Estruturas de Dados

Cristina Gomes Fernandes

Fila

Lista linear em que todas as inserções são feitas em uma das extremidades (fim) e todas as remoções são feitas na outra extremidade (início).

Fila

Lista linear em que todas as inserções são feitas em uma das extremidades (fim) e todas as remoções são feitas na outra extremidade (início).

Implementação sequencial:
um vetor $F[1 \mathinner{.\,.} \text{MAX}]$ e duas variáveis inteiras ini e fim.

Fila

Lista linear em que todas as inserções são feitas em uma das extremidades (fim) e todas as remoções são feitas na outra extremidade (início).

Implementação sequencial:
um vetor $F[1\,..\,\textsc{max}]$ e duas variáveis inteiras ini e fim.

Operações:

- Inicialize(F, ini, fim)
- Insira(F, ini, fim, x)
- Remova(F, ini, fim)
- Primeiro(F, ini, fim)
- Vazia(F, ini, fim)

Implementação das operações

Inicialize (F, ini, fim)

$ini \leftarrow 0$

$\mathit{fim} \leftarrow 0$

Implementação das operações

Inicialize (F, ini, fim)

1 $ini \leftarrow 0$

2 $fim \leftarrow 0$

Insira (F, ini, fim, x)

1 $fim \leftarrow (fim \text{ mod } \text{MAX}) + 1$

2 $F[fim] \leftarrow x$

Remova (F, ini, fim)

1 $ini \leftarrow (ini \text{ mod } \text{MAX}) + 1$

2 **devolva** $F[ini]$

Implementação das operações

Inicialize (F, ini, fim)

1 $ini \leftarrow 0$

2 $\mathit{fim} \leftarrow 0$

Insira $(F, ini, \mathit{fim}, x)$

1 $\mathit{fim} \leftarrow (\mathit{fim} \bmod \textsc{max}) + 1$

2 $F[\mathit{fim}] \leftarrow x$

Remova (F, ini, fim)

1 $ini \leftarrow (ini \bmod \textsc{max}) + 1$

2 **devolva** $F[ini]$

Primeiro (F, ini, fim)

1 **devolva** $F[(ini \bmod \textsc{max}) + 1]$

Implementação das operações

Inicialize (F, ini, fim)

1 $ini \leftarrow 0$
2 $fim \leftarrow 0$

Insira (F, ini, fim, x)

1 $fim \leftarrow (fim \bmod \text{MAX}) + 1$
2 $F[fim] \leftarrow x$

Remova (F, ini, fim)

1 $ini \leftarrow (ini \bmod \text{MAX}) + 1$
2 **devolva** $F[ini]$

Primeiro (F, ini, fim)

1 **devolva** $F[(ini \bmod \text{MAX}) + 1]$

Vazia (F, ini, fim)

1 **se** $ini = fim$
2 **então devolva** VERDADE
3 **senão devolva** FALSO

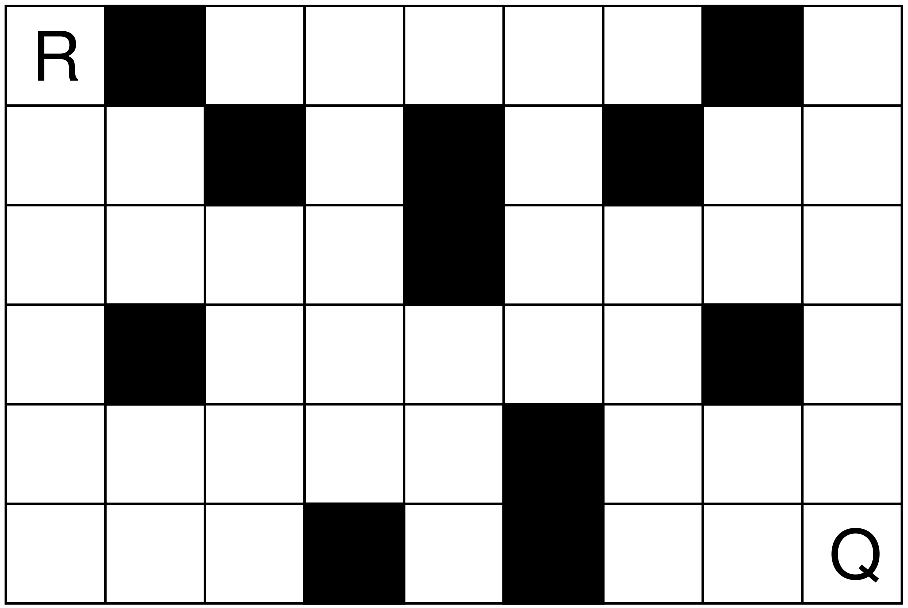
Aplicações

- cálculo de distância
  (Problema do ratinho)

- implementação de Round-Robin
  (escalonamento de processos numa CPU, RRDtool)

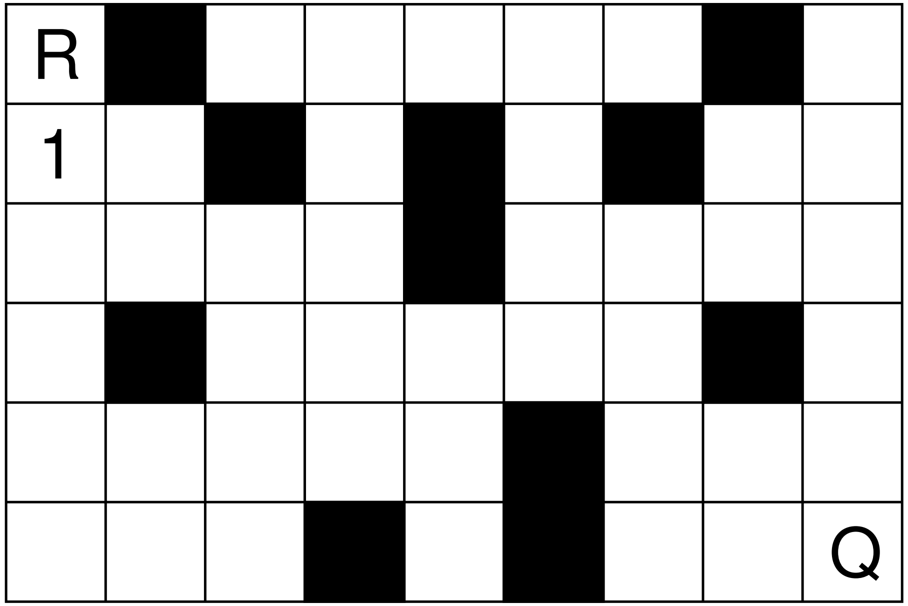
  Round-Robin é uma técnica usada para distribuição de carga entre servidores ou para interpolar valores em um gráfico.

Problema do ratinho



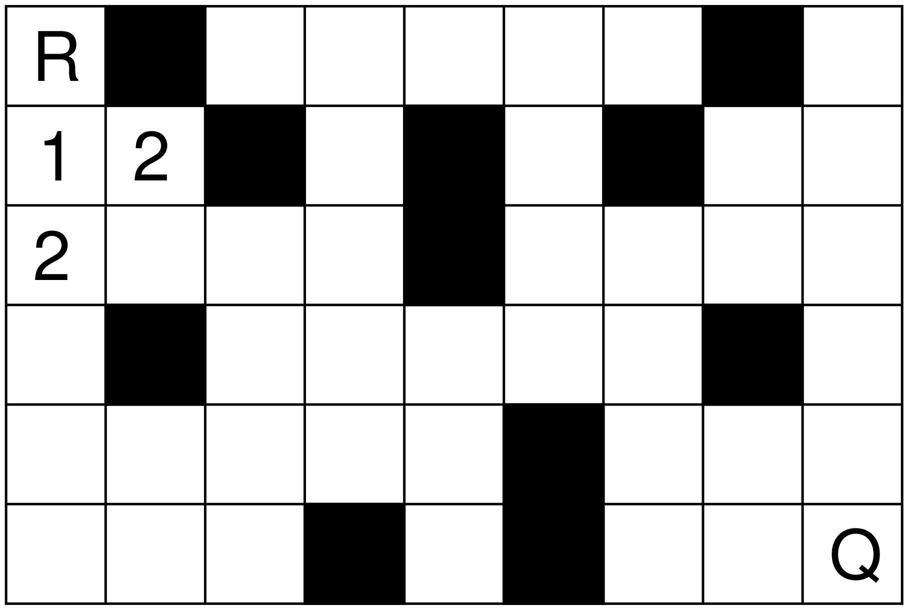
Problema: Dado um labirinto e as posições do rato e do queijo no labirinto, determinar o número mínimo de passos que o rato precisa para chegar até o queijo.

Problema do ratinho



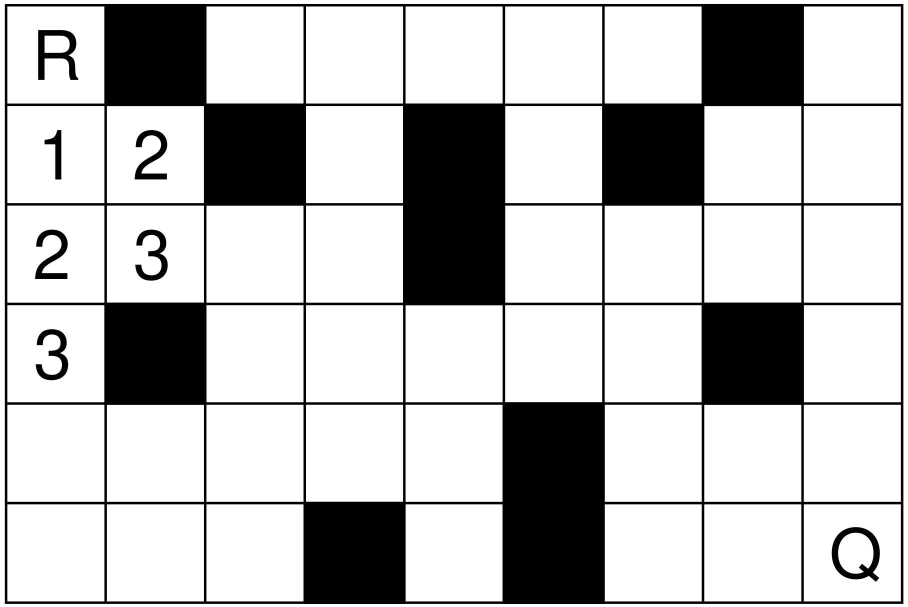
Problema: Dado um labirinto e as posições do rato e do queijo no labirinto, determinar o número mínimo de passos que o rato precisa para chegar até o queijo.

Problema do ratinho



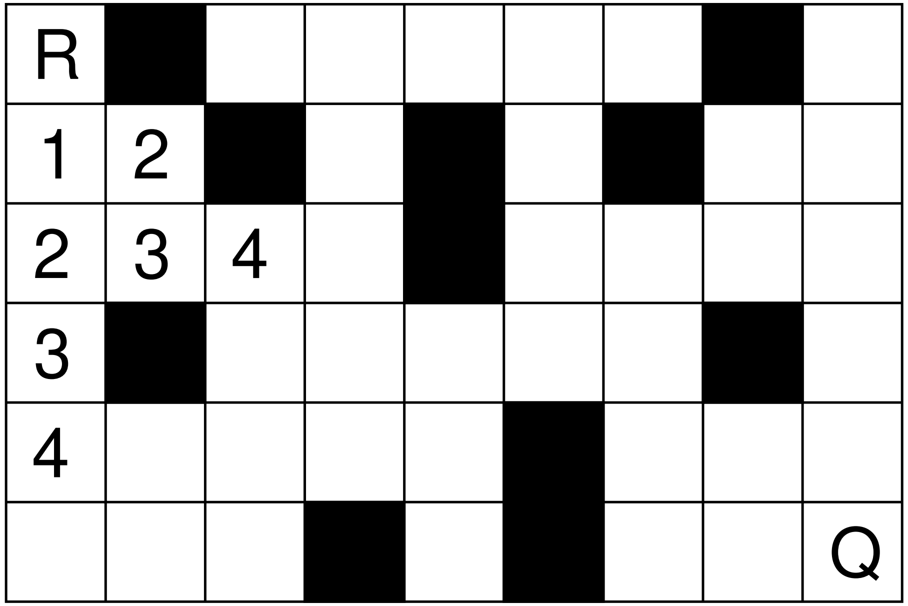
Problema: Dado um labirinto e as posições do rato e do queijo no labirinto, determinar o número mínimo de passos que o rato precisa para chegar até o queijo.

Problema do ratinho



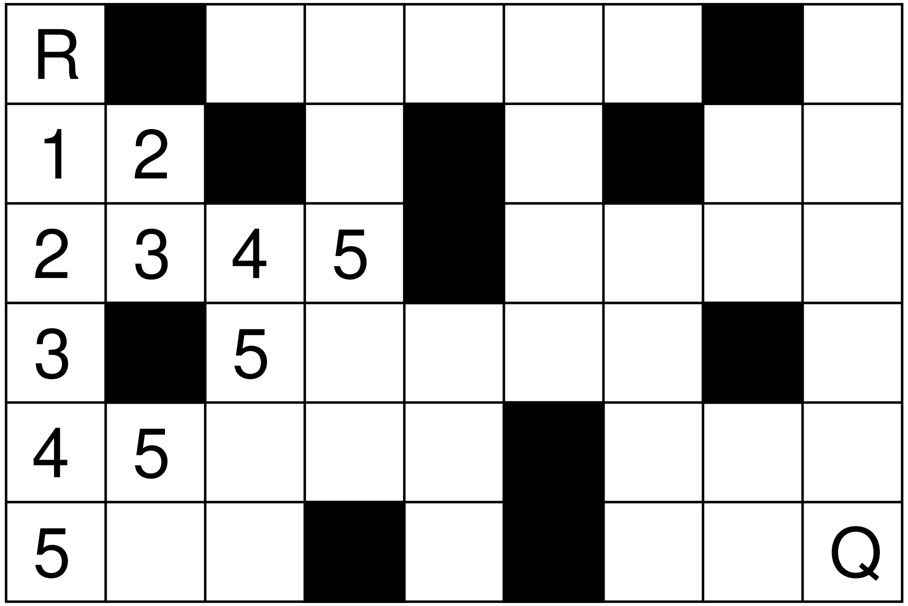
Problema: Dado um labirinto e as posições do rato e do queijo no labirinto, determinar o número mínimo de passos que o rato precisa para chegar até o queijo.

Problema do ratinho



Problema: Dado um labirinto e as posições do rato e do queijo no labirinto, determinar o número mínimo de passos que o rato precisa para chegar até o queijo.

Problema do ratinho



Problema: Dado um labirinto e as posições do rato e do queijo no labirinto, determinar o número mínimo de passos que o rato precisa para chegar até o queijo.

Problema do ratinho

R		8	7	8	9	10		14
1	2		6		10		12	13
2	3	4	5		9	10	11	12
3		5	6	7	8	9		13
4	5	6	7	8		10	11	12
5	6	7		9		11	12	Q

Problema: Dado um labirinto e as posições do rato e do queijo no labirinto, determinar o número mínimo de passos que o rato precisa para chegar até o queijo.

Problema do ratinho

Entrada: inteiros m e n, matriz $A_{m\times n}$ tal que

$$A[i,j] = \begin{cases} -1 & \text{se } (i,j) \text{ é parede} \\ 0 & \text{se } (i,j) \text{ é livre} \end{cases}$$

e posições xr, yr do rato e xq, yq do queijo.
Saída: a distância do rato até o queijo.

Vamos assumir que a matriz A vem com uma moldura de -1's para simplificar o algoritmo. Tal moldura, se ausente, pode ser acrescentada numa fase anterior.

Cálculo de distância

Distância (A, xr, yr, xq, yq)

para todo i **e** j **faça**
    se $A[i,j] = 0$
        então $D[i,j] \leftarrow \infty$
        senão $D[i,j] \leftarrow -1$
$D[xr, yr] \leftarrow 0$
Inicialize(F, ini, fim)
Insira(F, ini, fim, xr, yr)

continua...

Cálculo de distância

enquanto não Vazia(F, ini, fim) **faça**
 x, $y \leftarrow$ Remova(F, ini, fim)
 se $D[x-1, y] = \infty$
  então $D[x-1, y] \leftarrow D[x, y] + 1$
    Insira($F, ini, fim, x-1, y$)
 se $D[x+1, y] = \infty$
  então $D[x+1, y] \leftarrow D[x, y] + 1$
    Insira($F, ini, fim, x+1, y$)
 se $D[x, y-1] = \infty$
  então $D[x, y-1] \leftarrow D[x, y] + 1$
    Insira($F, ini, fim, x, y-1$)
 se $D[x, y+1] = \infty$
  então $D[x, y+1] \leftarrow D[x, y] + 1$
    Insira($F, ini, fim, x, y+1$)
devolva $D[xq, yq]$

Filas de prioridade

Tipo abstrato de dados para armazenar um conjunto S de elementos, cada um com uma chave, que suporta as seguintes operações:

- Insira(S, x)
- Máximo(S)
- Extraia-Max(S)

Filas de prioridade

Tipo abstrato de dados para armazenar um conjunto S de elementos, cada um com uma chave, que suporta as seguintes operações:

- Insira(S, x)
- Máximo(S)
- Extraia-Max(S)

Como implementá-lo de um jeito eficiente?

Heap

Um vetor $A[1 \dots m]$ é um (max-)heap se

$$A[\lfloor i/2 \rfloor] \geq A[i]$$

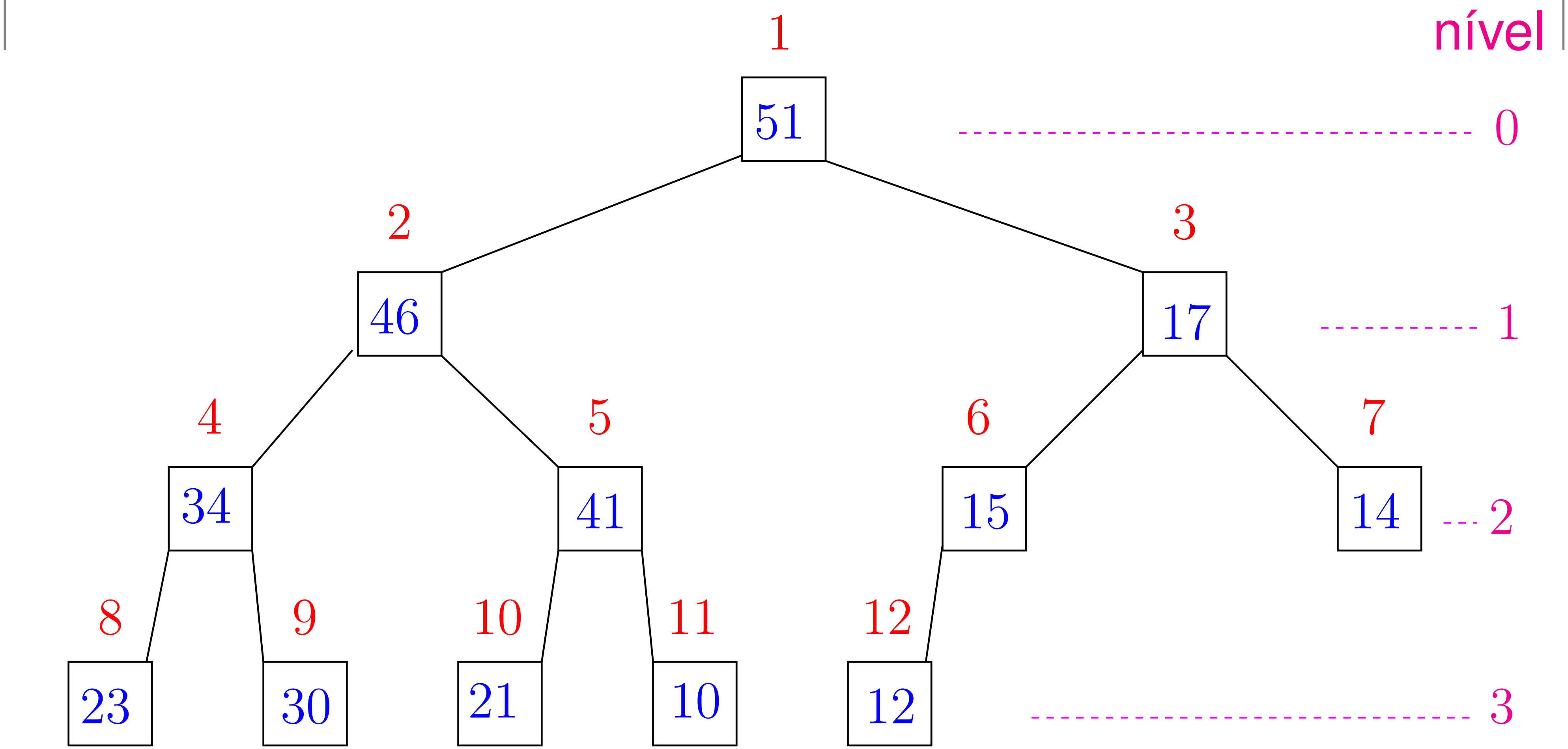
para todo $i = 2, 3, \dots, m$.

De uma forma mais geral, $A[j \dots m]$ é um heap se

$$A[\lfloor i/2 \rfloor] \geq A[i]$$

para todo $i = 2j, 2j+1, 4j, \dots, 4j+3, 8j, \dots, 8j+7, \dots$.
Neste caso também diremos que a subárvore com raiz j é um heap.

Exemplo



1	2	3	4	5	6	7	8	9	10	11	12
51	46	17	34	41	15	14	23	30	21	10	12

Desce-Heap

Recebe $A[1\,..\,m]$ e $i \geq 1$ tais que subárvores com raiz $2i$ e $2i+1$ são heaps e rearranja A de modo que subárvore com raiz i seja heap.

DESCE-HEAP (A, m, i)

$e \leftarrow 2i$
$d \leftarrow 2i+1$
se $e \leq m$ e $A[e] > A[i]$
    então $maior \leftarrow e$
    senão $maior \leftarrow i$
se $d \leq m$ e $A[d] > A[maior]$
    então $maior \leftarrow d$
se $maior \neq i$
    então $A[i] \leftrightarrow A[maior]$
        DESCE-HEAP $(A, m, maior)$

Sobe-Heap

Exercício: Escreva uma função SOBE-HEAP (A, m, i) que recebe $A[1 \mathinner{..} m]$ e $i \geq 1$ tais que $A[1 \mathinner{..} m]$ é um heap exceto pela condição $A[\lfloor i/2 \rfloor] \geq A[i]$ que pode estar violada, e rearranja A de modo que passe a ser um heap.

Sua função deve ter complexidade $O(\lg m)$.

Sobe-Heap

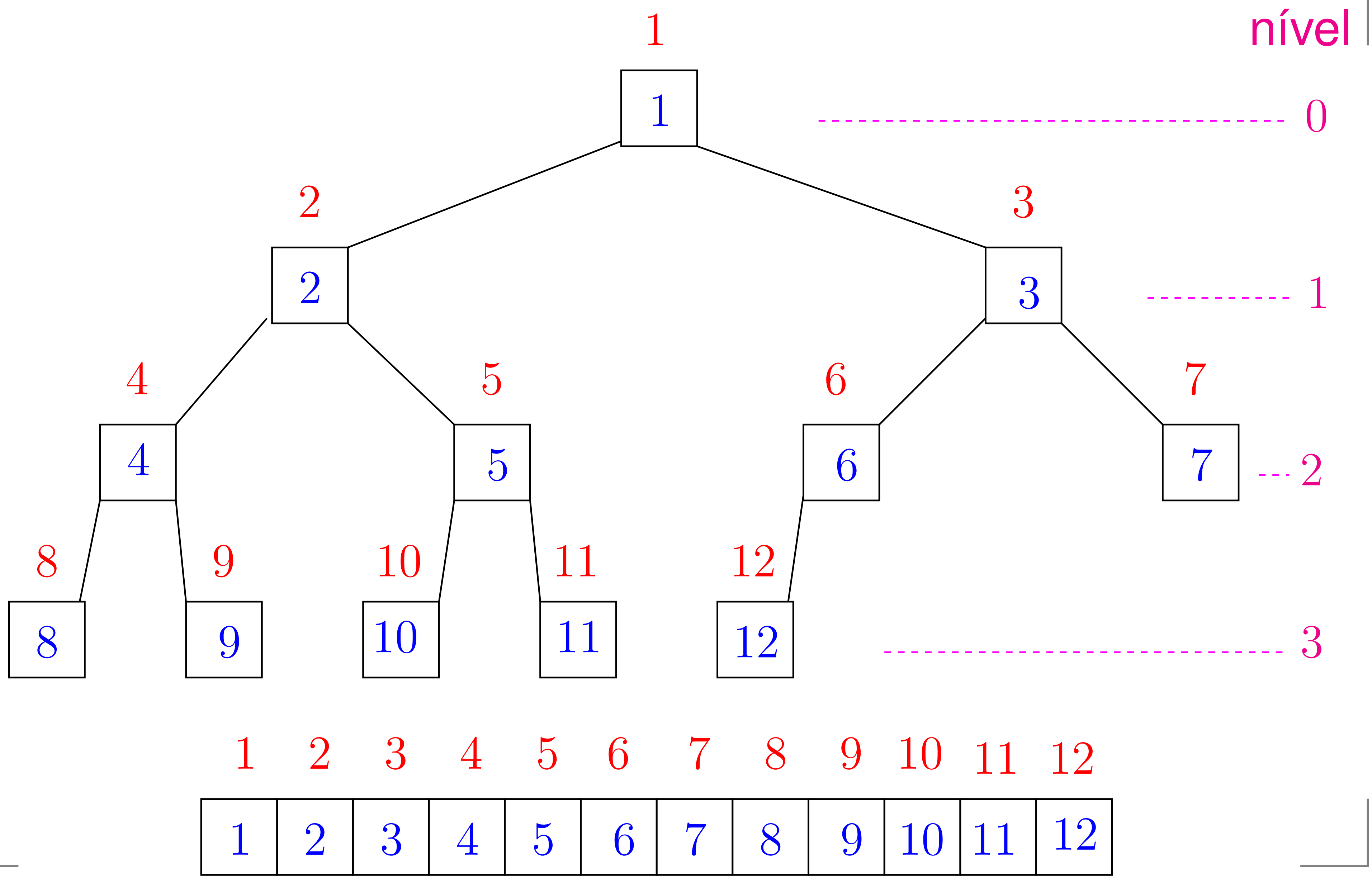
Exercício: Escreva uma função SOBE-HEAP (A, m, i) que recebe $A[1\,..\,m]$ e $i \geq 1$ tais que $A[1\,..\,m]$ é um heap exceto pela condição $A[\lfloor i/2 \rfloor] \geq A[i]$ que pode estar violada, e rearranja A de modo que passe a ser um heap.

Sua função deve ter complexidade $O(\lg m)$.

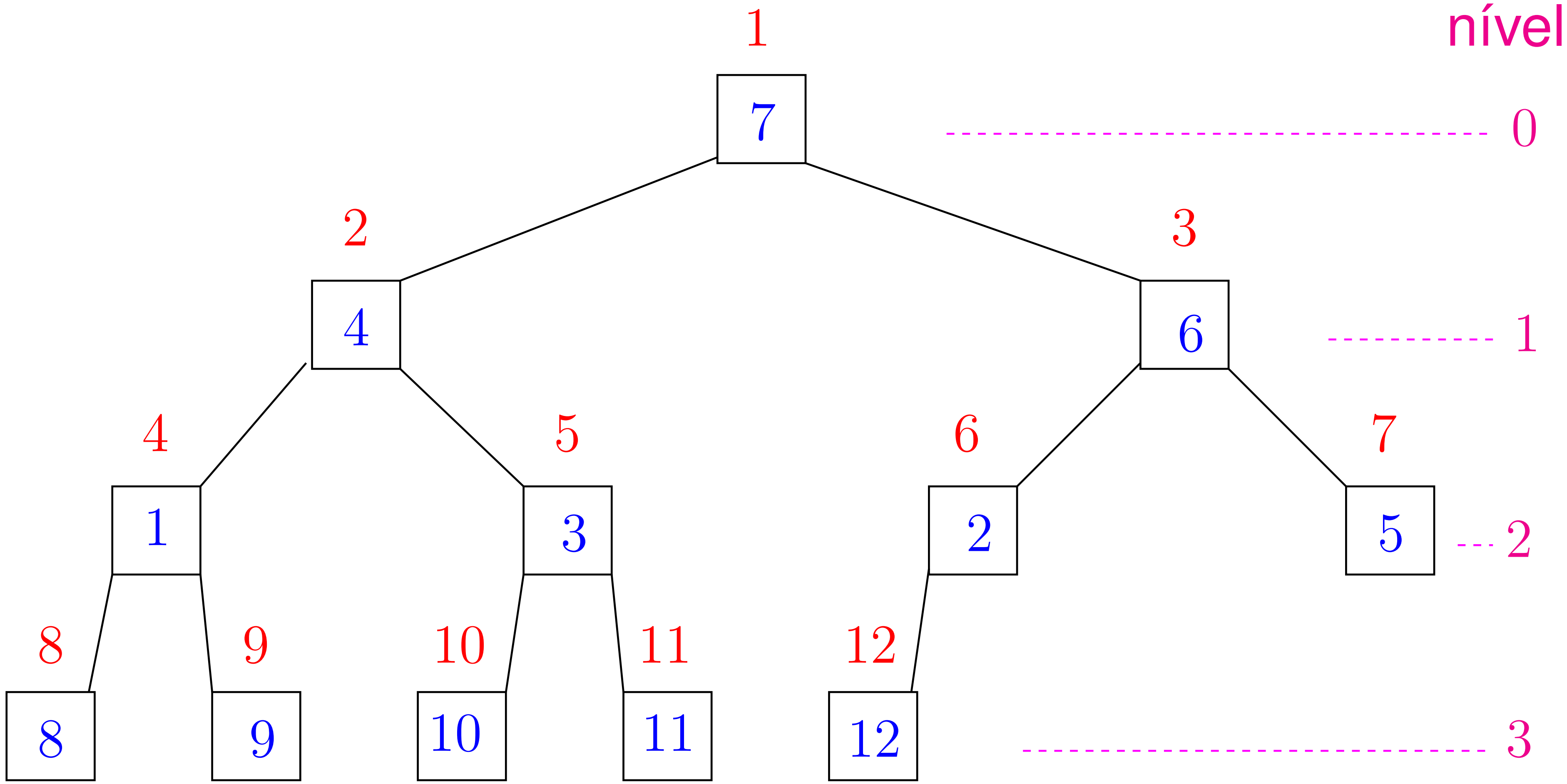
CONSTRÓI-HEAP (A, m)
1 **para** $i \leftarrow 2$ **até** n **faça**
2   SOBE-HEAP (A, i, i)

Qual é a complexidade dessa implementação do CONSTRÓI-HEAP?

Exemplo



Exemplo



1	2	3	4	5	6	7	8	9	10	11	12
7	4	6	1	3	2	5	8	9	10	11	12

Construção de um heap

Recebe um vetor $A[1..n]$ e rearranja A para que seja heap.

CONSTRÓI-HEAP (A, n)
1 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
2 DESCE-HEAP (A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i+1, \ldots, n$ são raízes de heaps.

$T(n)$:= consumo de tempo no pior caso

Construção de um heap

Recebe um vetor $A[1\,.\,.\,n]$ e rearranja A para que seja heap.

CONSTRÓI-HEAP (A, n)
1 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
2 DESCE-HEAP (A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i+1, \ldots, n$ são raízes de 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 ????.

Construção de um heap

Recebe um vetor $A[1\,..\,n]$ e rearranja A para que seja heap.

CONSTRÓI-HEAP (A, n)
1 **para** $i \leftarrow \lfloor n/2 \rfloor$ **decrescendo até** 1 **faça**
2 DESCE-HEAP (A, n, i)

Relação invariante:

(i0) no início de cada iteração, $i+1, \ldots, n$ são raízes de 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 $\mathrm{O}(n)$.

Heap sort

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

HEAPSORT (A, n)
CONSTRÓI-HEAP (A, n) $\quad \triangleright$ pré-processamento
$m \leftarrow n$
para $i \leftarrow n$ **decrescendo até** 2 **faça**
$\quad A[1] \leftrightarrow A[i]$
$\quad m \leftarrow m - 1$
$\quad$ DESCE-HEAP $(A, m, 1)$

Exercícios

Exercício 9.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 9.B
Mostre que um heap $A[1\,..\,m]$ tem no máximo $\lceil m/2^{h+1} \rceil$ nós com altura h.

Exercício 9.C
Mostre que $\lceil m/2^{h+1} \rceil \leq m/2^h$ quando $h \leq \lfloor \lg m \rfloor$.

Exercício 9.D
Mostre que um heap $A[1\,..\,m]$ tem no mínimo $\lfloor m/2^{h+1} \rfloor$ nós com altura h.

Exercício 9.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 9.F

Mostre que a solução da recorrência

$$\begin{aligned} T(1) &= 1 \\ T(k) &\leq T(2k/3) + 5 \quad \text{para } k \geq 2 \end{aligned}$$

é $\mathrm{O}(\log k)$. Mais geral: mostre que se $T(k) = T(2k/3) + \mathrm{O}(1)$ então $\mathrm{O}(\log k)$.
(Curiosidade: Essa é a recorrência do DESCE-HEAP (A, m, i) se interpretarmos k como sendo o número de nós na subárvore com raiz i).

Exercício 9.G

Escreva uma versão iterativa do algoritmo DESCE-HEAP. Faça uma análise do consumo de tempo do algoritmo.

Mais exercícios ainda

Exercício 9.H

Discuta a seguinte variante do algoritmo DESCE-HEAP:

D-H (A, m, i)

$e \leftarrow 2i$

$d \leftarrow 2i + 1$

se $e \leq m$ e $A[e] > A[i]$

  então $A[i] \leftrightarrow A[e]$

    D-H (A, m, e)

se $d \leq m$ e $A[d] > A[i]$

  então $A[i] \leftrightarrow A[d]$

    D-H (A, m, d)

Limites inferiores

CLRS 8.1

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 $\mathrm{O}(n \lg n)$.

Existe algoritmo assintoticamente melhor?

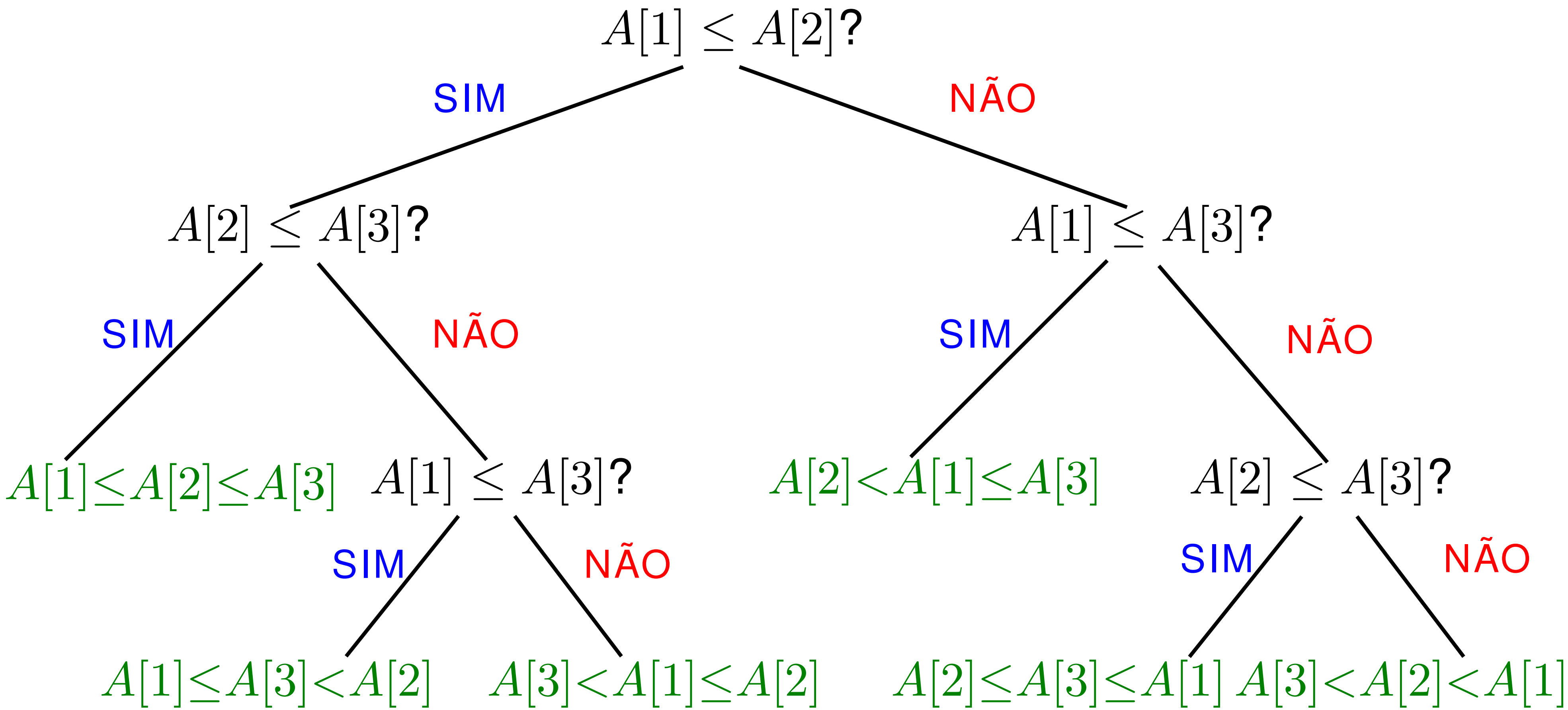
NÃO, se o algoritmo é baseado em comparações.

Prova?

Qualquer algoritmo baseado em comparações é uma "árvore de decisão".

Exemplo

ORDENA-POR-INSERÇÃO $(A[1\,.\,.\,3])$:



Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

Número de comparações, no pior caso?
Resposta: altura, h, da árvore de decisão.

Limite inferior

Considere uma árvore de decisão para $A[1\,..\,n]$.

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 $A[1\,..\,n]$.

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: 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,

$$h \geq \lg(n!) \geq \frac{1}{2} n \lg n.$$

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 $\sum_{k=n/2}^{n} \lg k$. Use as técnicas de CLRS A.2.