ESTRUTURAS DE DADOS

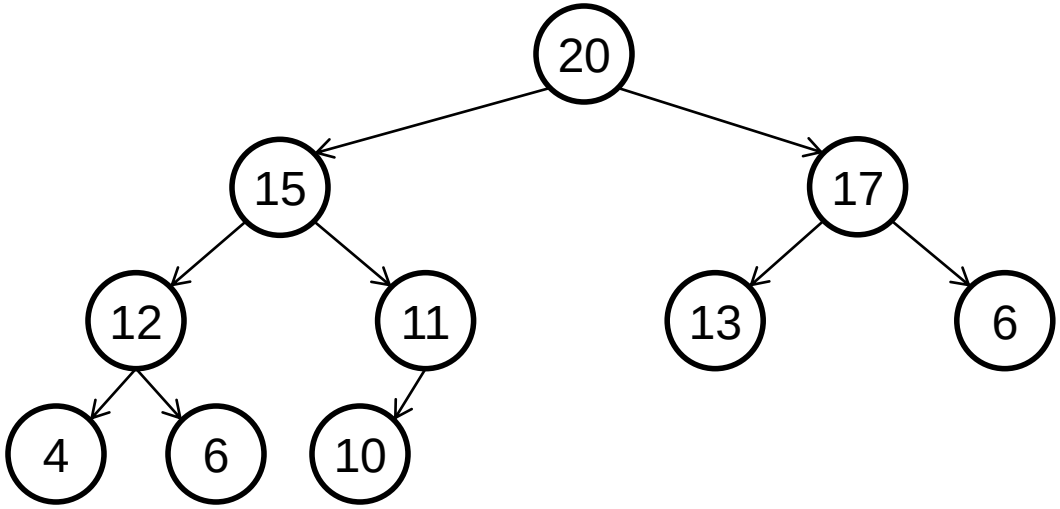
Aula 10 – Heaps

- Conceitos
- Estrutura
- Algoritmos

Centro Universitário da FEI

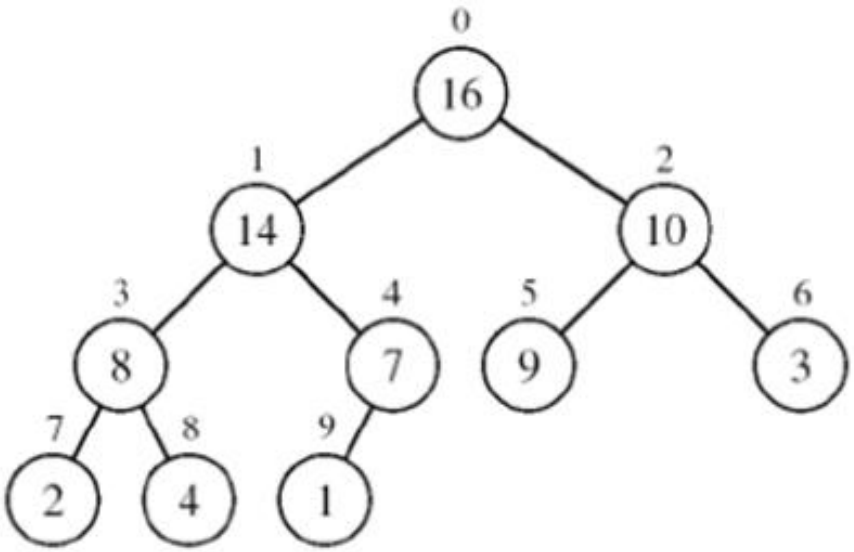
Heap

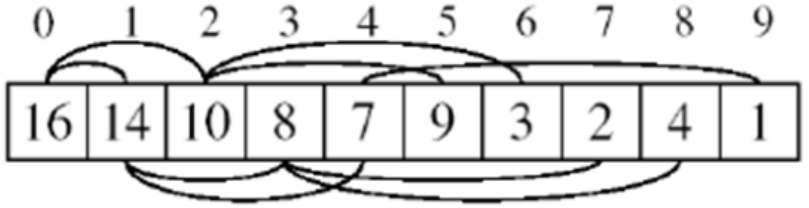
- Um Heap é uma estrutura de árvore binária com a seguinte propriedade:
  - Balanceamento: Uma árvore binária quase completa, com eventual excessão do último nível. Quando o último nível não está completo, as folhas devem estar mais a esquerda da árvore.
  - Estrutural: O valor armazenado em cada nó não é menor que os de seus filhos.



Heap

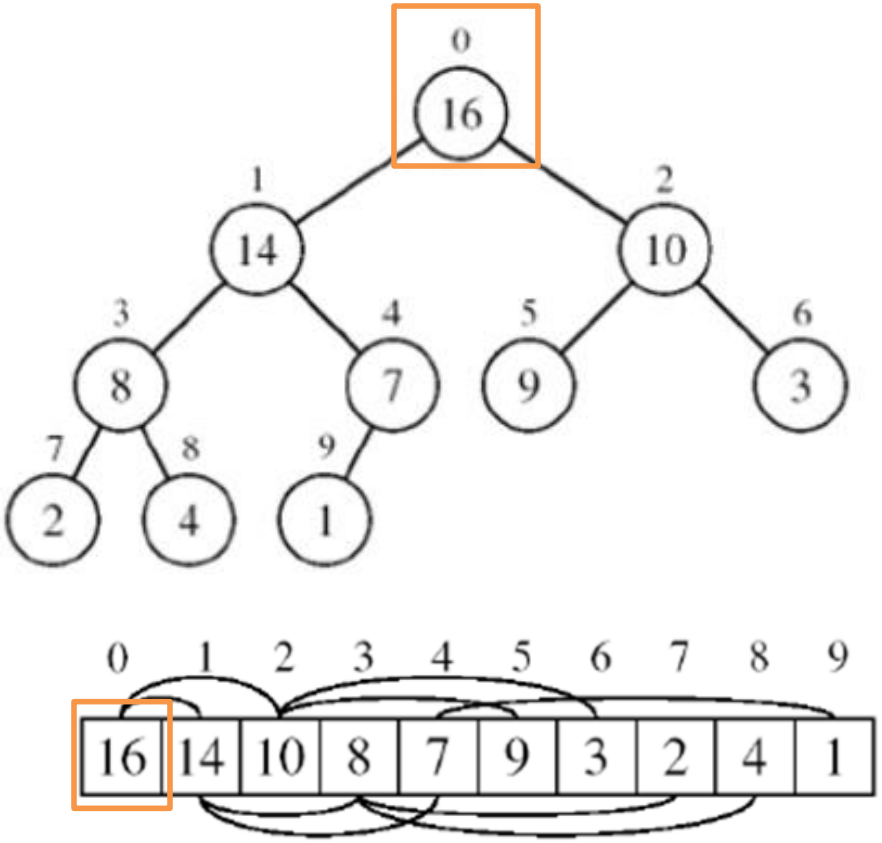
- Normalmente, um heap é armazenado em um vetor, onde cada posição deste vetor é um nó da árvore.





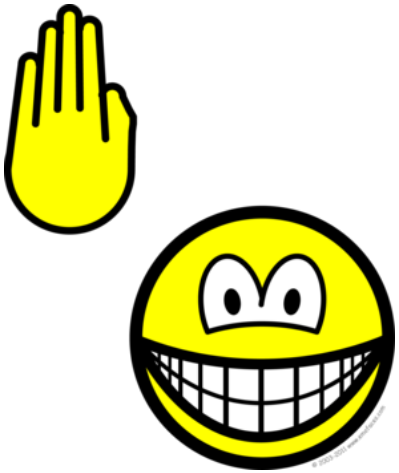
Representação de Heap em um Vetor

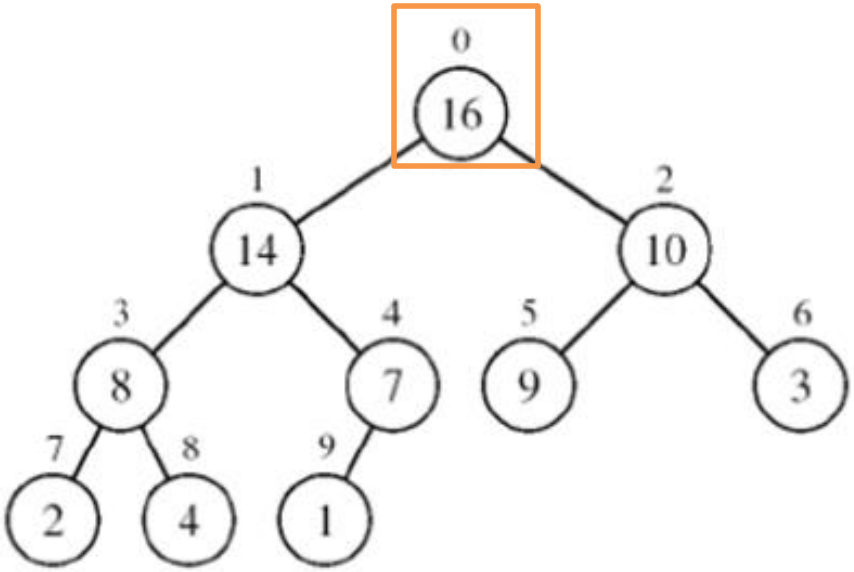
- A raiz da árvore está sempre localizada na posição V[0]



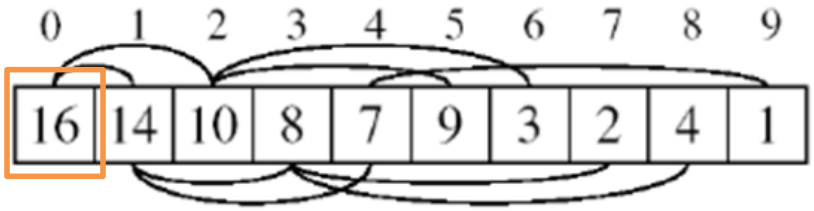
Representação de Heap em um Vetor

- A raiz da árvore está sempre localizada na posição V[0]



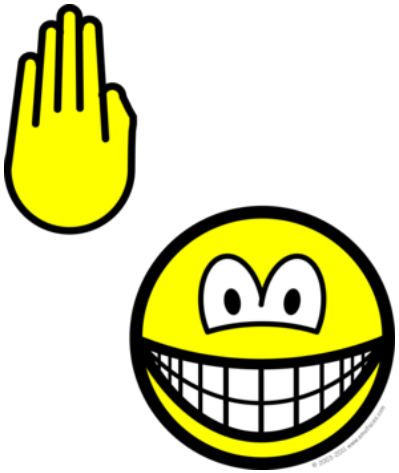


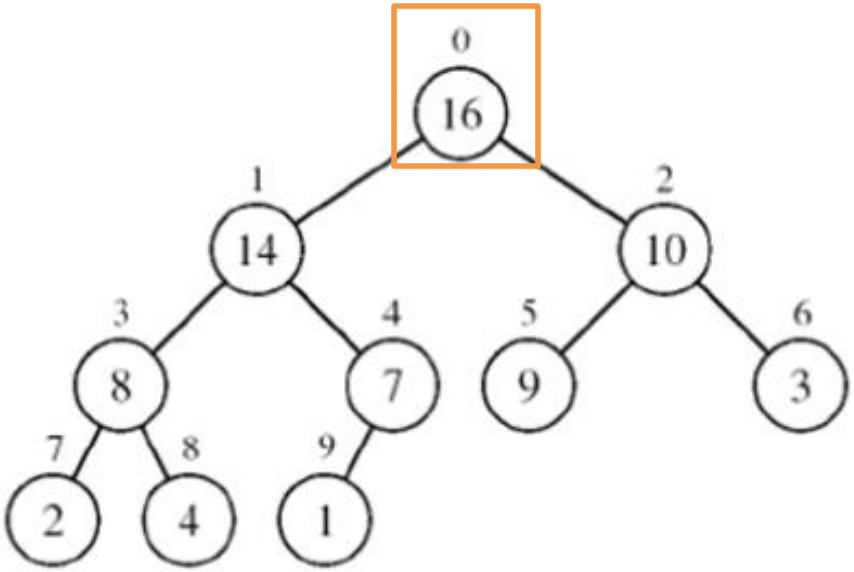
Isso significa que o maior valor ficará sempre em v[0] ?

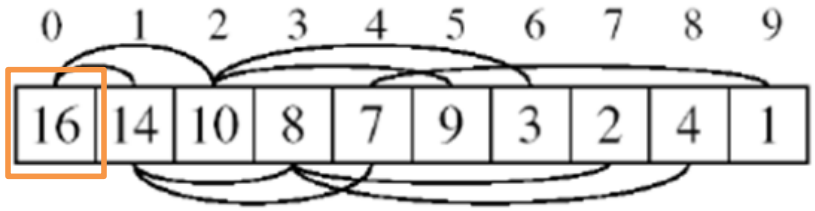


Representação de Heap em um Vetor

- A raiz da árvore está sempre localizada na posição V[0]







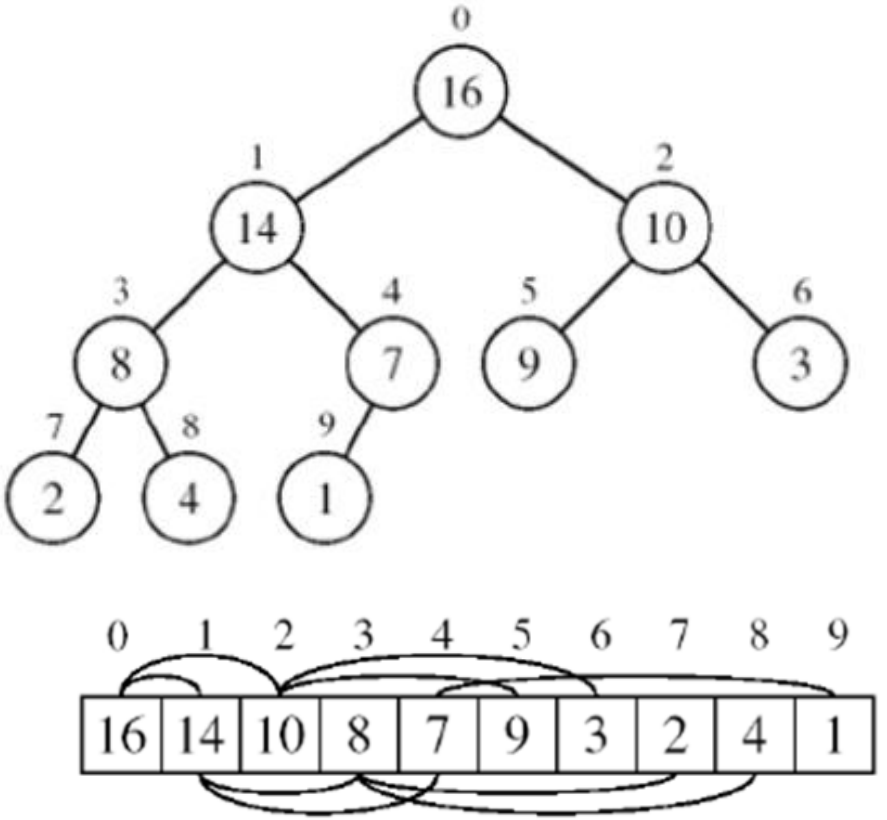
Isso significa que o maior valor ficará sempre em v[0] ? SIM!!

Representação de Heap em um Vetor

- Filho esquerdo

O filho esquerdo de um nó de índice i é dado por:

$$fe = 2.i + 1$$

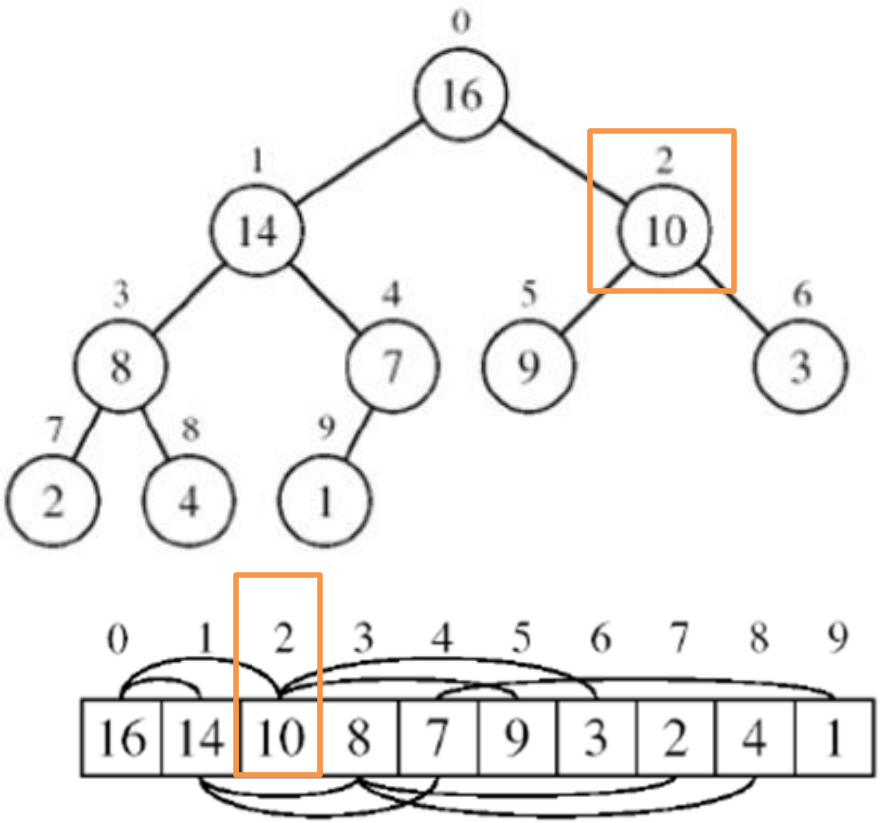


Representação de Heap em um Vetor

- Filho esquerdo

O filho esquerdo de um nó de índice i é dado por:

$$fe = 2.i + 1$$

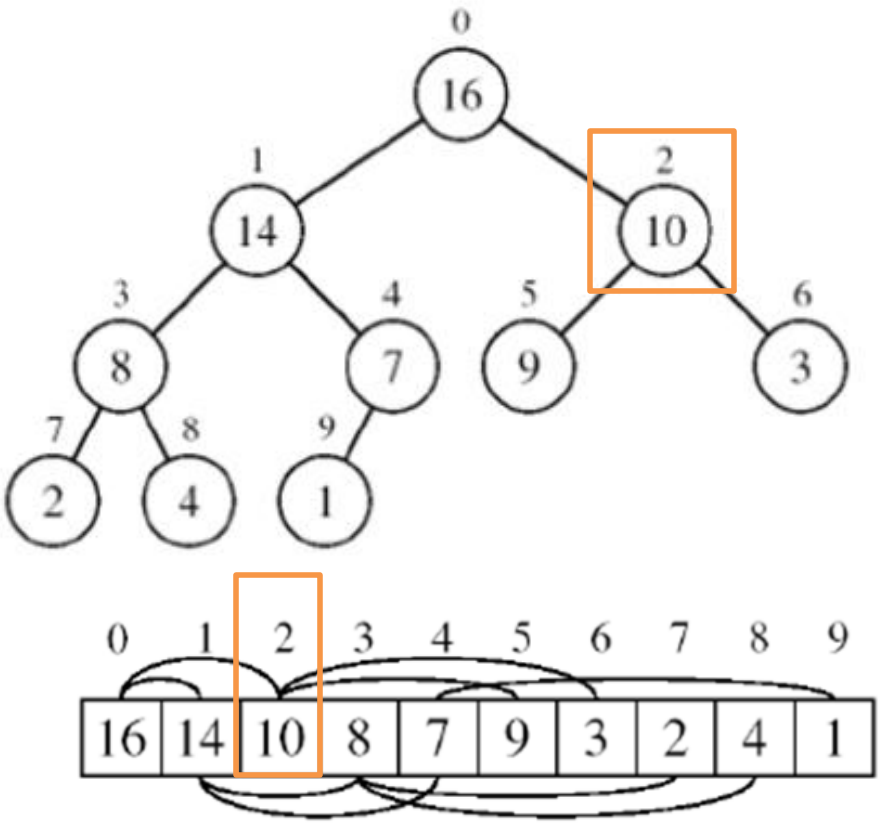


Representação de Heap em um Vetor

- Filho esquerdo

O filho esquerdo de um nó de índice i é dado por:

$$fe = 2.i + 1$$



Filho esquerdo do nó de índice 2 é

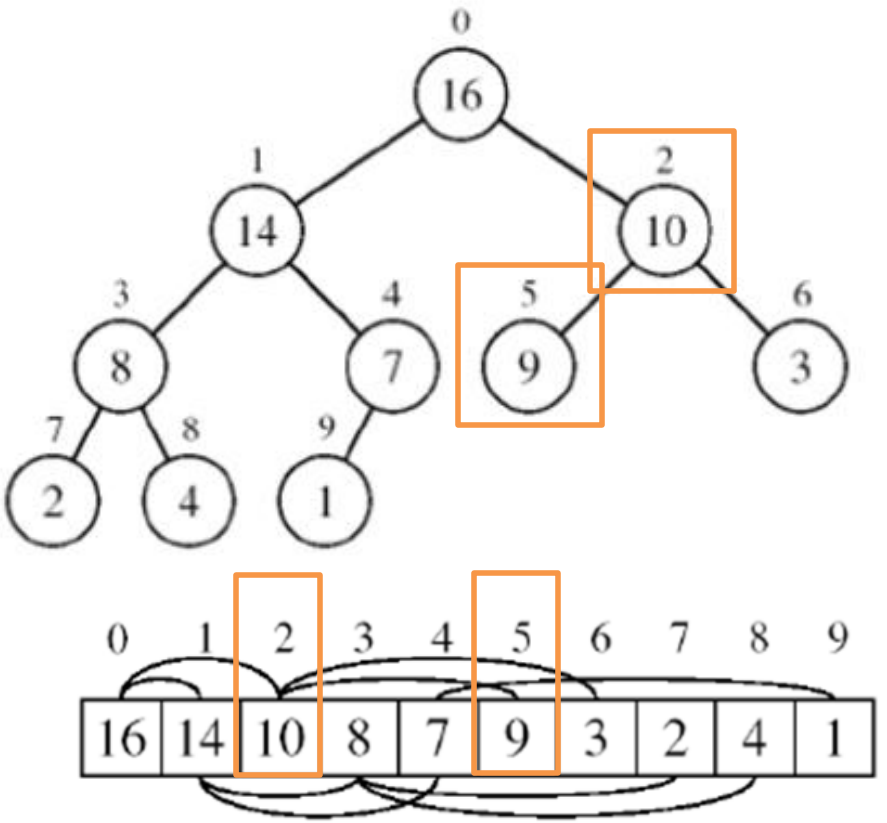
$$2.(2) + 1 = 5$$

Representação de Heap em um Vetor

- Filho esquerdo

O filho esquerdo de um nó de índice i é dado por:

$$fe = 2.i + 1$$



Filho esquerdo do nó de índice 2 é

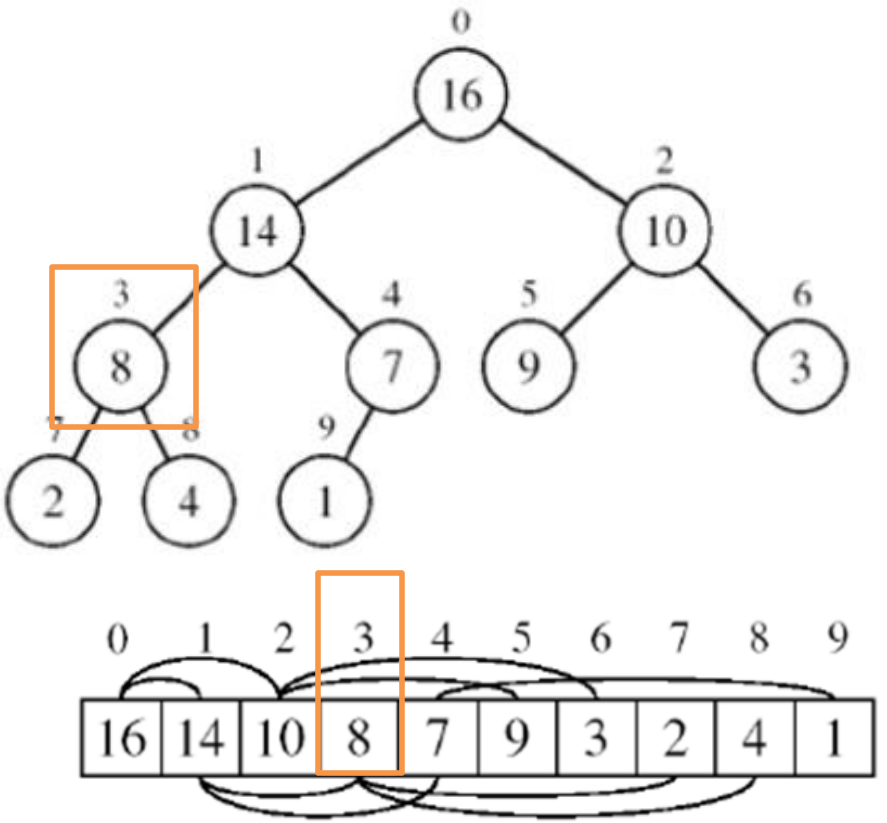
$$2.(2) + 1 = 5$$

Representação de Heap em um Vetor

- Filho esquerdo

O filho esquerdo de um nó de índice i é dado por:

$$fe = 2.i + 1$$



Filho esquerdo do nó de índice 3 é

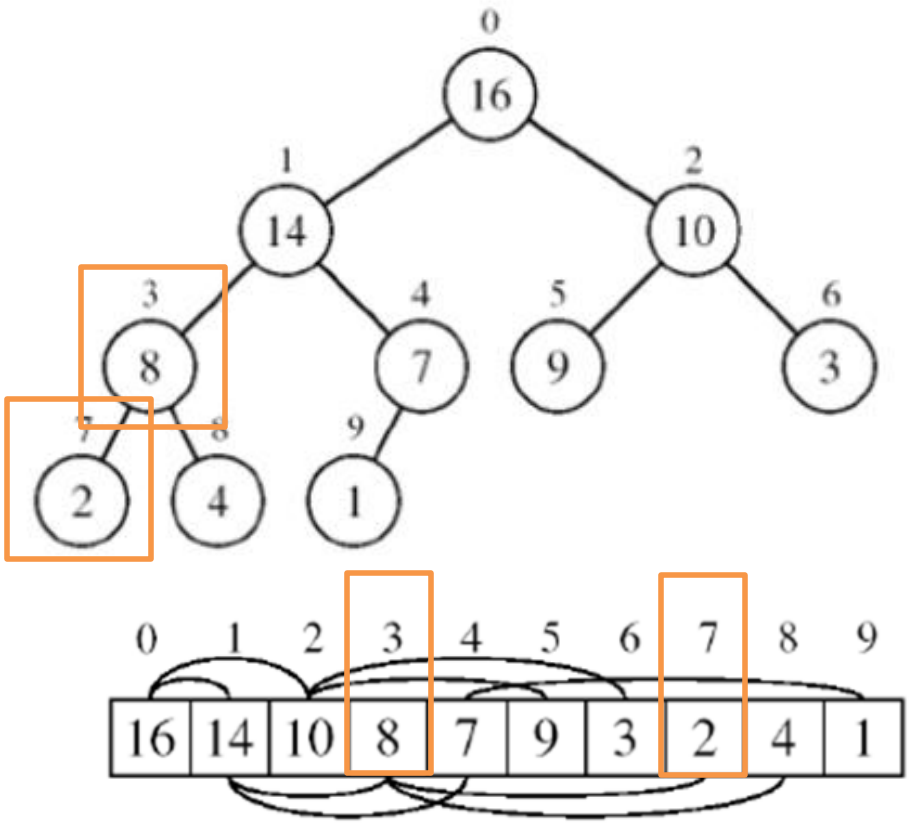
$$2.(3) + 1 = 7$$

Representação de Heap em um Vetor

- Filho esquerdo

O filho esquerdo de um nó de índice i é dado por:

$$fe = 2.i + 1$$



Filho esquerdo do nó de índice 3 é

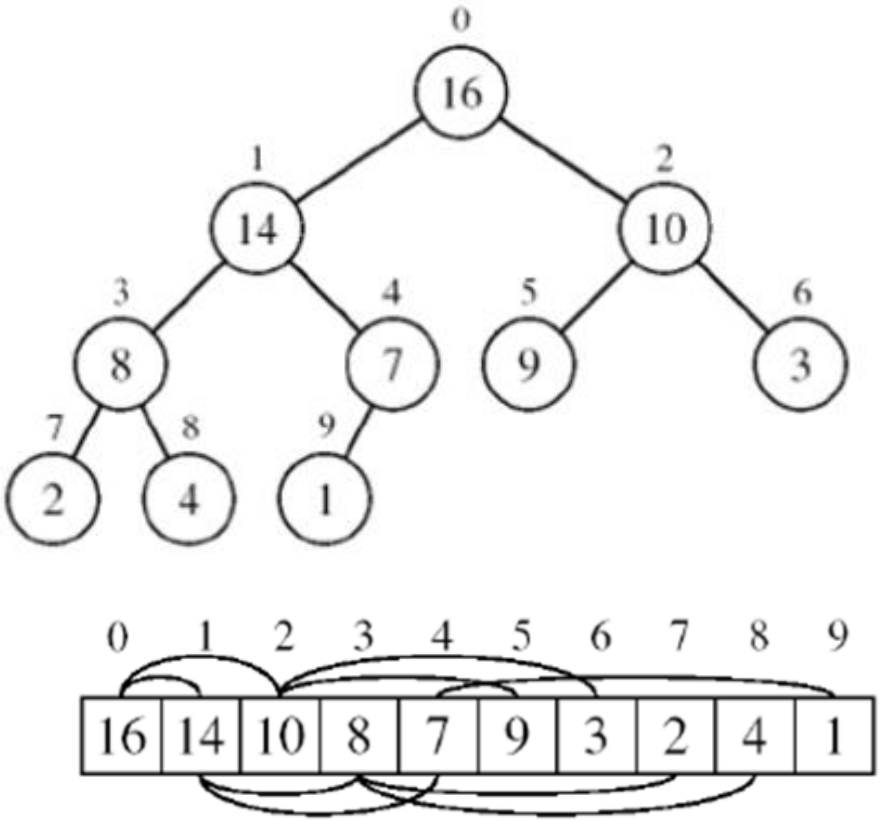
$$2.(3) + 1 = 7$$

Representação de Heap em um Vetor

- Filho direito

O filho direito de um nó de índice i é dado por:

$$fd = 2.i + 2$$



Representação de Heap em um Vetor

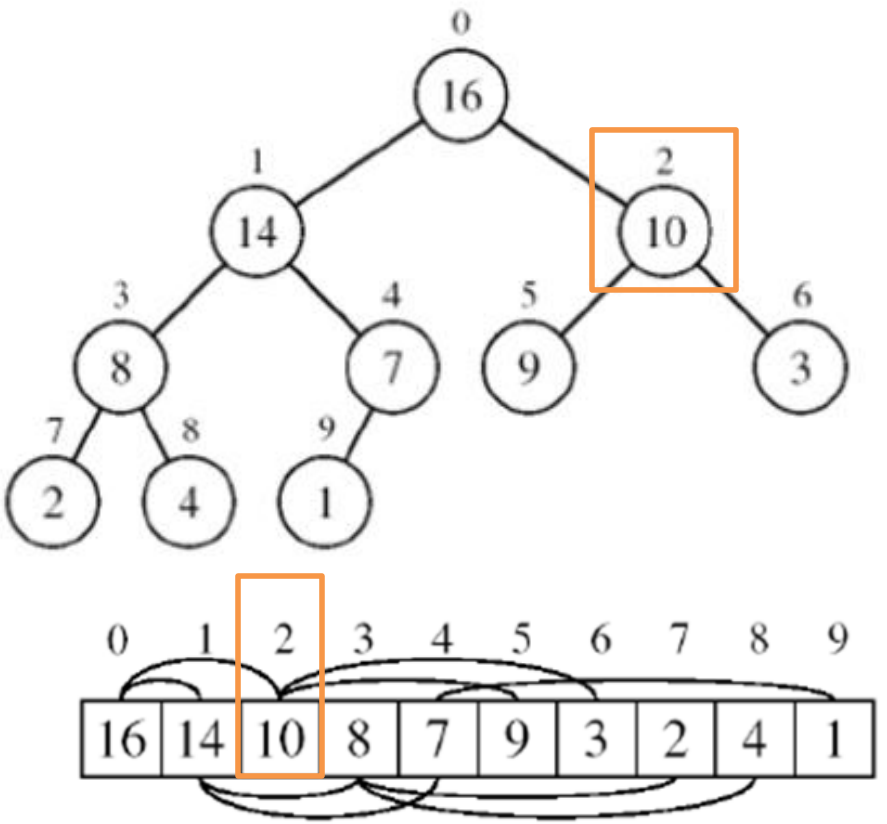
- Filho direito

O filho direito de um nó de índice i é dado por:

$$fd = 2.i + 2$$

Filho direito do nó de índice 2 é

$$2.(2) + 2 = 6$$



Representação de Heap em um Vetor

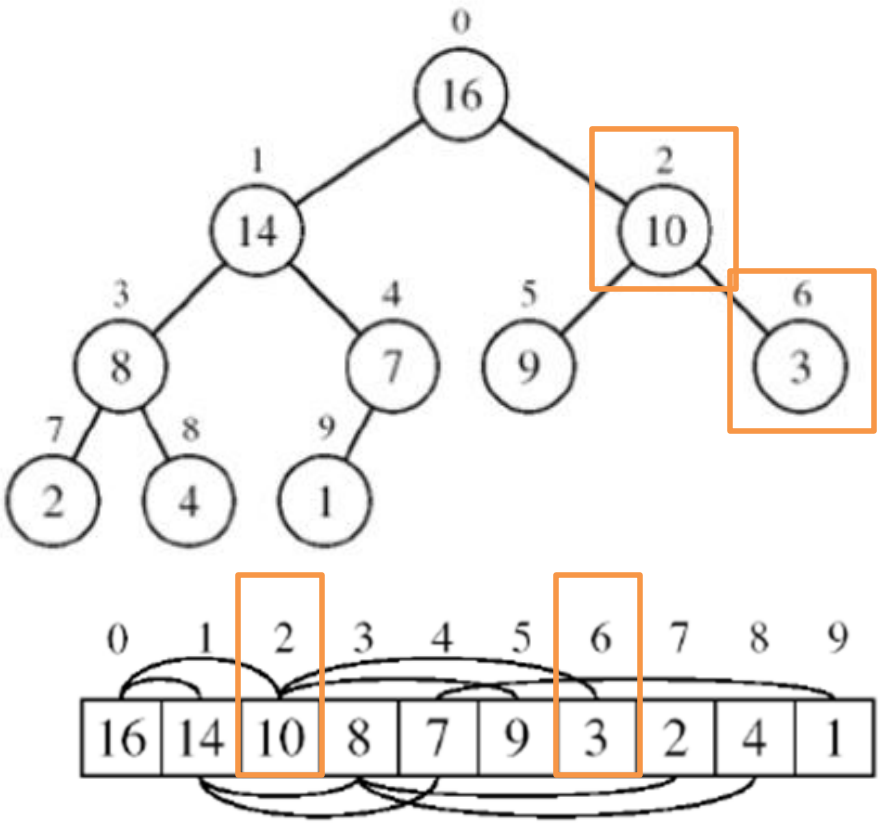
- Filho direito

O filho direito de um nó de índice i é dado por:

$$fd = 2.i + 2$$

Filho direito do nó de índice 2 é

$$2.(2) + 2 = 6$$



Representação de Heap em um Vetor

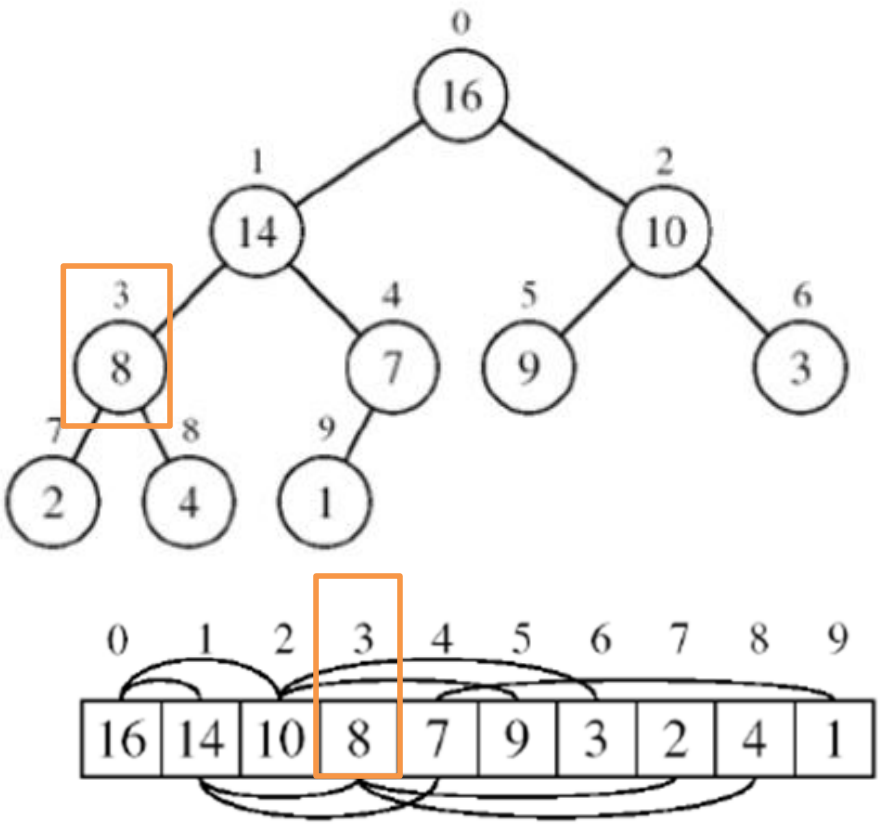
- Filho direito

O filho direito de um nó de índice i é dado por:

$$fd = 2.i + 2$$

Filho direito do nó de índice 3 é

$$2.(3) + 2 = 8$$



Representação de Heap em um Vetor

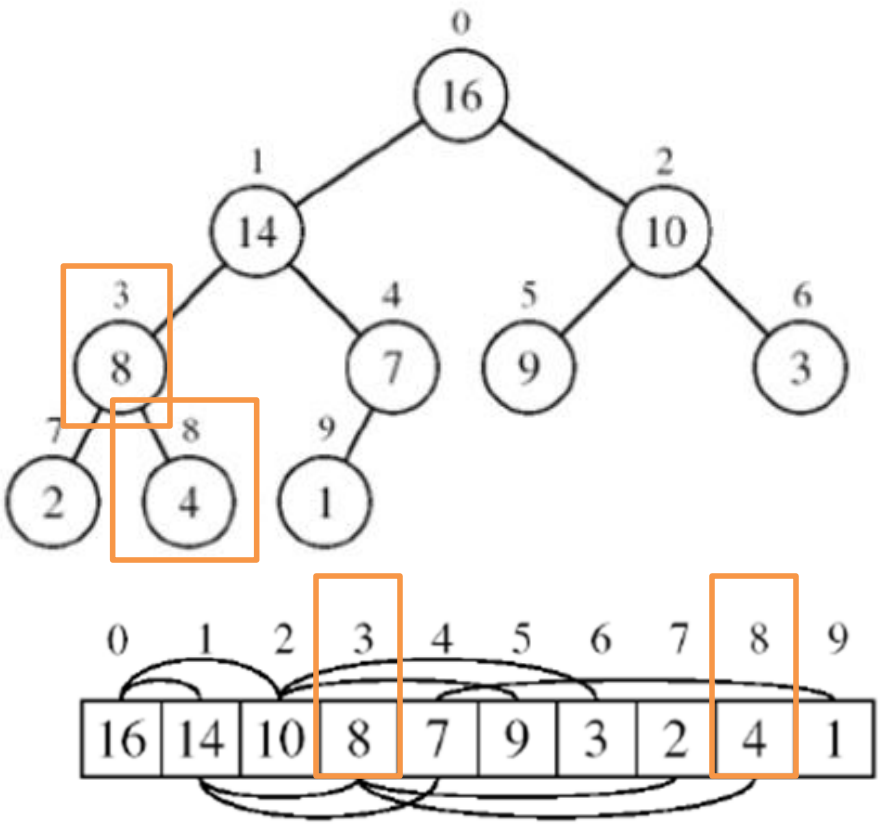
- Filho direito

O filho direito de um nó de índice i é dado por:

$$fd = 2.i + 2$$

Filho direito do nó de índice 3 é

$$2.(3) + 2 = 8$$

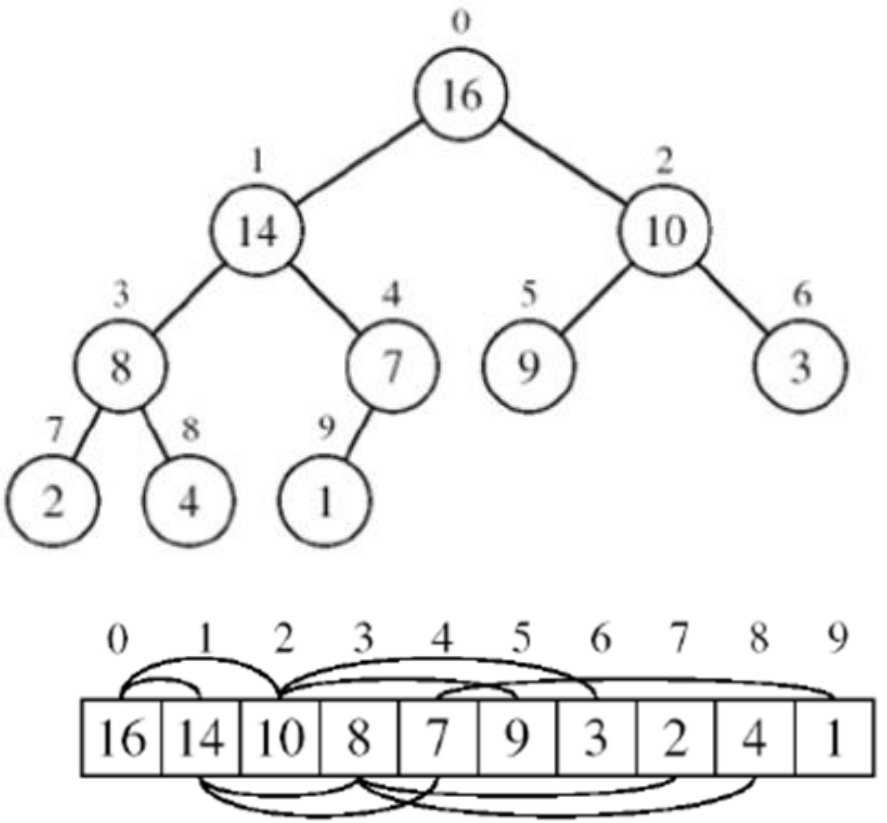


Representação de Heap em um Vetor

- Pai

O pai de um nó de índice i é dado por:

$$p = \lfloor i/2 \rfloor$$



Representação de Heap em um Vetor

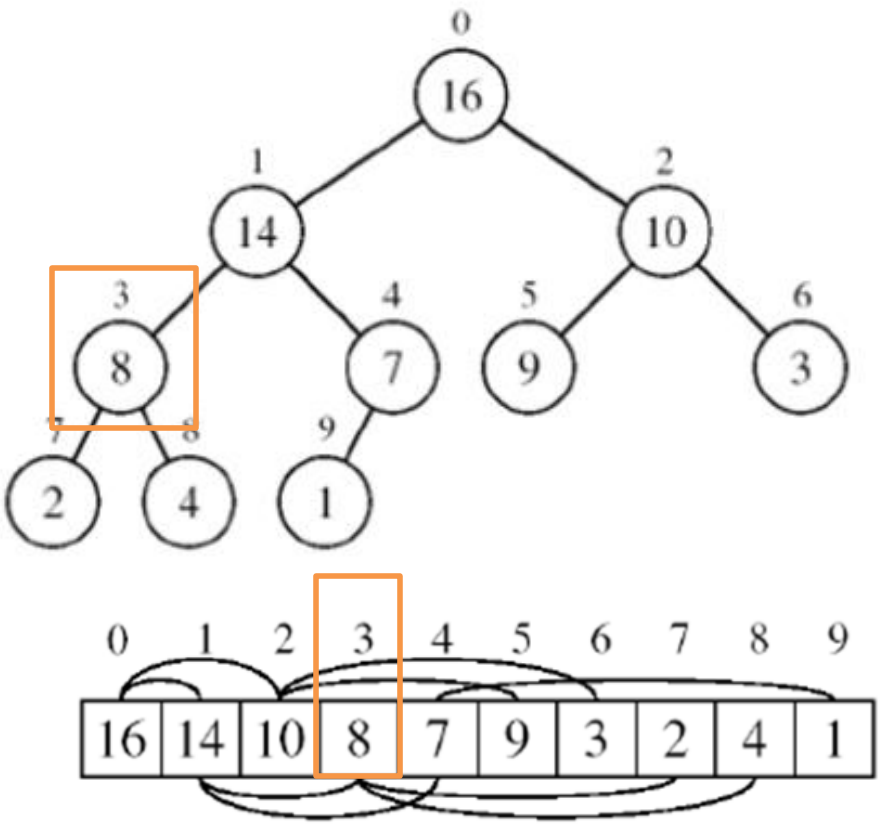
- Pai

O pai de um nó de índice i é dado por:

$$p = \lfloor (i-1)/2 \rfloor$$

Pai do nó de índice 3 é

$$\lfloor (3-1)/2 \rfloor = 1$$



Representação de Heap em um Vetor

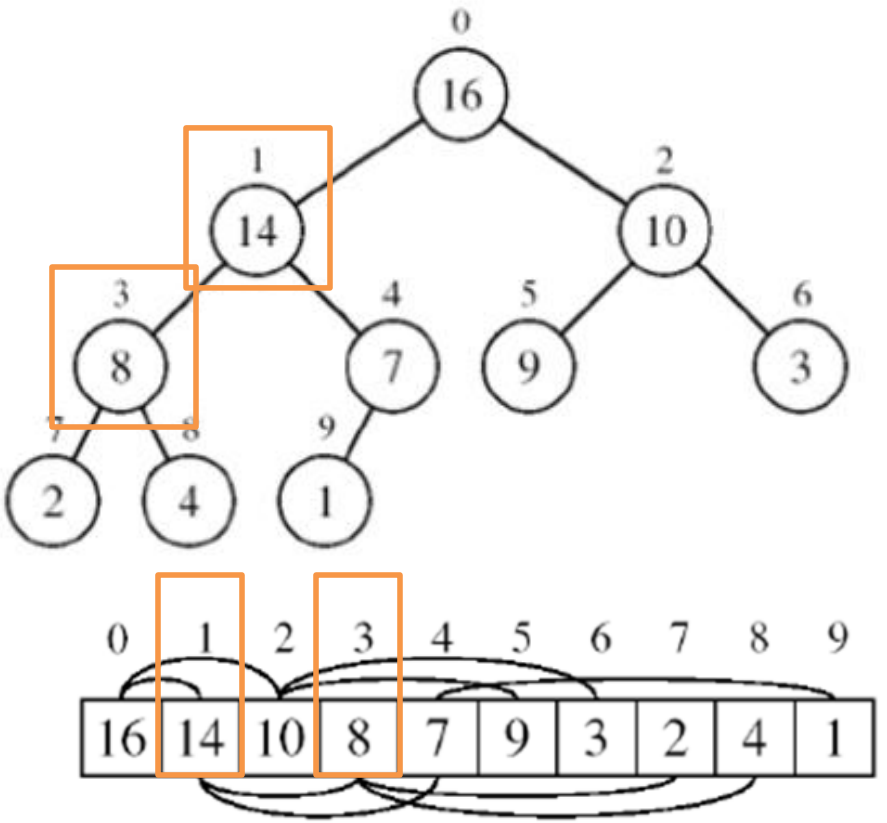
- Pai

O pai de um nó de índice i é dado por:

$$p = \lfloor (i-1)/2 \rfloor$$

Pai do nó de índice 3 é

$$\lfloor (3-1)/2 \rfloor = 1$$

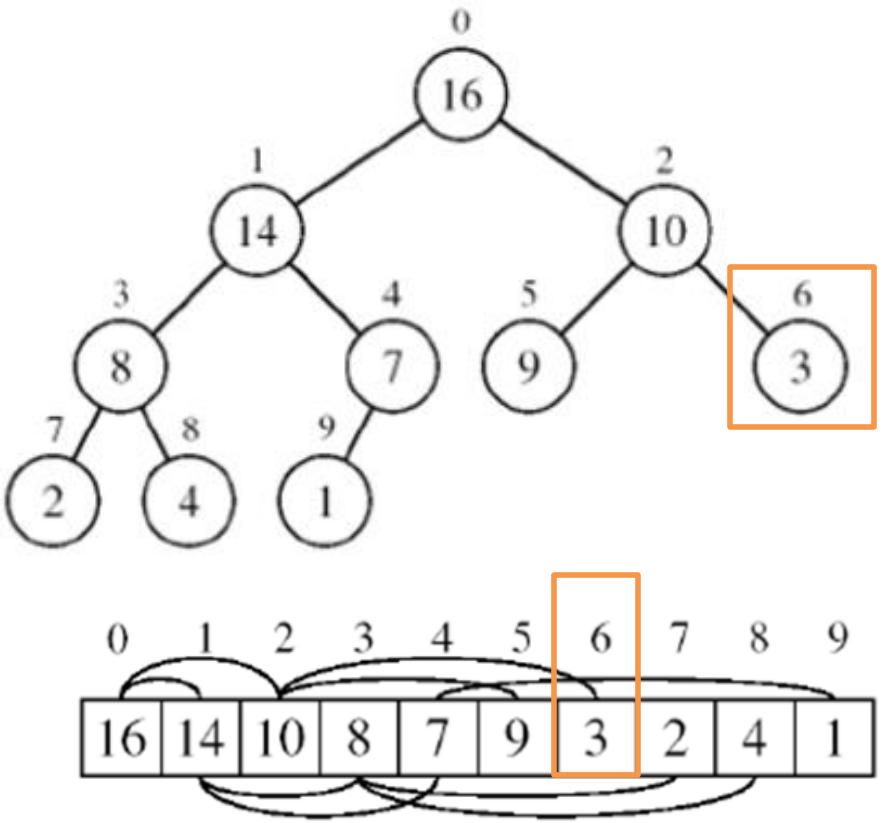


Representação de Heap em um Vetor

- Pai

O pai de um nó de índice i é dado por:

$$p = \lfloor (i-1)/2 \rfloor$$



Pai do nó de índice 6 é

$$\lfloor (6-1)/2 \rfloor = 2$$

Representação de Heap em um Vetor

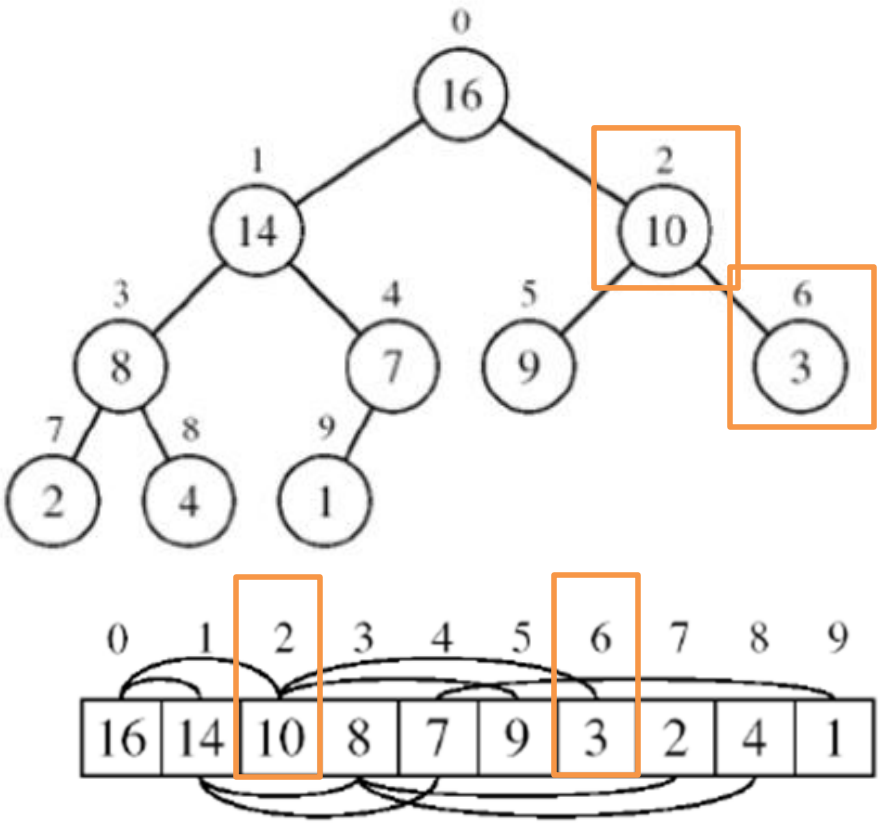
- Pai

O pai de um nó de índice i é dado por:

$$p = \lfloor (i-1)/2 \rfloor$$

Pai do nó de índice 6 é

$$\lfloor (6-1)/2 \rfloor = 2$$

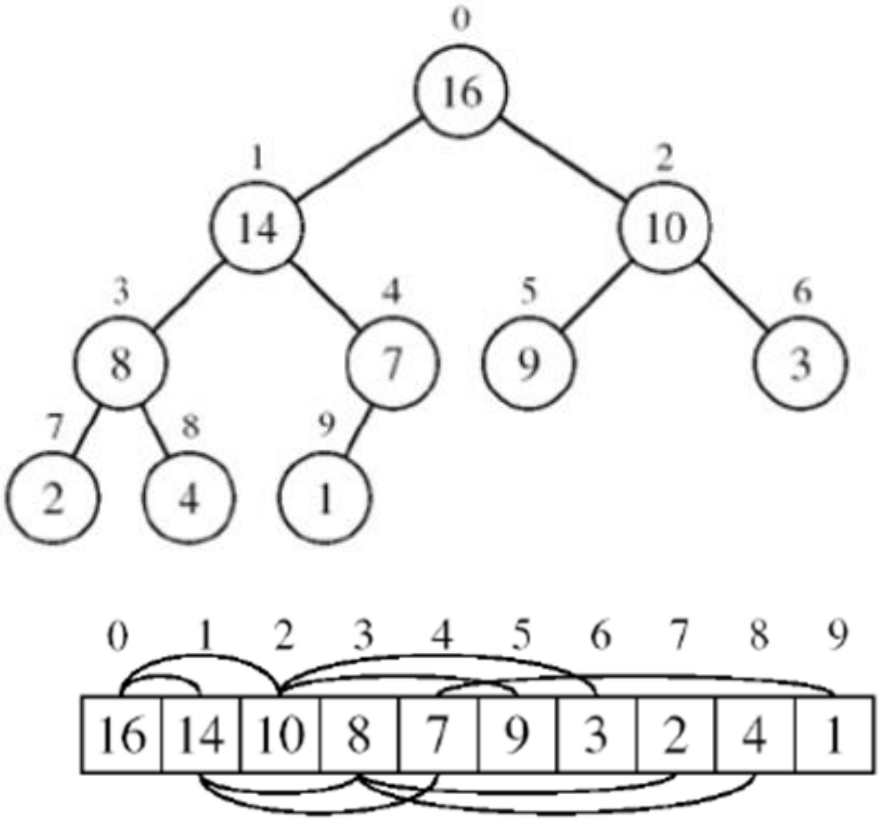


Representação de Heap em um Vetor

- Último Pai

O pai de um nó de índice i é dado por:

$$up = \lfloor n/2 \rfloor - 1$$

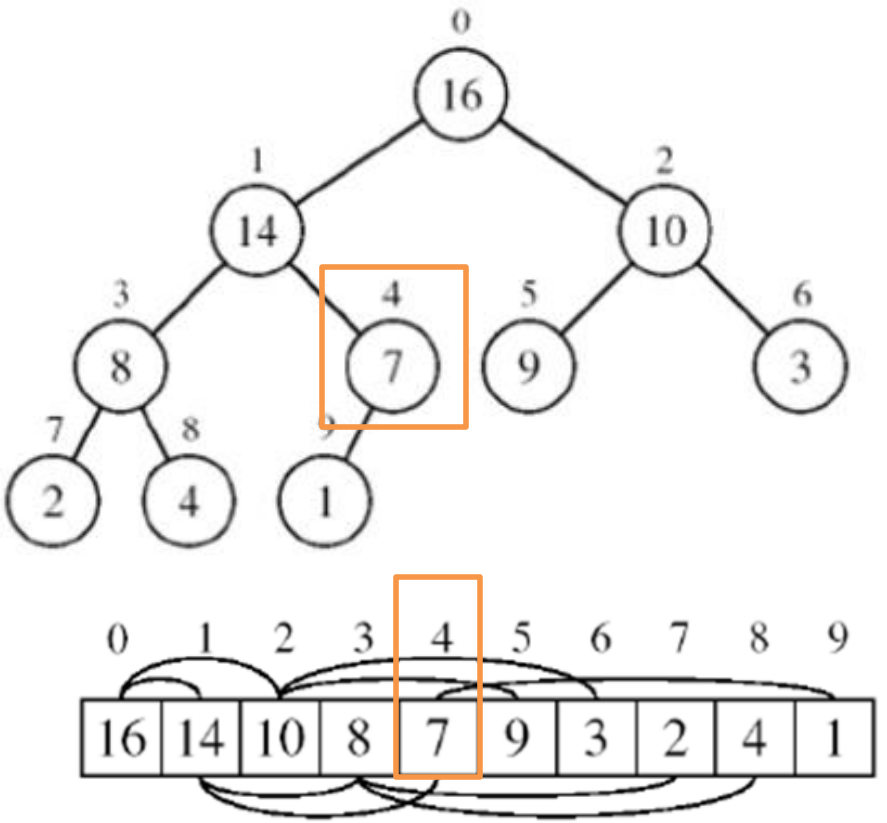


Representação de Heap em um Vetor

- Último Pai

O pai de um nó de índice i é dado por:

$$up = \lfloor n/2 \rfloor - 1$$



Esta árvore contém $n = 10$ nós.
Portanto, o último pai é:

$$\lfloor 10/2 \rfloor - 1 = 4$$

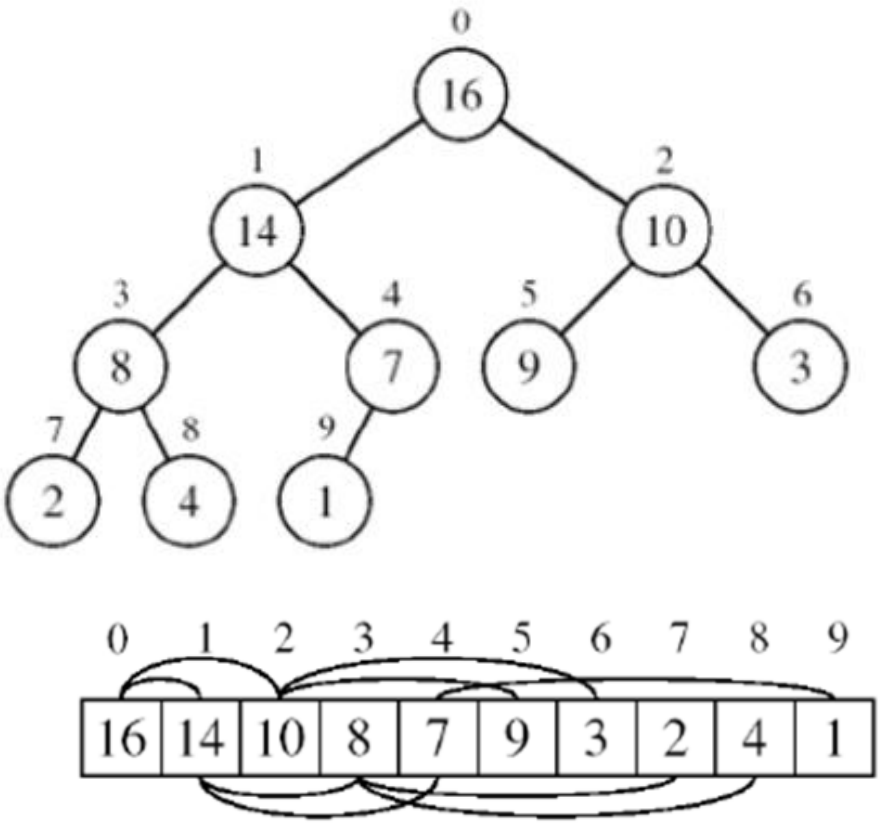
Representação de Heap em um Vetor

- Folhas

As folhas de um heap é dado por:

$$\lfloor n/2 \rfloor \leq f < n$$

Onde f é o índice da folha



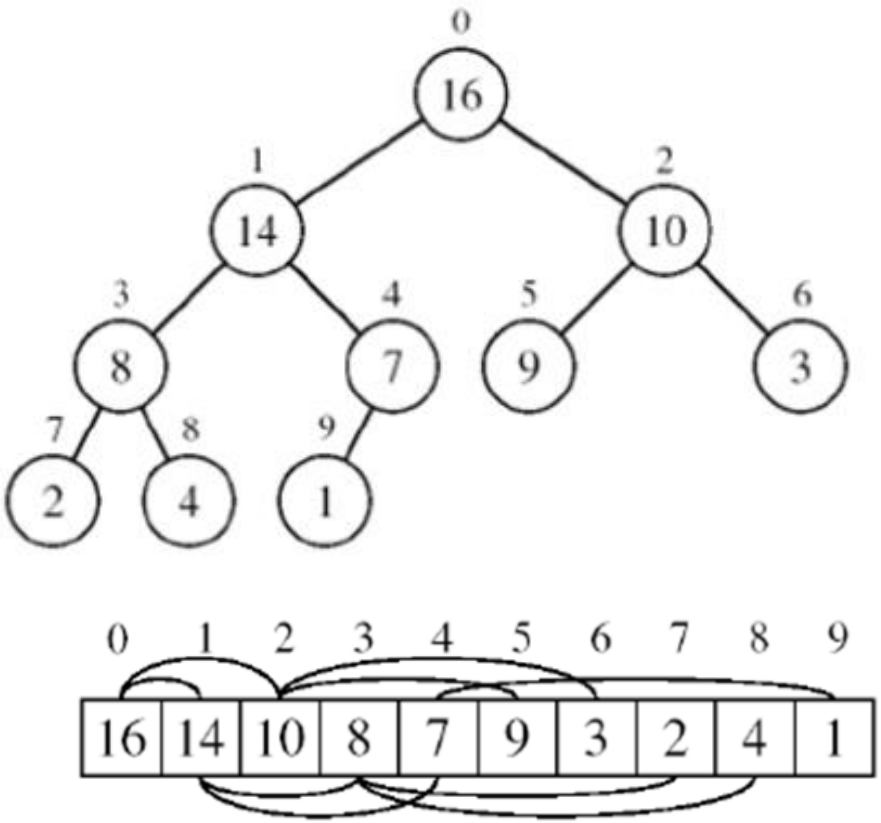
Representação de Heap em um Vetor

- Folhas

As folhas de um heap é dado por:

$$\lfloor n/2 \rfloor \leq f < n$$

Onde f é o índice da folha



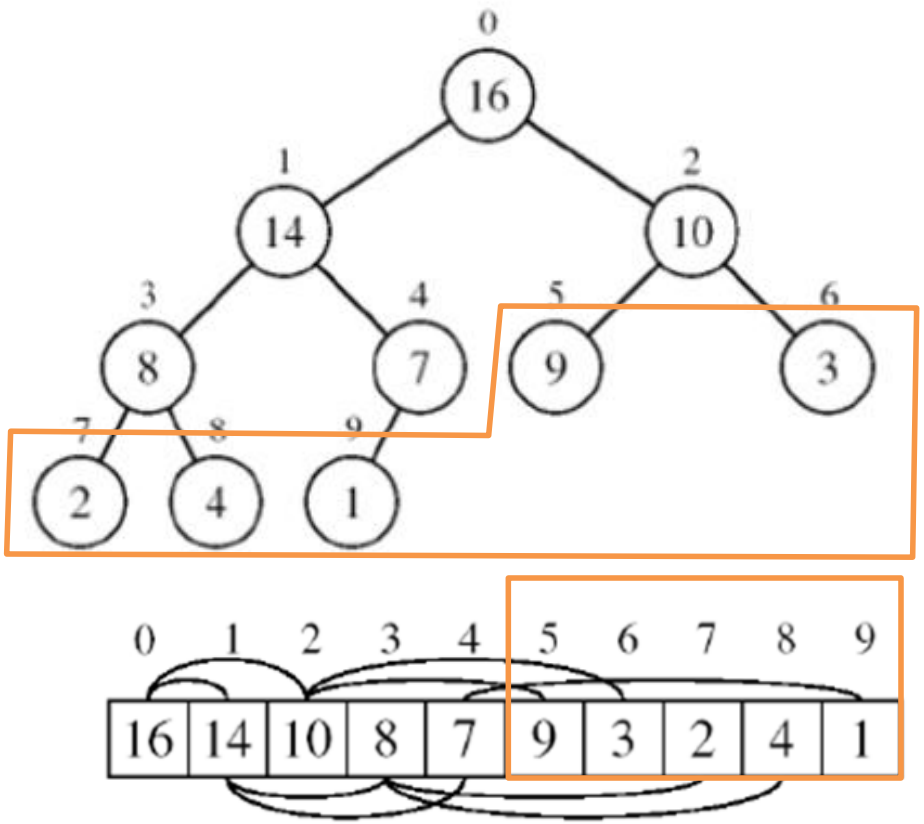
Representação de Heap em um Vetor

- Folhas

As folhas de um heap é dado por:

$$\lfloor n/2 \rfloor \leq f < n$$

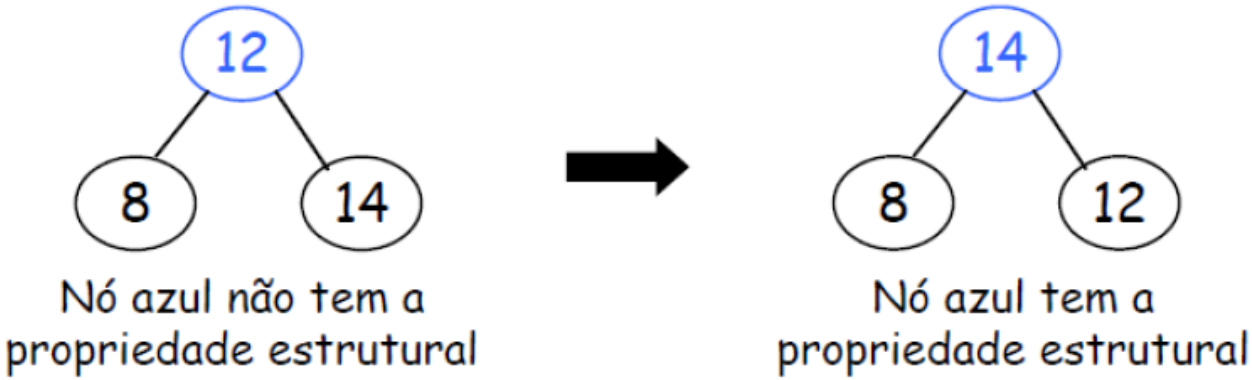
Onde f é o índice da folha



Manutenção da Propriedade Estrutural

Caso um nó de um heap perca a sua propriedade estrutural, poderá recuperá-la trocando de valor com o seu filho maior.

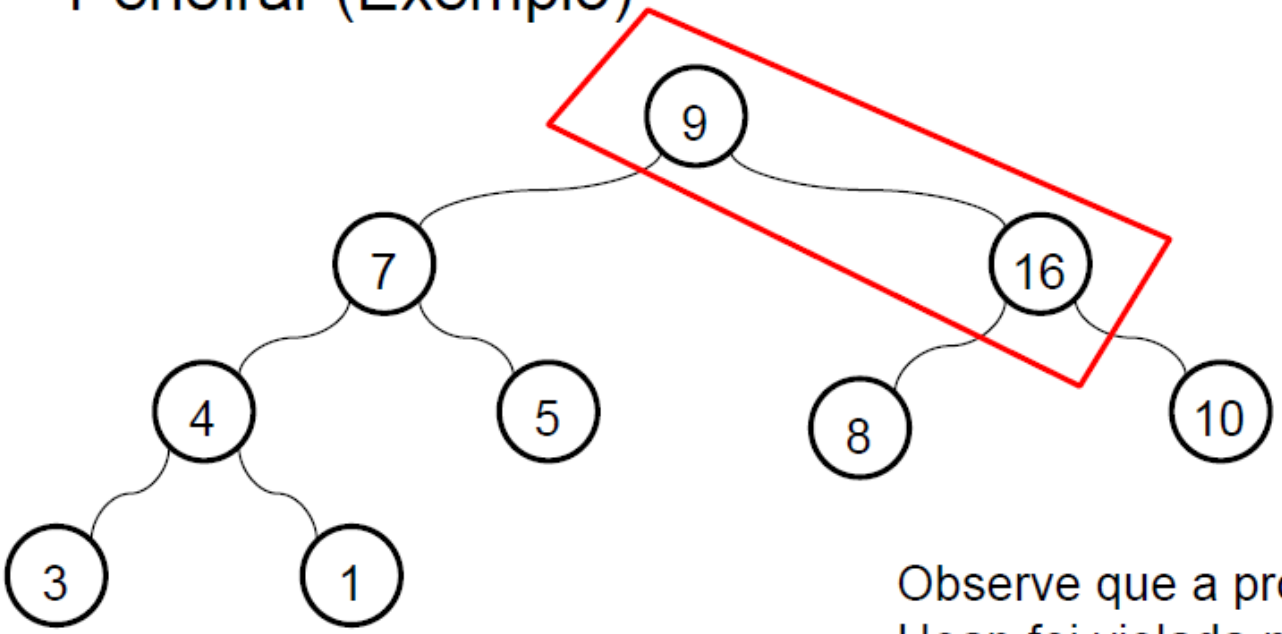
Isso pode ser feito através do algoritmo PENEIRAR (Sift).



Uma vez que o filho trocou de lugar com o pai, a subárvore que protagonizou a troca pode ter perdido a propriedade estrutural de heap, e também precisará invocar PENEIRAR para ela.

Peneirar

- Peneirar (Exemplo)

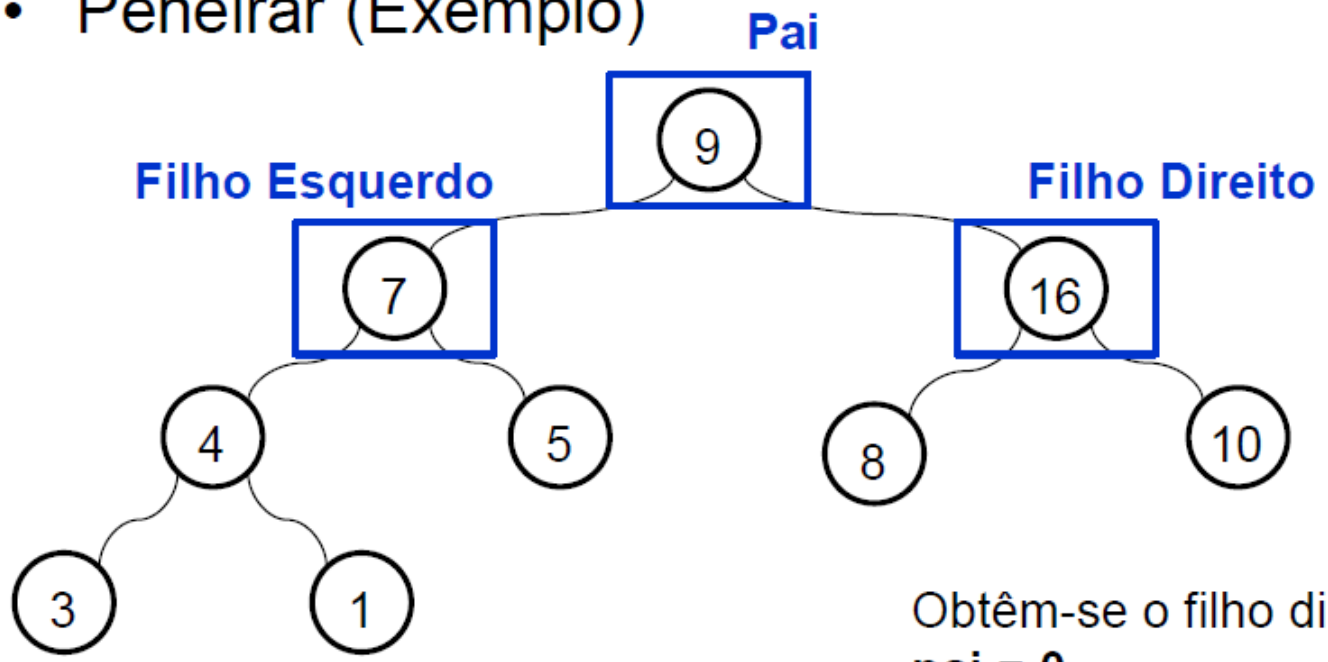


Observe que a propriedade estrutural de Heap foi violada no ponto destacado. Deste modo, invoca-se **Peneirar** para **pai = 0**

0	1	2	3	4	5	6	7	8
9	7	16	4	5	8	10	3	1

Peneirar

- Peneirar (Exemplo)

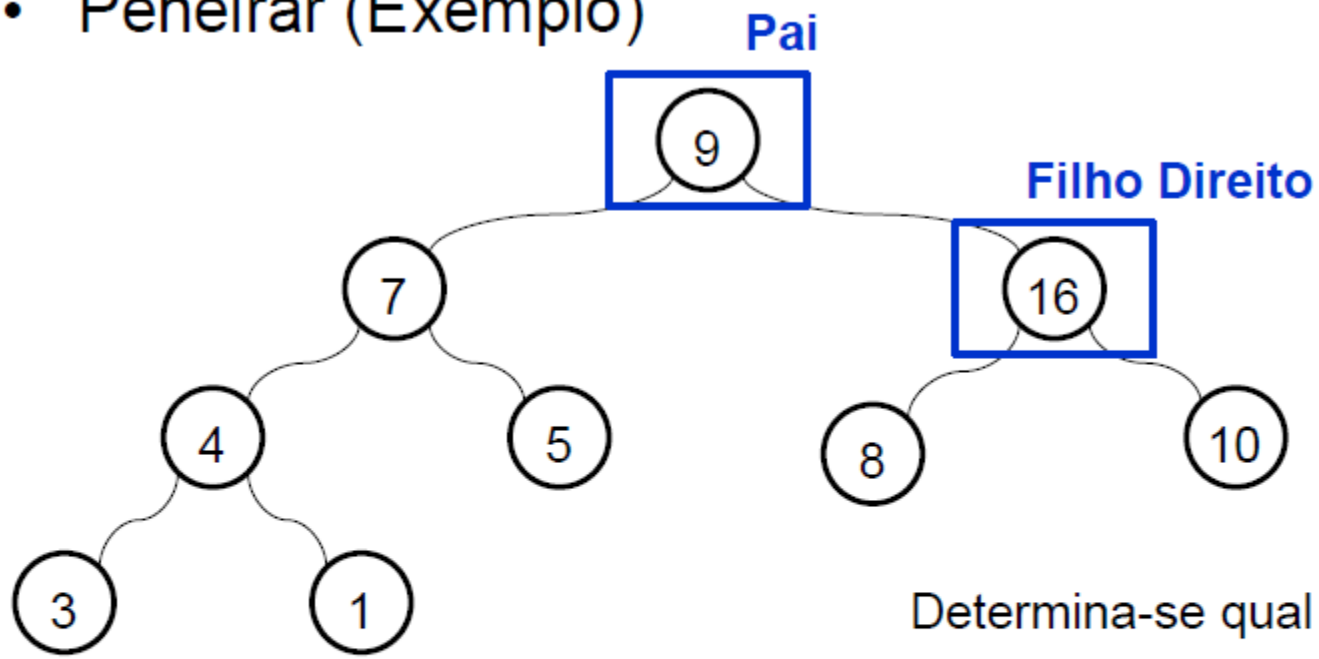


Obtêm-se o filho direito e esquerdo de **pai = 0**

0	1	2	3	4	5	6	7	8
9	7	16	4	5	8	10	3	1

Peneirar

- Peneirar (Exemplo)

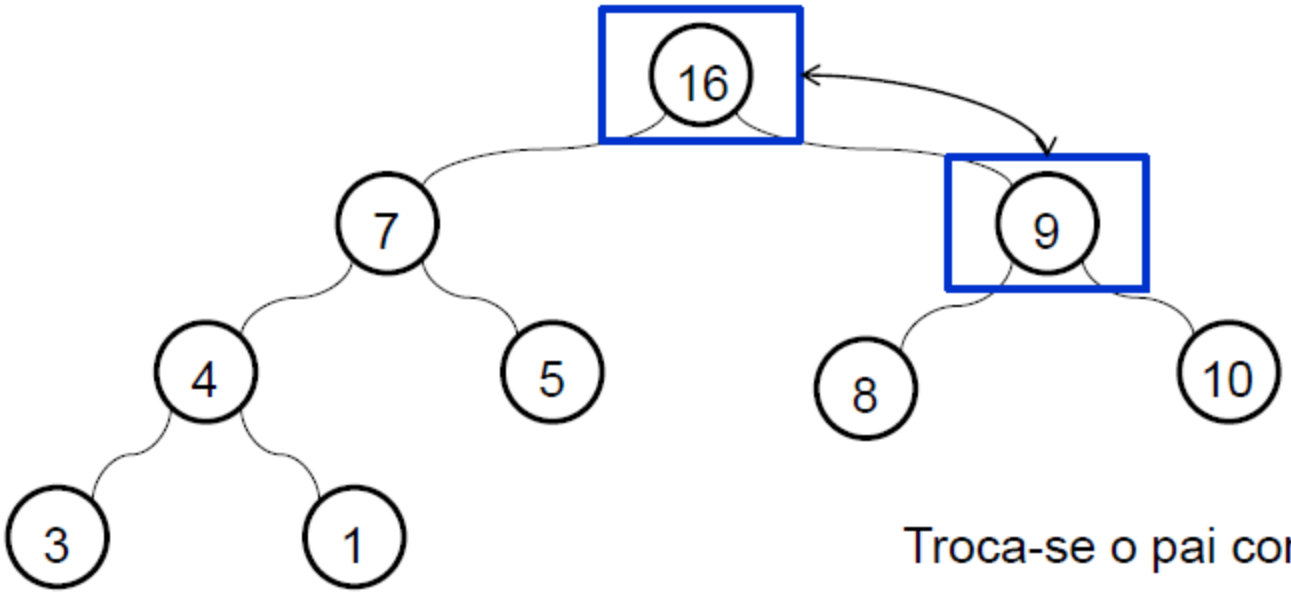


Determina-se qual dos filhos é maior.

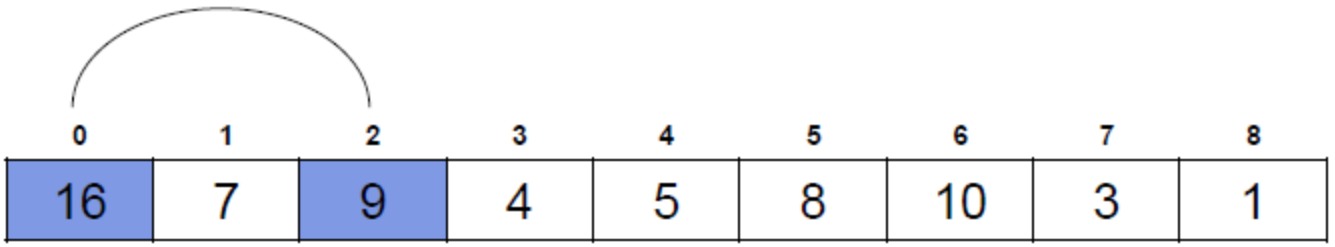
0	1	2	3	4	5	6	7	8
9	7	16	4	5	8	10	3	1

Peneirar

- Peneirar (Exemplo)

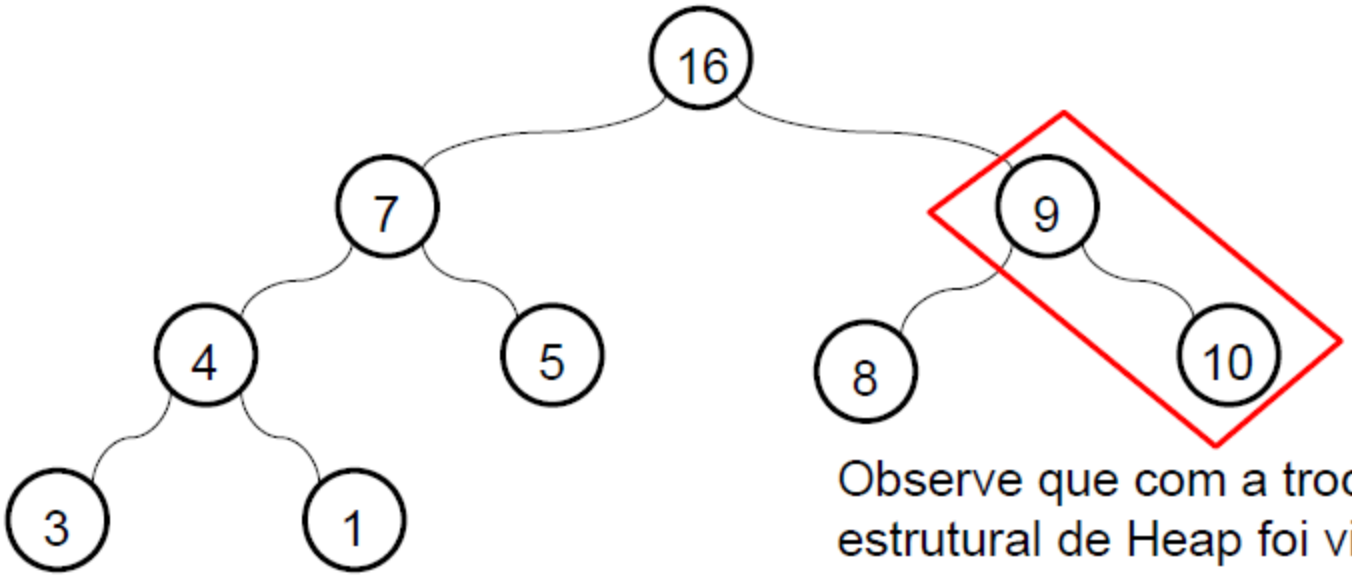


Troca-se o pai com o maior filho.



Peneirar

- Peneirar (Exemplo)

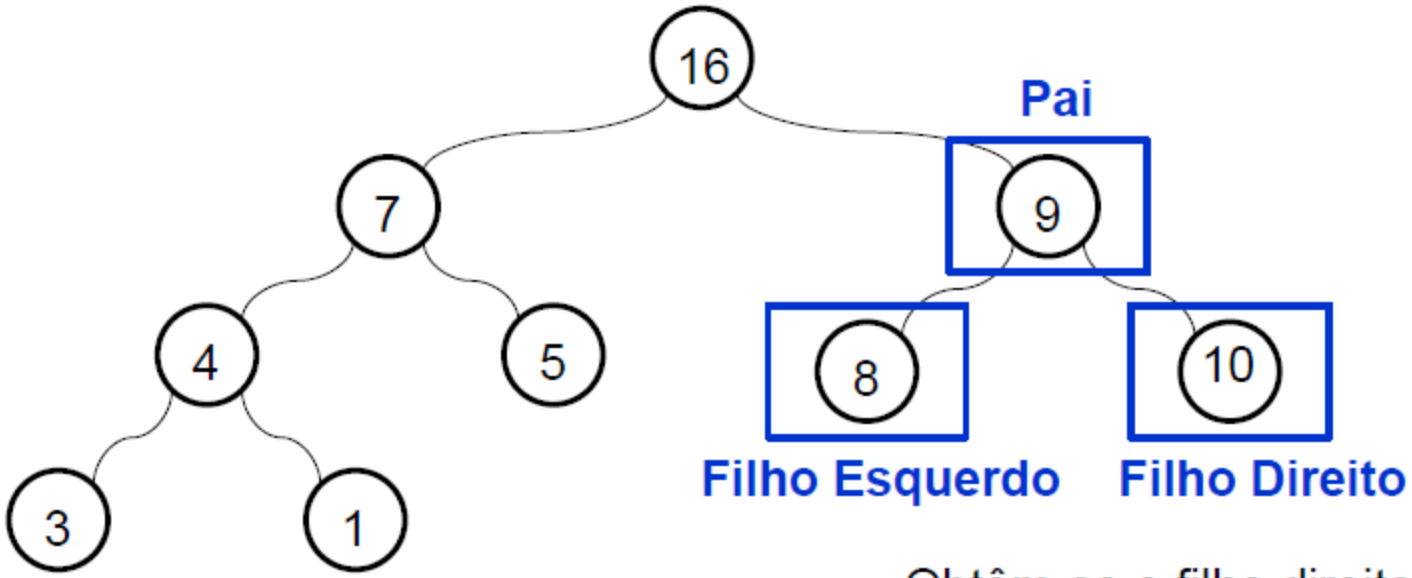


Observe que com a troca, propriedade estrutural de Heap foi violada exatamente na subárvore que promoveu a troca. Deste modo, invoca-se **Peneirar** com pai igual onde ocorreu a troca, no caso, em **pai** = **filhoDireito** = **2**

0	1	2	3	4	5	6	7	8
16	7	9	4	5	8	10	3	1

Peneirar

- Peneirar (Exemplo)

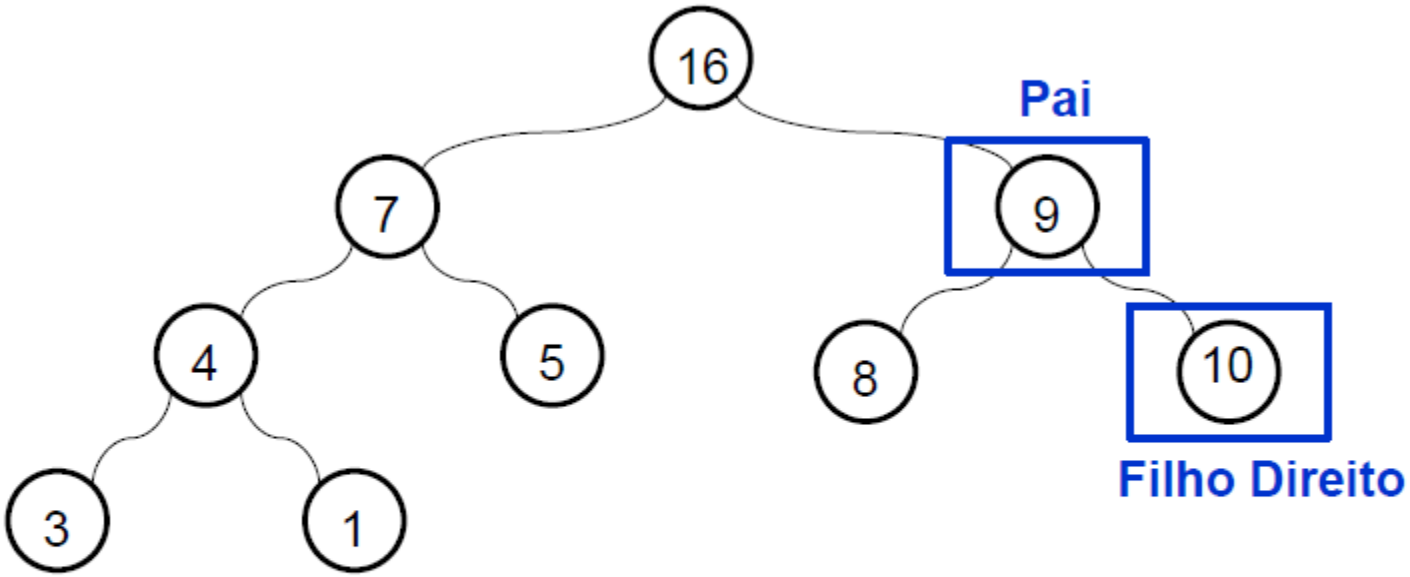


Obtêm-se o filho direito e esquerdo de **pai = 2**

0	1	2	3	4	5	6	7	8
16	7	9	4	5	8	10	3	1

Peneirar

- Peneirar (Exemplo)

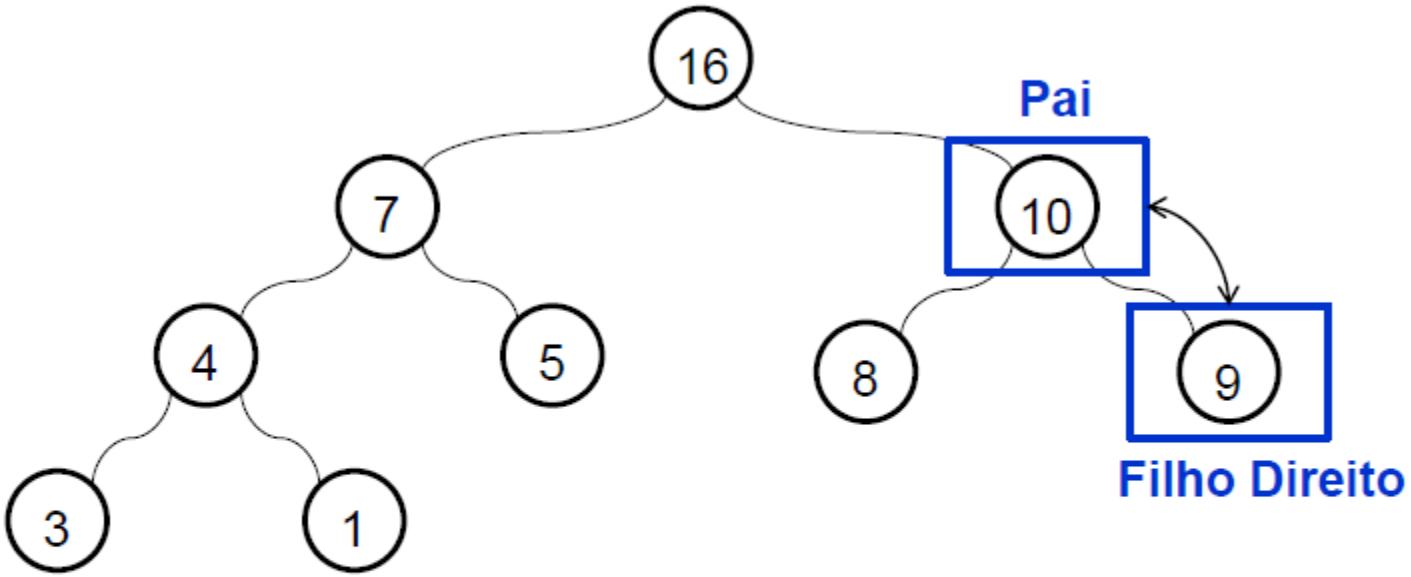


Determina-se qual dos filhos é maior.

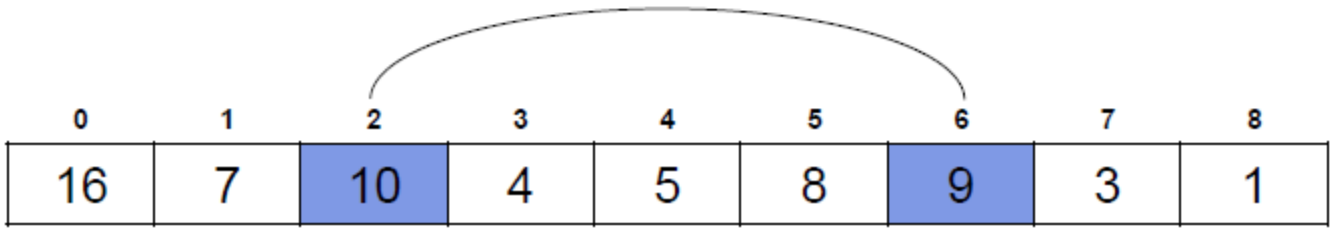
0	1	2	3	4	5	6	7	8
16	7	9	4	5	8	10	3	1

Peneirar

- Peneirar (Exemplo)

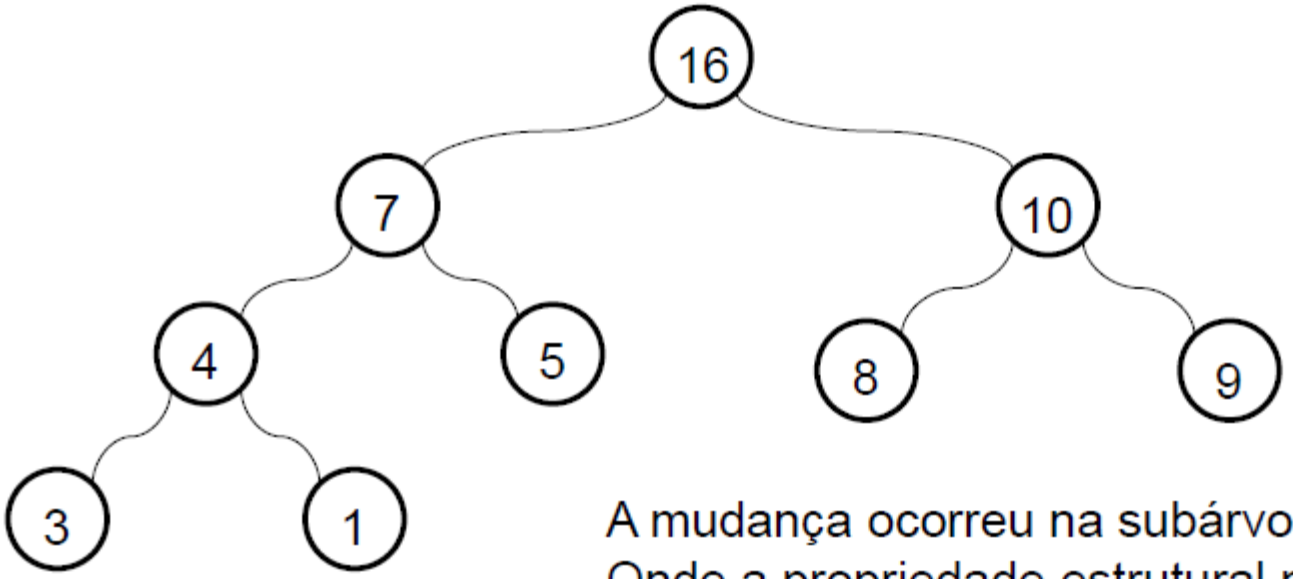


Determina-se qual dos filhos é maior.



Peneirar

- Peneirar (Exemplo)

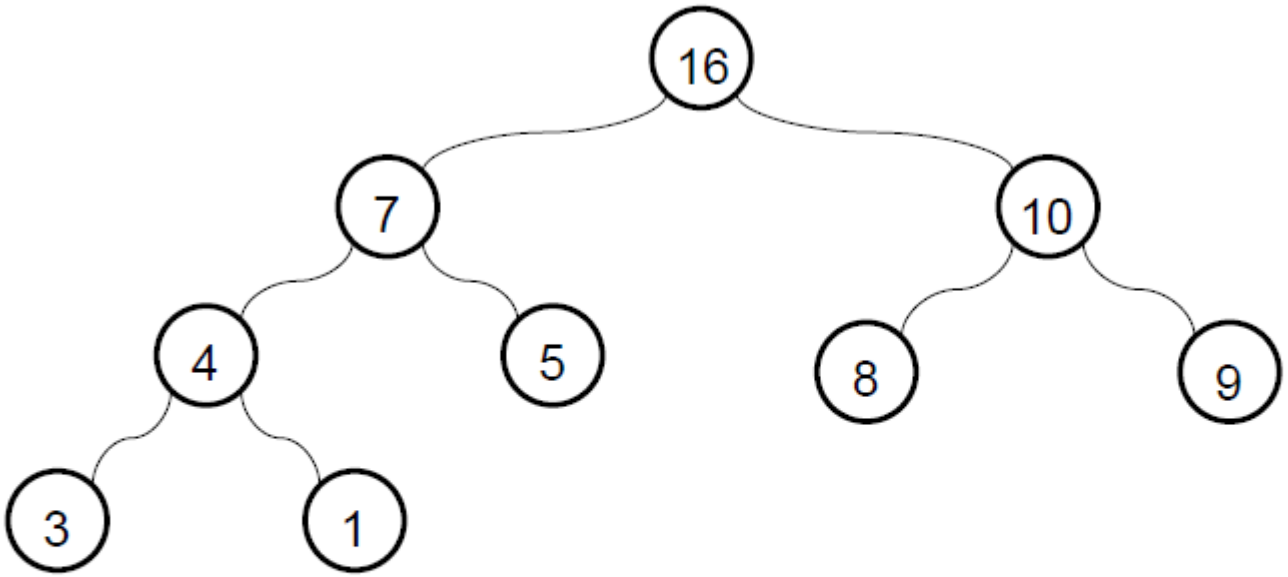


A mudança ocorreu na subárvore do **filhoDireito** = **6**. Onde a propriedade estrutural poderia ser violada novamente, então invoca-se **Peneirar** para **pai** = **FilhoDireito** = **6**, o que não resultará em nova chamada, pois o nó **6** é uma folha.

0	1	2	3	4	5	6	7	8
16	7	10	4	5	8	9	3	1

Peneirar

- Peneirar (Exemplo)



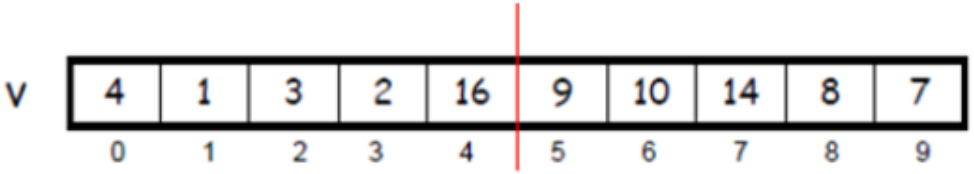
O que significa que o vetor resultante é um Heap!

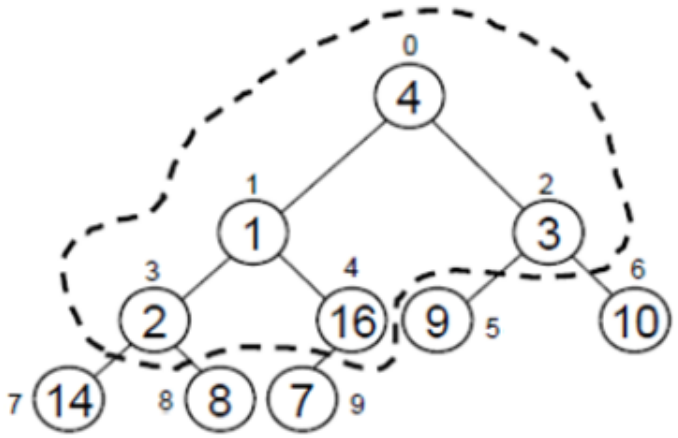
0	1	2	3	4	5	6	7	8
16	7	10	4	5	8	9	3	1

Contrução de um HEAP

Construir um Heap a partir de um vetor qualquer:

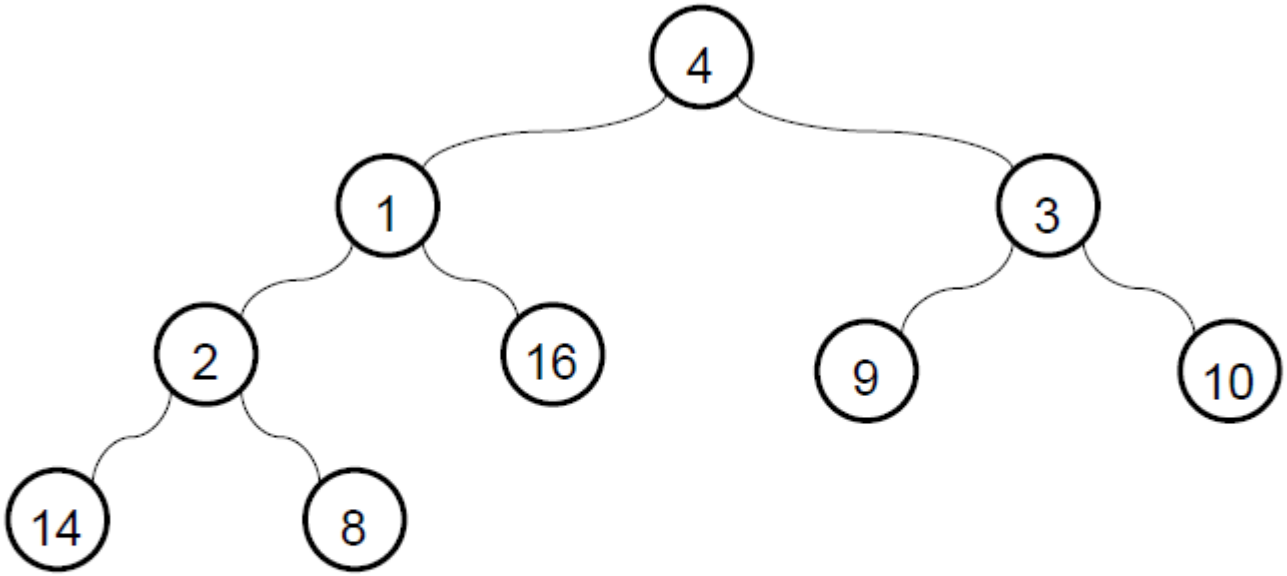
- O algoritmo Construir transforma um vetor qualquer em um heap.
- Como os índices **i**, $\lfloor n/2 \rfloor \leq i < n$, são folhas, basta aplicar **Peneirar** entre as posições 0 e $\lfloor n/2 \rfloor - 1$, ou seja em todos os nós que são pais.





Contrução de um HEAP

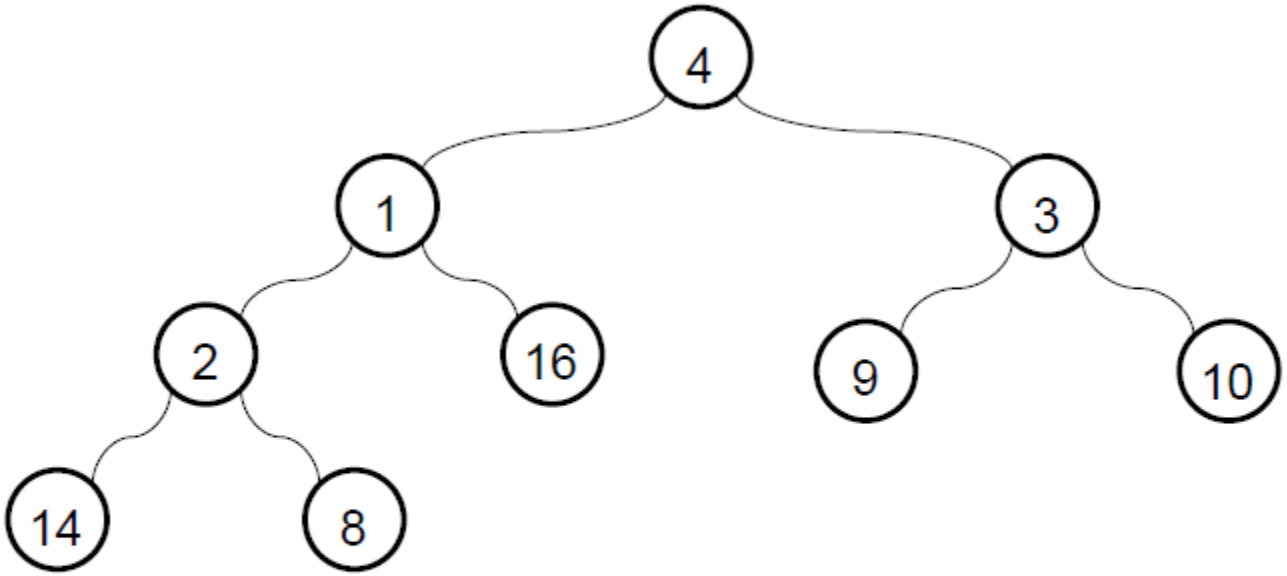
- Construir (Exemplo)



0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

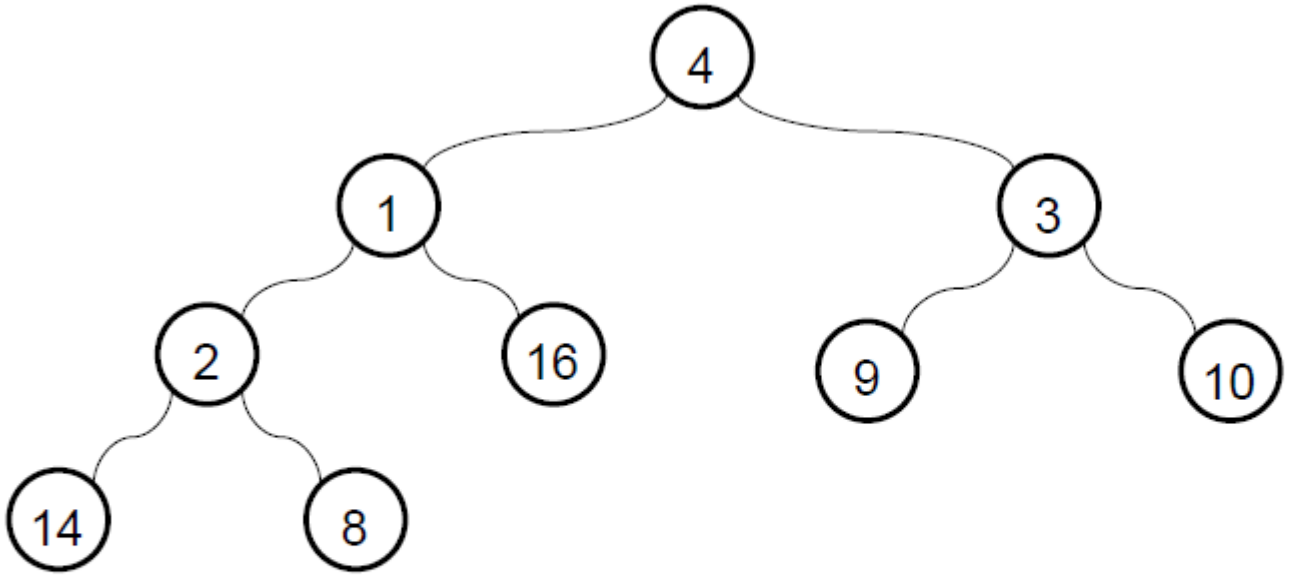


O que fazer?

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

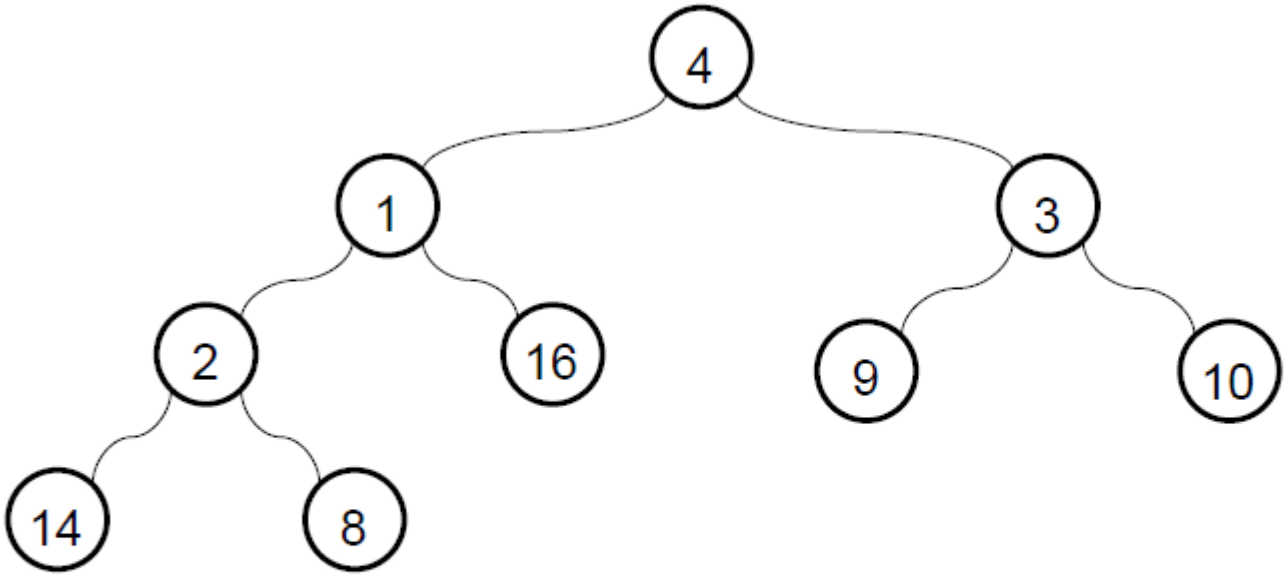


O que fazer?
Aplica-se Peneirar em todos os nós pais.

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

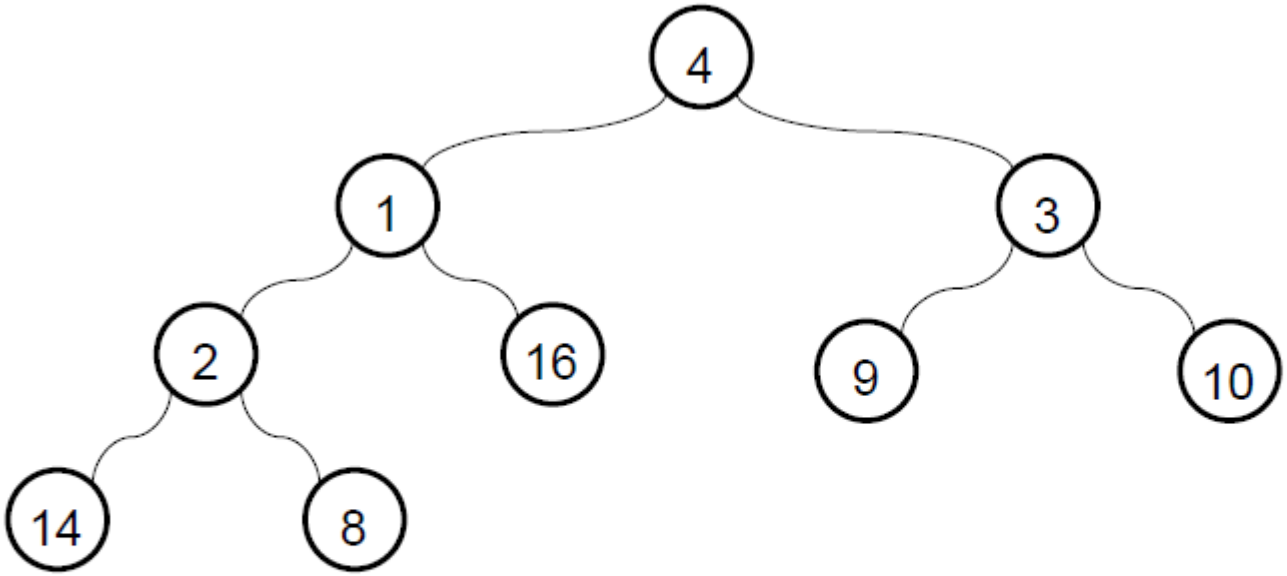


Começando onde ?

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)



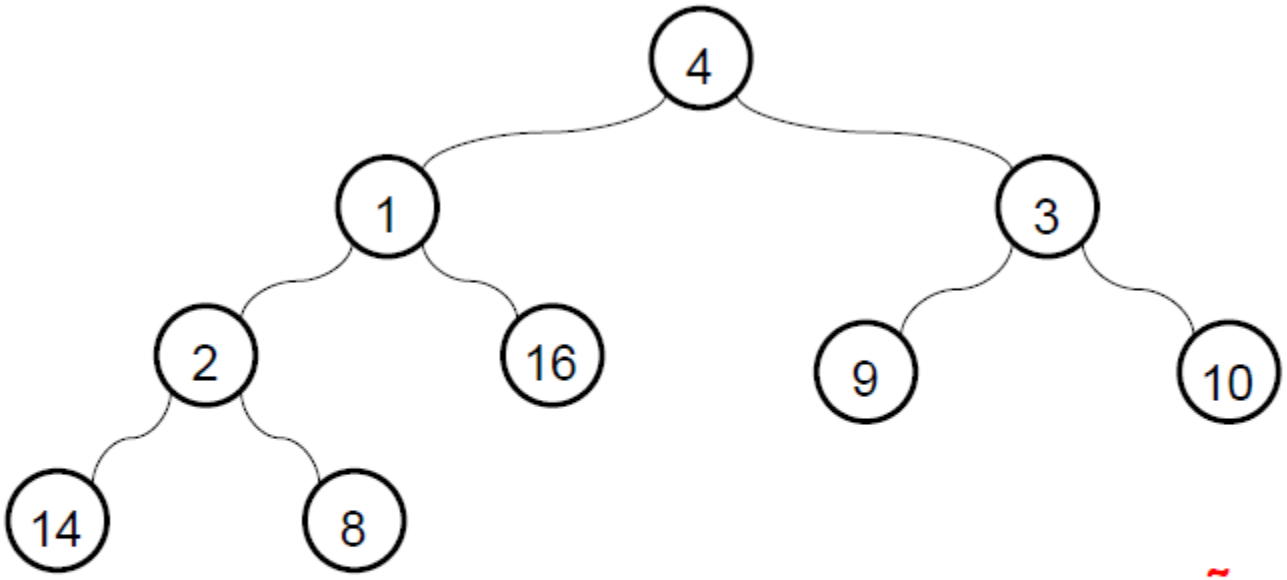
Começando onde ?

Na Raiz!

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

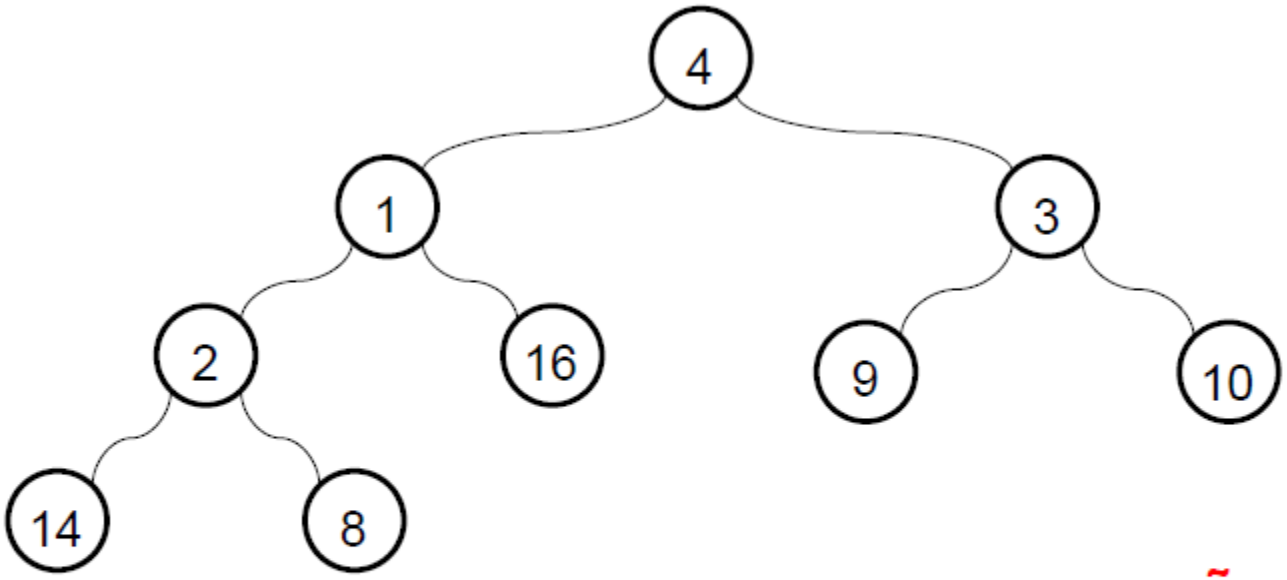


NÃO!!!

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

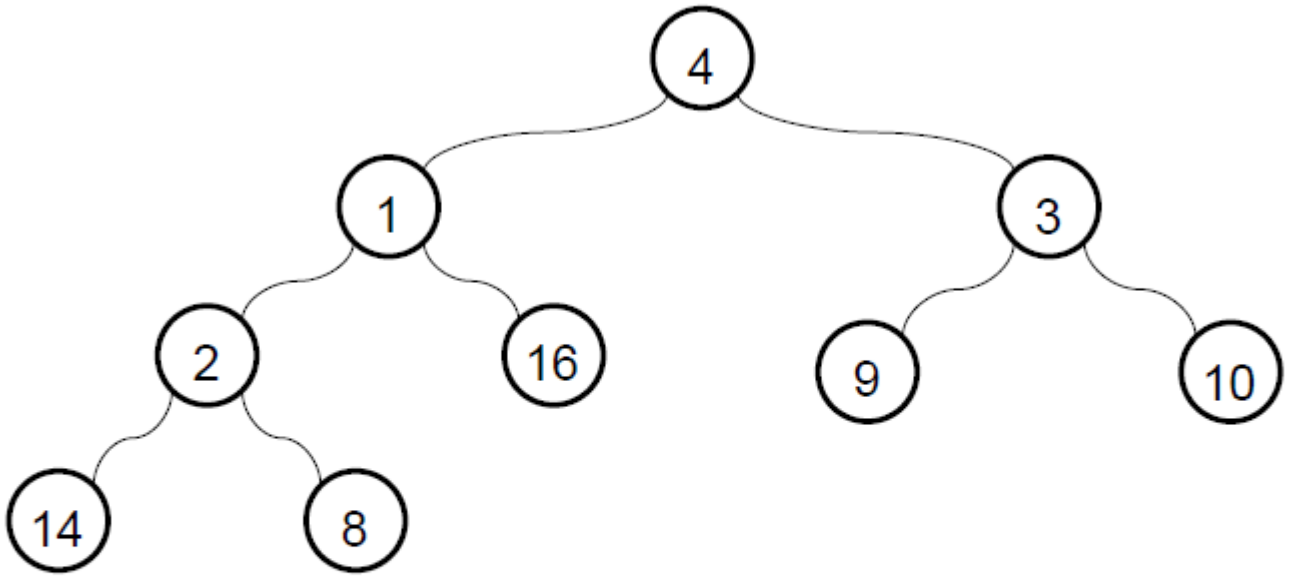


NÃO!!!
POR QUE?

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

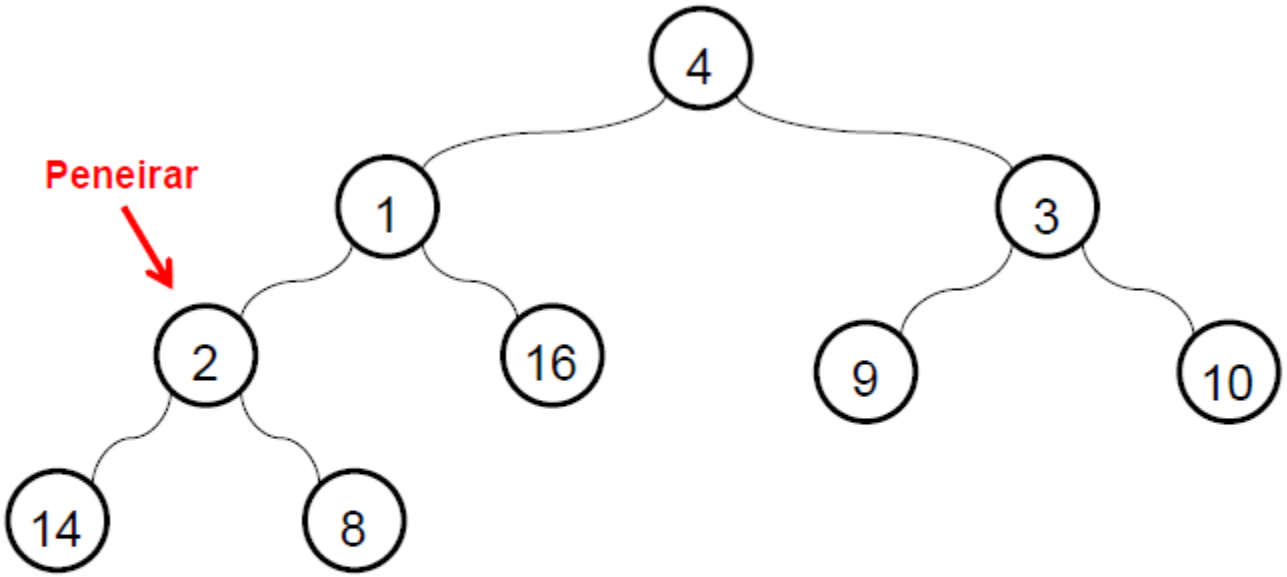


Parte-se do **Último Pai** = **3**, até a raiz invocando-se a função **Peneirar**.

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

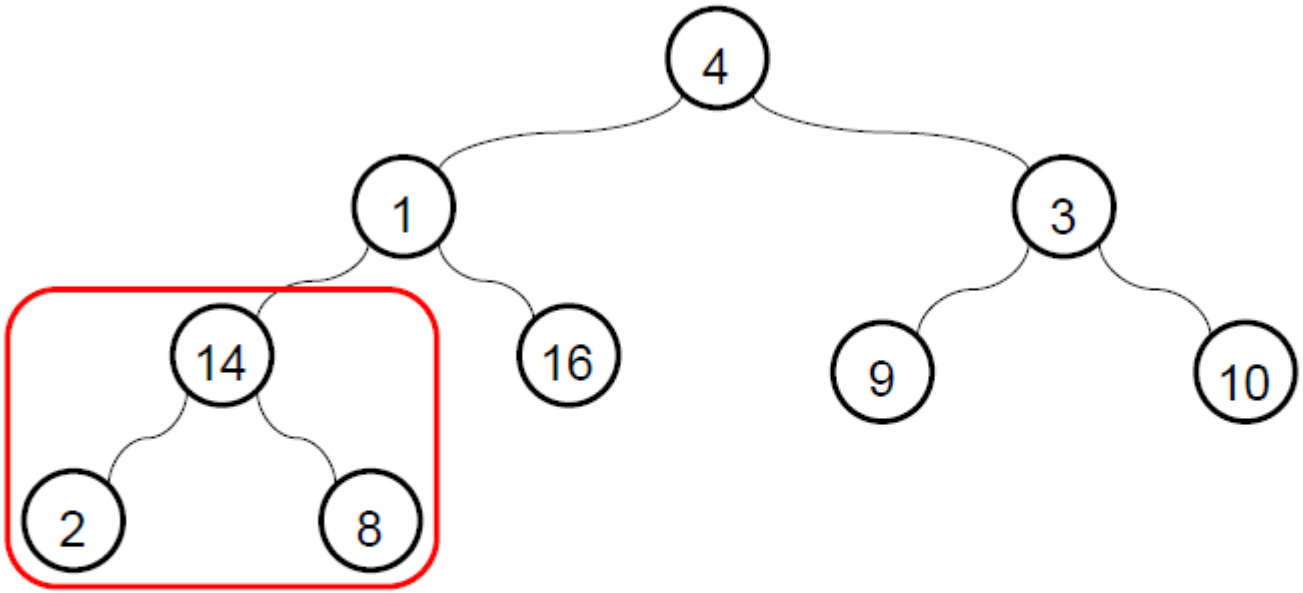


Invoca-se **Peneirar** para **pai** = **3**.

0	1	2	3	4	5	6	7	8
4	1	3	2	16	9	10	14	8

Contrução de um HEAP

- Construir (Exemplo)

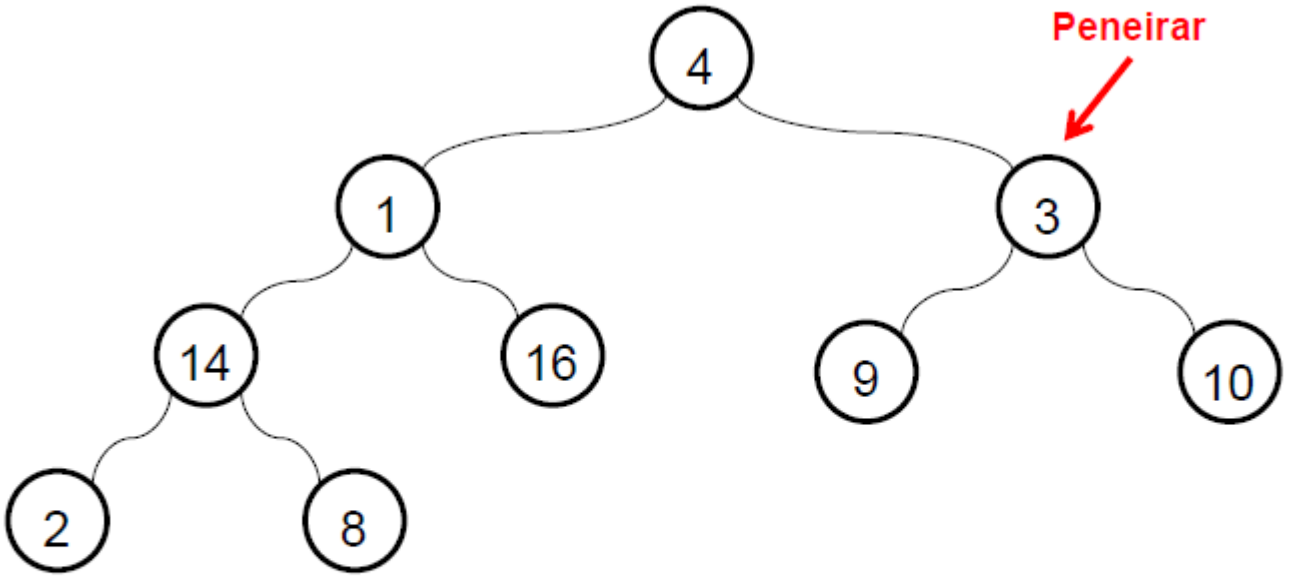


Invoca-se **Peneirar** para **pai** = **3**, como resultado a subárvore será um heap.

0	1	2	3	4	5	6	7	8
4	1	3	14	16	9	10	2	8

Contrução de um HEAP

- Construir (Exemplo)

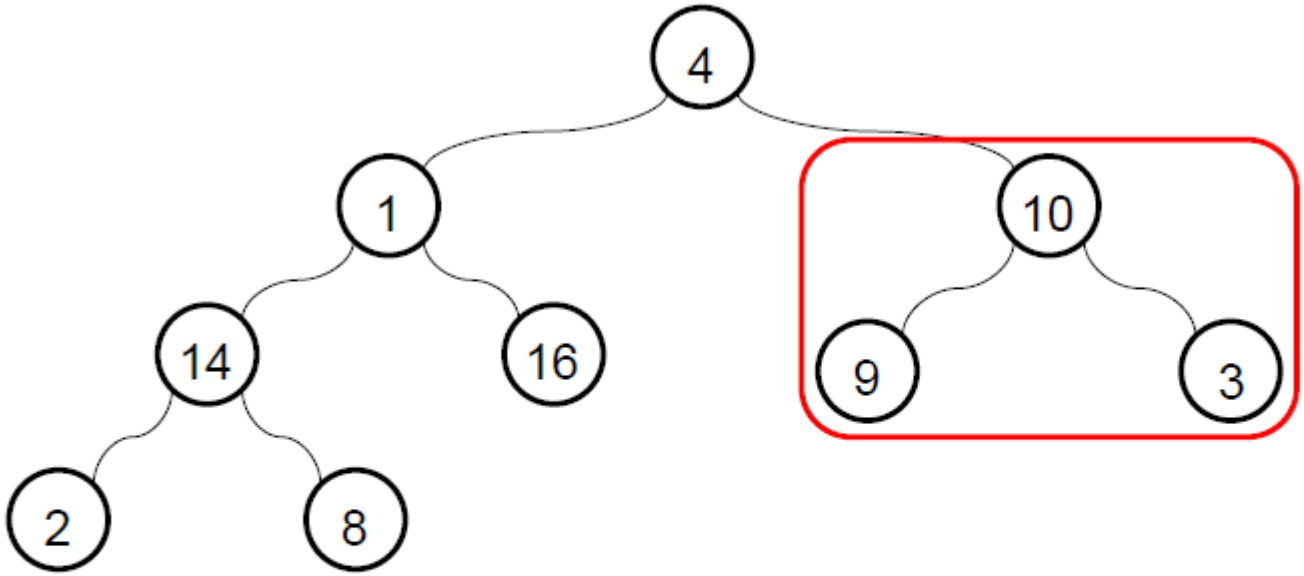


Invoca-se **Peneirar** para **pai** = **2**.

0	1	2	3	4	5	6	7	8
4	1	3	14	16	9	10	2	8

Contrução de um HEAP

- Construir (Exemplo)

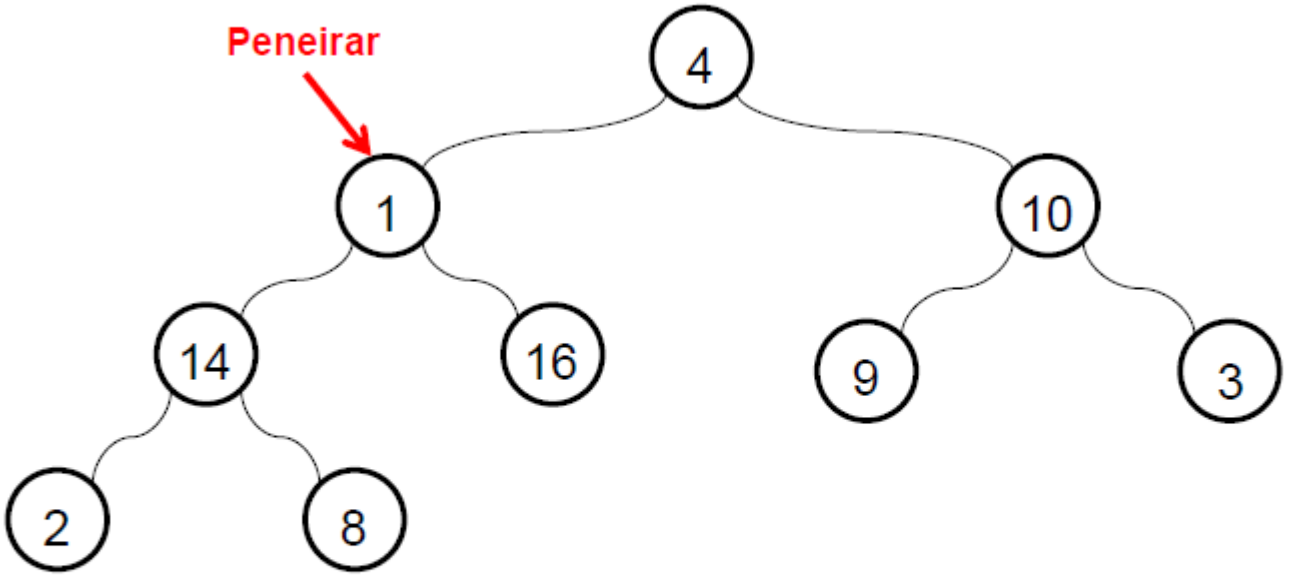


Invoca-se **Peneirar** para **pai** = **2**, como resultado a subárvore será um heap.

0	1	2	3	4	5	6	7	8
4	1	10	14	16	9	3	2	8

Contrução de um HEAP

- Construir (Exemplo)

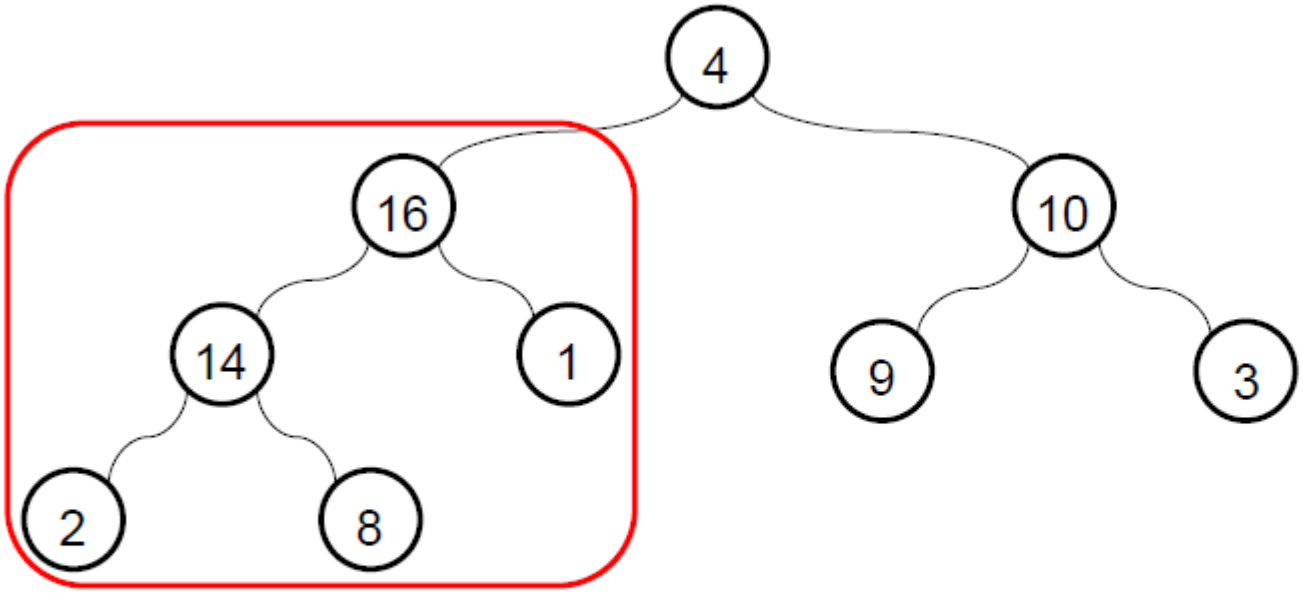


Invoca-se **Peneirar** para **pai** = **1**.

0	1	2	3	4	5	6	7	8
4	1	10	14	16	9	3	2	8

Contrução de um HEAP

- Construir (Exemplo)

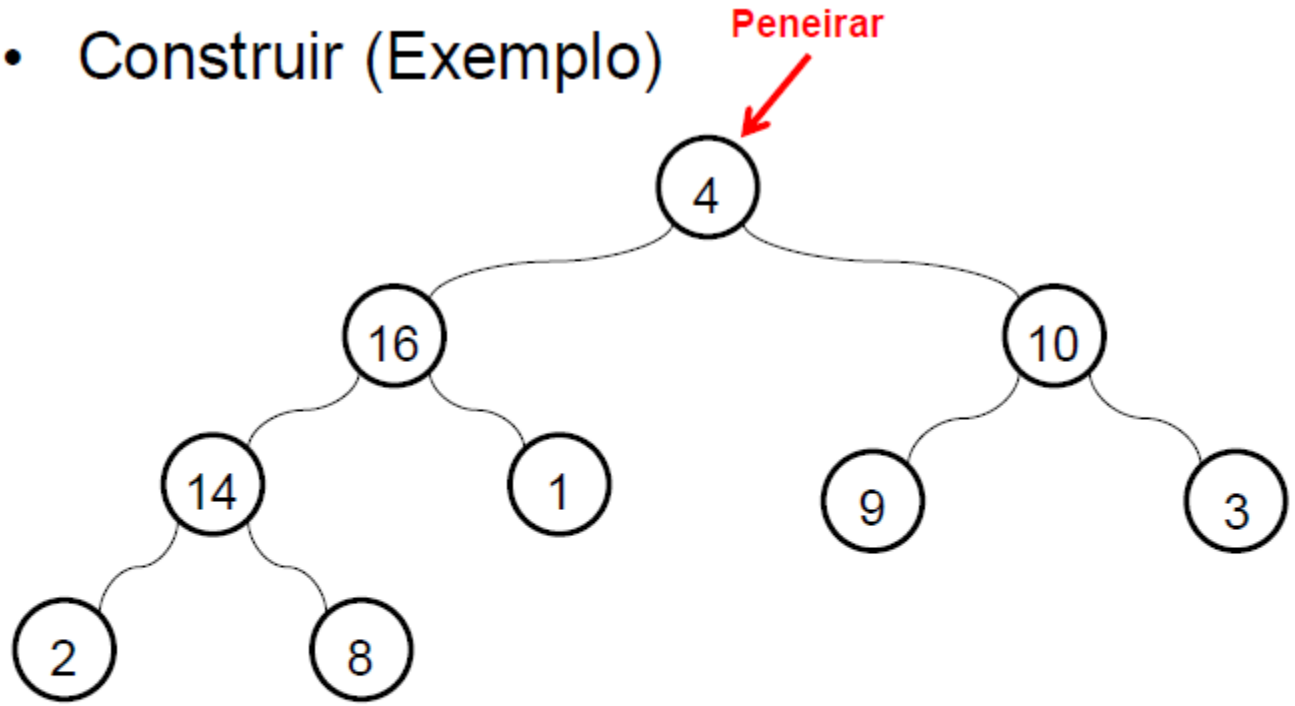


Invoca-se **Peneirar** para **pai** = **1**, como resultado a subárvore será um heap.

0	1	2	3	4	5	6	7	8
4	16	10	14	1	9	3	2	8

Contrução de um HEAP

- Construir (Exemplo)

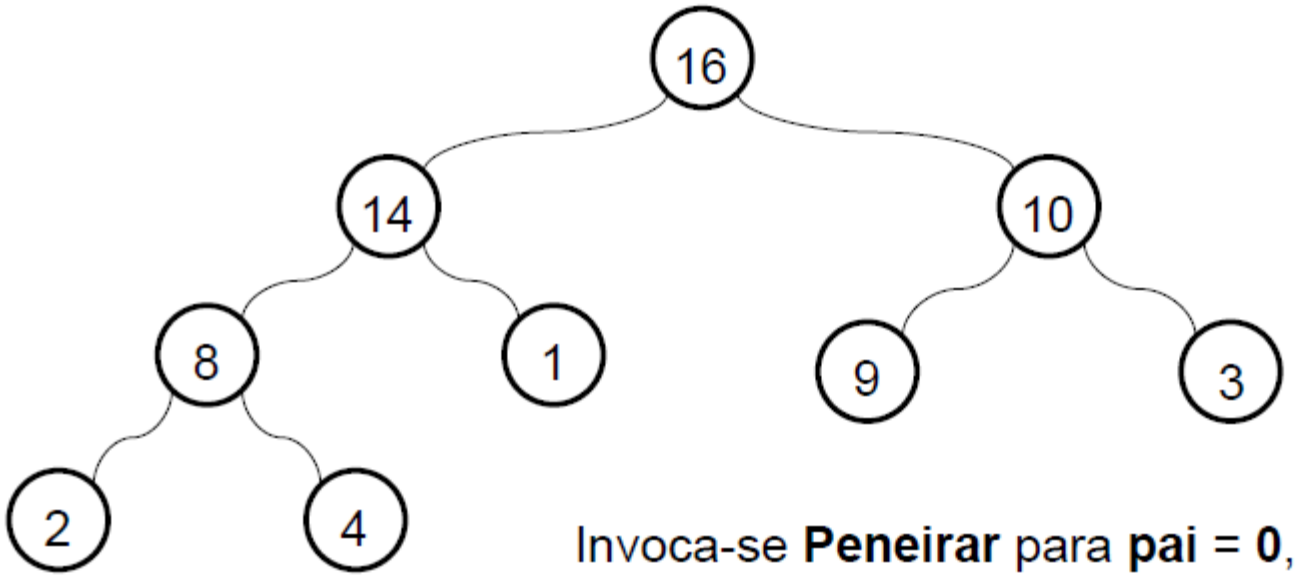


Invoca-se **Peneirar** para **pai** = **0**.

0	1	2	3	4	5	6	7	8
4	16	10	14	1	9	3	2	8

Contrução de um HEAP

- Construir (Exemplo)



Invoca-se **Peneirar** para **pai** = **0**, como resultado a subárvore (no caso toda a árvore) será um heap.

** Observe que Peneirar será invocado recursivamente mais 2 vezes.*

0	1	2	3	4	5	6	7	8
16	14	10	8	1	9	3	2	4

Operações Básicas

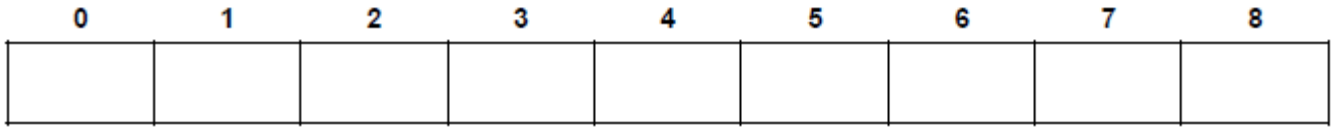
- Operações Principais:
  - Incluir Item no Heap
  - Remover Item no Heap (remover o máximo)

- Operações Secundárias
  - Peneirar
  - Construir (obtém heap a partir de qualquer vetor)

Inclusão no HEAP

- Inclusão:

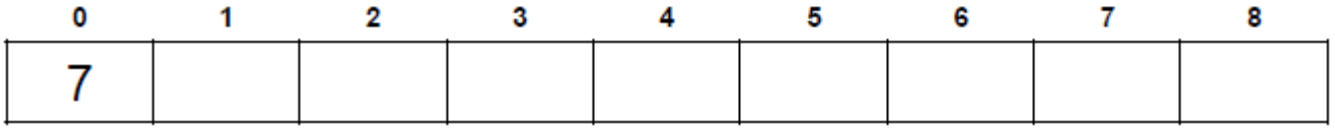
Uma árvore com um único nó já é automaticamente um Heap.



Inclusão no HEAP

- Inclusão:

Uma árvore com um único nó já é automaticamente um Heap.



Inclusão no HEAP

- Inclusão:

Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
7								

Inclusão no HEAP

- Inclusão:

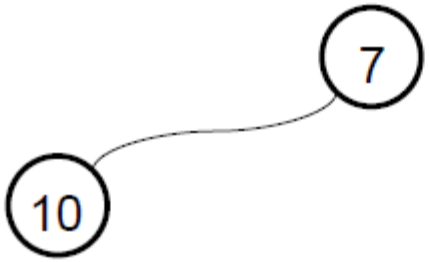
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
7								

Inclusão no HEAP

- Inclusão:



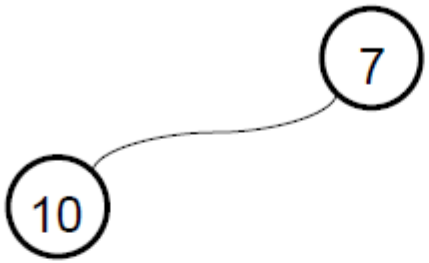
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
7	10							

Inclusão no HEAP

- Inclusão:



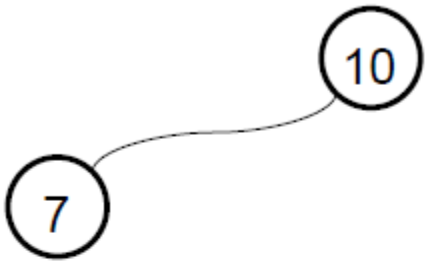
Procedimento para adição de novos nós a um heap:

- Se houve violação da estrutura de heap, então
  - Invoque a **Construir** para o heap.

0	1	2	3	4	5	6	7	8
7	10							

Inclusão no HEAP

- Inclusão:



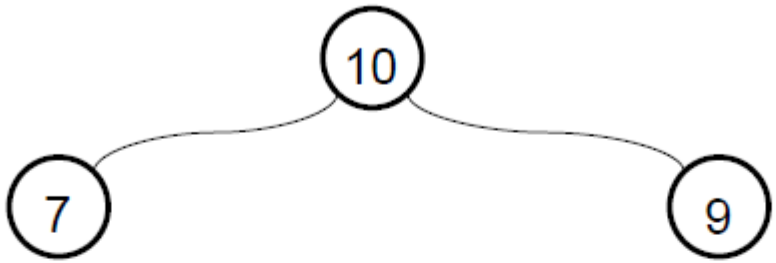
Procedimento para adição de novos nós a um heap:

- Após a chamada por **Construir**, obter-se-á um heap.

0	1	2	3	4	5	6	7	8
10	7							

Inclusão no HEAP

- Inclusão:



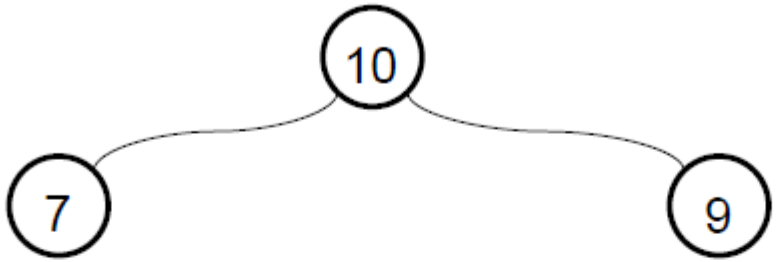
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
10	7	9						

Inclusão no HEAP

- Inclusão:



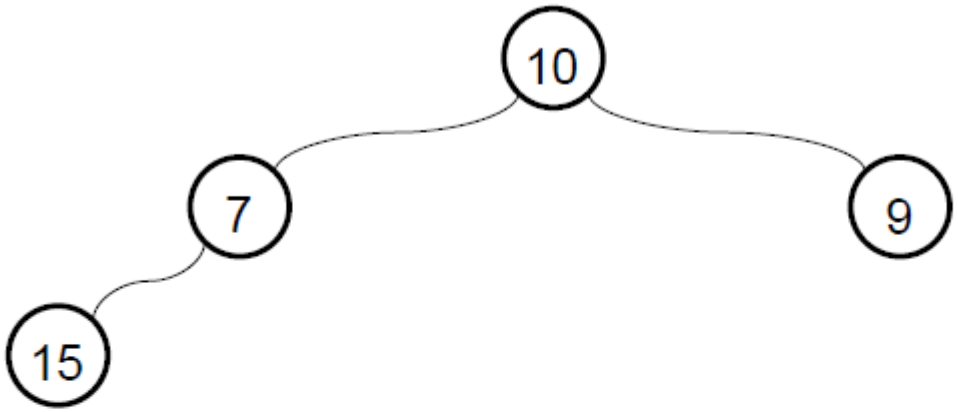
Procedimento para adição de novos nós a um heap:

- Se houve violação da estrutura de heap, então
  - Invoque a **Construir** para o heap.

0	1	2	3	4	5	6	7	8
10	7	9						

Inclusão no HEAP

- Inclusão:



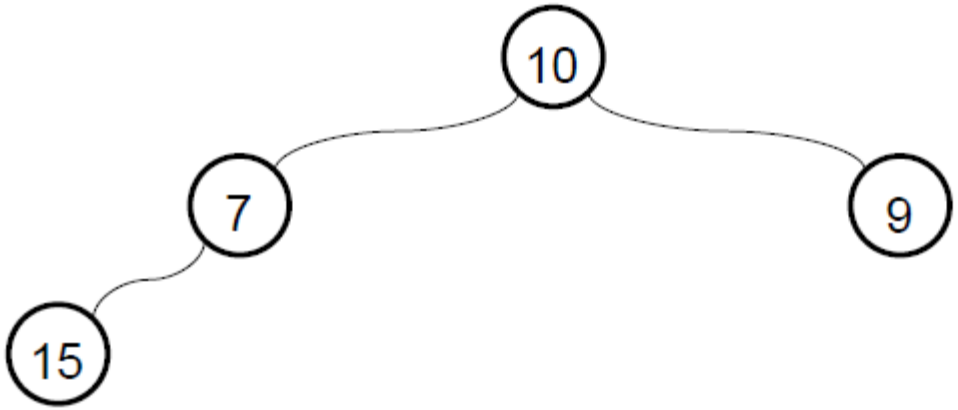
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
10	7	9	15					

Inclusão no HEAP

- Inclusão:



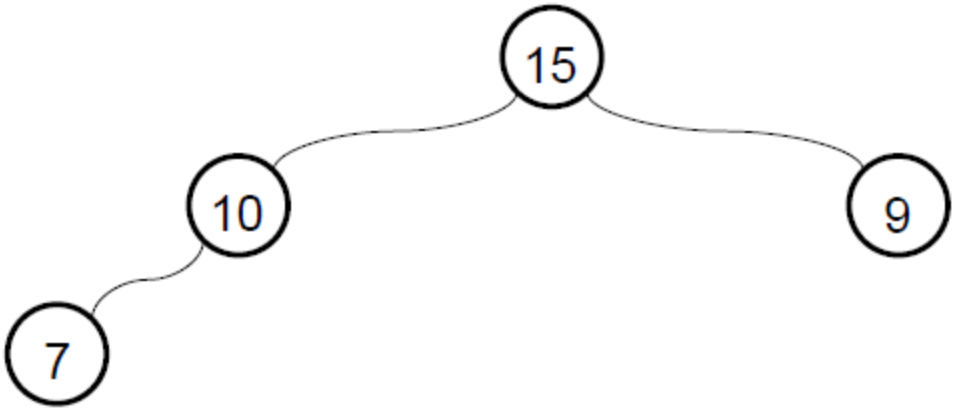
Procedimento para adição de novos nós a um heap:

- Se houve violação da estrutura de heap, então
  - Invoque a **Construir** para o heap.

0	1	2	3	4	5	6	7	8
10	7	9	15					

Inclusão no HEAP

- Inclusão:



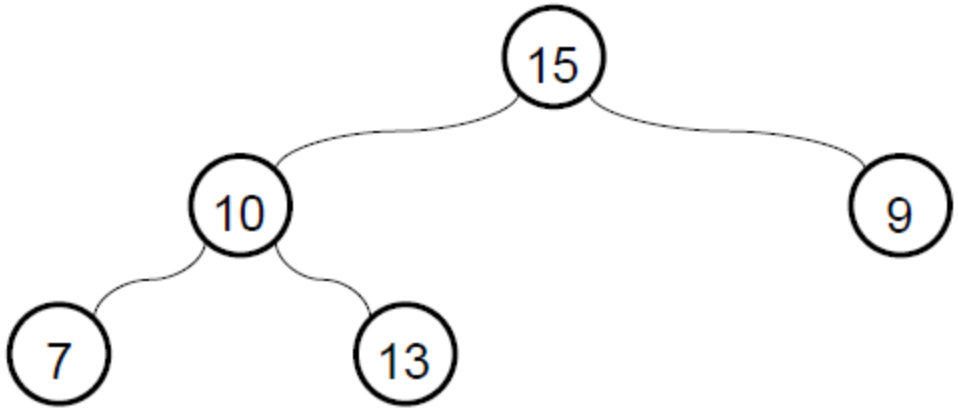
Procedimento para adição de novos nós a um heap:

- Após a chamada por **Construir**, obter-se-á um heap.

0	1	2	3	4	5	6	7	8
15	10	9	7					

Inclusão no HEAP

- Inclusão:



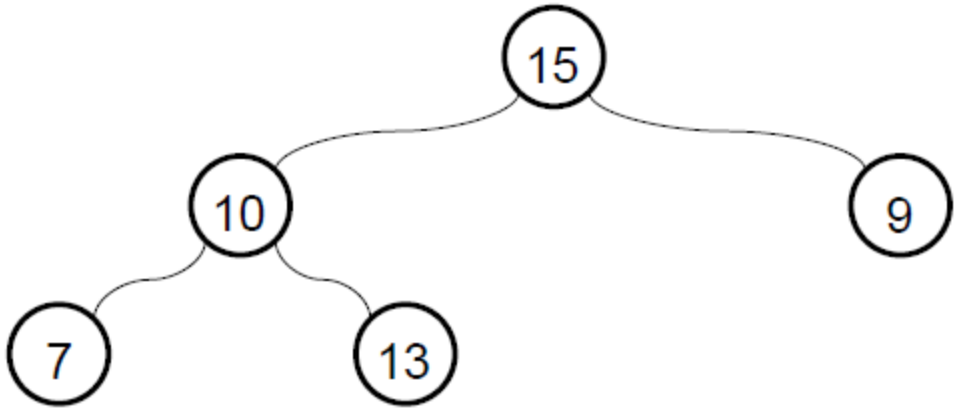
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
15	10	9	7	13				

Inclusão no HEAP

- Inclusão:



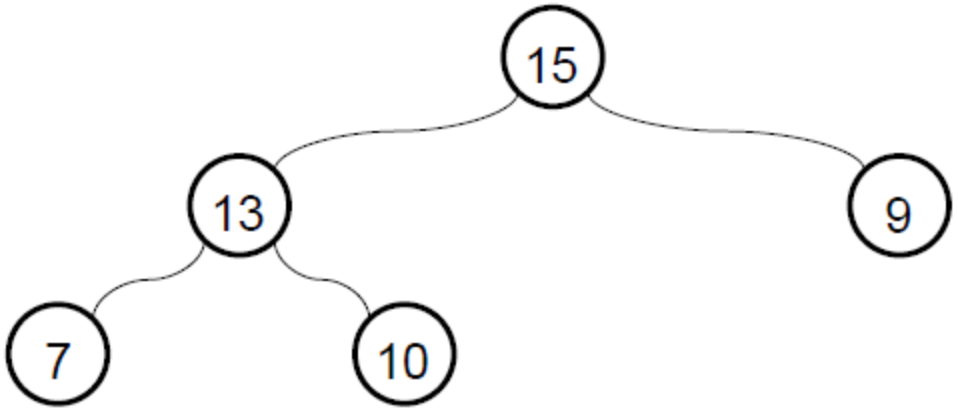
Procedimento para adição de novos nós a um heap:

- Se houve violação da estrutura de heap, então
  - Invoque a **Construir** para o heap.

0	1	2	3	4	5	6	7	8
15	10	9	7	13				

Inclusão no HEAP

- Inclusão:



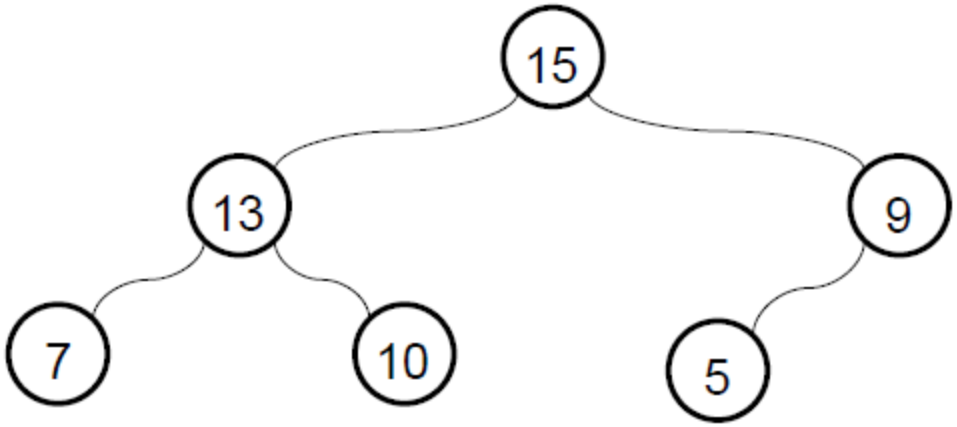
Procedimento para adição de novos nós a um heap:

- Após a chamada por **Construir**, obter-se-á um heap.

0	1	2	3	4	5	6	7	8
15	13	9	7	10				

Inclusão no HEAP

- Inclusão:



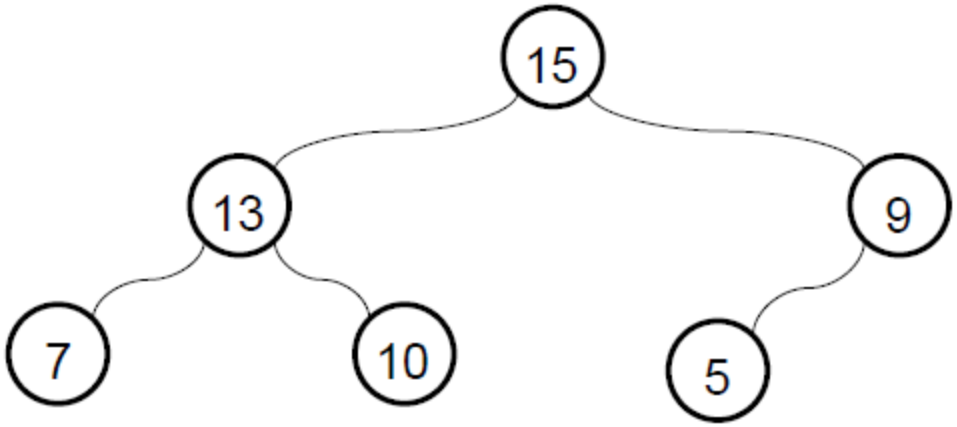
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5			

Inclusão no HEAP

- Inclusão:



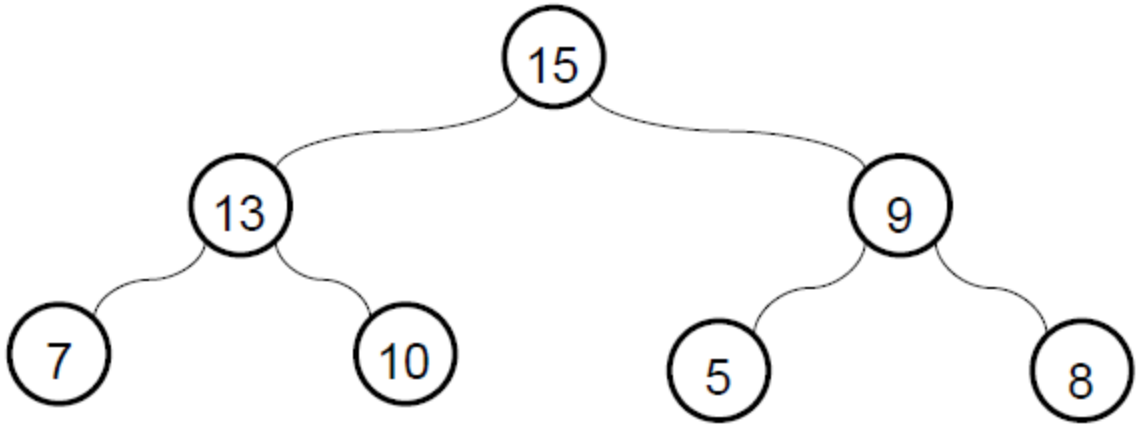
Procedimento para adição de novos nós a um heap:

- Se houve violação da estrutura de heap, então
  - Invoque a **Construir** para o heap.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5			

Inclusão no HEAP

- Inclusão:



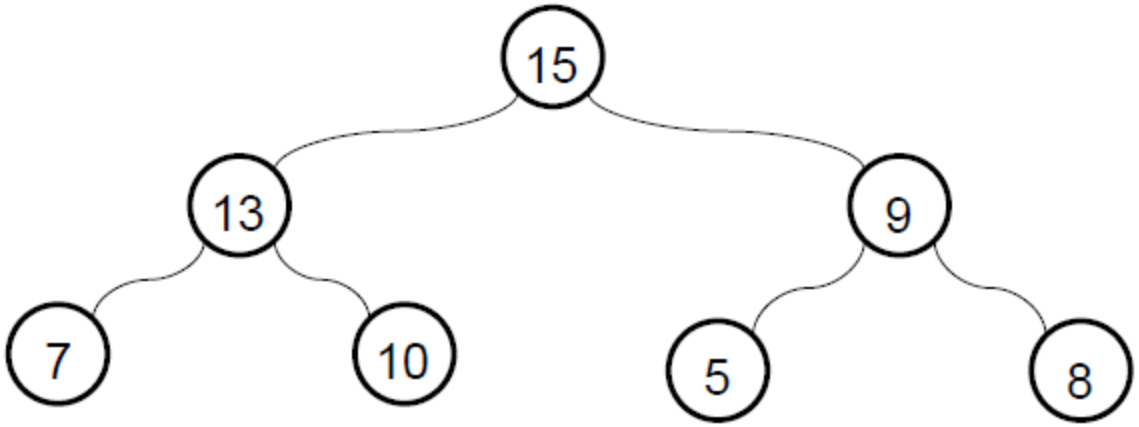
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5	8		

Inclusão no HEAP

- Inclusão:



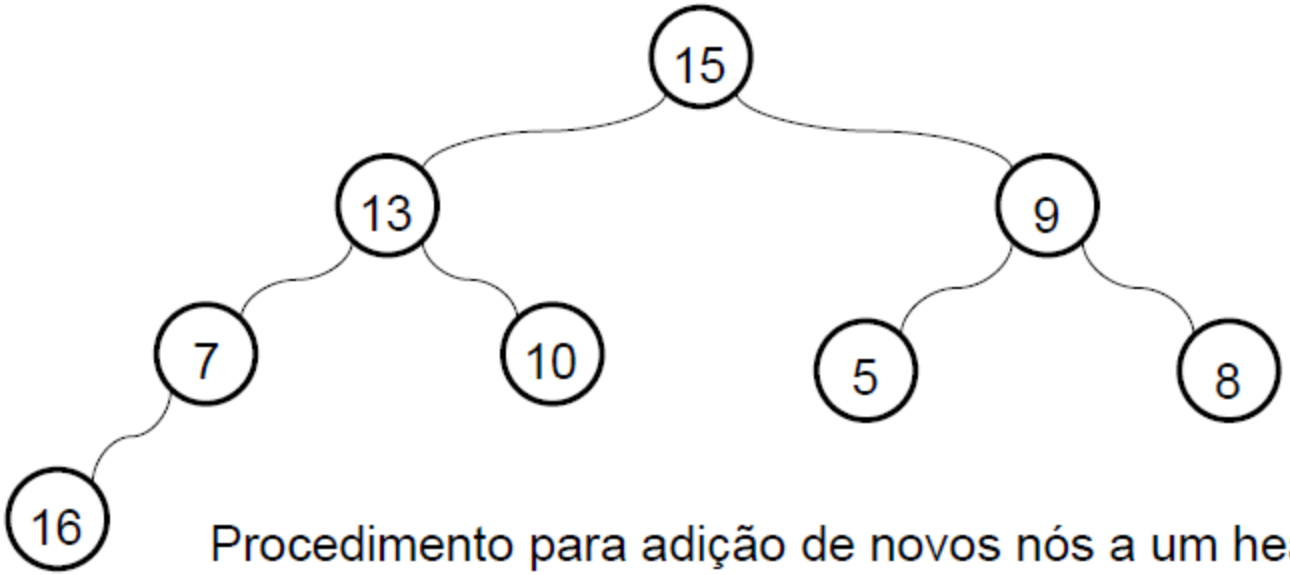
Procedimento para adição de novos nós a um heap:

- Se houve violação da estrutura de heap, então
  - Invoque a **Construir** para o heap.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5	8		

Inclusão no HEAP

- Inclusão:



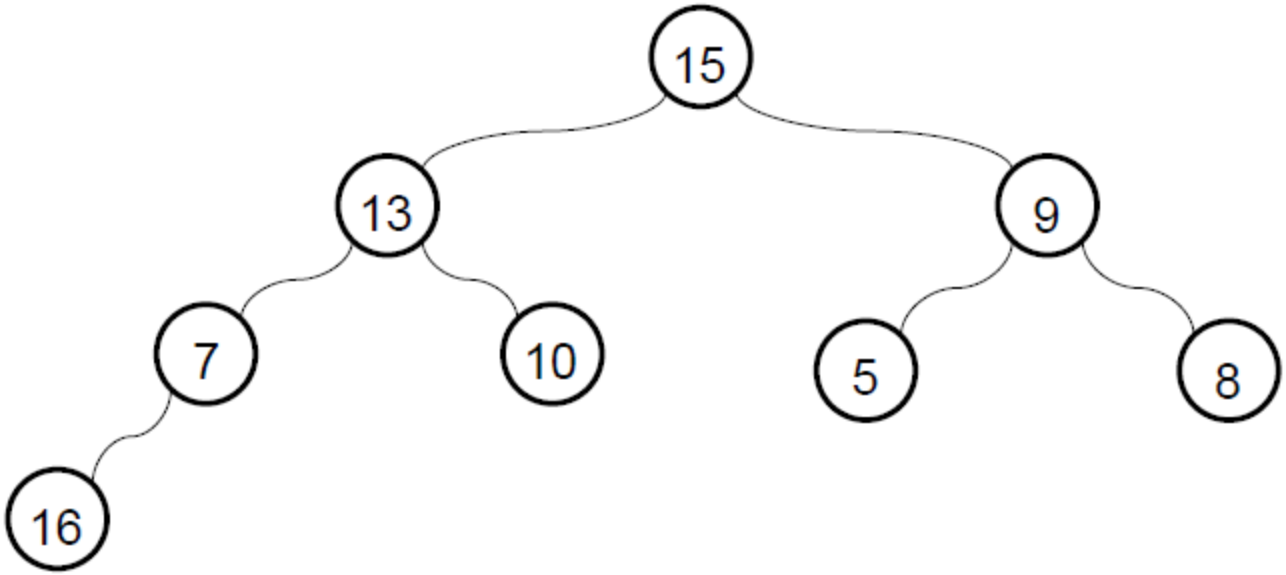
Procedimento para adição de novos nós a um heap:

- Se houver espaço no vetor, então
  - Deve ser folha no último nível, na primeira posição disponível mais à esquerda
  - Se este nível estiver cheio, comece um novo nível.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5	8	16	

Inclusão no HEAP

- Inclusão:



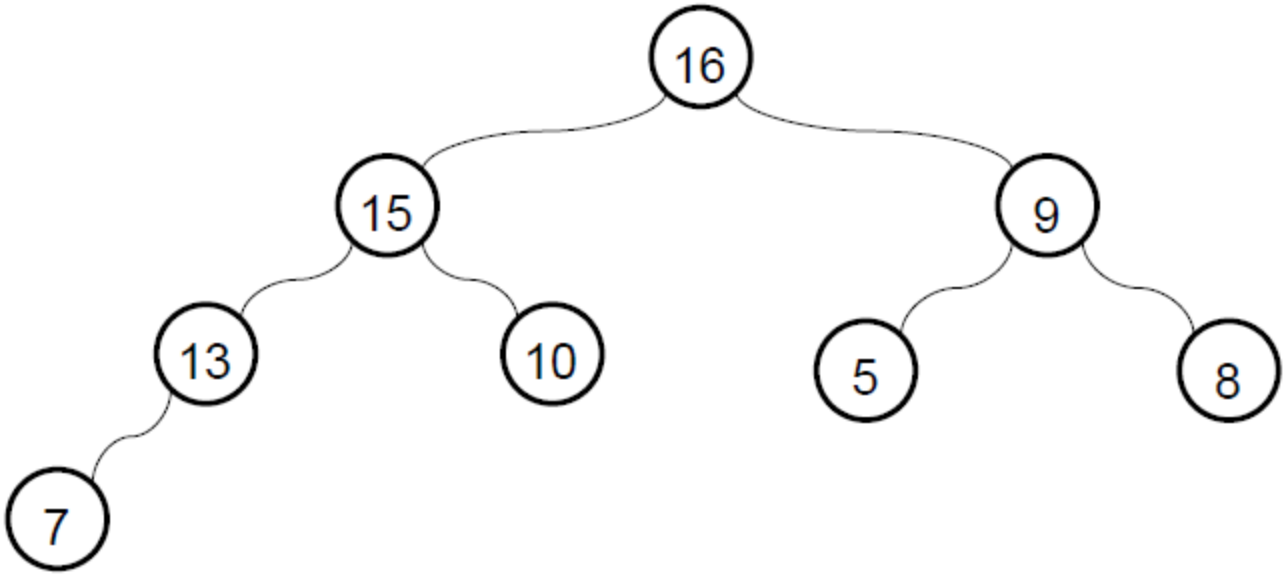
Procedimento para adição de novos nós a um heap:

- Se houve violação da estrutura de heap, então
  - Invoque a **Construir** para o heap.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5	8	16	

Inclusão no HEAP

- Inclusão:



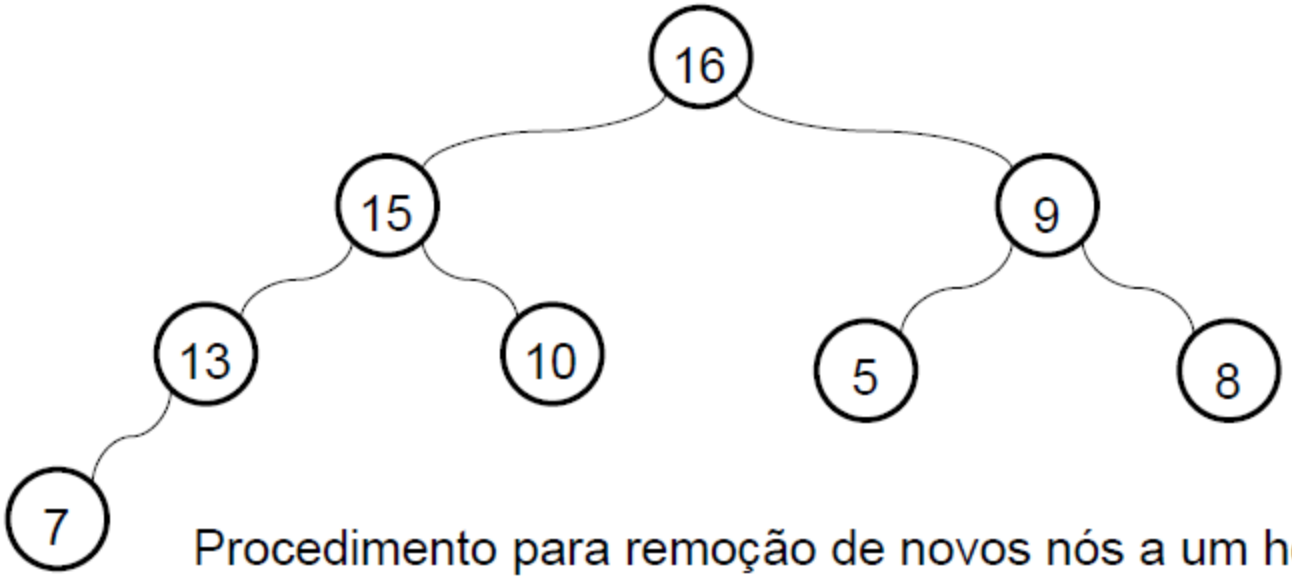
Procedimento para adição de novos nós a um heap:

- Após a chamada por **Construir**, obter-se-á um heap.

0	1	2	3	4	5	6	7	8
16	15	9	13	10	5	8	7	

Remoção do maior elemento

- Remoção



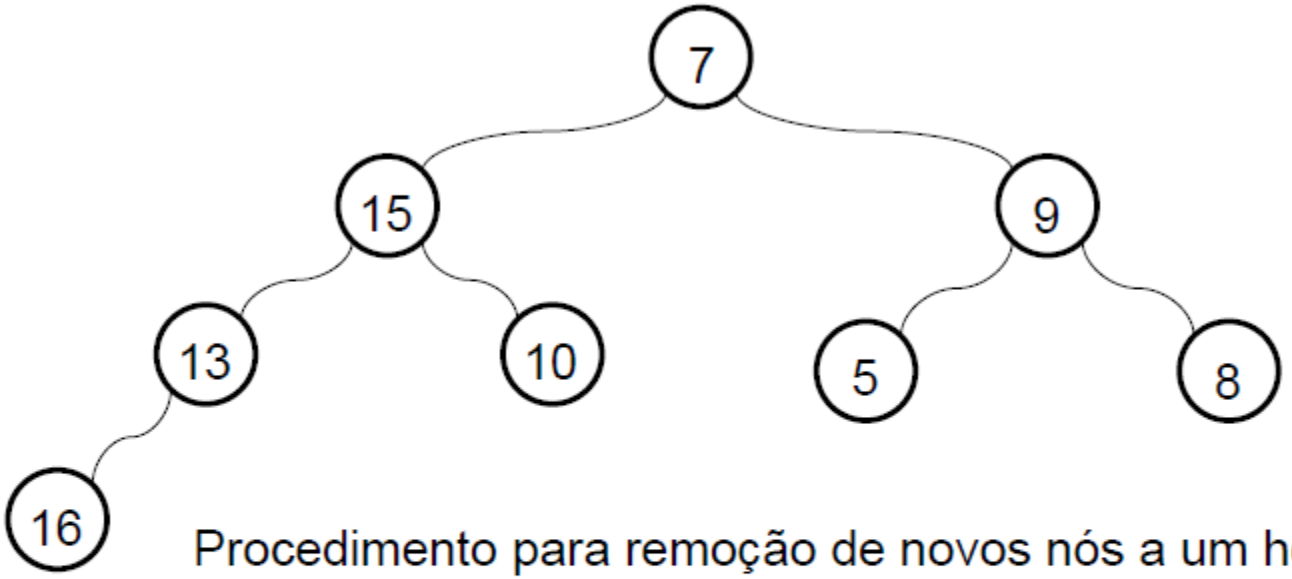
Procedimento para remoção de novos nós a um heap:

- Troca-se o elemento da raiz, índice **0**, com o último elemento do heap;
- Decrementa-se a quantidade;
- Invoca-se **Constroi** para o heap.

0	1	2	3	4	5	6	7	8
16	15	9	13	10	5	8	7	

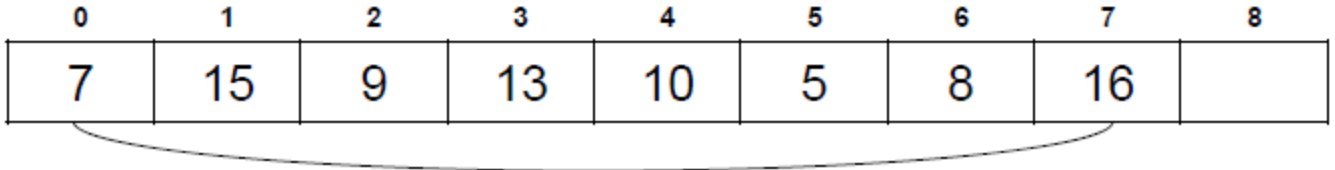
Remoção do maior elemento

- Remoção



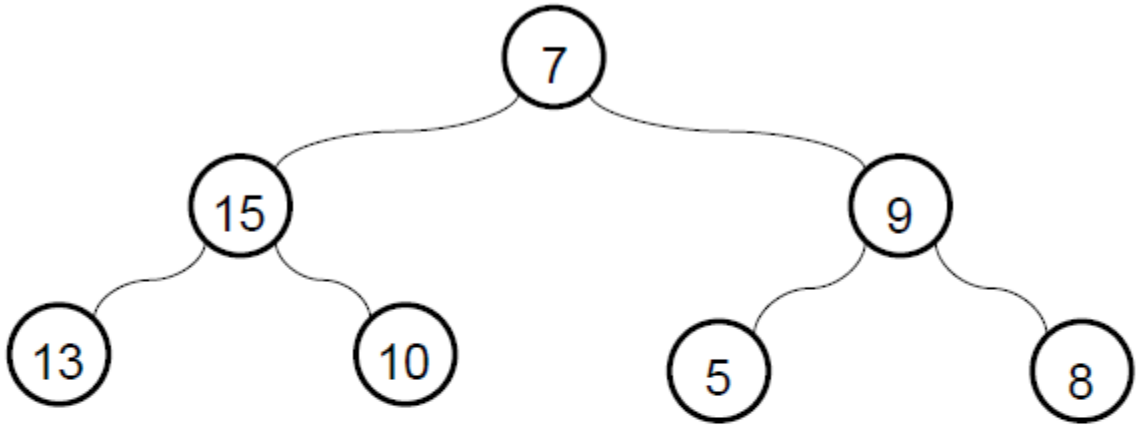
Procedimento para remoção de novos nós a um heap:

- Troca-se o elemento da raiz, índice **0**, com o último elemento do heap;
- Decrementa-se a quantidade;
- Invoca-se **Constroi** para o heap.



Remoção do maior elemento

- Remoção



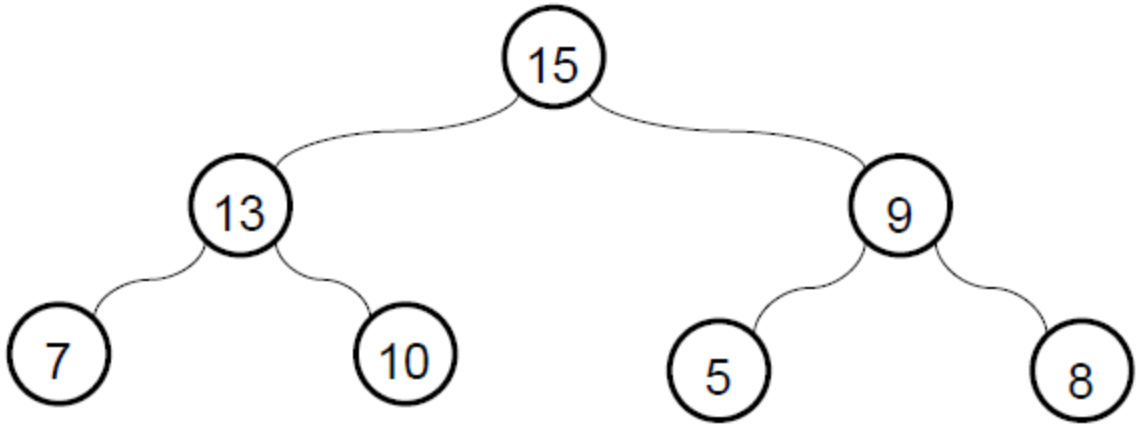
Procedimento para remoção de novos nós a um heap:

- Troca-se o elemento da raiz, índice **0**, com o último elemento do heap;
- Decrementa-se a quantidade;
- Invoca-se **Constroi** para o heap.

0	1	2	3	4	5	6	7	8
7	15	9	13	10	5	8		

Remoção do maior elemento

- Remoção



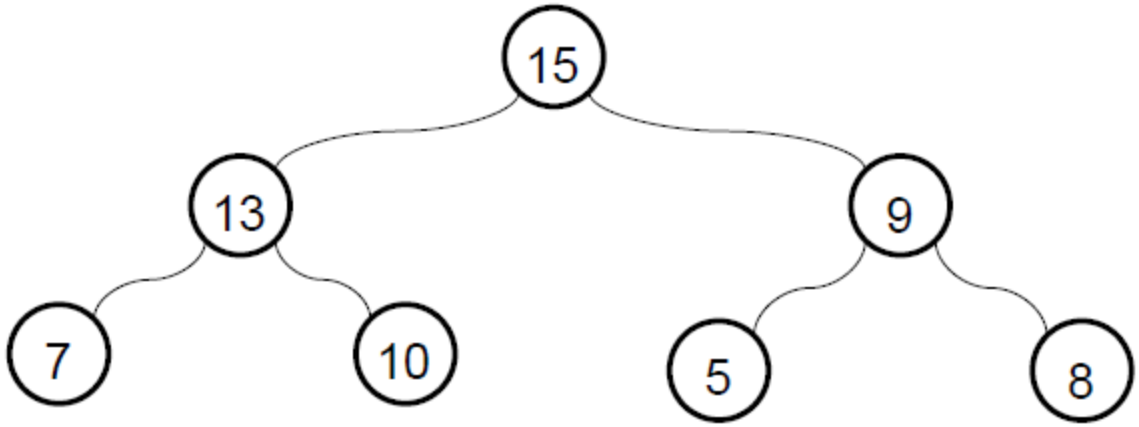
Procedimento para remoção de novos nós a um heap:

- Ao final, **Constroi** garante que o vetor resultante é heap.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5	8		

Remoção do maior elemento

- Remoção



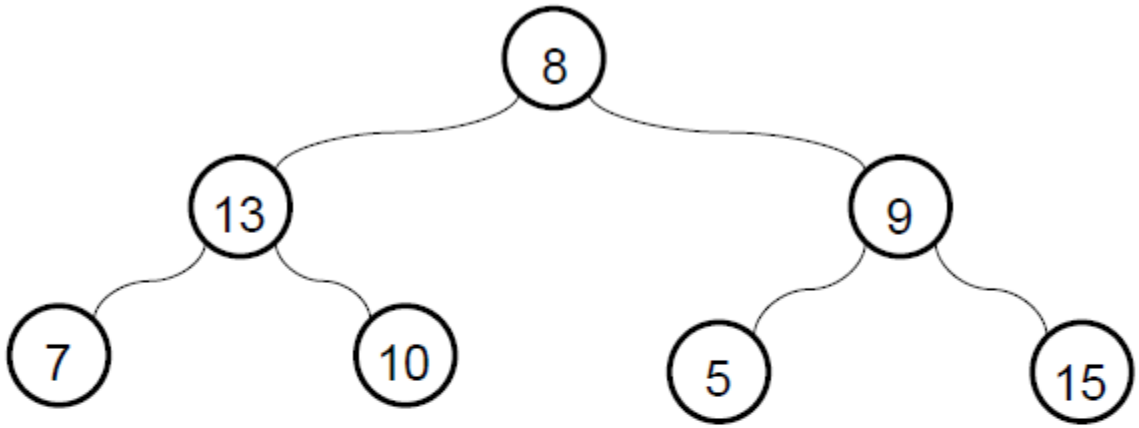
Procedimento para remoção de novos nós a um heap:

- Troca-se o elemento da raiz, índice **0**, com o último elemento do heap;
- Decrementa-se a quantidade;
- Invoca-se **Constroi** para o heap.

0	1	2	3	4	5	6	7	8
15	13	9	7	10	5	8		

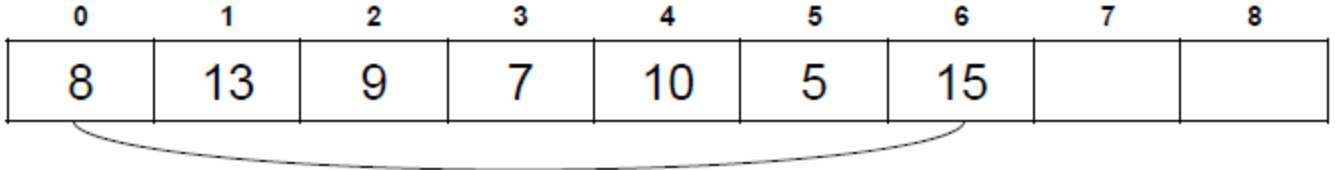
Remoção do maior elemento

- Remoção



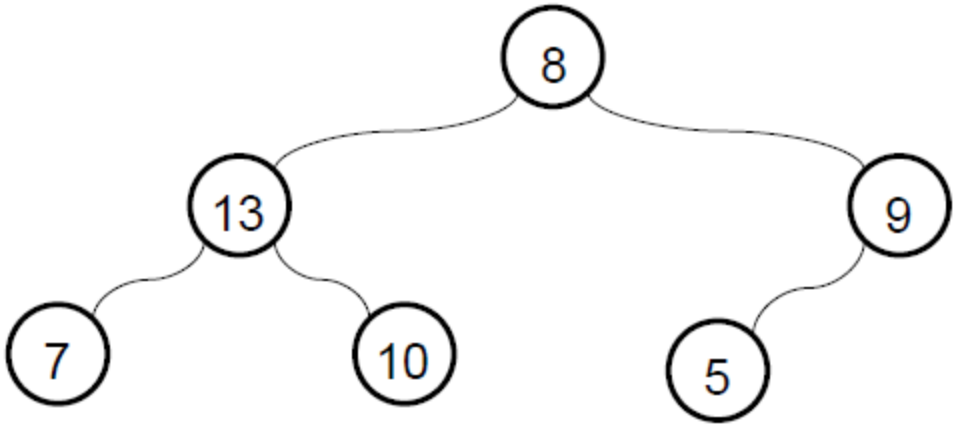
Procedimento para remoção de novos nós a um heap:

- Troca-se o elemento da raiz, índice **0**, com o último elemento do heap;
- Decrementa-se a quantidade;
- Invoca-se **Constroi** para o heap.



Remoção do maior elemento

- Remoção



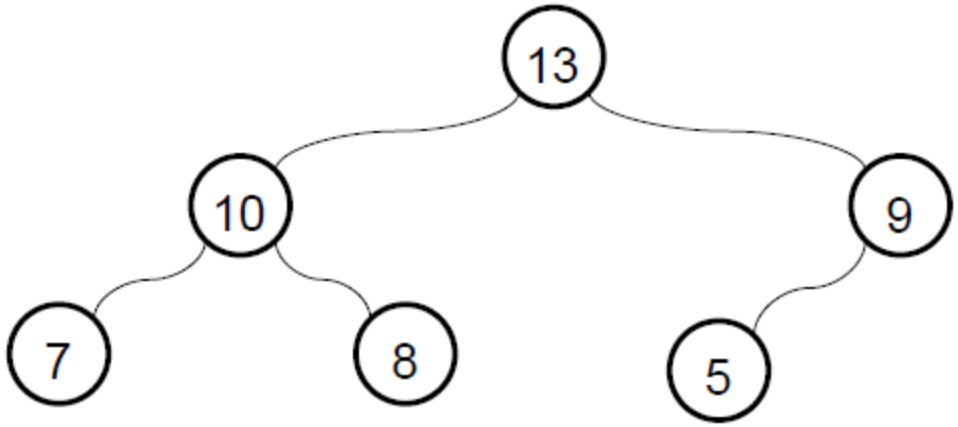
Procedimento para remoção de novos nós a um heap:

- Troca-se o elemento da raiz, índice **0**, com o último elemento do heap;
- Decrementa-se a quantidade;
- Invoca-se **Constroi** para o heap.

0	1	2	3	4	5	6	7	8
8	13	9	7	10	5			

Remoção do maior elemento

- Remoção



Procedimento para remoção de novos nós a um heap:
- Ao final, **Constroi** garante que o vetor resultante é heap.

0	1	2	3	4	5	6	7	8
13	10	9	7	8	5			

Struct do HEAP e Funções de Apoio

```
struct tHeap
{
        int itens[TAMANHO];
        int quantidade;
};
```

```
int pai(int filho)
{
    return (int)(filho-1)/2;
}
```

```
int ultimoPai(struct tHeap h)
{
    return (h.quantidade/2)-1;
}
```

```
int filhoEsq(int pai)
{
    return 2*pai+1;
}
```

```
int filhoDir(int pai)
{
    return 2*pai+2;
}
```

```
void peneirar ( struct tHeap *heap, int pai )      {
        int fEsq = filhoEsq ( pai ), fDir = filhoDir ( pai ), maior, aux;
        if ( fEsq < heap->quantidade && heap->itens[fEsq] > heap->itens[pai] )    {
                maior = fEsq;
        }
        else   {
                maior = pai;
        }

        if ( fDir < heap->quantidade && heap->itens[fDir] > heap->itens[maior] )    {
                        maior = fDir;
        }

        if ( maior != pai )    {
                        aux = heap->itens[pai];
                        heap->itens[pai] = heap->itens[maior];
                        heap->itens[maior] = aux;
                        peneirar(heap, maior);
        }
}
```

```
void construirHeap(struct tHeap *heap)
{
        int i;
        for ( i = ultimoPai ( *heap ); i >= 0; i--)
        {
                peneirar ( heap, i );
        }
}
```

```
void inserirHeap(struct tHeap *heap)    {
        int novo;
        int novoInd = heap->quantidade;
        if(heap->quantidade != TAMANHO)   {
                novo = lerItem();
                heap->itens[novoInd] = novo;
                heap->quantidade++;
                if(heap->quantidade != 1)    {
                        if(heap->itens[pai(novoInd)] < heap->itens[novoInd])    {
                                construirHeap(heap);
                        }
                }
        } else  {
                printf("\nHeap Cheio!\n");
                system("pause");
        }
 }
```

```
void removerMaxHeap(struct tHeap *heap)    {
        int aux;
        if(heap->quantidade > 0)    {
                aux = heap->itens[0];
                heap->itens[0] = heap->itens[heap->quantidade-1];
                heap->itens[heap->quantidade-1] = aux;
                heap->quantidade--;
                construirHeap(heap);
                printf("\nItem Maximo Removido: %d\n", aux);
        }
        else    {
                printf("\nHeap Vazio!!!\n");
        }
        system("pause");
}
```