

Estruturas de Dados II

Filas de prioridades

Prof. Rodrigo Minetto

Prof^a. Juliana de Santi

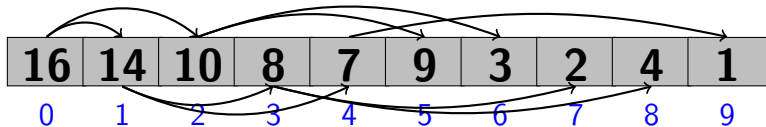
Universidade Tecnológica Federal do Paraná

Sumário

- 1 Heap
- 2 Filas de Prioridades
- 3 Operação Heap-Maximum
- 4 Operação Heap-Extract-Max
- 5 Operação Heap-Increase-Key
- 6 Operação Max-Heap-Insert

Relembrando: a estrutura de dados **heap** (**binário**) é um arranjo que pode ser visualizado como uma árvore binária praticamente completa (a exceção ocorre no último nível que é preenchido da esquerda para direita enquanto existirem elementos). Cada nó da árvore corresponde a um elemento do arranjo.

Heap



PAI (*i*)

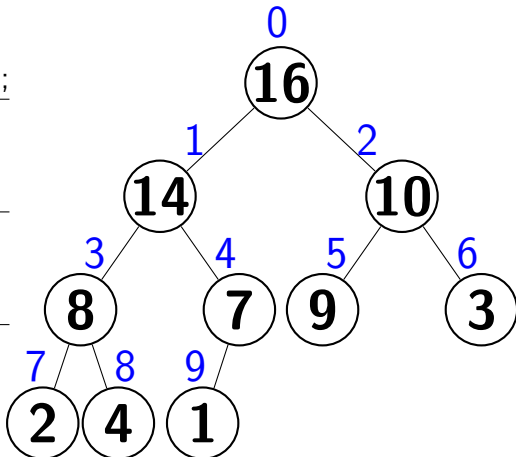
Return $\lfloor (i - 1) / 2 \rfloor$;

ESQUERDA (*i*)

Return $i * 2 + 1$;

DIREITA (*i*)

Return $i * 2 + 2$;



Heap

A raiz da árvore para um vetor V sempre se encontra no elemento $V[0]$. Existem dois tipos de heaps: **heaps máximos** e **heaps mínimos**:

$$V[\text{PAI}(i)] \geq V[i] \quad (\text{máximo})$$

$$V[\text{PAI}(i)] \leq V[i] \quad (\text{mínimo})$$

Em um heap máximo o maior valor está armazenado na raiz (em um heap mínimo o menor). O algoritmo *heap-sort* usa um *heap-máximo*.

Sumário

- 1 Heap
- 2 Filas de Prioridades**
- 3 Operação Heap-Maximum
- 4 Operação Heap-Extract-Max
- 5 Operação Heap-Increase-Key
- 6 Operação Max-Heap-Insert

Filas de prioridades

O tipo abstrato de dados **heap** tem uma utilidade enorme: seu uso como uma **fila de prioridades**. Os elementos de um vetor V são chamados de **chaves** ou **prioridades**, podendo existir outros dados associados a cada elemento. As operações básicas em uma fila de prioridade são: **encontrar** o elemento máximo (ou mínimo), **inserir**, **extrair** ou **modificar** uma chave.

Filas de prioridades

Uma **aplicação** comum **de filas de prioridade máxima** é o seu uso por **sistemas operacionais** para escalonar processos em uma CPU compartilhada. Quando um processo é finalizado ou interrompido, o processo de prioridade mais alta é selecionado para executar dentre àqueles processos pendentes. Filas de prioridade também são usadas em algoritmos famosos na computação como os algoritmos de Dijkstra, Prim e Huffman.

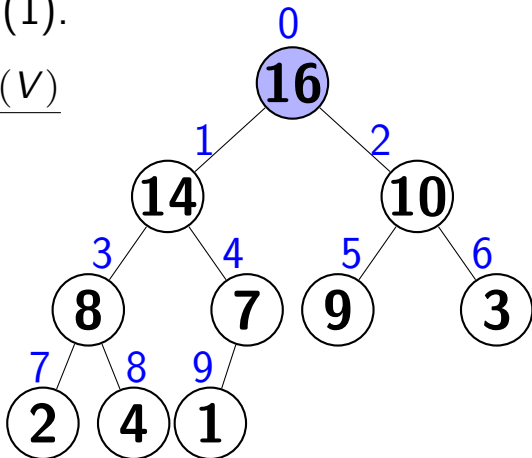
Sumário

- 1 Heap
- 2 Filas de Prioridades
- 3 Operação Heap-Maximum**
- 4 Operação Heap-Extract-Max
- 5 Operação Heap-Increase-Key
- 6 Operação Max-Heap-Insert

Heap-Maximum

A operação Heap-Maximum (V) **retorna** o elemento de V com **maior chave (prioridade)**.
Complexidade: $\Theta(1)$.

HEAP-MAXIMUM (V)
return $V[0]$;



Sumário

- 1 Heap
- 2 Filas de Prioridades
- 3 Operação Heap-Maximum
- 4 Operação Heap-Extract-Max**
- 5 Operação Heap-Increase-Key
- 6 Operação Max-Heap-Insert

Heap-Extract-Max

A operação Heap-Extract-Max (V) **remove** e **re-torna** o elemento de V com a **maior chave (prioridade)**. Ele é semelhante a extração da maior chave pelo algoritmo Heap-Sort. Note que uma vez que a maior chave é retirada é **necessário refazer o heap com** o procedimento **Max-Heapify**.

Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if SIZE < 1 **then**

printf ("erro: heap underflow");

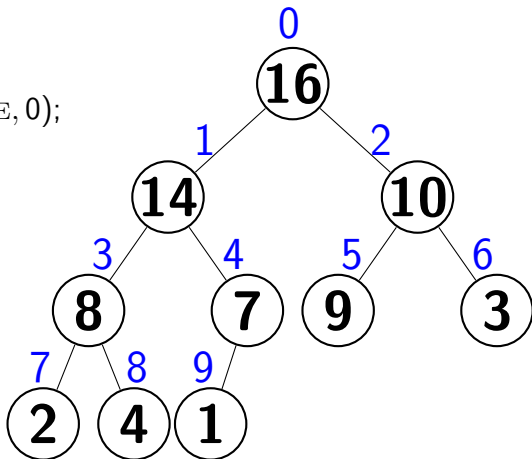
max = $V[0]$;

$V[0] = V[\text{SIZE} - 1]$;

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

▷ **if** SIZE < 1 **then**

printf ("erro: heap underflow");

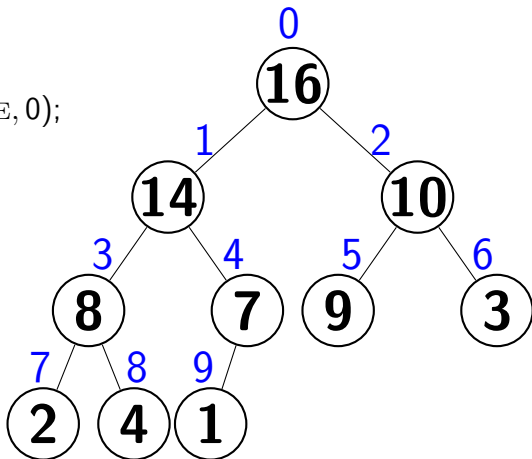
max = $V[0]$;

$V[0] = V[\text{SIZE} - 1]$;

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

▷ **if** $10 < 1$ **then**

printf ("erro: heap underflow");

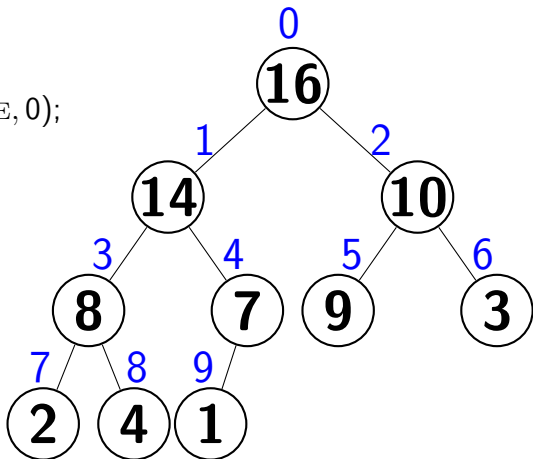
max = $V[0]$;

$V[0] = V[\text{SIZE} - 1]$;

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

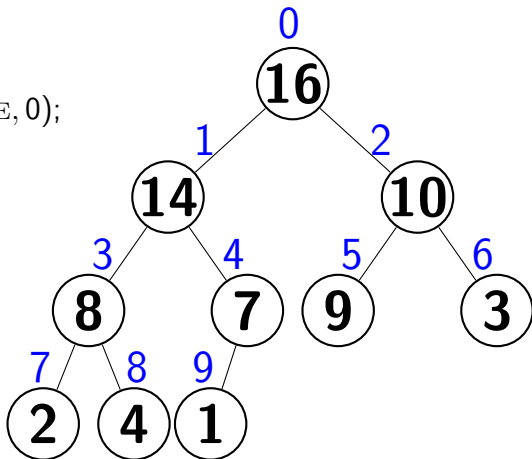
▷ $\text{max} = V[0]$;

$V[0] = V[\text{SIZE} - 1]$;

$\text{SIZE} = \text{SIZE} - 1$;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

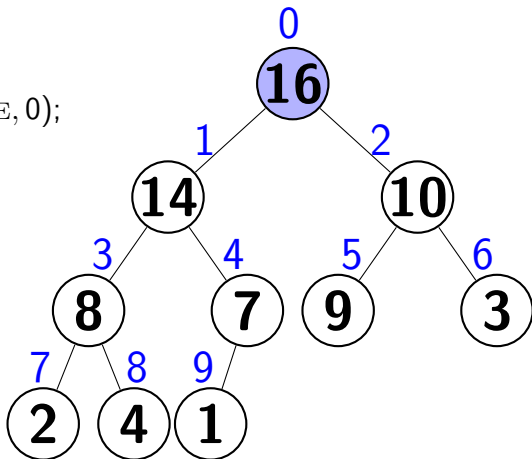
▷ max = 16;

$V[0] = V[SIZE - 1];$

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

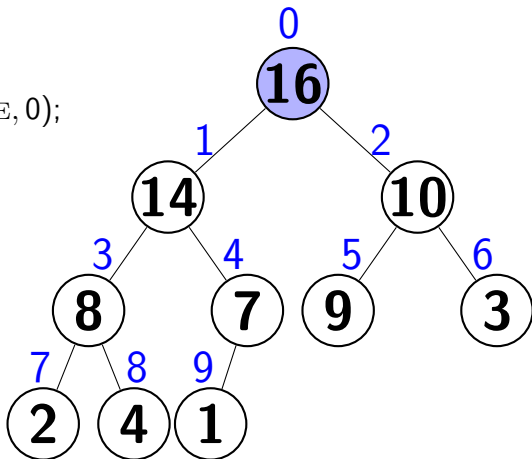
max = 16;

▷ $V[0] = V[SIZE - 1]$;

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

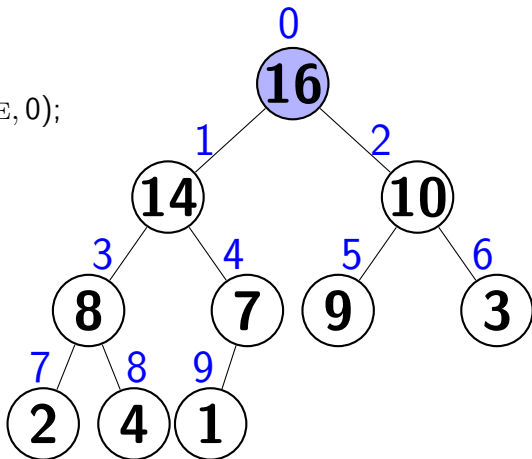
max = 16;

▷ $V[0] = V[10 - 1]$;

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

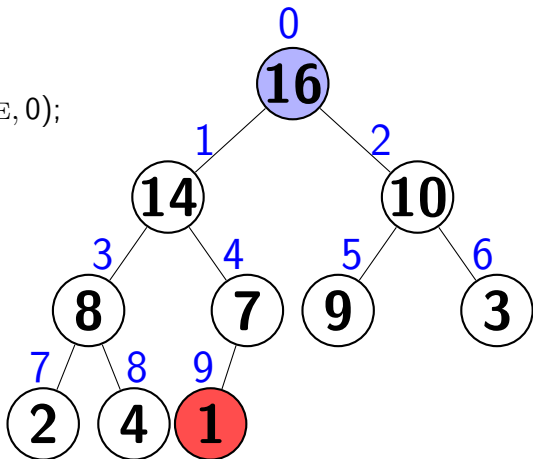
max = 16;

▷ $V[0] = V[9]$;

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

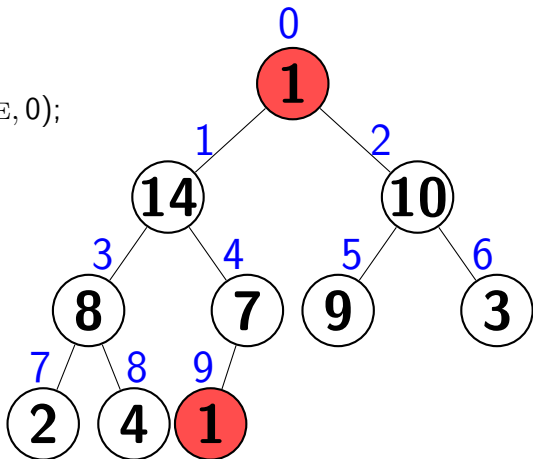
max = 16;

▷ $V[0] = V[9]$;

SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

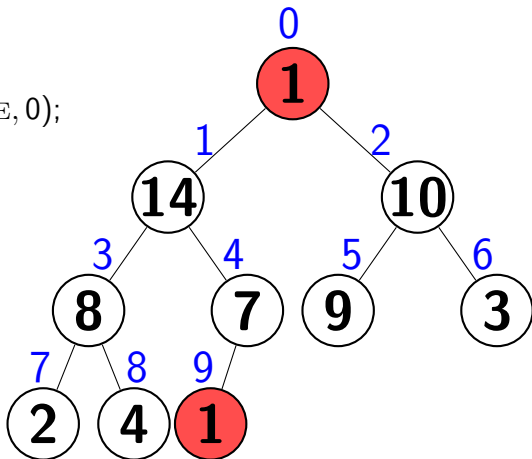
max = 16;

$V[0] = V[9]$;

▷ SIZE = SIZE - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

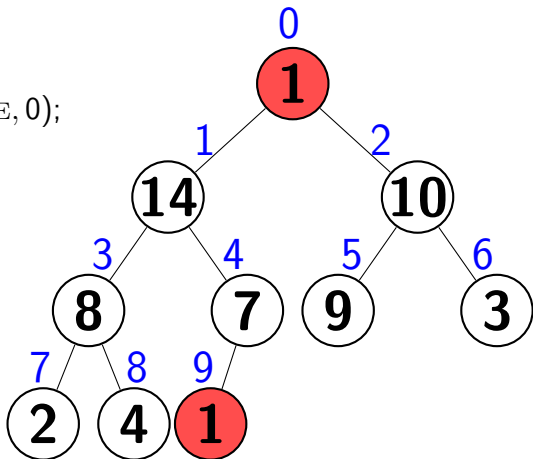
max = 16;

$V[0] = V[9]$;

▷ SIZE = 10 - 1;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

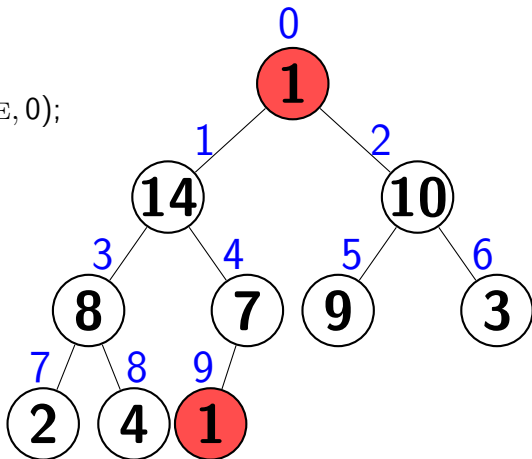
max = 16;

$V[0] = V[9]$;

▷ SIZE = 9;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

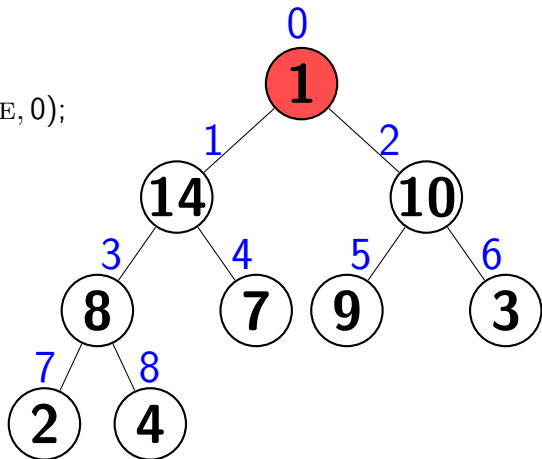
max = 16;

$V[0] = V[9]$;

▷ SIZE = 9;

MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

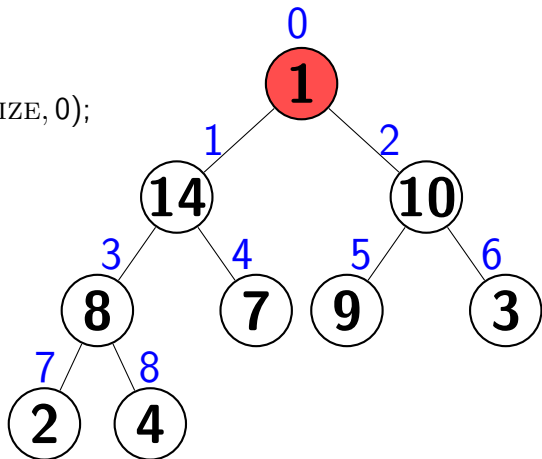
max = 16;

$V[0] = V[9]$;

SIZE = 9;

▷ MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

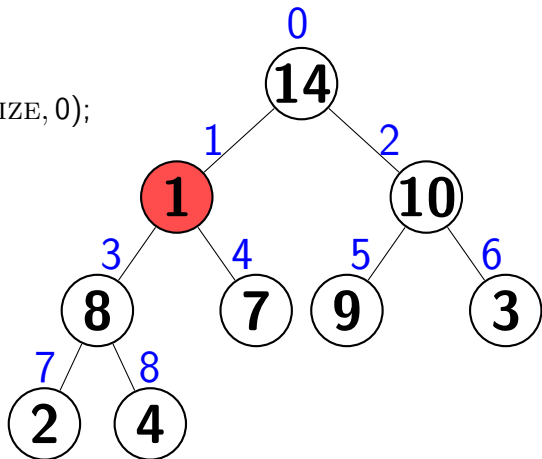
max = 16;

$V[0] = V[9]$;

SIZE = 9;

▷ MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

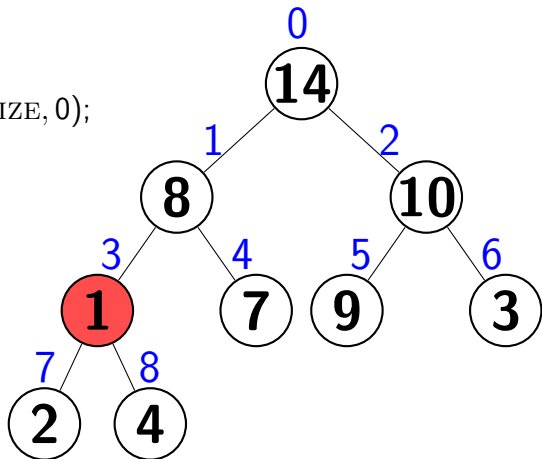
max = 16;

$V[0] = V[9]$;

SIZE = 9;

▷ MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

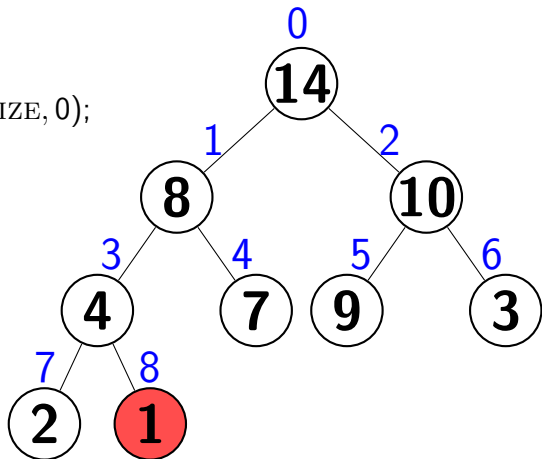
max = 16;

$V[0] = V[9]$;

SIZE = 9;

▷ MAX-HEAPIFY (V , SIZE, 0);

return max;



Heap-Extract-Max

HEAP-EXTRACT-MAX (V , SIZE)

if $10 < 1$ **then**

printf ("erro: heap underflow");

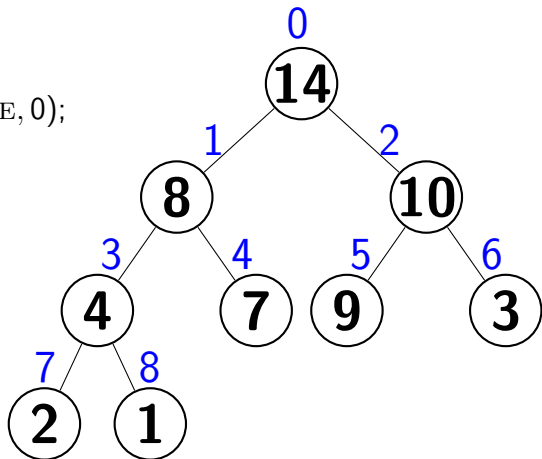
max = 16;

$V[0] = V[9]$;

SIZE = 9;

MAX-HEAPIFY (V , SIZE, 0);

▷ **return** max;



Complexidade: o procedimento Heap-Extract-Max possui complexidade $\mathcal{O}(\log n)$ para um heap com n elementos, pois o procedimento Max-Heapify possui complexidade $\mathcal{O}(\log n)$ (comprimento máximo da raiz até um nó folha). Os outros passos do procedimento Heap-Extract-Max possuem complexidade $\mathcal{O}(1)$.

Sumário

- 1 Heap
- 2 Filas de Prioridades
- 3 Operação Heap-Maximum
- 4 Operação Heap-Extract-Max
- 5 Operação Heap-Increase-Key
- 6 Operação Max-Heap-Insert

Heap-Increase-Key

A operação Heap-Increase-Key (V, x, k) **umenta** o valor da chave (prioridade) do elemento x para um novo valor k , que se presume ser pelo menos tão grande quanto o valor da chave atual x . Note que essa operação **pode violar** a propriedade de **heap máximo**, assim, é necessário encontrar a posição apropriada para o elemento recém atualizado.

Heap-Increase-Key

HEAP-INCREASE-KEY (V , i , **chave**)

if **chave** $< V[i]$ **then**

printf ("erro: chave menor que atual");

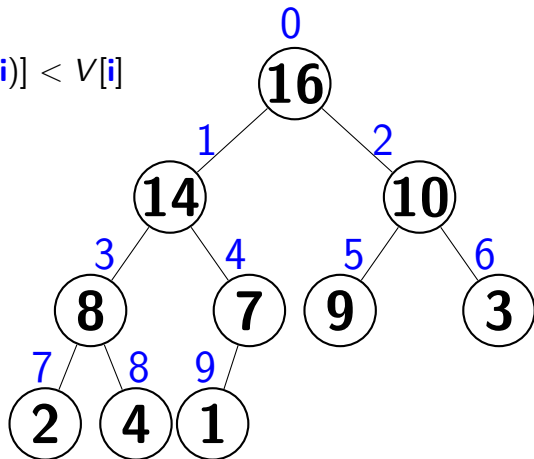
return;

$V[i] = \text{chave}$;

while $i > 0$ **and** $V[\text{PAI}(i)] < V[i]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , **8**, **15**)

if **chave** < $V[i]$ **then**

printf ("erro: chave menor que atual");

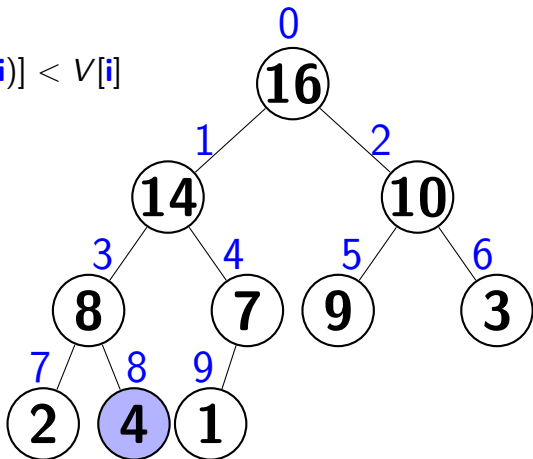
return;

$V[i]$ = **chave**;

while $i > 0$ **and** $V[\text{PAI}(i)] < V[i]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

▷ if 15 < 4 then

 printf ("erro: chave menor que atual");

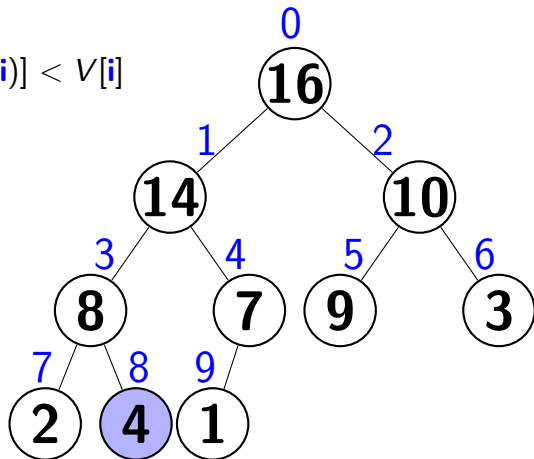
 return;

$V[i]$ = chave;

while $i > 0$ and $V[\text{PAI}(i)] < V[i]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

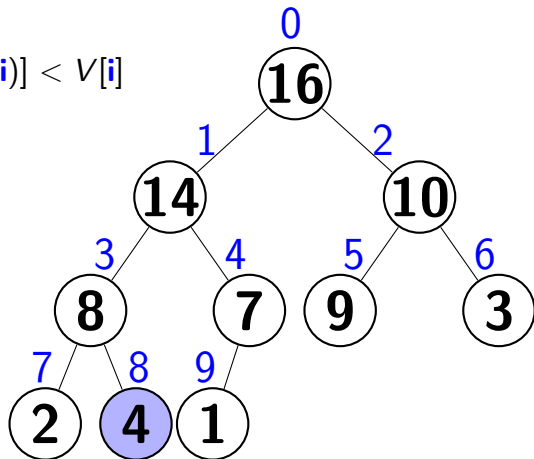
 return;

▷ $V[i] = \text{chave}$;

while $i > 0$ and $V[\text{PAI}(i)] < V[i]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

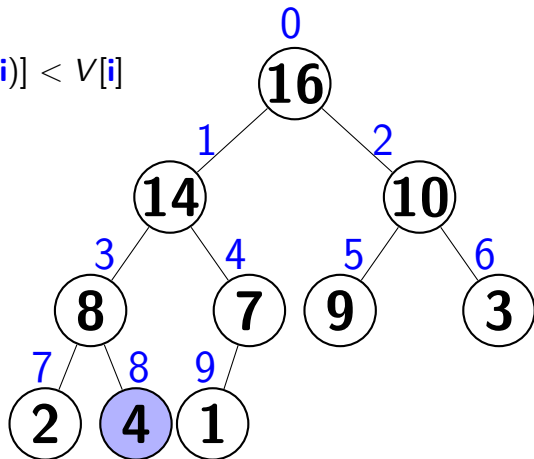
 return;

▷ $V[8] = 15$;

while $i > 0$ and $V[\text{PAI}(i)] < V[i]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

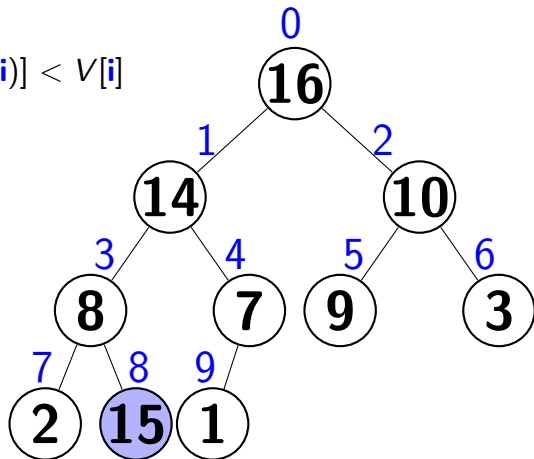
 return;

▷ $V[8] = 15$;

while $i > 0$ and $V[\text{PAI}(i)] < V[i]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , **8**, **15**)

if **15** < **4** **then**

printf ("erro: chave menor que atual");

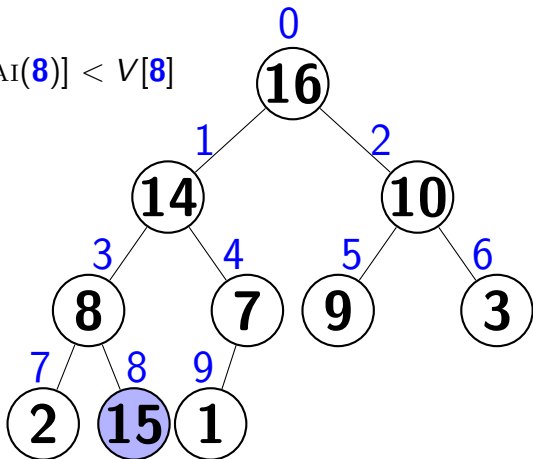
return;

$V[\mathbf{8}] = \mathbf{15}$;

 ▷ **while** **8** > 0 **and** $V[\text{PAI}(\mathbf{8})] < V[\mathbf{8}]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , **8**, **15**)

if **15** < **4** **then**

printf ("erro: chave menor que atual");

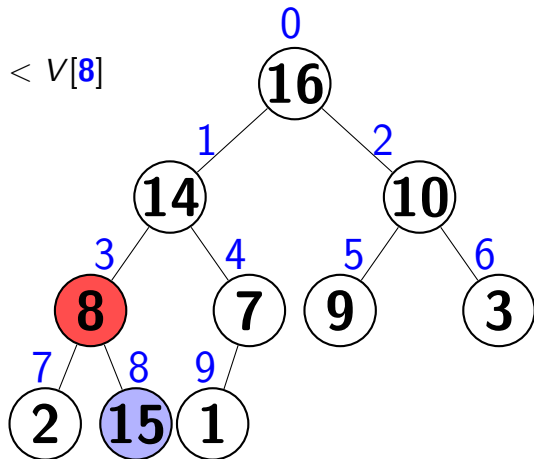
return;

$V[\mathbf{8}] = \mathbf{15}$;

 ▷ **while** **8** > 0 **and** $V[\mathbf{3}] < V[\mathbf{8}]$

$V[i] \leftrightarrow V[\text{PAI}(i)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , **8**, **15**)

if **15** < **4** **then**

printf ("erro: chave menor que atual");

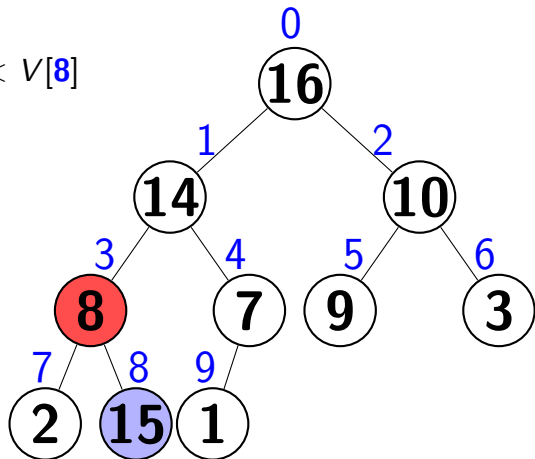
return;

$V[8] = 15$;

while **8** > 0 **and** $V[3] < V[8]$

$\triangleright V[8] \leftrightarrow V[\text{PAI}(8)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

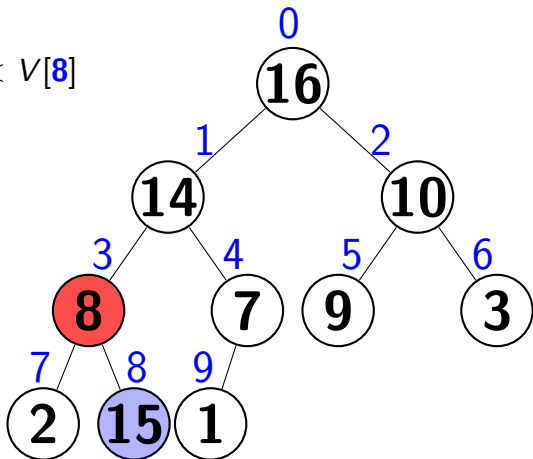
 return;

$V[8] = 15$;

while 8 > 0 and $V[3] < V[8]$

$\triangleright V[8] \leftrightarrow V[3]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

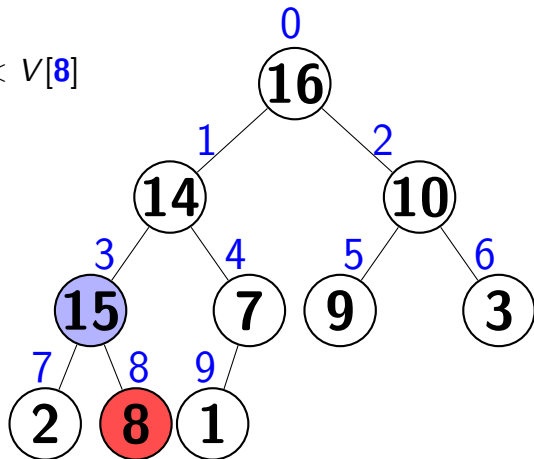
 return;

$V[8] = 15$;

while 8 > 0 and $V[3] < V[8]$

$\triangleright V[8] \leftrightarrow V[3]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V, 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

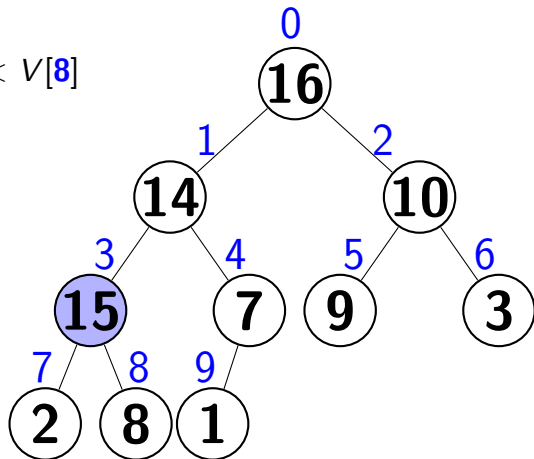
 return;

V[8] = 15;

while 8 > 0 and V[3] < V[8]

 V[8] \leftrightarrow V[3];

\triangleright i = PAI(i);



Heap-Increase-Key

HEAP-INCREASE-KEY (V, 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

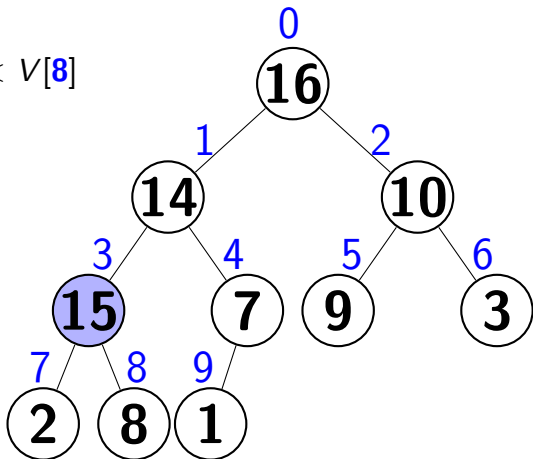
 return;

V[8] = 15;

while 8 > 0 and V[3] < V[8]

 V[8] \leftrightarrow V[3];

\triangleright 3 = PAI(8);



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

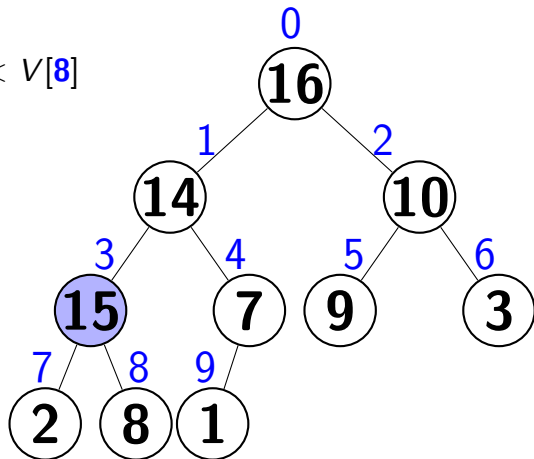
 return;

$V[8] = 15$;

while 8 > 0 and $V[3] < V[8]$

$V[8] \leftrightarrow V[3]$;

$\triangleright i = 3$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

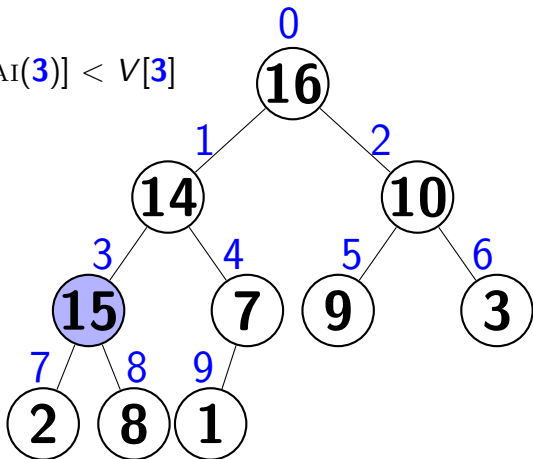
 return;

$V[8] = 15$;

▷ while 3 > 0 and $V[\text{PAI}(3)] < V[3]$

$V[i] \leftrightarrow V[\text{PAI}(3)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

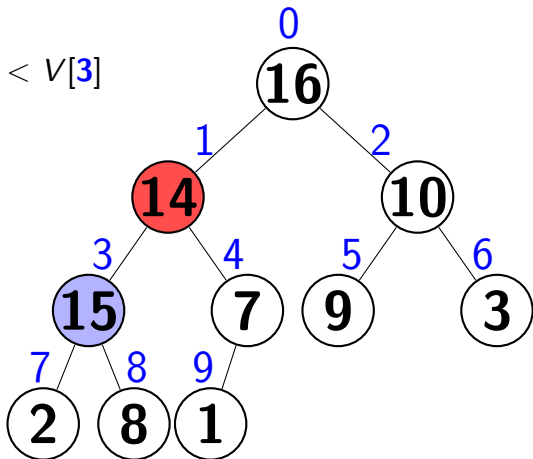
 return;

$V[8] = 15$;

▷ while 3 > 0 and $V[1] < V[3]$

$V[i] \leftrightarrow V[\text{PAI}(3)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

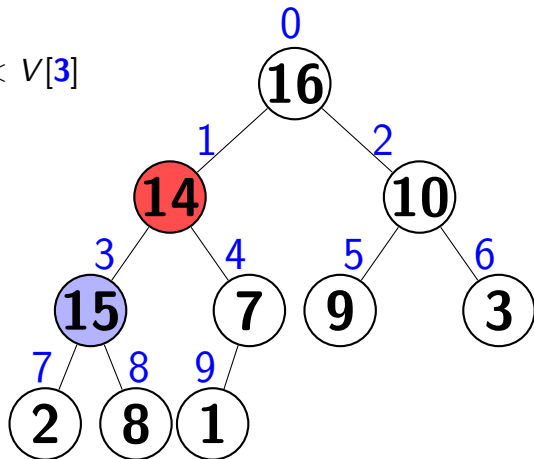
 return;

$V[8] = 15$;

while 3 > 0 and $V[1] < V[3]$

$\triangleright V[i] \leftrightarrow V[\text{PAI}(3)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

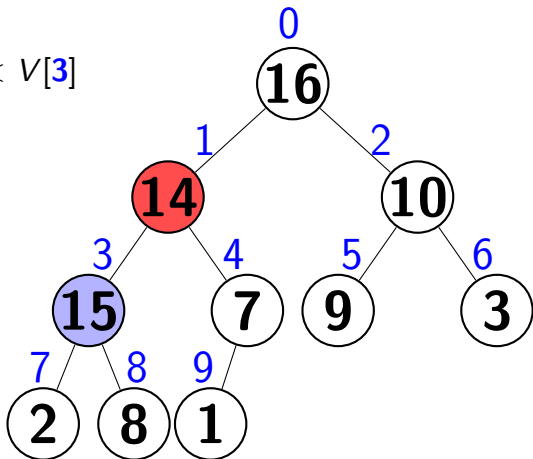
 return;

$V[8] = 15$;

while 3 > 0 and $V[1] < V[3]$

$\triangleright V[3] \leftrightarrow V[1]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

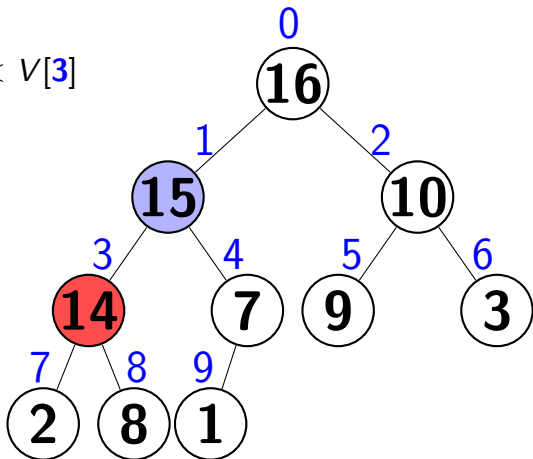
 return;

$V[8] = 15$;

while 3 > 0 and $V[1] < V[3]$

$\triangleright V[3] \leftrightarrow V[1]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

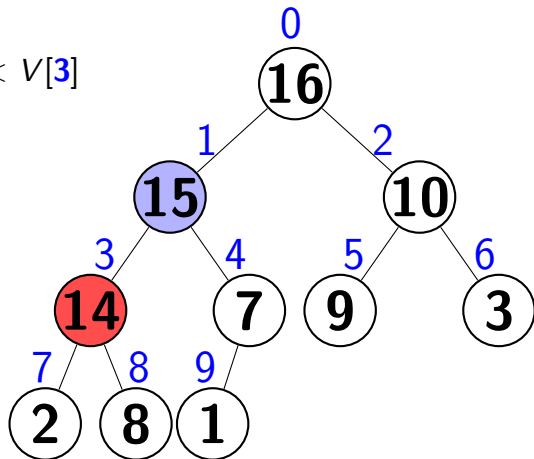
 return;

$V[8] = 15$;

while 3 > 0 and $V[1] < V[3]$

$V[3] \leftrightarrow V[1]$;

$\triangleright i = \text{PAI}(3)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

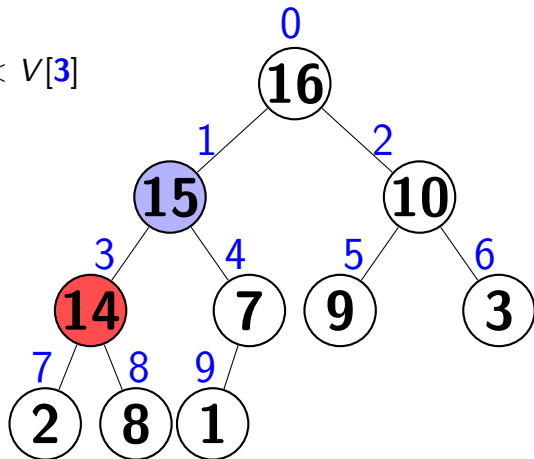
 return;

$V[8] = 15$;

while 3 > 0 and $V[1] < V[3]$

$V[3] \leftrightarrow V[1]$;

$\triangleright i = 1$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

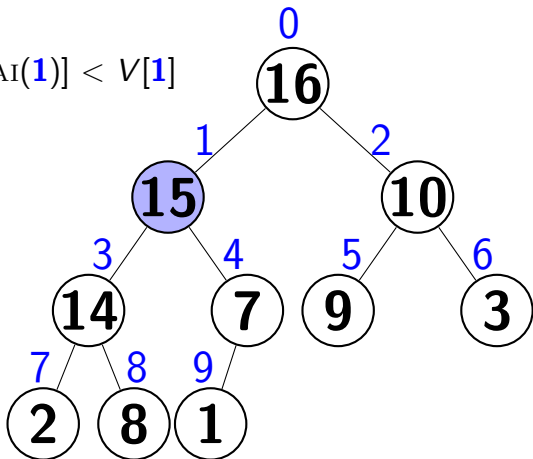
 return;

$V[8] = 15$;

▷ while 1 > 0 and $V[\text{PAI}(1)] < V[1]$

$V[i] \leftrightarrow V[\text{PAI}(1)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

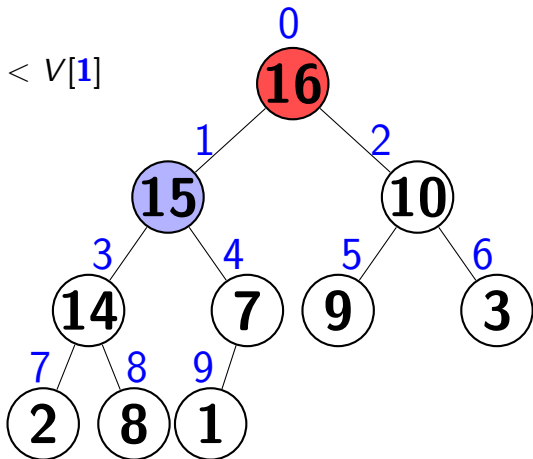
 return;

$V[8] = 15$;

▷ while 1 > 0 and $V[0] < V[1]$

$V[i] \leftrightarrow V[\text{PAI}(1)]$;

$i = \text{PAI}(i)$;



Heap-Increase-Key

HEAP-INCREASE-KEY (V , 8, 15)

if 15 < 4 then

 printf ("erro: chave menor que atual");

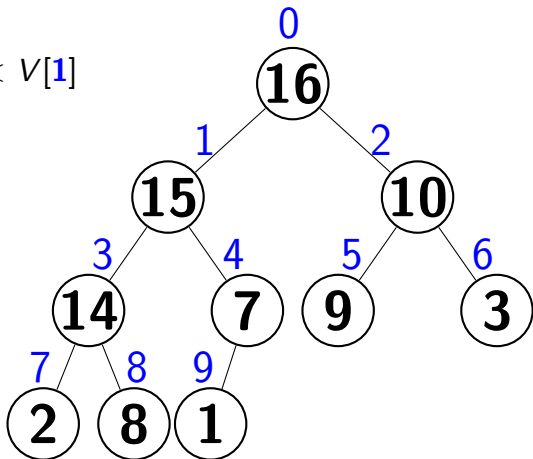
 return;

$V[8] = 15$;

while 1 > 0 and $V[0] < V[1]$

$V[i] \leftrightarrow V[\text{PAI}(1)]$;

$i = \text{PAI}(i)$;



Complexidade: o procedimento Heap-Increase-Key possui complexidade $\mathcal{O}(\log n)$ para um heap com n elementos, pois o caminho traçado do nó atualizado até a raiz tem comprimento máximo de $\mathcal{O}(\log n)$.

Sumário

- 1 Heap
- 2 Filas de Prioridades
- 3 Operação Heap-Maximum
- 4 Operação Heap-Extract-Max
- 5 Operação Heap-Increase-Key
- 6 Operação Max-Heap-Insert**

Max-Heap-Insert

A operação Max-Heap-Insert (V, x) insere o elemento x no conjunto V . Essa operação poderia ser escrita como $V \leftarrow V \cup \{x\}$. Essa operação recebe como entrada a **chave** a ser inserida e **expande o heap com a nova chave**. Note que nessa operação é necessário encontrar a posição apropriada para o elemento recém adicionado.

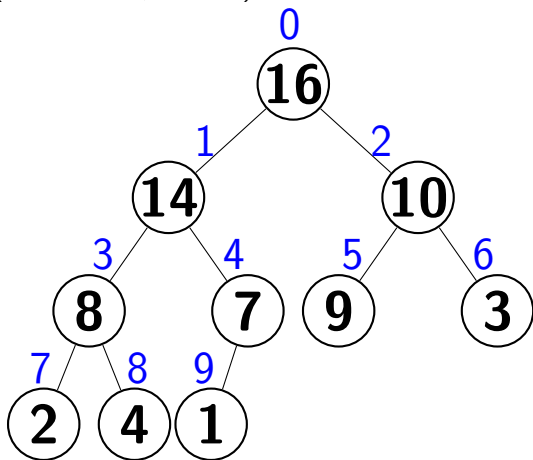
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size , **chave**)

$\text{Size} = \text{Size} + 1;$

$V[\text{Size} - 1] = -\infty;$

HEAP-INCREASE-KEY (V , $\text{Size} - 1$, **chave**);



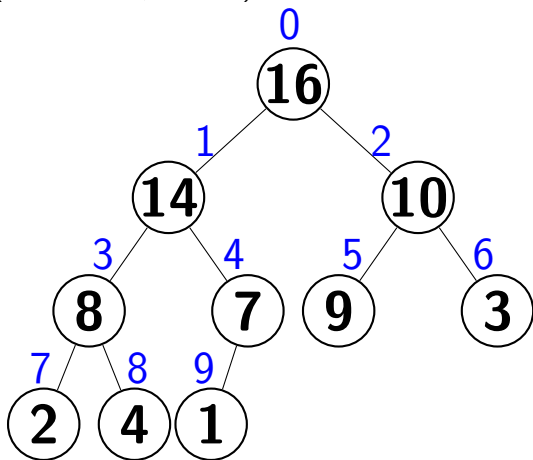
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, 20)

Size = Size+1;

$V[\text{Size}-1] = -\infty$;

HEAP-INCREASE-KEY (V , Size-1, **chave**);



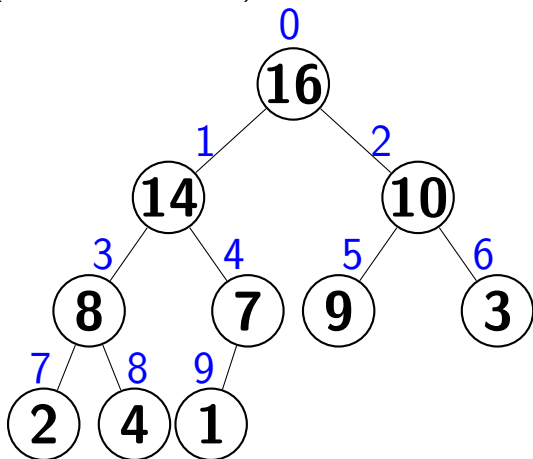
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

▷ Size = Size+1;

$V[\text{Size}-1] = -\infty$;

HEAP-INCREASE-KEY (V , Size-1, **chave**);



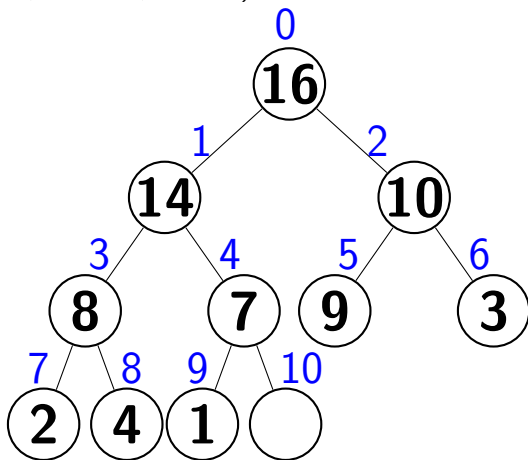
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

▷ Size = 10+1;

$V[\text{Size} - 1] = -\infty$;

HEAP-INCREASE-KEY (V , Size-1, **chave**);



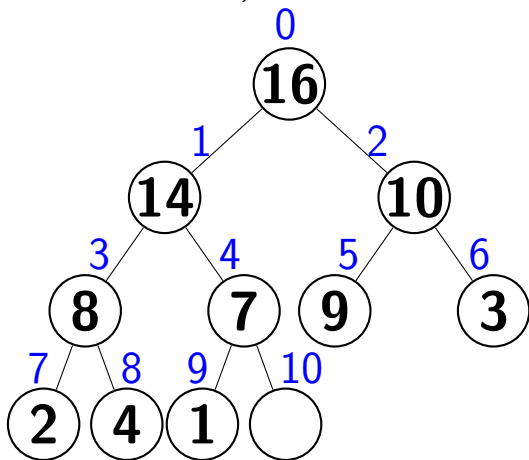
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, 20)

Size = 11;

$\triangleright V[Size - 1] = -\infty$;

HEAP-INCREASE-KEY (V , Size-1, **chave**);



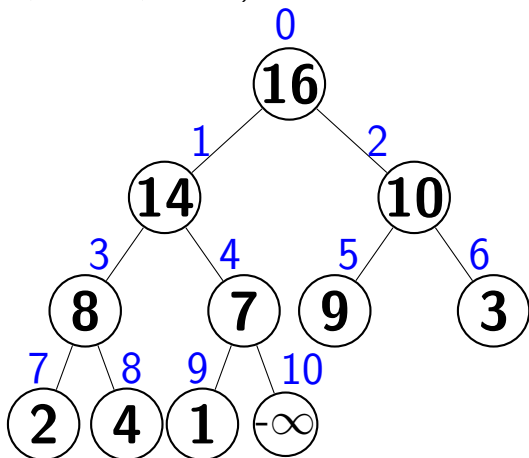
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

Size = 11;

$\triangleright V[10] = -\infty$;

HEAP-INCREASE-KEY (V , Size-1, **chave**);



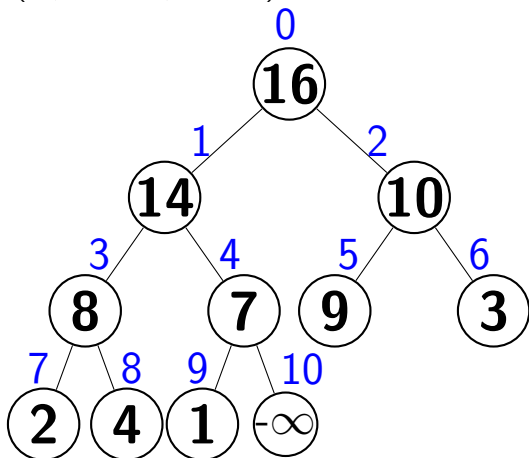
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

Size = 11;

$V[10] = -\infty$;

▷ HEAP-INCREASE-KEY (V , Size-1, **chave**);



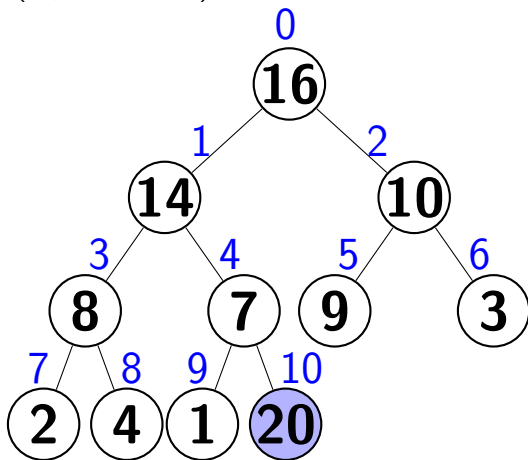
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

Size = 11;

$V[10] = -\infty$;

▷ HEAP-INCREASE-KEY (V , 10, **chave**);



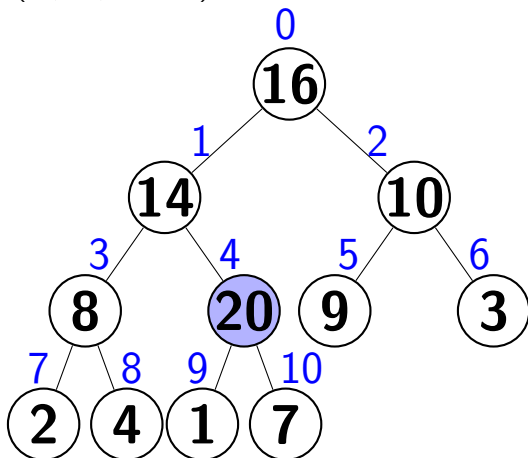
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

Size = 11;

$V[10] = -\infty$;

▷ HEAP-INCREASE-KEY (V , 10, **chave**);



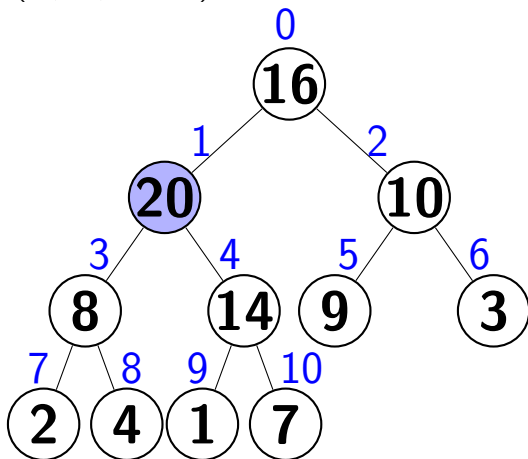
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

Size = 11;

$V[10] = -\infty$;

▷ HEAP-INCREASE-KEY (V , 10, **chave**);



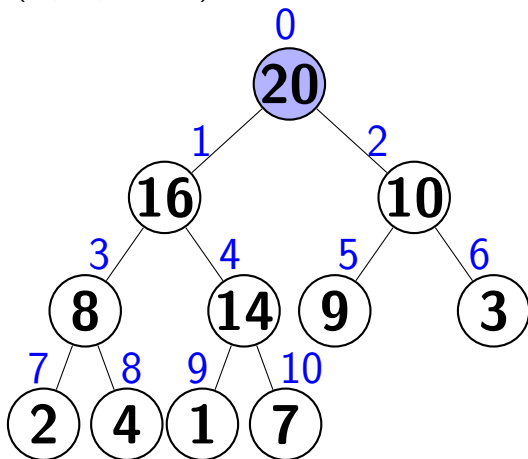
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

Size = 11;

$V[10] = -\infty$;

▷ HEAP-INCREASE-KEY (V , 10, **chave**);



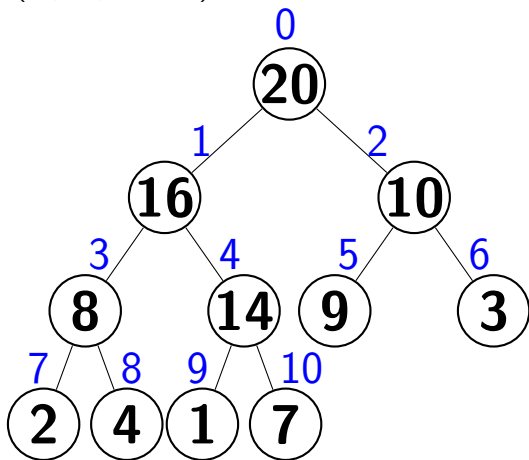
Max-Heap-Insert

MAX-HEAP-INSERT (V , Size, **20**)

Size = 11;

$V[10] = -\infty$;

▷ HEAP-INCREASE-KEY (V , 10, **chave**);



Complexidade: o procedimento Max-Heap-Insert possui complexidade $\mathcal{O}(\log n)$ para um heap com n elementos, pois a complexidade do procedimento Heap-Increase-Key é de $\mathcal{O}(\log n)$. Em resumo, um heap pode admitir qualquer operação de fila de prioridades em um conjunto de tamanho n no tempo $\mathcal{O}(\log n)$.

Referências

Algoritmos: Teoria e prática. Cormen et al.