Aula 10: Algoritmos de ordenação em arranjos

Ordenação por *heap*

David Déharbe
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada

Download me from `http://DavidDeharbe.github.io`.

PPgSC

Plano da aula

Heap

Listas de prioridade

Ordenação

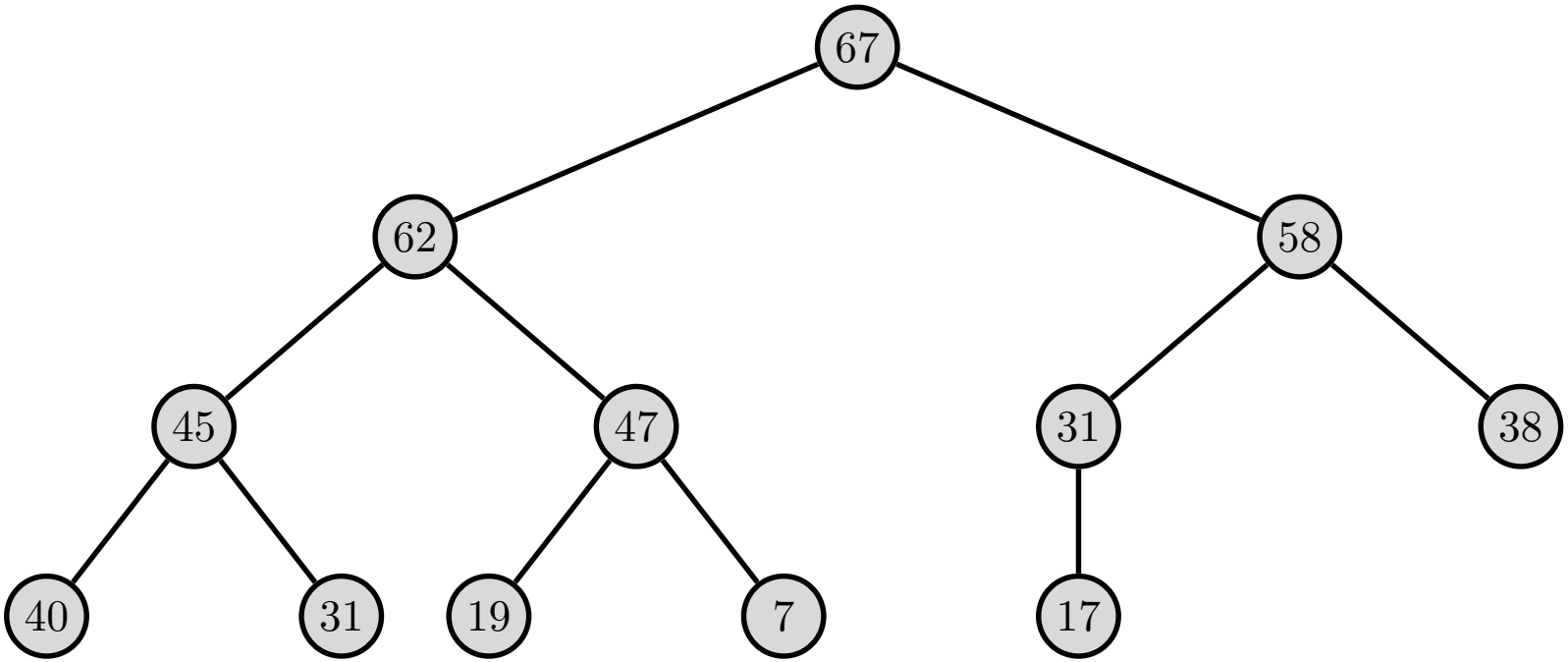
Introdução

- Ordenação em $\Theta(n \log n)$ (como ordenação por fusão);
- Sem arranjo auxiliar (como ordenação por inserção);
- Baseado no conceito de *heap*.
  - Listas de prioridade

PPgSC

Heap

Conceito

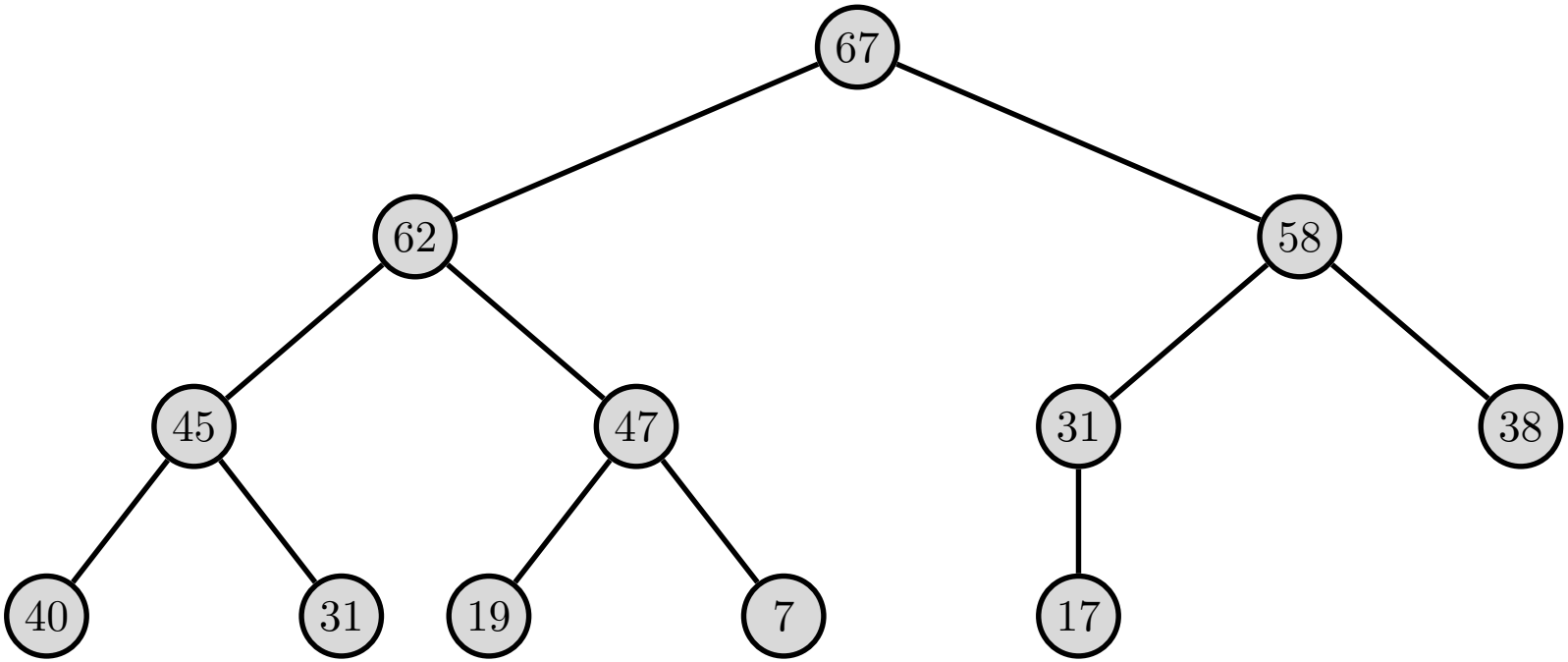


- Árvore binária
  - *value*(N): valor no nó N;
  - *left*(N): filho esquerdo do nó N (ou Nil);
  - *right*(N): filho direito do nó N (ou Nil).

PPgSC

Heap

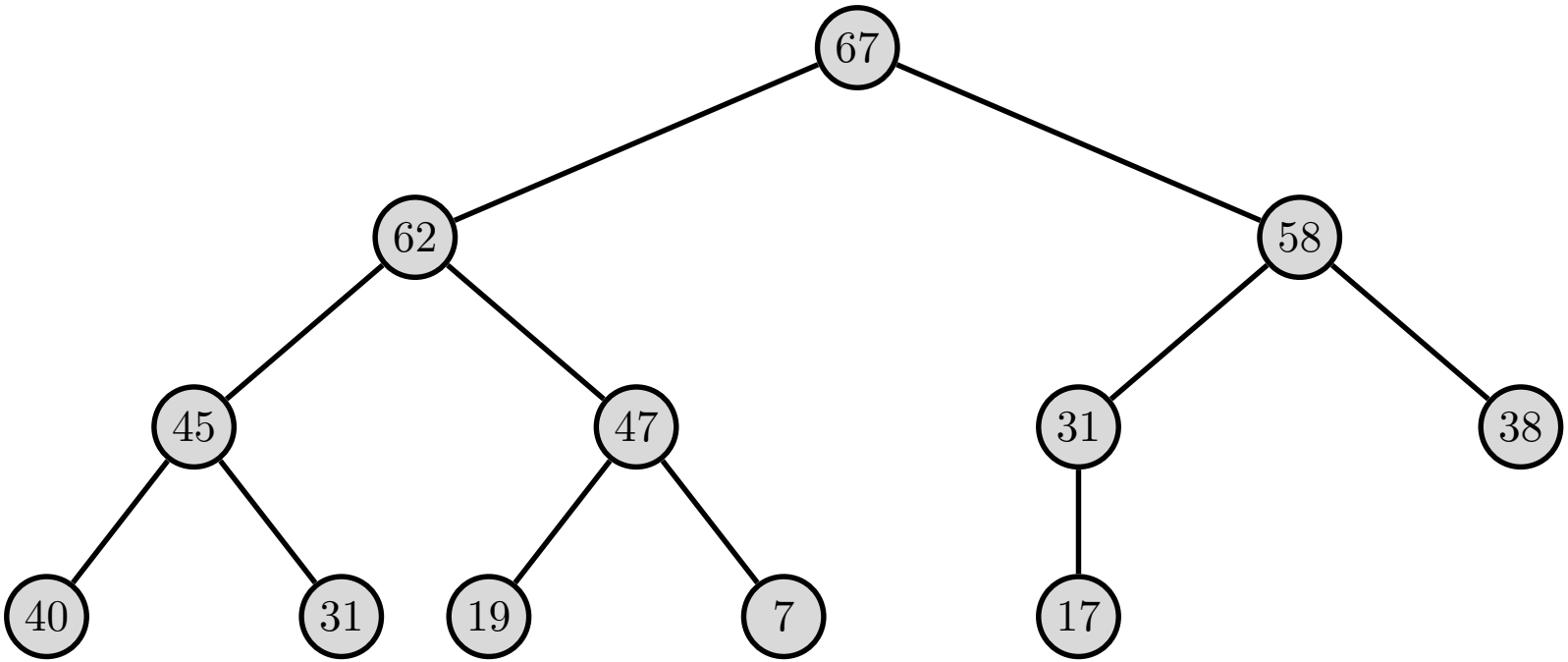
Conceito



- Árvore binária cheia, e o último nível é preenchido da esquerda para a direita. Propriedade estrutural

Heap

Conceito

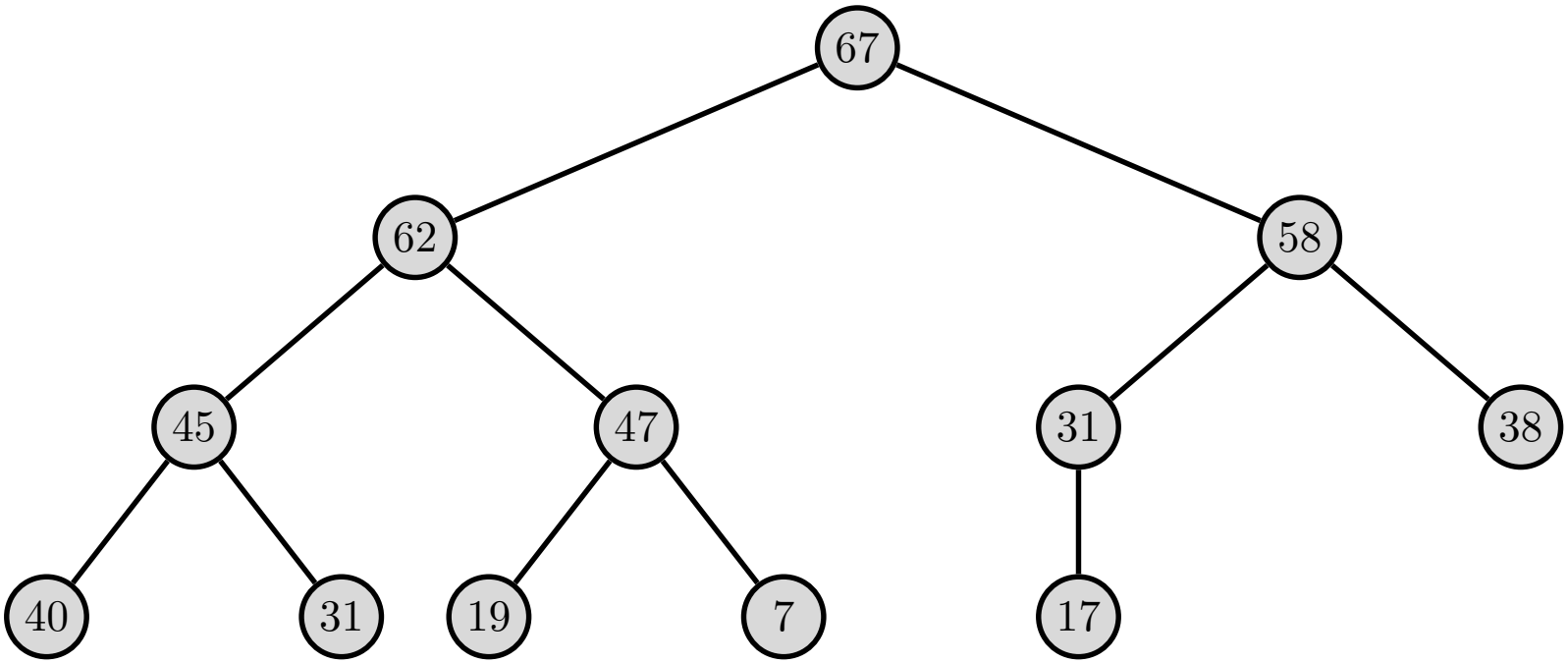


- Em cada nó, o valor é maior ou igual aos valores nos nós filhos
  - Propriedade de ordenação
  - $\forall N \cdot \mathit{left}(N) = \textsc{Nil} \vee \mathit{value}(N) \geq \mathit{value}(\mathit{left}(N))$;
  - $\forall N \cdot \mathit{right}(N) = \textsc{Nil} \vee \mathit{value}(N) \geq \mathit{value}(\mathit{right}(N))$.
  - Corolário: o maior valor encontra-se sempre na raiz.

PPgSC

Heap

Exercício

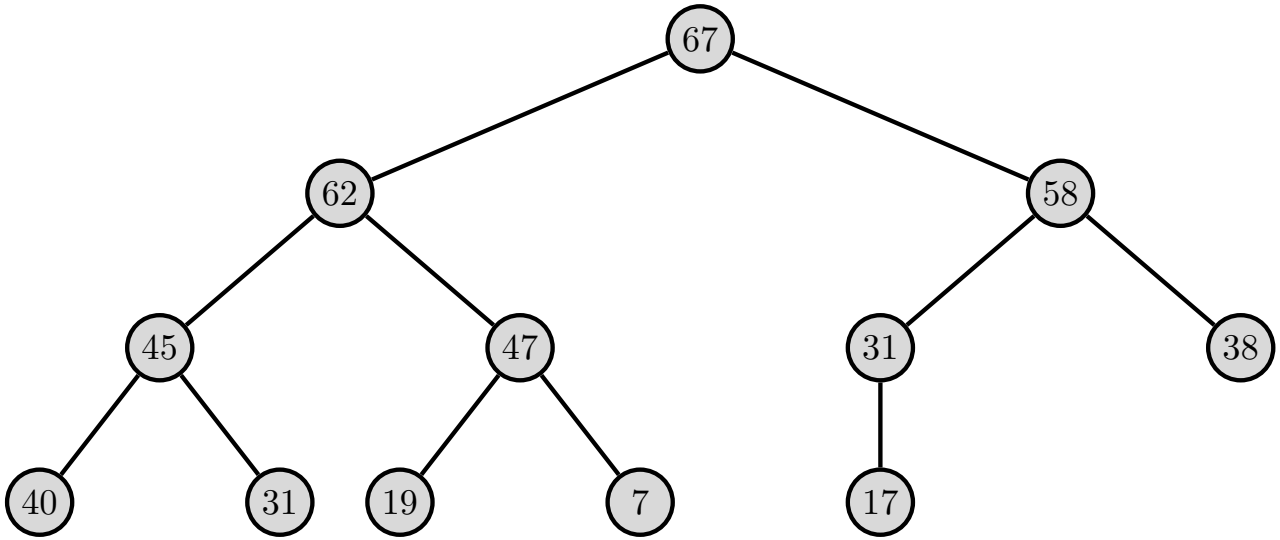


- Seja um *heap* com n elementos.
- Qual é a quantidade máxima de elementos no nível i (assumindo que 1 é o nível da raiz, 2 o nível seguinte, etc.)?
- Qual é a altura máxima do *heap*?
- O que podemos dizer de cada sub-árvore de um *heap*?

PPgSC

Heap

Representação dos dados



- Representação em um arranjo, digamos A:
  - seja N_i o nó na posição i;
  - N_i: $value(N_i) = A[i]$, $left(N_i) = 2i$, $right(N_i) = 2i + 1$, $up(N_i) = \lfloor i/2 \rfloor$.
  - A raiz fica na posição 1.

67	62	58	45	47	31	38	40	31	19	7	17
1	2	3	4	5	6	7	8	9	10	11	12

PPgSC

Heap para listas de prioridade

Especificação

A lista de prioridade é uma coleção de elementos com as seguintes operações:

1. Obter elemento de maior prioridade
2. Inserir um novo elemento
3. Retirar o elemento de maior prioridade
4. Construir a lista a partir de uma sequência qualquer inicial

PPgSC

Heap para listas de prioridade

Representação dos dados

Um arranjo A onde

- $num(A)$: número de elementos na lista
- $size(A)$: capacidade máxima da lista
- $0 \leq num(A) \leq size(A)$
- $\forall i | 2 \leq i \leq num(A) \cdot A(i) \leq A(\lfloor i/2 \rfloor)$
- Exemplo:

67	62	58	45	47	31	38	40	31	19	7	17	···	···	···	···
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- $num = 12$
- $size = 16$

Heap para listas de prioridade

Obtenção do maior elemento

GET-MAX(H)

// $H = \langle a_1, \ldots, a_{size(H)} \rangle \wedge 0 < num(H)$

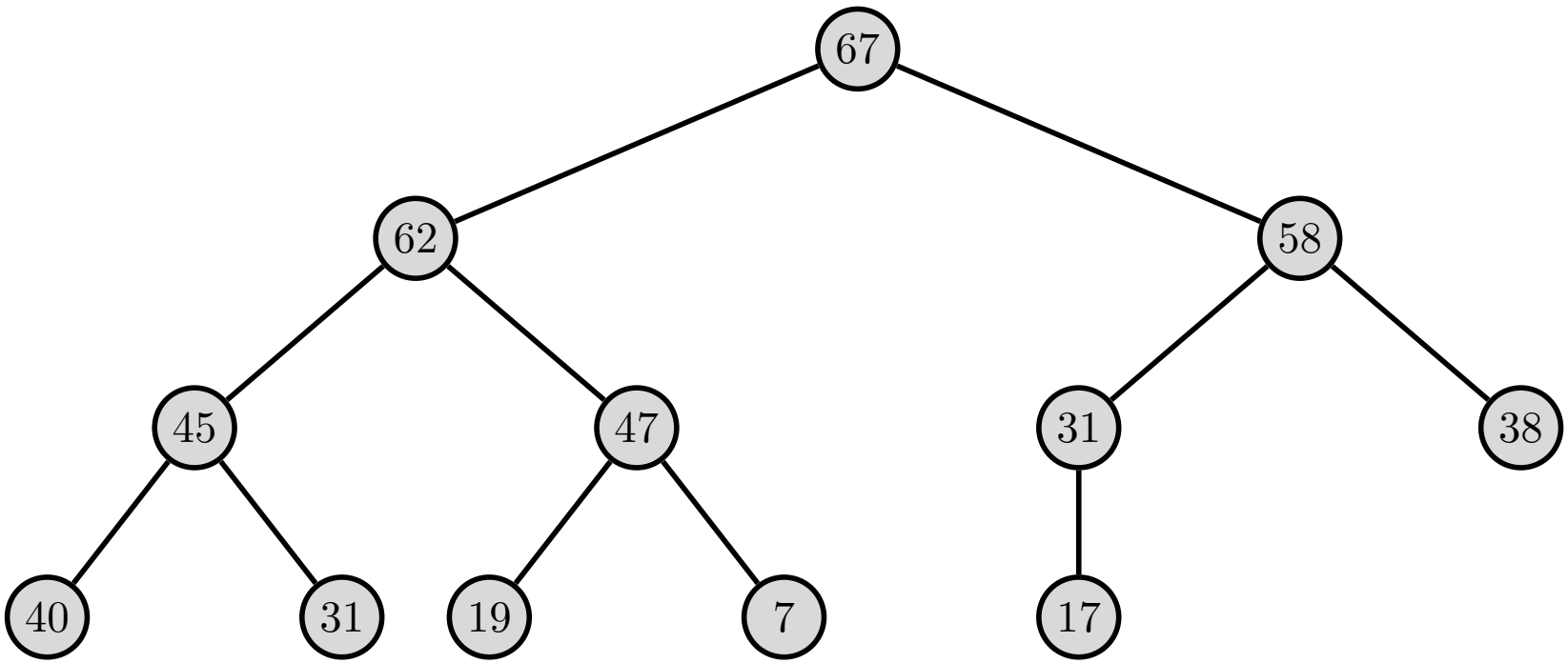
1 **return** $H[1]$

// $r = \mathbf{max}\{a_1, \ldots, a_{num(H)}\}$

PPgSC

Heap para listas de prioridade

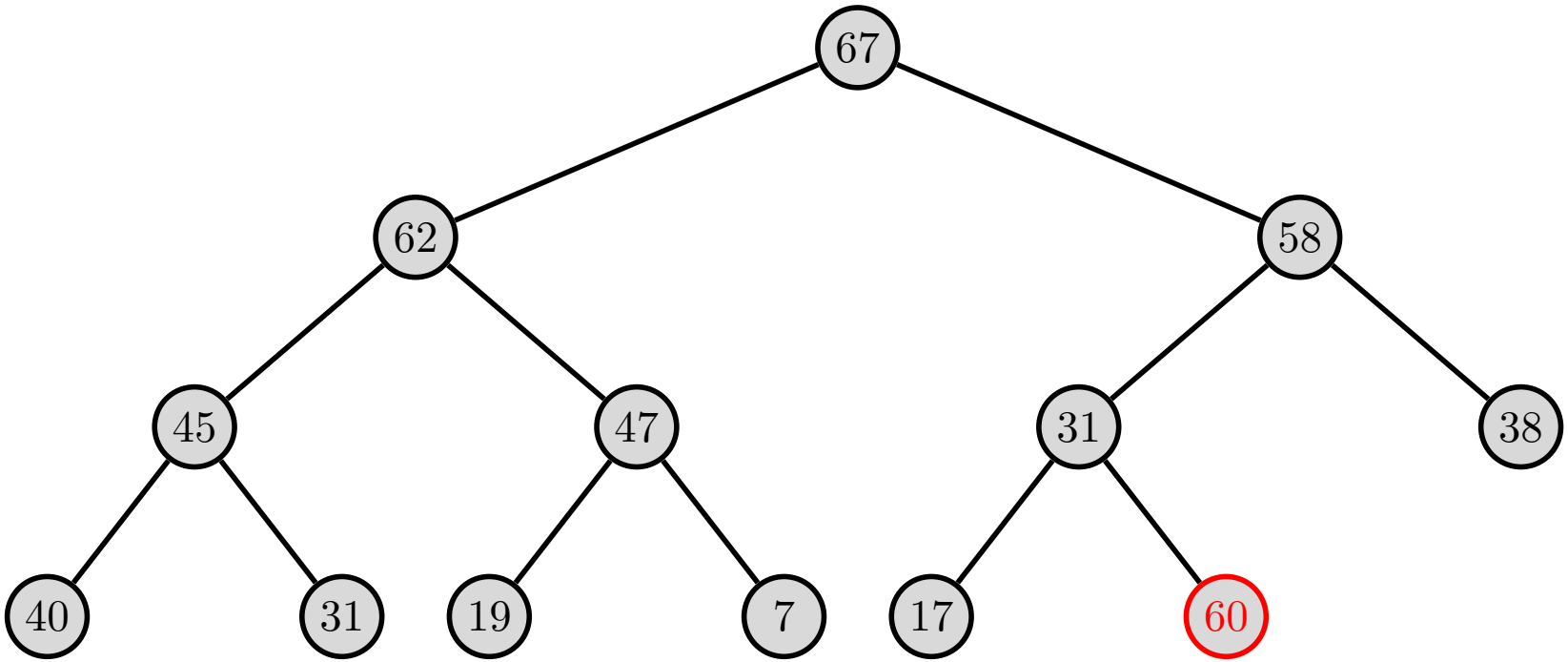
Inserção de um elemento: princípio e exemplo



- Inserir 60.

Heap para listas de prioridade

Inserção de um elemento: princípio e exemplo

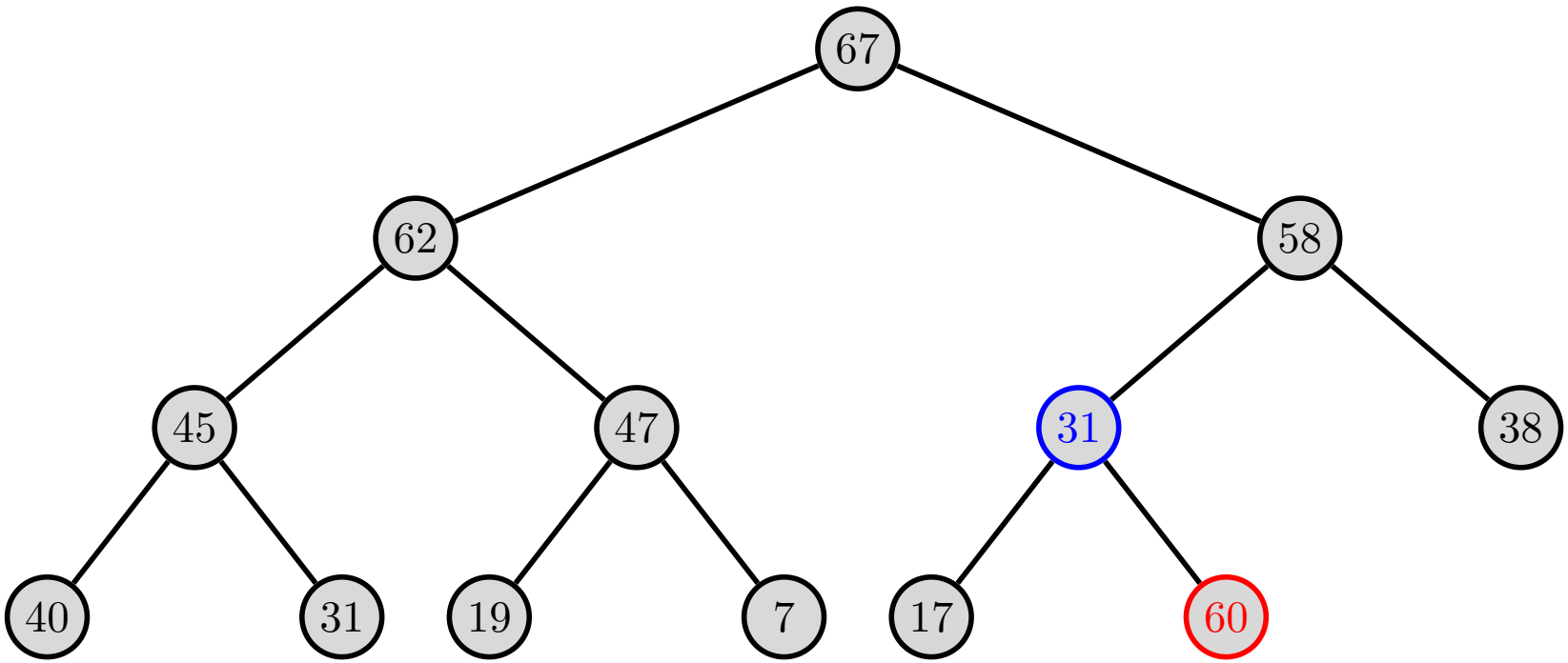


- Restrição: Preservar a propriedade estrutural.
- Criar uma nova folha a direita no último nível.

PPgSC

Heap para listas de prioridade

Inserção de um elemento: princípio e exemplo

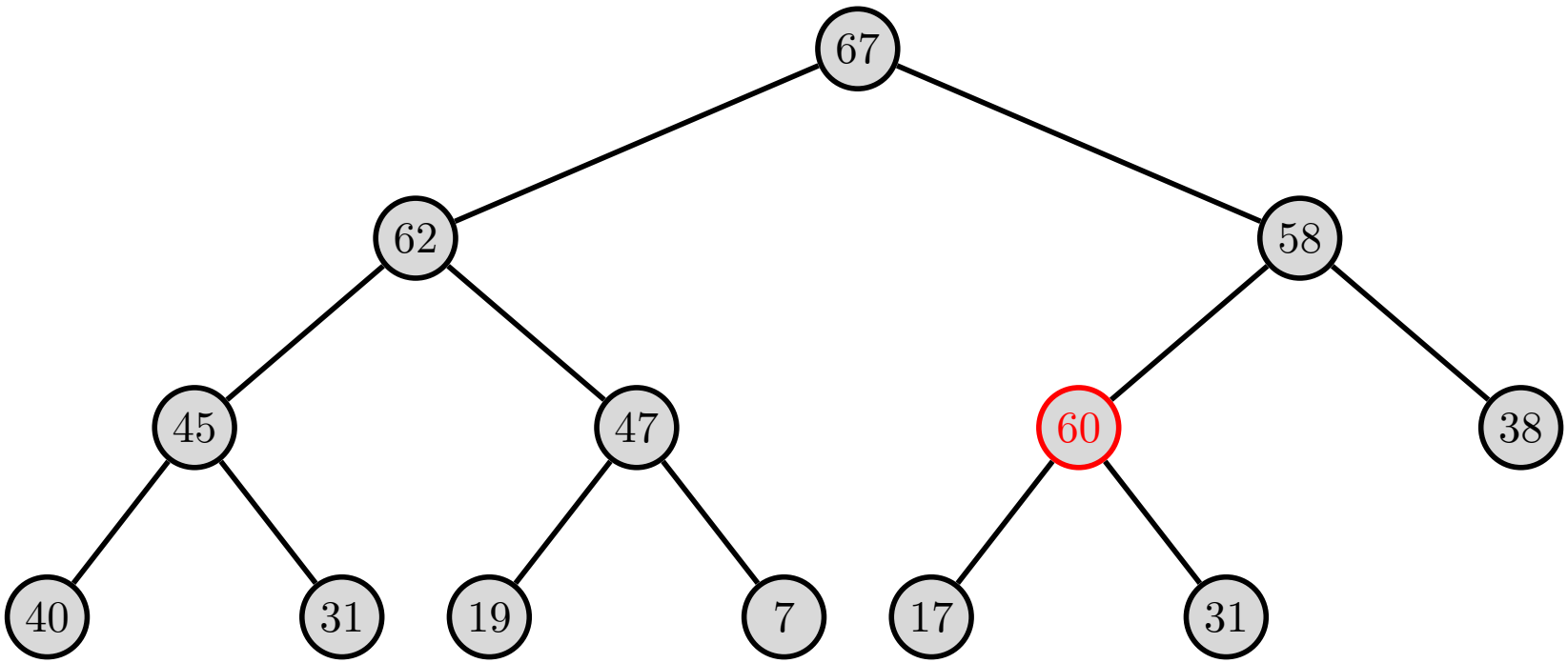


- Restrição 2: Reestabelecer a propriedade de ordenação.
- Ideia: trocar valores com pai.

PPgSC

Heap para listas de prioridade

Inserção de um elemento: princípio e exemplo



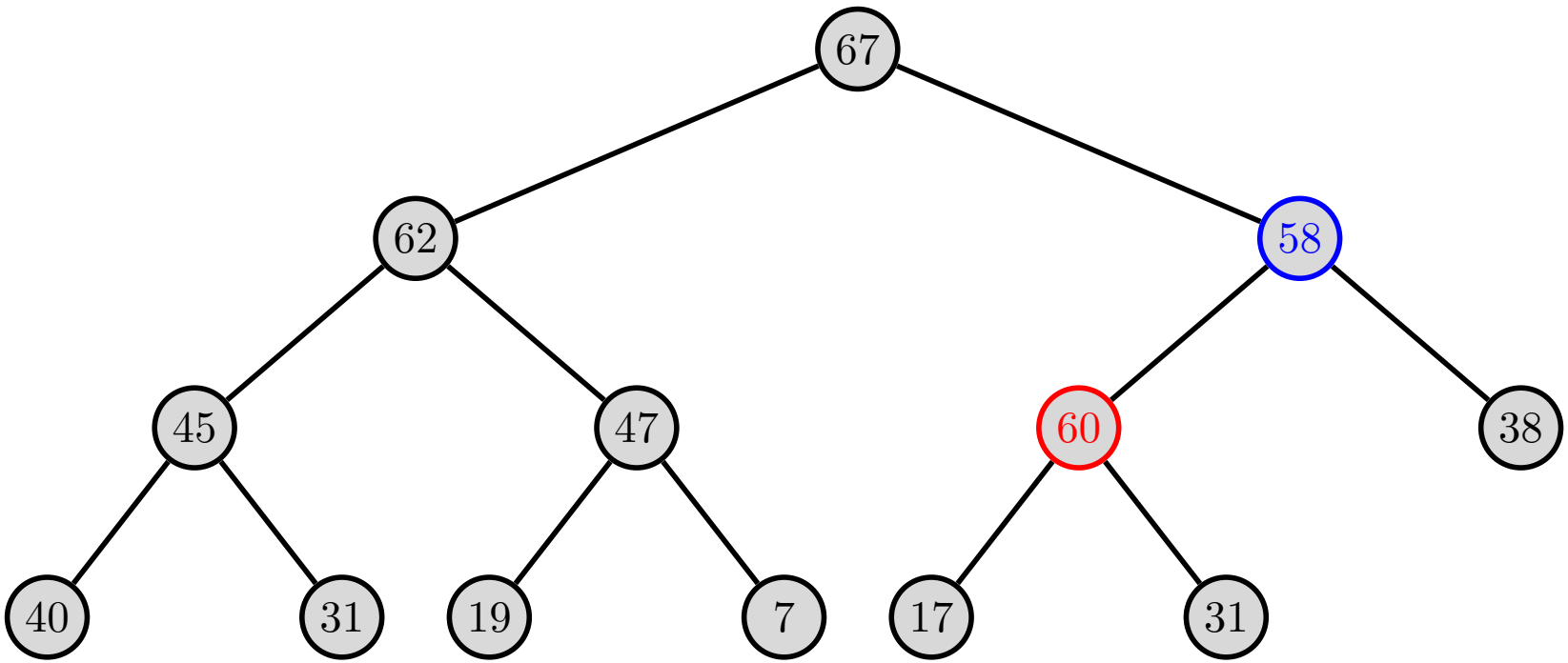
Continuar enquanto o valor "subindo"

- está em um nó com nó pai;
- é maior que o valor do pai.

PPgSC

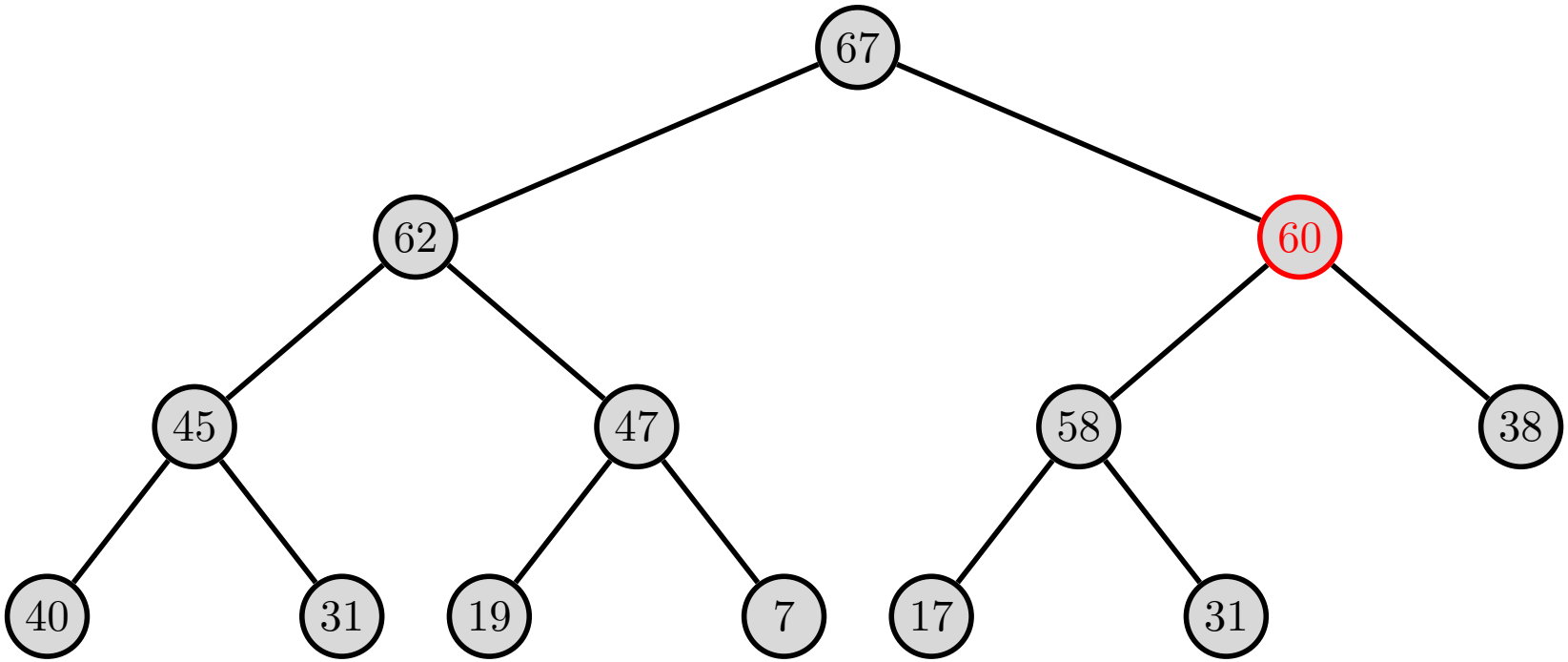
Heap para listas de prioridade

Inserção de um elemento: princípio e exemplo



Heap para listas de prioridade

Inserção de um elemento: princípio e exemplo



Exercício:

- Desenhar o estado do *heap* após inserir 50, 73, 65.

PPgSC

Heap para listas de prioridade

Algoritmo

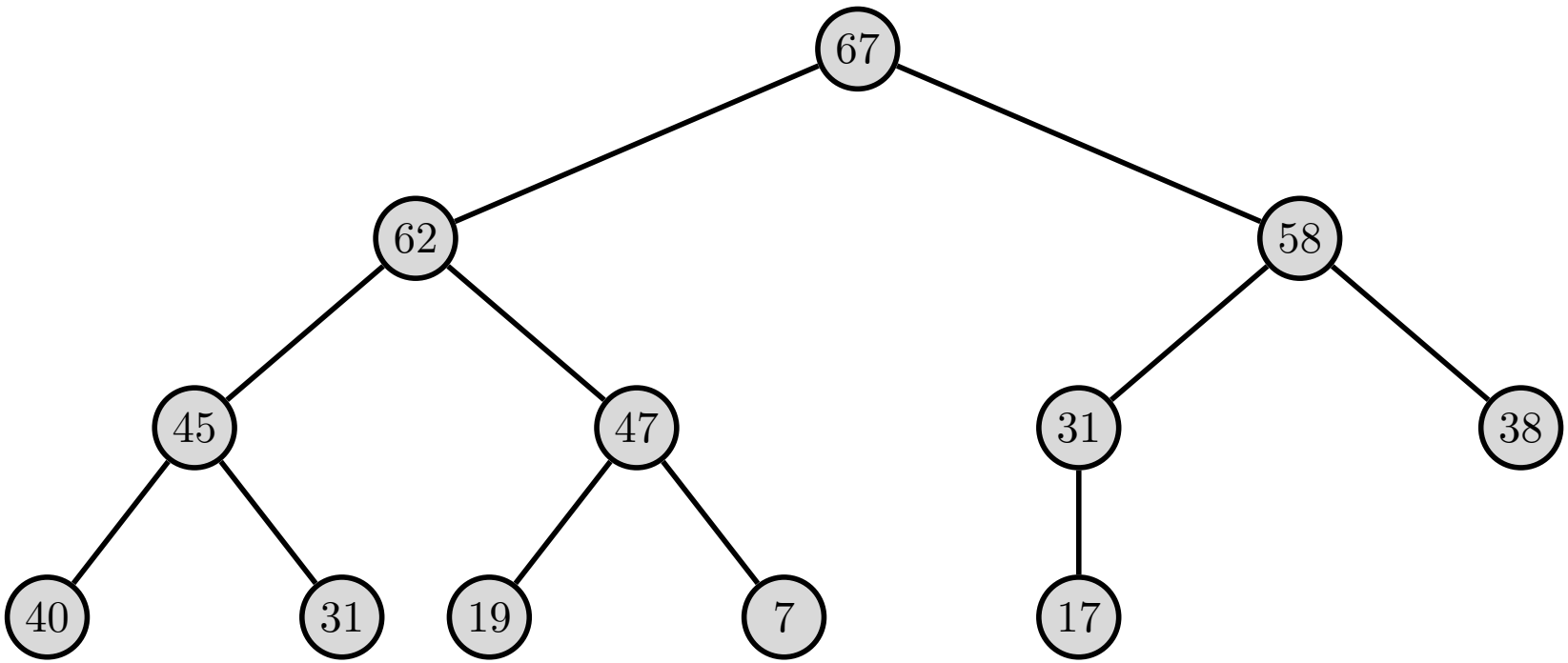
```
INSERT(H, v)
    // H = ⟨a_1, ..., a_size(H)⟩ ∧ num(H) < size(H)
1   num(H) = num(H) + 1
2   H[num(H)] = v
3   SIFT-UP(H, num(H))

SIFT-UP(H, i)
1   if i > 1 and H[i] > H[up(i)]
2       SWAP(H, i, up(i))
3       SIFT-UP(H, up(i))
```

- Complexidade no pior caso
  - Uma chamada por nível da árvore
  - $\Theta(\log Num(H))$

PPgSC

Heap para listas de prioridade

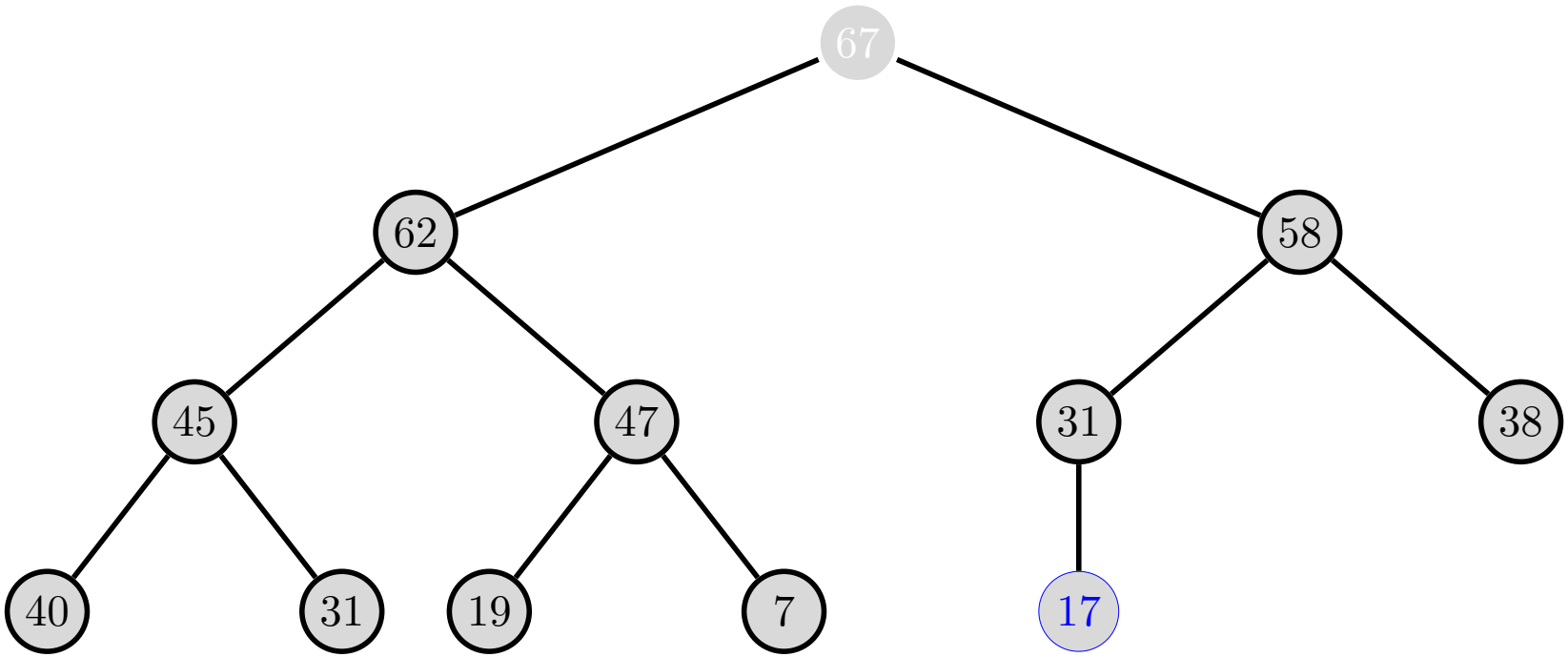
Remoção do maior elemento: princípio e exemplo



- O elemento 67 deve ser removido.

Heap para listas de prioridade

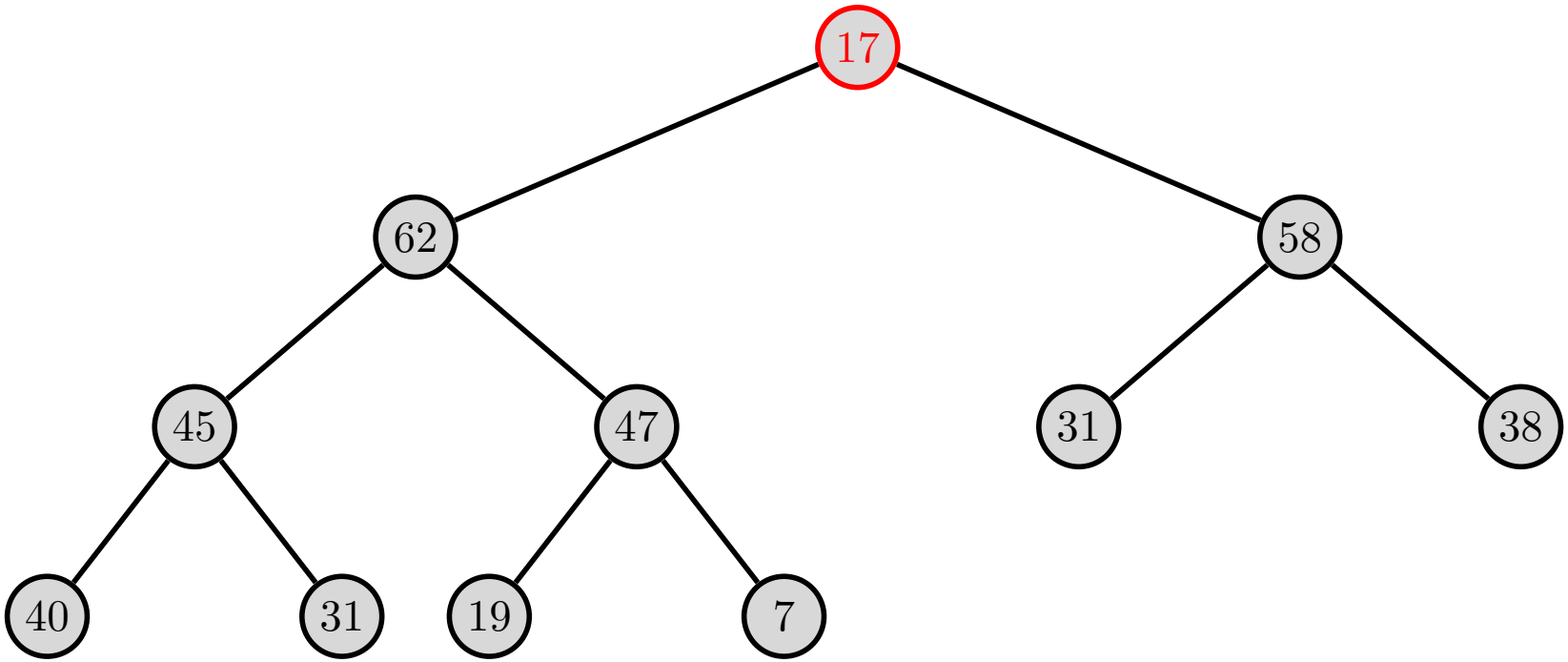
Remoção do maior elemento: princípio e exemplo



- Restrição: Preserva a propriedade estrutural.
- A folha a mais a direita é o único nó que pode ser removido.
- Mover o elemento nesta para raiz e eliminar o nó.

Heap para listas de prioridade

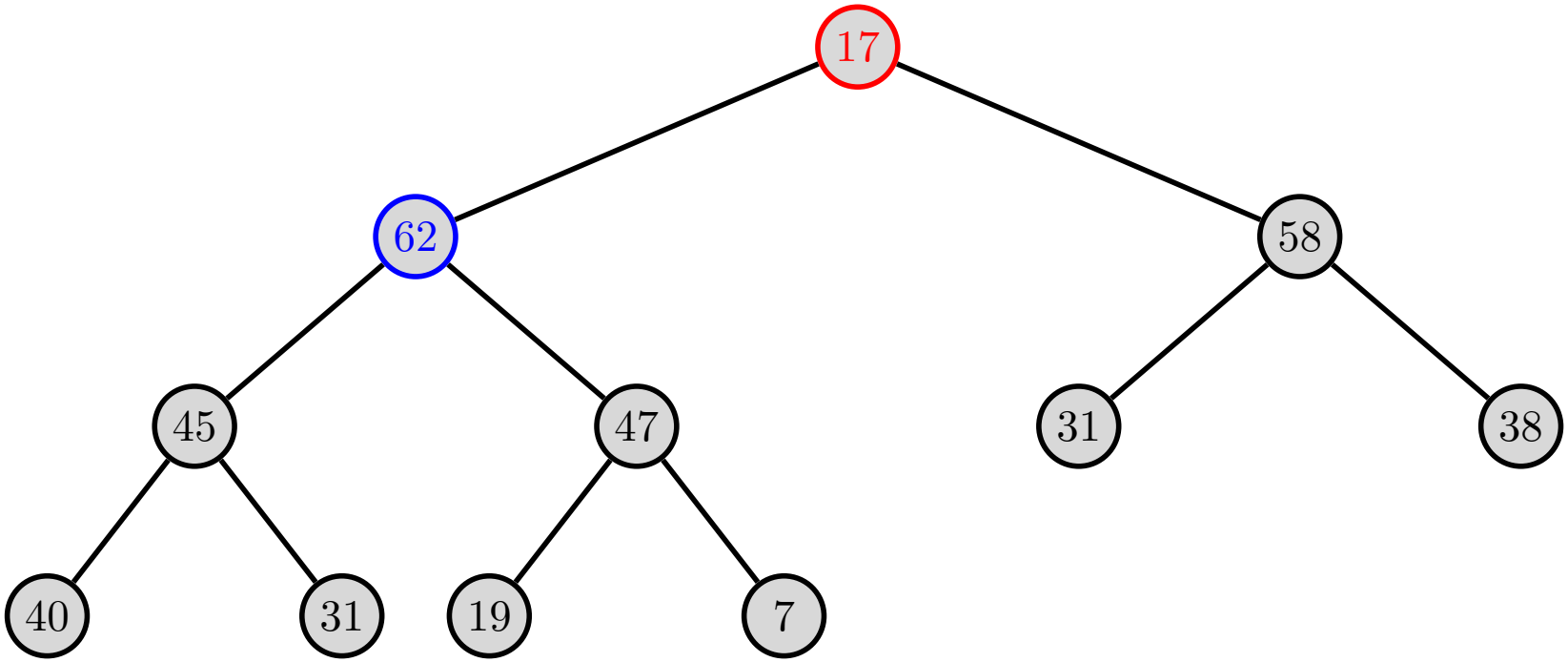
Remoção do maior elemento: princípio e exemplo



- Restrição 2: Reestabelecer a propriedade de ordenação.

Heap para listas de prioridade

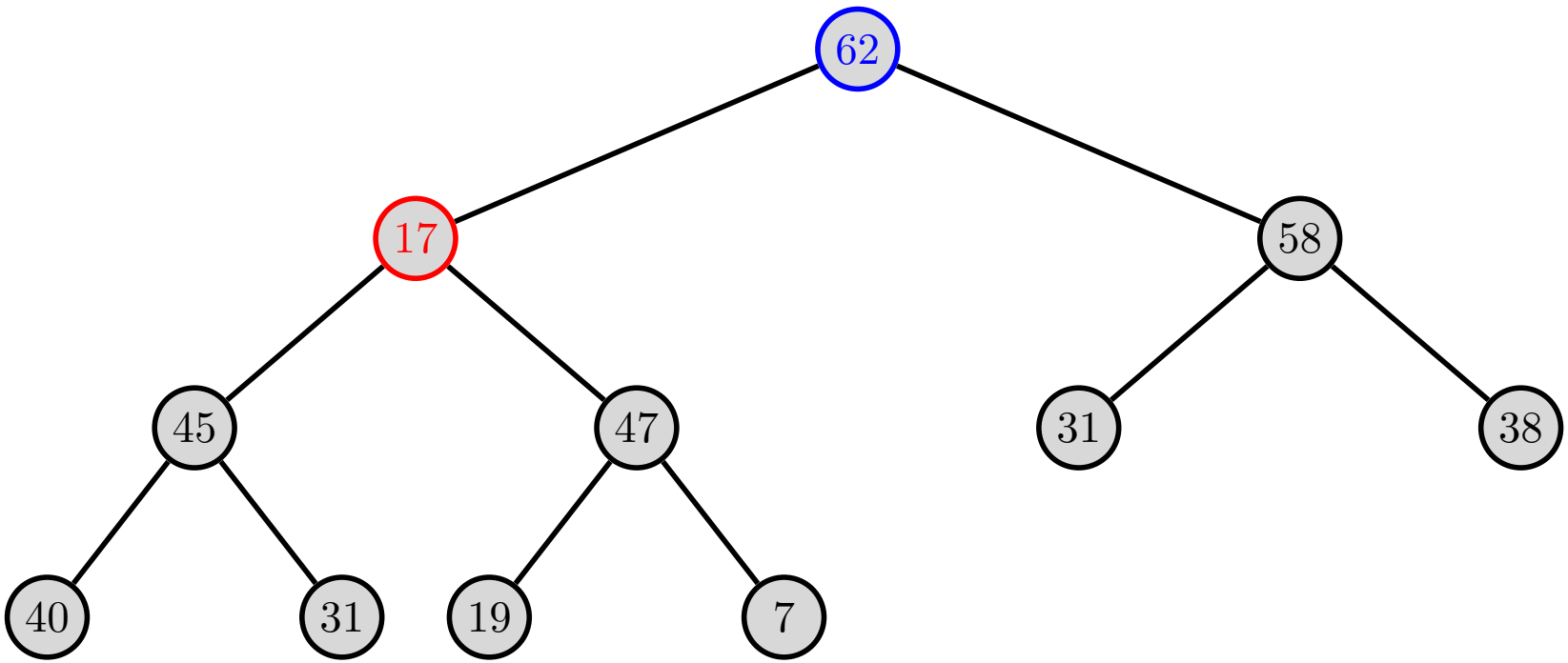
Remoção do maior elemento: princípio e exemplo



- Restrição 2: Reestabelecer a propriedade de ordenação.
- Ideia: trocar valores com filho que tem maior elemento.

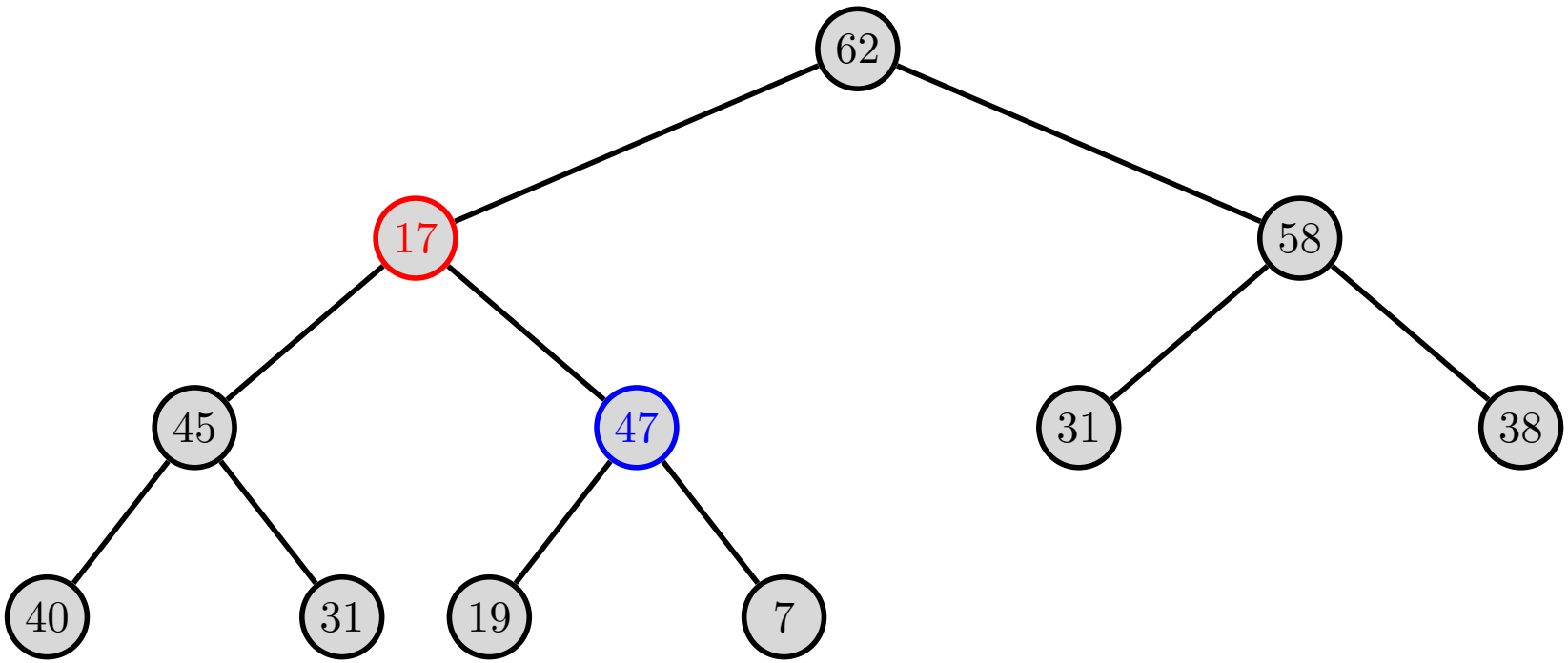
Heap para listas de prioridade

Remoção do maior elemento: princípio e exemplo



Heap para listas de prioridade

Remoção do maior elemento: princípio e exemplo



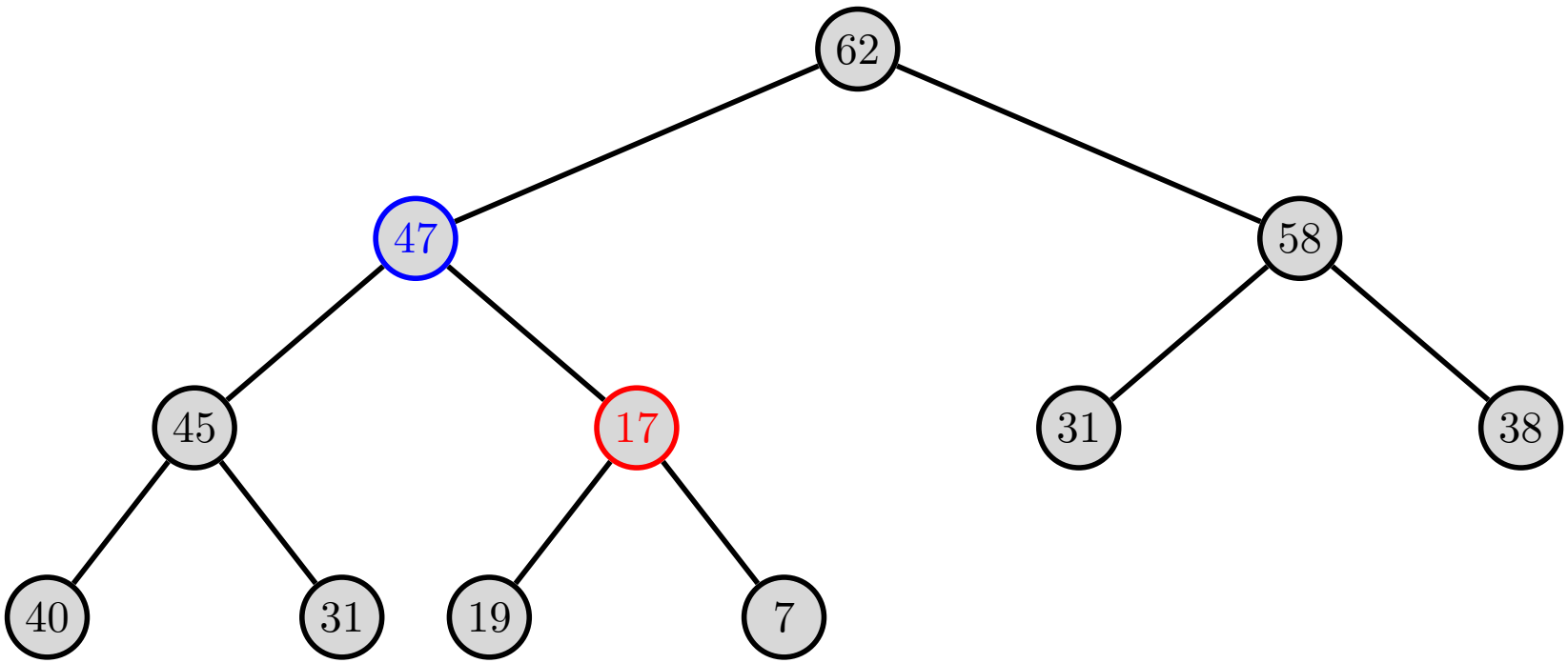
Continuar enquanto o valor "descendo"

- tem um ou dois filhos;
- é menor que o maior dos filhos.

PPgSC

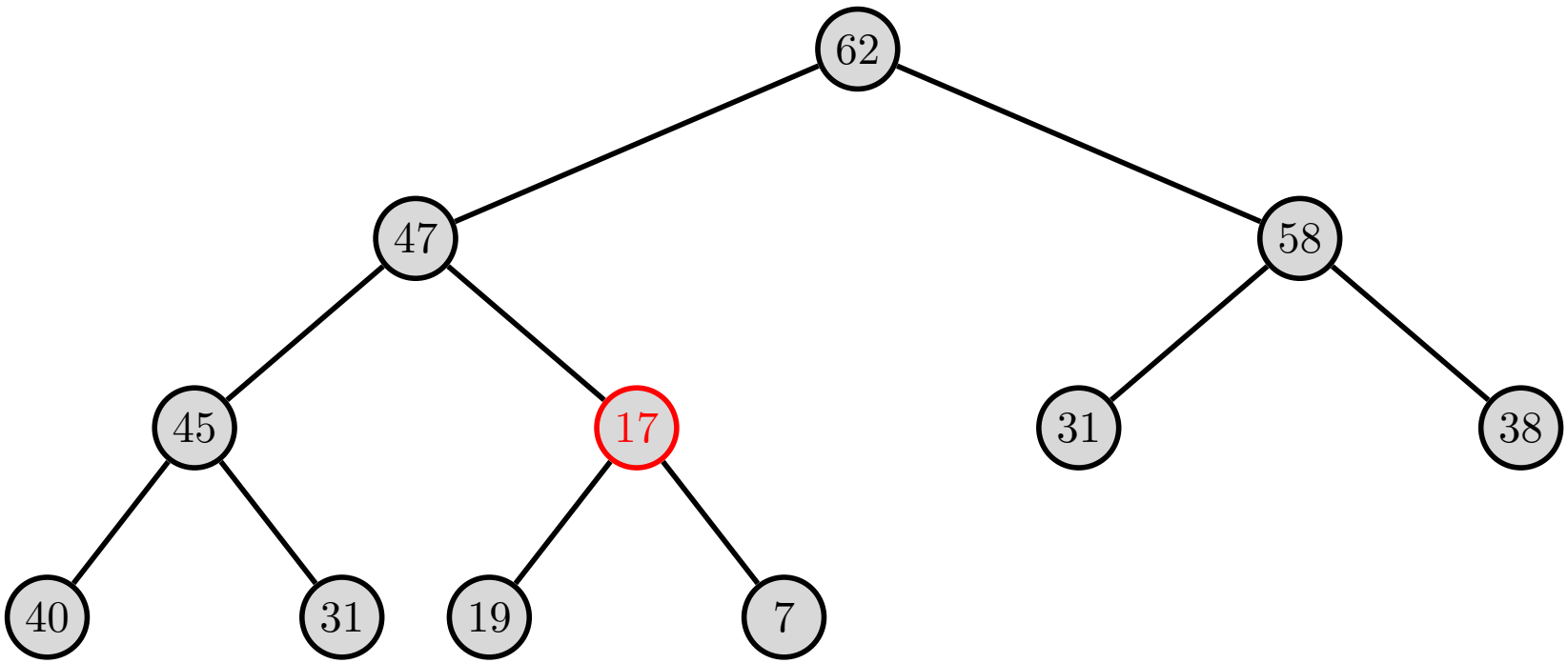
Heap para listas de prioridade

Remoção do maior elemento: princípio e exemplo



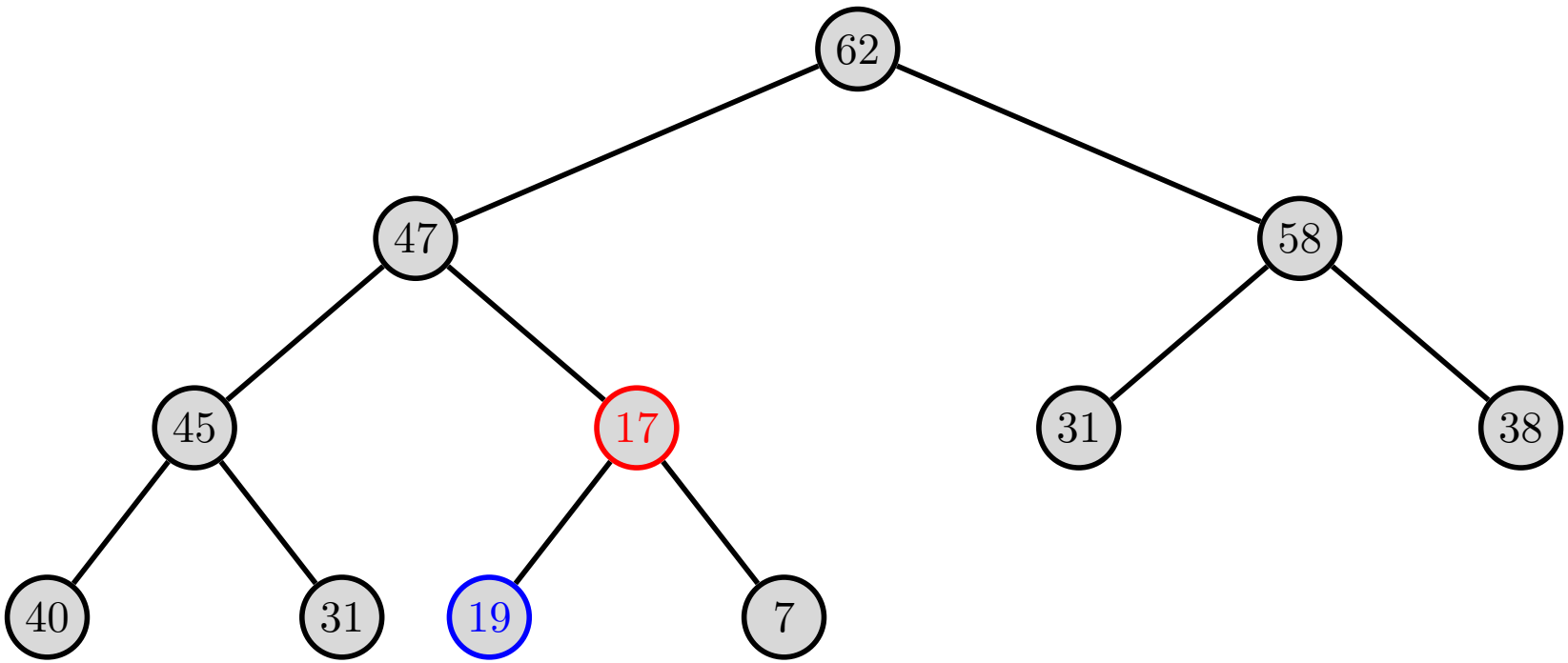
Heap para listas de prioridade

Remoção do maior elemento: princípio e exemplo



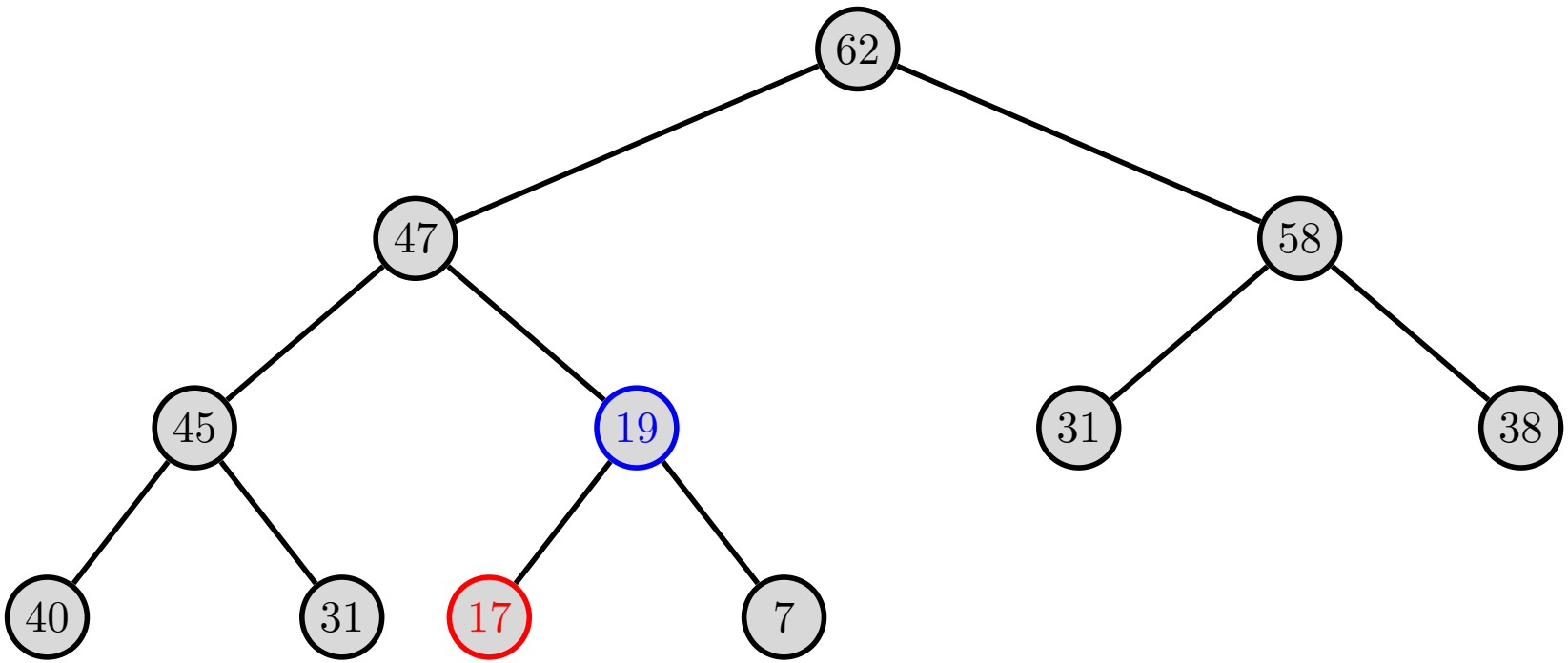
Heap para listas de prioridade

Remoção do maior elemento: princípio e exemplo



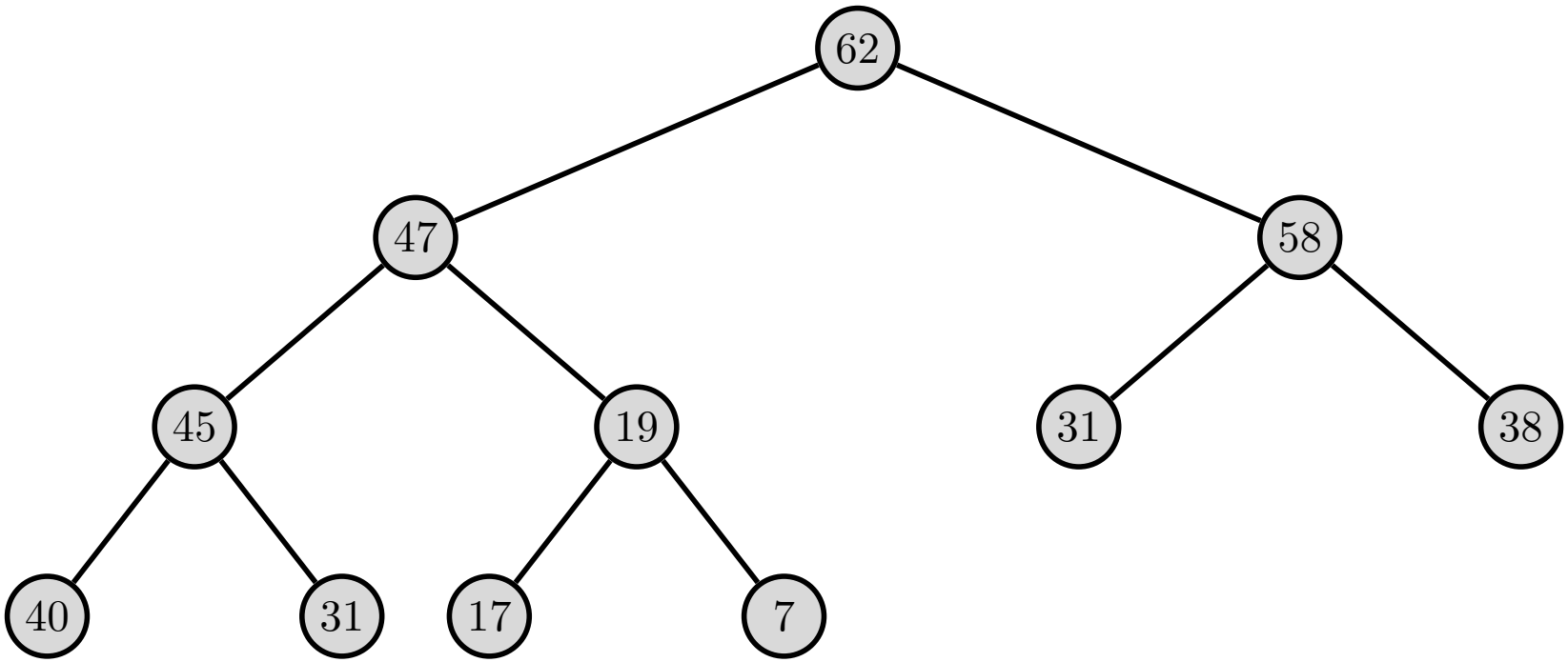
Heap para listas de prioridade

Remoção do maior elemento: princípio e exemplo



Heap para listas de prioridade

Remoção do maior elemento: princípio e exemplo



Exercício:

- Desenhar o estado do *heap* após uma, duas, três,... remoções do maior elemento.

REMOVE-MIN(H)

// $H = \langle a_1, \ldots, a_{size(H)} \rangle \wedge 0 < num(H)$

1 $H[1] = H[num(H)]$
2 $num(H) = num(H) - 1$
3 SIFT-DOWN($H, 1$)

Heap para listas de prioridade

Algoritmo

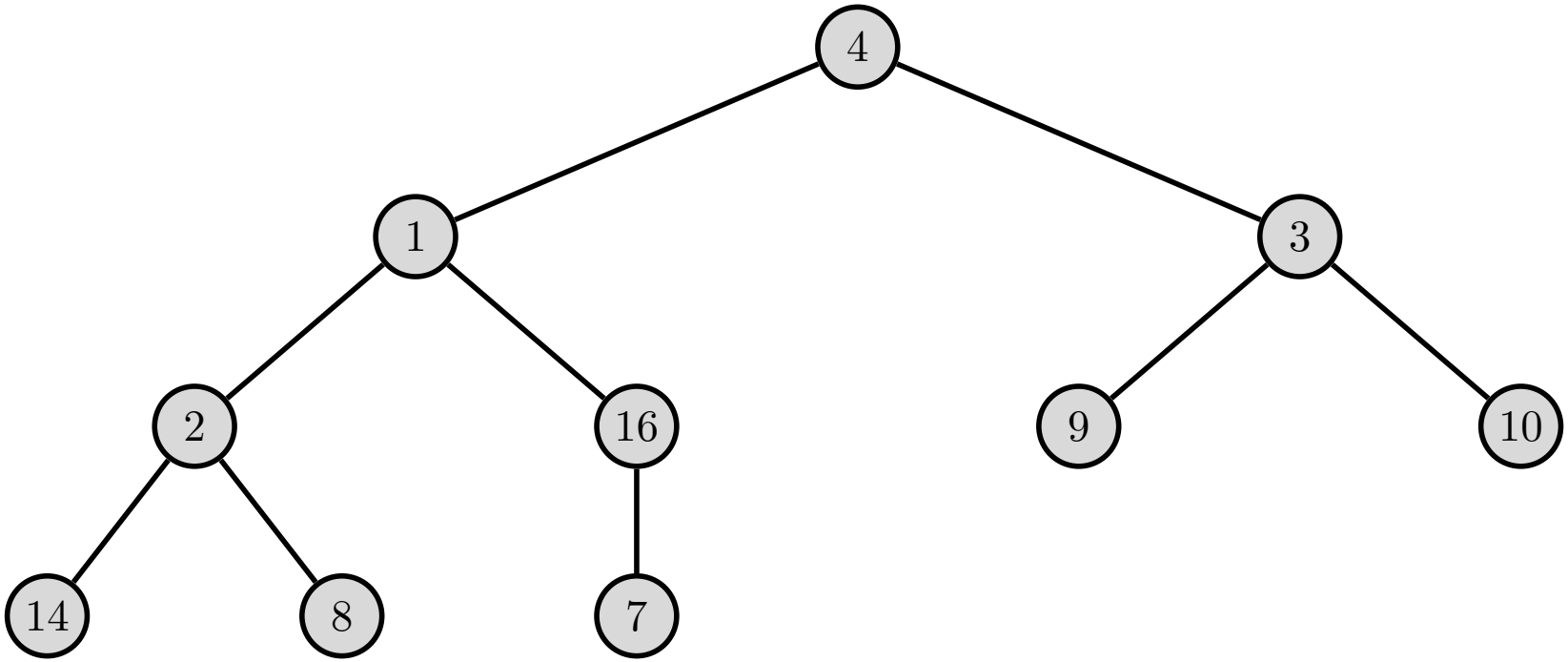
```
SIFT-DOWN(H, i)
 1  if LEFT(i) > num(H)
 2        return
 3  elseif LEFT(i) = num(H) and H[i] > H[LEFT(i)]
 4        SWAP(H, i, LEFT(i))
 5        SIFT-DOWN(H, LEFT(i))
 6  else
 7        if H[LEFT(i)] ≥ H[RIGHT(i)]
 8              m = LEFT(i)
 9        else m = RIGHT(i)
10        if H[LEFT(i)] > H[m]
11              SWAP(H, i, LEFT(i))
12              SIFT-DOWN(H, LEFT(i))
```

- Complexidade no pior caso
  - Uma chamada por nível da árvore: $\Theta(\log Num(H))$

PPgSC

Heap para listas de prioridade

Construção

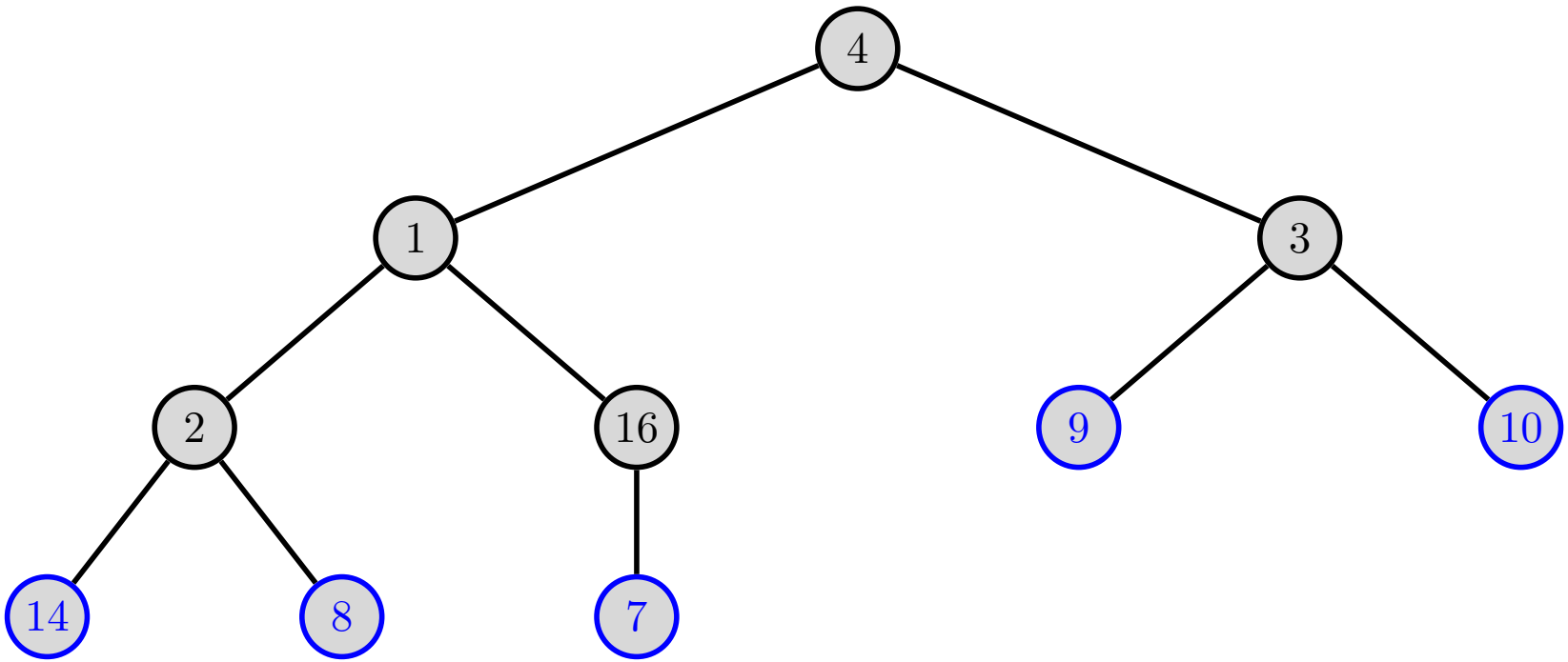


- Considerando uma sequência de valores em uma ordem qualquer, como construir um *heap*?
- Se os valores estão em um arranjo, a propriedade estrutural é inicialmente satisfeita.

PPgSC

Heap para listas de prioridade

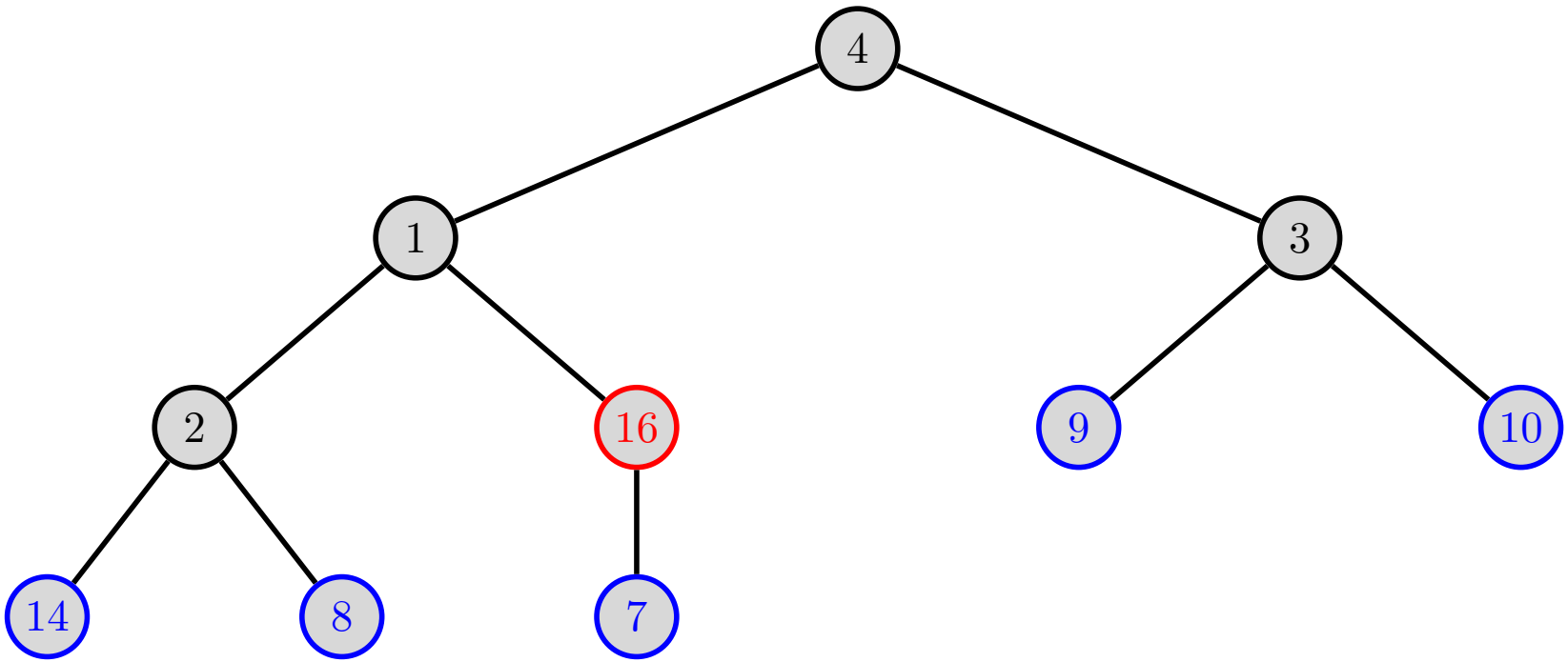
Construção



- Inicialmente, as folhas formam sub-árvores que são *heaps*.

Heap para listas de prioridade

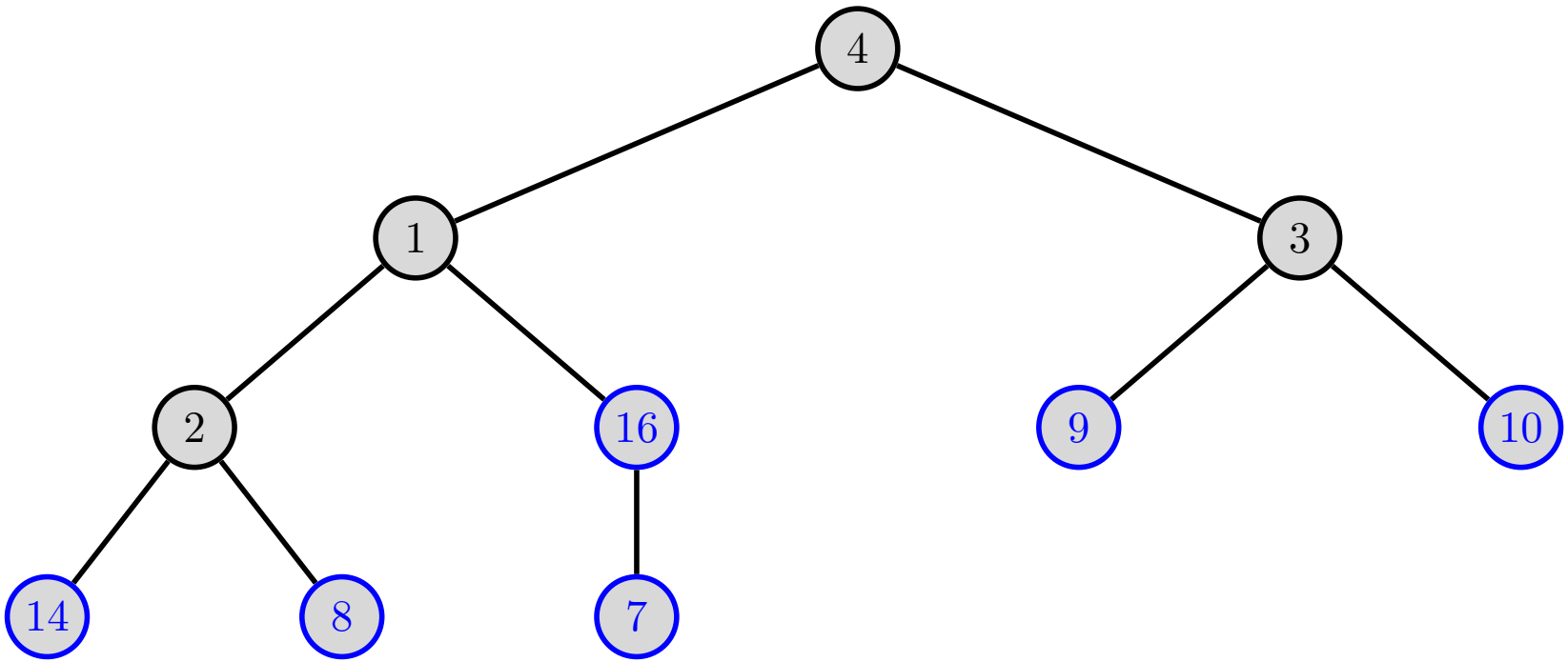
Construção



- Cada nó interno cujas sub-árvores são *heaps* é analizado.

Heap para listas de prioridade

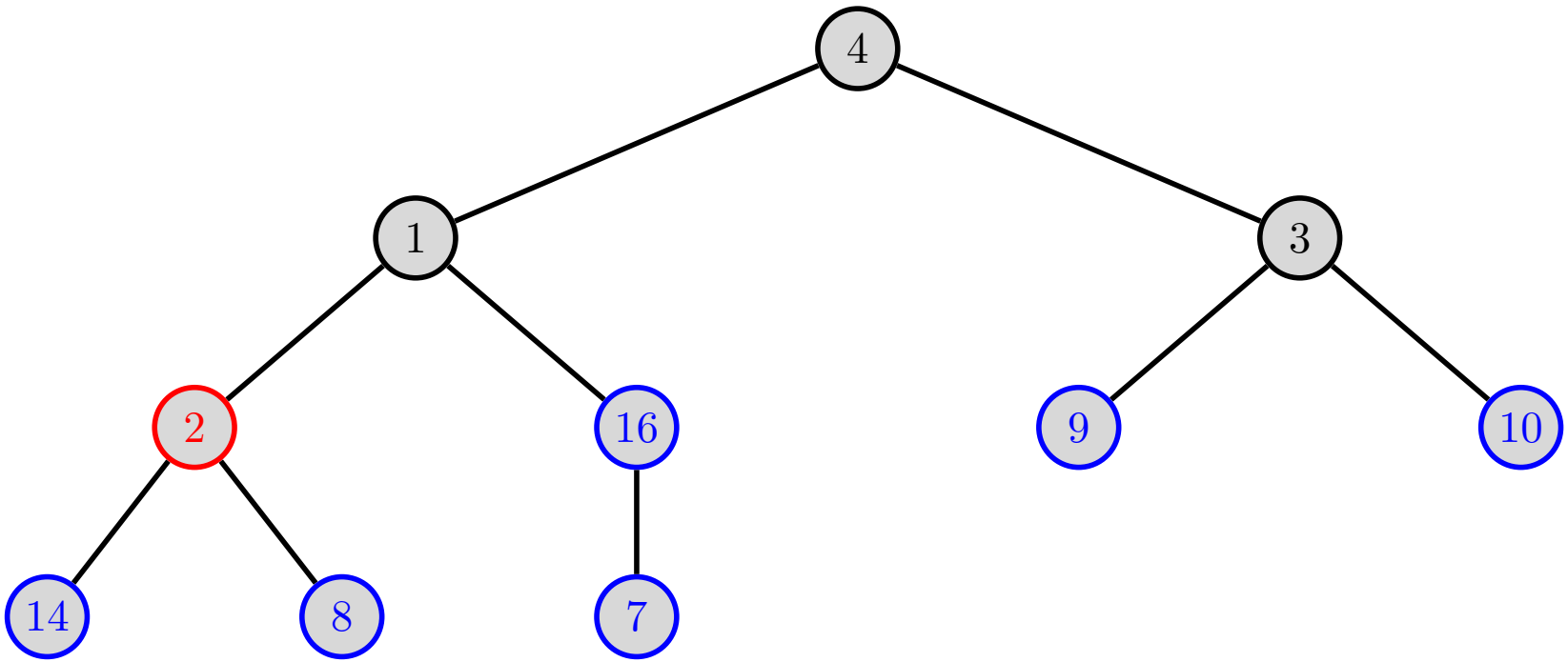
Construção



- Se é maior que os filhos, nada é feito.

Heap para listas de prioridade

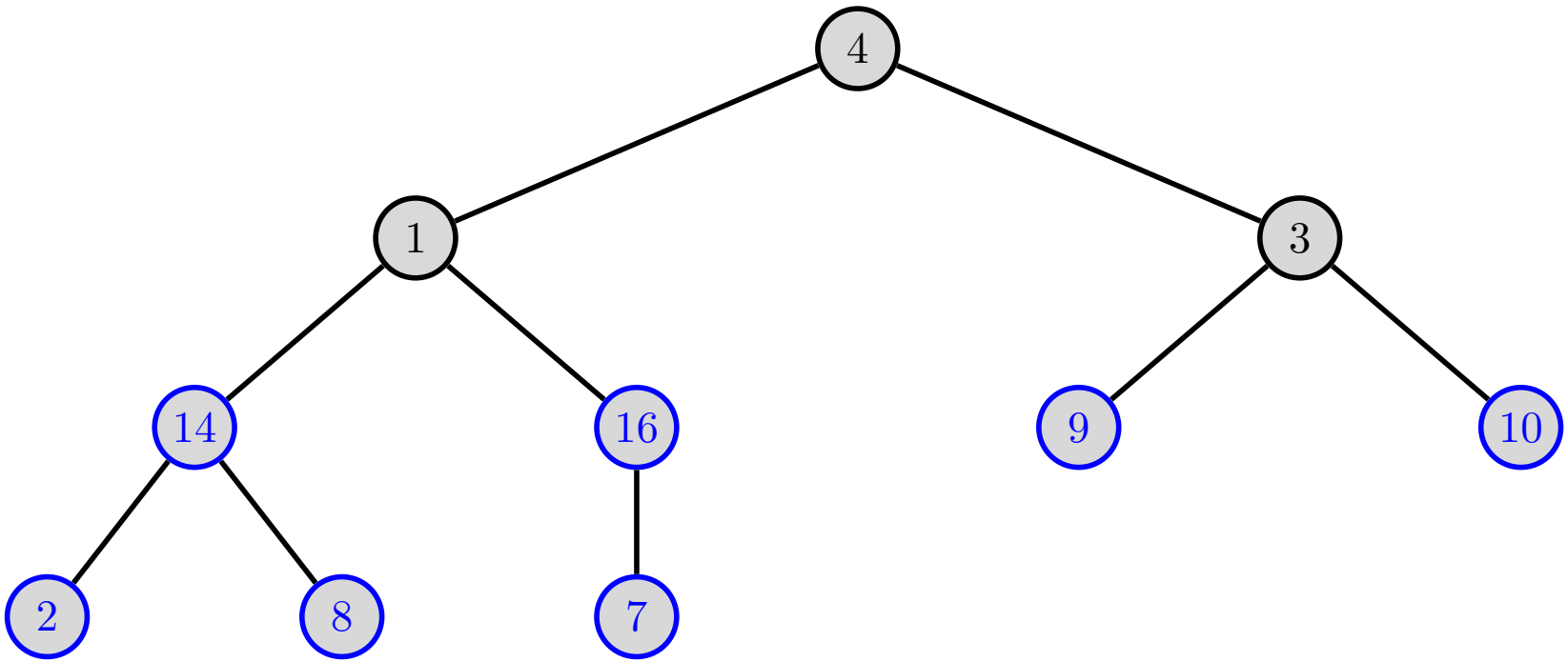
Construção



- Senão, a sub-árvore é corrigida aplicando SIFT-DOWN.

Heap para listas de prioridade

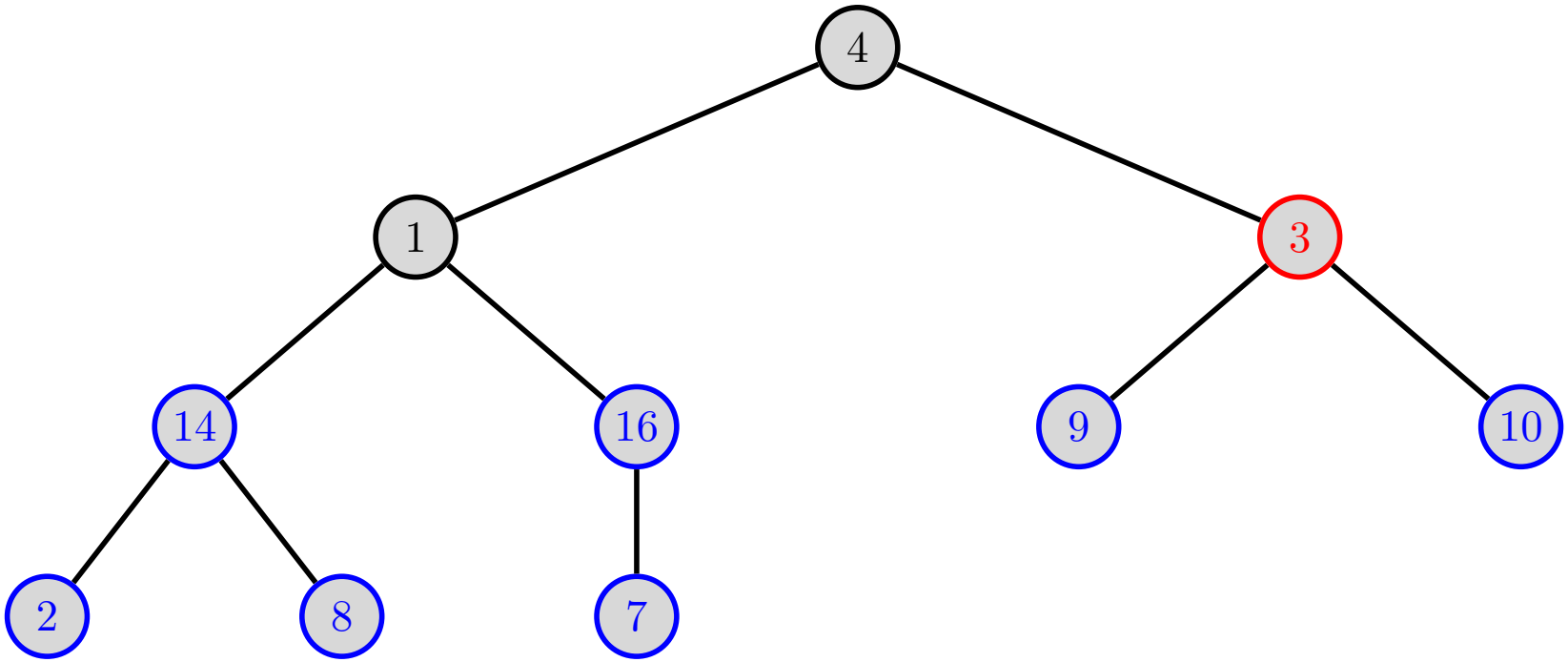
Construção



- A sub-árvore passa a ser um *heap*.

Heap para listas de prioridade

Construção

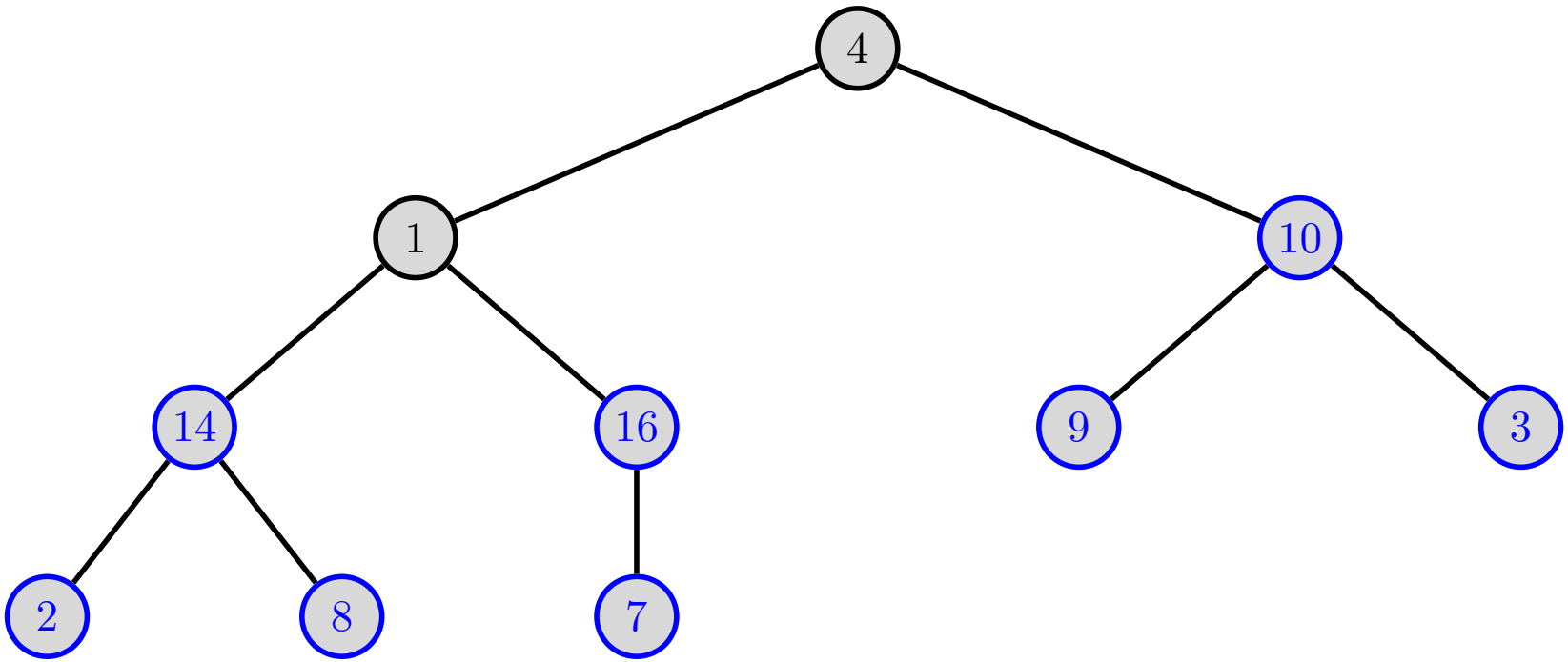


- E assim sucessivamente...

PPgSC

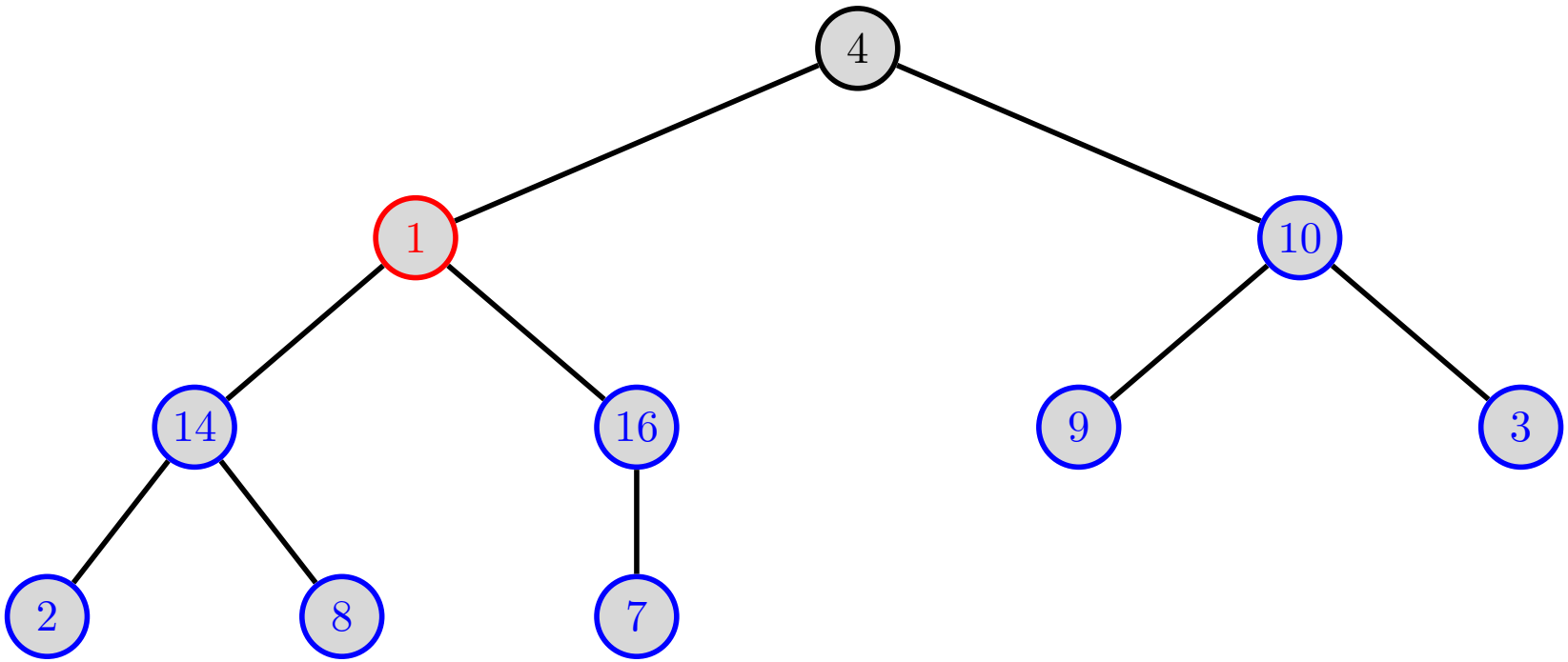
Heap para listas de prioridade

Construção



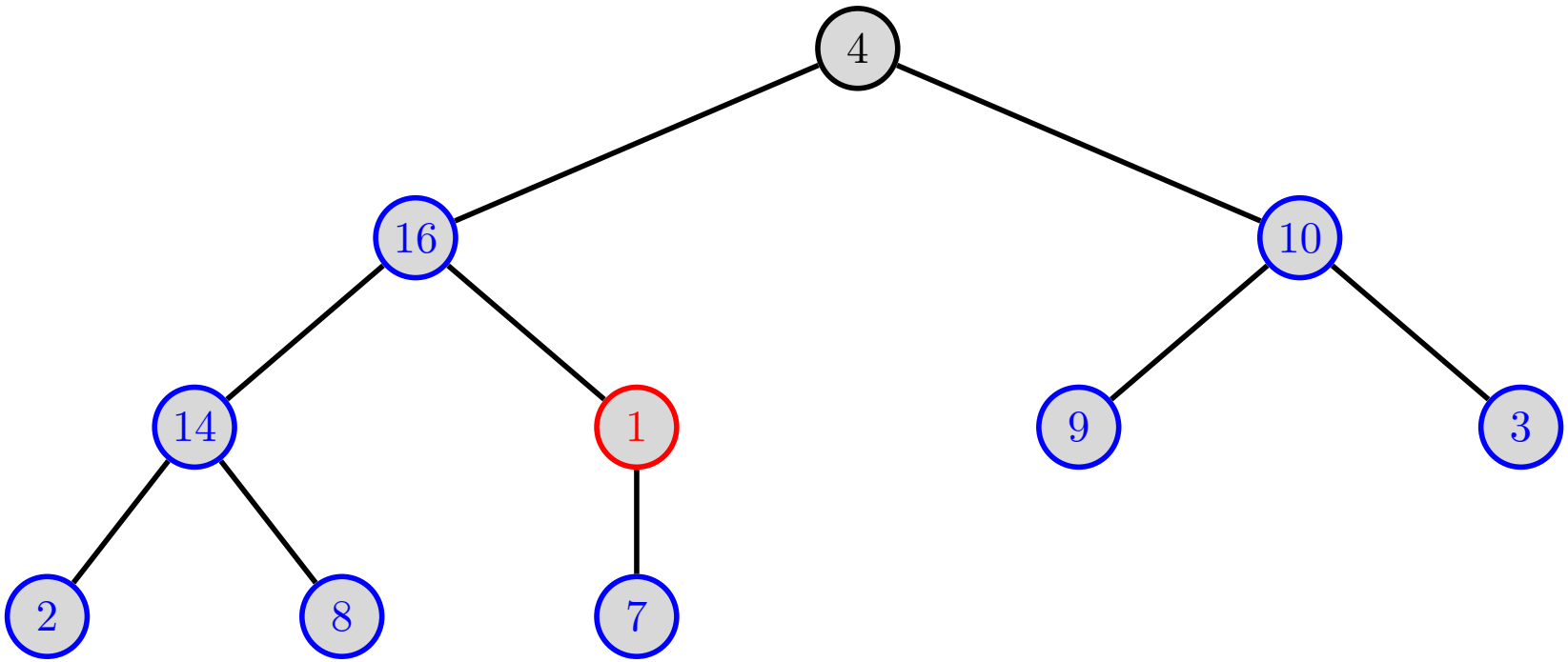
Heap para listas de prioridade

Construção



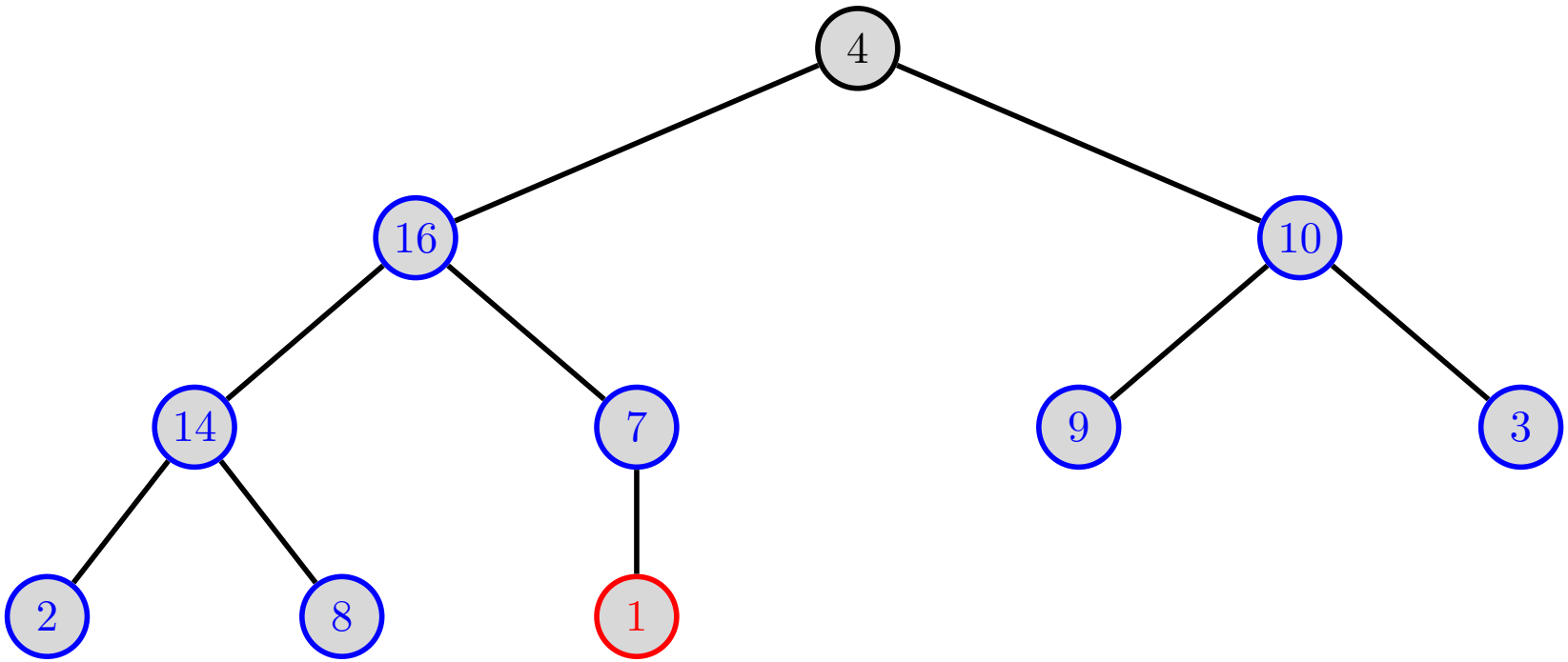
Heap para listas de prioridade

Construção



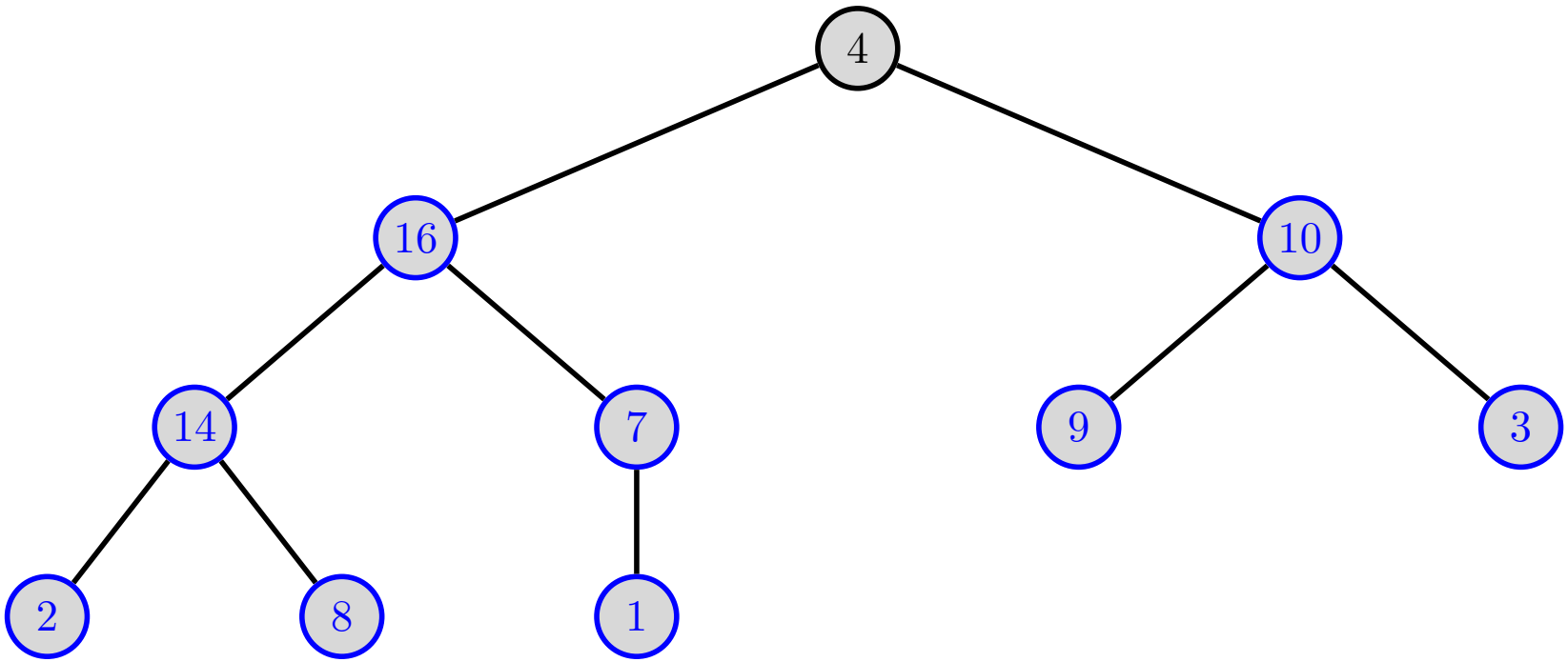
Heap para listas de prioridade

Construção



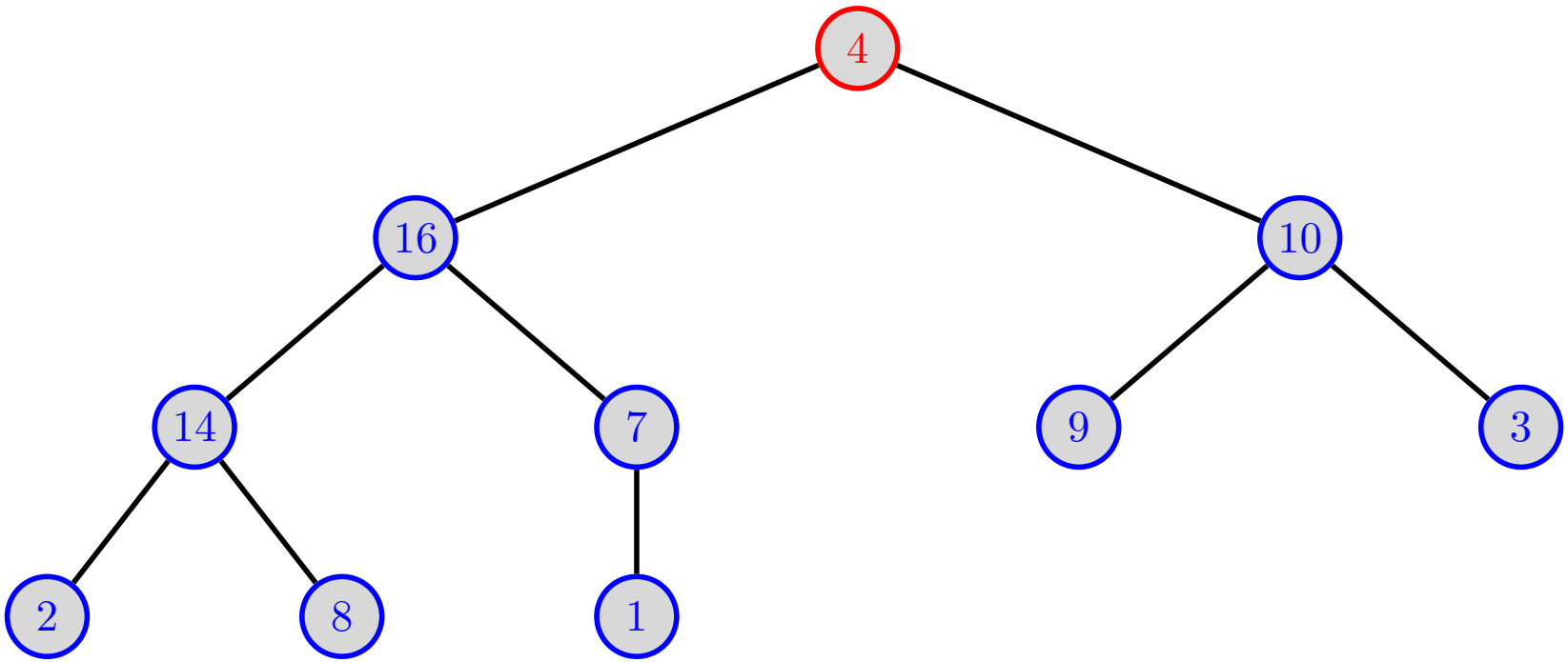
Heap para listas de prioridade

Construção



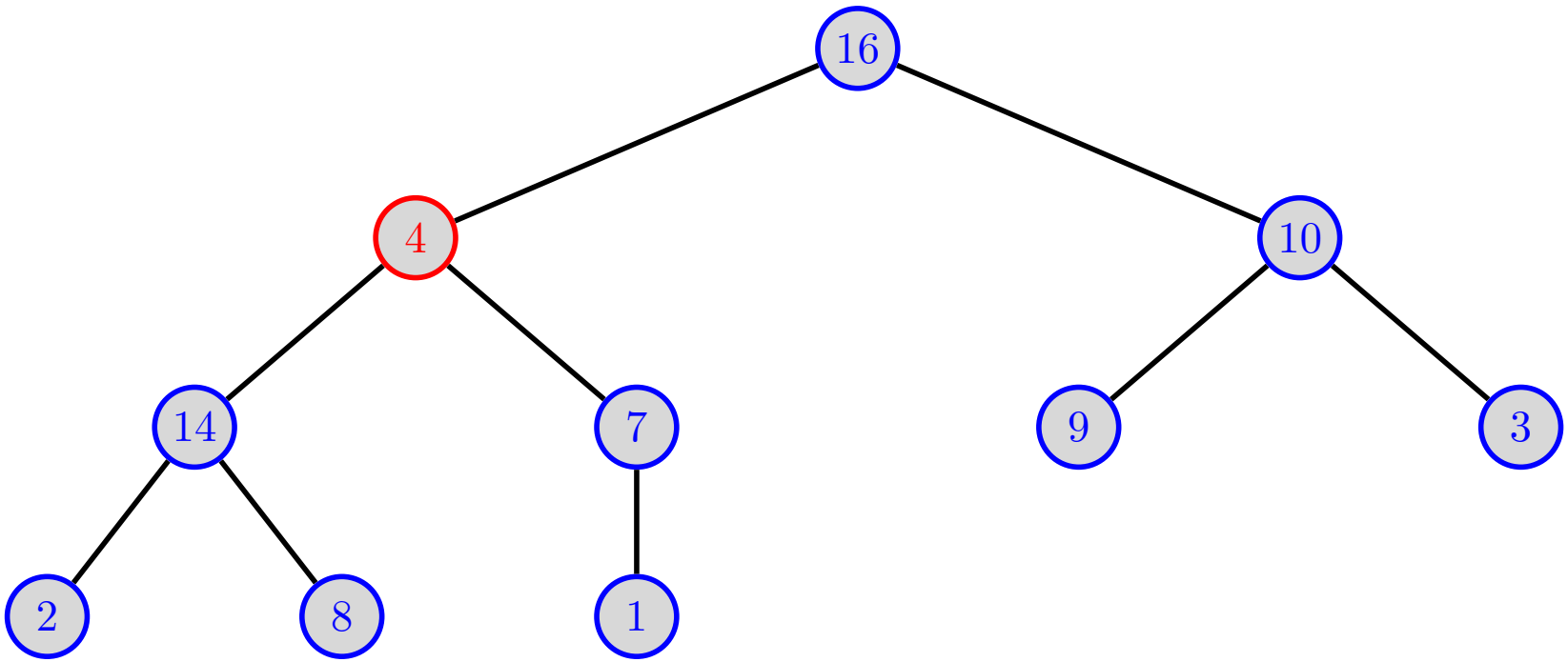
Heap para listas de prioridade

Construção



Heap para listas de prioridade

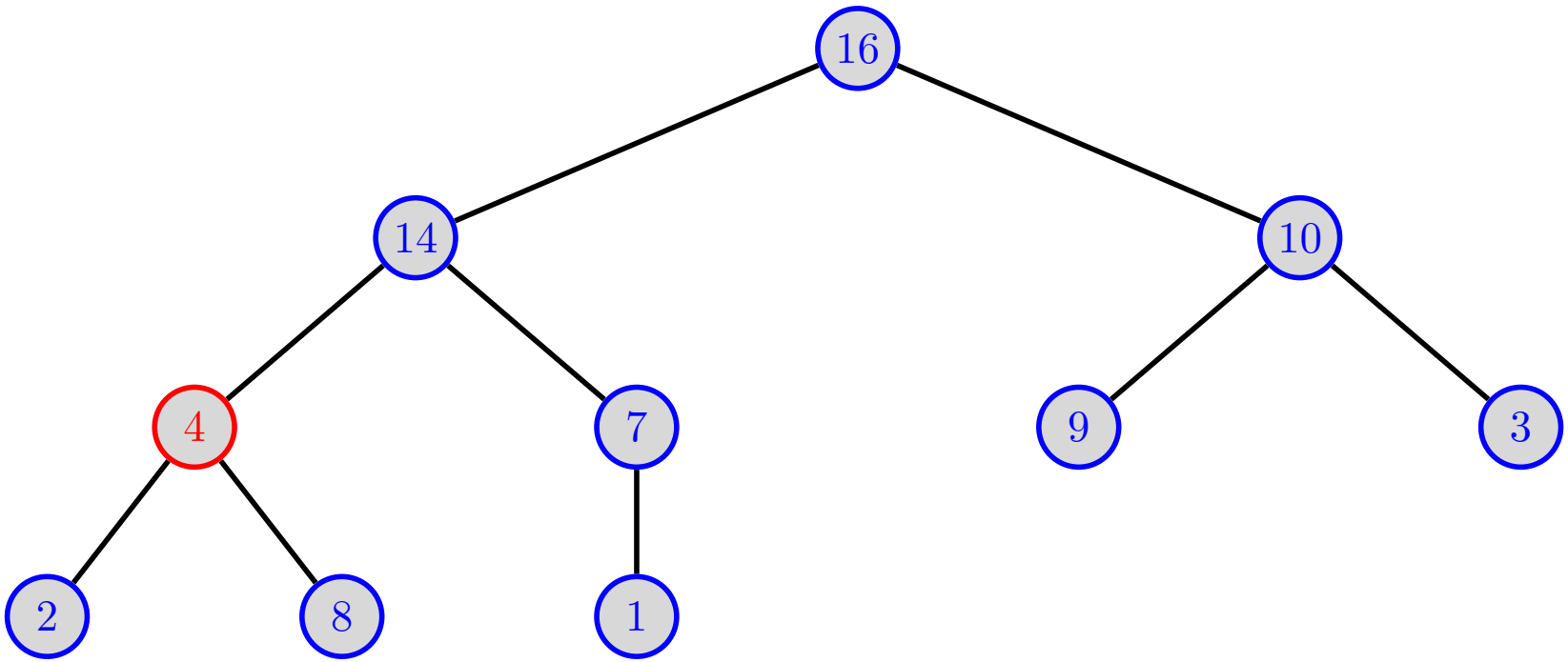
Construção



PPgSC

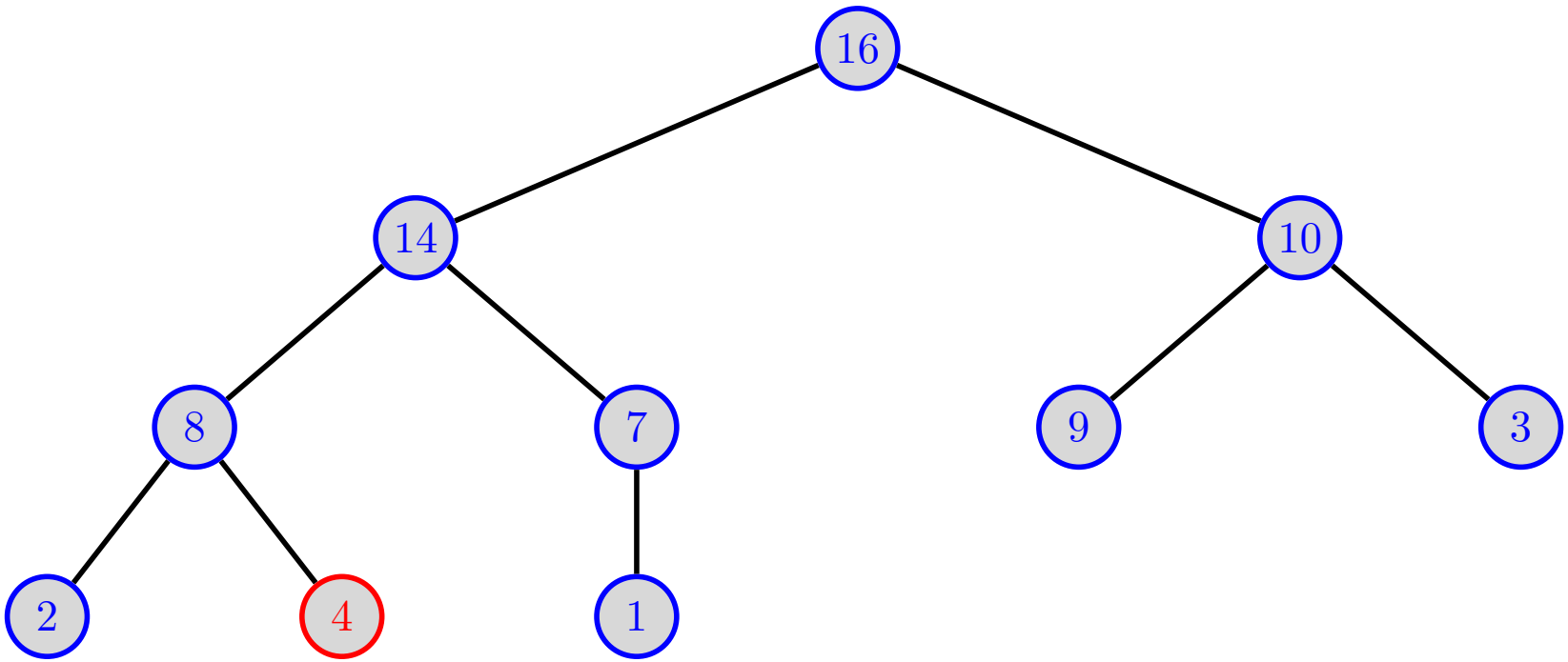
Heap para listas de prioridade

Construção



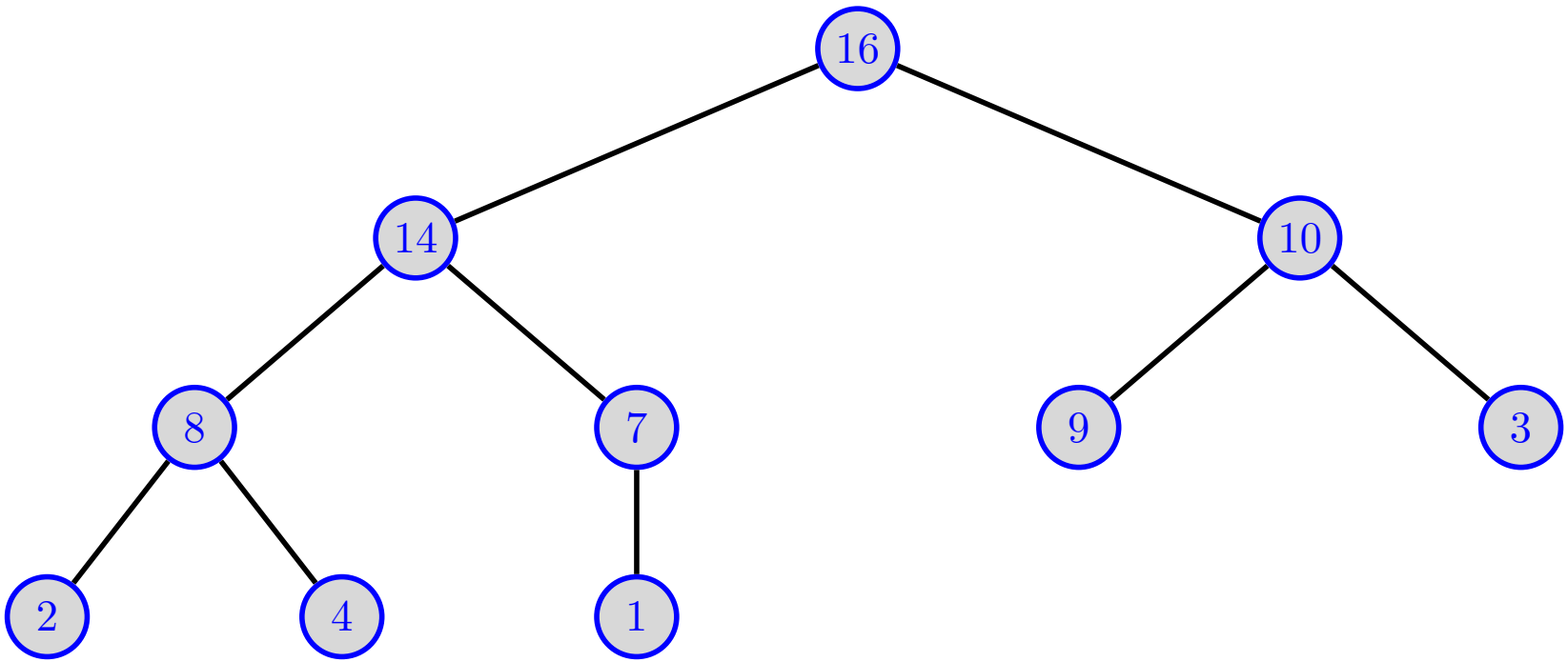
Heap para listas de prioridade

Construção



Heap para listas de prioridade

Construção



- Finalmente, o arranjo forma um *heap*.

Heap para listas de prioridade

Algoritmo de construção

BUILD(H)

1 **for** $i = num(H)/2$ **downto** 1
2 SIFT-DOWN(H, i)

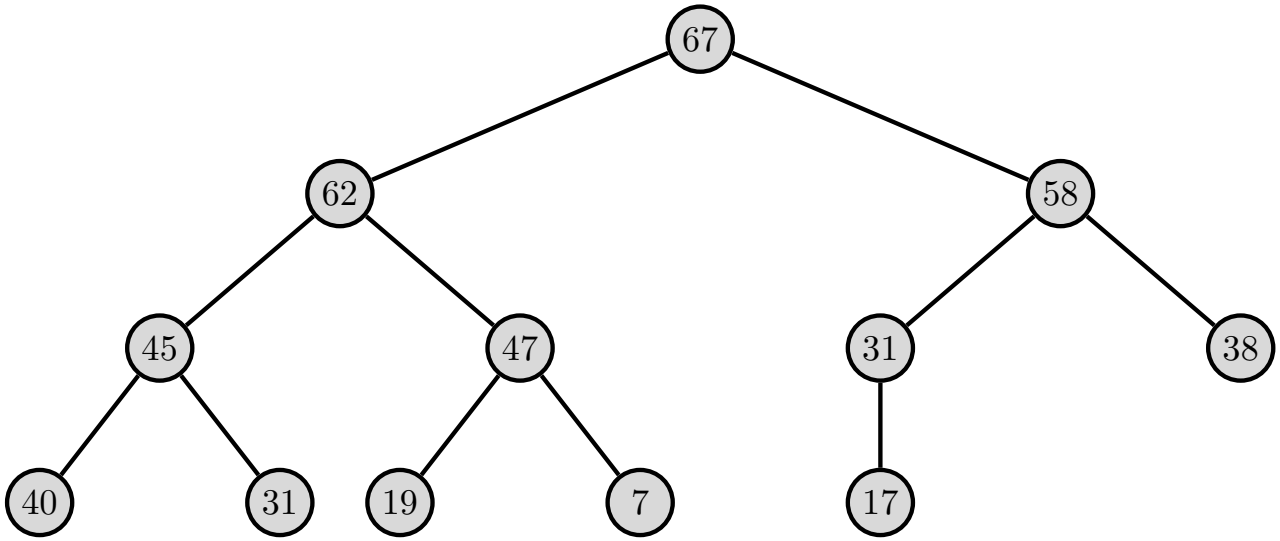
- Complexidade
  - n nós,
  - Há, no máximo, $n/2^{h+1}$ nós na altura h,
  - O custo na altura h é $O(h)$,
  - $$\begin{aligned} T(n) &= \sum_{h=0}^{\lfloor \lg n \rfloor} \lceil n/2^{h+1} \rceil O(h) \\ &= O(n \sum_{h=0}^{\lfloor \lg n \rfloor} h/2^h) \\ &= O(n \sum_{h=0}^{\infty} h/2^h) \end{aligned}$$
  - $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$ $(|x| < 1)$
  - $T(n) = O(n)$

Listas de prioridade e *heaps*

Síntese da complexidade no pior caso

1. Obter elemento de maior prioridade $\Theta(1)$
2. Inserir um novo elemento $\Theta(\lg n)$
3. Retirar o elemento de maior prioridade $\Theta(\lg n)$
4. Construir a lista a partir de uma sequência qualquer inicial $\Theta(n)$

Ordenação e *heaps*



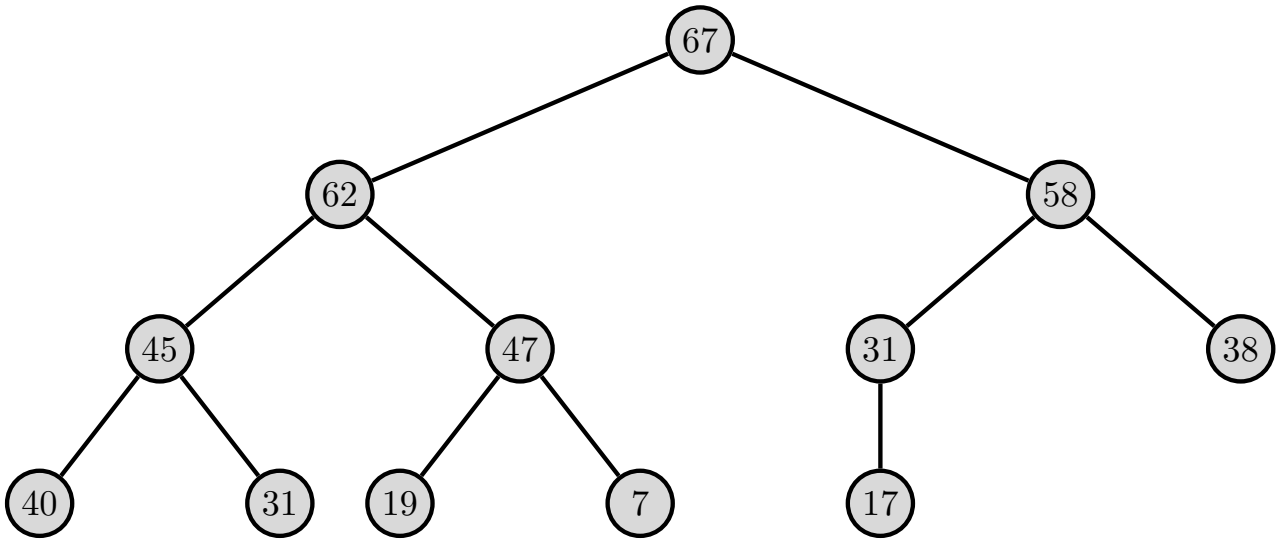
67	62	58	45	47	31	38	40	31	19	7	17	···	···	···	···
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- Como utilizar o conceito de *heap* para ordenar o arranjo em ordem crescente?

PPgSC

Ordenação e *heaps*

Princípios

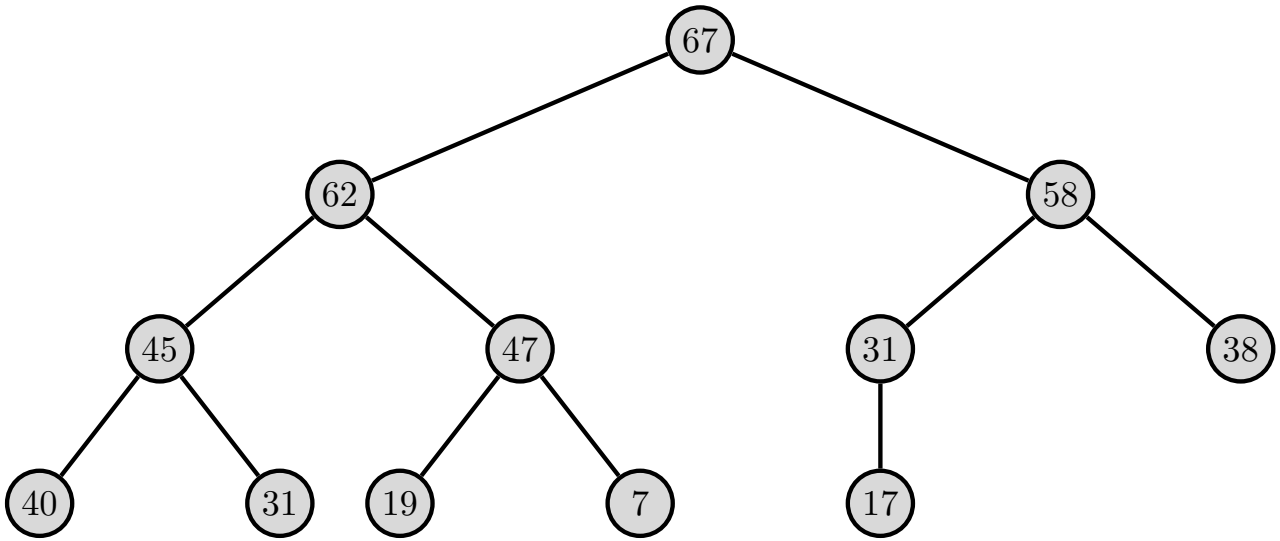


- O maior valor está na posição 1 do *heap*.
- Deve ficar na última posição do arranjo ordenado.

PPgSC

Ordenação e *heaps*

Princípios



- O maior valor está na posição 1 do *heap*.
- Deve ficar na última posição do arranjo ordenado.
- Inverter o conteúdo das posições 1 e $num(H)$.
- Decrementar $num(H)$.
- Reestabelecer a propriedade de ordenação aplicando SIFT-DOWN à posição 1.
- Repetir enquanto $num(H) > 2$.

PPgSC

- Há $\Theta(n)$ chamadas a SIFT-DOWN
- Cada chamada custa $O(\lg n)$.
- Logo o custo total é $O(n \lg n)$