Filas de Prioridades

Letícia Rodrigues Bueno

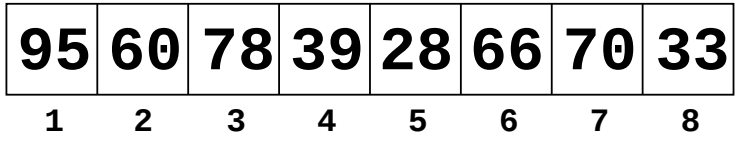
UFABC

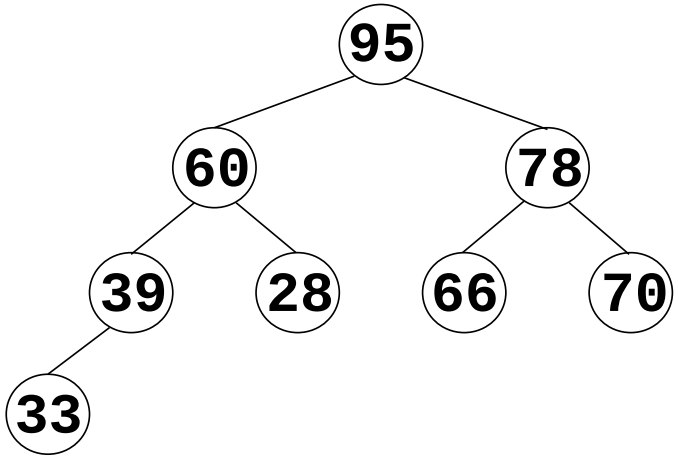
Heaps

- ***Heaps:*** lista linear com chaves $s_1, \ldots, s_n$ com propriedade $s_i \leq s_{\lfloor i/2 \rfloor}$, para $1 < i < n$;

Heaps

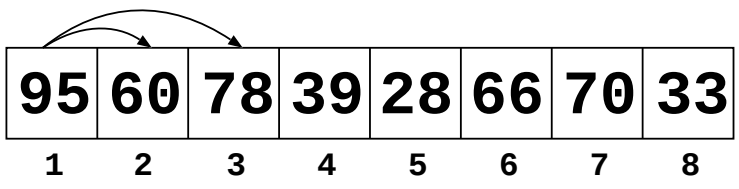
- ***Heaps:*** lista linear com chaves $s_1, \ldots, s_n$ com propriedade $s_i \leq s_{\lfloor i/2 \rfloor}$, para $1 < i < n$;

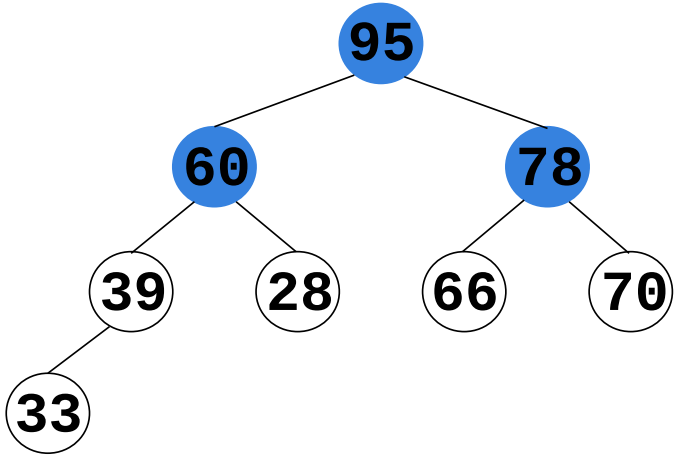




Heaps

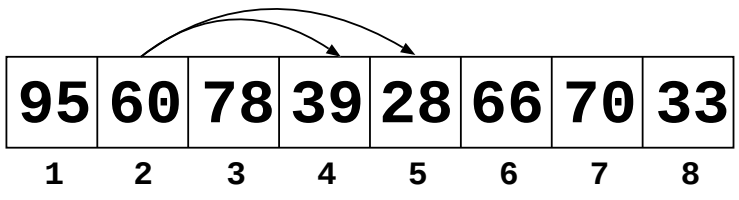
- ***Heaps:*** lista linear com chaves $s_1, \ldots, s_n$ com propriedade $s_i \leq s_{\lfloor i/2 \rfloor}$, para $1 < i < n$;

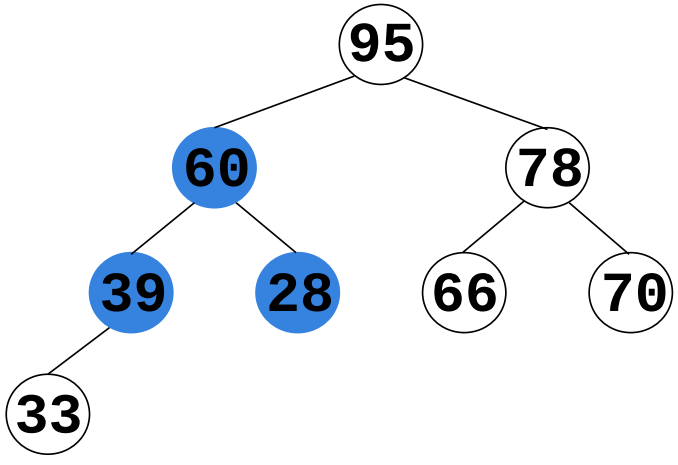




Heaps

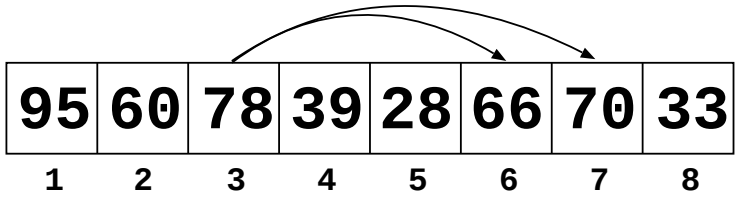
- ***Heaps:*** lista linear com chaves $s_1, \ldots, s_n$ com propriedade $s_i \leq s_{\lfloor i/2 \rfloor}$, para $1 < i < n$;

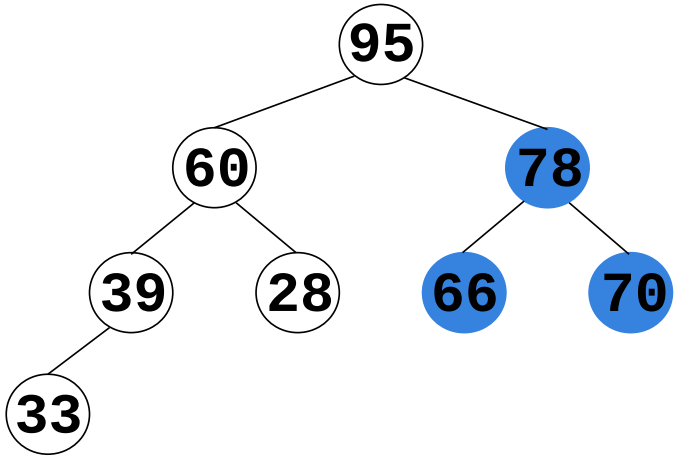




Heaps

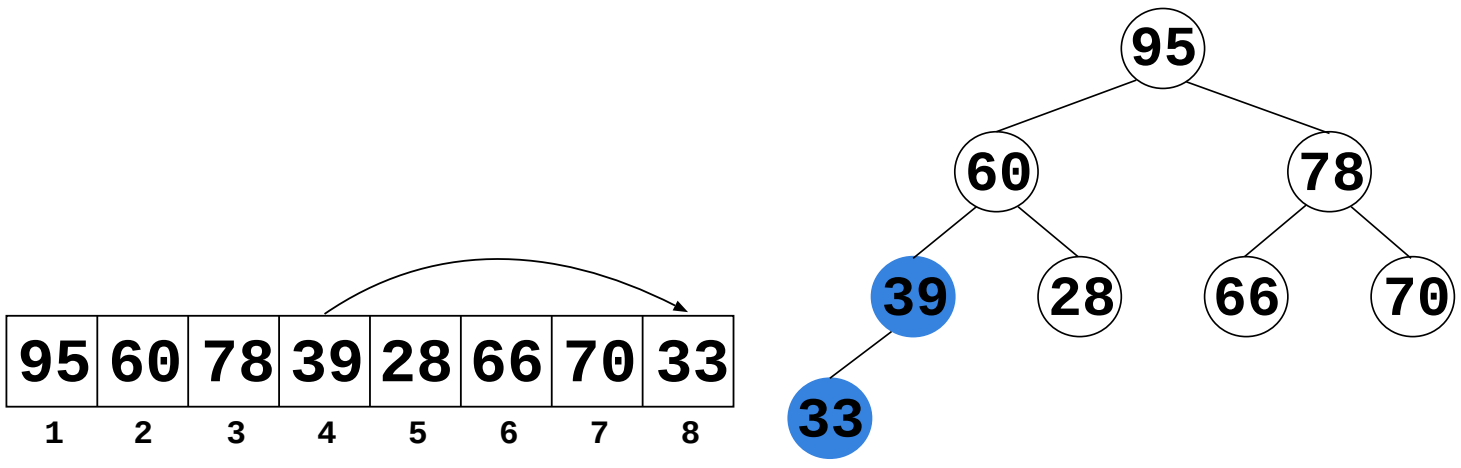
- ***Heaps:*** lista linear com chaves $s_1, \ldots, s_n$ com propriedade $s_i \leq s_{\lfloor i/2 \rfloor}$, para $1 < i < n$;





Heaps

- ***Heaps:*** lista linear com chaves $s_1, \ldots, s_n$ com propriedade $s_i \leq s_{\lfloor i/2 \rfloor}$, para $1 < i < n$;



Heaps - Aplicações

- Dois tipos: *heap* mínimo e *heap* máximo;

Heaps - Aplicações

- Dois tipos: *heap* mínimo e *heap* máximo;
- **Aplicações:** implementação de lista de prioridades;

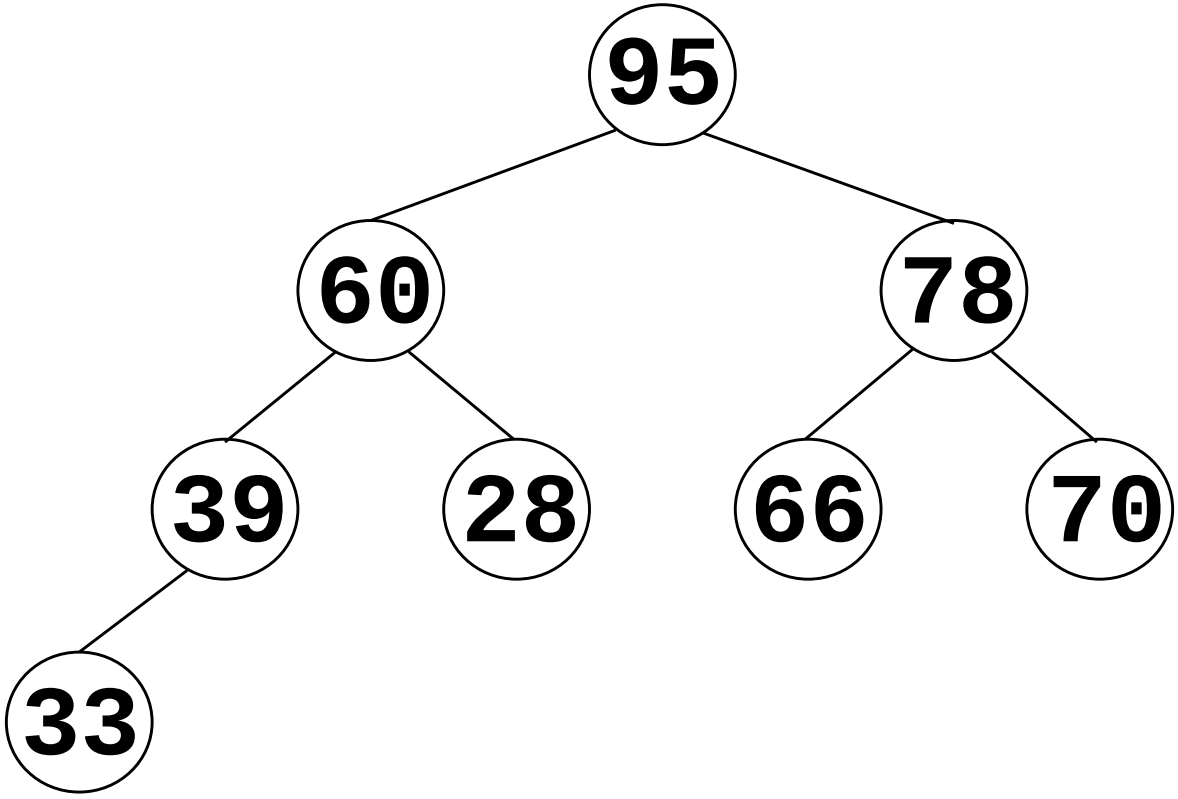
Heaps - Aplicações

- Dois tipos: *heap* mínimo e *heap* máximo;
- **Aplicações:** implementação de lista de prioridades;

Operação	**Lista não-ordenada**	**Lista Ordenada**	**Heap**
Seleção	$O(n)$	$O(1)$	$O(1)$
Inserção	$O(1)$	$O(n)$	$O(\log n)$
Remoção	$O(n)$	$O(1)$	$O(\log n)$
Alteração	$O(n)$	$O(n)$	$O(\log n)$
Construção	$O(n)$	$O(n \log n)$	$O(n)$

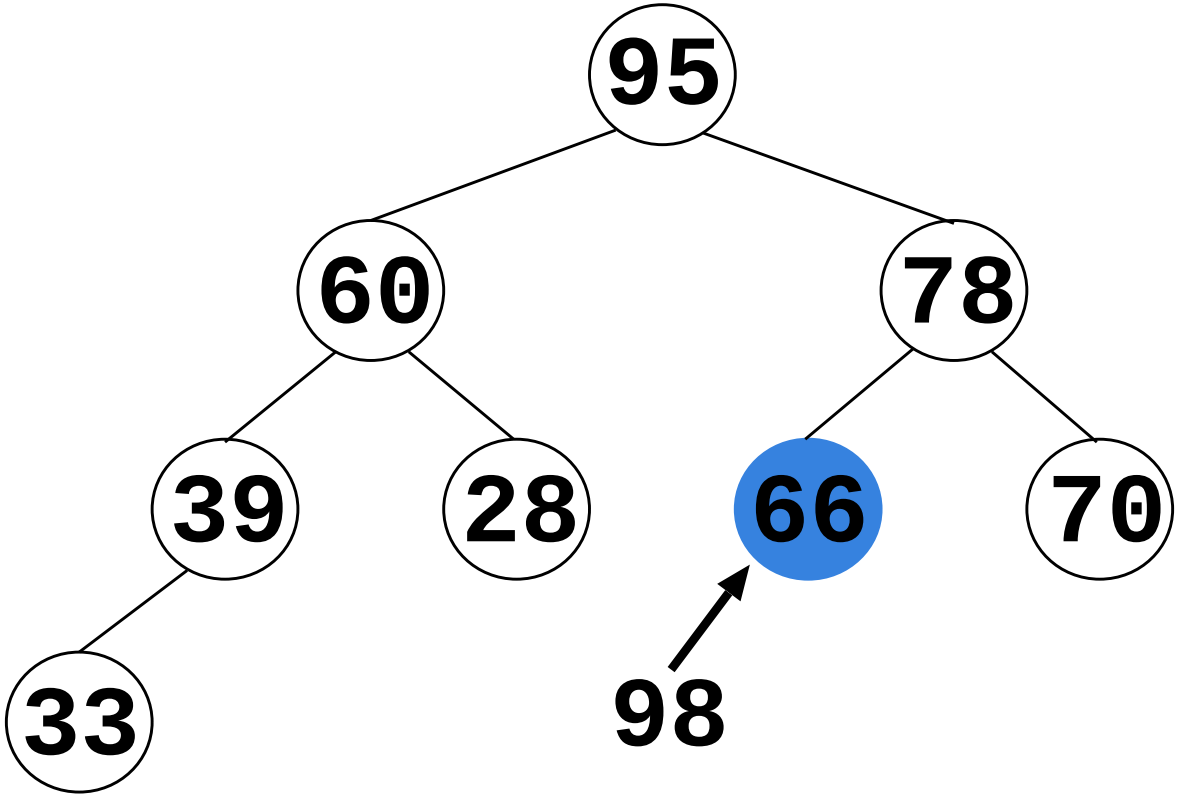
Métodos *Heap*: aumentando prioridade

Método subir:



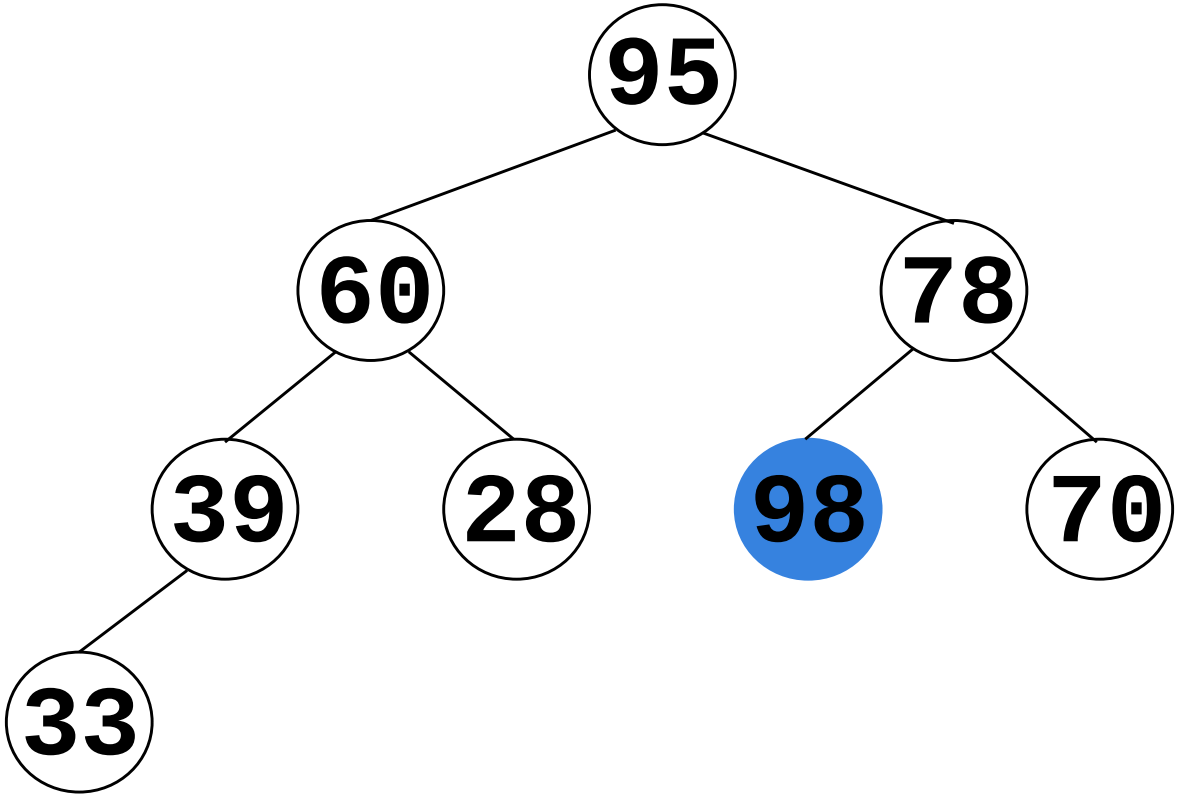
Métodos *Heap*: aumentando prioridade

Método subir:



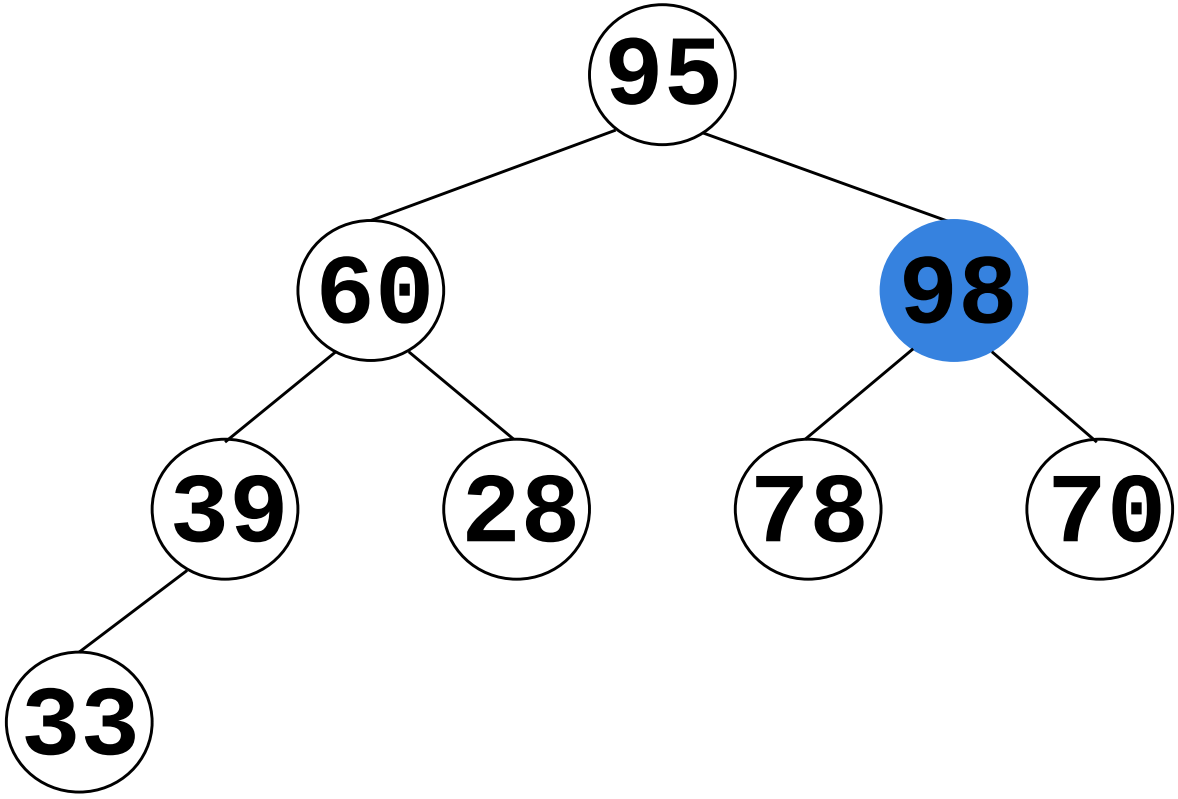
Métodos *Heap*: aumentando prioridade

Método subir:



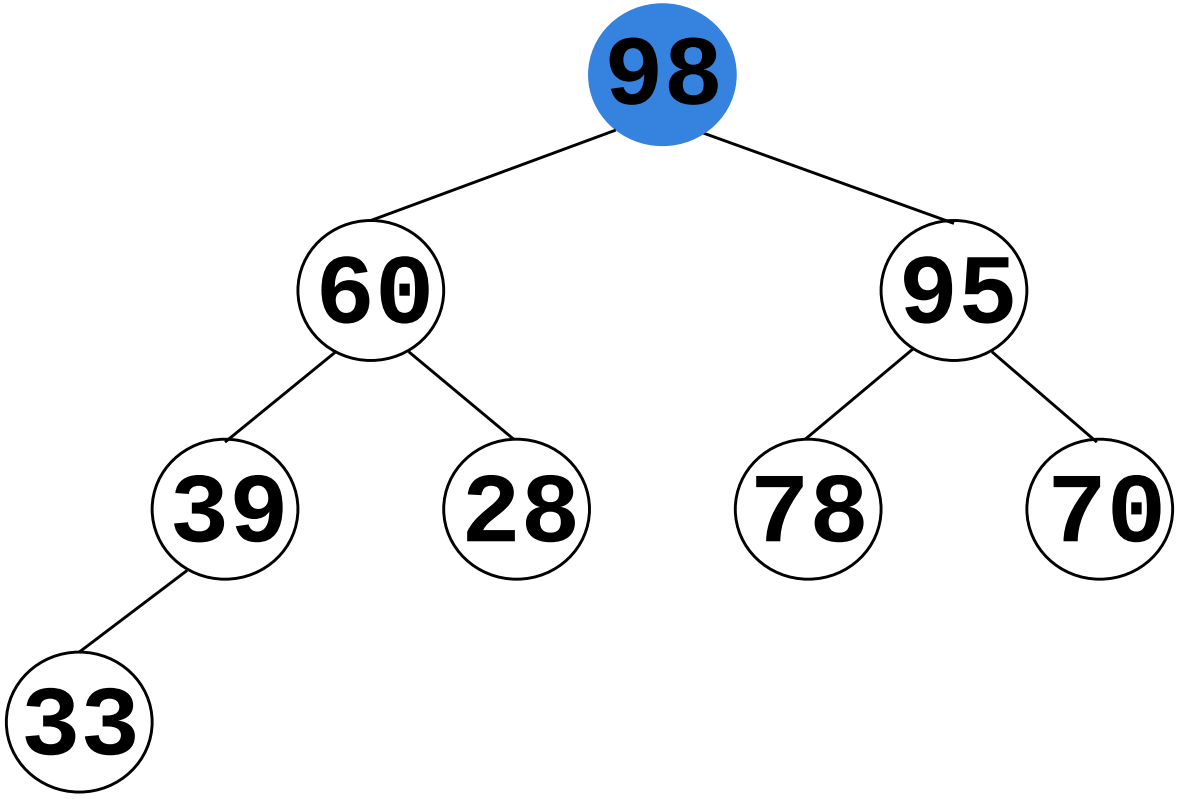
Métodos *Heap*: aumentando prioridade

Método subir:



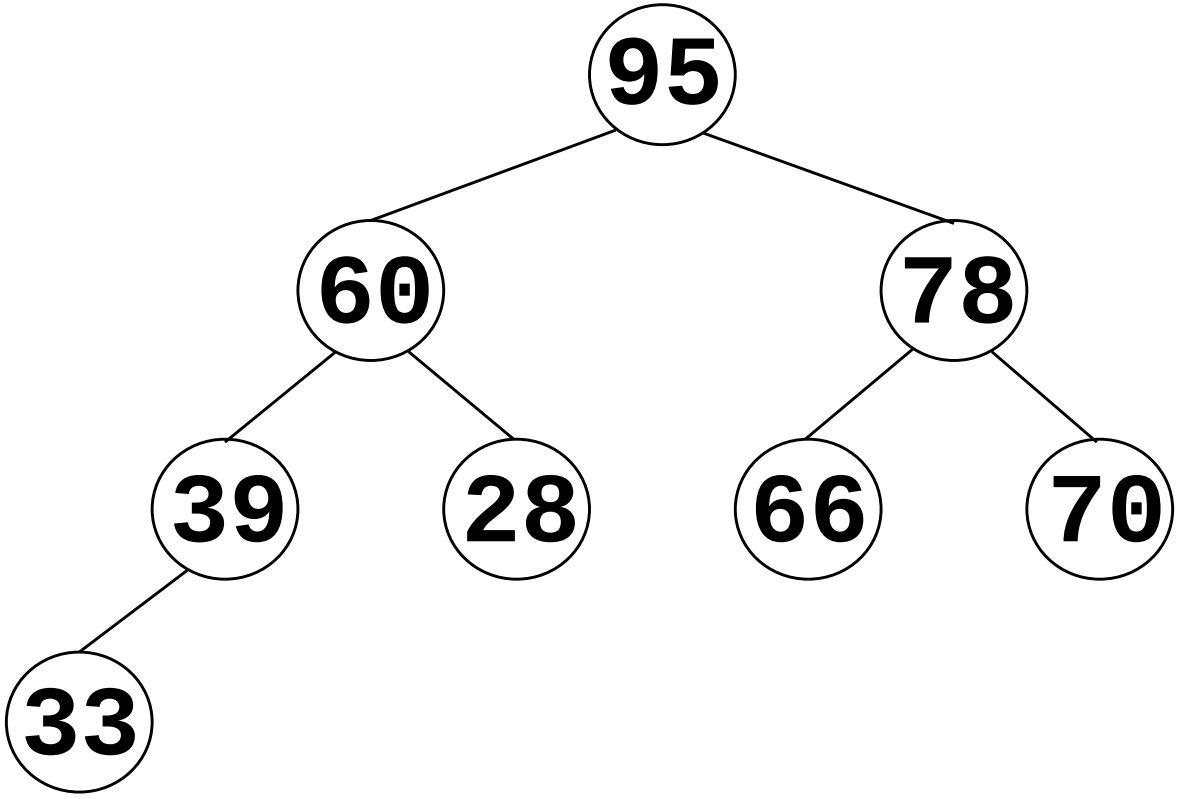
Métodos *Heap*: aumentando prioridade

Método subir:



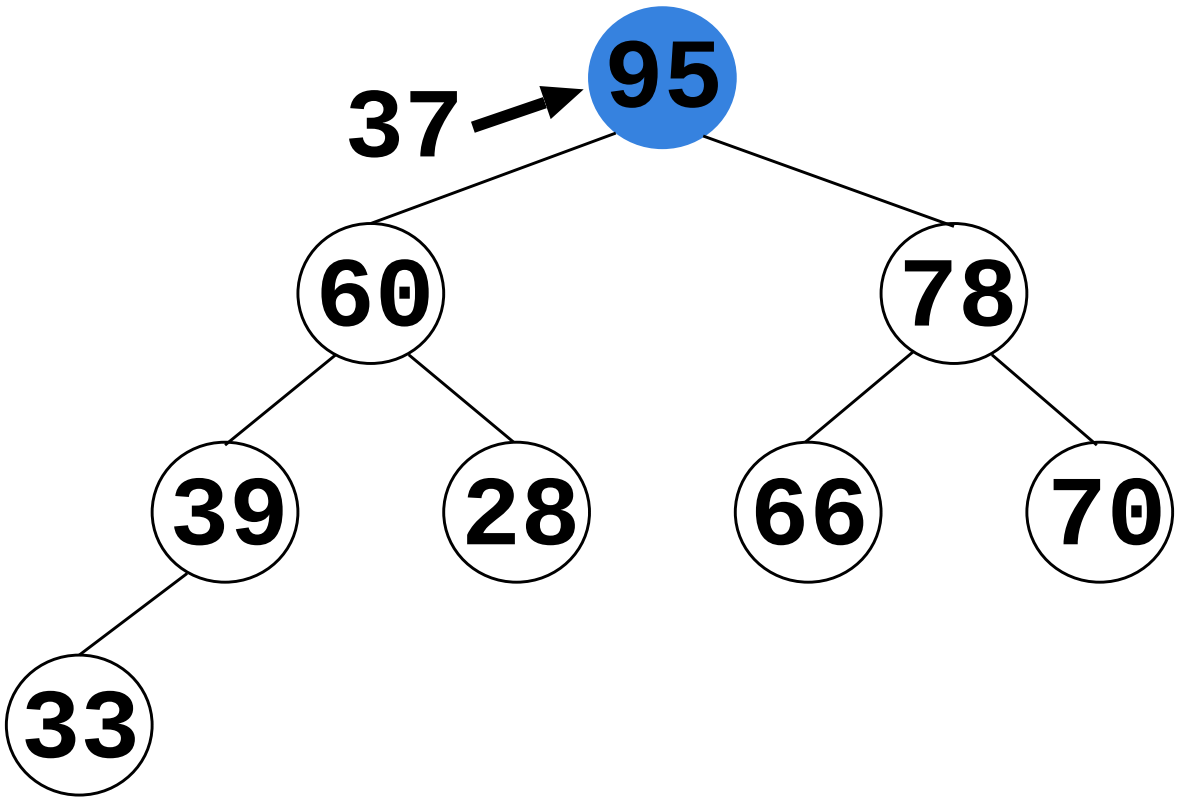
Métodos *Heap*: diminuindo prioridade

Método descer:



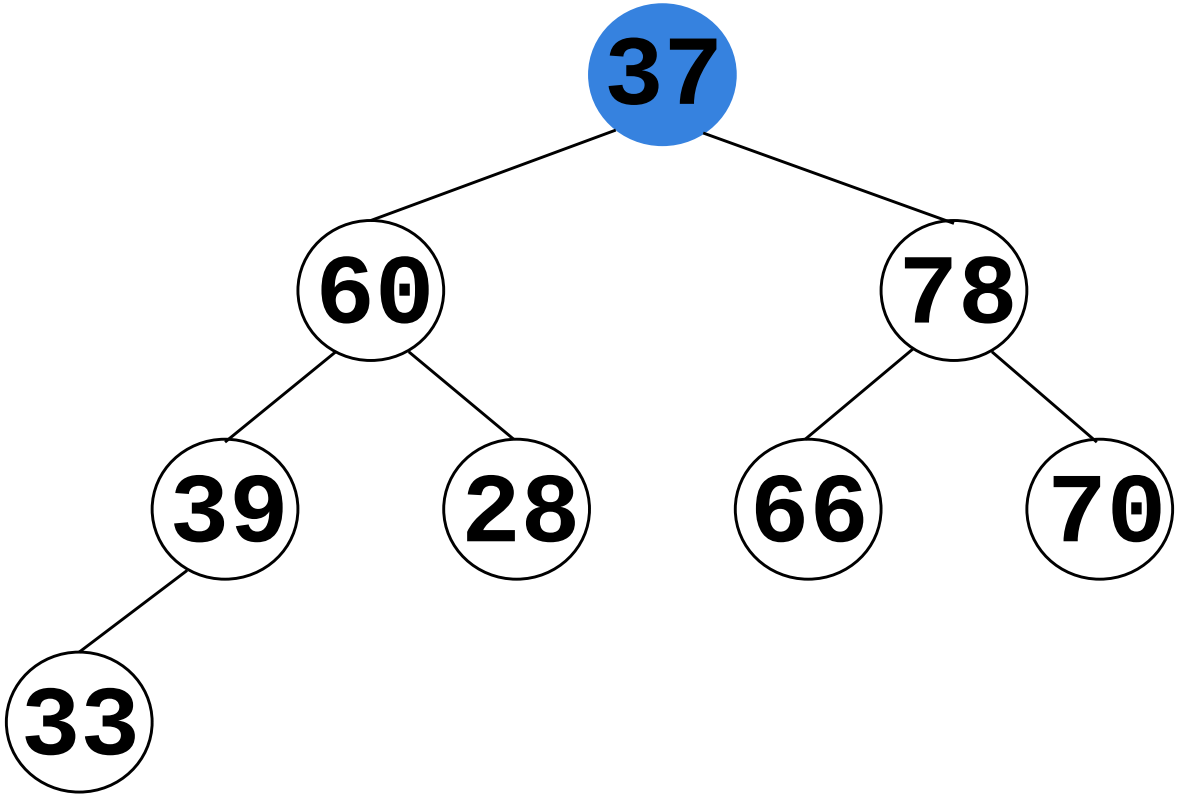
Métodos *Heap*: diminuindo prioridade

Método descer:



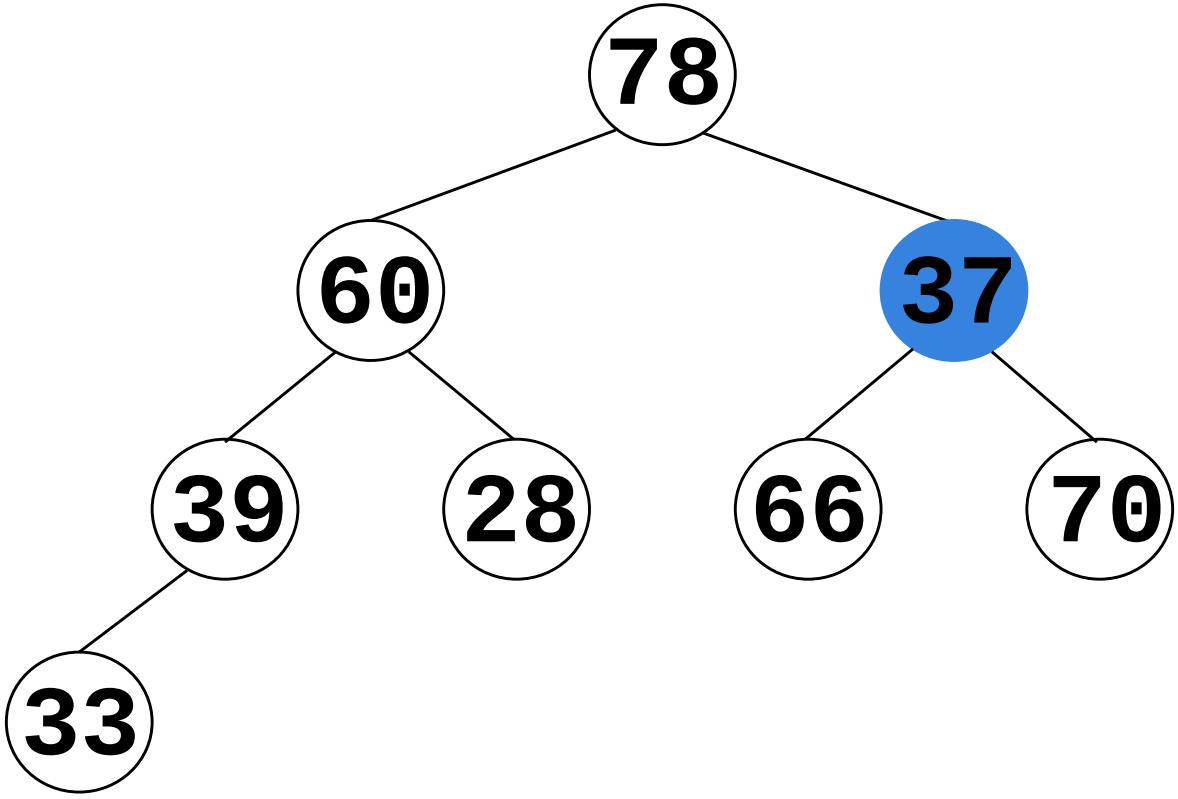
Métodos *Heap*: diminuindo prioridade

Método descer:



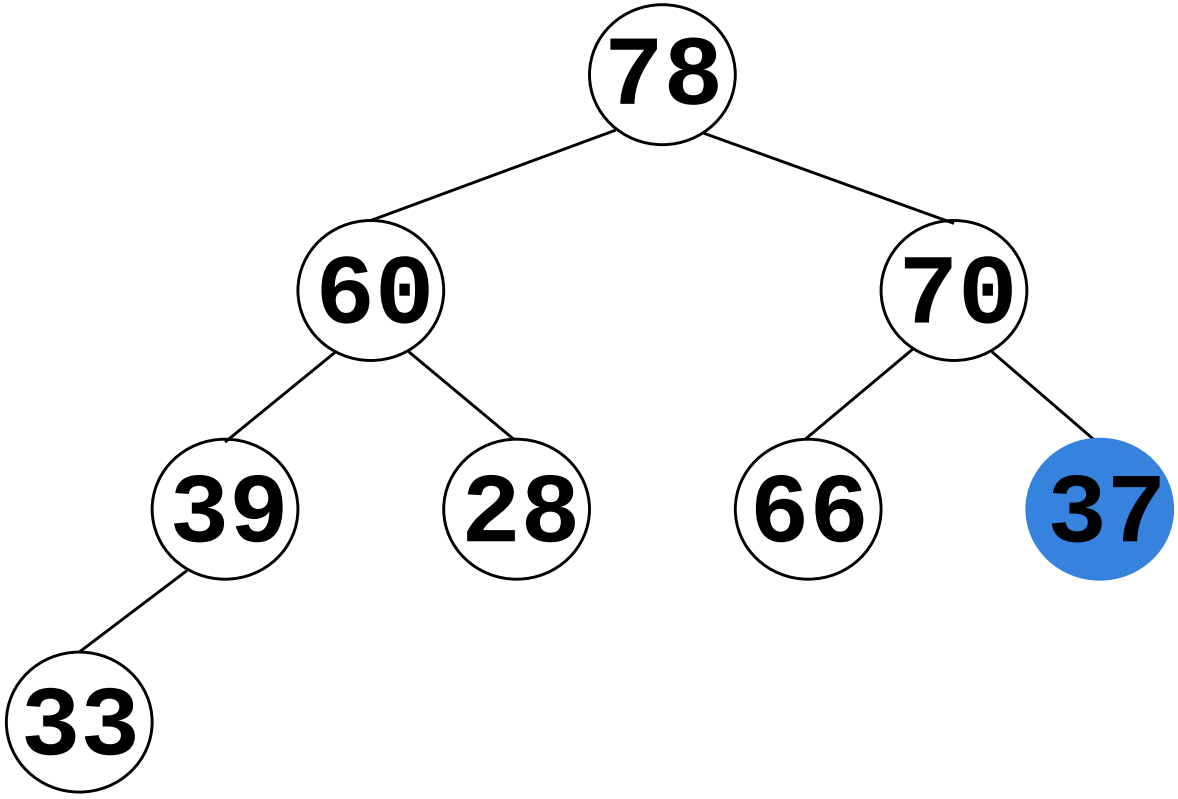
Métodos *Heap*: diminuindo prioridade

Método descer:



Métodos *Heap*: diminuindo prioridade

Método descer:



Métodos *Heap*

```
1  subir(i):
2      j = ⌊i/2⌋
3      se j ≥ 1 então
4          se T[i] > T[j] então
5              T[i] ⇔ T[j]
6              subir(j)
```

```
1  descer(i, n):
2      j = 2 * i
3      se j ≤ n então
4          se j < n então
4              se T[j + 1] > T[j] então
5                  j = j + 1
6          se T[i] < T[j] então
7              T[i] ⇔ T[j]
8              descer(j, n)
```

Métodos *Heap*

```
1  subir(i):
2      j = ⌊i/2⌋
3      se j ≥ 1 então
4          se T[i] > T[j] então
5              T[i] ⇔ T[j]
6              subir(j)
```

Análise da complexidade:

```
1  descer(i, n):
2      j = 2 * i
3      se j ≤ n então
4          se j < n então
4              se T[j + 1] > T[j] então
5                  j = j + 1
6          se T[i] < T[j] então
7              T[i] ⇔ T[j]
8              descer(j, n)
```

Métodos *Heap*

```
1  subir(i):
2      j = ⌊i/2⌋
3      se j ≥ 1 então
4          se T[i] > T[j] então
5              T[i] ⇔ T[j]
6              subir(j)
```

```
1  descer(i, n):
2      j = 2 * i
3      se j ≤ n então
4          se j < n então
4              se T[j + 1] > T[j] então
5                  j = j + 1
6          se T[i] < T[j] então
7              T[i] ⇔ T[j]
8              descer(j, n)
```

Análise da complexidade:

- **Subir:** percurso de caminho em árvore binária completa;

Métodos *Heap*

```
1  subir(i):
2      j = ⌊i/2⌋
3      se j ≥ 1 então
4          se T[i] > T[j] então
5              T[i] ⇔ T[j]
6              subir(j)
```

```
1  descer(i, n):
2      j = 2 * i
3      se j ≤ n então
4          se j < n então
4              se T[j + 1] > T[j] então
5                  j = j + 1
6          se T[i] < T[j] então
7              T[i] ⇔ T[j]
8              descer(j, n)
```

Análise da complexidade:

- **Subir:** percurso de caminho em árvore binária completa;
- **Descer:** percurso de caminho em árvore binária completa;

Métodos *Heap*

```
1  subir(i):
2      j = ⌊i/2⌋
3      se j ≥ 1 então
4          se T[i] > T[j] então
5              T[i] ⇔ T[j]
6              subir(j)
```

```
1  descer(i, n):
2      j = 2 * i
3      se j ≤ n então
4          se j < n então
4              se T[j + 1] > T[j] então
5                  j = j + 1
6          se T[i] < T[j] então
7              T[i] ⇔ T[j]
8              descer(j, n)
```

Análise da complexidade:

- **Subir:** percurso de caminho em árvore binária completa;
- **Descer:** percurso de caminho em árvore binária completa;
- **Complexidade:** **O**(log **n**)

Métodos *Heap*: inserir, remover e construir

Métodos *Heap*: inserir, remover e construir

- **Inserir:** insere em $n + 1$ e aplica *subir*(). **Complexidade:**

Métodos *Heap*: inserir, remover e construir

- **Inserir:** insere em $n + 1$ e aplica *subir*(). **Complexidade:** $O(\log n)$;

Métodos *Heap*: inserir, remover e construir

- **Inserir:** insere em $n + 1$ e aplica *subir*(). **Complexidade:** $O(\log n)$;
- **Remover:** remove posição 1, coloca chave n em 1 e aplica método *descer*(). **Complexidade:**

Métodos *Heap*: inserir, remover e construir

- **Inserir:** insere em $n + 1$ e aplica *subir*(). **Complexidade:** $O(\log n)$;
- **Remover:** remove posição 1, coloca chave n em 1 e aplica método *descer*(). **Complexidade:** $O(\log n)$;

Métodos *Heap*: inserir, remover e construir

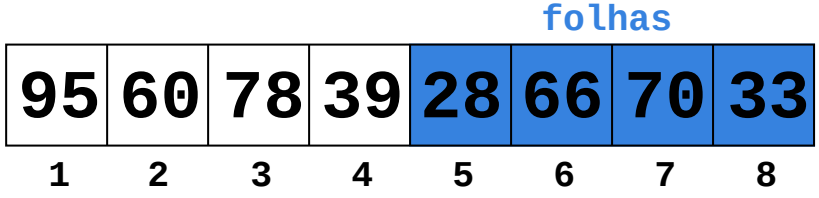
- **Inserir:** insere em $n+1$ e aplica *subir*(). **Complexidade:** $O(\log n)$;
- **Remover:** remove posição 1, coloca chave n em 1 e aplica método *descer*(). **Complexidade:** $O(\log n)$;
- **Construção:** para $i = \lfloor n/2 \rfloor$ até 1 aplique *descer*(i, n).

Métodos *Heap*: inserir, remover e construir

- **Inserir:** insere em $n+1$ e aplica *subir*(). **Complexidade:** $O(\log n)$;
- **Remover:** remove posição 1, coloca chave n em 1 e aplica método *descer*(). **Complexidade:** $O(\log n)$;
- **Construção:** para $i = \lfloor n/2 \rfloor$ até 1 aplique *descer*(i, n). **Complexidade:**

Métodos *Heap*: inserir, remover e construir

- **Inserir:** insere em $n+1$ e aplica *subir*(). **Complexidade:** $O(\log n)$;
- **Remover:** remove posição 1, coloca chave n em 1 e aplica método *descer*(). **Complexidade:** $O(\log n)$;
- **Construção:** para $i = \lfloor n/2 \rfloor$ até 1 aplique *descer*(i, n). **Complexidade:** $O(n)$;



Exercícios

1. Escreva uma função que decida se um vetor $V[1 \ldots n]$ é um *heap*.

Exercícios

1. Escreva uma função que decida se um vetor $V[1 \ldots n]$ é um *heap*.
2. O que você acha de ordenarmos um vetor em ordem decrescente com o objetivo de transformá-lo em um *heap*?

Bibliografia Utilizada

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. *Introduction to Algorithms*, 3ª edição, MIT Press, 2009.

SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Pascal e C, 2ª edição, Cengage Learning, 2009.