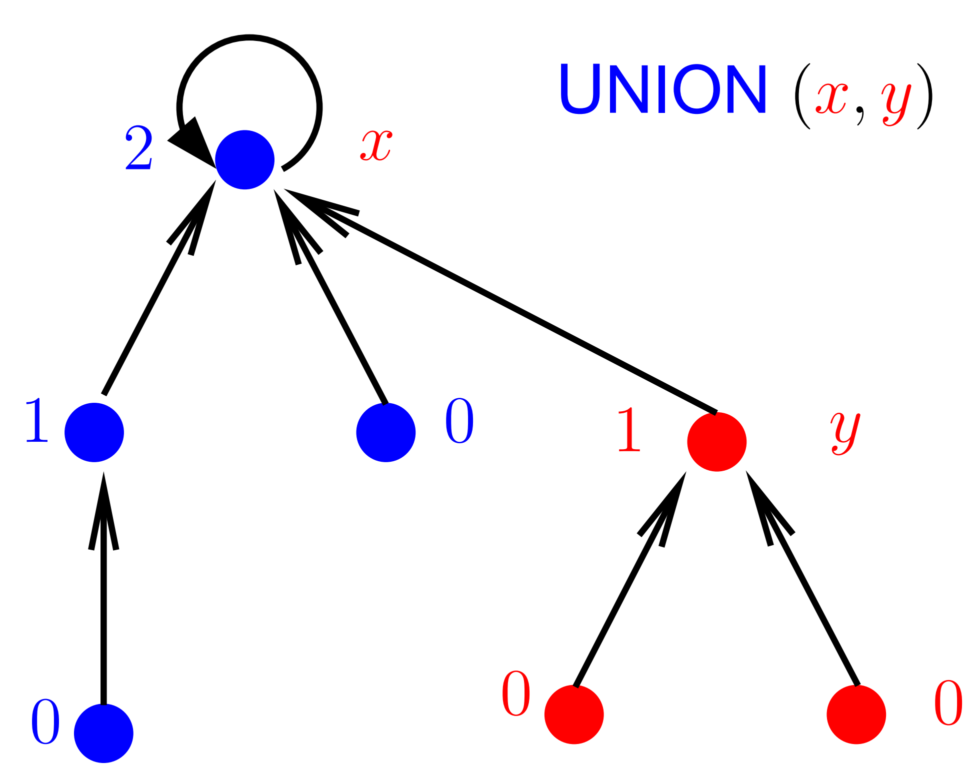
$rank[x]$ = posto do nó x

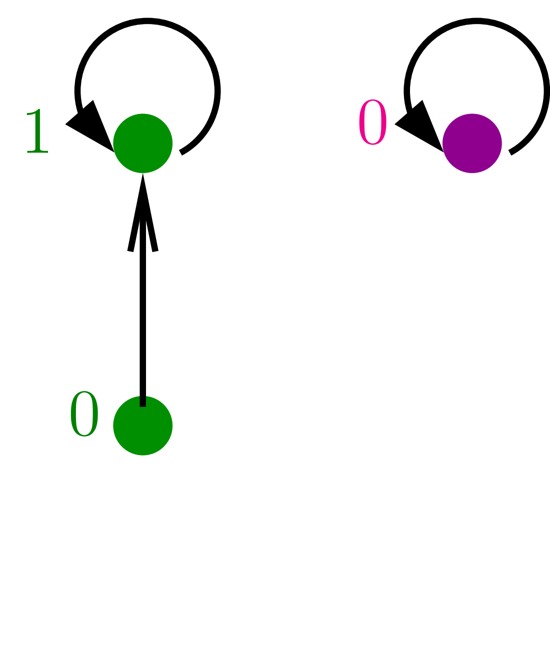
MAKESET (x)

1 $pai[x] \leftarrow x$

2 $rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*





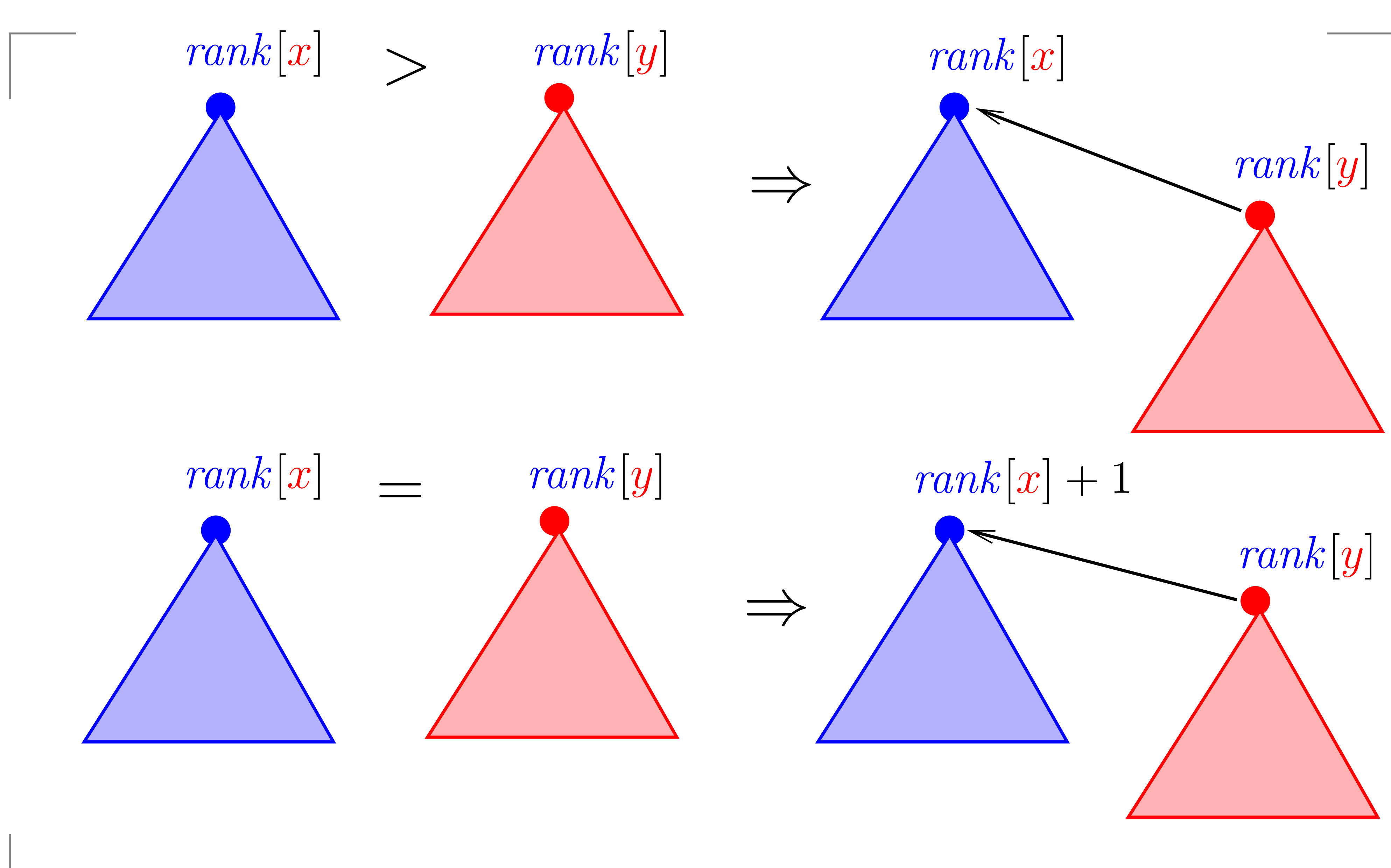
$rank[x]$ = posto do nó x

MAKESET (x)

1 $pai[x] \leftarrow x$

2 $rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*



Melhoramento 1: *union by rank*

UNION (x, y) $\triangleright$ com “union by rank”

1 $x' \leftarrow$ FINDSET (x)

2 $y' \leftarrow$ FINDSET (y) $\triangleright$ supõe que $x' \neq y'$

3 **se** $rank[x'] > rank[y']$

4 **então** $pai[y'] \leftarrow x'$

5 **senão** $pai[x'] \leftarrow y'$

6 **se** $rank[x'] = rank[y']$

7 **então** $rank[y'] \leftarrow rank[y'] + 1$

Melhoramento 1: estrutura

- $rank[x] \leq rank[pai[x]]$ para cada nó x
- $rank[x] = rank[pai[x]]$ se e só se x é raiz
- $rank[pai[x]]$ é uma função não-descrente do tempo
- número de nós de uma árvore de raiz x é $\geq 2^{rank[x]}$.
- número de nós de posto k é $\leq n/2^k$.
- $altura(x) = rank[x] \leq \lg n$ para cada nó x

$altura(x) :=$ comprimento do mais longo caminho que vai de x até uma folha

Melhoramento 1: custo

Seqüência de operações MAKESET, UNION, FINDSET

M M M U F U U F U F F F U F

$\underbrace{\underbrace{\text{M M M}}_{n}\ \text{U F U U F U F F F U F}}_{m}$

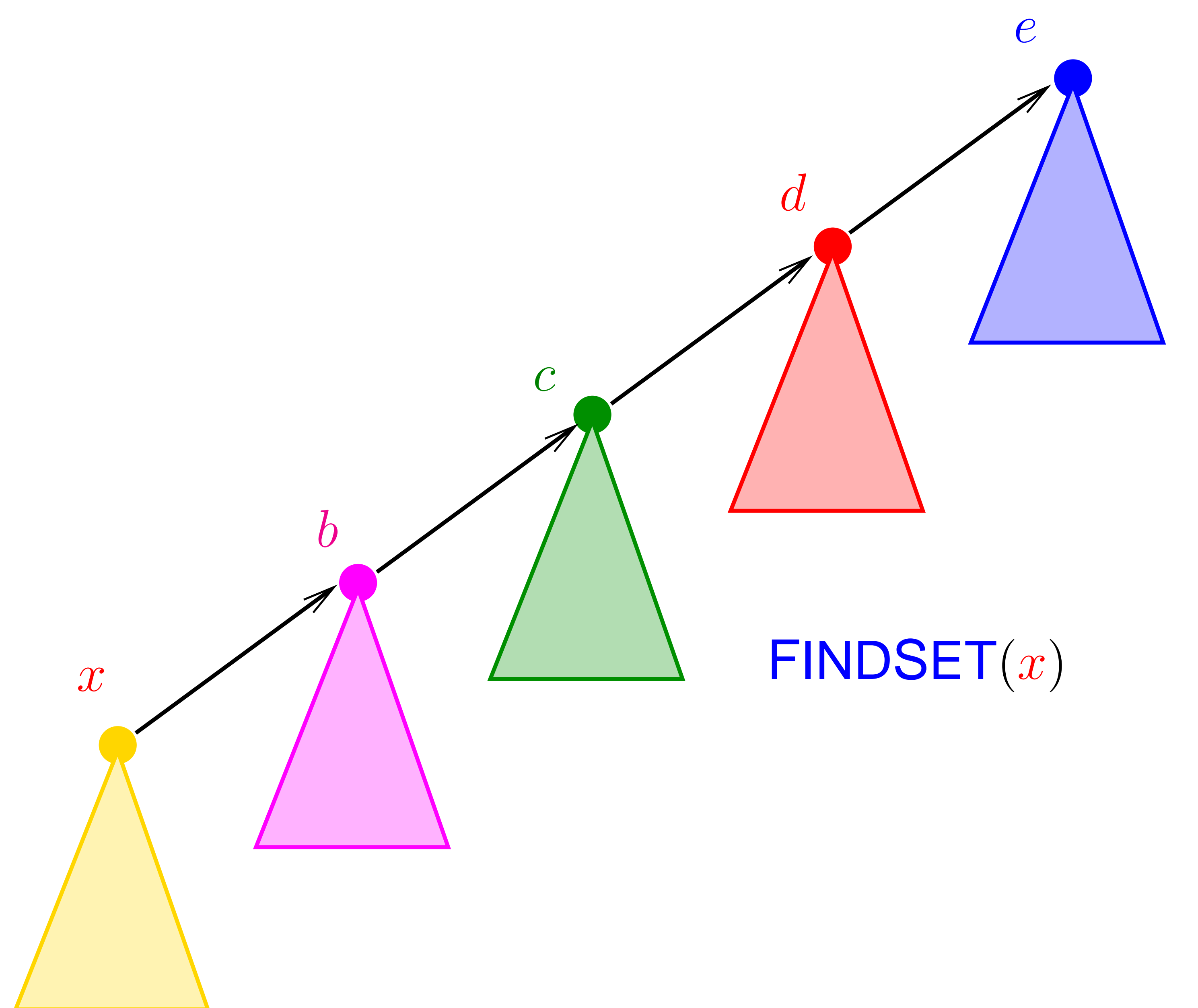
$altura(x) \leq \lg n$ para cada nó x

Consumos de tempo:

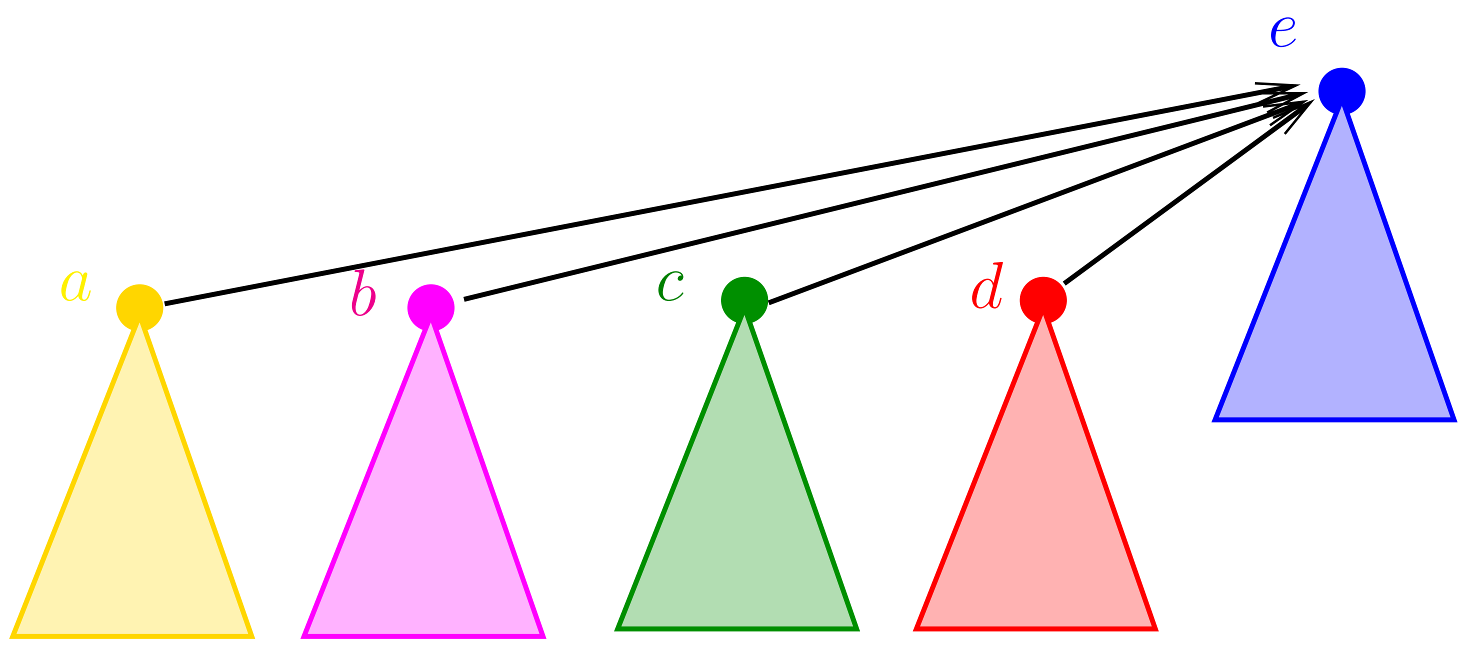
MAKESET	$\Theta(1)$
UNION	$\mathrm{O}(\lg n)$
FINDSET	$\mathrm{O}(\lg n)$

Consumo de tempo total da seqüência: $\mathrm{O}(m \lg n)$

Melhoramento 2: *path compression*



Melhoramento 2: *path compression*



FINDSET(x)

Melhoramento 2: *path compression*

FINDSET (x) ▷ com "path compression"
1 **se** $x \neq pai[x]$
2 **então** $pai[x] \leftarrow$ FINDSET $(pai[x])$
3 **devolva** $pai[x]$

- $rank[x] \leq rank[pai[x]]$ para cada nó x
- $rank[x] = rank[pai[x]]$ se e só se x é raiz
- $rank[pai[x]]$ é uma função não-descrente do tempo
- número de nós de uma árvore de raiz x é $\geq 2^{rank[x]}$.
- número de nós de posto k é $\leq n/2^k$.
- $altura(x) \leq rank[x] \leq \lg n$ para cada nó x

Melhores momentos

AULA 22

Conjuntos disjuntos dinâmicos

CLR 22 CLRS 21

Conjuntos disjuntos

Seja $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo $i \neq j$.

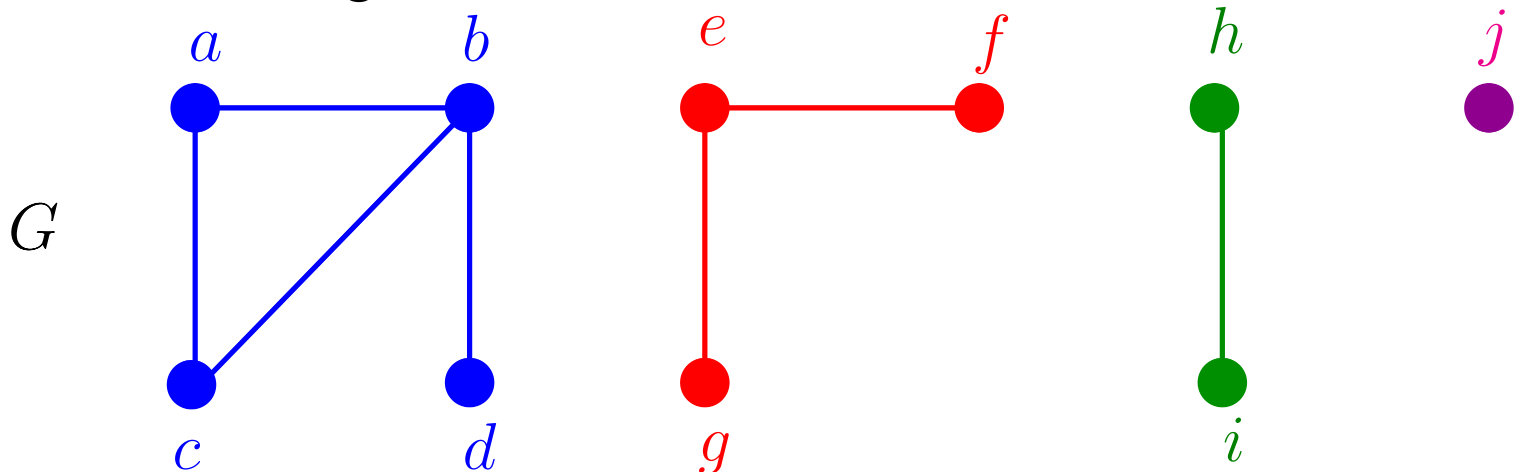
Conjuntos disjuntos

Seja $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo $i \neq j$.

Exemplo de coleção disjunta de conjuntos: componentes conexos de um grafo



componentes = conjuntos disjuntos de vértices

$$\{a, b, c, d\} \quad \{e, f, g\} \quad \{h, i\} \quad \{j\}$$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

a b e f h j

G

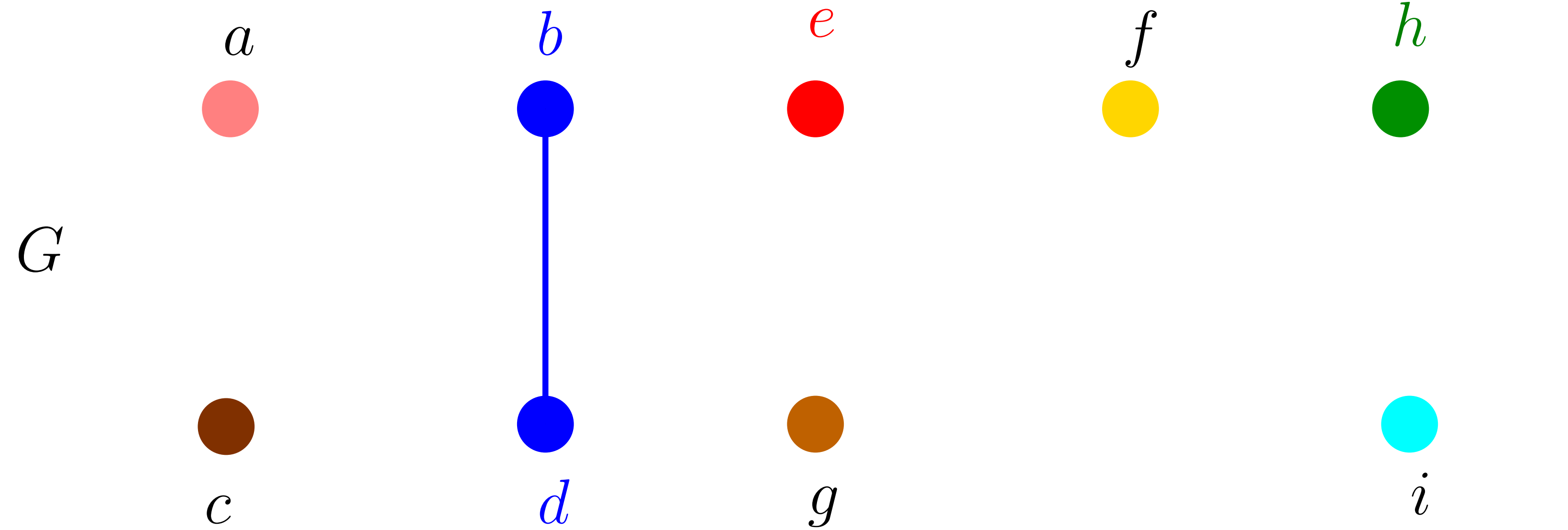
c d g i

aresta	componentes
	$\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

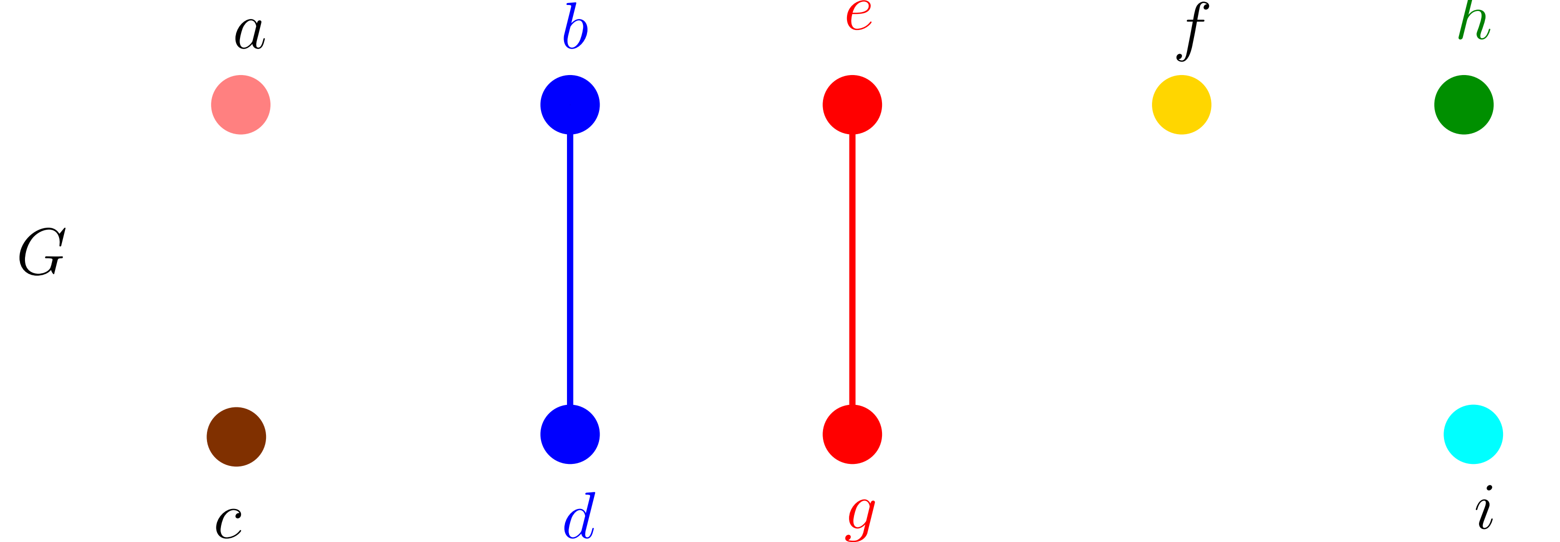


aresta	componentes
(b, d)	$\{a\}$ $\{b, d\}$ $\{c\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

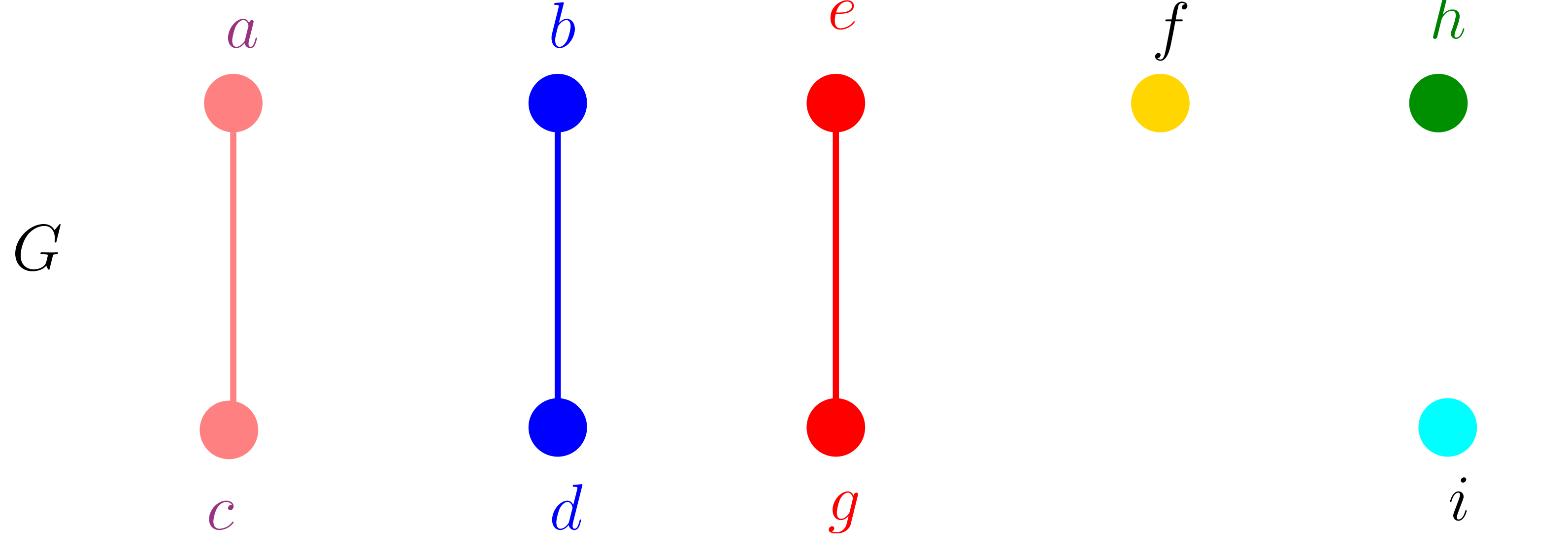


aresta	componentes
(e, g)	$\{a\}$ $\{b, d\}$ $\{c\}$ $\{e, g\}$ $\{f\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

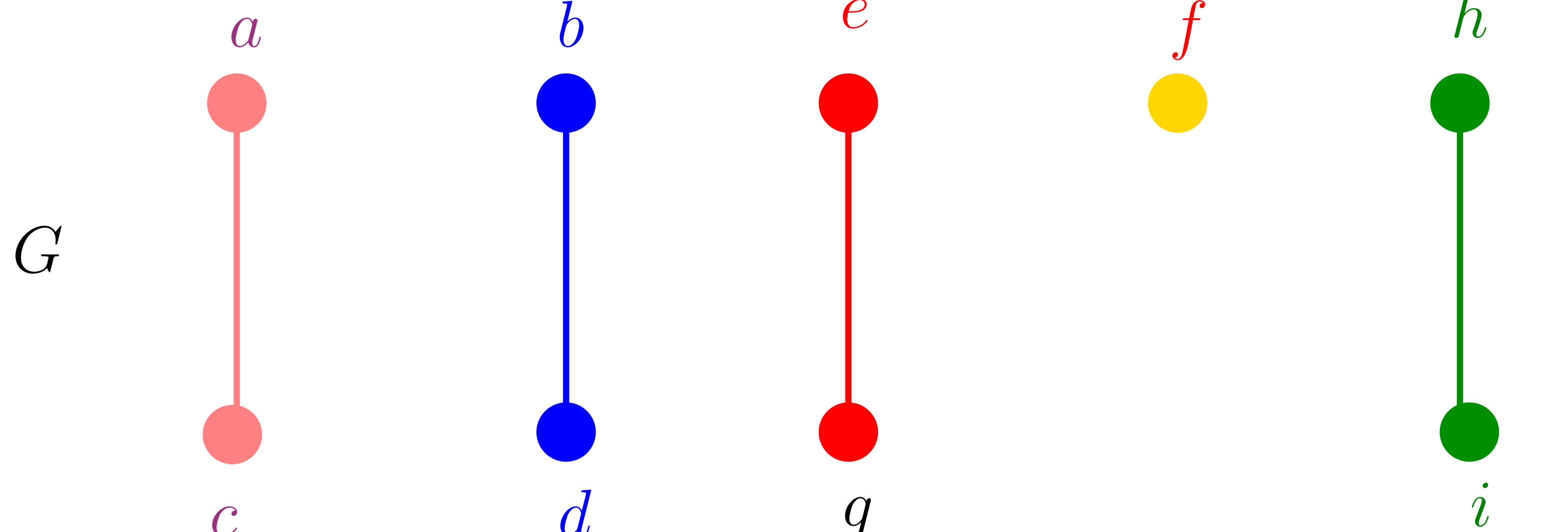


aresta	componentes
(a,c)	$\{a,c\}$ $\{b,d\}$ $\{e,g\}$ $\{f\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

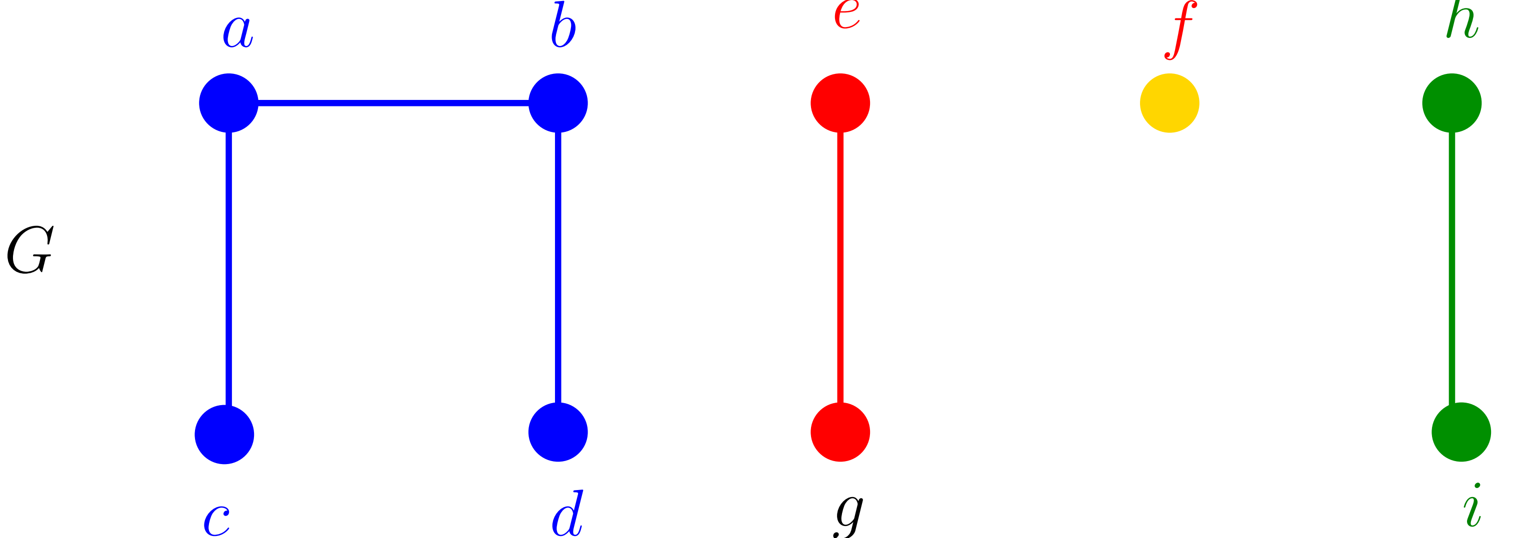


aresta	componentes
(h, i)	$\{a, c\}$ $\{b, d\}$ $\{e, g\}$ $\{f\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

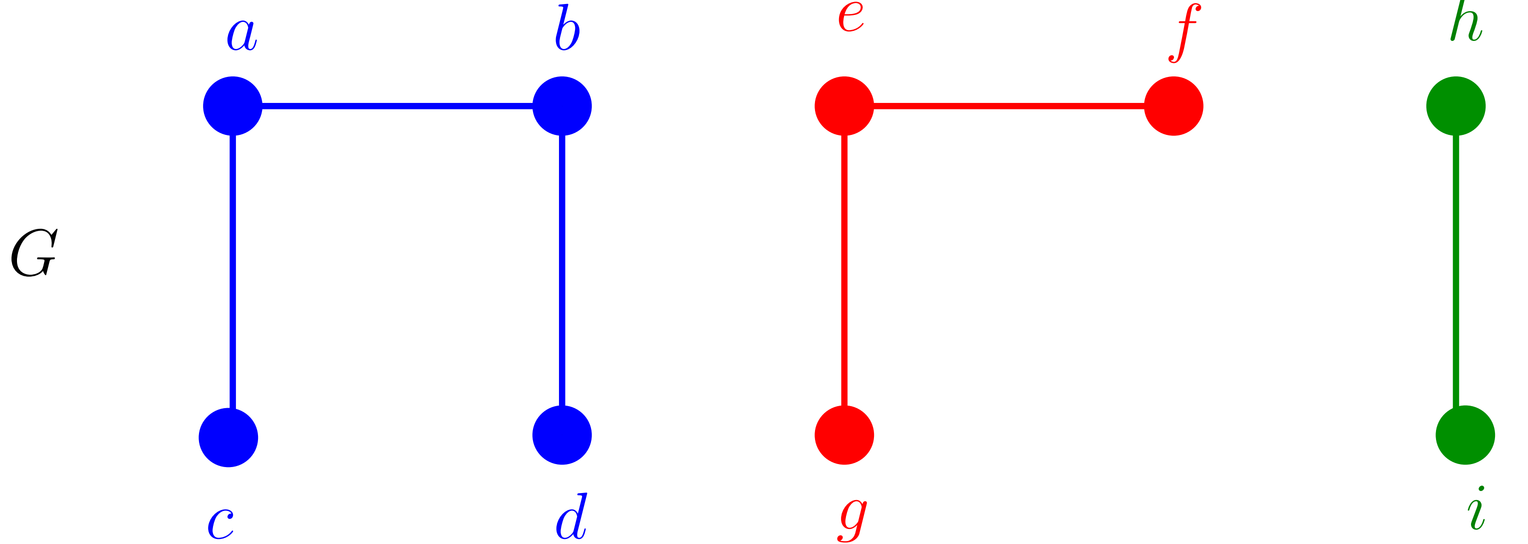


aresta	componentes
(a,b)	$\{a,b,c,d\}$ $\{e,g\}$ $\{f\}$ $\{h,i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

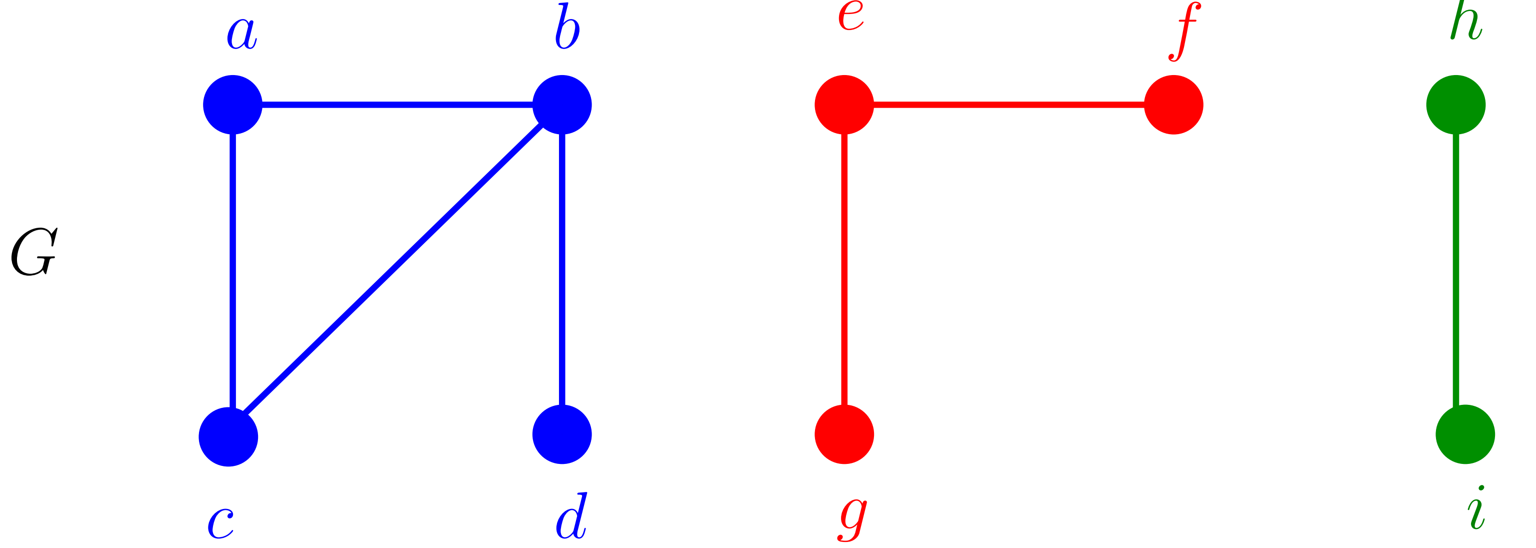


aresta	componentes
(e,f)	$\{a,b,c,d\}$ $\{e,f,g\}$ $\{h,i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



aresta	componentes
(b,c)	$\{a,b,c,d\}$ $\{e,g\}$ $\{f\}$ $\{h,i\}$ $\{j\}$

Operações básicas

$\mathcal{S}$ coleção de conjuntos disjuntos.

Cada conjunto tem um representante.

MAKESET (x): x é elemento novo

$\mathcal{S} \leftarrow \mathcal{S} \cup \{x\}$

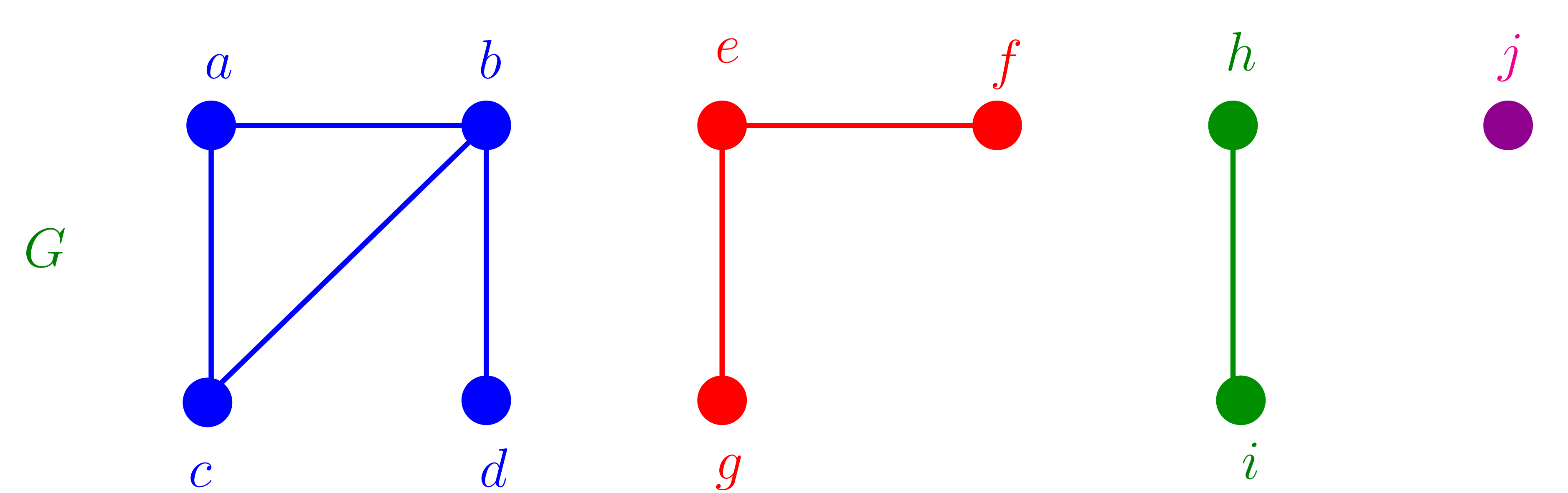
UNION (x, y): x e y em conjuntos diferentes

$\mathcal{S} \leftarrow \mathcal{S} - \{S_x, S_y\} \cup \{S_x \cup S_y\}$

x está em S_x e y está em Sy

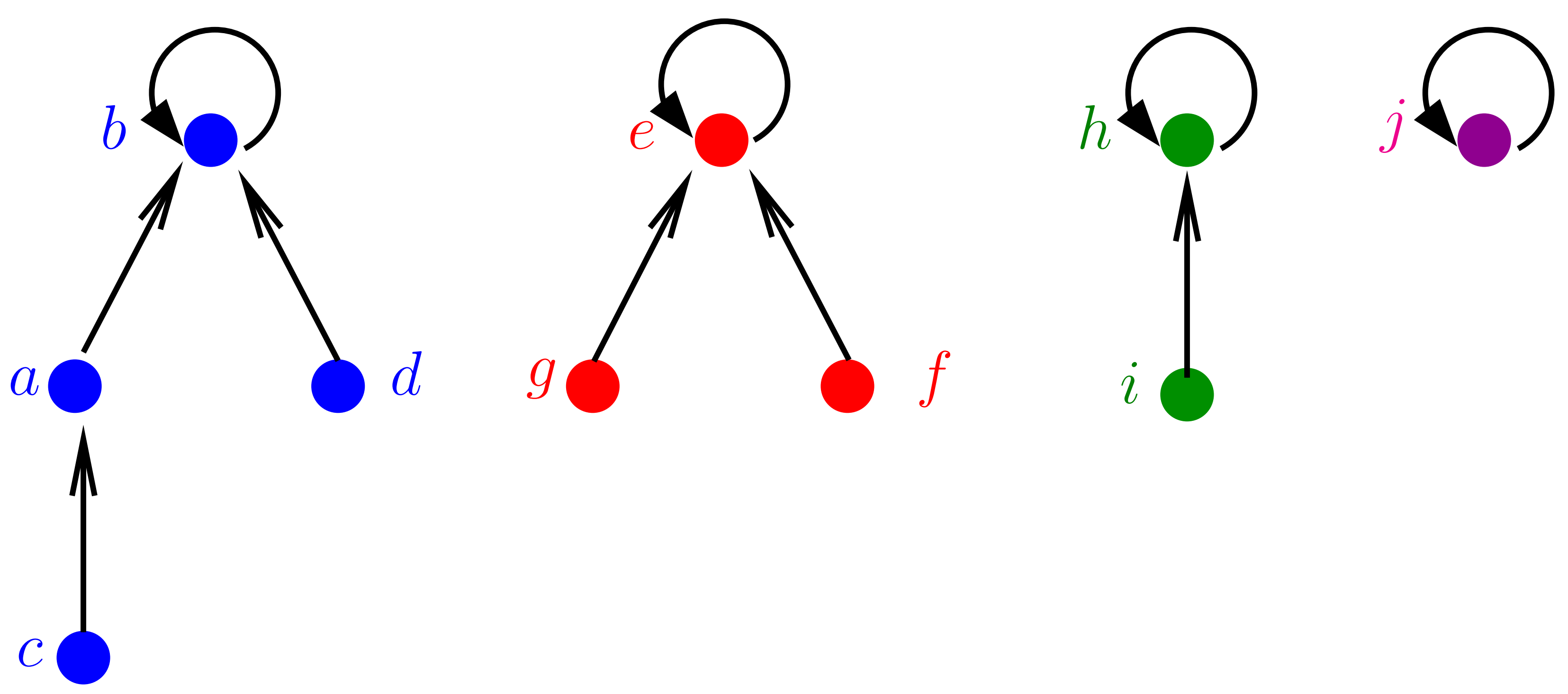
FINDSET (x): devolve representante do conjunto que contém x

Estrutura *disjoint-set forest*



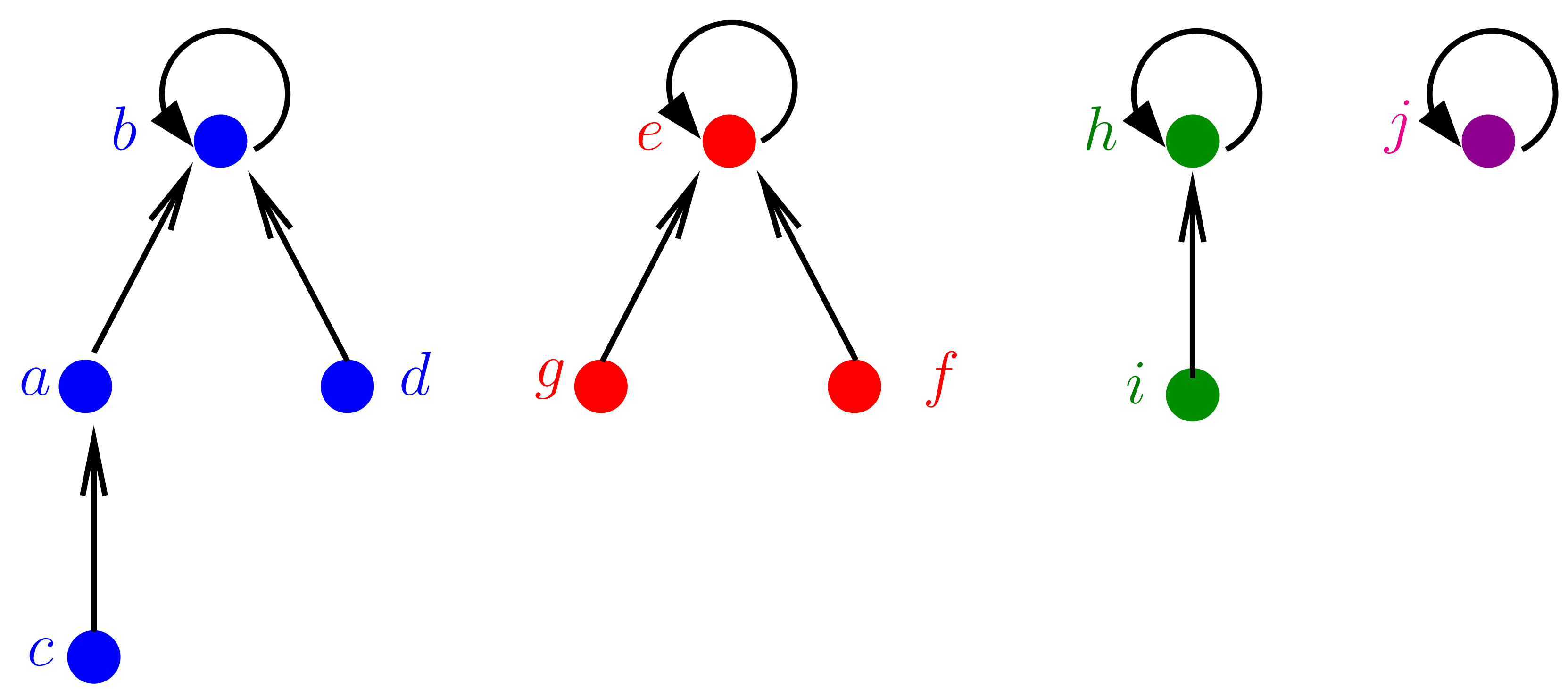
- cada conjunto tem uma $raiz$, que é o seu representate
- cada nó x tem um pai
- $pai[x] = x$ se e só se x é uma raiz

Estrutura *disjoint-set forest*



- cada conjunto tem uma $raiz$
- cada nó x tem um pai
- $pai[x] = x$ se e só se x é uma raiz

MakeSet$_0$ e FindSet$_0$



MAKESET$_0$ (x)

1 $\quad pai[x] \leftarrow x$

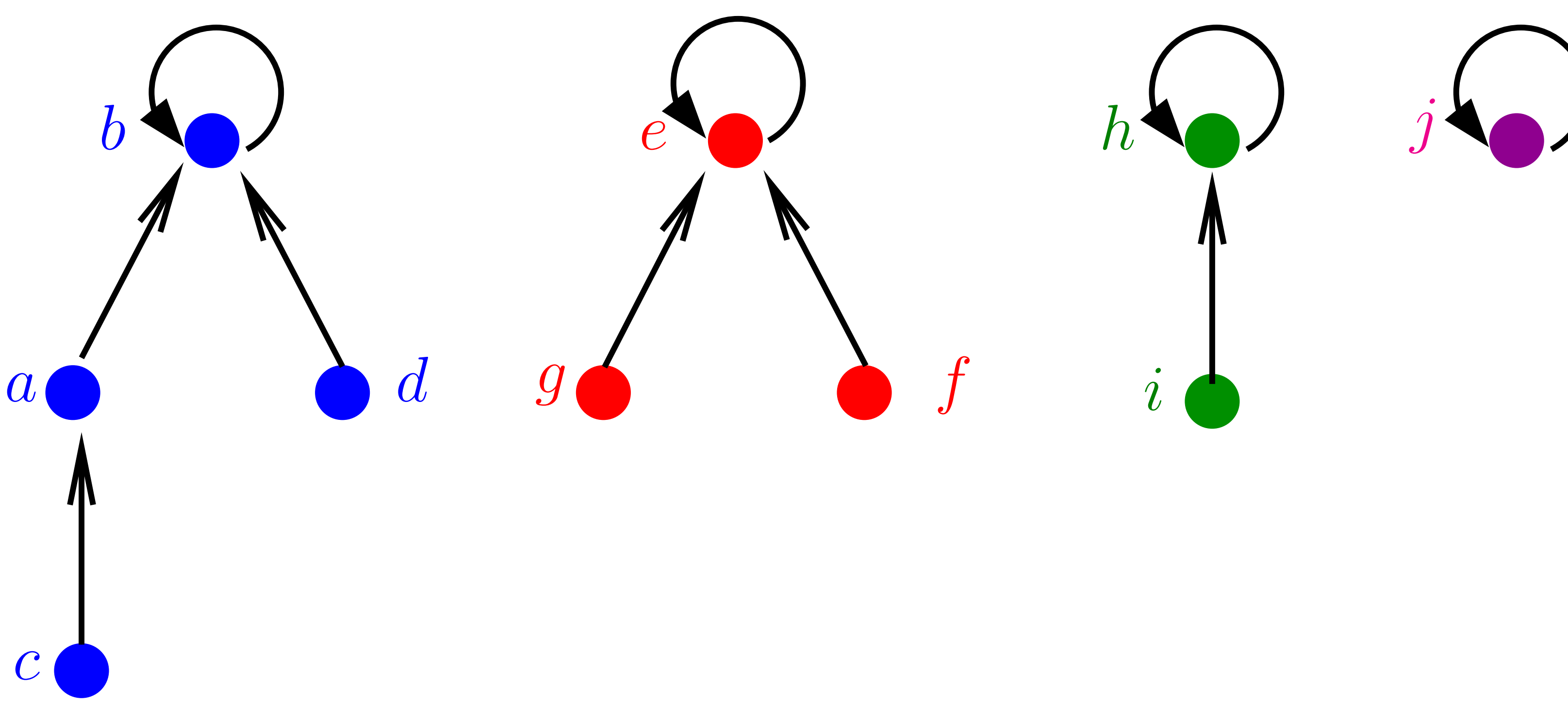
FINDSET$_0$ (x)

1 **enquanto** $pai[x] \neq x$ **faça**

2 $\quad\quad x \leftarrow pai[x]$

3 **devolva** x

FindSet$_1$



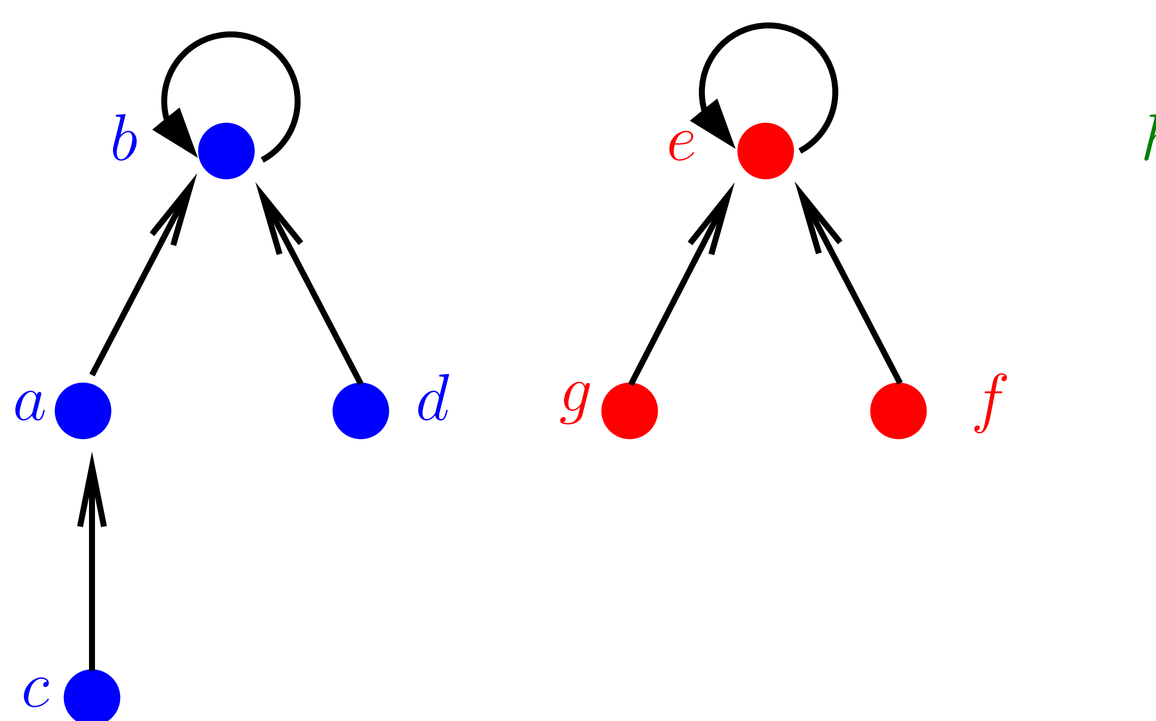
FINDSET$_1$ (x)

1 **se** $pai[x] = x$

2 **então devolva** x

3 **senão devolva** FINDSET$_1$ $(pai[x])$

Union$_0$



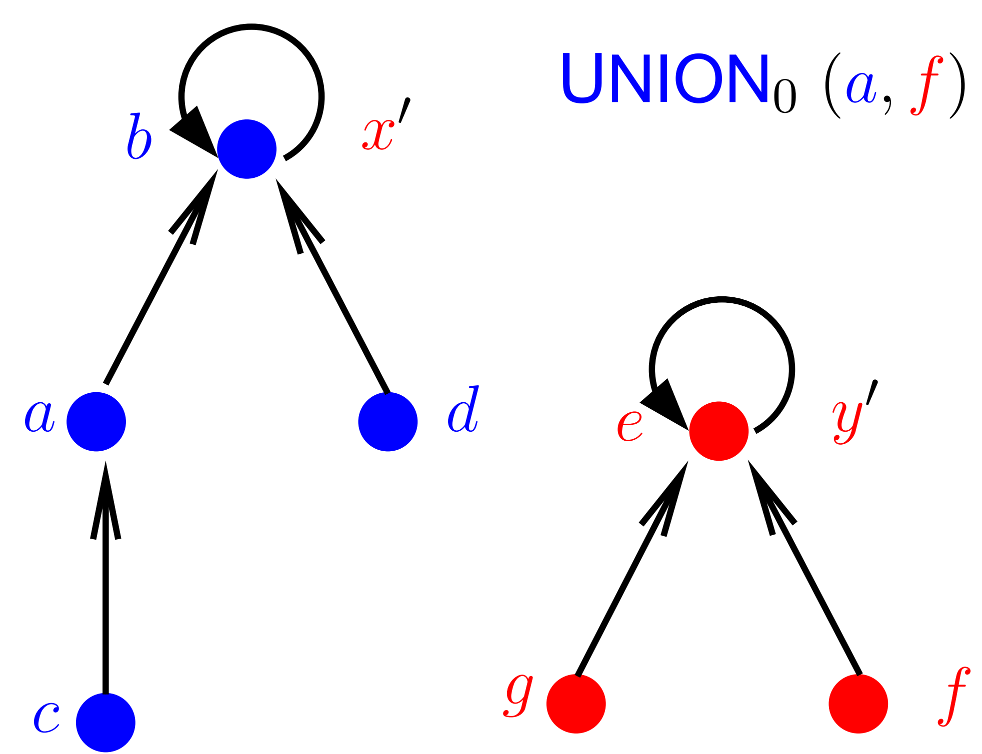
UNION$_0$ (x, y)

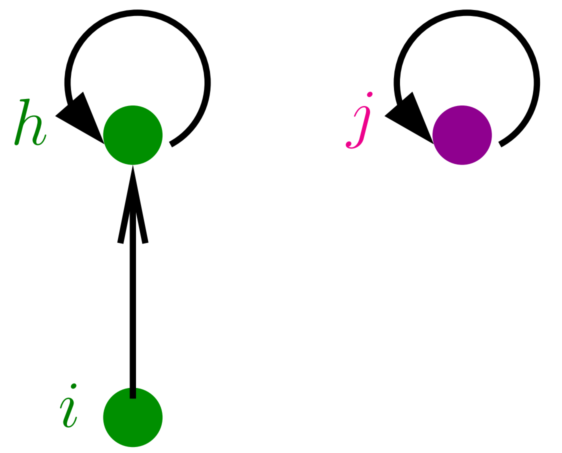
1 $\quad x' \leftarrow$ FINDSET$_0$ (x)

2 $\quad y' \leftarrow$ FINDSET$_0$ (y)

3 $\quad pai[y'] \leftarrow x'$

Union$_0$





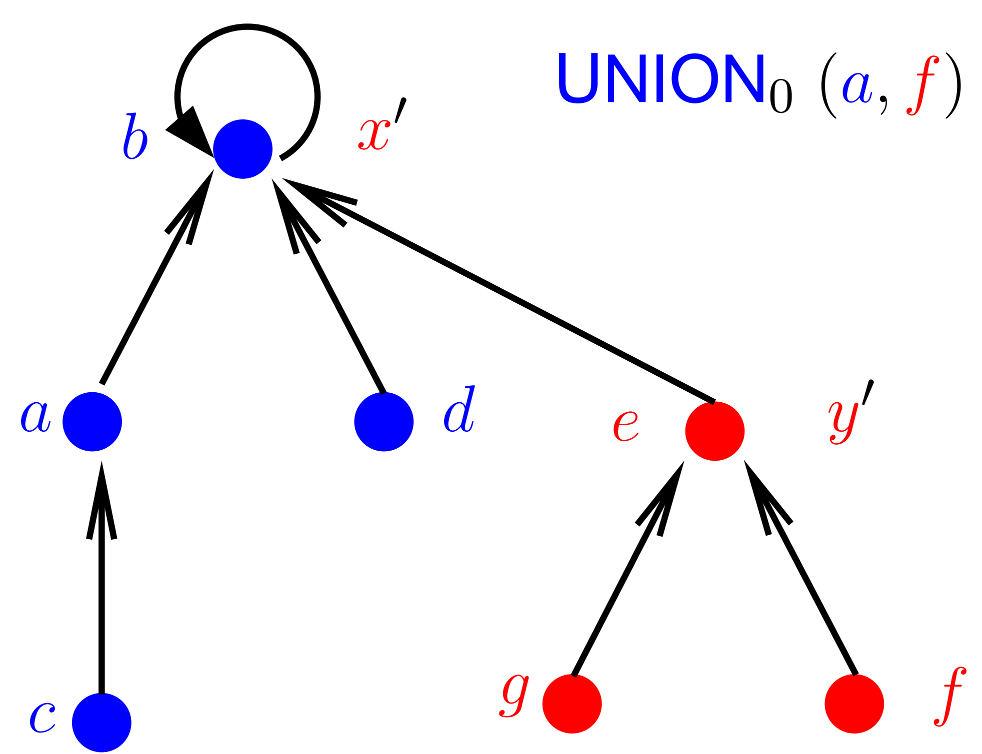
UNION$_0$ (x, y)

1 $x' \leftarrow$ FINDSET$_0$ (x)

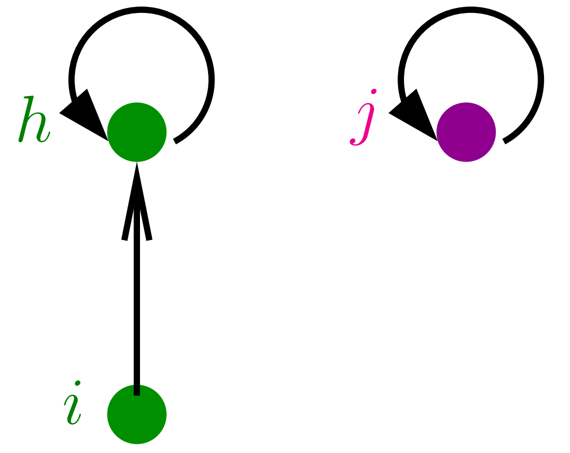
2 $y' \leftarrow$ FINDSET$_0$ (y)

3 $pai[y'] \leftarrow x'$

Union$_0$



UNION$_0$ (a, f)



UNION$_0$ (x, y)

1 $x' \leftarrow$ FINDSET$_0$ (x)

2 $y' \leftarrow$ FINDSET$_0$ (y)

3 $pai[y'] \leftarrow x'$

MakeSet$_0$, Union$_0$ e FindSet$_1$

MAKESET$_0$ (x)

1 $pai[x] \leftarrow x$

UNION$_0$ (x, y)

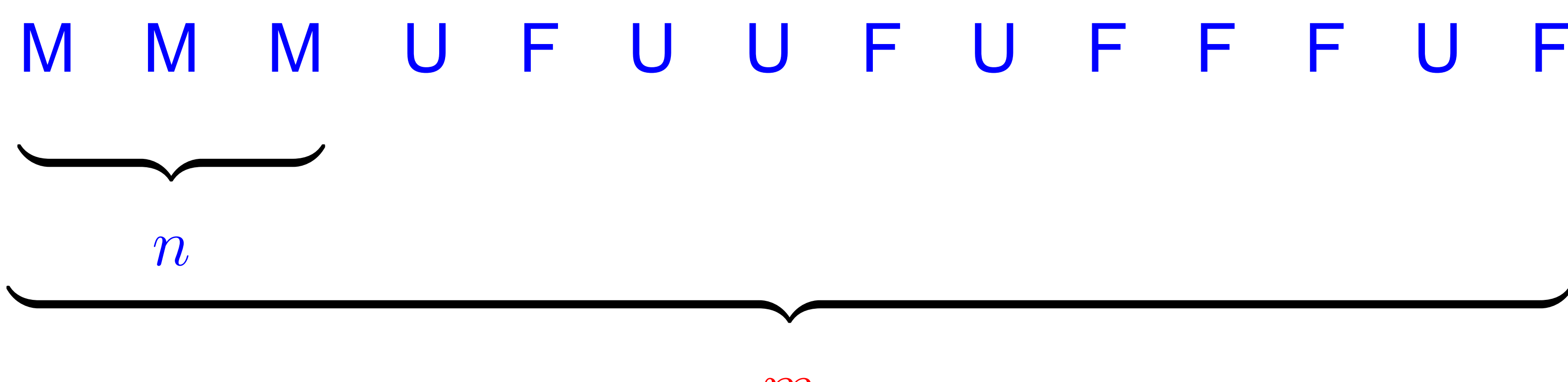
1 $x' \leftarrow$ FINDSET$_0$ (x)
2 $y' \leftarrow$ FINDSET$_0$ (y)
3 $pai[y'] \leftarrow x'$

FINDSET$_1$ (x)

1 **se** $pai[x] = x$
2 **então devolva** x
3 **senão devolva** FINDSET$_1$ $(pai[x])$

Consumo de tempo

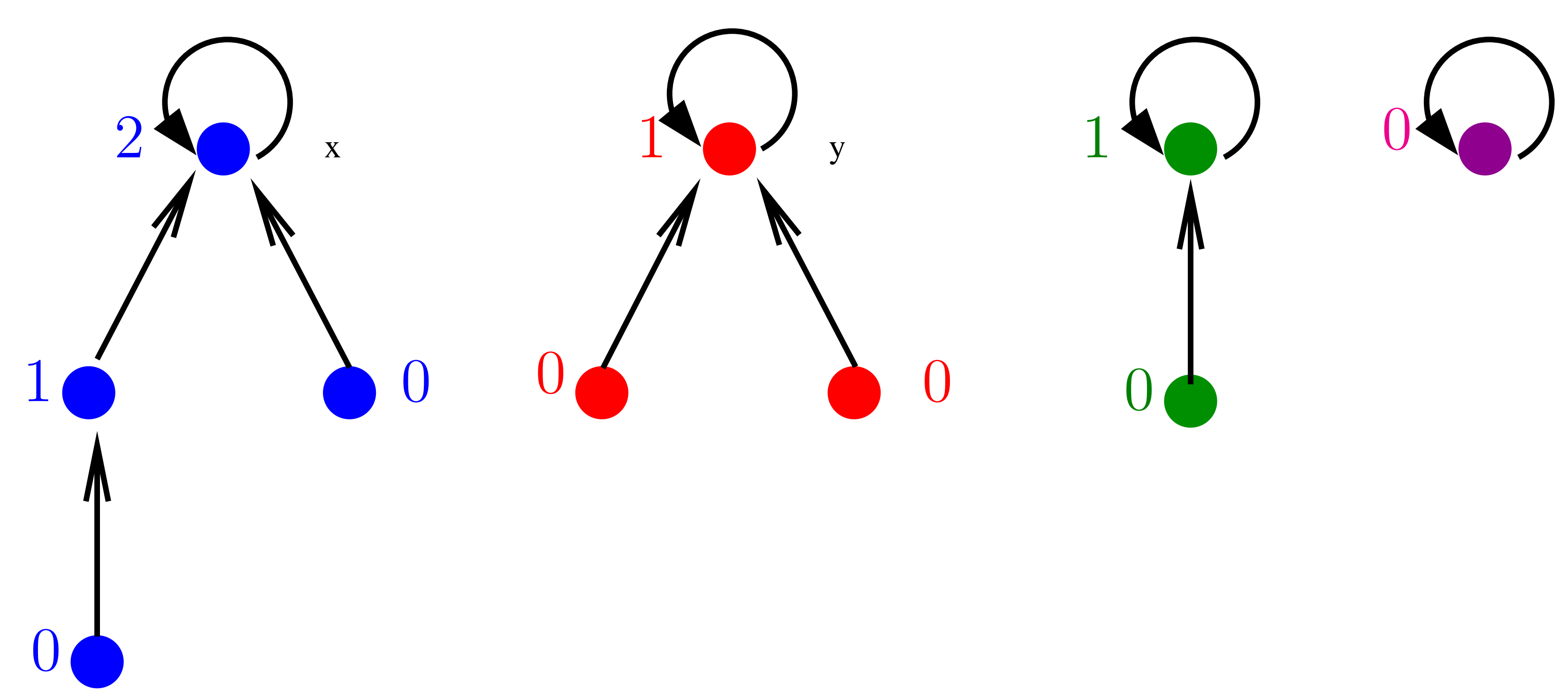
MAKESET$_0$	$\Theta(1)$
UNION$_0$	$O(n)$
FINDSET$_0$	$O(n)$



Custo total da seqüência:

$$n\,\Theta(1) + n\,O(n) + m\,O(n) = O(mn)$$

Melhoramento 1: *union by rank*



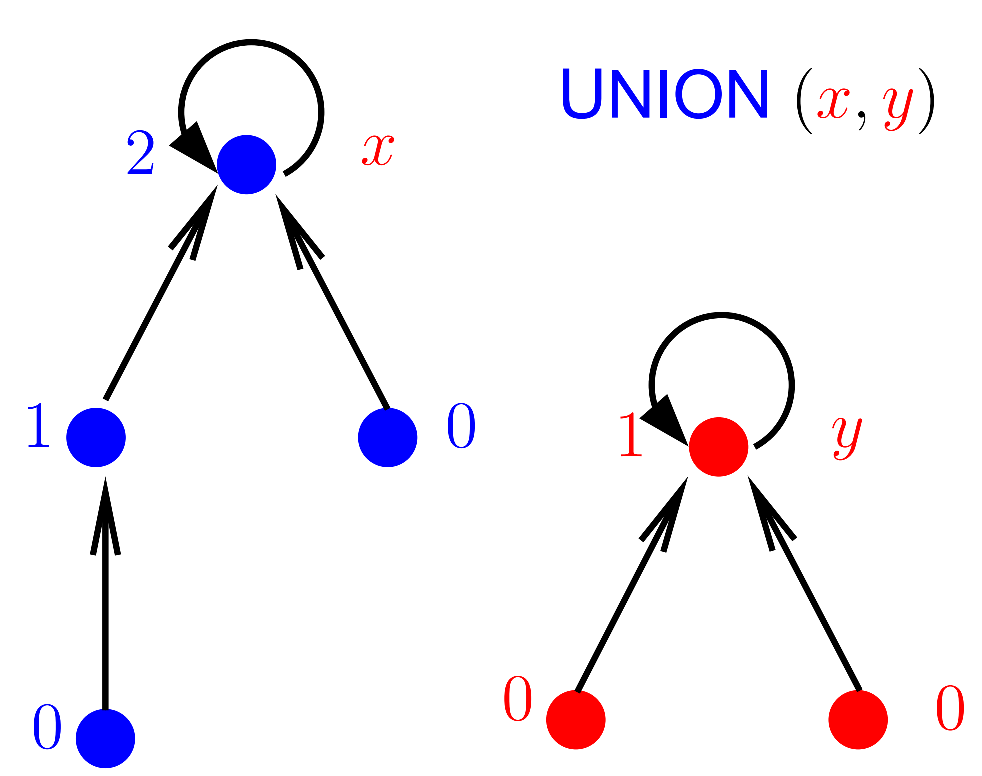
$rank[x]$ = posto do nó x

MAKESET (x)

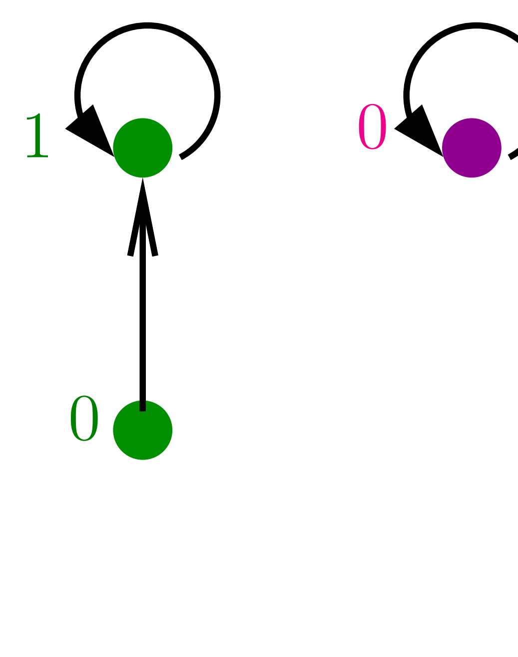
1 $\quad pai[x] \leftarrow x$

2 $\quad rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*



UNION (x, y)



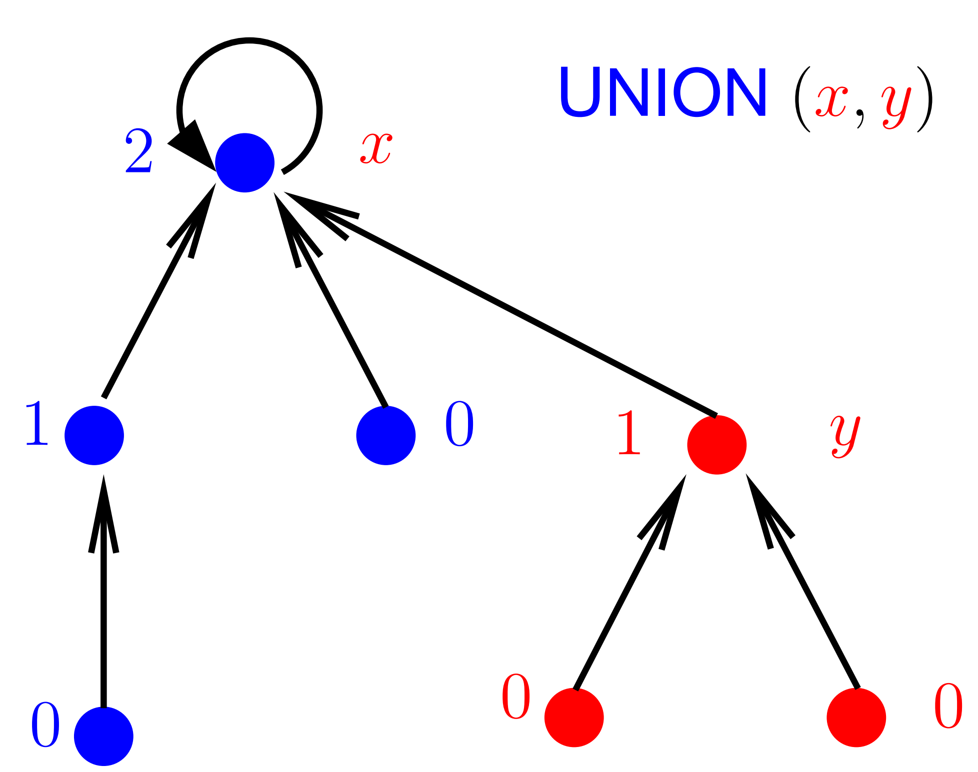
$rank[x]$ = posto do nó x

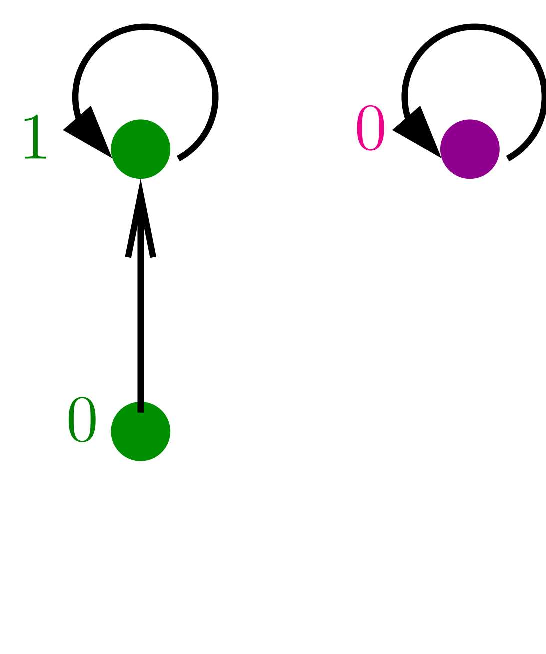
MAKESET (x)

1 $pai[x] \leftarrow x$

2 $rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*





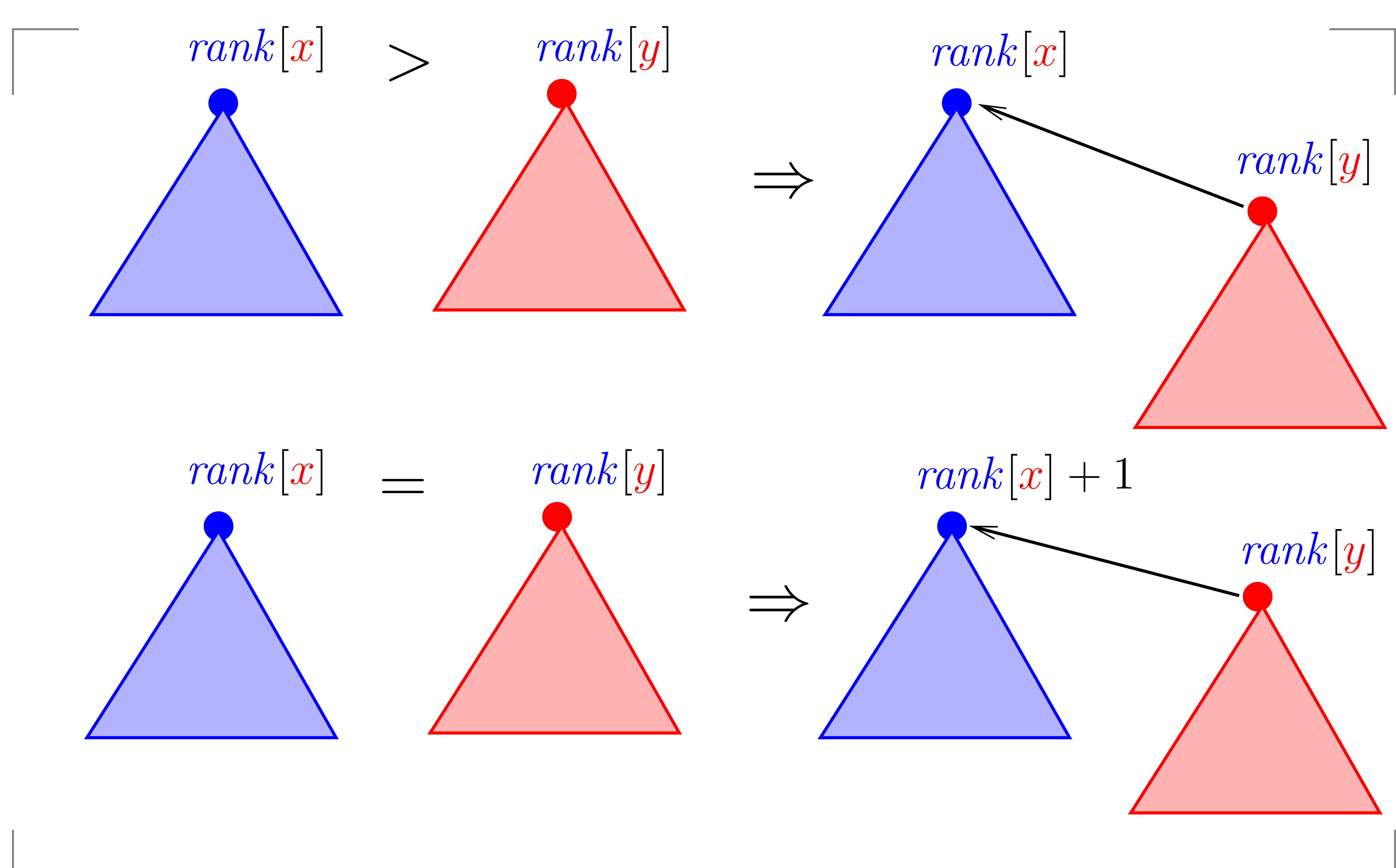
$rank[x]$ = posto do nó x

MAKESET (x)

1 $\quad pai[x] \leftarrow x$

2 $\quad rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*



Melhoramento 1: *union by rank*

UNION (x, y) $\triangleright$ com "union by rank"

1 $x' \leftarrow$ FINDSET (x)

2 $y' \leftarrow$ FINDSET (y) $\triangleright$ supõe que $x' \neq y'$

3 **se** $rank[x'] > rank[y']$

4 **então** $pai[y'] \leftarrow x'$

5 **senão** $pai[x'] \leftarrow y'$

6 **se** $rank[x'] = rank[y']$

7 **então** $rank[y'] \leftarrow rank[y'] + 1$

Melhoramento 1: estrutura

- $rank[x] \leq rank[pai[x]]$ para cada nó x
- $rank[x] = rank[pai[x]]$ se e só se x é raiz
- $rank[pai[x]]$ é uma função não-descrente do tempo
- número de nós de uma árvore de raiz x é $\geq 2^{rank[x]}$.
- número de nós de posto k é $\leq n/2^k$.
- $altura(x) = rank[x] \leq \lg n$ para cada nó x

$altura(x) :=$ comprimento do mais longo caminho que vai de x até uma folha

Melhoramento 1: custo

Seqüência de operações MAKESET, UNION, FINDSET

M M M U F U U F U F F F U F

$$\underbrace{\underbrace{\text{M M M}}_{n}\ \text{U F U U F U F F F U F}}_{m}$$

$altura(x) \leq \lg n$ para cada nó x

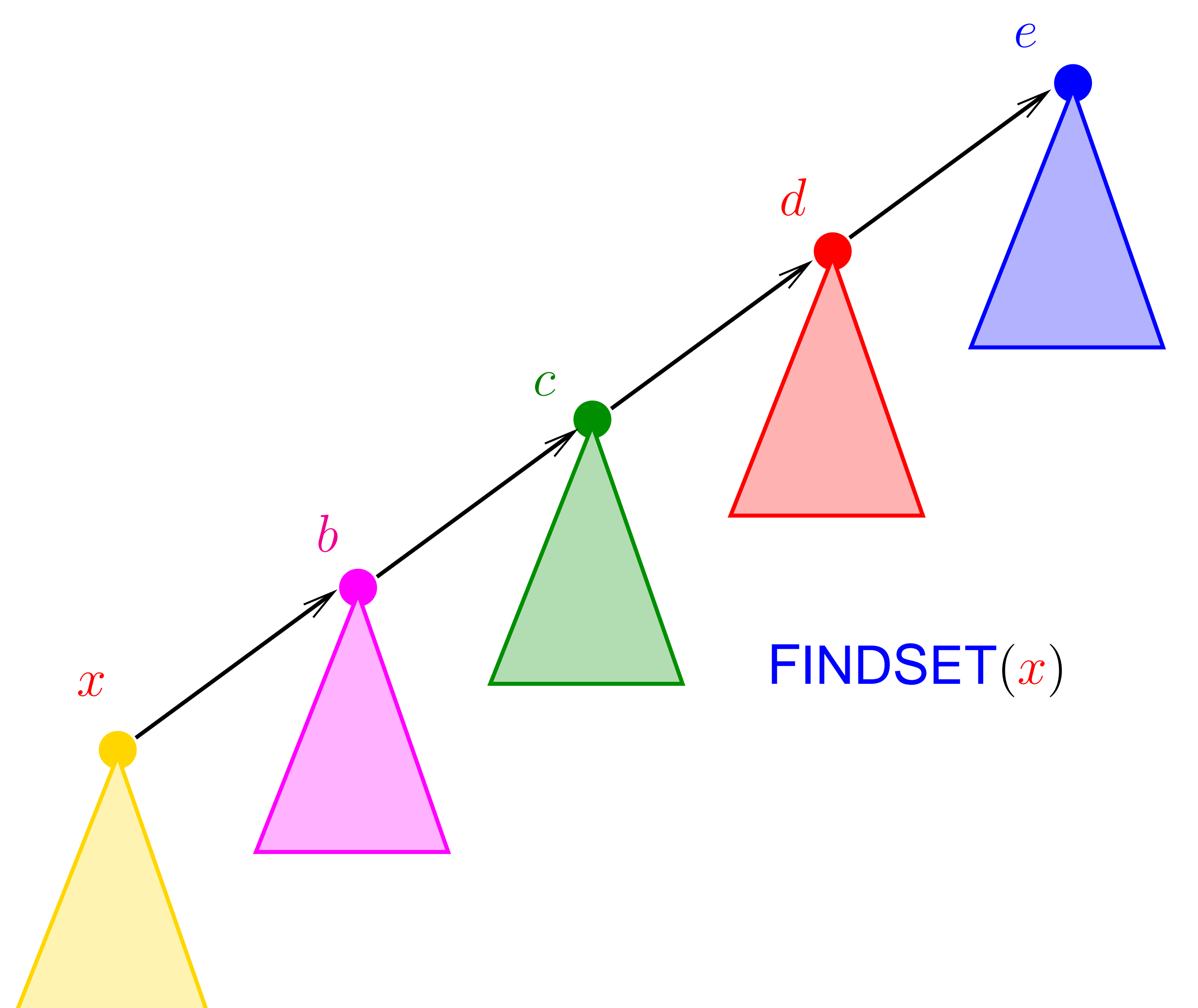
Consumos de tempo:

MAKESET	$\Theta(1)$
UNION	$O(\lg n)$
FINDSET	$O(\lg n)$

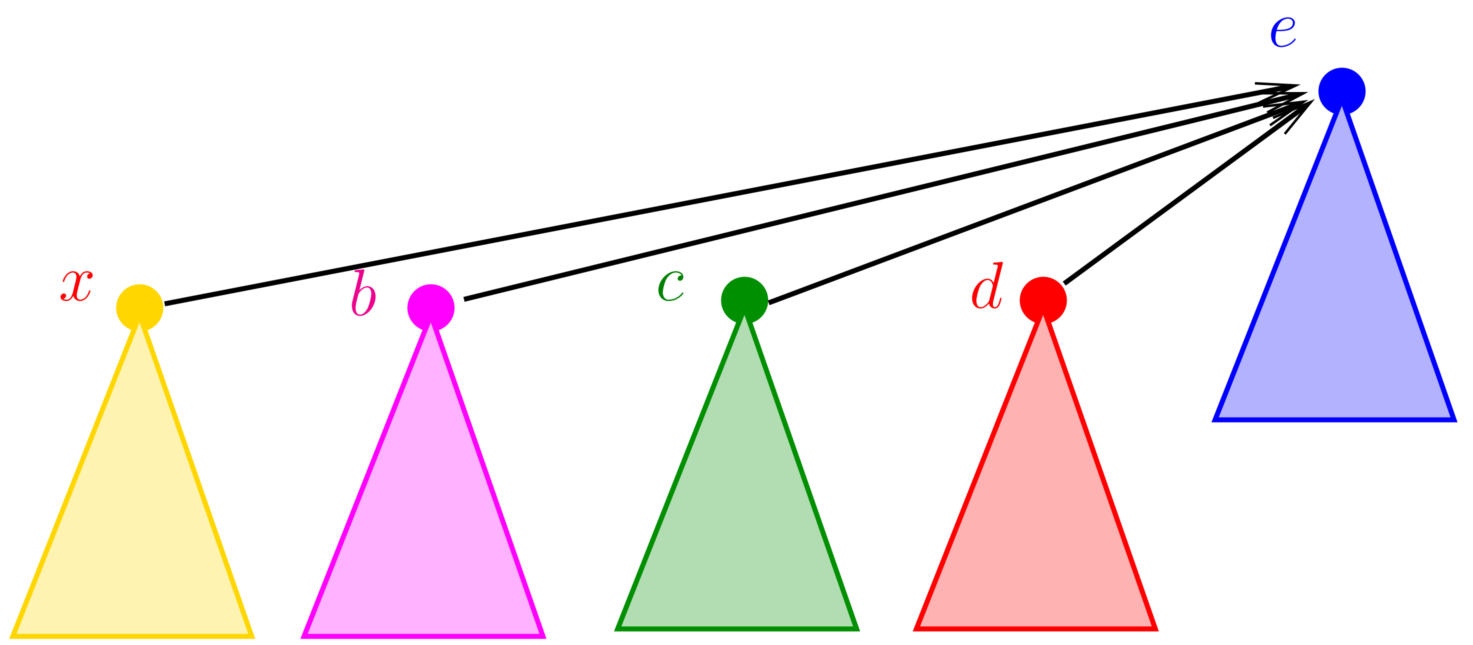
Consumo de tempo total da seqüência: $O(m \lg n)$

AULA 23

Melhoramento 2: *path compression*



Melhoramento 2: *path compression*



FINDSET(x)

Melhoramento 2: *path compression*

FINDSET (x) $\triangleright$ com "path compression"
1 **se** $x \neq pai[x]$
2 **então** $pai[x] \leftarrow$ FINDSET $(pai[x])$
3 **devolva** $pai[x]$

- $rank[x] \leq rank[pai[x]]$ para cada nó x
- $rank[x] = rank[pai[x]]$ se e só se x é raiz
- $rank[pai[x]]$ é uma função não-descrente do tempo
- número de nós de uma árvore de raiz x é $\geq 2^{rank[x]}$
- número de nós de posto k é $\leq n/2^k$
- $altura(x) \leq rank[x] \leq \lg n$ para cada nó x

Função log-estrela

$\lg^* n$ é o menor k tal que

$$\underbrace{\lg \lg \ldots \lg}_{k} n \leq 1$$

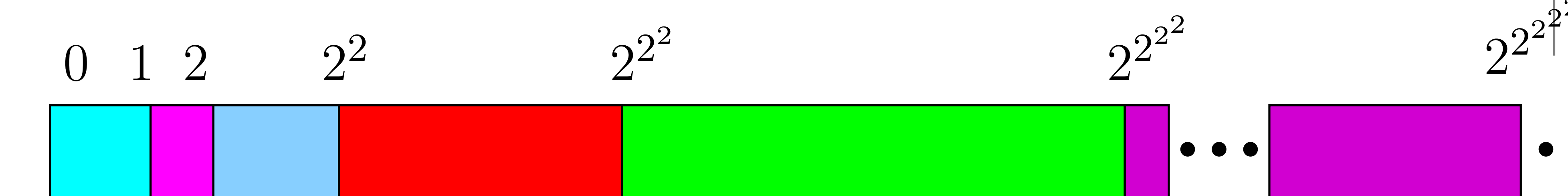
n	$\lg^* n$
1	0
2	1
3	2
4	2
5	3
$\vdots$	$\vdots$
15	3
16	3
$\vdots$	$\vdots$
65535	4
65536	4
$\vdots$	$\vdots$
$\underbrace{10000000000000000000 \cdots 000000000000}_{80}$	5
$\vdots$	$\vdots$

Função 'torre'

$$t(i) := \begin{cases} 1 & \text{se } i = 0 \\ 2^{t(i-1)} & \text{se } i = 1, 2, 3, \ldots \end{cases}$$

i	$t(i)$
0	1
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^{2^2} = 16$
4	$2^{2^{2^2}} = 2^{16} = 65536$
5	$2^{2^{2^{2^2}}} > \underbrace{100000000000000000 \cdots 000000000000}_{80}$
$\vdots$	$\vdots$

Blocos



$bloco[0] = [0\,..\,1]$

$bloco[1] = [2\,..\,2]$

$bloco[2] = [3\,..\,4]$

$bloco[3] = [5\,..\,16]$

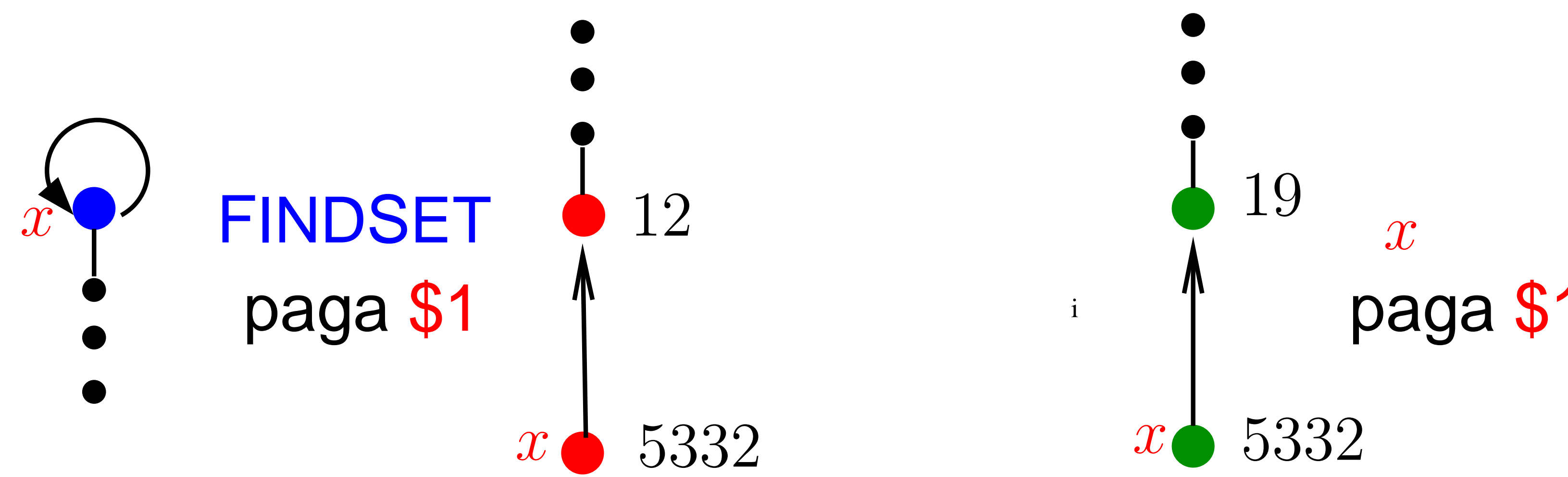
$bloco[i] = [t(i-1)+1\,..\,t(i)]$

Contabilidade

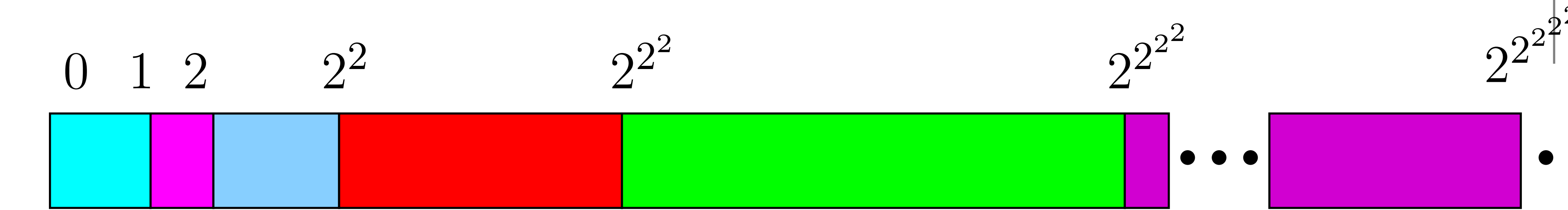
Custo de cada operação FINDSET(y) será contabilizado da seguinte maneira.

Para cada nó x no caminho de y até raiz:

- se x é a raiz ou $rank[x]$ e $rank[pai[x]]$ estão em blocos diferentes cobre $1 da operação FINDSET
- se $rank[x]$ e $rank[pai[x]]$ estão no mesmo bloco cobre $1 de x.



Pagamento de cada FindSet



Cada operação FINDSET paga $O(\lg^* n)$:

- $rank[x] \leq \log n$ para cada nó x
- há vértices em $\leq \lg^* n$ blocos:

$$\begin{array}{ccccc} t(i-1) & < & \lg n & \leq & t(i) \\ 0 & < & \lg^i(\lg n) & \leq & 1 \end{array}$$

$\Rightarrow i \leq \lg^* \lg n < \lg^* n$

Pagamento de cada vértice

Suponha que x não é raiz.

Se $rank[x]$ está em $bloco[i]$, então
x paga $\leq t(i) - t(i-1)$.

Seja $N(i)$ o número de vértices em $bloco[i]$, $i > 0$.
Temos que

$$
\begin{aligned}
N(i) &\leq \frac{n}{2^{t(i-1)+1}} + \frac{n}{2^{t(i-1)+2}} + \cdots + \frac{n}{2^{t(i)}} \\
&< \frac{n}{2^{t(i-1)+1}}(1 + 1/2 + 1/4 + \cdots) \\
&= \frac{n}{2^{t(i-1)}} \\
&= \frac{n}{t(i)}
\end{aligned}
$$

Pagamento de todos os vértices

Para cada i o valor pago por vértices em $bloco[i]$ é limitado por

$$\frac{n}{t(i)} \times (t(i) - t(i-1)) < n$$

Para cada x tem-se que $rank[x] \leq \lg n$
$\Rightarrow$ há vertices em $< \lg^* n$ blocos

Portanto, os vértices da *disjoint-set forest* pagam um total de $< n \lg^* n$.

Custo total da seqüência $\mathrm{O}(m \lg^* n + n \lg^* n) = \mathrm{O}(m \lg^* n)$

Conclusão

Se conjuntos disjuntos são representados através de disjoint-set forest com *union by rank* e *path compression*, então uma seqüência de m operações MAKESET, UNION e FINDSET, sendo que n são MAKESET, consome tempo $\mathrm{O}(m \lg^* n)$.

Exercícios

Exercício 24.A [CLRS 21.1-3]
Quando CONNECTED-COMPONENTS é aplicado a um grafo $G = (V, E)$ com k componentes, quantas vezes FINDSET é chamado? Quantas vezes UNION é chamado? Dê respostas em termos de k, $|V|$ e $|E|$.

Exercício 24.B [CLRS 21.3-1]
Faça uma figura da floresta produzida pela seguinte seqüência de operacões:

```
01    para i ← 1 até 16
02          faça MAKESET (x_i)
03    para i ← 1 até 15 em passos de 2
04          faça UNION (x_i, x_{i+1})
05    para i ← 1 até 13 em passos de 4
06          faça UNION (x_i, x_{i+2})
07    UNION (x_1, x_5)
08    UNION (x_11, x_13)
09    UNION (x_1, x_10)
10    FINDSET (x_2)
11    FINDSET (x_9)
```

Mais exercícios

Exercício 24.C [CLRS 21.3-2]
Escreva uma versão iterativa de FINDSET com "path compression".

Exercício 24.D [CLRS 21.3-3]
Dê uma seqüência de m MAKESET, UNION e FINDSET$_0$, n das quais são MAKESET, que consome $\Omega(m \lg n)$.

Exercício 24.E
Digamos que $h[x]$ é a altura do nó x (= comprimento do mais longo caminho que vai de x até uma folha) na estrutura disjoint-set forest. Mostre que $rank[x] \geq h[x]$. Mostre que UNION (x, y) nem sempre pendura a árvore mais baixa na mais alta.

Exercício 24.F [CLRS 21.4-2]
Mostre que o pôsto de cada nó na estrutura *disjoint-set forest* é no máximo $\lfloor \lg n \rfloor$ ou seja, que $rank[x] \leq \lg n$. (Sugestão: Mostre inicialmente que para cada raiz x temos $2^{rank[x]} \leq n_x$, onde n_x é o número de nós na árvore que contém x.)

Mais um exercício

Exercício 24.G [CLRS 21.4-4]
Considere uma versão simplificada da estrutura *disjoint-set forest* que usa a heurística "union by rank" mas não usa a heurística "path compression". (Em outras palavras, usa as operações MAKESET, UNION e FINDSET_0.) Mostre que essa simplificação consome tempo

$$O(m \lg n) .$$

Como de hábito, m é o número total de operações e n é o número de operações MAKESET. (*Sugestão*: Use o exercício CLRS 21.4-2.)

AULA 24

Máximo divisor comum

CLRS 31.1 e 31.2

Divisibilidade

Suponha que a, b e d são números inteiros.

Dizemos que d divide a se $a = k\,d$ para algum número inteiro k.

$d \mid a$ é uma abreviação de "d divide a

Se d divide a, então dizemos que a é um multiplo de d.

Se d divide a e $d > 0$, então dizemos que d é um divisor de a

Se d divide a e d divide b, então d é um divisor comum de a e b.

Exemplo:
os divisores de 30 são: $1, 2, 3, 5, 6, 10, 15$ e 30
os divisores de 24 são: $1, 2, 3, 4, 6, 8, 12$ e 24
os divisores comuns de 30 e 24 são: $1, 2, 3$ e 6

Máximo divisor comum

O máximo divisor comum de dois números inteiros a e b, onde pelo menos um é não nulo, é o maior divisor comum de a e b.

O máximo divisor comum de a e b é denotado por $\text{mdc}(a, b)$.

Exemplo:
máximo divisor comum de 30 e 24 é 6
máximo divisor comum de 514229 e 317811 é 1
máximo divisor comum de 3267 e 2893 é 11

Problema: Dados dois números inteiros não-negativos a e b, determinar $\text{mdc}(a, b)$.

Café com leite

Recebe números inteiros não-negativos a e b e devolve mdc(a, b).

Café-Com-Leite (a, b) $\triangleright$ supõe $a \neq 0$ ou $b \neq 0$

1 **se** $b = 0$ **então devolva** a
2 **se** $a = 0$ **então devolva** b
3 $d \leftarrow b$
4 **enquanto** $d \nmid a$ **ou** $d \nmid b$ **faça**
5 $\quad d \leftarrow d - 1$
6 **devolva** d

Relações invariantes: na linha 4 vale que

(i0) na linha 4 vale que $1 \leq d \leq b$;
(i1) na linha 4 vale que $k \nmid a$ ou $k \nmid b$ para cada $k > d$; e
(i2) na linha 5 vale que $d \nmid a$ ou $d \nmid b$.

Consumo de tempo

linha	consumo de todas as execuções da linha
1-2	$\Theta(1)$
3	$\Theta(1)$
4	$O(b)$
5	$O(b)$
6	$\Theta(1)$
total	$\Theta(3) + O(b) = O(b)$

Quando a e b são relativamente primos, ou seja mdc$(a, b) = 1$, o consumo de tempo do algoritmo é $\Theta(b)$.

Conclusão

O consumo de tempo do algoritmo Café-Com-Leite é $O(b)$.

No pior caso, o consumo de tempo do algoritmo Café-Com-Leite é $\Theta(b)$.

Se o valor de b dobra, o consumo de tempo pode dobrar.

Seja $\beta := \langle b \rangle$ o número de bits ou tamanho de b.
O consumo de tempo do algoritmo Café-Com-Leite é $O(2^\beta)$.

Brincadeira

Usei uma implementação do algoritmo Café-Com-Leite para determinar mdc$(2147483647, 2147483646)$.
Vejam quanto tempo gastou:

```
meu_prompt> time mdc 2147483647 2147483646
mdc: mdc de 2147483647 e 2147483646 e' 1.

real      2m49.306s
user      1m33.412s
sys       0m0.099s
```

Algoritmo de Euclides

Recebe números inteiros não-negativos a e b e devolve mdc(a, b).

EUCLIDES (a, b) $\triangleright$ supõe $a \neq 0$ ou $b \neq 0$

1 **se** $b = 0$
2 **então devolva** a
3 **senão devolva** EUCLIDES $(b, a \mod b)$

"$a \mod b$" é o resto da divisão de a por b.

Exemplo: mdc$(12, 18) = 6$, pois

```
mdc(12,18)
   mdc(18,12)
      mdc(12,6)
         mdc(6,0)
```

Correção

A correção do algoritmo EUCLIDES é baseada no seguinte fato.

> Suponha que $a \geq 0$ e $b > 0$. Para cada $d > 0$ vale que
>
> $d \mid a$ e $d \mid b$ se e só se $d \mid b$ e $d \mid a \bmod b$.

Em outras palavras, os pares (a, b) e $(b, a \bmod b)$ têm os mesmos divisores.

Outro exemplo

```
mdc(317811,514229)
  mdc(514229,317811)
    mdc(317811,196418)
      mdc(196418,121393)
        mdc(121393,75025)
          mdc(75025,46368)
            mdc(46368,28657)
              mdc(28657,17711)
                mdc(17711,10946)
                  mdc(10946,6765)
                    mdc(6765,4181)
                      mdc(4181,2584)
                        mdc(2584,1597)
                          mdc(1597,987)
                            mdc(987,610)
                              mdc(610,377)
```

Outro exemplo (cont.)

```
mdc(377,233)
   mdc(233,144)
      mdc(144,89)
         mdc(89,55)
            mdc(55,34)
               mdc(34,21)
                  mdc(21,13)
                     mdc(13,8)
                        mdc(8,5)
                           mdc(5,3)
                              mdc(3,2)
                                 mdc(2,1)
                                    mdc(1,0)
```

mdc$(317811, 514229) = 1$

Mais brincadeirinha

Usei uma implementação do algoritmo EUCLIDES para determinar mdc$(2147483647, 2147483646)$.
Vejam quanto tempo gastou:

```
meu_prompt> time euclides 2147483647 21474836
mdc(2147483647,2147483646)
  mdc(2147483646,1)
    mdc(1,0)
euclides: mdc de 2147483647 e 2147483646 e' 1

real    0m0.007s
user    0m0.002s
sys     0m0.004s
```

Consumo de tempo

O consumo de tempo do algoritmo EUCLIDES é proporcional ao número de chamadas recursivas.

Suponha que a função EUCLIDES faz k chamadas recursivas e que na 1a. chamada ao algoritmo tem-se que $a \geq b > 0$.

Sejam

$$(a, b) = (a_0, b_0), (a_1, b_1), \ldots (a_k, b_k) = (\text{mdc}(a, b), 0)$$

os valores dos parâmetros no início de cada chamada.

Portanto, que $a_{i+1} = b_i$ e $b_{i+1} = a_i \bmod b_i$ para $i = 1, 2, \ldots, k$.

Número de chamadas recursivas

Para números inteiros p e q, $p \geq q > 0$ vale que

$$p \bmod q < p/2$$

Desta forma,

$$\begin{array}{ccccccccc}
b_2 & = & a_1 \bmod b_1 & = & b_0 \bmod b_1 & < & b_0/2 & = & b/2^1 \\
b_4 & = & a_3 \bmod b_3 & = & b_2 \bmod b_3 & < & b_2/2 & \leq & b/2^2 \\
b_6 & = & a_5 \bmod b_5 & = & b_4 \bmod b_5 & < & b_4/2 & \leq & b/2^3 \\
b_8 & = & a_7 \bmod b_7 & = & b_6 \bmod b_7 & < & b_6/2 & \leq & b/2^4 \\
\vdots & = & \vdots & = & \vdots & < & \vdots & \leq & \vdots
\end{array}$$

O valor do 2o. parâmetro é reduzido a menos da sua metade a cada 2 chamadas recursivas.

Número de chamadas recursivas

Seja t o número inteiro tal que

$$2^t \leq b < 2^{t+1},$$

ou seja, $t = \lfloor \lg b \rfloor$.

Da desigualde estrita concluímos que o número de chamadas recursivas é $\leq 2\lfloor \lg b \rfloor + 1$.

Por exemplo, para $a = 514229$ e $b = 317511$ temos que

$$2\lg(b) + 1 = 2\lg(317511) + 1 < 2 \times 18.3 + 1 = 37.56$$

e o número de chamadas recursivas feitas por EUCLIDES $(514229, 317511)$ é 27.

Conclusões

O consumo de tempo do algoritmo EUCLIDES é $\mathrm{O}(\lg b)$.

Seja $\beta := \langle b \rangle$ o número de bits ou tamanho de b.

O consumo de tempo do algoritmo EUCLIDES é $\mathrm{O}(\beta)$.

Se o valor de β dobra, o consumo de tempo pode dobrar.

Se o tamanho de b dobra, o consumo de tempo pode dobrar.

$b \times \lg b$

b	$\lfloor \lg b \rfloor$
4	2
5	2
6	2
10	3
64	6
100	6
128	7
1000	9
1024	10
1000000	19
1000000000	29
$\vdots$	$\vdots$

Números de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$

n	0	1	2	3	4	5	6	7	8	9
F_n	0	1	1	2	3	5	8	13	21	34

Algoritmo recursivo para F_n:

FIBO-REC (n)
1 **se** $n \leq 1$
2 **então devolva** n
3 **senão** $a \leftarrow$ FIBO-REC $(n-1)$
4 $b \leftarrow$ FIBO-REC $(n-2)$
5 **devolva** $a + b$

Euclides e Fibonacci

Os números de Fibonacci estão intimamente relacionados com o algoritmo de Euclides.

Verifique que

A chamada EUCLIDES (F_{k+2}, F_{k+1}) faz k chamadas recursivas.

Verifique ainda que

Se $a > b \geq 0$ e se EUCLIDES (a, b) faz $k \geq 1$ chamadas recursivas, então

$$a \geq F_{k+2} \quad \text{e} \quad b \geq F_{k+1}.$$

Euclides e Fibonacci

Uma conseqüência imediata do fato anterior é:

> Para todo inteiro $k \geq 1$, se $a > b \geq 0$ e $b < F_{k+1}$, então EUCLIDES (a, b) faz $\leq k - 1$ chamadas recursivas.

Exemplo:
$F_{29} = 514229$ e $F_{28} = 317811$ e e Euclides(514229,317811) faz 27 chamadas recursivas.

Razão Aurea

Suponha que um passarinho me contou que para todo $t \geq 2$ vale que

$$\phi^{t-2} \leq F_t < \phi^{t-1},$$

onde $\phi = (1+\sqrt{5})/2$ que é um número entre 1.618 e 1.619.

Exemplo:

$$\phi^{26} < 1.619^{26} < 275689 < F_{28} = 317811 < 438954 < 1.618^{27} < \phi^{27}$$

Verifique que $1 + \phi = \phi^2$.

Número de chamadas (novamente)

Suponha $b \geq 4$. Se t é o número inteiro tal que

$$\phi^{t-2} \leq b < \phi^{t-1} \leq F_{t+1}$$

então o número k de chamadas recursivas de EUCLIDES (a, b) é no máximo

$$t - 1 \leq (2 + \log_\phi b) - 1 = 1 + \log_\phi b.$$

Exemplo: Segundo está nova estimativa temos que EUCLIDES$(514229, 317811)$ faz no máximo

$$1 + \log_\phi(317811) < 1 + \log_{1.619}(317811) < 1 + 26.28 = 27.45$$

e o número de chamadas é 27.
[Uauuuu. Isto foi muito perto! Talvez tenha alguma conta errada.]

Conclusão

Se k é número de chamadas recursivas feitas por EUCLIDES (a, b) então

$$k \leq 1 + \log_\phi b, \text{ onde } \phi = (1 + \sqrt{5})/2.$$

Certificados

O algoritmo EUCLIDES pode ser facilmente modificado para nos fornecer números inteiros x e y tais que

$$ax + by = \text{mdc}(a, b).$$

Certificados

O algoritmo EUCLIDES pode ser facilmente modificado para nos fornecer números inteiros x e y tais que

$$ax + by = \text{mdc}(a, b).$$

E daí? Qual a graça nisto?

Certificados

O algoritmo EUCLIDES pode ser facilmente modificado para nos fornecer números inteiros x e y tais que

$$ax + by = \text{mdc}(a, b).$$

E daí? Qual a graça nisto?

Os números x e y são uma prova ou certificado da correção da resposta!

Certificados

Suponha que EUCLIDES (a, b) devolva d junto com x, y tais que $ax + by = d$

- podemos verificar se $d \mid a$ e $d \mid b$
- podemos verificar se $ax + by = d$

Se $d' \mid a$ e $d' \mid b$ então

$$d' \mid (ax + by) = d$$

e portanto $d' \leq d$

Conclusão: $d = \text{mdc}(a, b)$

Extended-Euclides

Recebe números inteiros não-negativos a e b e devolve números inteiros d, x e y tais que $d \mid a$, $d \mid b$ e $a\,x + b\,y = d$.

EXTENDED- EUCLIDES (a, b) $\triangleright$ supõe $a \neq 0$ ou $b \neq 0$

1 **se** $b = 0$

2 **então devolva** $(a, 1, 0)$

3 $(d', x', y') \leftarrow$ EXTENDED- EUCLIDES $(b, a \bmod b)$

4 $(d, x, y) \leftarrow (d', y', x' - \lfloor a/b \rfloor y')$

5 **devolva** (d, x, y)

Self-certifying algorithms

Algoritmos que devolvem certificados da sua correção são chamados *self-certifying*.

Exemplos:

- EXTENDED- EUCLIDES
- Algoritmo de Dijkstra (caminhos mínimos)
- Algoritmo de Ford e Fulkerson (fluxos em redes)
- Algoritmos para programação linear
  devolvem soluções do problema primal e dual

Na página pessoal de Kurt Mehlhorn há um link para uma palestra sobre *Certifying Algorithms*.
Cópia local:

`http://www.ime.usp.br/~coelho/mac0338-2007/aulas/CertifyingAlgs.pdf`

Complexidade computacional: P versus NP

CLR 36 ou CLRS 34

Complexidade computacional

Classifica os problemas em relação à dificuldade de resolvê-los algoritmicamente.

Disciplina:

MAC5722 Complexidade Computacional

Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma seqüência de símbolos retirados de algum alfabeto.

Este alfabeto pode ser, por exemplo, o conjunto de símbolos ASCII ou o conjunto $\{0, 1\}$.

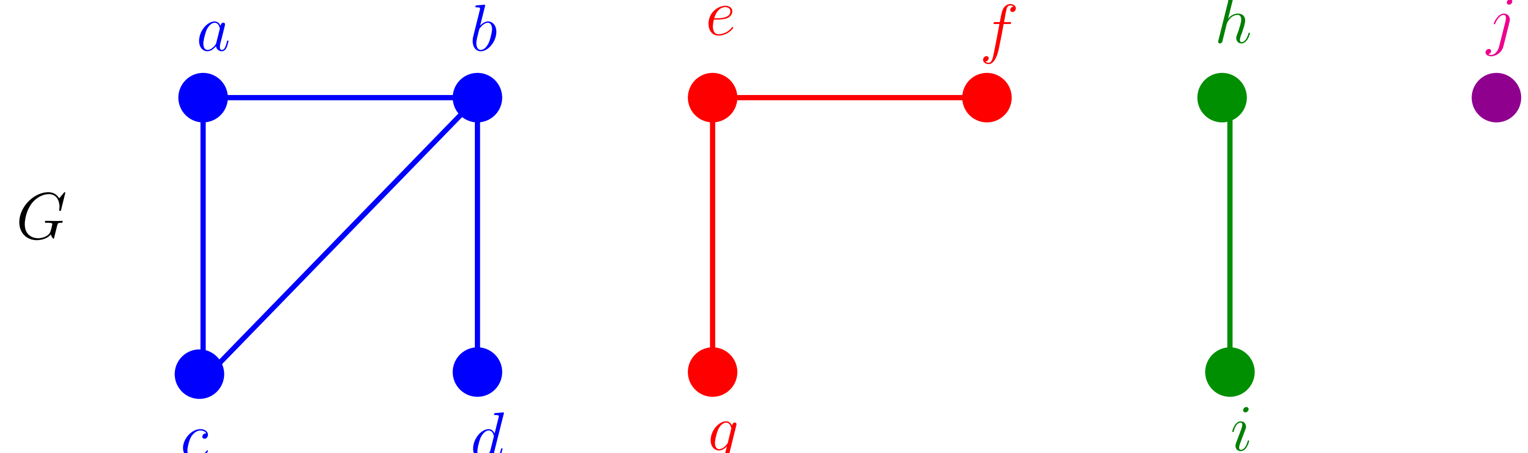
Qualquer seqüência dos elementos de um alfabeto é chamada de uma palavra.

Não é difícil codificar objetos tais como racionais, vetores, matrizes, grafos e funções como palavras.

O tamanho de uma palavra w, denotado por $\langle w \rangle$ é o número de símbolos usados em w, contando multiplicidades. O tamanho do racional '$123/567$' é 7.

Exemplo 1

Grafo



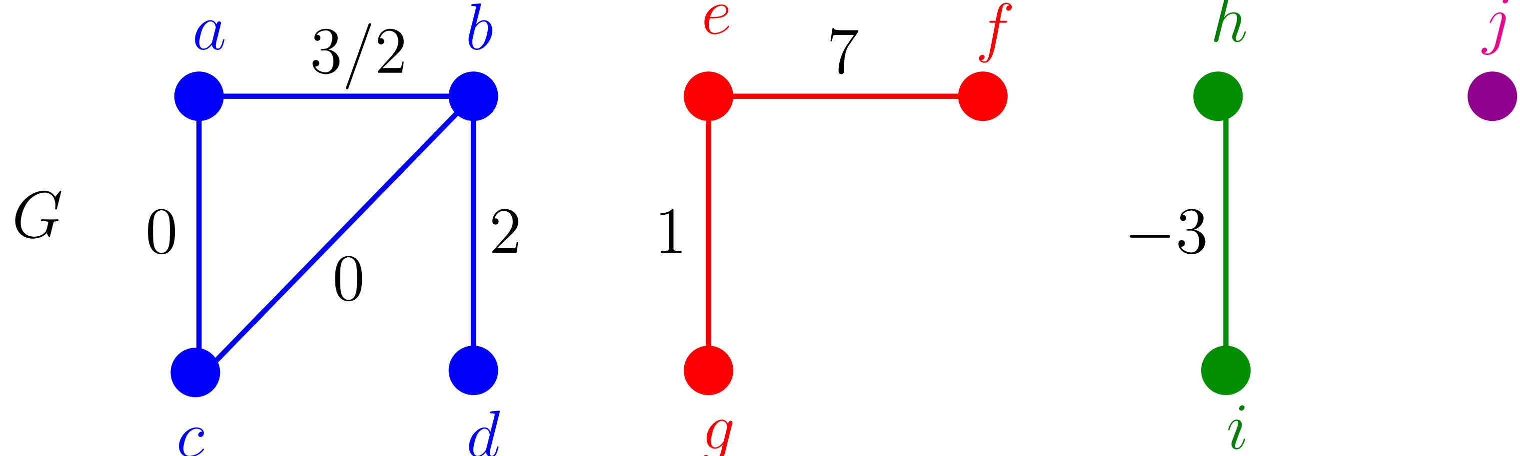
Palavra:

$(\{a, b, c, d, e, f, g, h, i, j\}, \{\{bd\}, \{eg\}, \{ac\}, \{hi\}, \{ab\}, \{ef\}, \{bc\}\})$

Tamanho da palavra: 59

Exemplo 2

Função



Palavra:

$$((\{bd\}, 2), (\{eg\}, 1), (\{ac\}, 0), (\{hi\}, -3), (\{ab\}, 3/2), (\{ef\}, 7), (\{bc, 0)\}\}$$

Tamanho da palavra: 67

Tamanho de uma palavra

Para os nossos propósitos, não há mal em subestimar o tamanho de um objeto.

Não é necessário contar rigorosamente os caracteres '{', '}', '(', ')' e ',' dos exemplos anteriores.

Tamanho de um inteiro p é essencialmente $\lg |p|$.

Tamanho do racional p/q é, essencialmente, $\lg |p| + \lg |q|$.

Tamanho de um vetor $A[1\,.\,.\,n]$ é a soma dos tamanhos de seus componentes

$$\langle A \rangle = \langle A[1] \rangle + \langle A[2] \rangle + \cdots + \langle A[n] \rangle.$$

Problemas e instâncias

Cada conjunto específico de dados de um problema define uma instância.

Tamanho de uma instância é o tamanho de uma palavra que representa a instância.

Problema que pede uma resposta do tipo SIM ou NÃO é chamado de problema de decisão.

Problema que procura um elemento em um conjunto é um problema de busca.

Problema que procura um elemento de um conjunto de soluções viáveis que seja melhor possível em relação a algum critério é um problema de otimização

Máximo divisor comum

Problema: Dados dois números inteiros não-negativos a e b, determinar mdc(a, b).

Exemplo:
máximo divisor comum de 30 e 24 é 6
máximo divisor comum de 514229 e 317811 é 1
máximo divisor comum de 3267 e 2893 é 11

Problema de busca

Instância: a e b

Tamanho da instância: $\langle a \rangle + \langle b \rangle$, essencialmente

$$\lg a + \lg b$$

Consumo de tempo do algoritmo Café-Com-Leite é $\mathrm{O}(b)$.
Consumo de tempo do algoritmo EUCLIDES é $\mathrm{O}(\lg b)$.

Máximo divisor comum (decisão)

Problema: Dados dois números inteiros não-negativos a, b e k, mdc$(a, b) = k$?

Exemplo:
máximo divisor comum de 30 e 24 é 6
máximo divisor comum de 514229 e 317811 é 1
máximo divisor comum de 3267 e 2893 é 11

Problema de decisão: resposta SIM ou NÃO

Instância: a, b, k

Tamanho da instância: $\langle a\rangle + \langle b\rangle + \langle k\rangle$, essencialmente

$$\lg a + \lg b + \lg k$$

Subseqüência comum máxima

Problema: Encontrar uma ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco máxima = B C A B

Problema de otimização

Instância: $X[1\,..\,m]$ e $Y[1\,..\,n]$

Tamanho da instância: $\langle X\rangle + \langle Y\rangle$, essencialmente

$$n + m$$

Consumo de tempo REC-LCS-LENGTH é $\Omega(2^{\min\{m,n\}})$.
Consumo de tempo LCS-LENGTH é $\Theta(mn)$.

Subseqüência comum máxima (decisão)

Problema: $X[1\,..\,m]$ e $Y[1\,..\,n]$ possuem uma ssco máxima $\geq k$?

Exemplo: X = A B C B D A B

Y = B D C A B A

ssco máxima = B C A B

Problema de decisão: resposta SIM ou NÃO

Instância: $X[1\,..\,m]$, $Y[1\,..\,n]$, k

Tamanho da instância: $\langle X\rangle + \langle Y\rangle + \langle k\rangle$, essencialmente

$$n + m + \lg k$$

Problema booleano da mochila

Problema (Knapsack Problem): Dados n, $w[1\,..\,n]$ $v[1\,..\,n]$ e W, encontrar uma mochila boolena ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	0	1	1	0

valor $= 1000$

Problema de otimização

Instância: n, $w[1\,..\,n]$ $v[1\,..\,n]$ e W

Tamanho da instância: $\langle n\rangle + \langle w\rangle + \langle v\rangle + \langle W\rangle$, essencialmente $\lg n + n \lg W + n \lg V + \lg W$

Consumo de tempo MOCHILA-BOOLEANA é $\Theta(nW)$.

Problema booleano da mochila (decisão)

Problema (Knapsack Problem): Dados n, $w[1 \,..\, n]$ $v[1 \,..\, n]$ e W e k, existeuma mochila boolena de valor $\geq k$.

Exemplo: $W = 50$, $n = 4$, $k = 1010$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	0	1	1	0

valor $= 1000$

Problema de decisão: resposta SIM ou NÃO

Instância: n, $w[1 \,..\, n]$ $v[1 \,..\, n]$, W e k

Tamanho da instância: $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle + \lg k$, essencialmente $\lg n + n \lg W + n \lg V + \lg W + \lg k$

Problema fracionário da mochila

Problema: Dados n, $w[1\,..\,n]$ $v[\,..\,n]$ e W, encontrar uma mochila ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	1	1/3	0	0

valor $= 1040$

Problema de otimização

Instância: n, $w[1\,..\,n]$ $v[1\,..\,n]$ e W
Tamanho da instância: $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle$, essencialmente $\lg n + n \lg W + n \lg V + \lg W$
Consumo de tempo MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$.

Problema fracionário da mochila (decisão)

Problema: Dados n, $w[1\,.\,.\,n]$ $v[\,.\,.\,n]$, W e k, existe uma mochila de valor $\geq k$?

Exemplo: $W = 50$, $n = 4$, $k = 1010$

	1	2	3	4
w	40	30	20	10
v	840	600	400	100
x	1	1/3	0	0

valor $= 1040$

Problema de decisão: resposta SIM ou NÃO

Instância: n, $w[1\,.\,.\,n]$ $v[1\,.\,.\,n]$ e W
Tamanho da instância: $\langle n\rangle + \langle w\rangle + \langle v\rangle + \langle W\rangle$, essencialmente $\lg n + n \lg W + n \lg V + \lg W$

Modelo de computação

É uma descrição abstrata e conceitual de um computador que será usado para executar um algoritmo.

Um modelo de computação especifica as operações elementares um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome.

Operações elementares típicas são operações aritméticas entre números e comparações.

No critério uniforme supõe-se que cada operação elementar consome uma quantidade de tempo constante.

Problemas polinomiais

Análise de um algoritmo em um determinado modelo de computação estima o seu consumo de tempo e quantidade de espaço como uma função do tamanho da instância do problema.

Exemplo: o consumo de tempo do algoritmo EUCLIDES (a, b) é expresso como uma função de $\langle a \rangle + \langle b \rangle$.

Um problema é solúvel em tempo polinomial se existe um algoritmo que consome tempo $\mathrm{O}(\langle I \rangle^c)$ para resolver o problema, onde c é uma constante e I é instância do problema.

Exemplos

- Máximo divisor comum

  Tamanho da intância: $\lg a + \lg b$

  Consumo de tempo Café-Com-Leite é $O(b)$ (não-polinomial)

  Consumo de tempo EUCLIDES é $O(\lg b)$ (polinomial)

- Subseqüência comum máxima

  Tamanho da instância: $n + m$

  Consumo de tempo REC-LEC-LENGTH é $\Omega(2^{\min\{m,n\}})$ (exponencial)

  Consumo de tempo LEC-LENGTH é $\Theta(mn)$ (polinomial).

Mais exemplos

- Problema booleano da mochila

  Tamanho da instância: $\lg n + n \lg W + n \lg V + \lg W$
  Consumo de tempo MOCHILA-BOOLEANA é $\Theta(nW)$ (não-polinomial)

- Problema fracionário da mochila
  Tamanho da instância: $\lg n + n \lg W + n \lg V + \lg W$
  Consumo de tempo MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$ (polinomial).

- Ordenação de inteiros $A[1\,..\,n]$

  Tamanho da instância: $n \lg M$,
  $M := \max\{|A[1]|, |A[2]|, \ldots, |A[n]|\} + 1$
  Consumo de tempo MERGE-SORT é $\Theta(n \lg n)$ (polinomial).

Classe P

Por um algoritmo eficiente entendemos um algoritmo polinomial.

A classe de todos os problemas de decisão que podem ser resolvidos por algoritmos polinomiais é denotada por P (classe de complexidade).

Exemplo: As versões de decisão dos problemas:

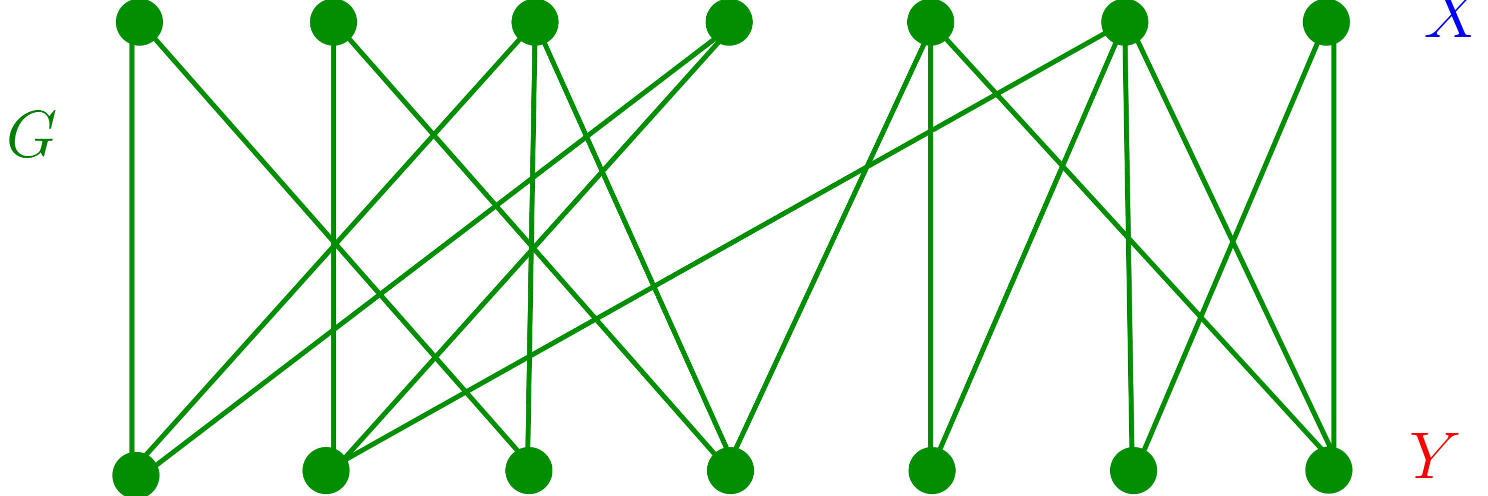
> máximo divisor comum, subseqüência comum máxima e mochila fracionária

estão em P.

Para muitos problemas, não se conhece algoritmo essencialmente melhor que “testar todas as possibilidades”. Em geral, isso não está em P.

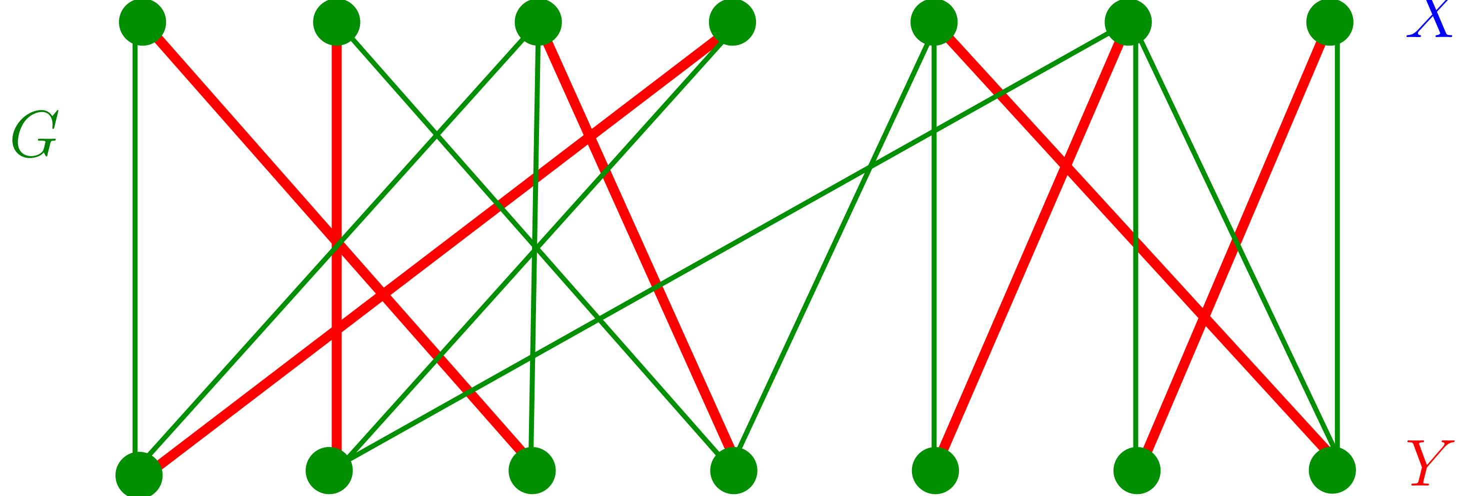
Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



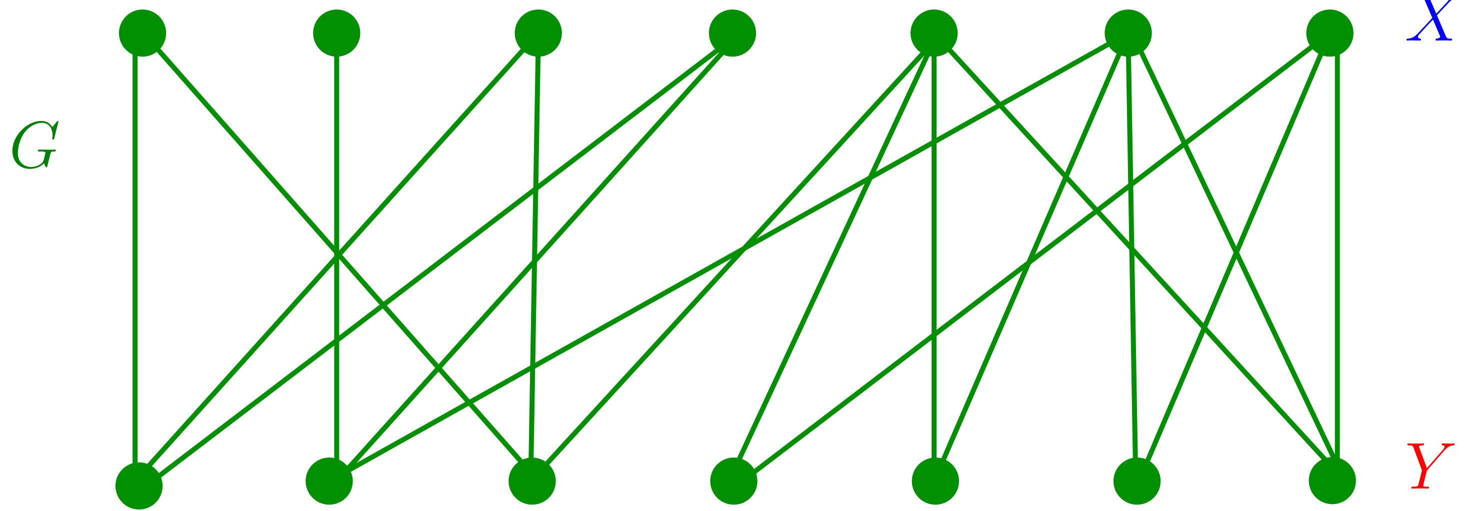
Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



Emparelhamentos

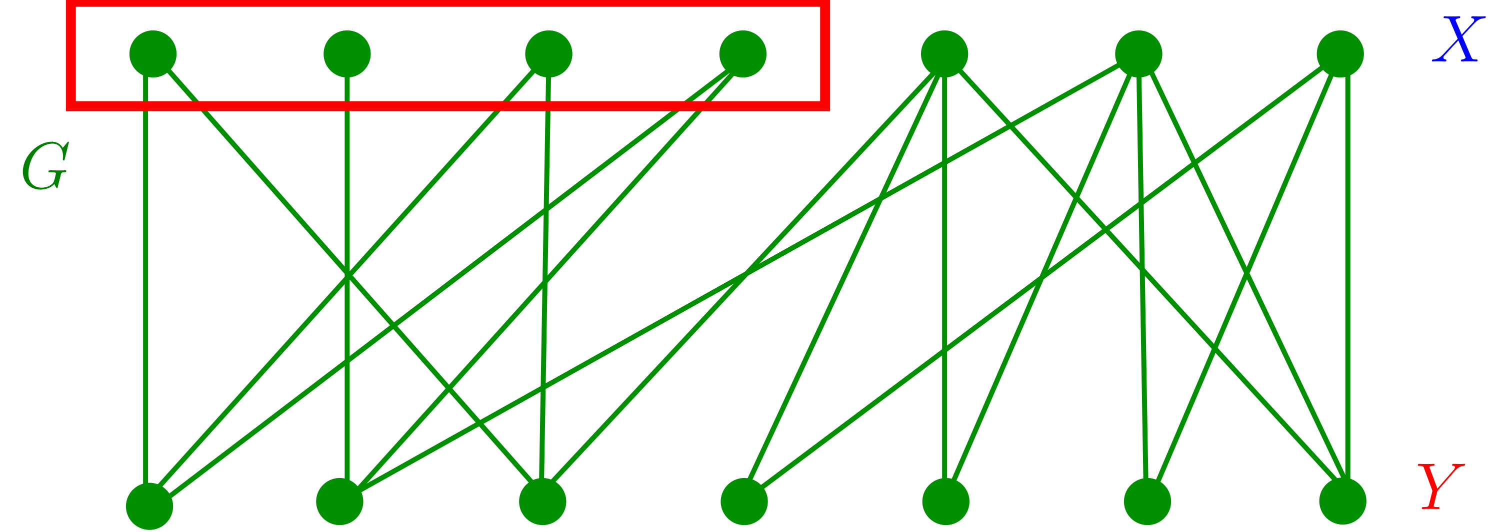
Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



NÃO existe! Certificado?

Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento bipartido.



NÃO existe! Certificado: $S \subseteq X$ tal que $|S| > |\text{vizinhos}(S)|$.

Melhores momentos

AULA 24

Problemas polinomiais

Analise de um algoritmo em um determinado modelo de computação estima o seu consumo de tempo e quantidade de espaço como uma função do tamanho da instância do problema.

Exemplo: o consumo de tempo do algoritmo EUCLIDES (a, b) é expresso como uma função de $\langle a \rangle + \langle b \rangle$.

Um problema é solúvel em tempo polinomial se existe um algoritmo que consome tempo $\mathrm{O}(\langle I \rangle^{c})$ para resolver o problema, onde c é uma constante e I é instância do problema.

Exemplos

- Máximo divisor comum

  Tamanho da intância: $\lg a + \lg b$

  Consumo de tempo Café-Com-Leite é $O(b)$ (não-polinomial)

  Consumo de tempo EUCLIDES é $O(\lg b)$ (polinomial)

- Subseqüência comum máxima

  Tamanho da instância: $n + m$

  Consumo de tempo REC-LEC-LENGTH é $\Omega(2^{\min\{m,n\}})$ (exponencial)

  Consumo de tempo LEC-LENGTH é $\Theta(mn)$ (polinomial).

Mais exemplos

- Problema booleano da mochila

  Tamanho da instância: $\lg n + n \lg W + n \lg V + \lg W$
  Consumo de tempo MOCHILA-BOOLEANA é $\Theta(nW)$ (não-polinomial)

- Problema fracionário da mochila
  Tamanho da instância: $\lg n + n \lg W + n \lg V + \lg W$
  Consumo de tempo MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$ (polinomial).

- Ordenação de inteiros $A[1 \mathinner{.\,.} n]$

  Tamanho da instância: $n \lg M$,
  $M := \max\{|A[1]|, |A[2]|, \ldots, |A[n]|\} + 1$
  Consumo de tempo MERGE-SORT é $\Theta(n \lg n)$ (polinomial).

Classe P

Por um algoritmo eficiente entendemos um algoritmo polinomial.

A classe de todos os problemas de decisão que podem ser resolvidos por algoritmos polinomiais é denotada por P (classe de complexidade).

Exemplo: As versões de decisão dos problemas:

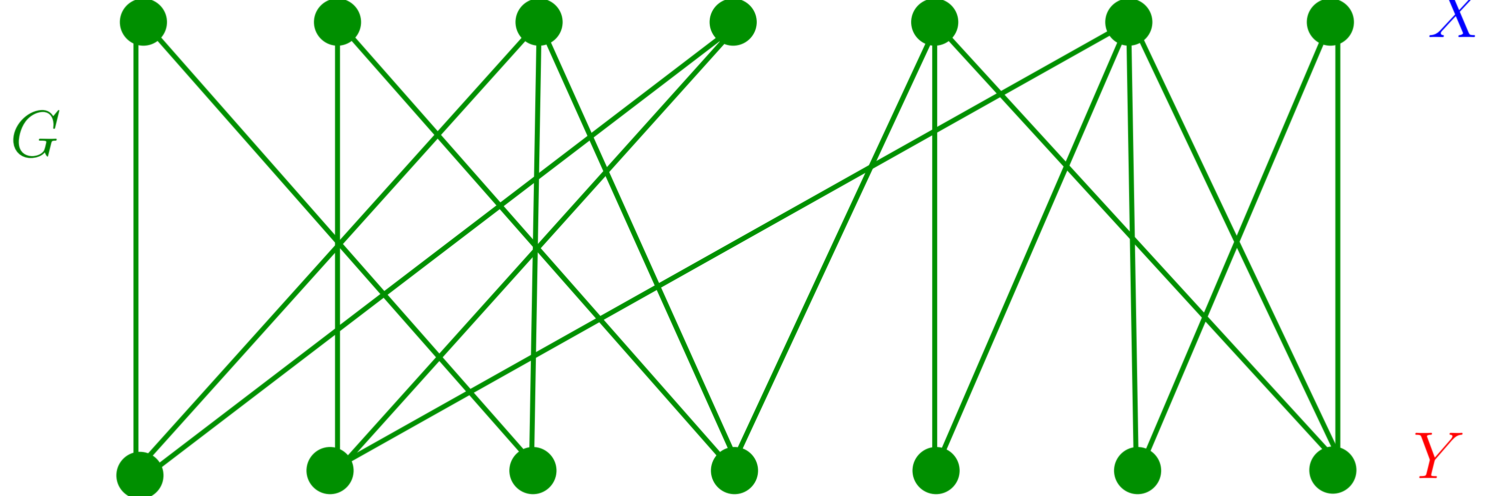
> máximo divisor comum, subseqüência comum máxima e mochila fracionária

estão em P.

Para muitos problemas, não se conhece algoritmo essencialmente melhor que "testar todas as possibilidades". Em geral, isso não está em P.

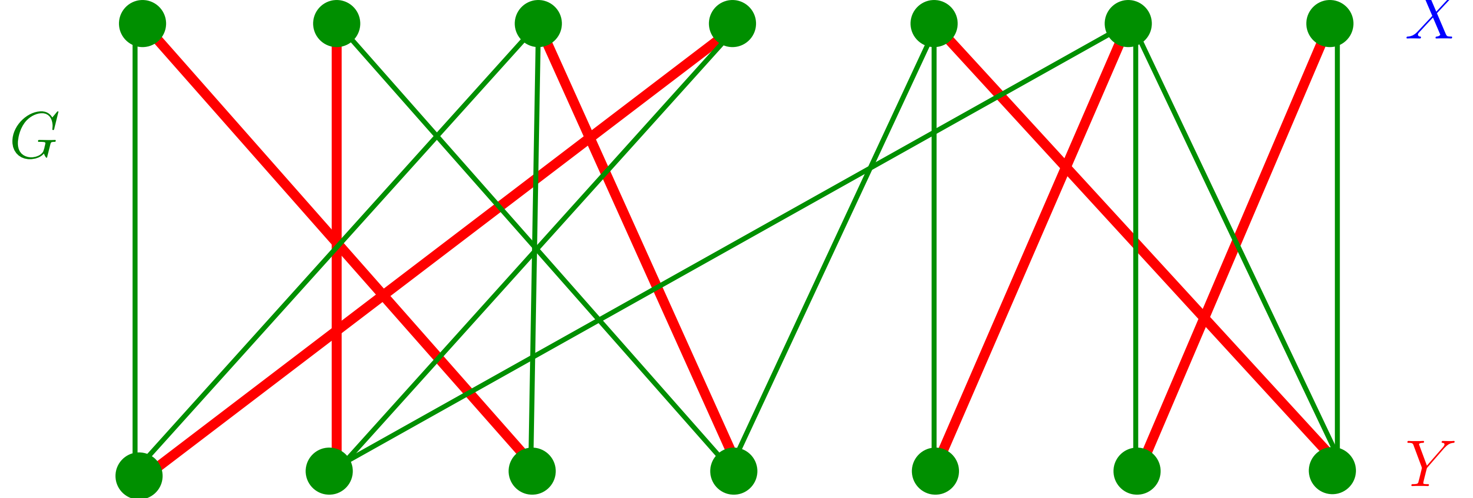
Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



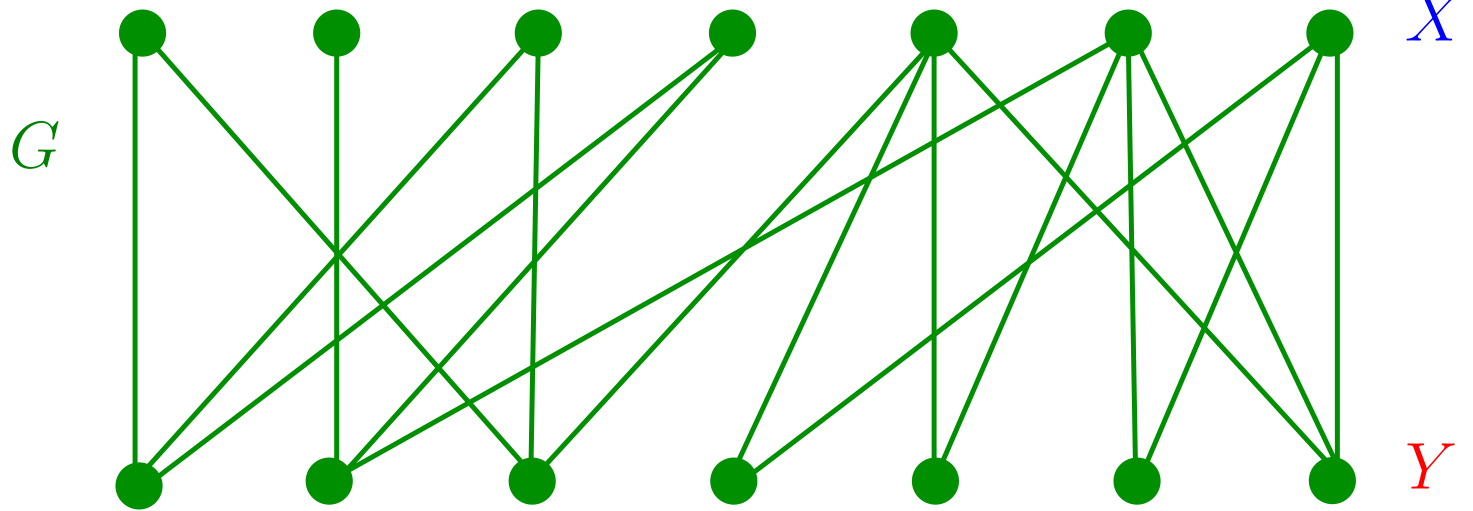
Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



Emparelhamentos

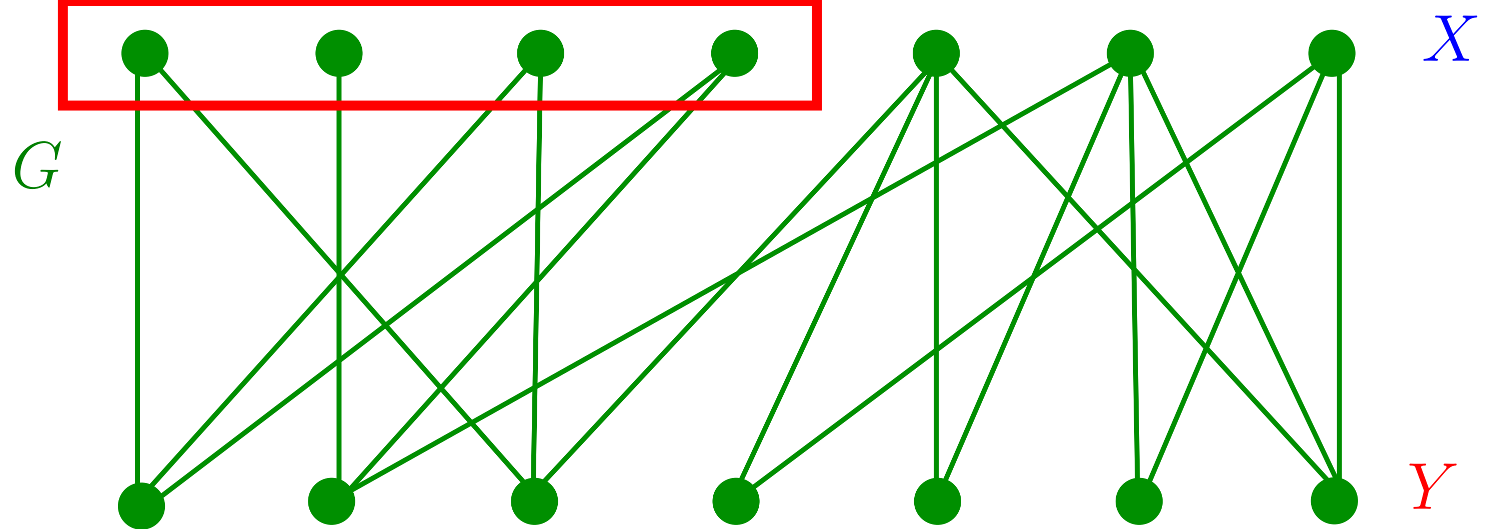
Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



NÃO existe! Certificado?

Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



NÃO existe! Certificado: $S \subseteq X$ tal que $|S| > |\text{vizinhos}(S)|$.
Teorema de Hall: G tem um emparelhamento perfeito se e somente se

$$|S| \leq |\text{vizinhos}(S)|, \quad \text{para todo } S \subseteq X.$$

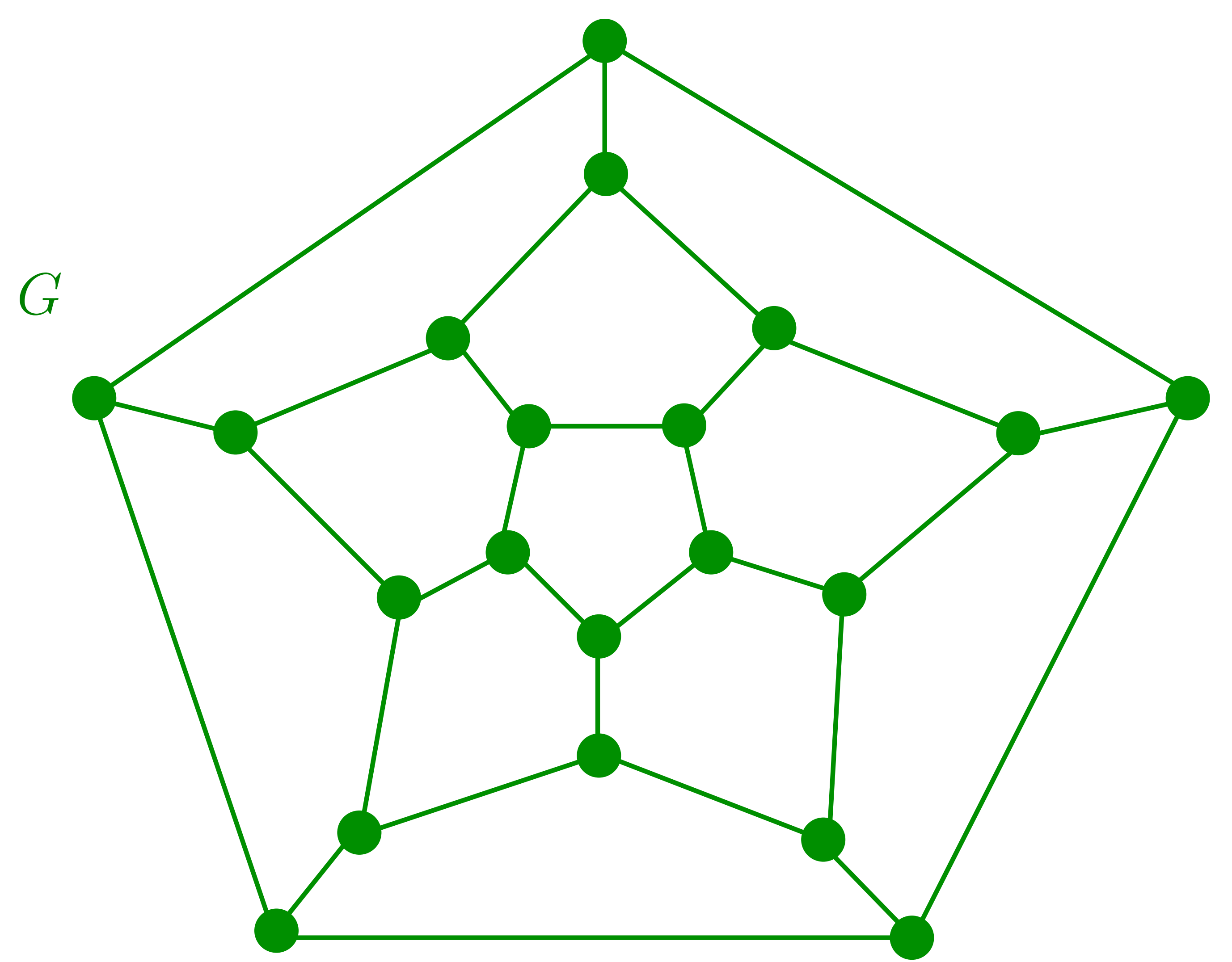
AULA 25

Mais complexidade computacional

CLR 36 ou CLRS 34

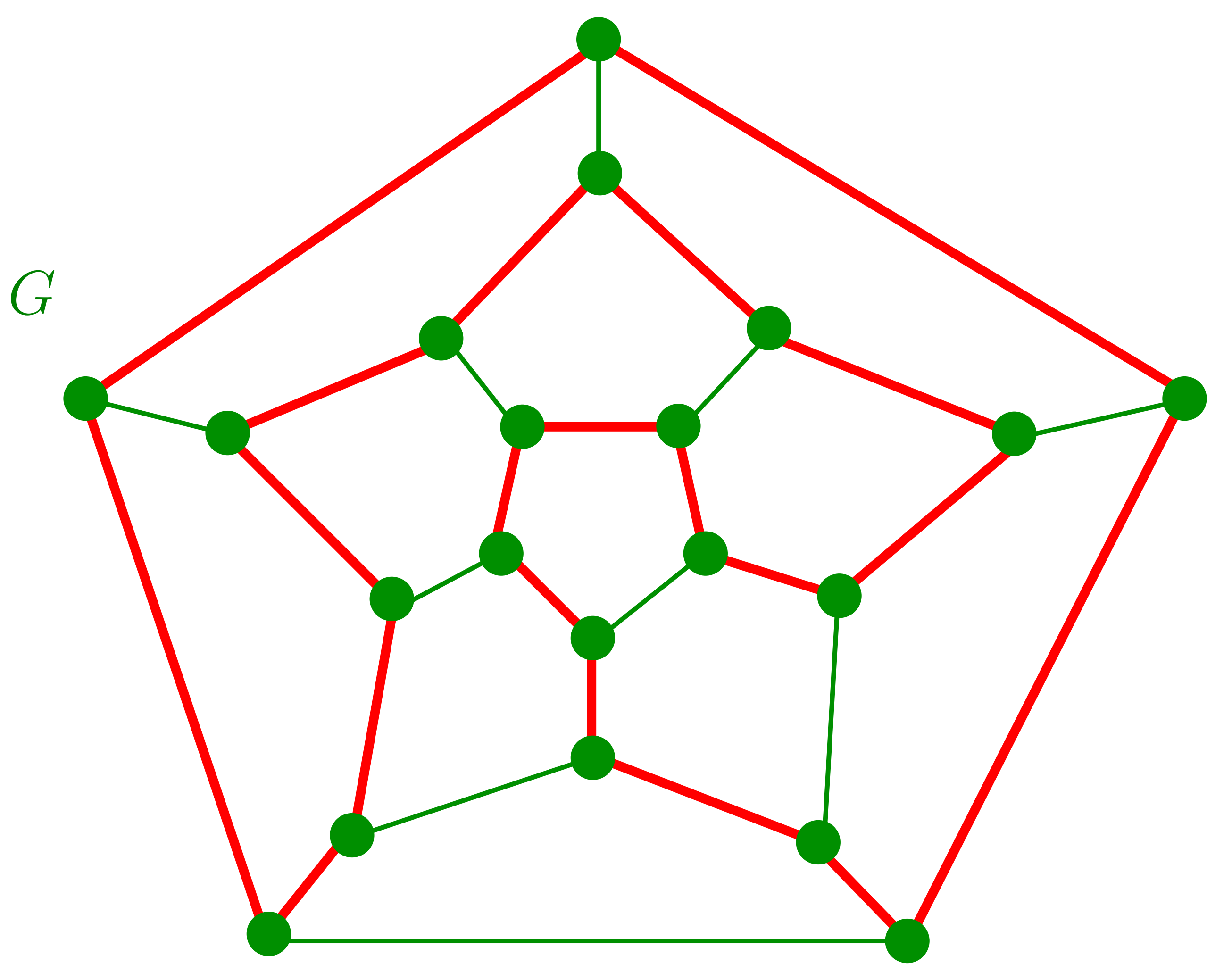
Grafos hamiltonianos

Problema: Dado um grafo encontrar um ciclo hamiltoniano.



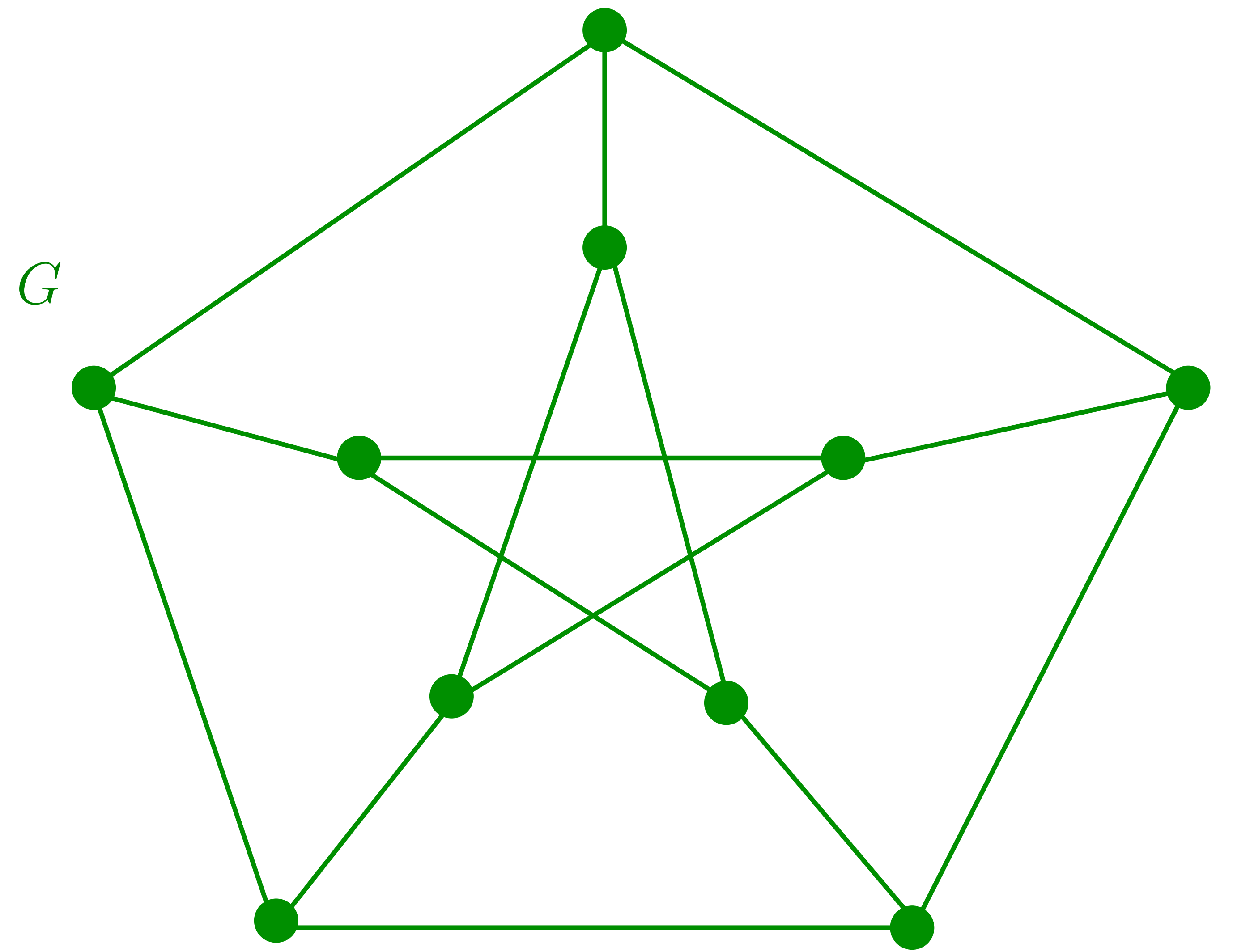
Grafos hamiltonianos

Problema: Dado um grafo encontrar um ciclo hamiltoniano.



Grafos hamiltonianos

Problema: Dado um grafo encontrar um ciclo hamiltoniano.



NÃO existe! Certificado? Hmmm . . .

Verificador polinomial para SIM

Um verificador polinomial para a resposta SIM a um problema Π é uma algoritmo polinomial ALG que recebe

> uma instância I de Π e um objeto C, tal que $\langle C \rangle$ é $\mathrm{O}(\langle I \rangle^c)$ para alguma constante c

e

> ALG$(I, C) =$ SIM se e somente se a resposta a $\Pi(I)$ é SIM.

O objeto C é dito um certificado polinomial ou certificado curto da resposta SIM a $\Pi(I)$.

Exemplos

- Se G é hamiltoniano,então um ciclo hamiltoniano de G é um certificado polinomial:

  dados um grafo G e C pode-se verificar em tempo $\mathrm{O}(\langle G \rangle)$ se C é um ciclo hamiltoniano.

- se $X[1\,.\,.\,m]$ e $Y[1\,.\,.\,n]$ possuem uma ssco $\geq k$, então uma subseqüência comum $Z[1\,.\,.\,k]$ é um certificado polinomial resposta:

  dados $X[1\,.\,.\,m], Y[1\,.\,.\,n]$ e $Z[1\,.\,.\,k]$ pode-se verificar em tempo $\mathrm{O}(m+n)$ se Z é ssco de X e Y.

- se n é um número composto, então um divisor $d > 1$ de n é um certificado polinomial.

Classe NP

Formanda pelos problemas de decisão que possuem um verificador polinomial para a resposta SIM.

Em outras palavras, a classe NP é formada pelos problemas de decisão Π para os quais existe um problema Π' em P e uma função polinomial $p(n)$ tais que, para cada instância I do problema Π, existe um objeto C com $\langle C \rangle \leq p(\langle I \rangle)$ tal que a

> resposta a $\Pi(I)$ é SIM se e somente se a resposta a $\Pi'(I, C)$ é SIM.

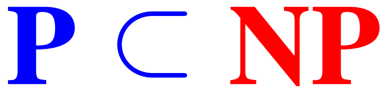
O objeto C é dito um certificado polinomial ou certificado curto da resposta SIM a $\Pi(I)$.

Exemplos

Problemas de decisão com certificado polinomial para SIM:

- existe subseqüência crescente $\geq k$?
- existe subcoleção disjunta $\geq k$ de intervalos?
- existe mochila booleana de valor $\geq k$?
- existe mochila de valor $\geq k$?
- existe subseqüência comum $\geq k$?
- grafo tem ciclo de comprimento $\geq k$?
- grafo tem ciclo hamiltoniano?
- grafo tem emparelhamento (casamento) perfeito?

Todos esses problemas estão em NP.



$\mathbf{P} \subseteq \mathbf{NP}$

Prova:

se Π é um problema em P, então pode-se tomar a seqüência de instruções realizadas por um algoritmo polinomial para resolver $\Pi(I)$ como certificado polinomial da resposta SIM a $\Pi(I)$.

Outra prova:

Pode-se construir um verificador polinomial para a resposta SIM a Π utilizando-se um algoritmo polinomial para Π como subrotina e ignorando-se o certificado C.

$\mathbf{P} \neq \mathbf{NP}$?

É crença de muitos que a classe NP é maior que a classe P, ainda que isso

não tenha sido provado até agora.

Este é o intrigante problema matemático conhecido pelo rótulo "P $\neq$ NP?"

Não confunda NP com "não-polinomial".

Verificado polinomial para NÃO

Um verificador polinomial para a resposta NÃO de um problema Π é uma algoritmo polinomial ALG que recebe

> uma instância I de Π e um objeto C, tal que $\langle C \rangle$ é $\mathrm{O}(\langle I \rangle^c)$ para alguma constante c

e devolve

> SIM se e somente se a resposta a $\Pi(I)$ é NÃO, em caso contrário ALG devolve NÃO.

O objeto C é dito um certificado polinomial ou certificado curto da resposta NÃO a $\Pi(I)$.

Classe co-NP

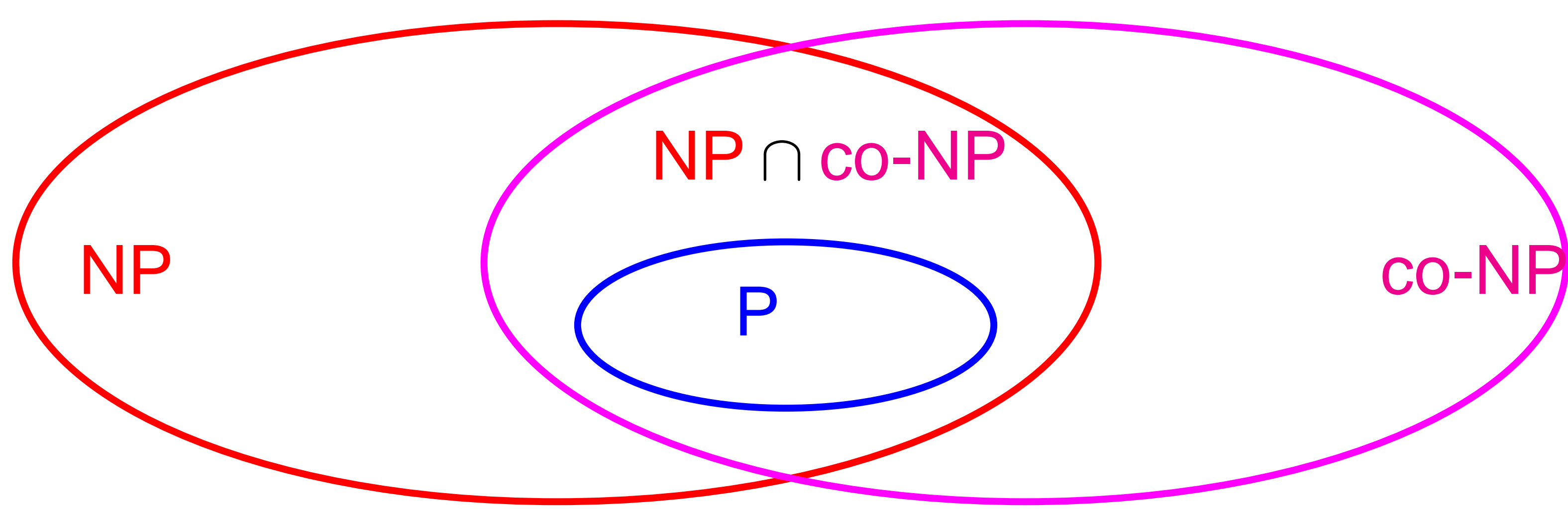
A classe co-NP é definida trocando-se SIM por NÃO na definição de NP.

Um problema de decisão Π está em co-NP se admite um certificado polinomial para a resposta NÃO.

Os problemas em NP $\cap$ co-NP admitem certificados polinomiais para as respostas SIM e NÃO.

Em particular, P $\subseteq$ NP $\cap$ co-NP.

P, NP e co-NP



P $\neq$ NP?

NP $\cap$ co-NP $\neq$ P?

NP $\neq$ co-NP?

Redução polinomial

Permite comparar o “grau de complexidade” de problemas diferentes.

Uma redução de um problema Π a um problema Π' é um algoritmo ALG que resolve Π usando uma subrotina hipotética ALG$'$ que resolve Π', de tal forma que, se ALG$'$ é um algoritmo polinomial, então ALG é um algoritmo polinomial.

$\Pi \leq_P \Pi' =$ existe uma redução de Π a Π'.

Se $\Pi \leq_P \Pi'$ e Π' está em P, então Π está em P.

Exemplo

Π = encontrar um ciclo hamiltoniano
Π' = existe um ciclo hamiltoniano?

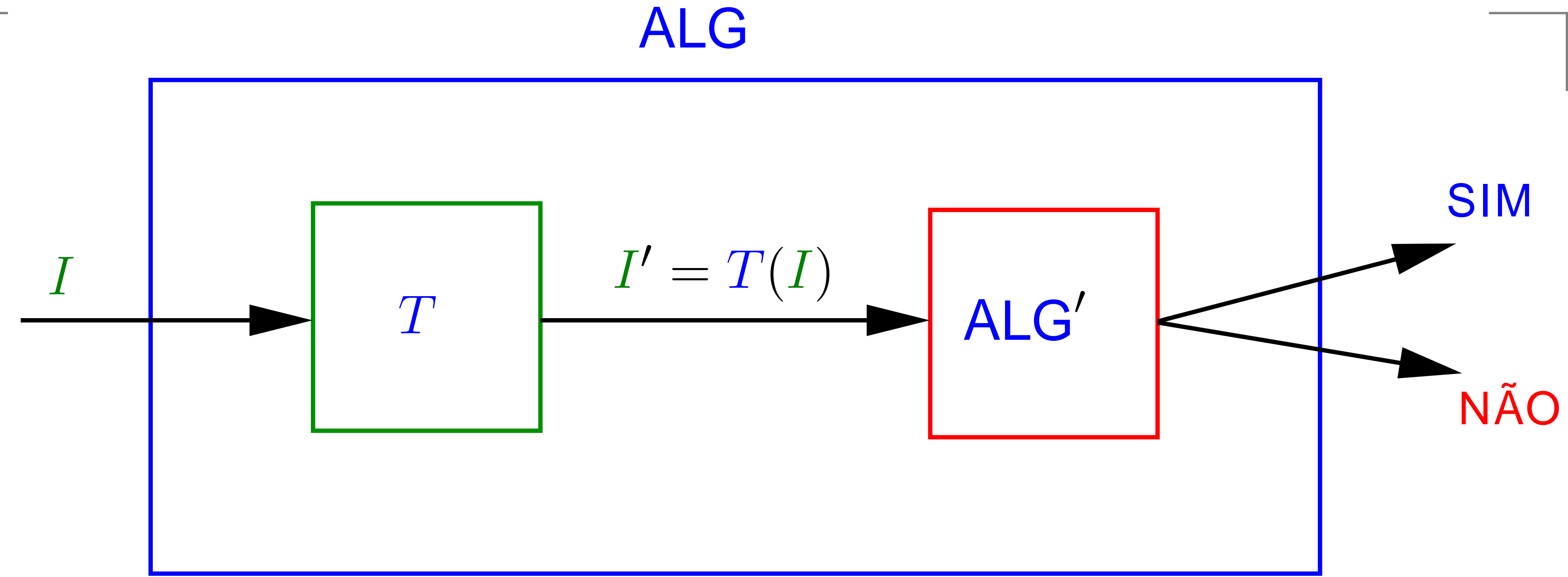
Redução de Π a Π': ALG$'$ é um algoritmo que resolve Π'

ALG (G)

1 **se** ALG$'(G) =$ NÃO
2 **então devolva** "G não é hamiltoniano"
3 **para cada** aresta uv de G **faça**
4 $H \leftarrow G - uv$
5 **se** ALG$'(H) =$ SIM
6 **então** $G \leftarrow G - uv$
7 **devolva** G

Se ALG$'$ consome tempo $\mathrm{O}(p(n))$, então ALG consome tempo $\mathrm{O}(m\, p(\langle G \rangle))$, onde m = número de arestas de G.

Esquema comum de redução



Faz apenas uma chamada ao algoritmo ALG$'$.

T transforma uma instância I de Π em uma instância $I' = T(I)$ de Π' tal que

$$\Pi(I) = \text{SIM se e somente se } \Pi'(I') = \text{SIM}$$

T é uma espécie de “filtro” ou “compilador”.

Satisfatibilidade

Problema: Dada um fórmula booleana ϕ nas variáveis $x_1, \ldots, x_n$, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \rightarrow \{\text{VERDADE}, \text{FALSO}\}$$

que torna ϕ verdadeira?

Exemplo:

$$\phi = (x_1) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_3)$$

Se $t(x_1) = \text{VERDADE}, t(x_2) = \text{FALSO}, t(x_3) = \text{FALSO}$, então $t(\phi) = \text{VERDADE}$

Se $t(x_1) = \text{VERDADE}, t(x_2) = \text{VERDADE}, t(x_3) = \text{FALSO}$, então $t(\phi) = \text{FALSO}$

Sistemas lineares 0-1

Problema: Dadas uma matriz A e um vetor b,

$$Ax \geq b$$

possui uma solução tal que $x_i = 0$ ou $x_i = 1$ para todo i?

Exemplo:

$$\begin{array}{rrrrrrrr} & x_1 & & & & & \geq & 1 \\ - & x_1 & - & x_2 & + & x_3 & \geq & -1 \\ & & & & - & x_3 & \geq & 0 \end{array}$$

tem uma solução 0-1?

Sim! $x_1 = 1, x_2 = 0$ e $x_3 = 0$ é solução.

Exemplo 1

Satisfatibilidade $\leq_P$ Sistemas lineares 0-1

A transformação T recebe uma fórmula booleana ϕ
e devolve um sistema linear $Ax \geq b$
tal que ϕ é satisfatível se e somente se o sistema $Ax \geq b$ admite uma solução 0-1.

Exemplo:

$$\phi = (x_1) \land (\neg x_1 \lor \neg x_2 \lor x_3) \land (\neg x_3)$$

$$\begin{array}{ccccccc} x_1 & & & & & \geq & 1 \\ 1 - x_1 & + & 1 - x_2 & + & x_3 & \geq & 1 \\ & & & & 1 - x_3 & \geq & 1 \end{array}$$

Exemplo 2

Verifique que

Ciclo hamiltoniano $\leq_P$ Caminho hamiltoniano entre u e v

Verifique que

Caminho hamiltoniano entre u e v $\leq_P$ Caminho hamiltoniano

Exemplo 3

Caminho hamiltoniano $\leq_P$ Satisfatibilidade

Descreveremos um algoritmo polinomial T que recebe um grafo G e devolve uma fórmula booleana $\phi(G)$ tais que

G tem caminho hamiltoniano $\Leftrightarrow$ $\phi(G)$ é satisfatível.

Suponha que G tem vértices $1, \ldots, n$.

$\phi(G)$ tem n^2 variáveis $x_{i,j}$, $1 \leq i, j \leq n$.

Interpretação: $x_{i,j} =$ VERDADE $\Leftrightarrow$ vértice j é o i-ésimo vértice do caminho.

Exemplo 3 (cont.)

Claúsulas de $\phi(G)$:

- “vértice j faz parte do caminho:

$$(x_{1,j} \vee x_{2,j} \vee \cdots \vee x_{n,j})$$

para cada j (n claúsulas).

- “vértice j não está em duas posições do caminho:

$$(\neg x_{i,j} \vee \neg x_{k,j})$$

para cada j e $i \neq k$ ($\mathrm{O}(n^3)$ claúsulas).

- “algum vértice é o i-ésimo do caminho”:

$$(x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,n})$$

para cada i (n claúsulas).

Exemplo 3 (cont.)

Mais claúsulas de $\phi(G)$:

- "dois vértices não podem ser o i-ésimo":

$$(\neg x_{i,j} \vee \neg x_{i,k})$$

  para cada i e $j \neq k$ ($\mathrm{O}(n^3)$ claúsulas).

- "se ij não é aresta, j não pode seguir i no caminho":

$$(\neg x_{k,i} \vee \neg x_{k+1,j})$$

  para cada ij que não é aresta ($\mathrm{O}(n^3)$ claúsulas).

A fórmula $\phi(G)$ tem $\mathrm{O}(n^3)$ claúsulas e cada claúsula tem $\leq n$ literais. Logo, $\langle\phi(G)\rangle$ é $\mathrm{O}(n^4)$.

Não é difícil construir o algoritmo polinomial T.

Exemplo 3 (cont.)

$\phi(G)$ satisfatível $\Rightarrow$ G tem caminho hamiltoniano.

Prova: Seja $t : \{\text{variáveis}\} \to \{\text{VERDADE}, \text{FALSO}\}$ tal que $t(\phi(G)) = \text{VERDADE}$.

Para cada i existe um único j tal que $t(x_{i,j}) = \text{VERDADE}$. Logo, t é a codificadação de uma permutação

$$\pi(1), \pi(2), \ldots, \pi(n)$$

dos vértices de G, onde

$$\pi(i) = j \Leftrightarrow t(x_{i,j}) = \text{VERDADE}.$$

Para cada k, $(\pi(k), \pi(k+1))$ é uma aresta de G.

Logo, $(\pi(1), \pi(2), \ldots, \pi(n))$ é um caminho hamiltoniano.

Exemplo 3 (cont.)

G tem caminho hamiltoniano $\Rightarrow$ $\phi(G)$ satisfatível.

Suponha que $(\pi(1), \pi(2), \ldots, \pi(n))$ é um caminho hamiltoniano, onde π é uma pernutação dos vértices de G. Então

$t(x_{i,j}) =$ VERDADE se $\pi(i) = j$ e

$t(x_{i,j}) =$ VERDADE se $\pi(i) \neq j$,

é uma atribuição de valores que satisfaz todas as claúsulas de $\phi(G)$.

3-Satisfatibilidade

Problema: Dada um fórmula booleana ϕ nas variáveis $x_1, \ldots, x_n$ em que cada claúsula tem exatamente 3 literais, existe uma atribuição

$$t : \{x_1, \ldots, x_n\} \to \{\text{VERDADE}, \text{FALSO}\}$$

que torna ϕ verdadeira?

Exemplo:

$$\phi = (x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

Um literal é uma variável x ou sua negação $\neg x$.

Exemplo 4

Satisfatibilidade $\leq_P$ 3-Satisfatibilidade

Descreveremos um algoritmo polinomial T que recebe um fórmula booleana ϕ e devolve uma fórmula booleana ϕ' com exatamente 3 literais por claúsula tais que

$$\phi \text{ é satisfatível} \Leftrightarrow \phi' \text{ é satisfatível.}$$

A transformação consiste em substituir cada claúsula de ϕ por uma coleção de claúsulas com exatamente 3 literais cada e equivalente a ϕ.

Exemplo 4 (cont.)

Seja $(l_1 \vee l_2 \vee \cdots \vee l_k)$ uma claúsula de ϕ.

Caso 1. $k = 1$

Troque (l_1) por

$$(l_1 \vee y_1 \vee y_2)\,(l_1 \vee \neg y_1 \vee y_2)\,(l_1 \vee y_1 \vee \neg y_2)\,(l_1 \vee \neg y_1 \vee \neg y_2)$$

onde y_1 e y_2 são variáveis novas.

Caso 2. $k = 2$

Troque $(l_1 \vee l_2)$ por $(l_1 \vee l_2 \vee y)\,(l_1 \vee l_2 \vee \neg y)$. onde y é uma variáveis nova.

Caso 3. $k = 3$

Mantenha $(l_1 \vee l_2 \vee l_3)$.

Exemplo 4 (cont.)

Caso 4. $k > 3$

Troque $(l_1 \vee l_2 \vee \cdots \vee l_k)$ por

$(l_1 \vee l_2 \vee y_1)$

$(\neg y_1 \vee l_3 \vee y_2)\,(\neg y_2 \vee l_4 \vee y_3)\,(\neg y_3 \vee l_5 \vee y_4)\ \ldots$

$(\neg y_{k-3} \vee l_{k-1} \vee l_k)$

onde $y_1, y_2, \ldots, y_{k-3}$ são variáveis novas

Verifique que ϕ é satisfátivel $\Leftrightarrow$ nova fórmula é satisfatível.

O tamanho da nova claúsula é $\mathrm{O}(m)$, onde m é o número de literais que ocorrem em ϕ (contando-se as repetições).

Problemas completos em NP

Um problema Π em NP é NP-completo se cada problema em NP pode ser reduzido a Π.

Teorema de S. Cook e L.A. Levin: Satisfatibilidade é NP-completo.

Se $\Pi \leq_P \Pi'$ e Π é NP-completo, então Π' é NP-completo.

Existe um algoritmo polinomial para um problema NP-completo se e somente se P $=$ NP.

Demonstração de NP-completude

Para demonstrar que um problema Π' é NP-completo podemos utilizar o Teorema de Cook e Levin.

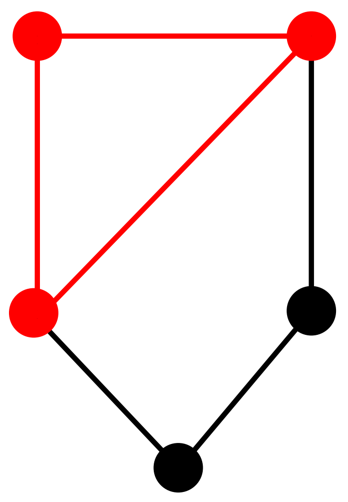
Para isto devemos:

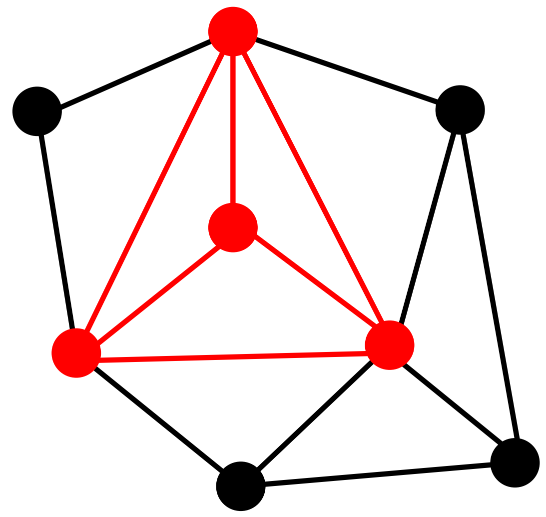
- Demonstrar que Π' está en NP.
- Escolher um problema Π sabidamente NP-completo.
- Demonstrar que $\Pi \leq_P \Pi'$.

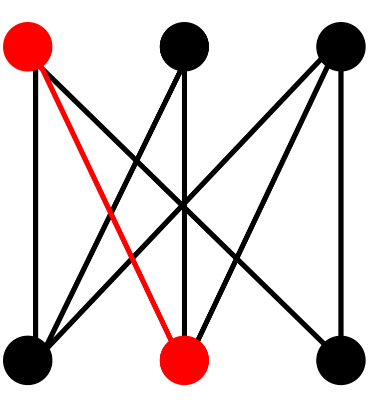
Clique

Problema: Dado um grafo G e um inteiro k, G possui um clique com $\geq k$ vértices?

Exemplos:







clique com k vértices $=$ subgrafo completo com k vértices

Clique é NP-completo

Clique está em NP e 3-Satisfatibilidade $\leq_P$ Clique.

Descreveremos um algoritmo polinomial T que recebe um fórmula booleana ϕ com k claúsulas e exatamente 3 literais por claúsula e devolve um grafo G tais que

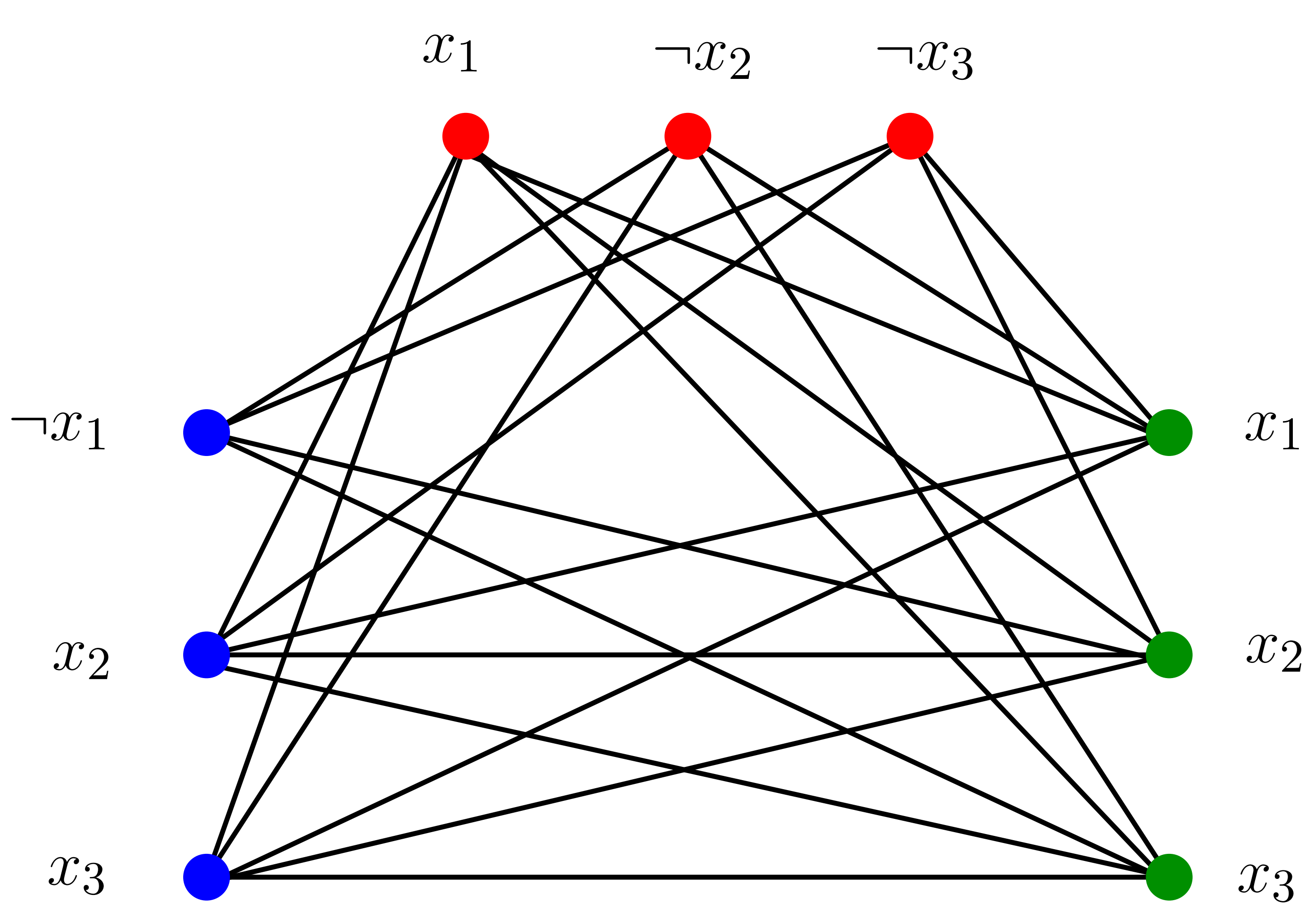
$$\phi \text{ é satisfatível} \Leftrightarrow G \text{ possui um clique} \geq k.$$

Para cada claúsula o grafo G terá três vértices, um correspondente a cada literal da cláusula. Logo, G terá $3k$ vértices. Teremos uma aresta ligando vértices u e v se

- u e v são vértices que correspondem a literais em diferentes claúsula; e
- se u corresponde a um literal x então v não corresponde ao literal $\neg x$.

Clique é NP-completo (cont.)

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



Problemas NP-difíceis

Um problema Π, não necessariamente em NP, é NP-díficil se a existência de um algoritmo polinomial para Π implica em P $=$ NP.

Todo problema NP-completo é NP-difícil.

Exemplos:

- Encontrar um ciclo hamiltoniano é NP-difícil, mas não é NP-completo, pois não é um problema de decisão e portanto não está em NP.
- Satisfabilidade é NP-completo e NP-difícil.

Mais problemas NP-difíceis

Os seguintes problema são NP-difíceis:

- mochila booleana
- caminho máximo
- caminho hamiltoniano
- escalonamento de tarefas
- subset-sum
- clique máximo
- cobertura por vértices
- sistemas 0-1

e mais um montão deles . . .

Exercícios

Exercício 25.A
Suponha que os algoritmos ALG e ALG′ transformam cadeias de caracteres em outras cadeias de caracteres. O algoritmo ALG consome $O(n^2)$ unidades de tempo e o algoritmo ALG′ consome $O(n^4)$ unidades de tempo, onde n é o número de caracteres da cadeia de entrada. Considere agora o algoritmo ALGALG′ que consiste na composição de ALG e ALG′, com ALG′ recebendo como entrada a saída de ALG. Qual o consumo de tempo de ALGALG′?

Exercício 25.B [CLRS 34.1-4]
O algoritmo MOCHILA-BOOLEANA é polinomial? Justifique a sua resposta.

Exercício 25.C [CLRS 34.1-5]
Seja ALG um algoritmo que faz um número constante de chamadas a um algoritmo ALG′. Suponha que se o consumo de tempo de ALG′ é constante então o consumo de tempo de ALG é polinomial.

1. Mostre que se o consumo de tempo de ALG′ é polinomial então o consumo de tempo de ALG é polinomial.
2. Mostre que se o consumo de tempo de ALG′ é polinomial e ALG faz um número polinomial de chamadas a ALG′, então é possível que o consumo de tempo de ALG seja exponencial.

Mais exercícios

Exercício 25.D [CLRS 34.2-1]
Mostre que o problema de decidir se dois grafos dados são isomorfos está em NP.

Exercício 25.E [CLRS 34.2-2]
Mostre que um grafo bipartido com um número ímpar de vértices não é hamiltoniano ($=$ possui um ciclo hamiltoniano).

Exercício 25.F [CLRS 34.2-3]
Mostre que se o problema do Ciclo hamiltoniano está em ¶, então o problema de listar os vértices de um ciclo hamiltoniano, na ordem em que eles ocorrem no ciclo, pode ser resolvido em tempo polinomial.

Exercício 25.G [CLRS 34.2-5]
Mostre que qualquer problema em NP pode ser resolvido por um algoritmo de consumo de tempo $2^{\mathrm{O} n^c}$, onde n é o tamanho da entrada e c é uma constante.

Exercício 25.H [CLRS 34.2-6]
Mostre que o problema do Caminho hamiltoniano está em NP.

Exercício 25.I [CLRS 34.2-7]
Mostre que o problema do caminho hamiltoniano pode ser resolvido em tempo polinomial em grafos orientado acíclicos.

Mais exercícios

Exercício 25.J [CLRS 34.2-8]
Uma fórmula booleana ϕ é uma tautologia se $t(\phi) = \text{VERDADE}$ para toda atribuição de $t : \{\text{variáveis}\} \rightarrow \{\text{VERDADE}, \text{FALSO}\}$. Mostre que o problema de decidir se uma dada fórmula booleana é uma tautologia está em co-NP.

Exercício 25.K [CLRS 34.2-9]
Prove que P $\subseteq$ co-NP.

Exercício 25.L [CLRS 34.2-10]
Prove que se NP $\neq$ co-NP, então P $\neq$ NP.

Exercício 25.M [CLRS 34.2-11]
Se G é um grafo conexo com pelo menos 3 vértices, então G^3 é o grafo que se obtém a partir de G ligando-se por uma aresta todos os pares de vértices que estão conectados em G por um caminho com no máximo três arestas. Mostre que G^3 é hamiltoniano.

Exercício 25.N [CLRS 34.3-2]
Mostre que se $\Pi_1 \leq_P \Pi_2$ e $\Pi_2 \leq_P \Pi_3$, então $\Pi_1 \leq_P \Pi_3$.

Mais exercícios

Exercício 25.O [CLRS 34.3-7]
Suponha que Π e Π' são problemas de decisão sobre o mesmo conjunto de instâncias e que $\Pi(I) = \text{SIM}$ se e somente se $\Pi'(I) = \text{NÃO}$. Mostre que Π é NP-completo se e somente se Π' é co-NP-completo.
(Um problema Π' é co-NP-completo se Π' está em co-NP e $\Pi \leq_P \Pi'$ para todo problema Π em co-NP.)

Exercício 25.P [CLRS 34.4-4]
Mostre que o problema de decidir se uma fórmula boolena é uma tautologia é co-NP-completo. (Dica: veja o exercício 25.O.)

Exercício 25.Q [CLRS 34.4-6]
Suponha que ALG$'$ é um algoritmo polinomial para Satisfatibilidade. Descreva um algoritmo polinomial ALG que recebe um fórmula booleana ϕ e devolve uma atribuição $t : \{\text{variáveis}\} \rightarrow \{\text{VERDADE}, \text{FALSO}\}$ tal que $t(\phi) = \text{VERDADE}$.

Exercício 25.Q [CLRS 34.5-3]
Prove que o problema Sistemas lineares 0-1 é NP-completo.

Exercício 25.R [CLRS 34.5-6]
Mostre que o problema C aminho hamiltoniano é NP-completo.

Mais um exercício

Exercício 25.S [CLRS 34.5-7]
Mostre que o problema de encontrar um ciclo de comprimento máximo é NP-completo.

Melhores momentos

AULA 25

Verificador polinomial para SIM

Um verificador polinomial para a resposta SIM a um problema Π é uma algoritmo polinomial ALG que recebe

uma instância I de Π e um objeto C, tal que $\langle C \rangle$ é $\mathrm{O}(\langle I \rangle^c)$ para alguma constante c e

$$\text{ALG}(I, C) = \text{SIM} \Leftrightarrow \Pi(I) = \text{SIM}$$

A constante c depende apenas do problema!

P, NP e co-NP

Classe P	formada por problemas de decisão que podem ser resolvidos em tempo polinomial
Classe NP	formada por problemas de decisão que possuem um verificador polinomial para a resposta SIM
Classe co-NP	formada por problemas de decisão que possuem um verificador polinomial para a resposta NÃO

Redução polinomial

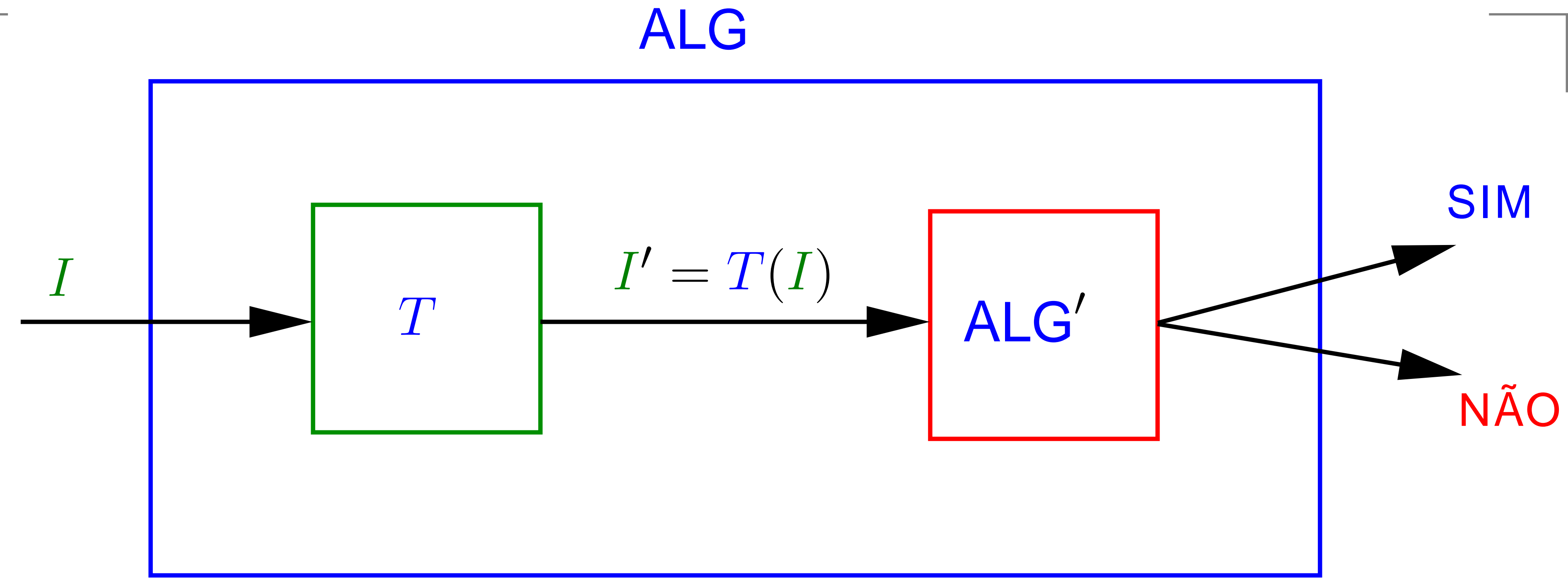
Permite comparar o “grau de complexidade” de problemas diferentes.

Redução (polinomial) de um problema Π a um problema Π' é um algoritmo ALG que resolve Π usando uma subrotina hipotética ALG$'$ que resolve Π', de tal forma que, se ALG$'$ é um algoritmo polinomial, então ALG é um algoritmo polinomial.

$\Pi \leq_P \Pi' =$ existe uma redução de Π a Π'.

Se $\Pi \leq_P \Pi'$ e Π' está em P, então Π está em P.

Esquema comum de redução



Faz apenas uma chamada ao algoritmo ALG$'$.

T transforma uma instância I de Π em uma instância $I' = T(I)$ de Π' tal que

$$\Pi(I) = \text{SIM se e somente se } \Pi'(I') = \text{SIM}$$

T é uma espécie de “filtro” ou “compilador”.

Reduções

Satisfatibilidade $\leq_P$ Sistemas lineares 0-1

Ciclo hamiltoniano $\leq_P$ Caminho hamiltoniano entre u e v

Caminho hamiltoniano entre u e v $\leq_P$ Caminho hamiltoniano

Caminho hamiltoniano $\leq_P$ Satisfatibilidade

Satisfatibilidade $\leq_P$ 3-Satisfatibilidade

Hoje: 3-Satisfatibilidade $\leq_P$ Clique

Problemas completos em NP

Um problema Π em NP é NP-completo se cada problema em NP pode ser reduzido a Π.

Teorema de S. Cook e L.A. Levin: Satisfatibilidade é NP-completo.

Se $\Pi \leq_P \Pi'$ e Π é NP-completo, então Π' é NP-completo.

Existe um algoritmo polinomial para um problema NP-completo se e somente se $\mathrm{P} = \mathrm{NP}$.

Demonstração de NP-completude

Para demonstrar que um problema Π' é NP-completo podemos utilizar o Teorema de Cook e Levin.

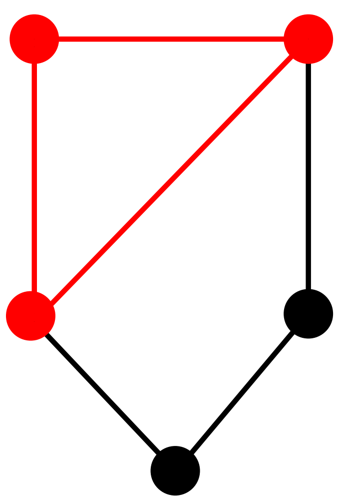
Para isto devemos:

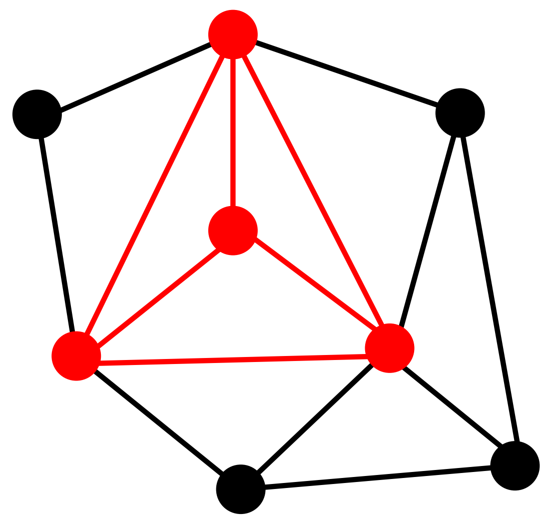
- Demonstrar que Π' está en NP.
- Escolher um problema Π sabidamente NP-completo.
- Demonstrar que $\Pi \leq_P \Pi'$.

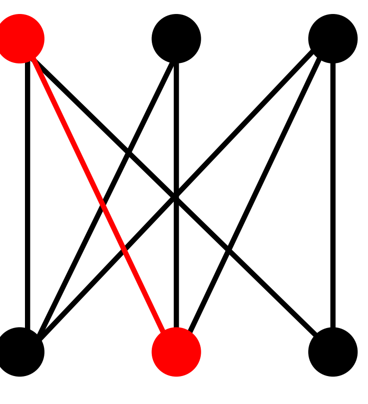
Clique

Problema: Dado um grafo G e um inteiro k, G possui um clique com $\geq k$ vértices?

Exemplos:







clique com k vértices $=$ subgrafo completo com k vértices

Clique é NP-completo

Clique está em NP e 3-Satisfatibilidade $\leq_P$ Clique.

Descreveremos um algoritmo polinomial T que recebe um fórmula booleana ϕ com k claúsulas e exatamente 3 literais por claúsula e devolve um grafo G tais que

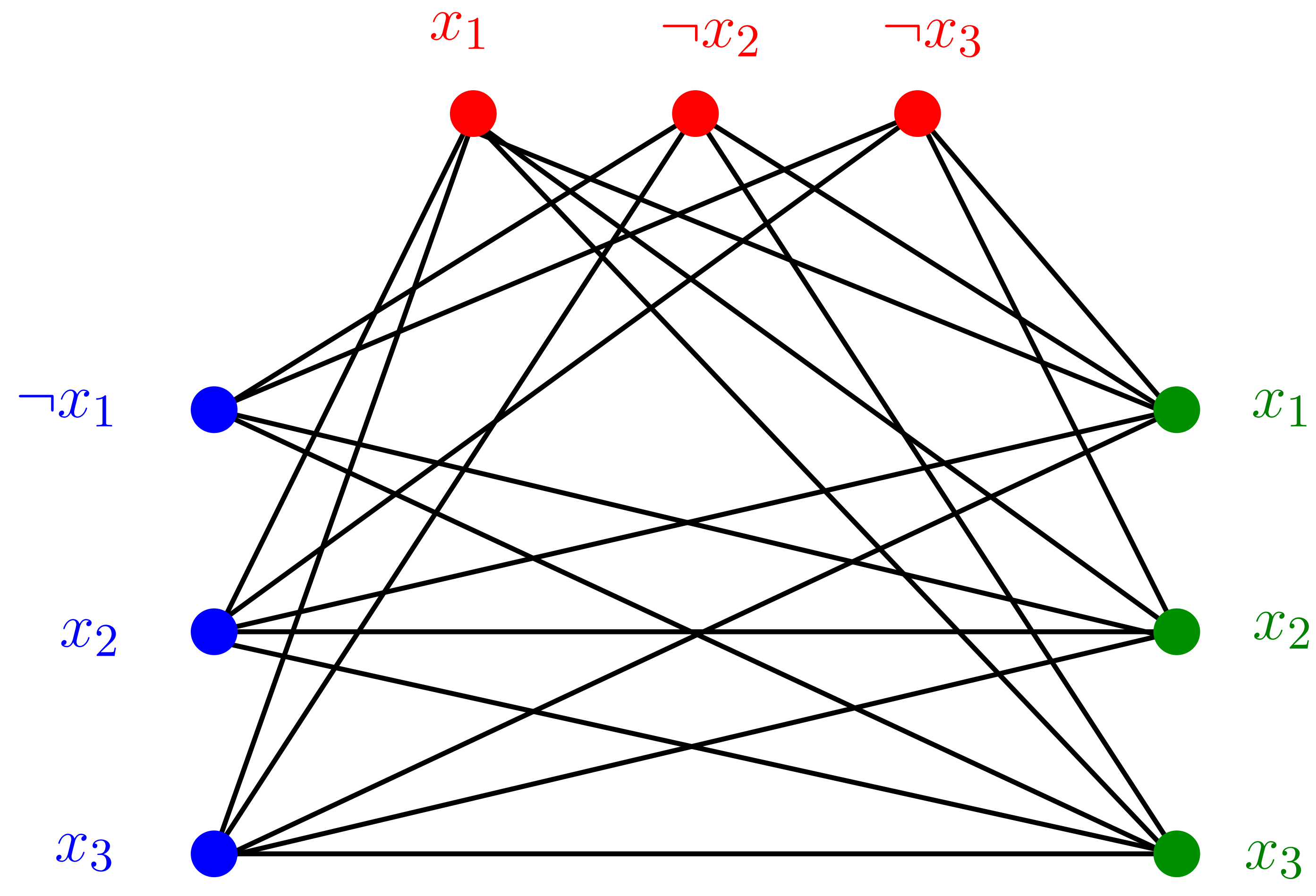
$$\phi \text{ é satisfatível} \Leftrightarrow G \text{ possui um clique} \geq k.$$

Para cada claúsula o grafo G terá três vértices, um correspondente a cada literal da cláusula. Logo, G terá $3k$ vértices. Teremos uma aresta ligando vértices u e v se

- u e v são vértices que correspondem a literais em diferentes claúsula; e
- se u corresponde a um literal x então v não corresponde ao literal $\neg x$.

Clique é NP-completo (cont.)

$$\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



Verifique: ϕ é satisfatível $\Leftrightarrow$ G possui um clique $\geq k$.

Problemas NP-difíceis

Um problema Π, não necessariamente em NP, é NP-díficil se a existência de um algoritmo polinomial para Π implica em P $=$ NP.

Todo problema NP-completo é NP-difícil.

Exemplos:

- Encontrar um ciclo hamiltoniano é NP-difícil, mas não é NP-completo, pois não é um problema de decisão e portanto não está em NP.
- Satisfabilidade é NP-completo e NP-difícil.

AULA 26

Algoritmos de aproximação

CLRS 35

Transparências de Cristina Gomes Fernandes

MAC5727 Algoritmos de Aproximação

Escalonamento de máquinas idênticas

Dados: m máquinas

t tarefas

duração $d[i]$ da tarefa i $(i = 1, \ldots, t)$

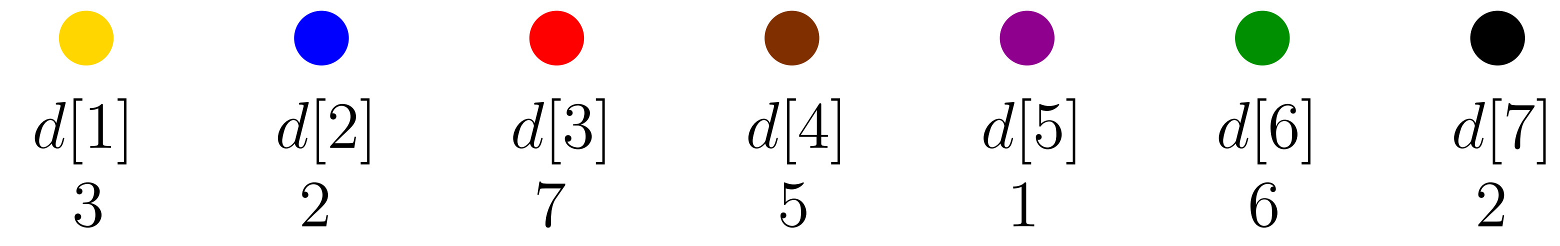
um escalonamento é uma partição $\{M[1], \ldots, M[m]\}$ de $\{1, \ldots, t\}$

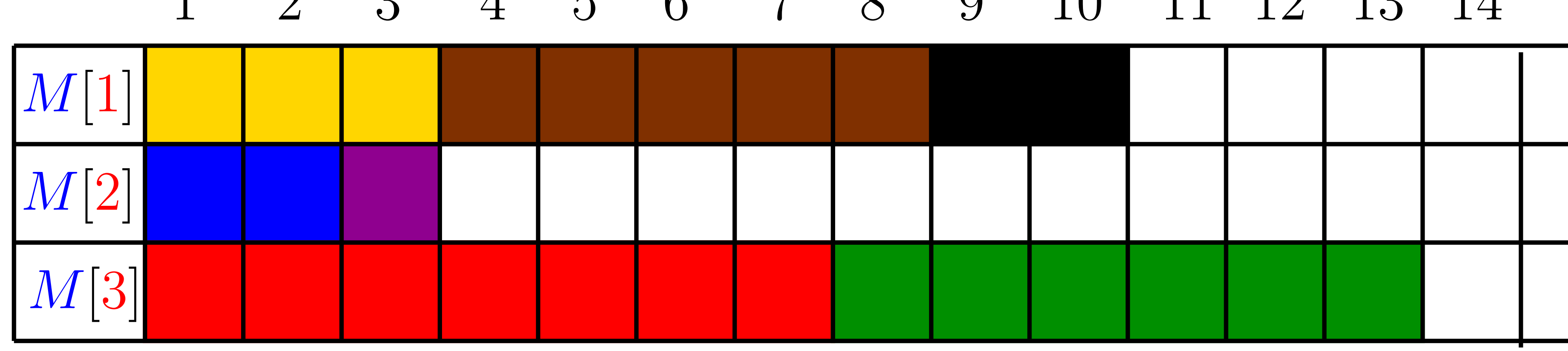
tarefas

máquinas

Exemplo 1

$m = 3 \quad t = 7$

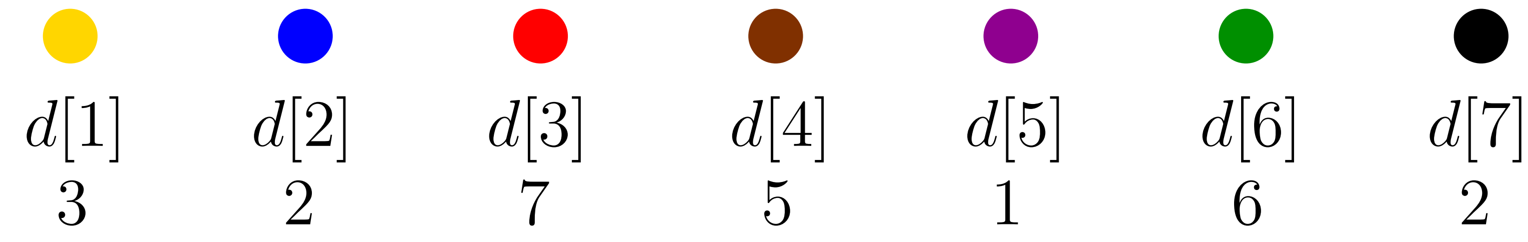


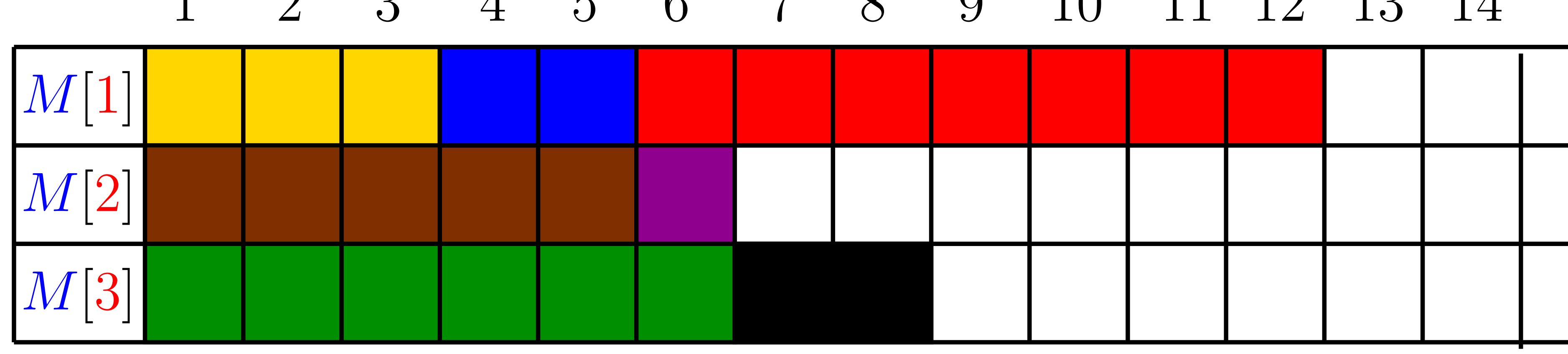


$\{\{1, 4, 7\}, \{2, 5\}, \{3, 6\}\} \Rightarrow$ Tempo de conclusão $= 13$

Exemplo 2

$m = 3 \quad t = 7$

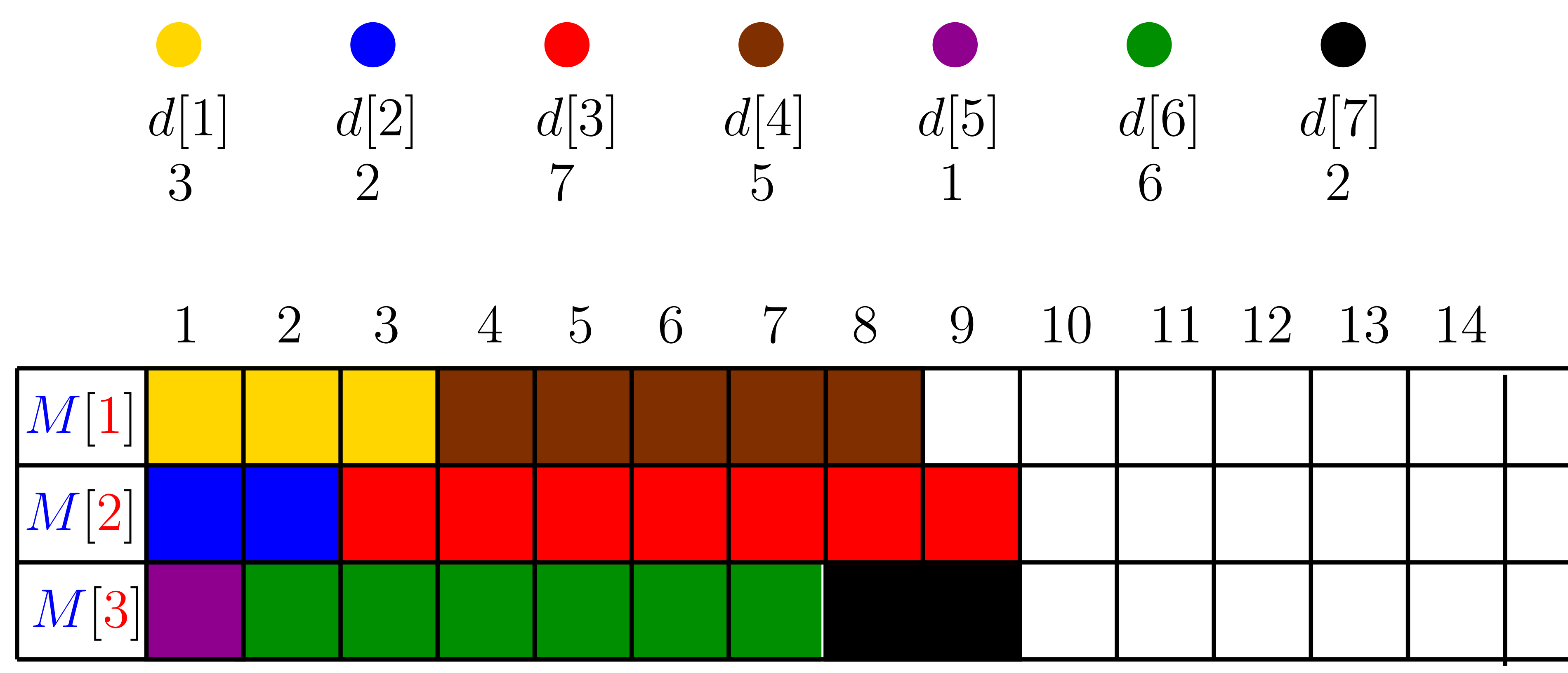




$\{\{1, 2, 3\}, \{4, 5\}, \{6, 7\}\} \Rightarrow$ Tempo de conclusão $= 12$

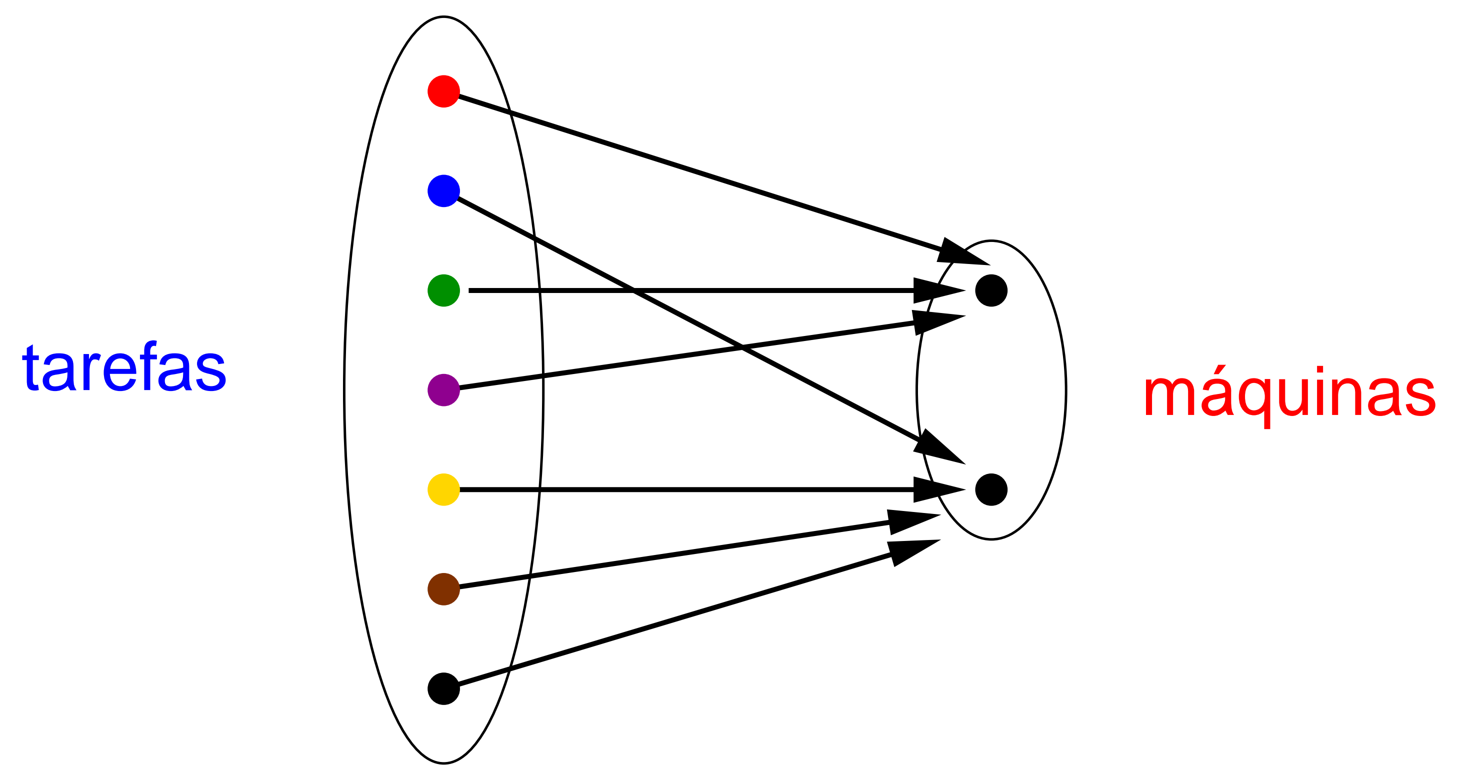
Problema

Encontrar um escalonamento com tempo de conclusão mínimo.



$\{\{1,4\},\{2,3\},\{5,6,7\}\} \Rightarrow$ Tempo de conclusão $= 9$

NP-difícil mesmo para $m = 2$



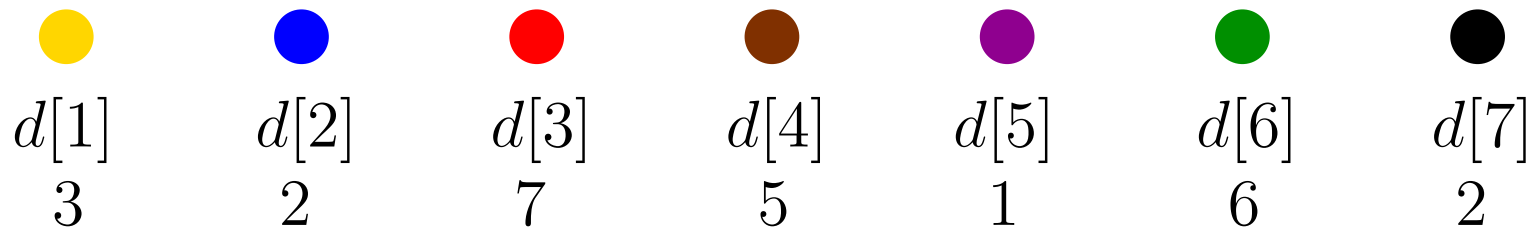
Algoritmo: testa todo $M[1] \subseteq \{1, \ldots, t\}$ e escolhe melhor

2^t subconjuntos $\Rightarrow$ exponencial

NP-difícil $\Rightarrow$ é improvável que exista algoritmo polinomial que resolva o problema (se exitir, P $=$ NP)

Algoritmo de Graham

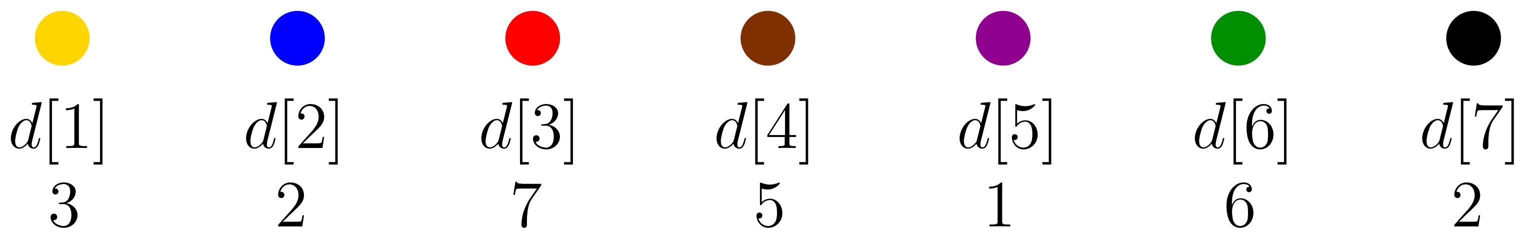
Atribui, uma a uma, cada tarefa à máquina menos ocupada.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$M[1]$														
$M[2]$														
$M[3]$														

Algoritmo de Graham

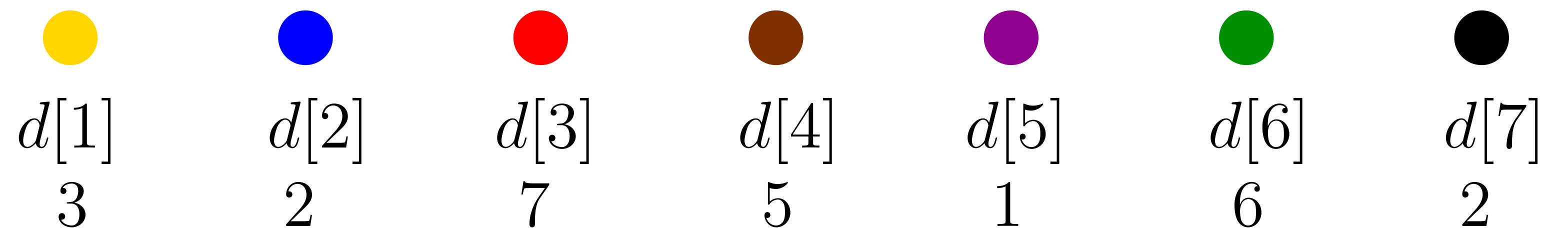
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

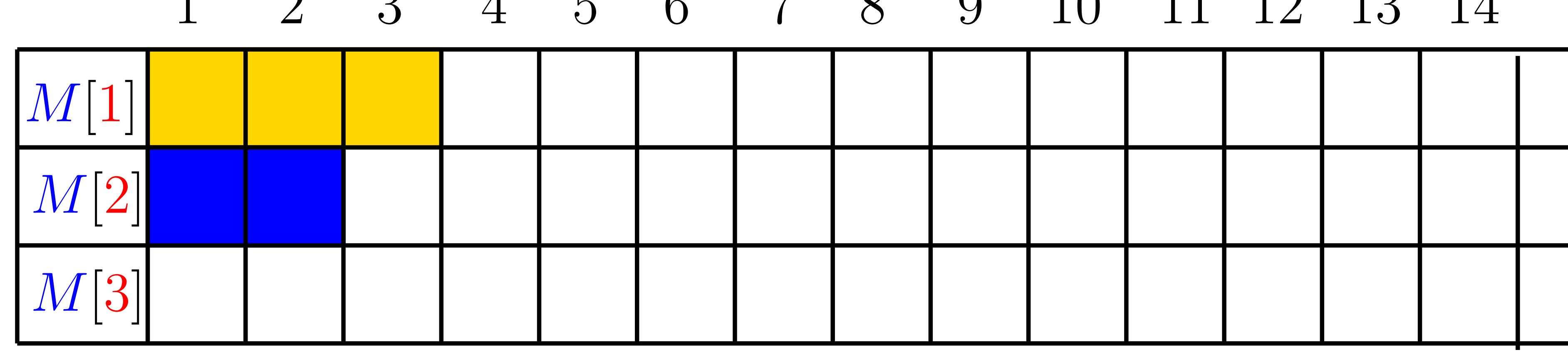


	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$M[1]$														
$M[2]$														
$M[3]$														

Algoritmo de Graham

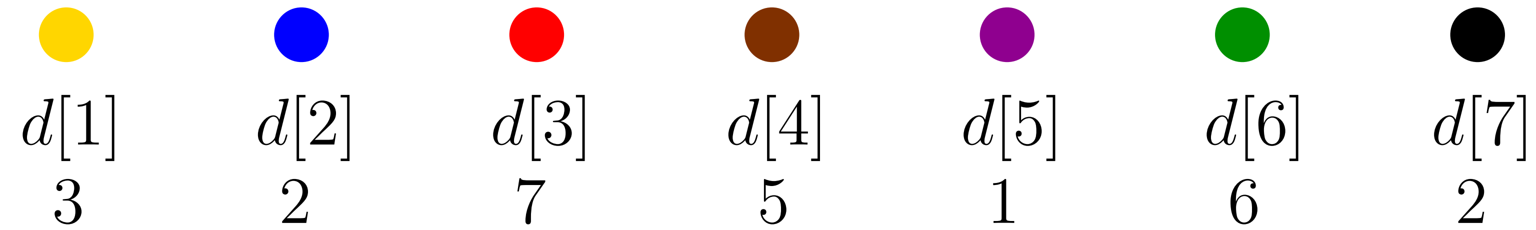
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

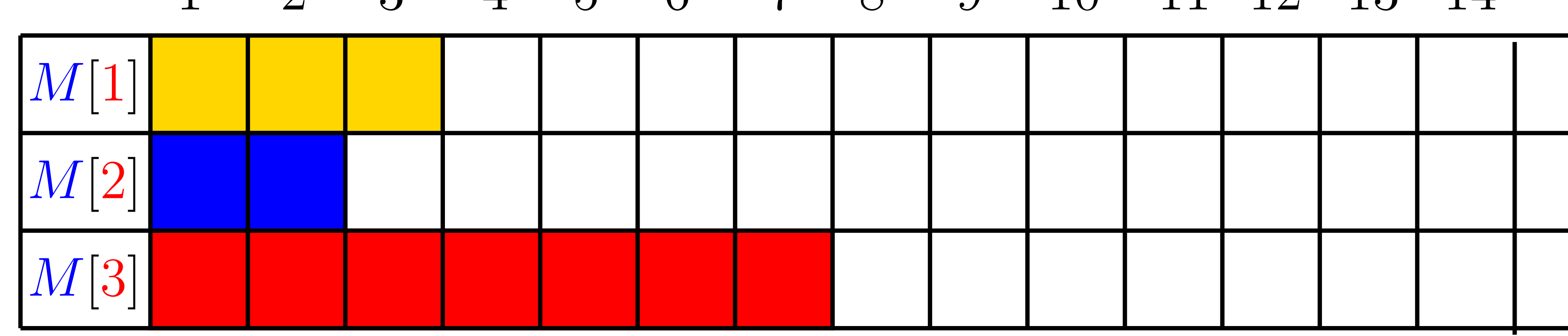




Algoritmo de Graham

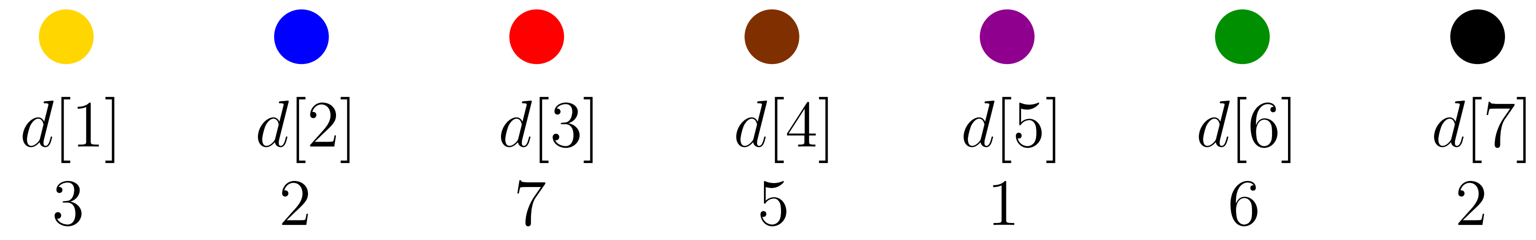
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

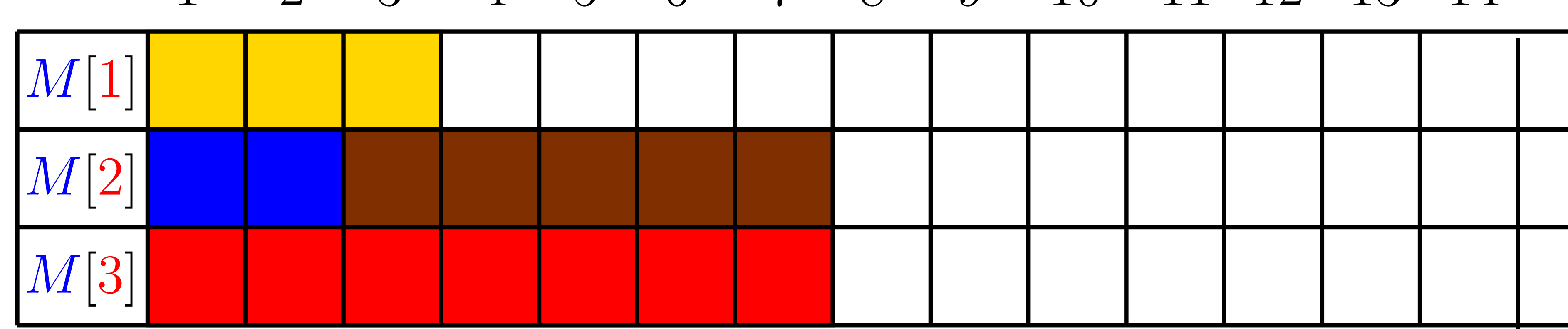




Algoritmo de Graham

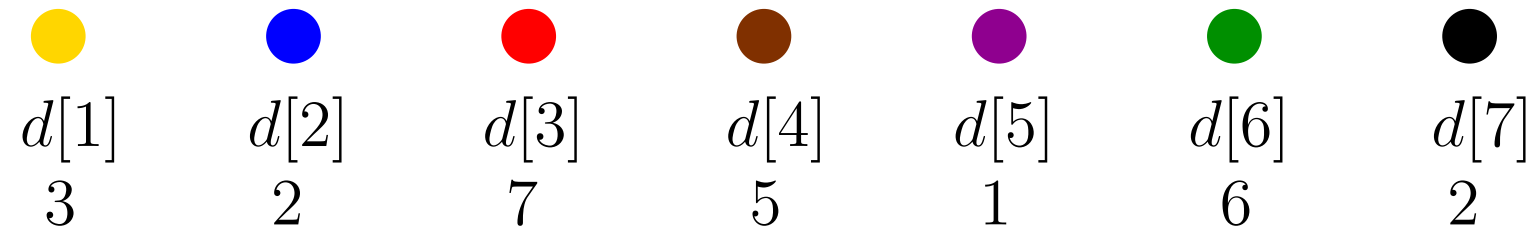
Atribui, uma a uma, cada tarefa à máquina menos ocupada.

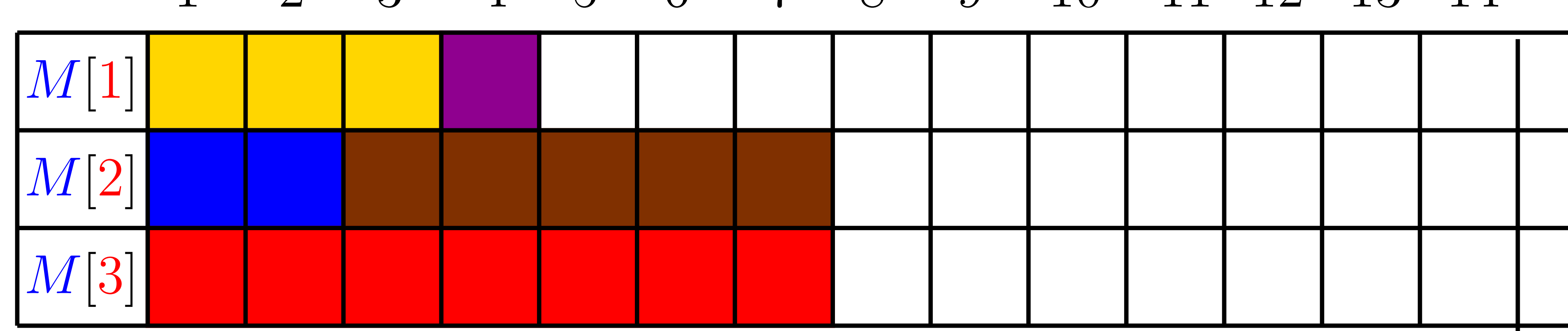




Algoritmo de Graham

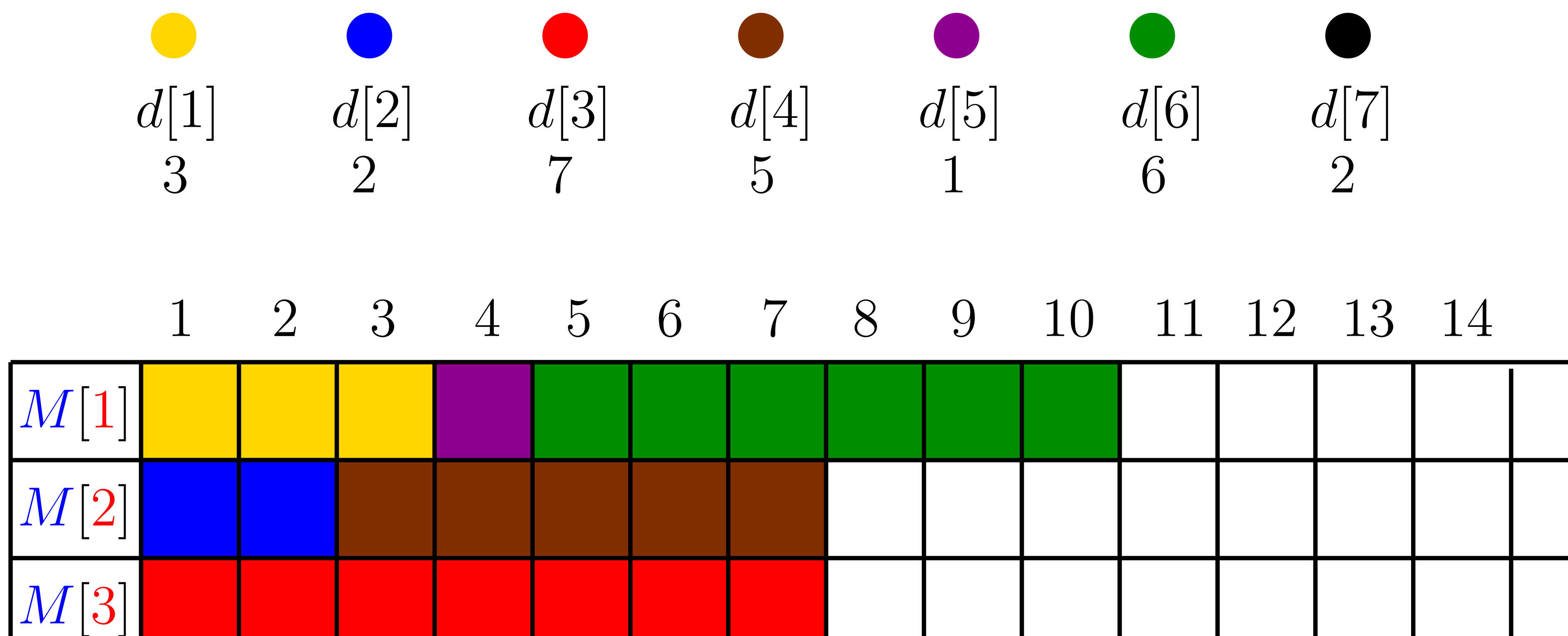
Atribui, uma a uma, cada tarefa à máquina menos ocupada.





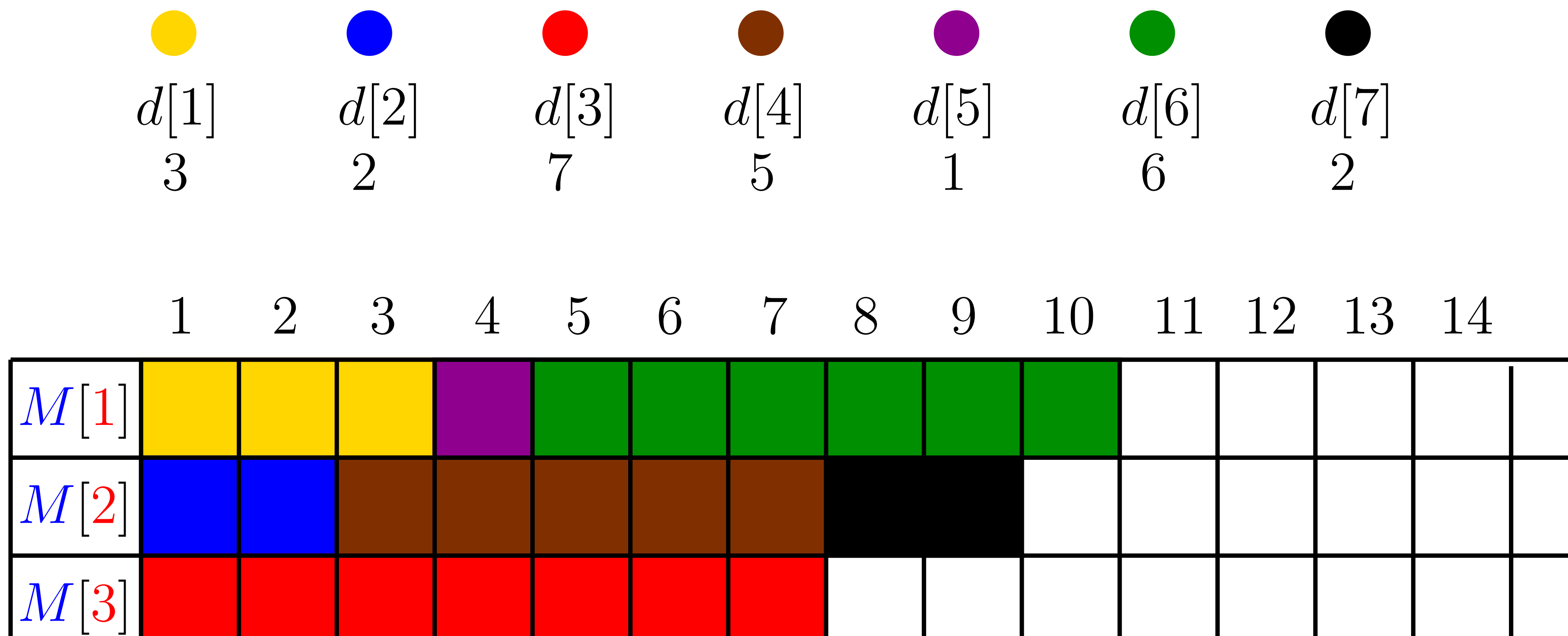
Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.



Algoritmo de Graham

Atribui, uma a uma, cada tarefa à máquina menos ocupada.



Escalonamento-Graham

Recebe números inteiros positivos m e t e um vetor $d[1\,.\,.\,t]$ e **devolve** um escalomento de $\{1,\ldots,t\}$ em m máquinas.

ESCALONAMENTO-GRAHAM (m, t, d)

```
1   para j ← 1 até m faça
2       M[j] ← ∅
3       T[j] ← 0
4   para i ← 1 até t faça
5       seja k tal que T[k] é mínimo
6       M[k] ← M[k] ∪ {i}
7       T[k] ← T[k] + d[i]
8   devolva {M[1], ..., M[m]}
```

Consumo de tempo

linha	consumo de todas as execuções da linha
1	$\Theta(m)$
2	$\Theta(m)$
3	$\Theta(m)$
4	$\Theta(t)$
5	$\Theta(mt)$
6	$\Theta(t)$
7	$\Theta(t)$
8	$\Theta(t)$
total	$3\Theta(m) + \Theta(mt) + 3\Theta(t) = \Theta(mt)$

Escalonamento-Graham

Se utilizarmos uma fila com prioridades (min-heap) para representar as m máquinas onde a prioridade da máquina i é $T[i]$ teremos:

ESCALONAMENTO-GRAHAM (m, t, d)

1 **para** $j \leftarrow 1$ **até** m **faça**
2 $\quad M[j] \leftarrow \emptyset$
3 $\quad T[j] \leftarrow 0$
4 BUILD-MIN-HEAP (T, m)
5 **para** $i \leftarrow 1$ **até** t **faça**
6 $\quad k \leftarrow$ HEAP-MIN (T, m)
7 $\quad M[k] \leftarrow M[k] \cup \{i\}$
8 $\quad$ HEAP-INCREASE-KEY $(T, k, T[k] + d[i])$
9 **devolva** $\{M[1], \ldots, M[m]\}$

Consumo de tempo

linha	consumo de todas as execuções da linha
1-4	$\Theta(m)$
5	$\Theta(t)$
6	$\Theta(t)$
7	$\Theta(t)$
8	$\mathrm{O}(t \lg m)$
9	$\Theta(t)$
total	$\Theta(m) + 4\Theta(t) + \mathrm{O}(t \lg m) = \mathrm{O}(m + t \lg m)$

O consumo de tempo do algoritmo ESCALONAMENTO-GRAHAM é $\mathrm{O}(m + t \lg m)$.

Delimitações para OPT

OPT = menor tempo de conclusão de um escalonamento

- Duração da tarefa mais longa:

$$\mathrm{OPT} \geq \max\{d[1], d[2], \ldots, d[t]\}$$

- Distribuição balanceada:

$$\mathrm{OPT} \geq \frac{d[1] + d[2] + \cdots + d[t]}{m}$$

Fator de aproximação

tarefa t é executada na máquina j a partir do instante T

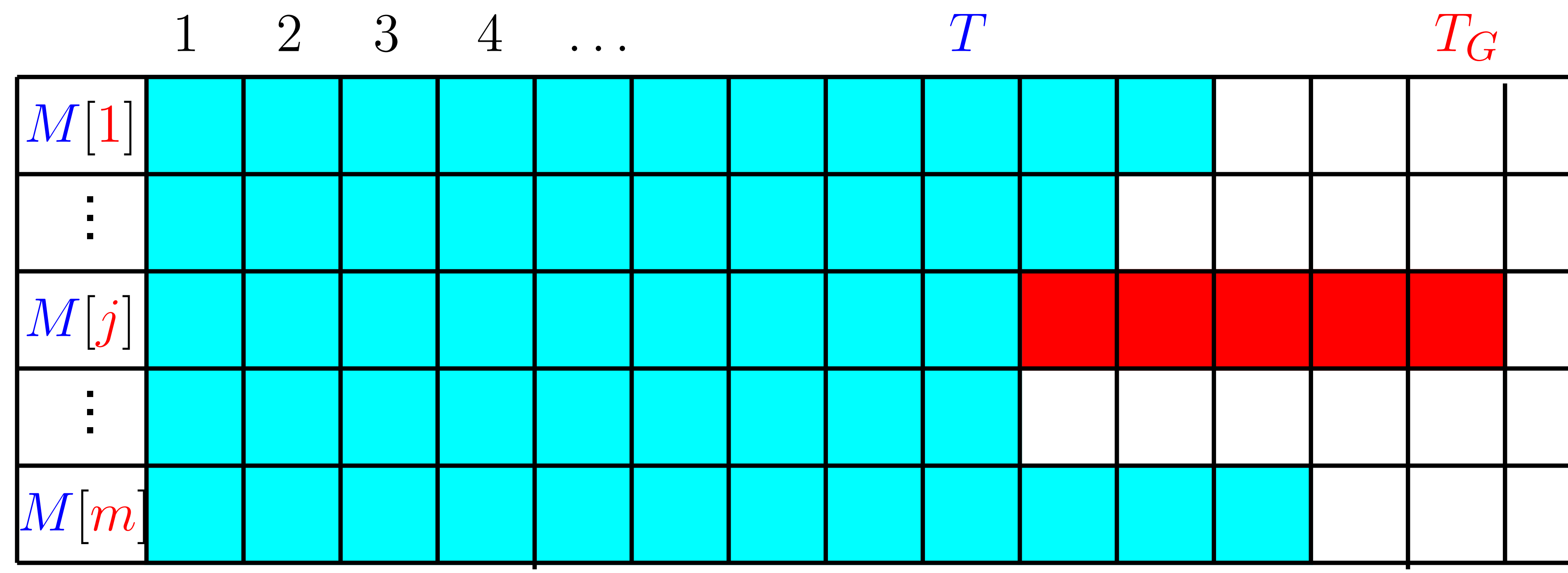
$T_G =$ conclusão do escalonamento obtido pelo algoritmo

	1	2	3	4	$\ldots$				T					T_G	
$M[1]$															
$\vdots$															
$M[j]$															
$\vdots$															
$M[m]$															

Fator de aproximação (cont.)

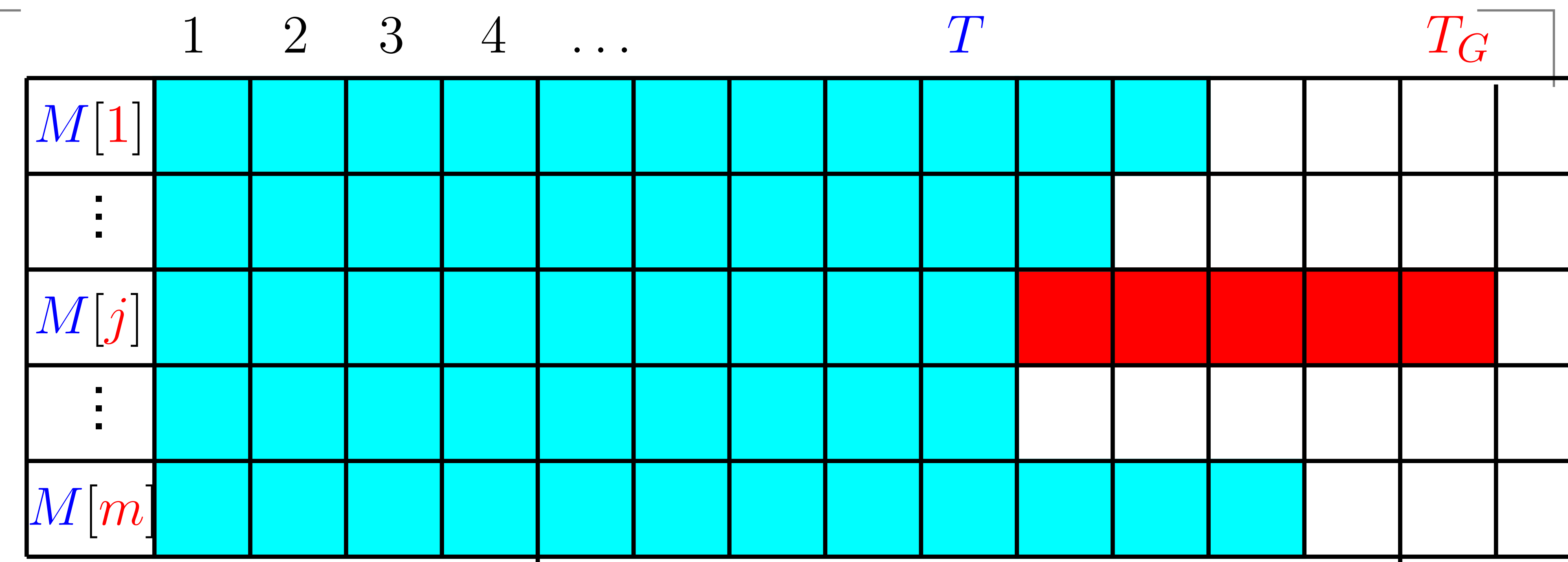
tarefa t é executada na máquina j a partir do instante T

$T_G =$ conclusão do escalonamento obtido pelo algoritmo



$$T \cdot m \leq d[1] + \cdots + d[t] \quad \Rightarrow \quad T \leq \frac{d[1] + \cdots + d[t]}{m}$$

Fator de aproximação (cont.)



$$
\begin{aligned}
T_G &= T + d[t] \\
&\le \frac{d[1] + \cdots + d[t]}{m} + d[t] \\
&\le \frac{d[1] + \cdots + d[t]}{m} + \max\{d[1], \ldots, d[t]\} \\
&\le \mathrm{OPT} + \mathrm{OPT} = 2\,\mathrm{OPT}
\end{aligned}
$$

Algoritmos de aproximação

Algoritmo ALG é de aproximação se existe $\alpha > 0$ tal que

$$\text{valor de ALG}(I) \leq \alpha \cdot \mathrm{OPT}(I)$$

para toda instância I do problema.

α é o fator de aproximação

Objetivo: α tão perto de 1 quanto possível

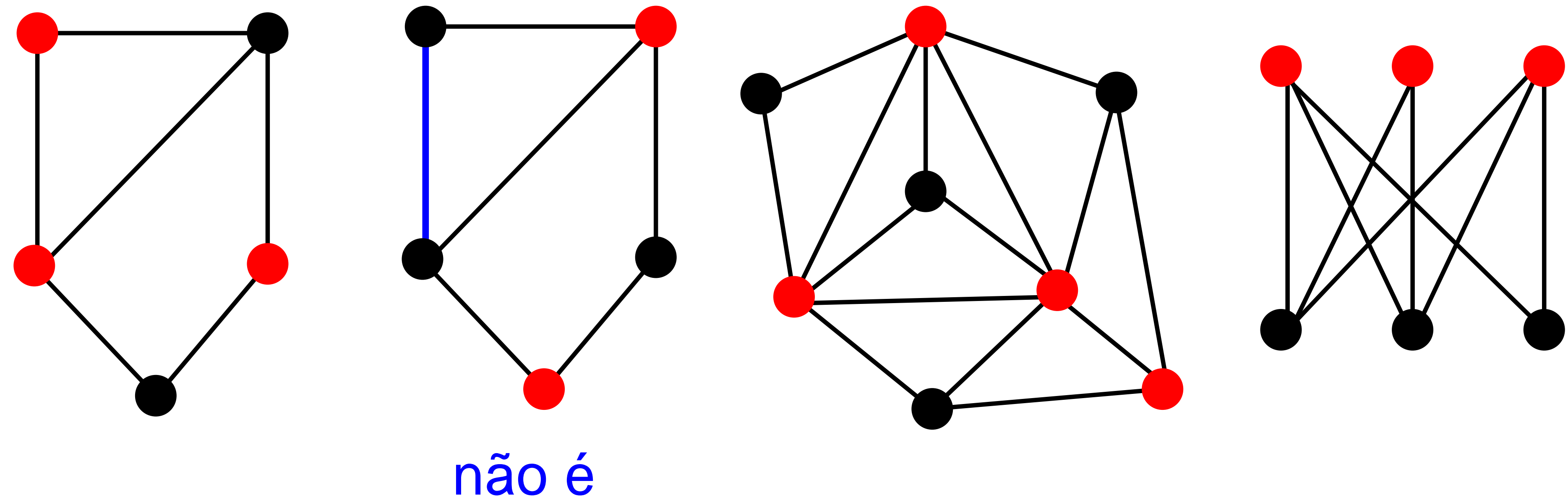
Conclusão

O algoritmo ESCALONAMENTO-GRAHAM é uma 2-aproximação polinomial.

Cobertura por vértices

Um conjunto de vértices C é uma cobertura por vértices de G se cada aresta tem pelo menos uma ponta em C

Exemplos:



Cobertura por vértices

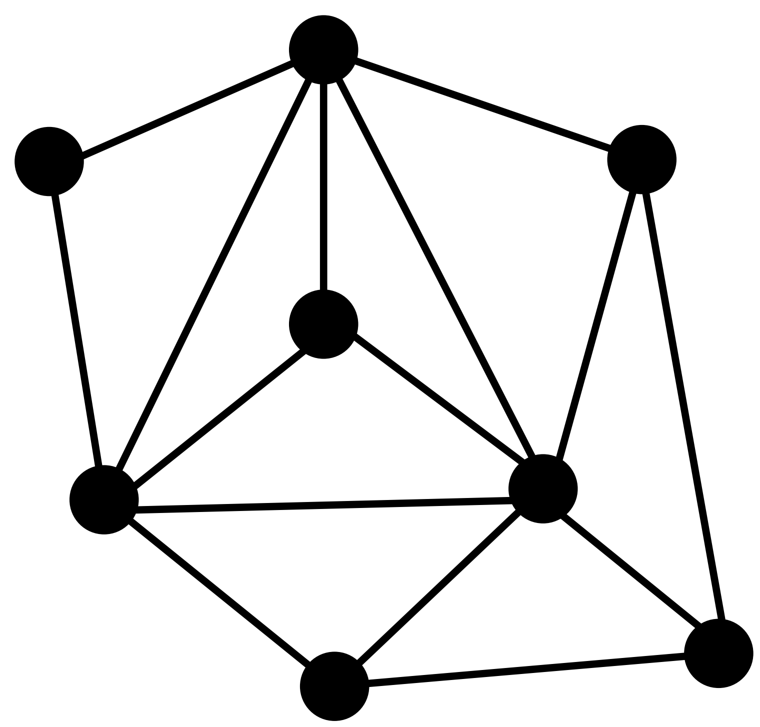
Problema: Dado um grafo G, encontrar um cobertura mínima.

Problema é NP-difícil.

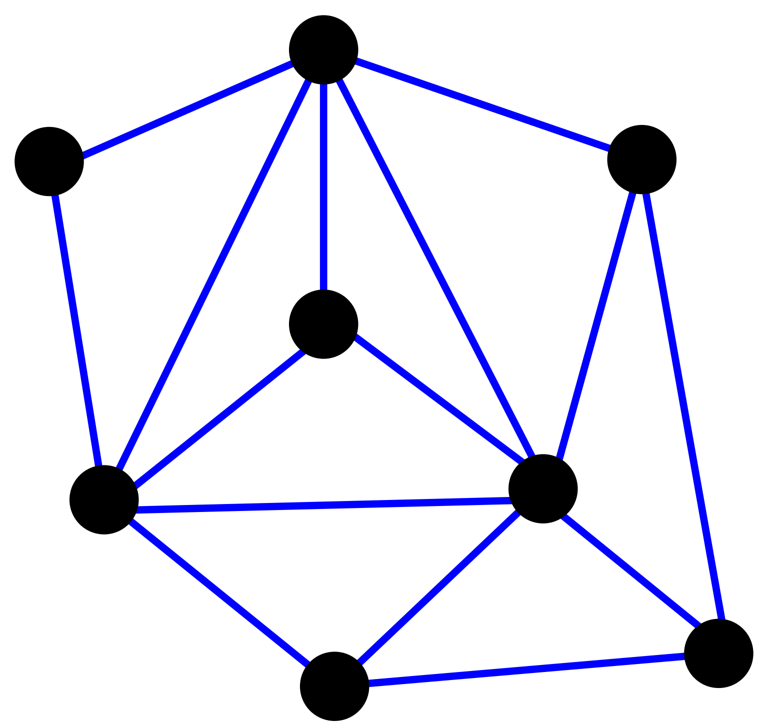
APPROX-VERTEX-COVER (G)

1 $C \leftarrow \emptyset$
2 $E \leftarrow \{$arestas de $G\}$
3 **enquanto** $E \neq \emptyset$ **faça**
4 seja (u, v) uma aresta qualquer em E
5 $C \leftarrow C \cup \{u, v\}$
6 $E \leftarrow E - \{$arestas incidentes a u ou $v\}$
7 **devolva** C

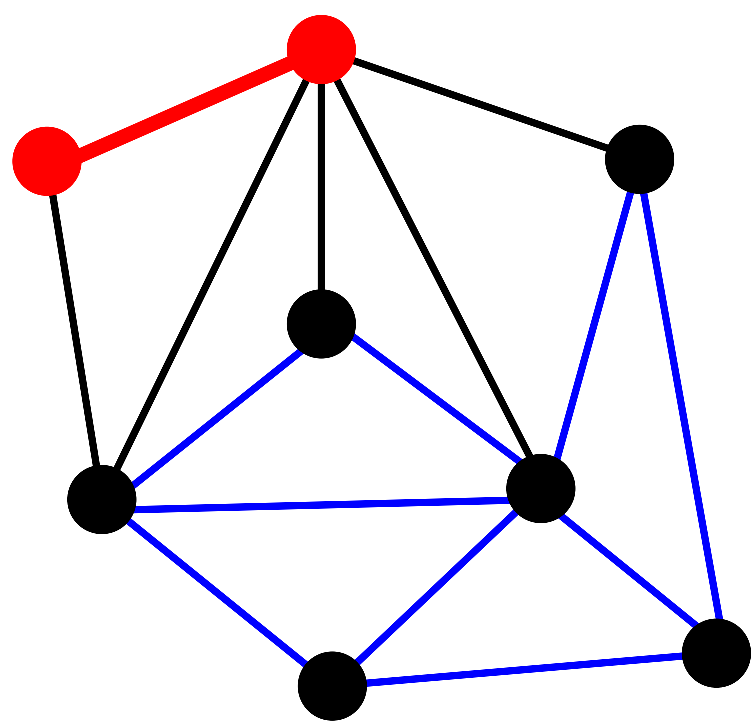
Exemplo



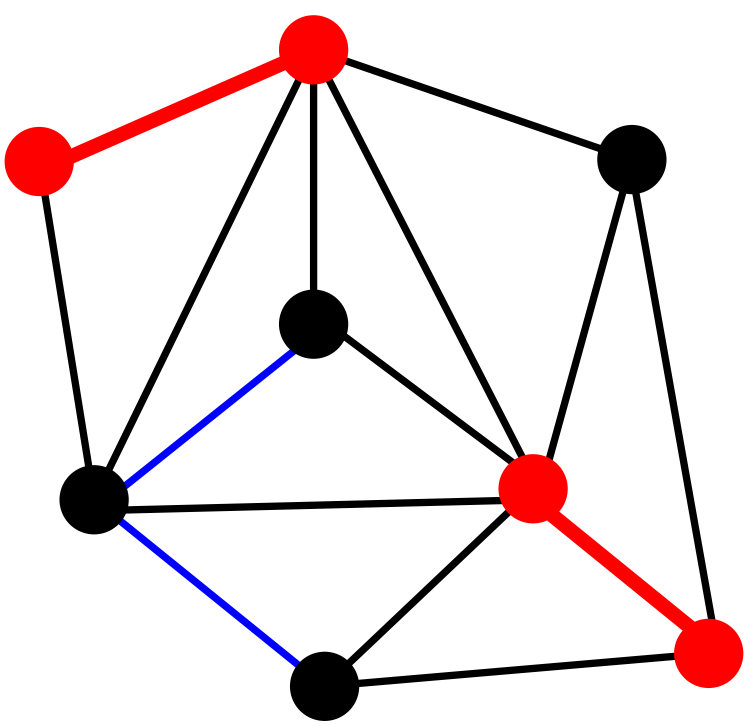
Exemplo



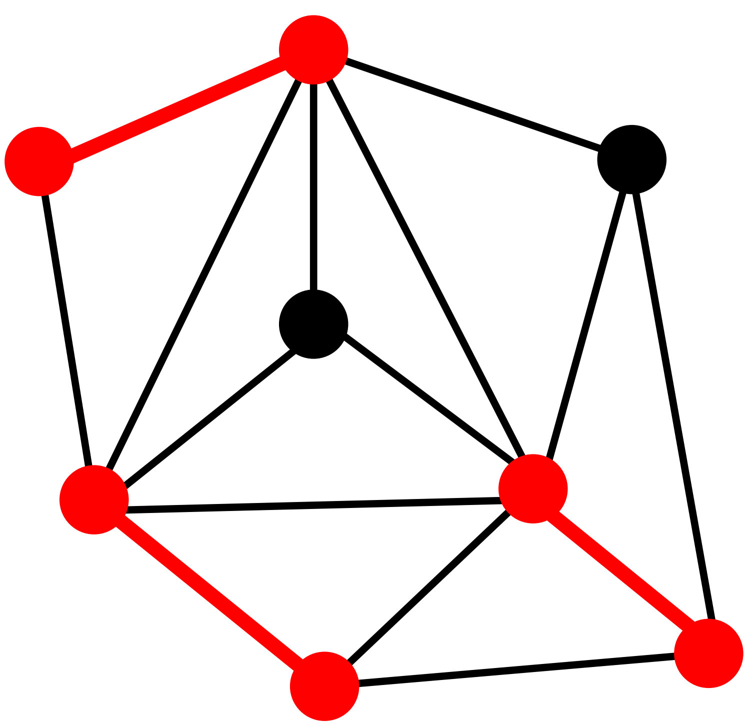
Exemplo



Exemplo



Exemplo

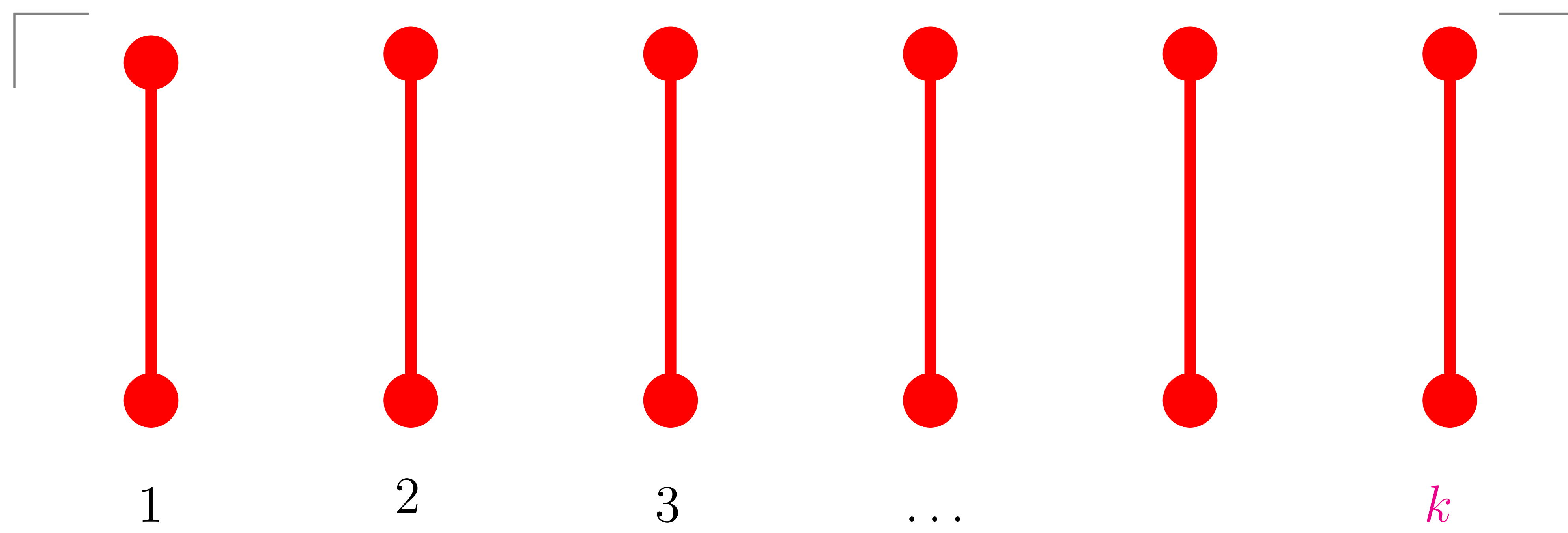


Consumo de tempo

E representado através de listas de adjacência.
$n =$ número de vértices
$m =$ número de arestas

linha	consumo de todas as execuções da linha
1	$\Theta(1)$
2	$\Theta(n+m)$
3	$\mathrm{O}(m)$
4	$\mathrm{O}(m)$
5	$\mathrm{O}(n)$
6	$\mathrm{O}(m)$
7	$\mathrm{O}(n)$
total	$\Theta(1) + \Theta(n+m) + 3\mathrm{O}(m) + 2\mathrm{O}(n) = \Theta(n+m)$

Delimitações para OPT

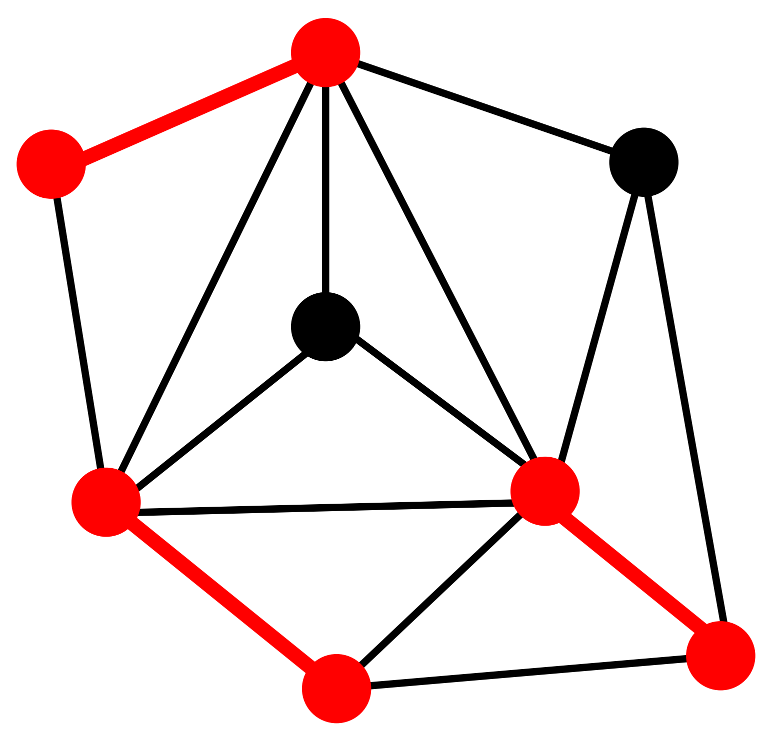


Se G possui um emparelhamento com k arestas, então

$$\mathrm{OPT} \geq k.$$

$\mathrm{OPT} =$ número mínimo de vértices de uma cobertura por vértices

Fator de aproximação



$C =$ cobertura devolvida pelo algoritmo.
G possui um emparelhamento com $|C|/2$ arestas.
Logo,

$$\frac{|C|}{2} \leq \mathrm{OPT} \quad \Rightarrow \quad |C| \leq 2\,\mathrm{OPT}.$$

Conclusão

O algoritmo APPROX-VERTEX-COVER é uma 2-aproximação polinomial.