Melhores momentos

AULA PASSADA

Heap

Um vetor $A[1 \ldots m]$ é um max-heap se

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

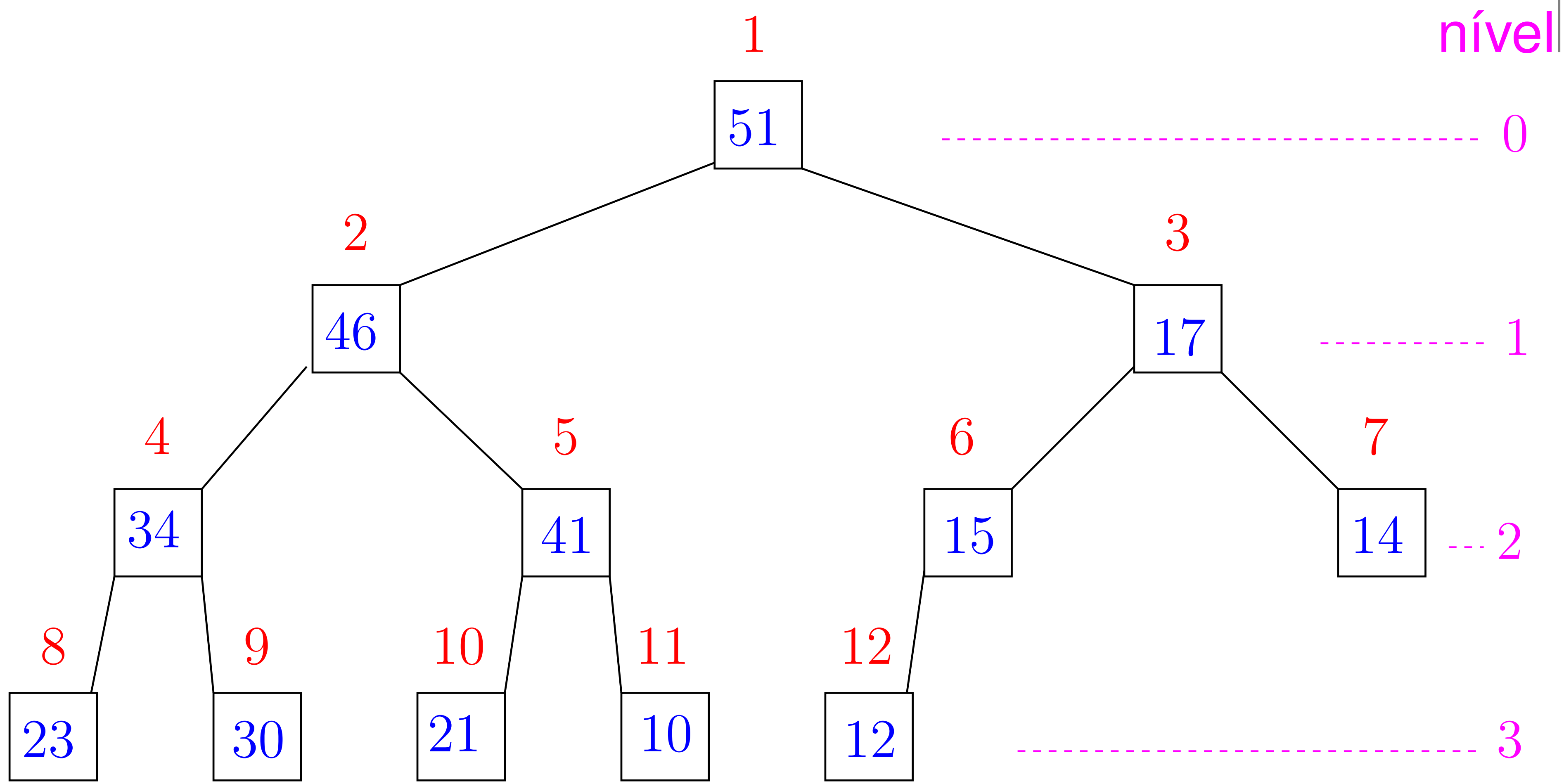
para todo $i = 2, 3, \ldots, m$.

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

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

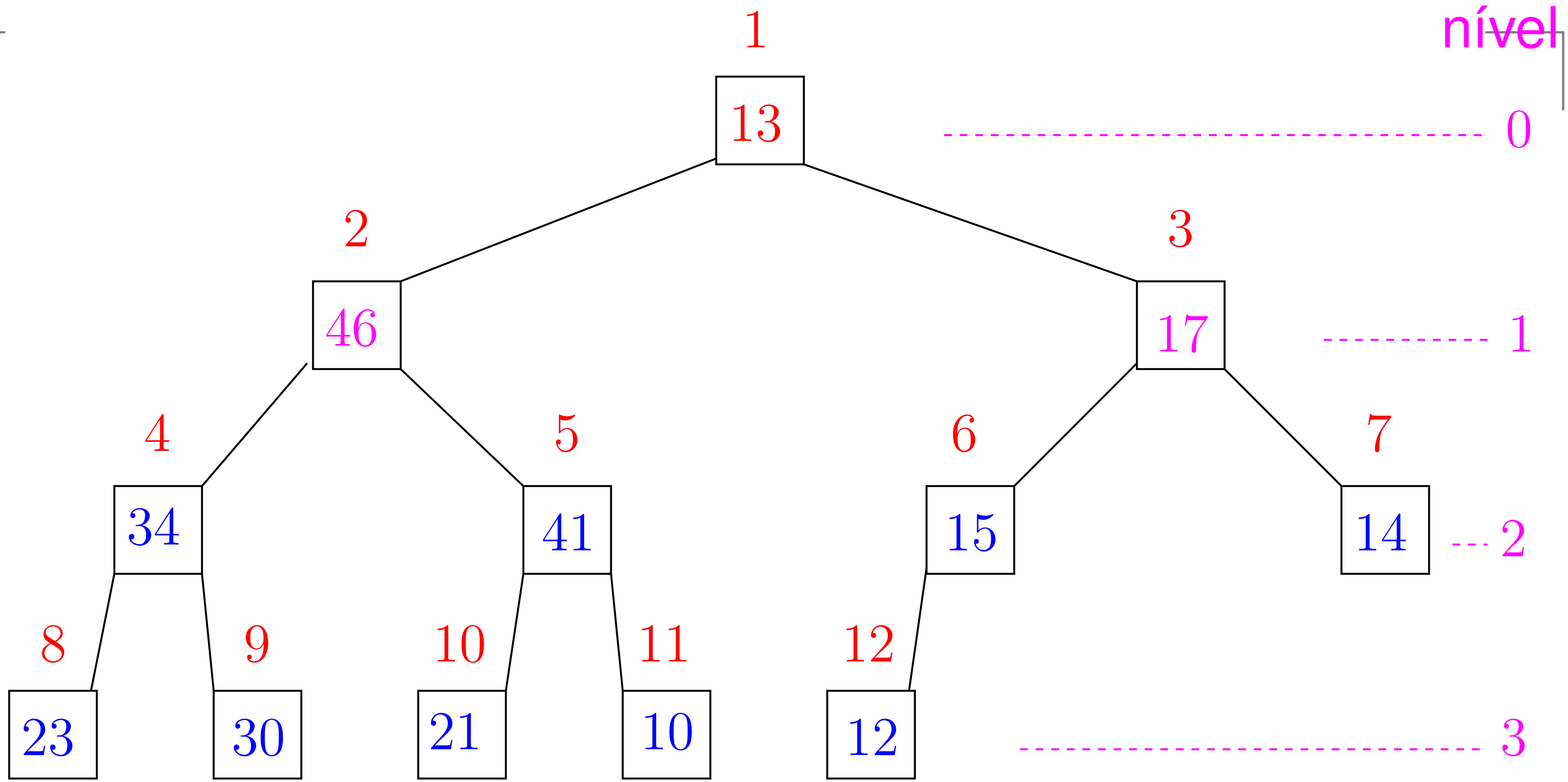
para todo $i = 2j, 2j+1, 4j, \ldots, 4j+3, 8j, \ldots, 8j+7, \ldots$.
Neste caso também diremos que a subárvore com raiz j é um max-heap.

Max-heap



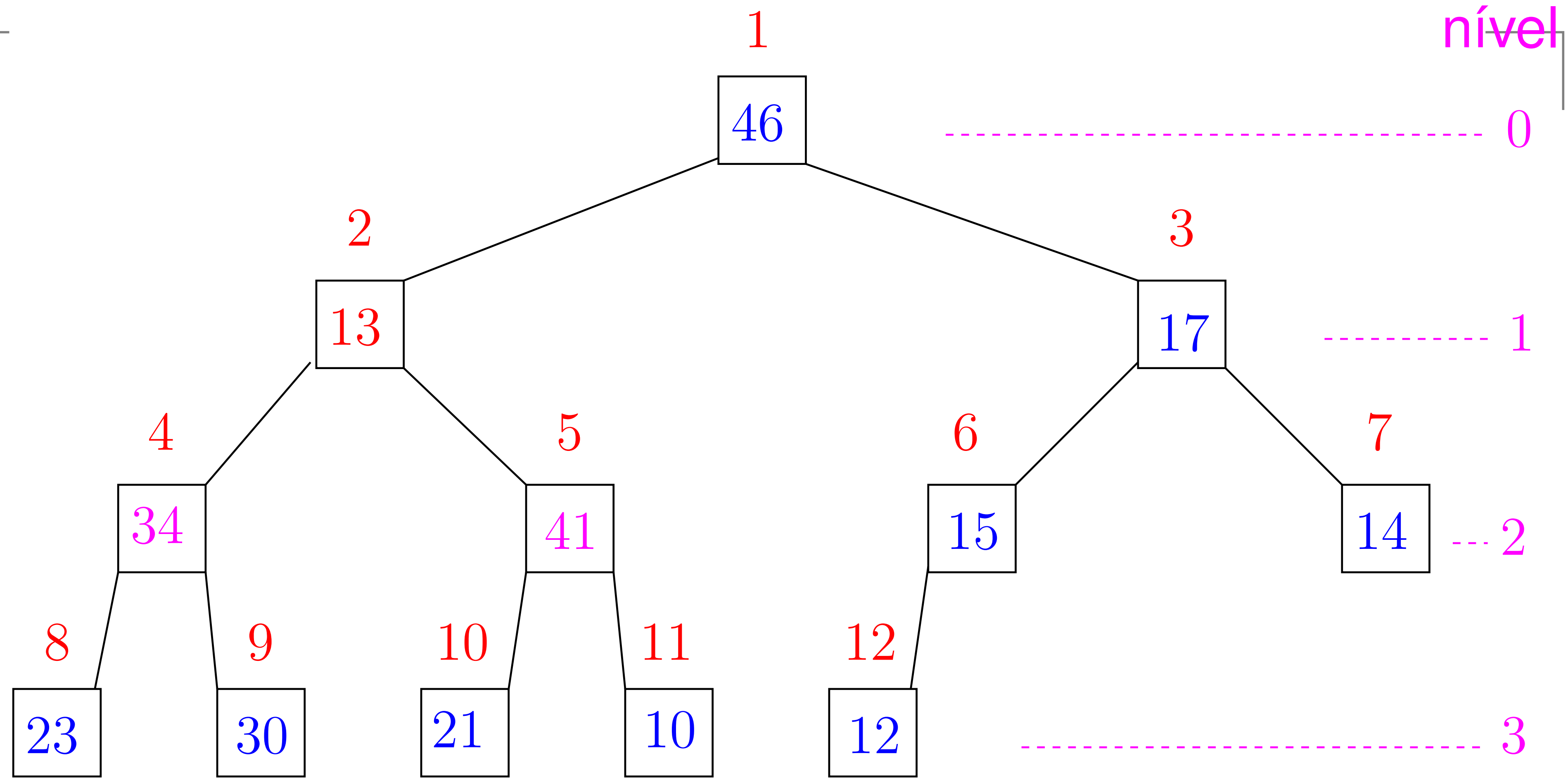
1	2	3	4	5	6	7	8	9	10	11	12
51	46	17	34	41	15	14	23	30	21	10	12

Rotina básica de manipulação de max-heap



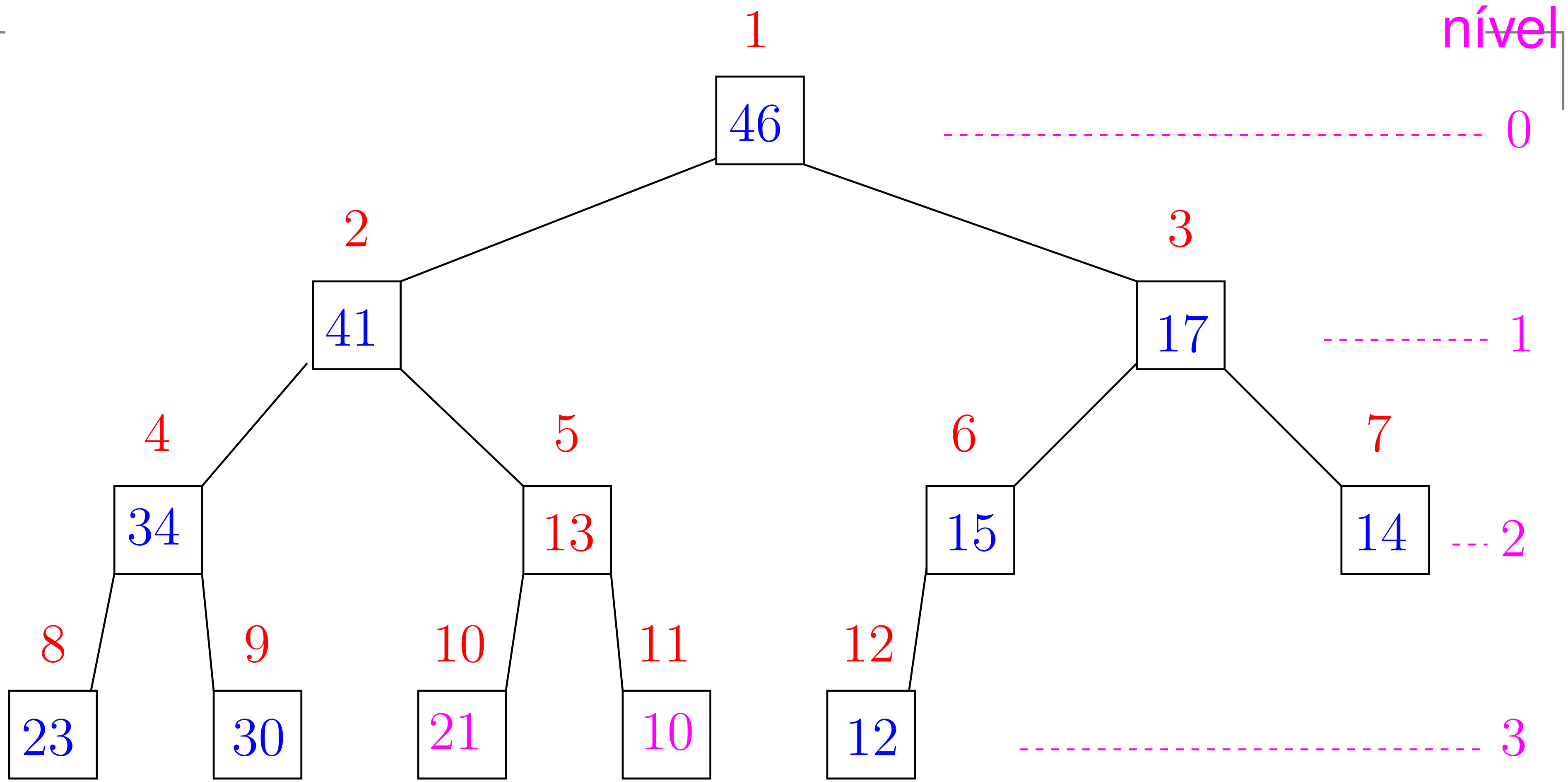
1	2	3	4	5	6	7	8	9	10	11	12
13	46	17	34	41	15	14	23	30	21	10	12

Rotina básica de manipulação de max-heap



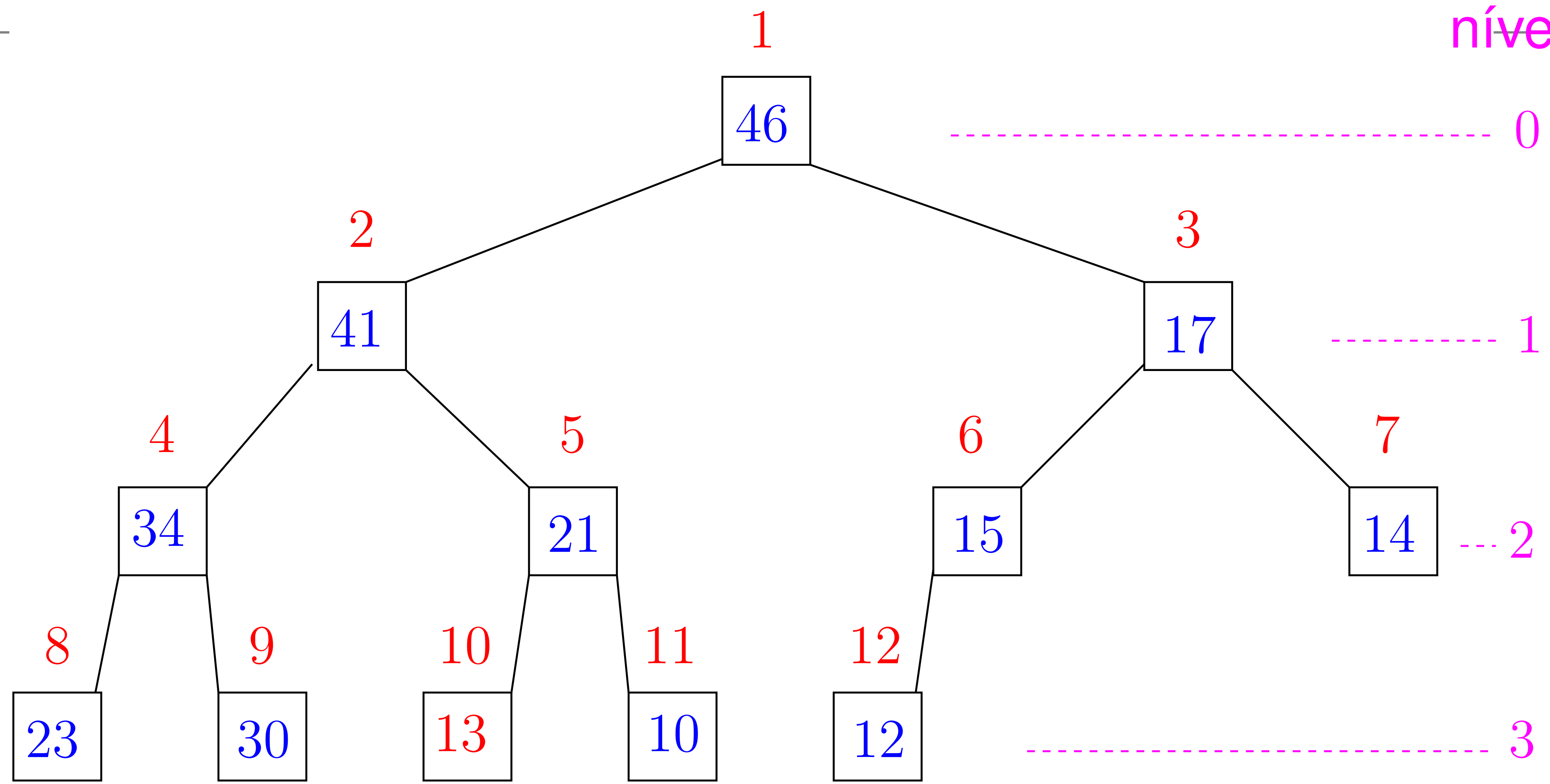
1	2	3	4	5	6	7	8	9	10	11	12
46	13	17	34	41	15	14	23	30	21	10	12

Rotina básica de manipulação de max-heap



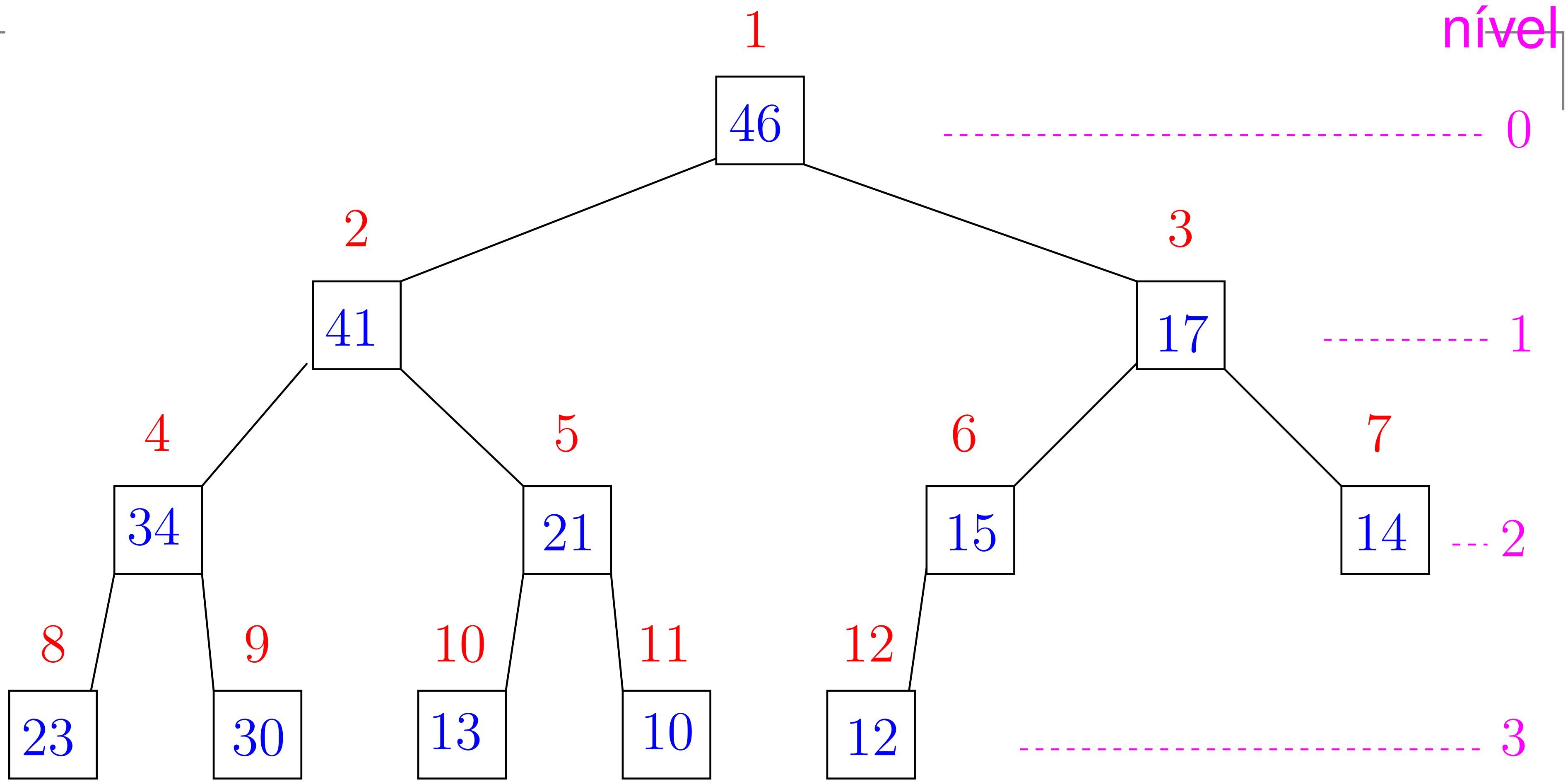
1	2	3	4	5	6	7	8	9	10	11	12
46	41	17	34	13	15	14	23	30	21	10	12

Rotina básica de manipulação de max-heap



1	2	3	4	5	6	7	8	9	10	11	12
46	41	17	34	21	15	14	23	30	13	10	12

Rotina básica de manipulação de max-heap



1	2	3	4	5	6	7	8	9	10	11	12
46	41	17	34	21	15	14	23	30	13	10	12

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)

$e \leftarrow 2i$
$d \leftarrow 2i + 1$
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 $A[i] \leftrightarrow A[maior]$
        MAX-HEAPIFY $(A, m, maior)$

Consumo de tempo

O consumo de tempo do algoritmo MAX-HEAPIFY é $\mathrm{O}(\lg m)$ (ou melhor ainda, $\mathrm{O}(\lg \frac{m}{i})$).

AULA 8

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)
devolva $A[1]$

Consome tempo $\Theta(1)$.

HEAP-EXTRACT-MAX (A, m) $\triangleright$ $m \geq 1$
$max \leftarrow A[1]$
$A[1] \leftarrow A[m]$
$m \leftarrow m - 1$
MAX-HEAPIFY $(A, m, 1)$
devolva max

Consome tempo $\mathrm{O}(\lg m)$.

Implementação com max-heap

HEAP-INCREASE-KEY $(A, i, prior)$ $\triangleright$ $prior \geq A[i]$

$A[i] \leftarrow prior$
enquanto $i > 1$ e $A[\lfloor i/2 \rfloor] < A[i]$ **faça**
  $A[i] \leftrightarrow A[\lfloor i/2 \rfloor]$
  $i \leftarrow \lfloor i/2 \rfloor$

Consome tempo $\mathrm{O}(\lg m)$.

MAX-HEAP-INSERT $(A, m, prior)$

$m \leftarrow m + 1$
$A[m] \leftarrow -\infty$
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 $A[1 \mathinner{..} m]$ e armazenar os elementos restantes, em forma de max-heap, no vetor $A[1 \mathinner{..} m-1]$.

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   $i \leftarrow m$
3   **enquanto** $i > 1$ e $A[\lfloor i/2 \rfloor] < A[i]$ **faça**
4     $A[\lfloor i/2 \rfloor] \leftrightarrow A[i]$ $\triangleright$ troca
5     $i \leftarrow \lfloor i/2 \rfloor$

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, $A[1 \mathinner{..} m]$ é um max-heap exceto talvez pela violação da relação $A[\lfloor i/2 \rfloor] \geq A[i]$.

Leftist heap

TAOCP 5.2.3 Vol. 3

Além das operações usuais de uma fila com prioridades permite que a união ("merge") de duas filas seja feita facilmente.

Estrutura simples, ultrapassada por outras como binomial heaps (CLRS 19) e fibonacci heaps (CLRS 20).

Árvores esquerdistas

Cada nó x tem quatro campos:

1. $esq[x]$: filho esquerdo de x;
2. $dir[x]$: filho direito de x;
3. $dist[x]$: menor comprimento de um caminho de x a NIL.

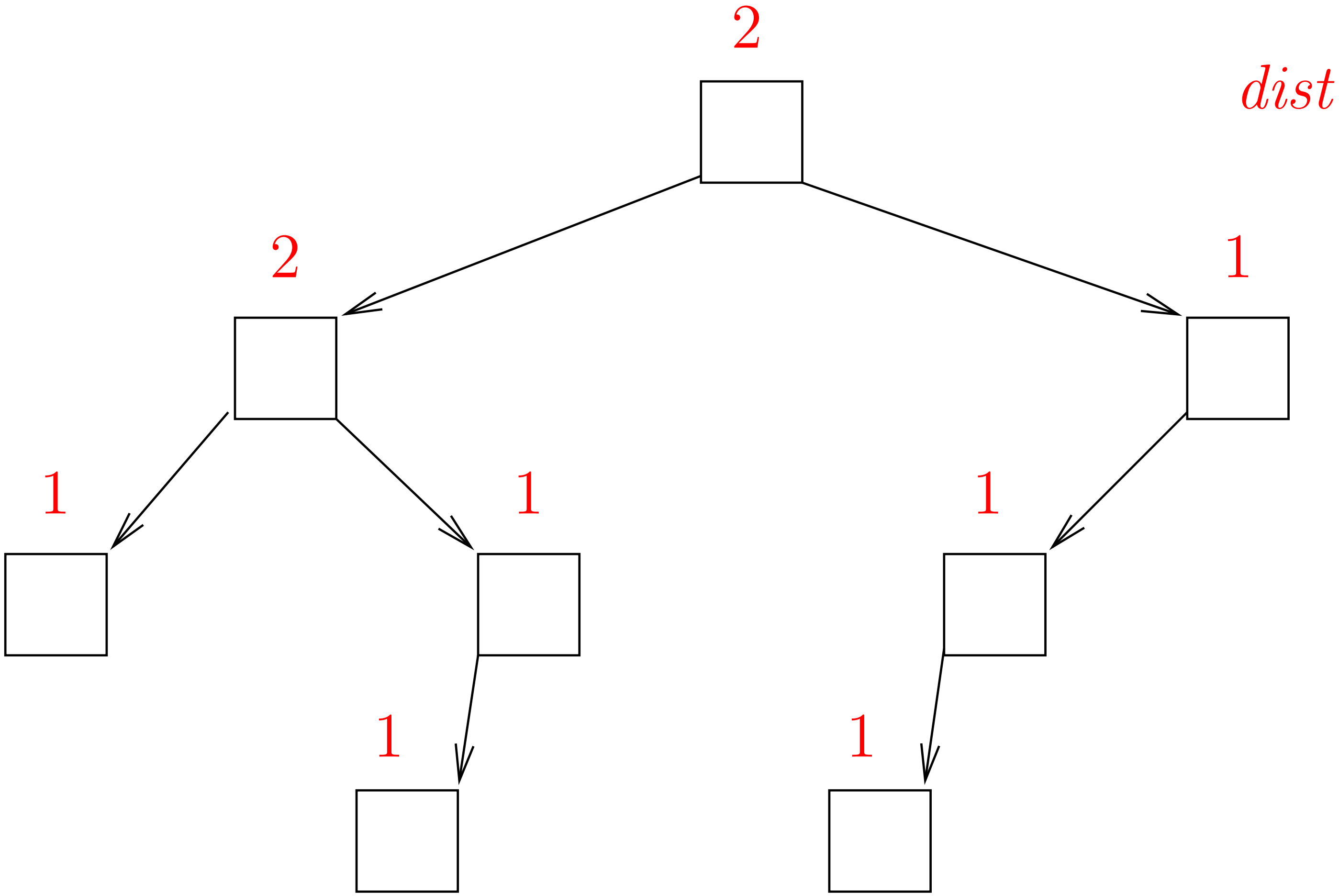
DIST (x)

se $x = \text{NIL}$

  então devolva 0

  senão devolva $1 + \min\{\text{DIST}(esq[x]), \text{DIST}(dir[x])\}$

Exemplo



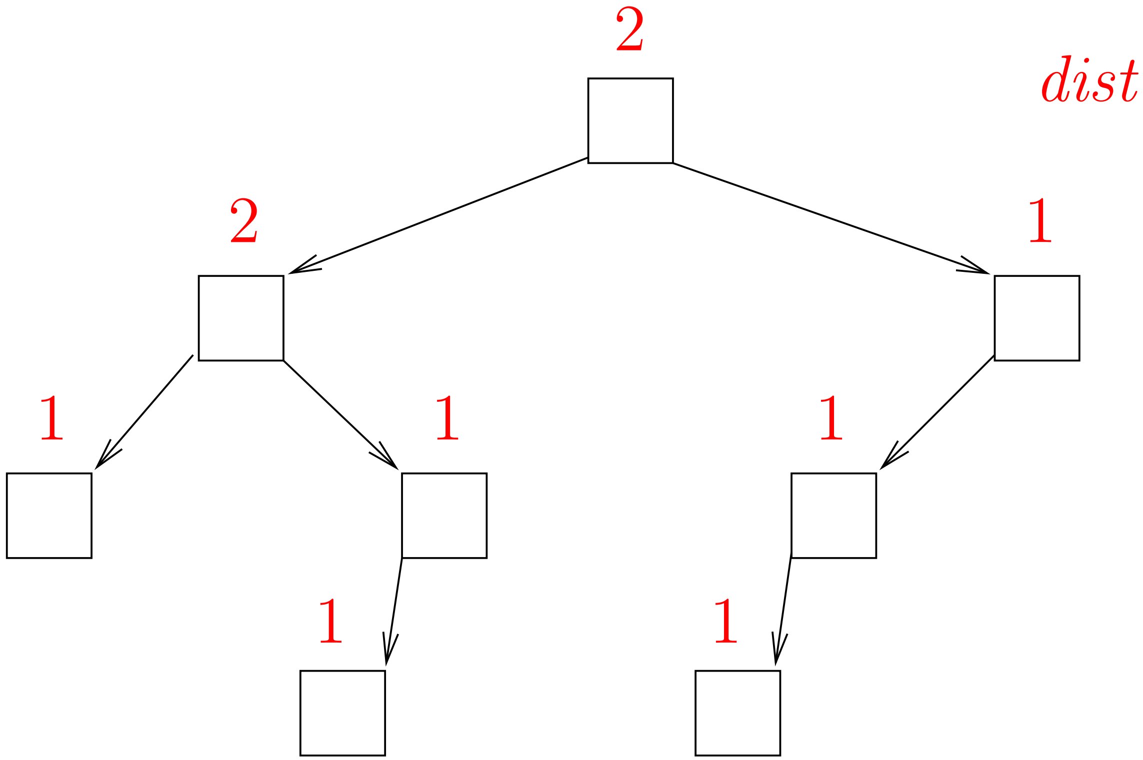
Árvores esquerdistas

Uma árvore é esquerdista se

$$dist[esq[x]] \geq dist[dir[x]]$$

para todo nó x ($dist[\text{NIL}] = 0$).

Exemplo 1:



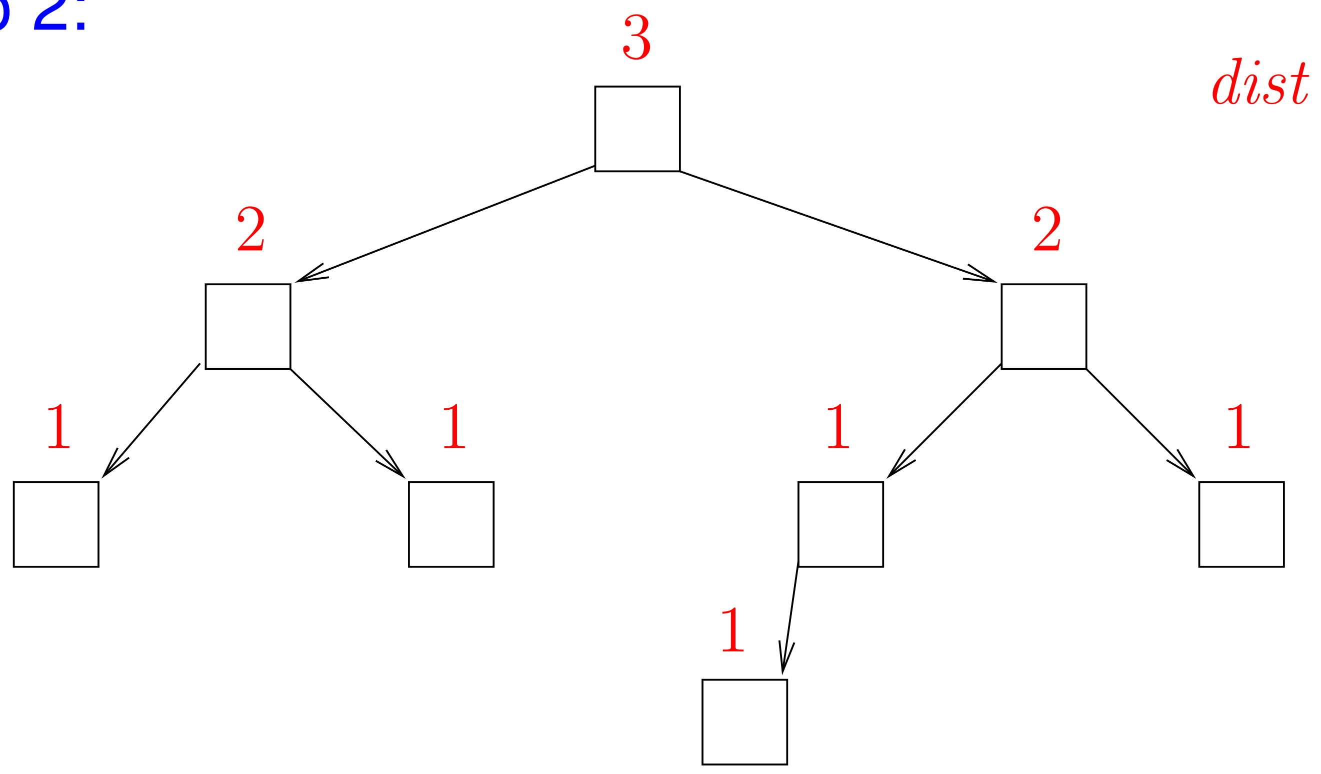
Árvores esquerdistas

Uma árvore é esquerdista se

$$dist[esq[x]] \geq dist[dir[x]]$$

para todo nó x ($dist[\text{NIL}] = 0$).

Exemplo 2:



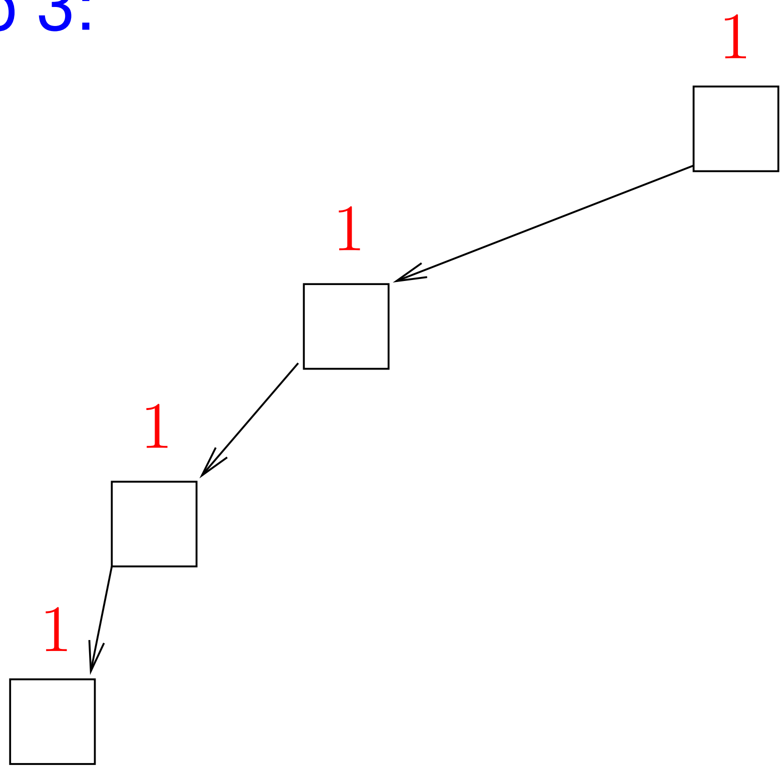
Árvores esquerdistas

Uma árvore é esquerdista se

$$dist[esq[x]] \geq dist[dir[x]]$$

para todo nó x ($dist[\mathrm{NIL}] = 0$).

Exemplo 3:



$dist$

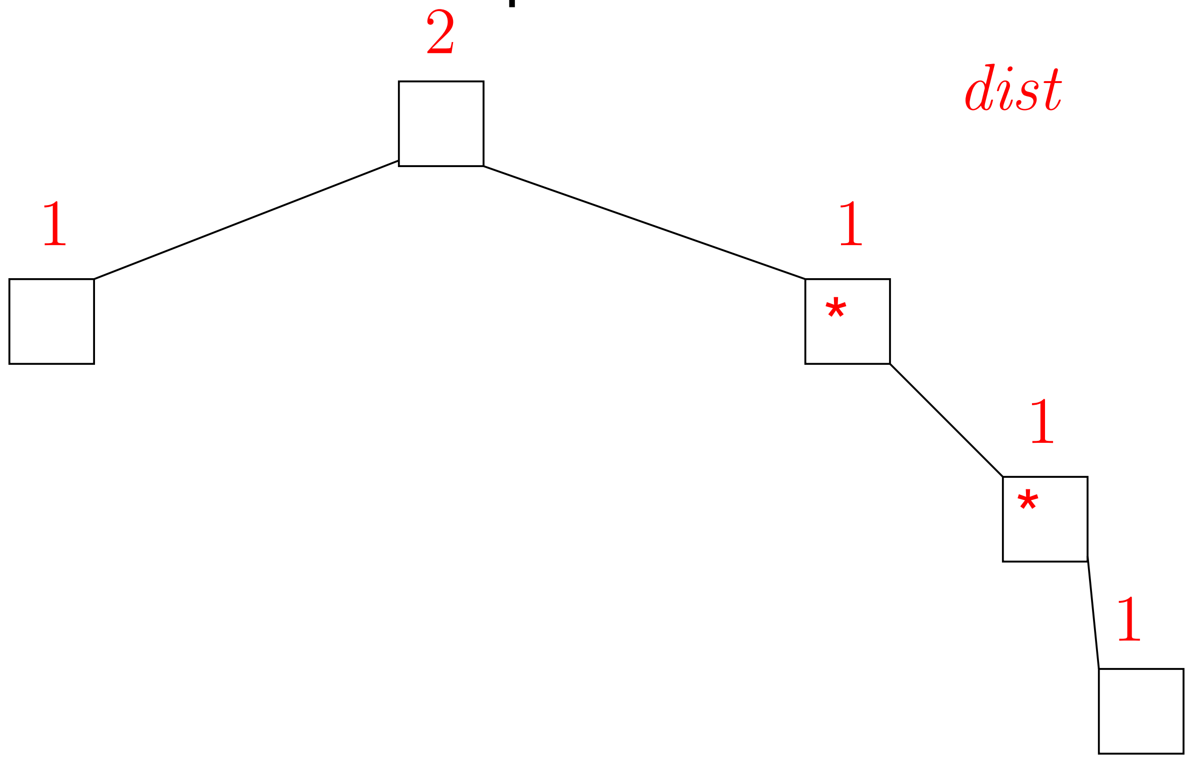
Árvores esquerdistas

Uma árvore é esquerdista se

$$dist[esq[x]] \geq dist[dir[x]]$$

para todo nó x ($dist[\text{NIL}] = 0$).

Exemplo 4: árvore não-esquerdista



Caminho direitista

O caminho direitista de um nó x é a seqüência

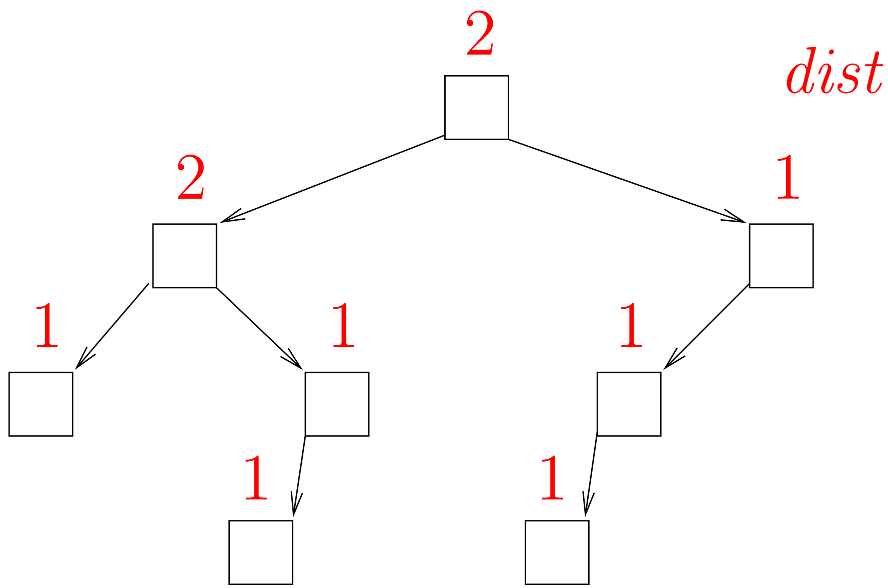
$$\langle x, dir[x], dir[dir[x]], \ldots, \text{NIL}\rangle.$$

$dcomp[x] :=$ número de nós no caminho direitista de x

$tam[x] :=$ número de nós na árvore de raiz x

Se x é um nó de uma árvore esquerdista, então

$$dist[x] = dcomp[x].$$



Fato importante

$tam[x]$:= número de nós na árvore de raiz x

Se x é um nó de uma árvore esquerdista, então

$$tam[x] \geq 2^{dist[x]} - 1.$$

Fato importante

$tam[x]$:= número de nós na árvore de raiz x

Se x é um nó de uma árvore esquerdista, então

$$tam[x] \geq 2^{dist[x]} - 1.$$

Prova: Seja $d := dist[x]$.

Se $d = 1$, então $tam[x] \geq 1 = 2^d - 1$.

Suponha que $d \geq 2$ e que a desigualdade vale para $d - 1$.

Temos que $dist[dir[x]] = d - 1$ e que existe um nó y na árvore de raiz $esq[x]$ tal que $dist[y] = d - 1$.

Fato importante

$tam[x]$:= número de nós na árvore de raiz x

Fato 2. Se x é um nó de uma árvore esquerdista, então

$$tam[x] \geq 2^{dist[x]} - 1.$$

Prova: (continuação)
Logo,

$$\begin{aligned} tam[x] &= tam[esq[x]] + tam[dir[x]] + 1 \\ &\geq tam[y] + tam[dir[x]] + 1 \\ &\overset{\text{hi}}{\geq} 2^{d-1} - 1 + 2^{d-1} - 1 + 1 \\ &= 2^d - 1 \end{aligned}$$

Conseqüência

Se x é um nó de uma árvore esquerdista, então

$$dist[x] = dcomp[x] \leq \lfloor \lg(tam[x] + 1) \rfloor = \mathrm{O}(tam[x]).$$

Em particular:

Se x é raiz de uma árvore esquerdista com m nós,

$$dist[x] = dcomp[x] \leq \lfloor \lg(m + 1) \rfloor = \mathrm{O}(\lg m).$$

Prova: $m \geq 2^d - 1 \Rightarrow m + 1 \geq 2^d \Rightarrow \lfloor \lg(m+1) \rfloor \geq d$.

Heap esquerdista

H := árvore

$raiz[H]$:= raiz de H

$prior[x]$:= prioridade do nó x

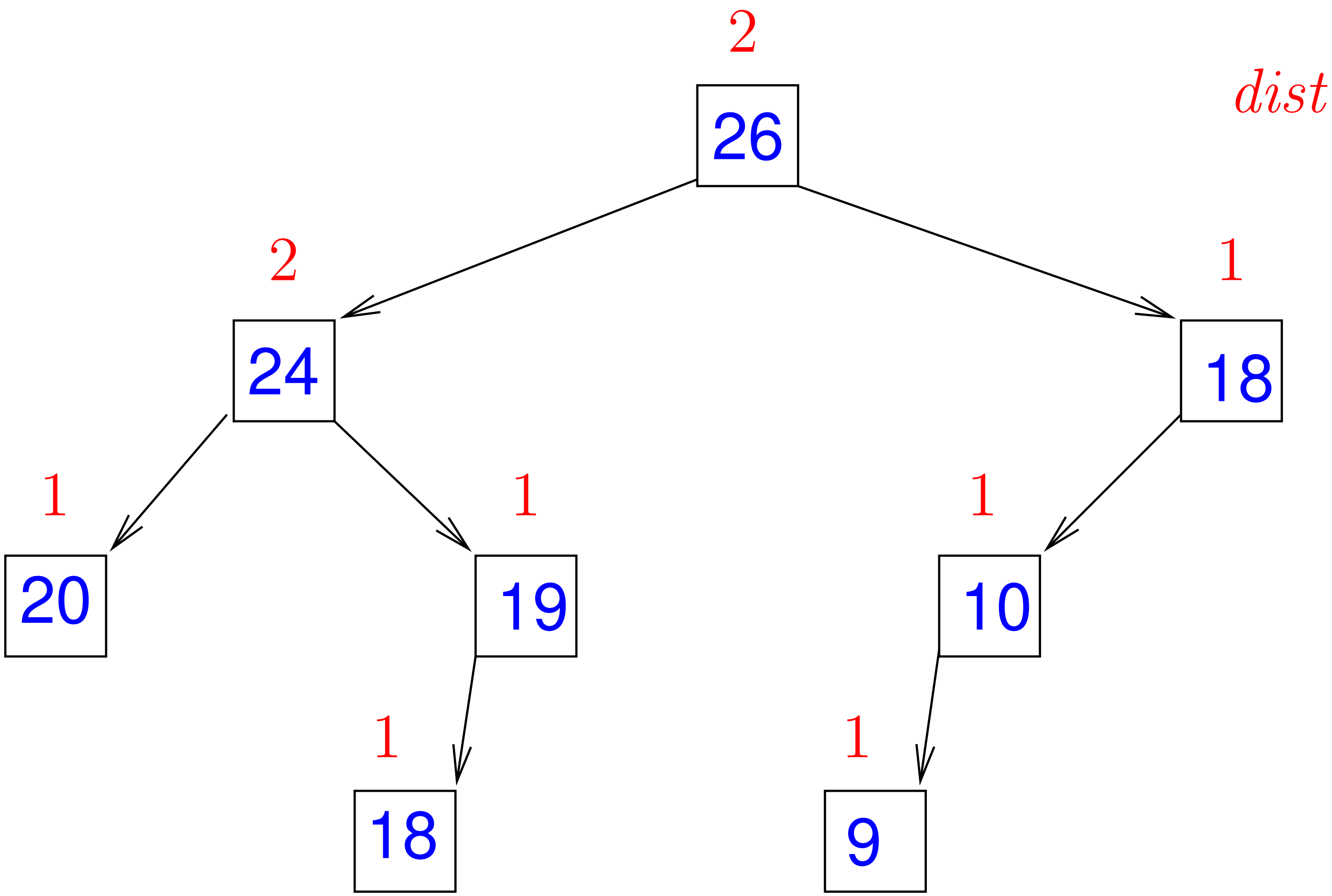
$pai[x]$:= pai do nó x

Um heap esquerdista H é uma árvore esquerdista que satisfaz

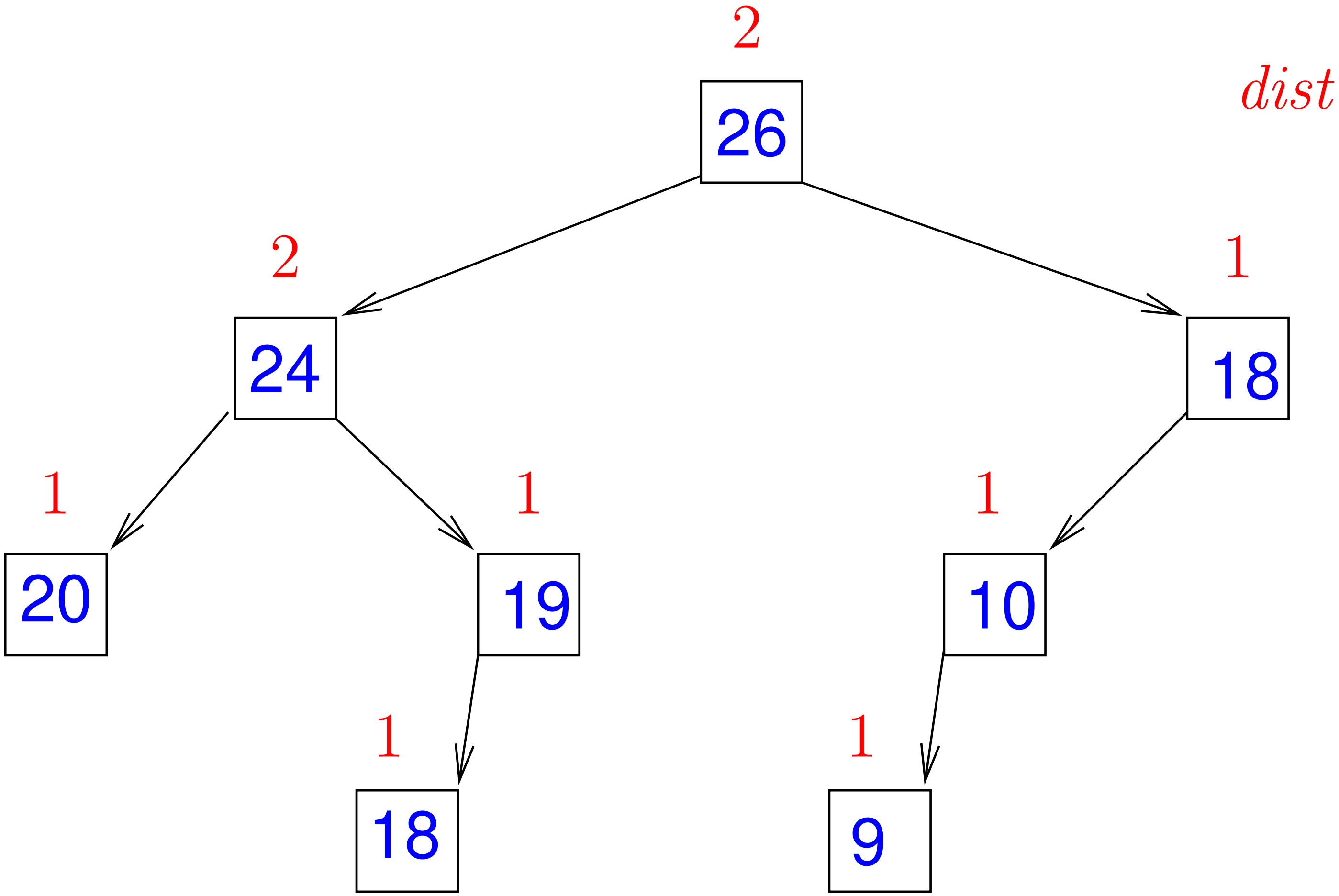
$$prior[pai[x]] \geq prior[x]$$

para todo nó $x \neq raiz[H]$.

Heap esquerdista

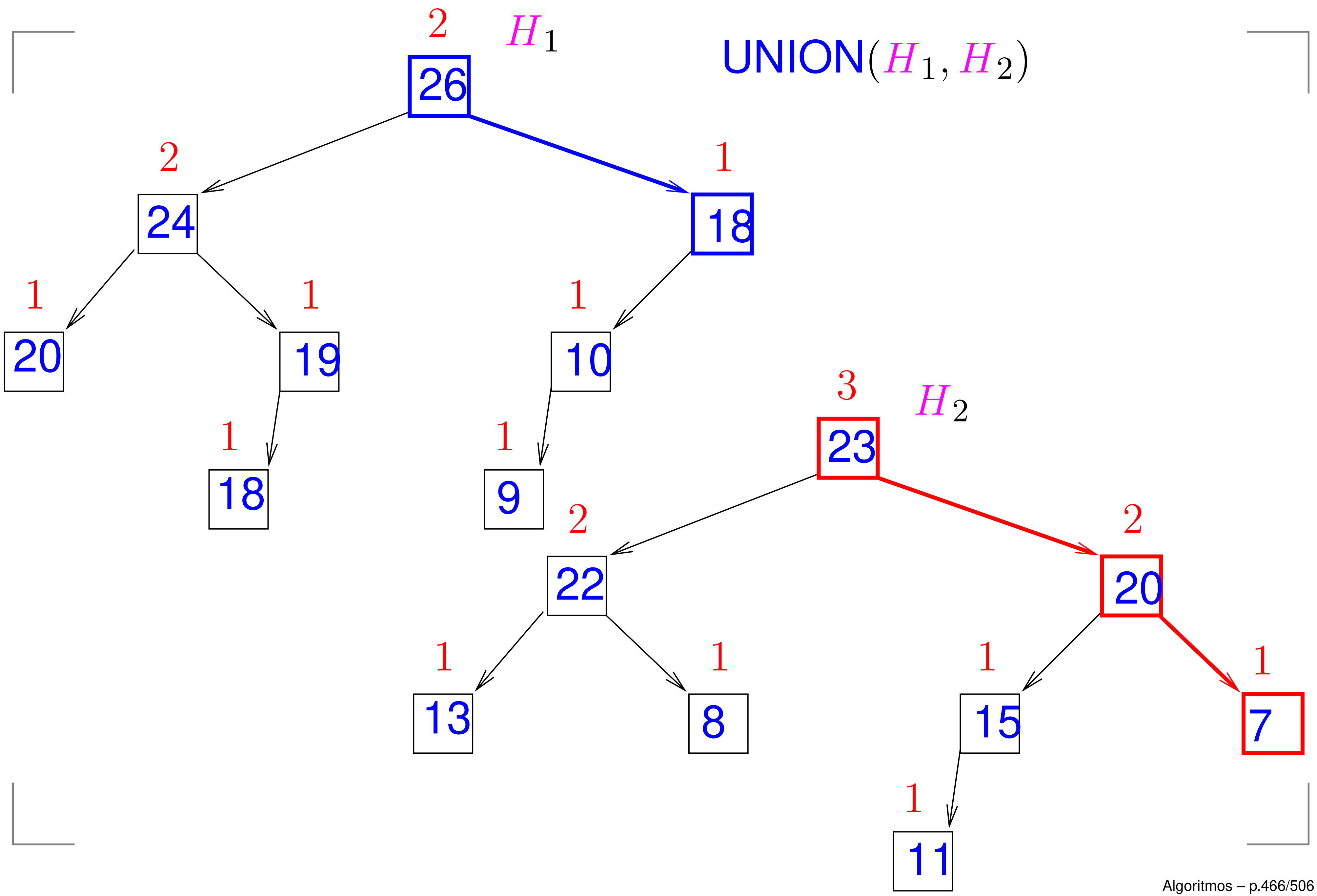


Heap esquerdista



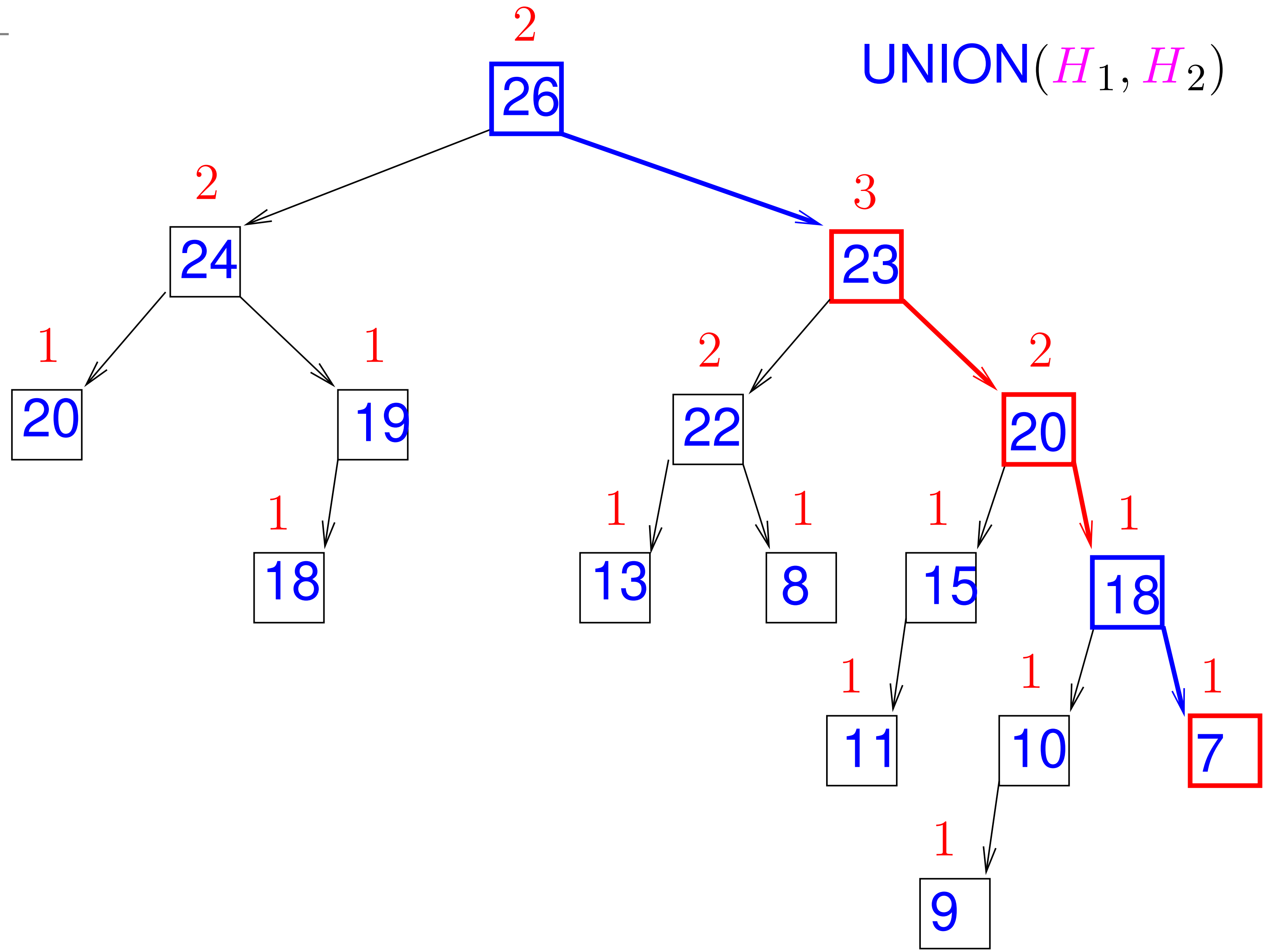
Rotina básica de manipulação

UNION(H_1, H_2)

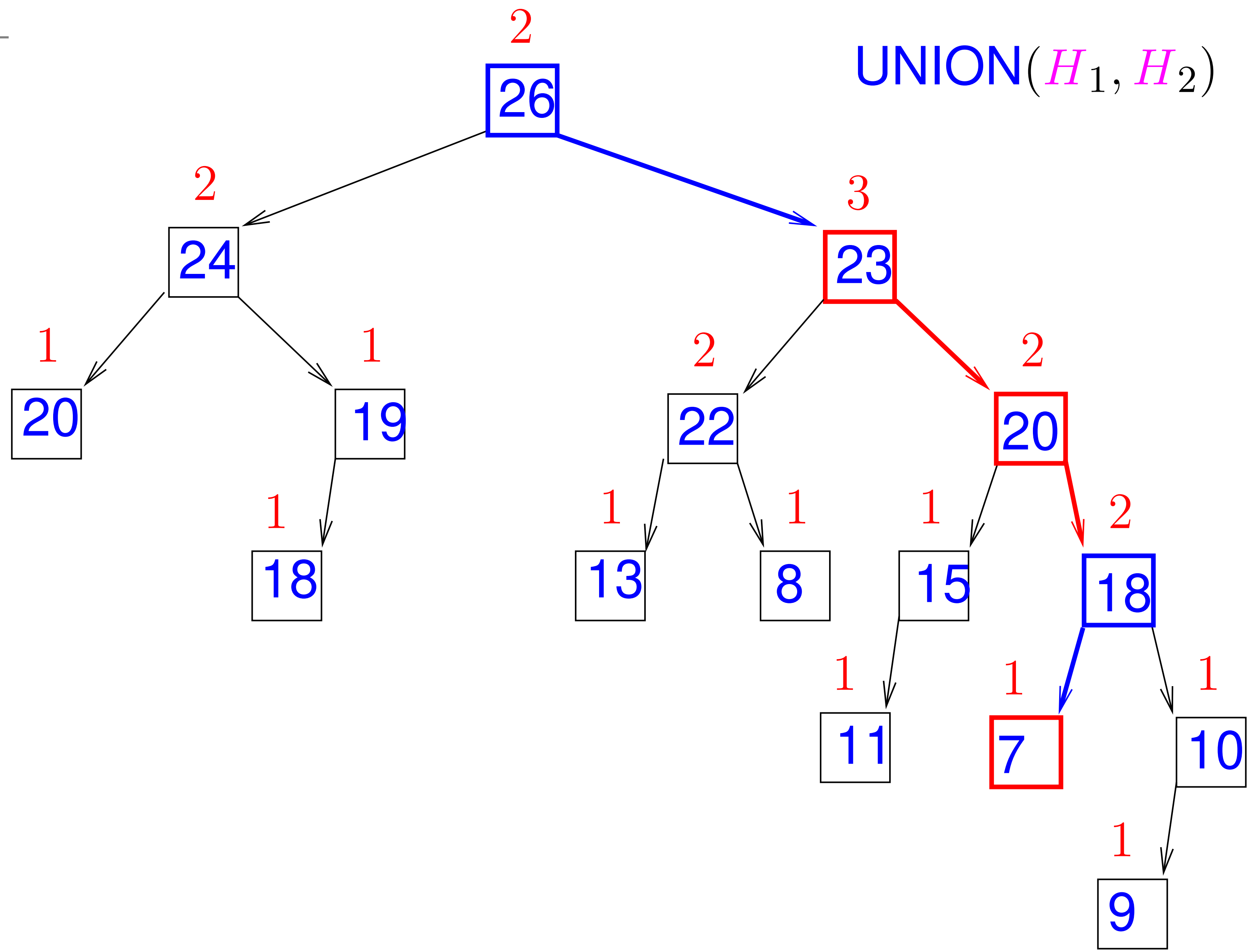


Rotina básica de manipulação

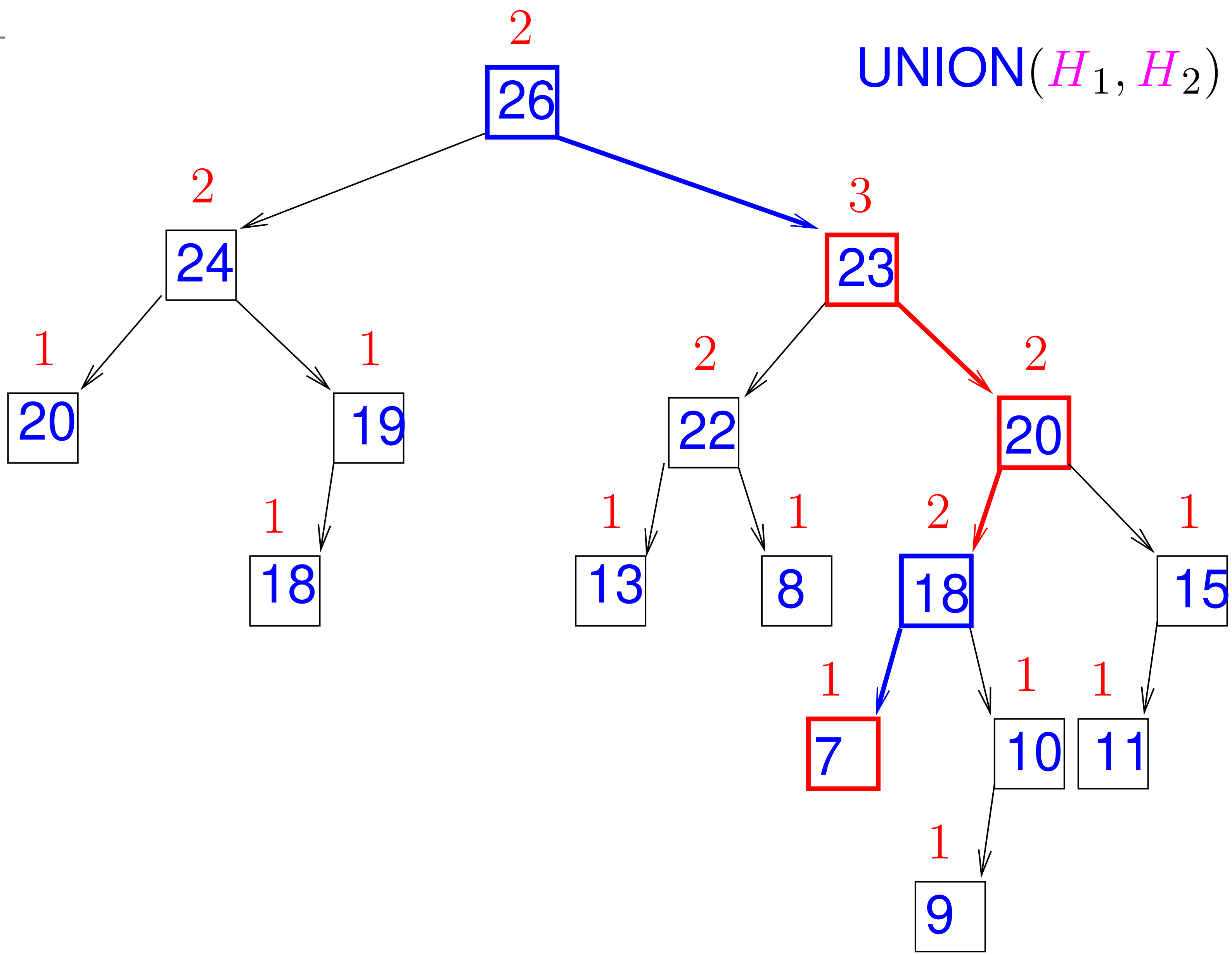
UNION(H_1, H_2)



Rotina básica de manipulação

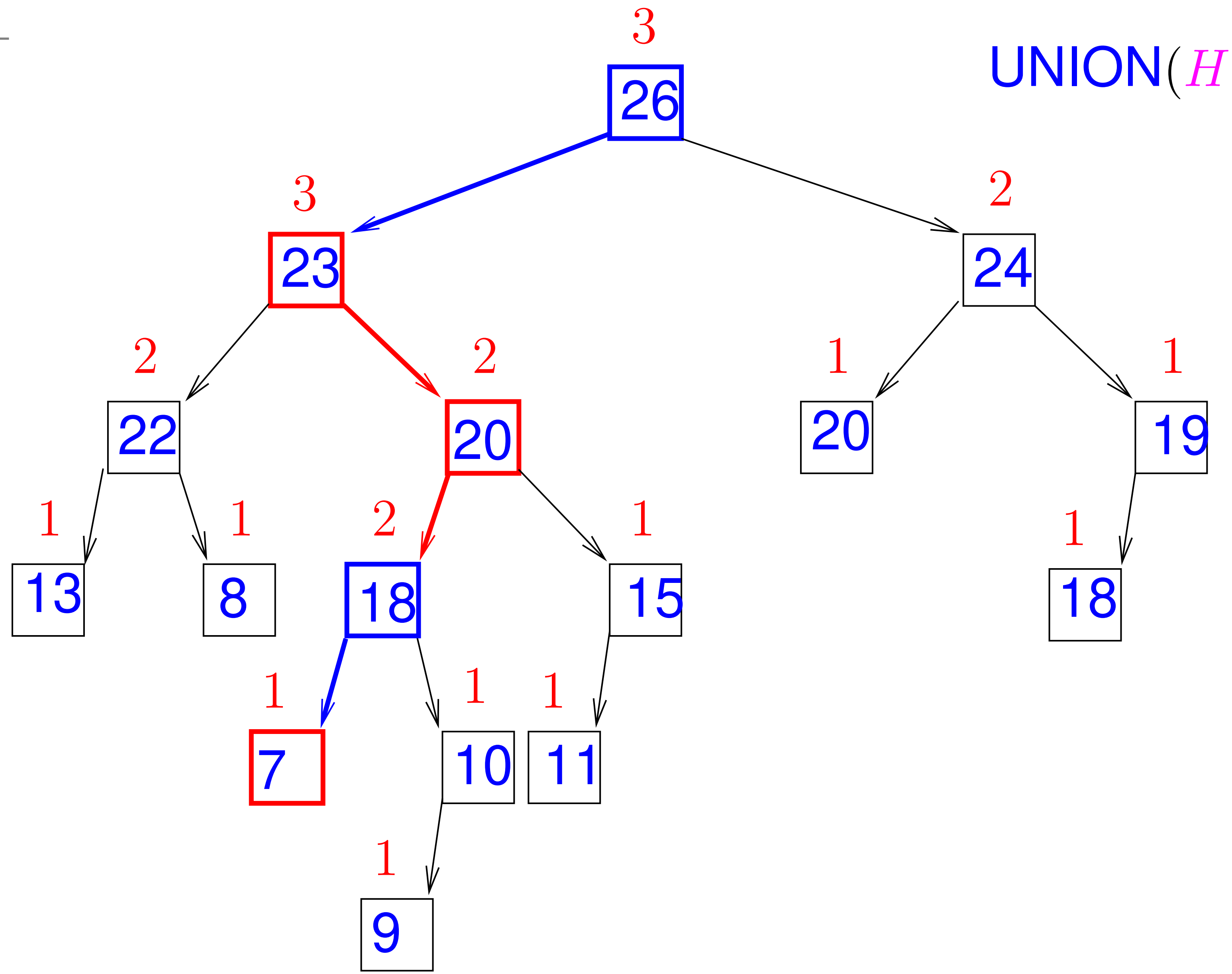


Rotina básica de manipulação



Rotina básica de manipulação

UNION(H_1, H_2)



Rotina básica de manipulação

LEFTIST-HEAP-UNION (H_1, H_2)

se $raiz[H_1] = \text{NIL}$ **então devolva** H_2
se $raiz[H_2] = \text{NIL}$ **então devolva** H_1
se $prior[raiz[H_1]] < prior[raiz[H_2]]$ **então** $H_1 \leftrightarrow H_2$
$x_1 \leftarrow raiz[H_1] \quad x_2 \leftarrow raiz[H_2]$
se $esq[x_1] = \text{NIL}$ **então** $esq[x_1] \leftarrow x_2$
senão $H' \leftarrow$ MAKE-LEFTIST-HEAP$()$
    $raiz[H'] \leftarrow dir[x_1]$
    $H' \leftarrow$ LEFTIST-HEAP-UNION(H', H_2)
    $dir[x_1] \leftarrow raiz[H']$
    se $dist[esq[x_1]] < dist[dir[x_1]]$
        então $esq[x_1] \leftrightarrow dir[x_2]$
    $dist[x_1] \leftarrow dist[dir[x_1]] + 1$
devolva H_1

Consumo de tempo

O consumo de tempo do algoritmo LEFTIST-HEAP-UNION no pior caso é proporcional a

$$dcomp(raiz[H_1]) + dcomp(raiz[H_2]) = \mathrm{O}(\lg m)$$

onde $m = tam(raiz[H_1]) + tam(raiz[H_2])$

O consumo de tempo do algoritmo LEFTIST-HEAP-UNION é $\mathrm{O}(\lg m)$.

Fila com heap esquerdista

Rotina que insere um nó x em um heap esquerdista H.

A rotina supões que $prior[x]$ já foi definido.

LEFTIST-HEAP-INSERT (H, x)

$H' \leftarrow$ MAKE-LEFTIST-HEAP$()$

$esq[x] \leftarrow \mathrm{NIL}$

$dir[x] \leftarrow \mathrm{NIL}$

$dist[x] \leftarrow 1$

$raiz[H'] \leftarrow x$

$H \leftarrow$ LEFTIST-HEAP-UNION(H, H')

Consome tempo $\mathrm{O}(\lg m)$.

Fila com heap esquerdista

Rotina que remove e devolve o nó de maior prioridade de um heap esquerdista H.

LEFTIST-HEAP-EXTRACT (H)

$x \leftarrow raiz[H]$
$H' \leftarrow$ MAKE-LEFTIST-HEAP$()$
$raiz[H'] \leftarrow esq[x]$
$raiz[H] \leftarrow dir[x]$
$H \leftarrow$ LEFTIST-HEAP-UNION(H, H')
devolva x

Consome tempo $\mathrm{O}(\lg m)$.

Quicksort

CLRS 7

Partição

Problema: Rearranjar um dado vetor $A[p \mathinner{.\,.} r]$ e devolver um índice q, $p \leq q \leq r$, tais que

$$A[p \mathinner{.\,.} q-1] \leq A[q] < A[q+1 \mathinner{.\,.} r]$$

Entra:

$$\begin{array}{cc} & \begin{array}{cccccccccc} p & & & & & & & & & r \end{array} \\ A & \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 99 & 33 & 55 & 77 & 11 & 22 & 88 & 66 & 33 & 44 \\ \hline \end{array} \end{array}$$

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:

$$\begin{array}{c|c|c|c|c|c|c|c|c|c|c|}
 & p & & & & & & & & & r \\
\cline{2-11}
A & 99 & 33 & 55 & 77 & 11 & 22 & 88 & 66 & 33 & 44 \\
\cline{2-11}
\end{array}$$

Sai:

$$\begin{array}{c|c|c|c|c|c|c|c|c|c|c|}
 & p & & & & q & & & & & r \\
\cline{2-11}
A & 33 & 11 & 22 & 33 & 44 & 55 & 99 & 66 & 77 & 88 \\
\cline{2-11}
\end{array}$$

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

		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	$= \Theta(2n+4) + \mathrm{O}(2n) \qquad = \Theta(n)$

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 $A[p \mathinner{.\,.} r]$ 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,

$$A[p \mathinner{.\,.} q-1] \le A[q] \le A[q+1 \mathinner{.\,.} r]$$

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

QUICKSORT (A, p, r)
se $p < r$
  então $q \leftarrow$ PARTICIONE (A, p, r)
    QUICKSORT $(A, p, q-1)$
    QUICKSORT $(A, q+1, r)$

	p				q					r
A	11	22	33	33	44	55	66	77	88	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 $(A, q + 1, r)$

No começo da linha 3,

$$A[p \,.\,.\, q{-}1] \leq A[q] \leq A[q{+}1 \,.\,.\, r]$$

Consumo de tempo?

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 $(A, q+1, r)$

No começo da linha 3,

$$A[p\,..\,q{-}1] \leq A[q] \leq A[q{+}1\,..\,r]$$

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 $n = r - p + 1$

$$T(0) = \Theta(1)$$

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

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

Recorrência

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

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = T(k) + T(n - k - 1) + \Theta(n) \quad \text{para } n = 3, 4, \ldots$$

Recorrência grosseira:

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

$T(n)$ é $\Theta(???)$.

Recorrência

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

$$T(0) = \Theta(1)$$
$$T(1) = \Theta(1)$$
$$T(n) = T(k) + T(n-k-1) + \Theta(n) \quad \text{para } n = 3, 4, \ldots$$

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 $n = r - p + 1$

$$T(0) = \Theta(1)$$

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

$$T(n) = \max_{0 \le k \le n-1} \{T(k) + T(n-k-1)\} + \Theta(n) \quad \text{para } n = 3, 4, \ldots$$

Exemplo concreto

$$S(0) = 1$$

$$S(1) = 1$$

$$S(n) = \max_{0 \leq k \leq n-1} \{S(k) + S(n-k-1)\} + n \quad \text{para } n = 3, 4, \ldots$$

n	0	1	2	3	4	5
$S(n)$	1	1	$2+2$	$5+3$	$9+4$	$14+5$

Exemplo concreto

$$S(0) = 1$$

$$S(1) = 1$$

$$S(n) = \max_{0 \le k \le n-1} \{S(k) + S(n-k-1)\} + n \quad \text{para } n = 3, 4, \ldots$$

n	0	1	2	3	4	5
$S(n)$	1	1	$2+2$	$5+3$	$9+4$	$14+5$

Vou mostrar que $S(n) \le n^2 + 1$ para $n \ge 0$.

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 $O(n^2)$.

O consumo de tempo do QUICKSORT é $O(n^2)$.

Quicksort no melhor caso

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

$$M(0) = \Theta(1)$$

$$M(1) = \Theta(1)$$

$$M(n) = \min_{0 \leq k \leq n-1} \{M(k) + M(n-k-1)\} + \Theta(n)$$ 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 $n \geq 1$,

$$M(n) \geq \frac{(n-1)}{2} \lg \frac{n-1}{2}.$$

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

$$S(n) = \max_{0 \le k \le n-1} \left(S(k) + S(n-k-1) \right) + n$$

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.)