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 $O(\lg m)$.

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 $O(\lg m)$.

MAX-HEAP-INSERT $(A, m, prior)$
$m \leftarrow m + 1$
$A[m] \leftarrow -\infty$
HEAP-INCREASE-KEY $(A, m, prior)$

Consome tempo $O(\lg m)$.

Outras implementações

Operação	max-heap (pior caso)	heap binomial (pior caso)	fibonacci heap (amortizado)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg m)$	$O(\lg m)$	$\Theta(1)$
MAXIMUM	$\Theta(1)$	$O(\lg m)$	$\Theta(1)$
EXTRACT-MAX	$\Theta(\lg m)$	$\Theta(\lg m)$	$O(\lg m)$
UNION	$\Theta(m)$	$O(\lg m)$	$\Theta(1)$
INCREASE-KEY	$\Theta(\lg m)$	$\Theta(\lg m)$	$\Theta(1)$
DELETE	$\Theta(\lg m)$	$\Theta(\lg m)$	$O(\lg m)$

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   $i \leftarrow m$
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, $A[1 \,.\,.\, m]$ 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

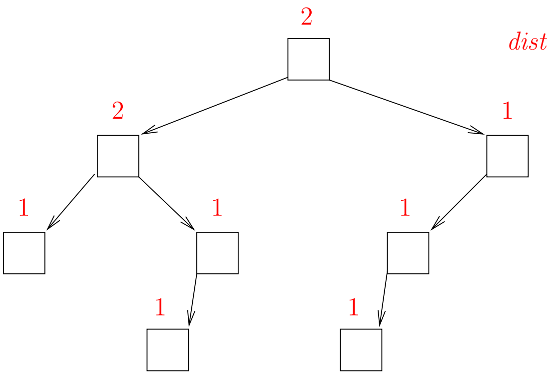
Cada nó x tem quatro campos:

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



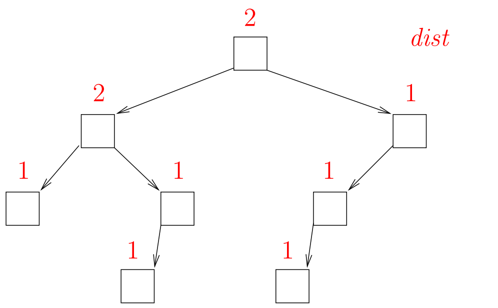
Árvores esquerdistas

Uma árvore é esquerdista se

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

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

Exemplo 1:



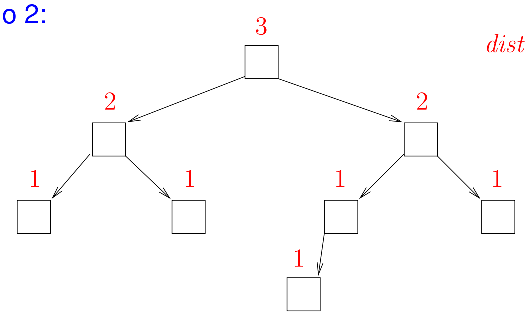
Árvores esquerdistas

Uma árvore é esquerdista se

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

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

Exemplo 2:



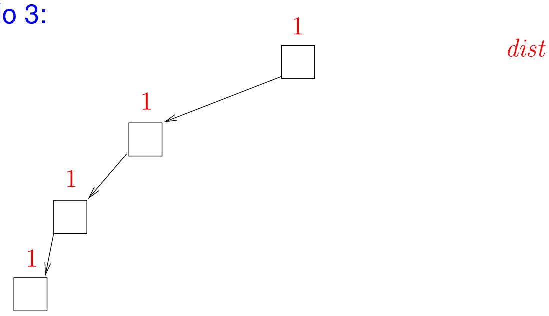
Árvores esquerdistas

Uma árvore é esquerdista se

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

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

Exemplo 3:



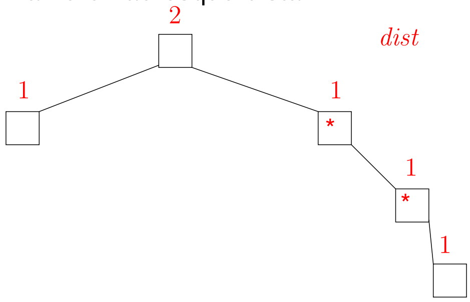
Árvores esquerdistas

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

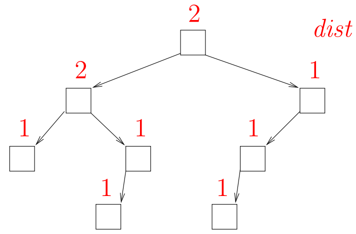
$$\langle x, dir[x], dir[dir[x]], \ldots, \text{NIL}\rangle.$$

$dcomp[x]$:= 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,

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

Conseqüência

Se x é um nó de uma árvore esquerdista, então

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

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

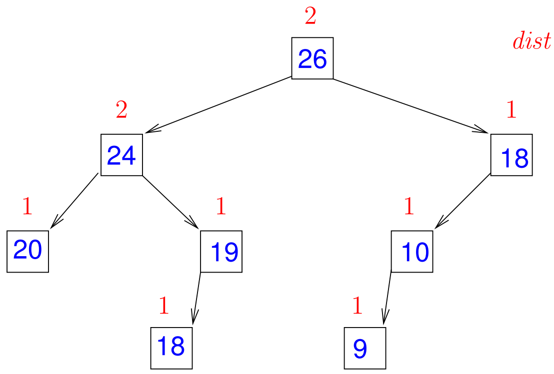
$pai[x]$:= pai do nó x

Um heap esquerdista H é uma árvore esquerdista que satisfaz

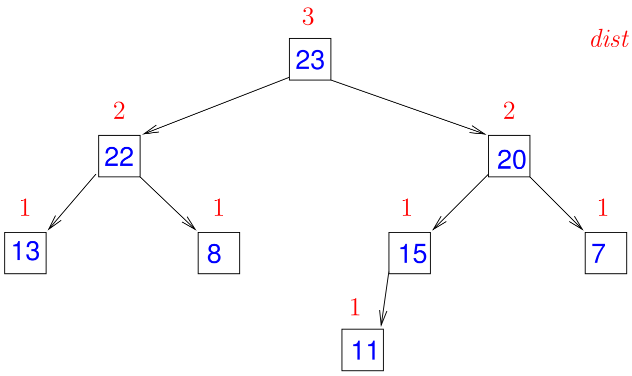
$$prior[pai[x]] \geq prior[x]$$

para todo nó $x \neq raiz[H]$.

Heap esquerdista

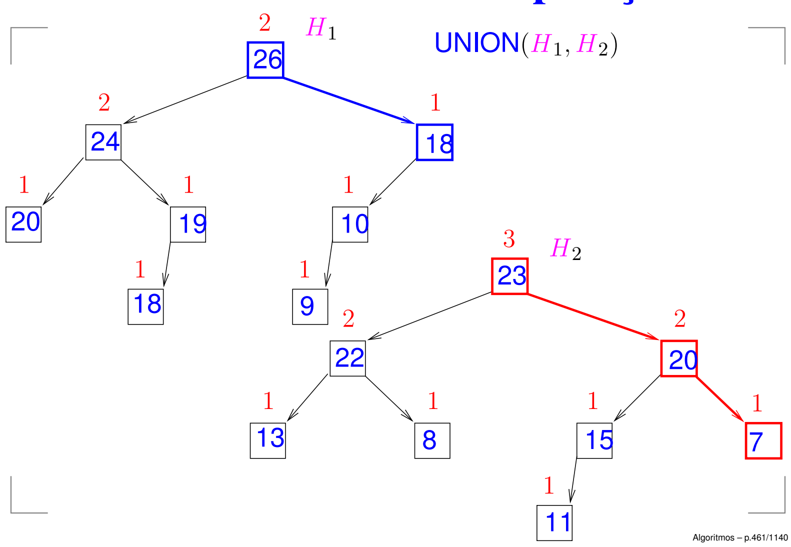


Heap esquerdista



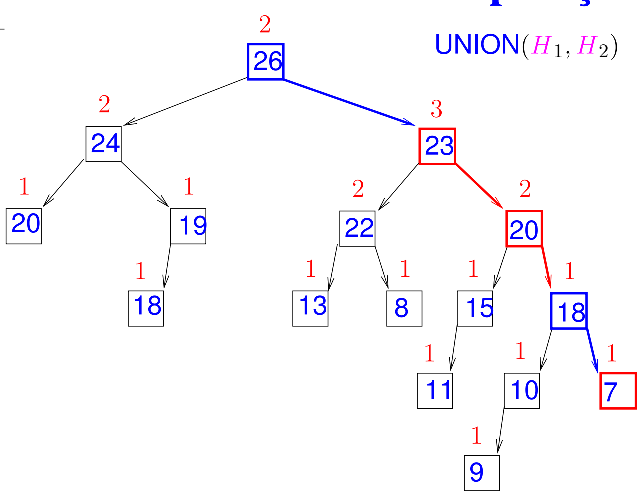
Rotina básica de manipulação

UNION(H_1, H_2)



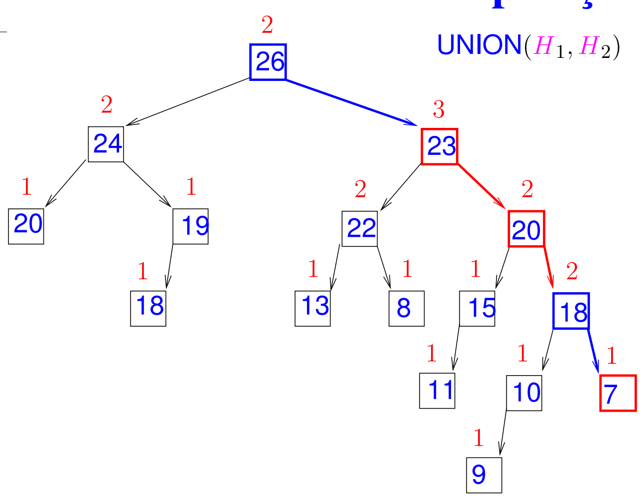
Rotina básica de manipulação

UNION(H_1, H_2)

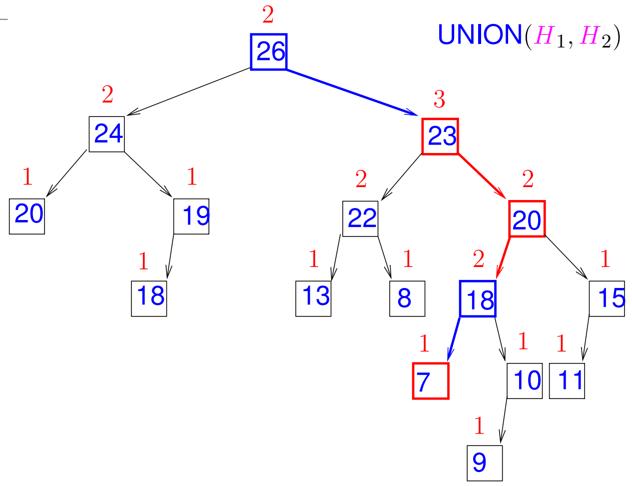


Rotina básica de manipulação

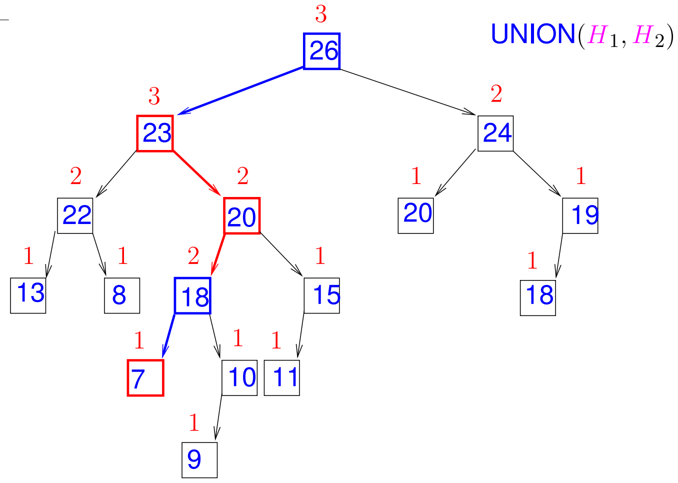
UNION(H_1, H_2)



Rotina básica de manipulação



Rotina básica de manipulação



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 supõe que $prior[x]$ já foi definido.

LEFTIST-HEAP-INSERT (H, x)

$H' \leftarrow$ MAKE-LEFTIST-HEAP()
$esq[x] \leftarrow \text{NIL}$
$dir[x] \leftarrow \text{NIL}$
$dist[x] \leftarrow 1$
$raiz[H'] \leftarrow x$
$H \leftarrow$ 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 $\mathrm{O}(\lg m)$.