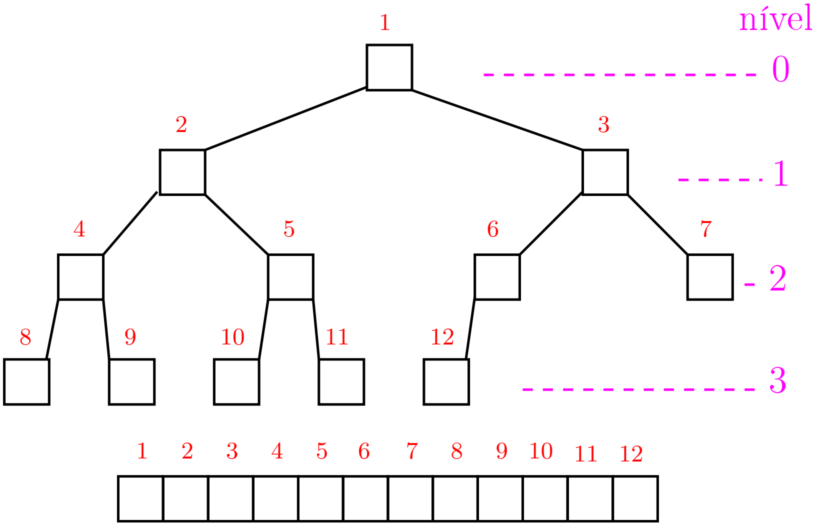
AULA 25

Representação de árvores em vetores e heaps

PF 10

http://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html

Representação de árvores em vetores



Pais e filhos

$\mathtt{v}[1\,..\,\mathtt{m}]$ é um vetor representando uma árvore.

Diremos que para qualquer **índice** ou **nó** $\mathtt{i}$,

- $\lfloor \mathtt{i}/2 \rfloor$ é o **pai** de $\mathtt{i}$;
- $2\,\mathtt{i}$ é o **filho esquerdo** de $\mathtt{i}$;
- $2\,\mathtt{i}+1$ é o **filho direito**.

Um nó $\mathtt{i}$ só tem filho esquerdo se $2\,\mathtt{i} \leq \mathtt{m}$.

Um nó $\mathtt{i}$ só tem filho direito se $2\,\mathtt{i}+1 \leq \mathtt{m}$.

Raiz e folhas

O nó 1 não tem pai e é chamado de **raiz**.

Um nó `i` é um **folha** se não tem **filhos**, ou seja $2\,\mathtt{i} > \mathtt{m}$.

Todo nó `i` é raiz da subárvore formada por

$$\mathtt{v}[\mathtt{i}, 2\mathtt{i}, 2\mathtt{i}+1, 4\mathtt{i}, 4\mathtt{i}+1, 4\mathtt{i}+2, 4\mathtt{i}+3, 8\mathtt{i}, \ldots, 8\mathtt{i}+7, \ldots]$$

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}} + 1, 2^{\mathtt{p}} + 2, \ldots, 2^{\mathtt{p}+1} - 1.$$

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ 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 $\lfloor \lg i \rfloor$.

Níveis

Cada nível $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ pertence ao nível $\lfloor \lg \mathtt{i} \rfloor$.

Prova: Se $\mathtt{p}$ é o nível do nó $\mathtt{i}$, então

$$\begin{array}{rcccll} 2^{\mathtt{p}} & \leq & \mathtt{i} & < & 2^{\mathtt{p}+1} & \Rightarrow \\ \lg 2^{\mathtt{p}} & \leq & \lg \mathtt{i} & < & \lg 2^{\mathtt{p}+1} & \Rightarrow \\ \mathtt{p} & \leq & \lg \mathtt{i} & < & \mathtt{p}+1 & \end{array}$$

Logo, $\mathtt{p} = \lfloor \lg \mathtt{i} \rfloor$.

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 $\mathtt{p}$, exceto talvez o último, tem exatamente $2^{\mathtt{p}}$ nós e esses são

$$2^{\mathtt{p}}, 2^{\mathtt{p}}+1, 2^{\mathtt{p}}+2, \ldots, 2^{\mathtt{p}+1}-1.$$

O nó $\mathtt{i}$ pertence ao nível $\lfloor \lg \mathtt{i} \rfloor$.

Prova: Se $\mathtt{p}$ é o nível do nó $\mathtt{i}$, então

$$\begin{array}{rcccll} 2^{\mathtt{p}} & \leq & \mathtt{i} & < & 2^{\mathtt{p}+1} & \Rightarrow \\ \lg 2^{\mathtt{p}} & \leq & \lg \mathtt{i} & < & \lg 2^{\mathtt{p}+1} & \Rightarrow \\ \mathtt{p} & \leq & \lg \mathtt{i} & < & \mathtt{p}+1 & \end{array}$$

Logo, $\mathtt{p} = \lfloor \lg \mathtt{i} \rfloor$.

Portanto, o número total de níveis é $1 + \lfloor \lg \mathtt{m} \rfloor$.

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

$$\langle \texttt{filho}(\texttt{i}), \texttt{filho}(\texttt{filho}(\texttt{i})), \texttt{filho}(\texttt{filho}(\texttt{filho}(\texttt{i}))), \ldots$$

onde $\texttt{filho}(\texttt{i})$ vale $2\texttt{i}$ ou $2\texttt{i} + 1$.

Os nós que têm **altura zero** são as folhas.

Altura

A **altura** de um nó $\mathtt{i}$ é o maior comprimento de um caminho de $\mathtt{i}$ a uma folha.

Em outras palavras, a altura de um nó $\mathtt{i}$ é o maior comprimento de uma seqüência da forma

$$\langle \mathtt{filho}(\mathtt{i}), \mathtt{filho}(\mathtt{filho}(\mathtt{i})), \mathtt{filho}(\mathtt{filho}(\mathtt{filho}(\mathtt{i}))), \ldots$$

onde $\mathtt{filho}(\mathtt{i})$ vale $2\mathtt{i}$ ou $2\mathtt{i} + 1$.

Os nós que têm **altura zero** são as folhas.

A altura de um nó $\mathtt{i}$ é $\lfloor \lg(\mathtt{m}/\mathtt{i}) \rfloor$ (MAC0338).

Resumão

filho esquerdo de $\texttt{i}$:	$2\,\texttt{i}$
filho direito de $\texttt{i}$:	$2\,\texttt{i}+1$
pai de $\texttt{i}$:	$\lfloor \texttt{i}/2 \rfloor$
nível da raiz:	0
nível de $\texttt{i}$:	$\lfloor \lg \texttt{i} \rfloor$
altura da raiz:	$\lfloor \lg \texttt{m} \rfloor$
altura da árvore:	$\lfloor \lg \texttt{m} \rfloor$
altura de $\texttt{i}$:	$\lfloor \lg(\texttt{m}/\texttt{i}) \rfloor$ (MAC0338)
altura de uma folha:	0
total de nós de altura $\texttt{h}$	$\leq \lceil \texttt{m}/2^{\texttt{h}+1} \rceil$ (MAC0338)

Heaps

Um vetor $v[1..m]$ é um max-heap se

$$v[i/2] \geq v[i]$$

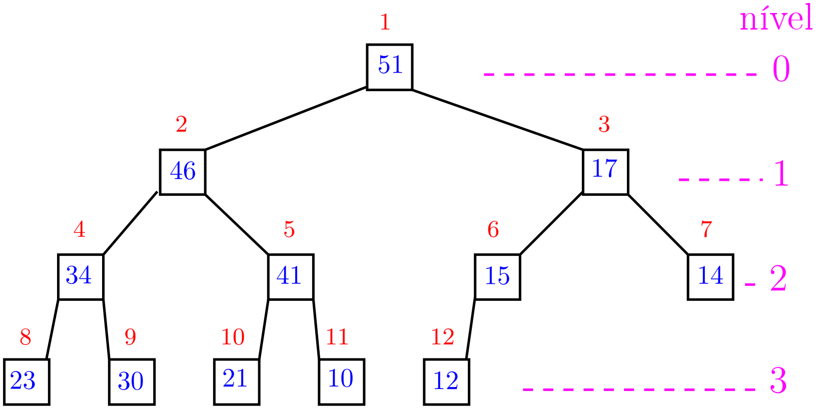
para todo $i = 2, 3, \ldots, m$.

De uma forma mais geral, $v[j..m]$ é um max-heap se

$$v[i/2] \geq v[i]$$

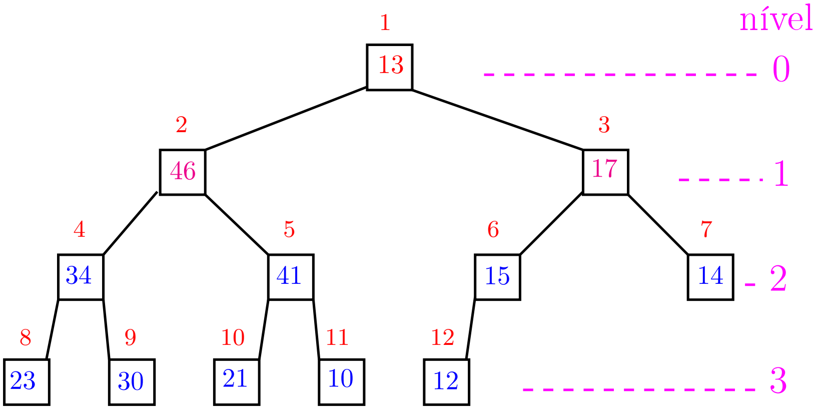
para todo
$i = 2j, 2j + 1, 4j, \ldots, 4j + 3, 8j, \ldots, 8j + 7, \ldots$.
Neste caso também diremos que a subárvore com raiz j é um max-heap.

max-heap



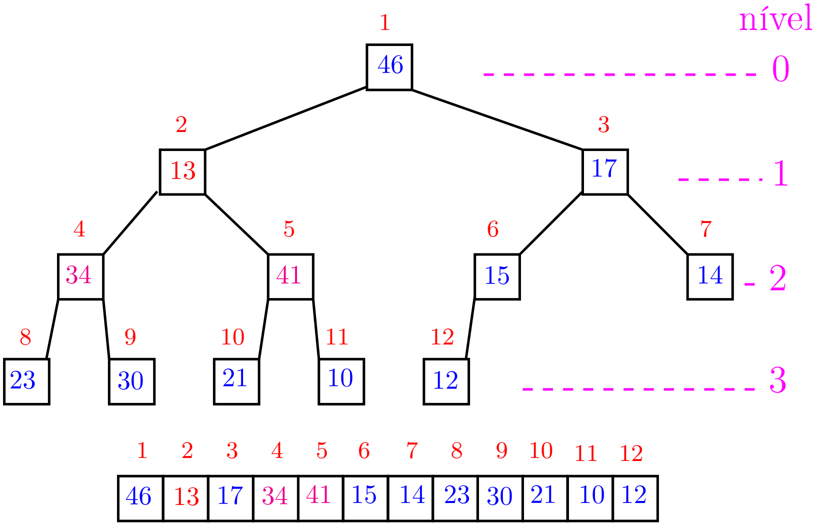
1	2	3	4	5	6	7	8	9	10	11	12
51	46	17	34	41	15	14	23	30	21	10	12

Função básica de manipulação de max-heap

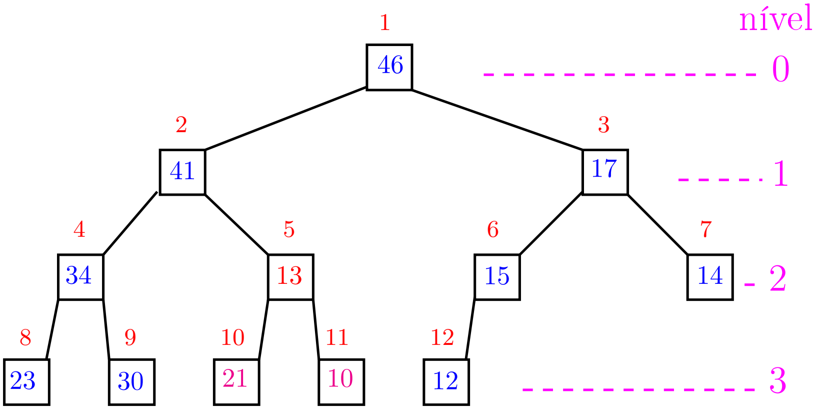


1	2	3	4	5	6	7	8	9	10	11	12
13	46	17	34	41	15	14	23	30	21	10	12

Função básica de manipulação de max-heap

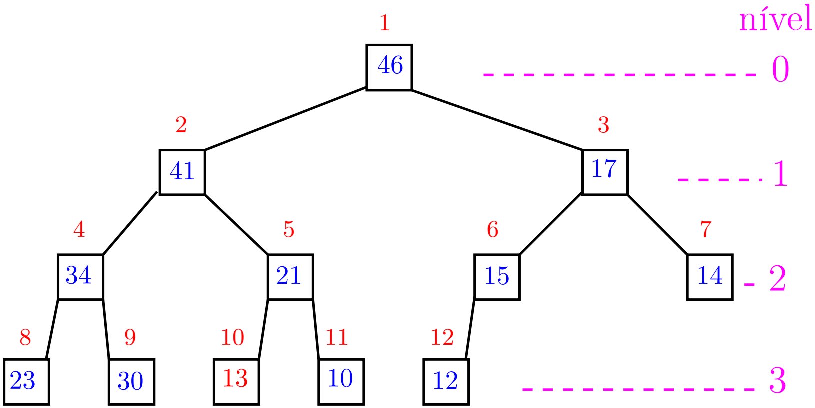


Função básica de manipulação de max-heap



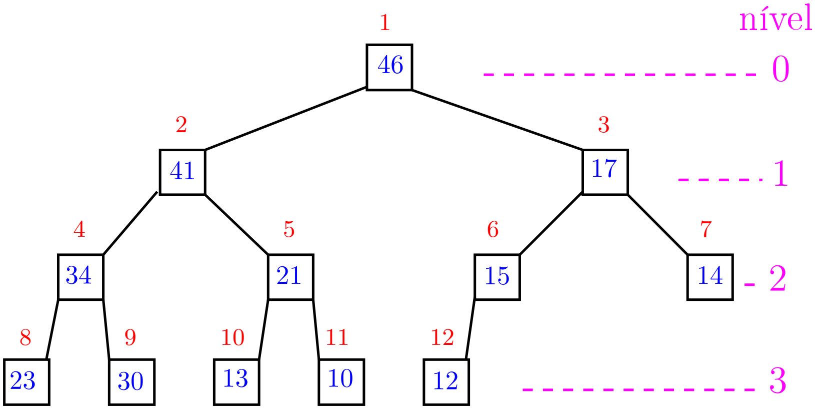
1	2	3	4	5	6	7	8	9	10	11	12
46	41	17	34	13	15	14	23	30	21	10	12

Função 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

Função 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

Função `peneira`

O coração de qualquer algoritmo que manipule um `max-heap` é uma função que recebe um vetor arbitrário `v`[1 . . `m`] e um índice `i` e faz `v`[`i`] "descer" para sua posição correta.

Função peneira

Rearranja o vetor v[1 . . m] de modo que o "subvetor" cuja raiz é i seja um max-heap.

```
void peneira (int i, int m, int v[]) {
    int f = 2*i, x;
    while (f <= m) {
        if (f < m && v[f] < v[f+1]) f++;
        if (v[i] >= v[f]) break;
        x = v[i]; v[i] = v[f]; v[f] = x;
        i = f; f = 2*i;
    }
}
```

Função peneira

Supõe que os "subvetores"cujas raízes são filhos de i já são max-heap.

```
void peneira (int i, int m, int v[]) {
    int f = 2*i, x;
    while (f <= m) {
        if (f < m && v[f] < v[f+1]) f++;
        if (v[i] >= v[f]) break;
        x = v[i]; v[i] = v[f]; v[f] = x;
        i = f; f = 2*i;
    }
}
```

Função peneira

A seguinte implementação é um pouco melhor pois em vez de trocas faz apenas deslocamentos (linha 5).

```
void peneira (int i, int m, int v[]) {
1   int f = 2*i, x = v[i];
2   while (f <= m) {
3       if (f < m && v[f] < v[f+1]) f++;
4       if (x >= v[f]) break;
5       v[i] = v[f];
6       i = f; f = 2*i;
    }
7   v[i] = x;
}
```

Consumo de tempo

linha	todas as execuções da linha	
1	$=$	1
2	$\leq$	$1 + \lg \mathtt{m}$
3	$\leq$	$\lg \mathtt{m}$
4	$\leq$	$\lg \mathtt{m}$
5	$\leq$	$\lg \mathtt{m}$
6	$\leq$	$\lg \mathtt{m}$
7	$=$	1
total	$\leq$	$3 + 5 \lg \mathtt{m} = \mathrm{O}(\lg \mathtt{m})$

Conclusão

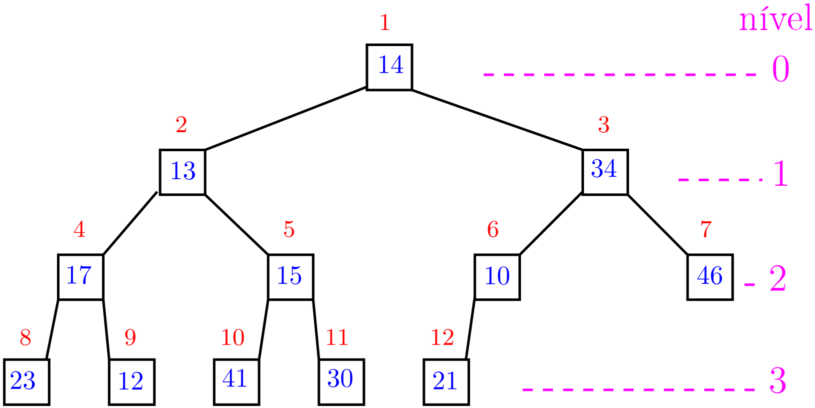
O consumo de tempo da função peneira é proporcional a $\lg \mathtt{m}$.

O consumo de tempo da função peneira é $O(\lg \mathtt{m})$.

Verdade seja dita ... (MAC0338)

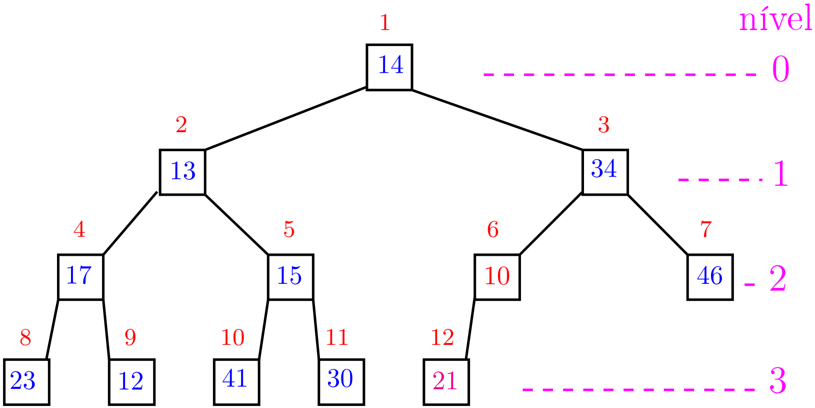
O consumo de tempo da função peneira é proporcional a $O(\lg \mathtt{m}/\mathtt{i})$.

Construção de um max-heap



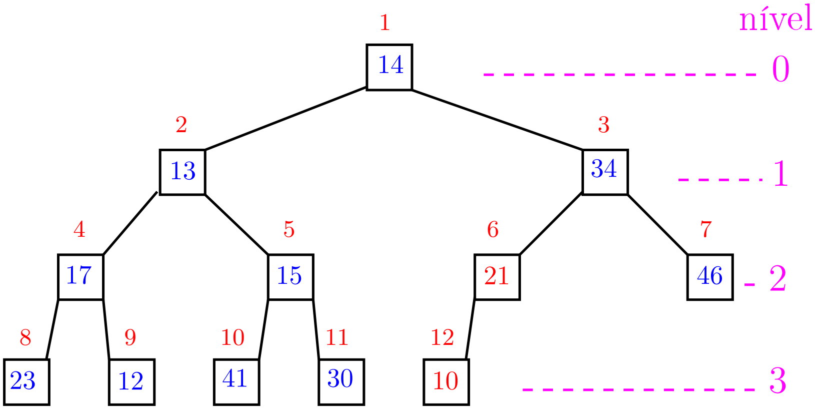
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	10	46	23	12	41	30	21

Construção de um `max-heap`



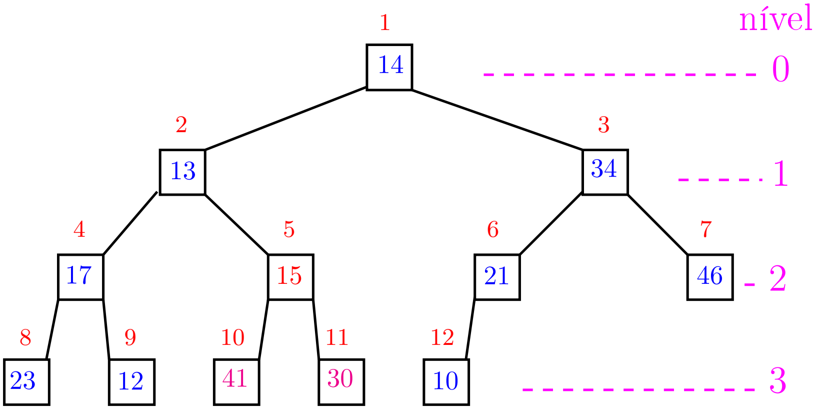
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	10	46	23	12	41	30	21

Construção de um max-heap



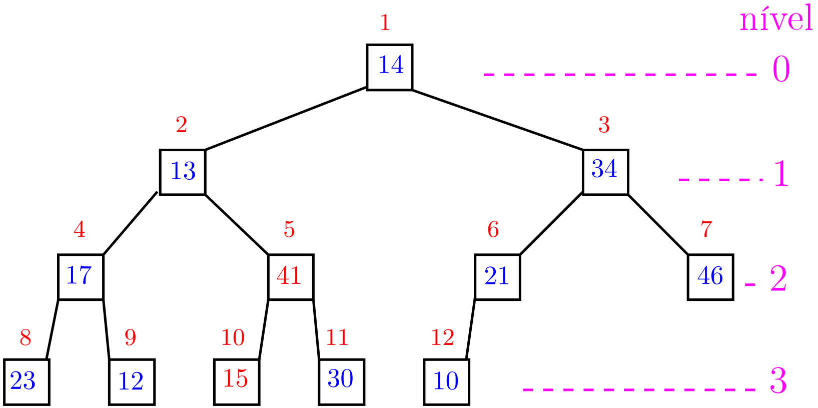
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	21	46	23	12	41	30	10

Construção de um max-heap



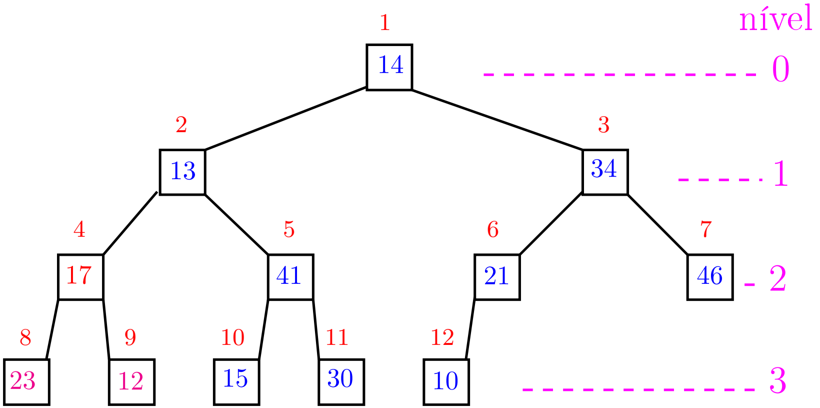
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	15	21	46	23	12	41	30	10

Construção de um max-heap



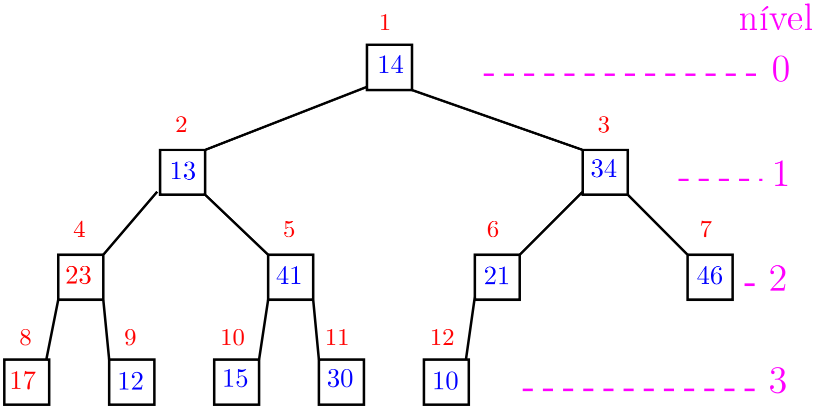
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	41	21	46	23	12	15	30	10

Construção de um max-heap



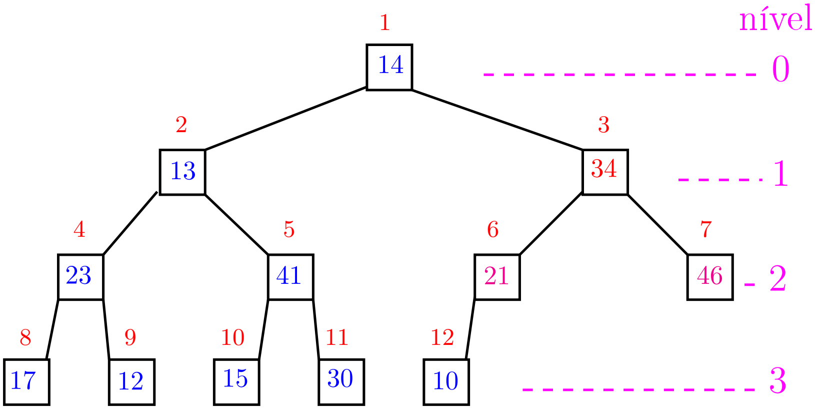
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	17	41	21	46	23	12	15	30	10

Construção de um max-heap



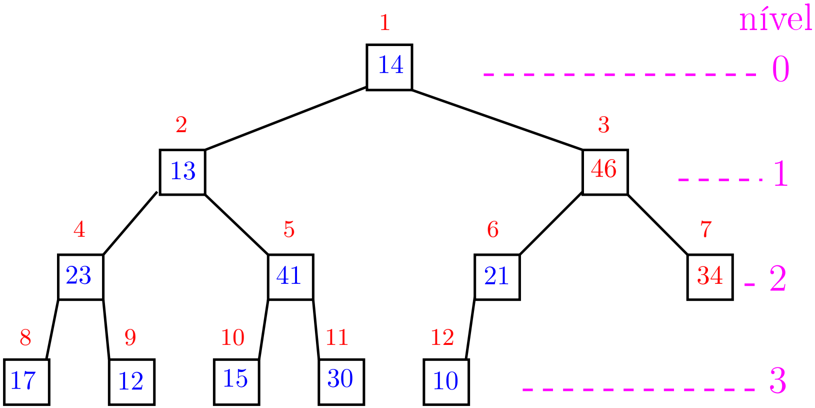
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	23	41	21	46	17	12	15	30	10

Construção de um max-heap



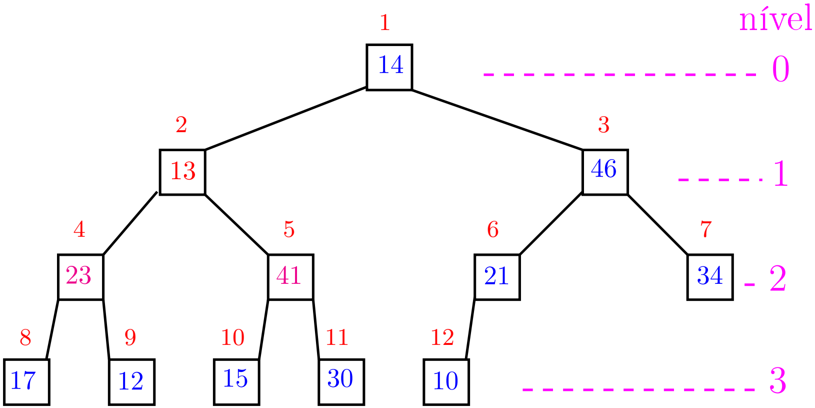
1	2	3	4	5	6	7	8	9	10	11	12
14	13	34	23	41	21	46	17	12	15	30	10

Construção de um max-heap



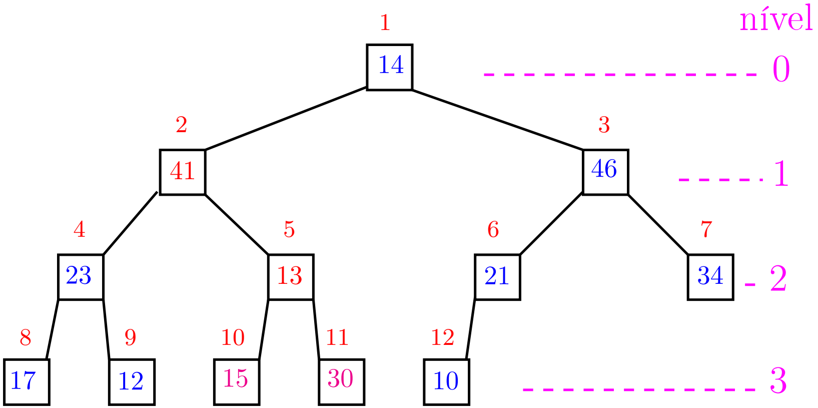
1	2	3	4	5	6	7	8	9	10	11	12
14	13	46	23	41	21	34	17	12	15	30	10

Construção de um max-heap



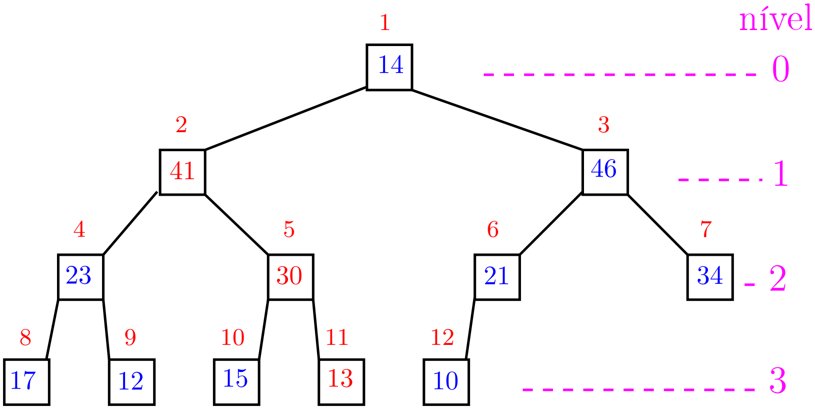
1	2	3	4	5	6	7	8	9	10	11	12
14	13	46	23	41	21	34	17	12	15	30	10

Construção de um max-heap



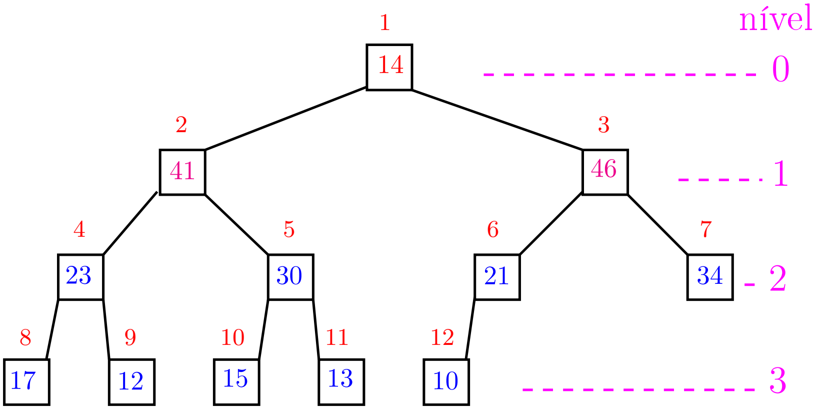
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	13	21	34	17	12	15	30	10

Construção de um max-heap



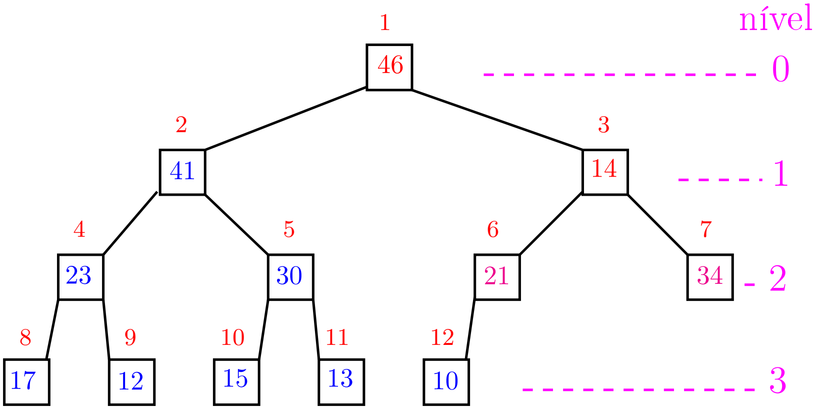
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	30	21	34	17	12	15	13	10

Construção de um max-heap



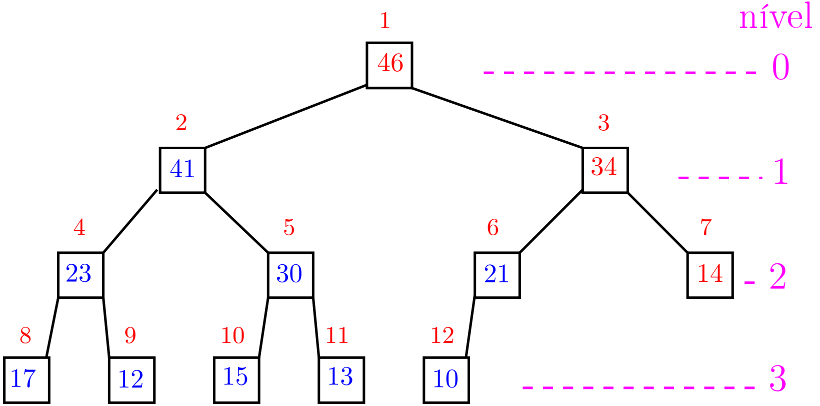
1	2	3	4	5	6	7	8	9	10	11	12
14	41	46	23	30	21	34	17	12	15	13	10

Construção de um max-heap



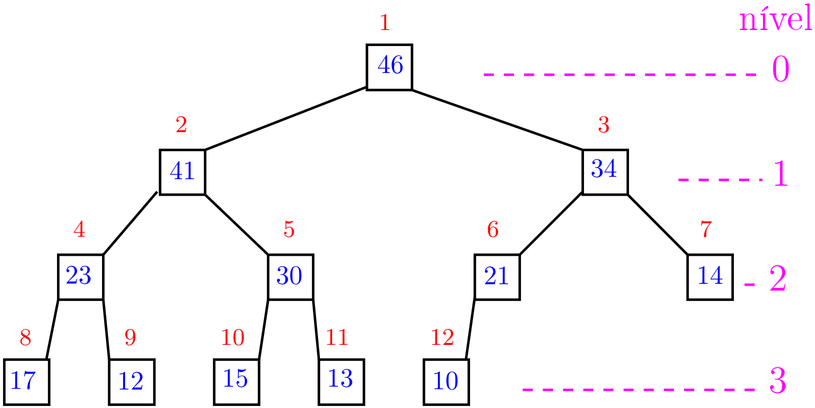
1	2	3	4	5	6	7	8	9	10	11	12
46	41	14	23	30	21	34	17	12	15	13	10

Construção de um max-heap



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

Construção de um `max-heap`



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

Construção de um max-heap

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

```
for (i = n/2; /*A*/ i >= 1; i--)
     peneira(i, n, v);
```

Relação invariante:

(i0) em /*A*/ vale que, $i+1, \ldots, n$ são raízes de max-heaps.

Consumo de tempo

Análise grosseira: consumo de tempo é

$$\frac{n}{2} \times \lg n = O(n \lg n).$$

Verdade seja dita . . . (MAC0338)

Análise mais cuidadosa: consumo de tempo é $O(n)$.

Conclusão

O consumo de tempo para construir um `max-heap` é $O(n \lg n)$.

Verdade seja dita . . . (MAC0338)

O consumo de tempo para construir um `max-heap` é $O(n)$.