CENTRO DE CIÊNCIA E TECNOLOGIA
LABORATÓRIO DE CIÊNCIAS MATEMÁTICAS
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

Heaps:
Definição e Operações

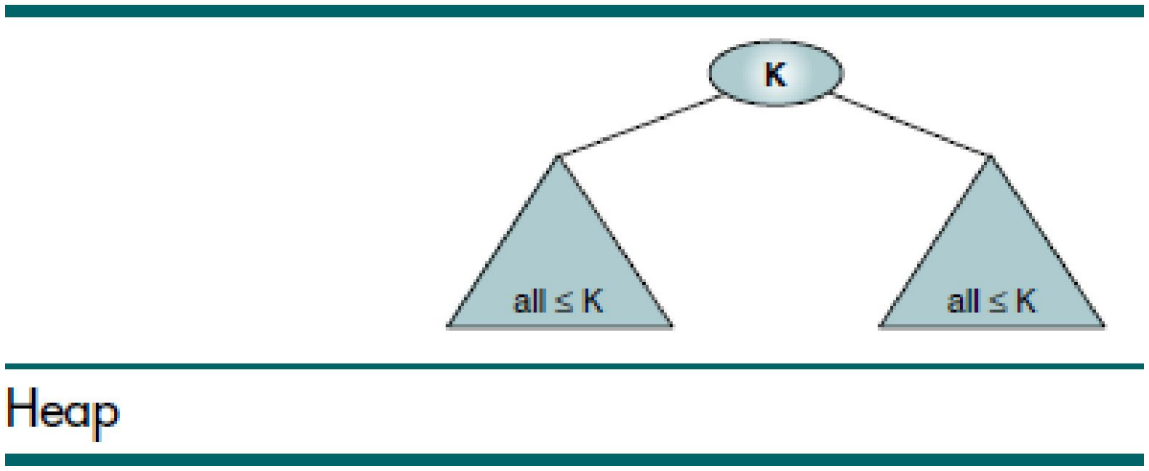
Disciplina: Estrutura de Dados II

Prof. Fermín Alfredo Tang Montané

Curso: Ciência da Computação

Heap
Definição

- Um **heap** é uma estrutura de árvore binária que segue as seguintes propriedades:
  - i) A árvore é completa ou quase completa;
  - ii) O valor da chave (key value) de cada nó é maior ou igual que o valor da chave em cada um dos seus descendentes.

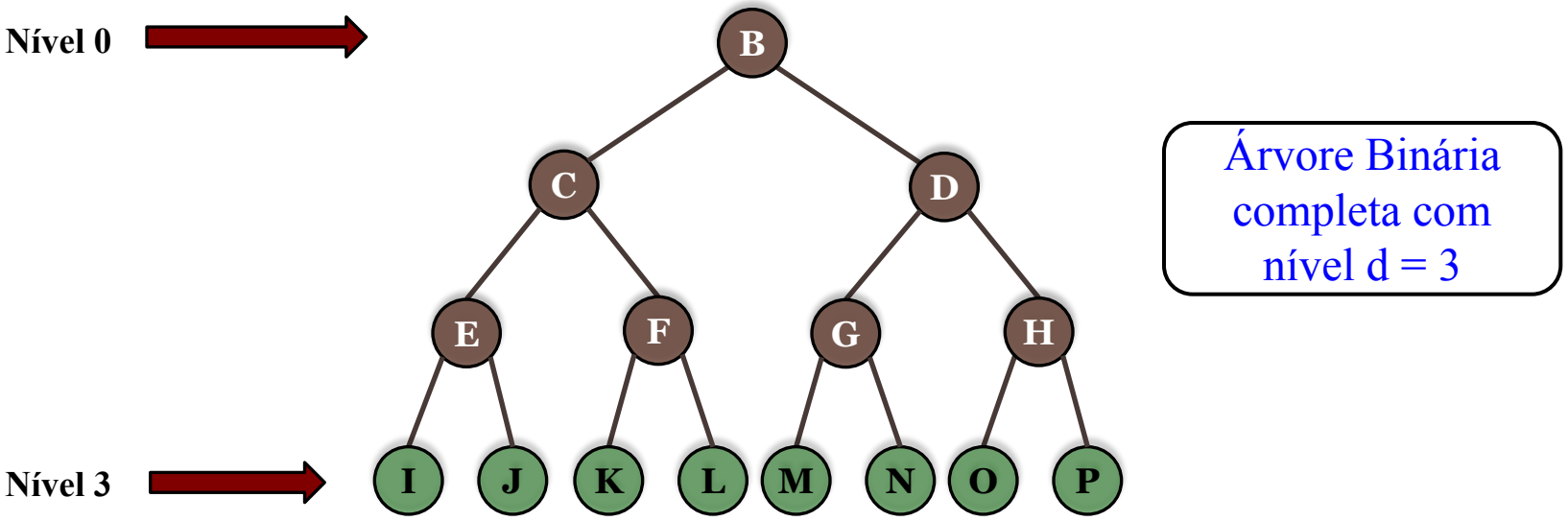


- Este tipo de estrutura por vezes é chamada de heap máximo ou **max-heap**. Uma estrutura análoga chamada de **min-heap** surge ao inverter a segunda propriedade.

Árvores Binárias

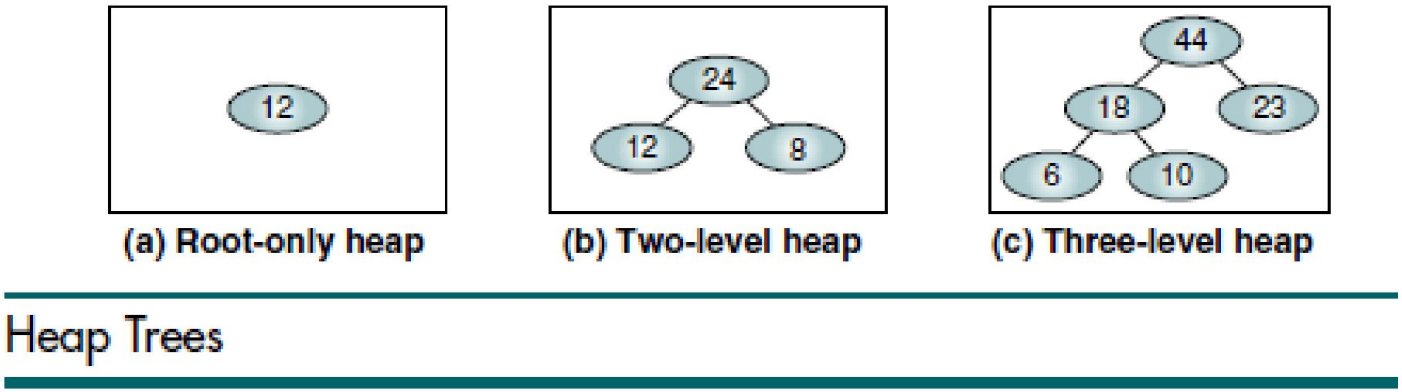
Classificação

- **Árvore Binária Completa.-** É uma árvore estritamente binária em que todas as folhas se encontram no mesmo **nivel d.** (do inglês *deep*).
- O nível de profundidade d, identifica à árvore, ele costuma ser denotada como árvore binária completa de nível d.



Heap
Exemplos

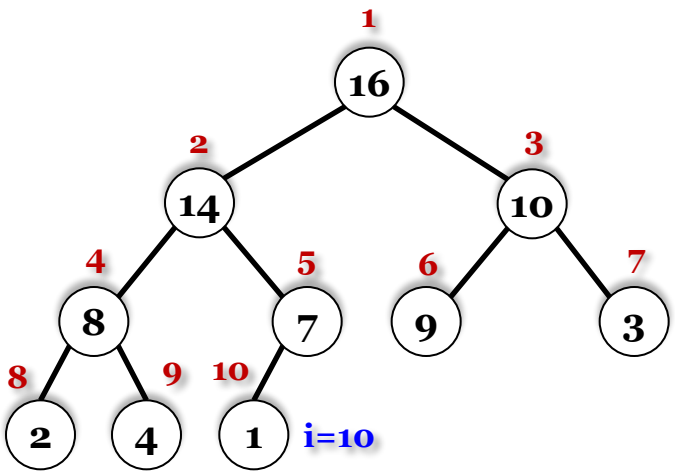
- Observe que o valor do filho esquerdo pode ser maior ou menor que o valor do filho direito.
- Além disso, o último nível da árvore é sempre preenchido pela esquerda.



Árvores Binárias

Heap Máximo

- Um **heap máximo,** ou **heap descendente**, também conhecido como árvore descendente parcialmente ordenada, de tamanho n, é uma árvore binária quase completa (com exceção do último nível) com n nós, tal que:
  - O conteúdo de cada nó seja menor ou igual ao conteúdo de seu pai.



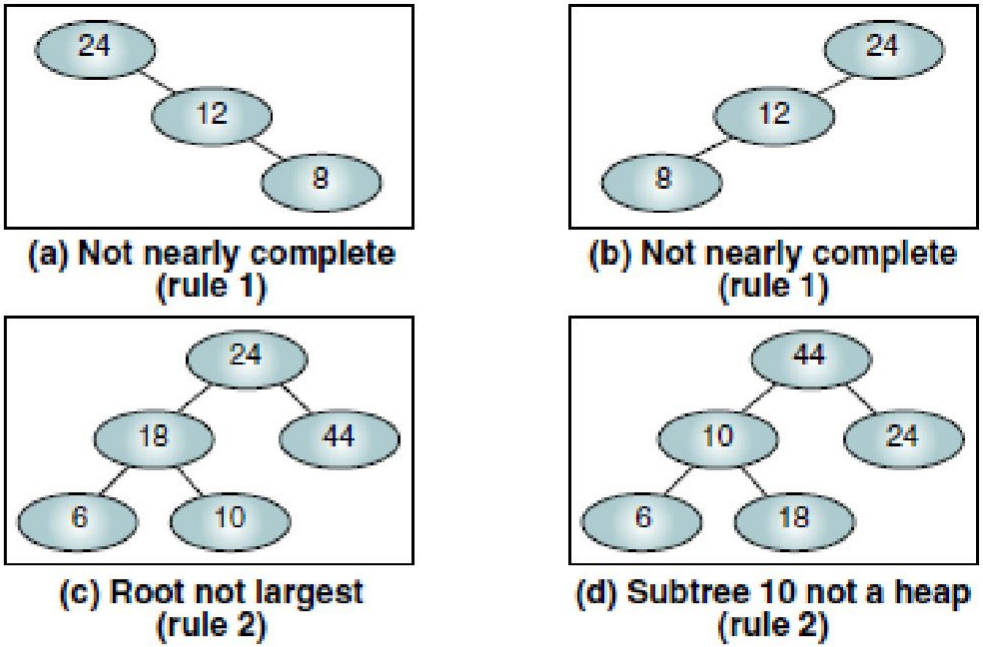
	1	2	3	4	5	6	7	8	9	10
A	16	14	10	8	7	9	3	2	4	1

$$A[PAI[i]] \geq A[i]$$

Não confundir:
i) Arvores binárias;
ii) Árvores de Pesquisa Binária;
iii) Heaps Máximos

Heap
Contra-Exemplos

- Alguma das propriedades de heap é violada nesses exemplos.

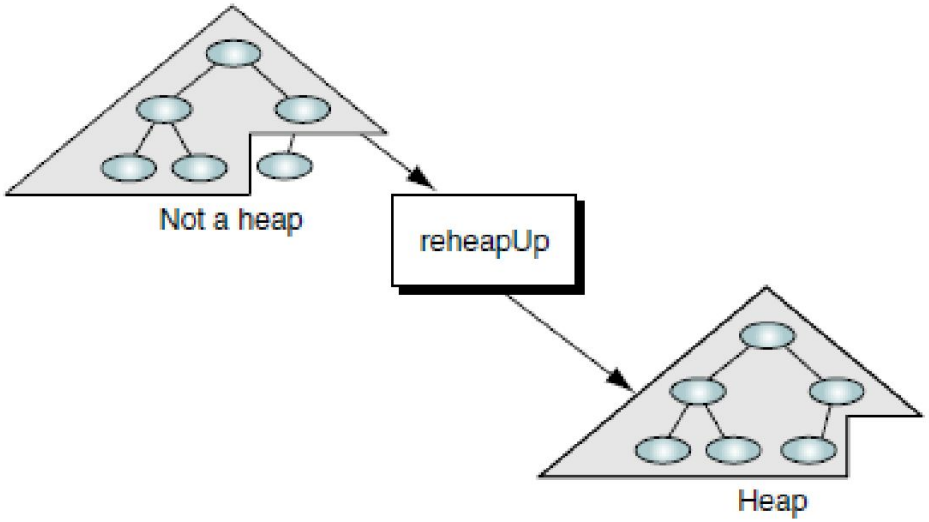


Invalid Heaps

Operações de Manutenção

Reheap Up

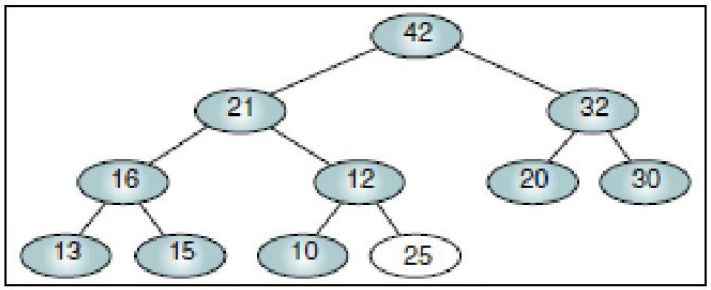
- A operação **Reheap Up**, ou reconstrução para cima (de baixo para cima), restaura as propriedades do heap deslocando o último elemento na árvore em direção à raiz até achar a posição correta.
- Neste caso um novo nó foi inserido na última posição do heap.



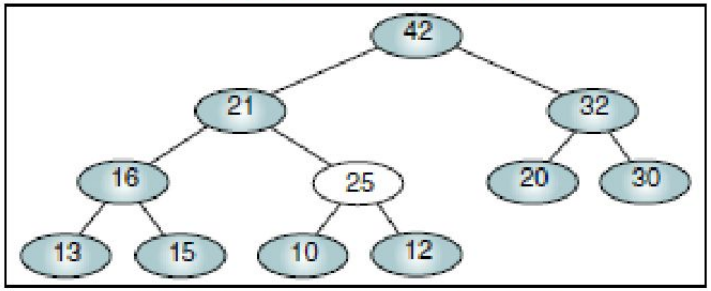
Reheap Up Operation

Operações de Manutenção

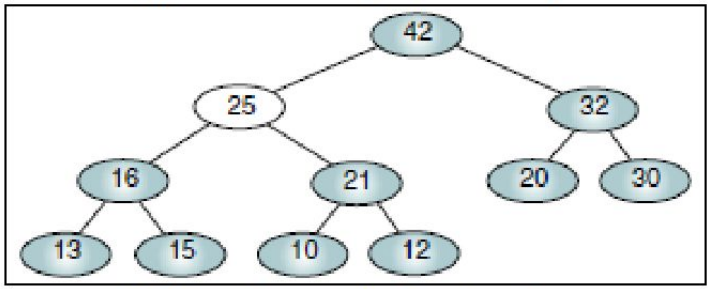
Exemplo - Reheap Up



(a) Original tree: not a heap



(b) Last element (25) moved up



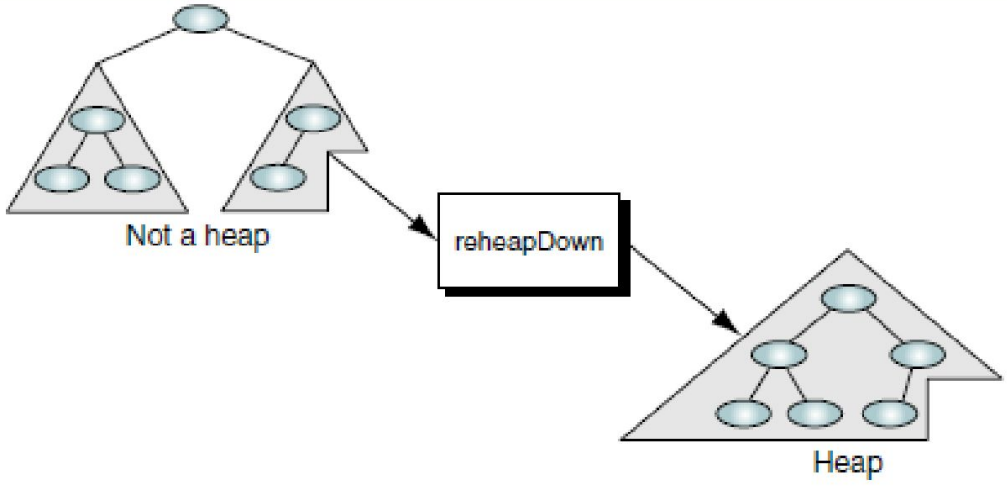
(c) Moved up again: tree is a heap

Reheap Up Example

Operações de Manutenção

Reheap Down

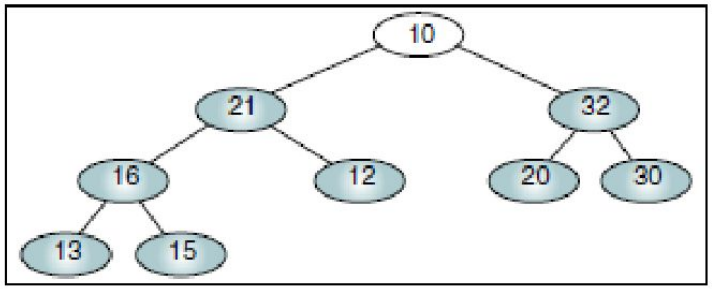
- A operação **Reheap Down**, ou reconstrução para baixo (de cima para baixo), restaura as propriedades do heap deslocando o primeiro elemento da árvore (raiz) em direção a uma das folhas, até achar a posição correta.
- Neste caso, a raiz, foi eliminada pela substituição do último elemento.



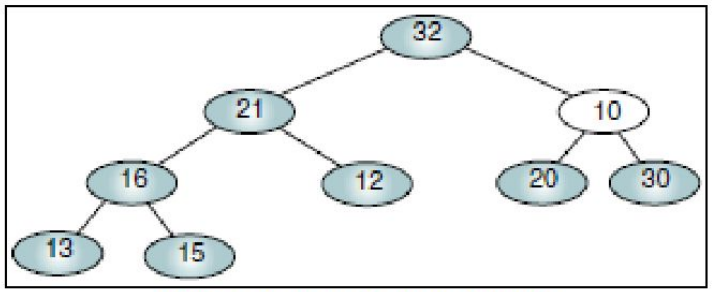
Reheap Down Operation

Operações de Manutenção

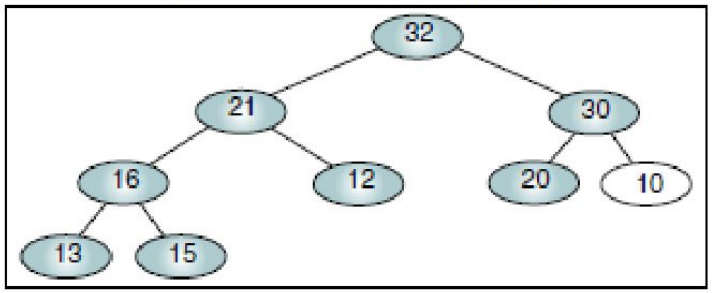
Exemplo - Reheap Down



(a) Original tree: not a heap



(b) Root moved down (right)



(c) Moved down again: tree is a heap

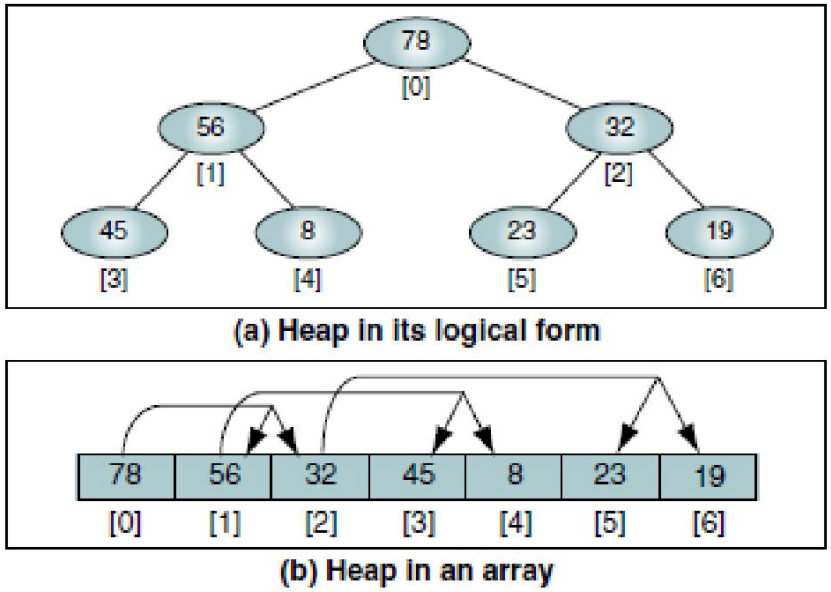
Reheap Down Example

Heaps

Implementação usando Vetores

- Um heap pode ser implementado usando uma estrutura dinâmica, no entanto costuma ser implementado em um vetor com mais frequência. Esta implementação é possível pelo fato da estrutura de heap ser completa ou quase completa.
- As relações entre as posições de um nó e as posições de seus filhos é bem definida e pode ser calculadas:

1. Para um nó localizado no índice i, seus filhos são localizados em:
   a. i) Filho esquerdo: 2i+1
   b. ii) Filho direito: 2i+2
2. O pai de um nó localizado no índice i esta localizado em $\lfloor (i-1)/2 \rfloor$.
3. Dado o índice de um filho esquerdo, j, seu irmão direito, caso exista, estará localizado em j+1. De maneira análoga, dado o índice de um filho direito, k, seu irmão esquedo deve existir, localizado em k-1.



Heaps in Arrays

Algoritmos
Reheap Up

- Algoritmo recursivo para o **Reheap up**.
- Onde: Heap é um vetor que contêm os elementos do heap ainda inviável.
- NewNode é o índice do novo elemento.

```
Algorithm reheapUp (heap, newNode)
Reestablishes heap by moving data in child up to its
correct location in the heap array.
   Pre  heap is array containing an invalid heap
        ¬newNode is index location to new data in heap
   Post heap has been reordered
1  if (newNode not the root)
   1   set parent to parent of newNode
   2   if (newNode key > parent key)
       1   exchange newNode and parent)
       2   reheapUp (heap, parent)
   3   end if
2  end if
end reheapUp
```

Algoritmos
Reheap Down

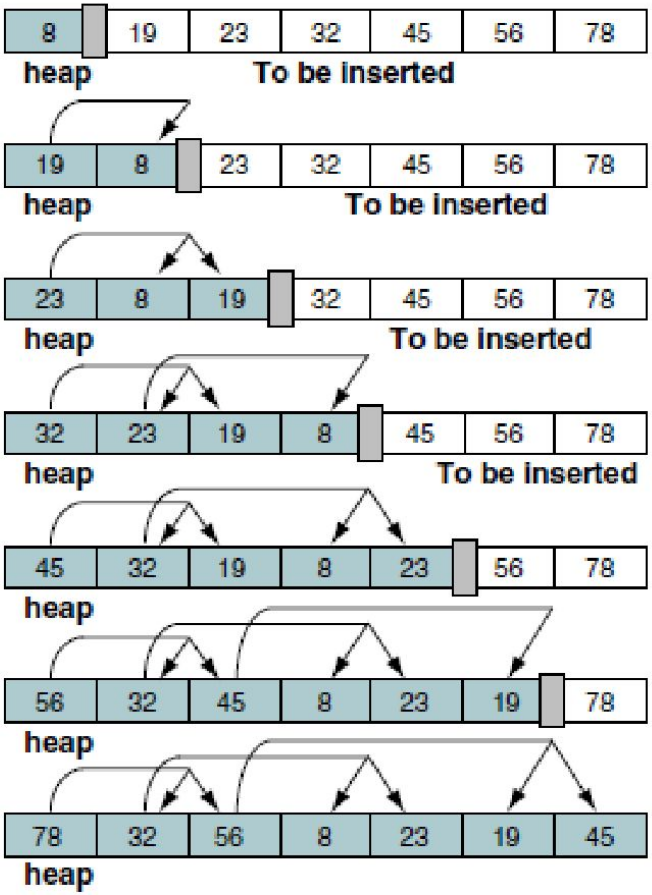
```
Algorithm reheapDown (heap, root, last)
Reestablishes heap by moving data in root down to its
correct location in the heap.
   Pre    heap is an array of data
          root is root of heap or subheap
          last is an index to the last element in heap
   Post   heap has been restored
   Determine which child has larger key
 1 if (there is a left subtree)
   1  set leftKey to left subtree key
   2  if (there is a right subtree)
      1  set rightKey to right subtree key
   3  else
      1  set rightKey to null key
   4  end if
   5  if (leftKey > rightKey)
      1  set largeSubtree to left subtree
   6  else
      1  set largeSubtree to right subtree
   7  end if
      Test if root > larger subtree
   8  if (root key < largeSubtree key)
      1  exchange root and largeSubtree
      2  reheapDown (heap, largeSubtree, last)
   9  end if
 2 end if
end reheapDown
```

Algoritmos
Construir um Heap

```
Algorithm buildHeap (heap, size)
Given an array, rearrange data so that they form a heap.
   Pre    heap is array containing data in nonheap order
          size is number of elements in array
   Post   array is now a heap
 1 set walker to 1
 2 loop (walker < size)
   1   reheapUp(heap, walker)
   2   increment walker
 3 end loop
end buildHeap
```

Algoritmos

Construir um Heap



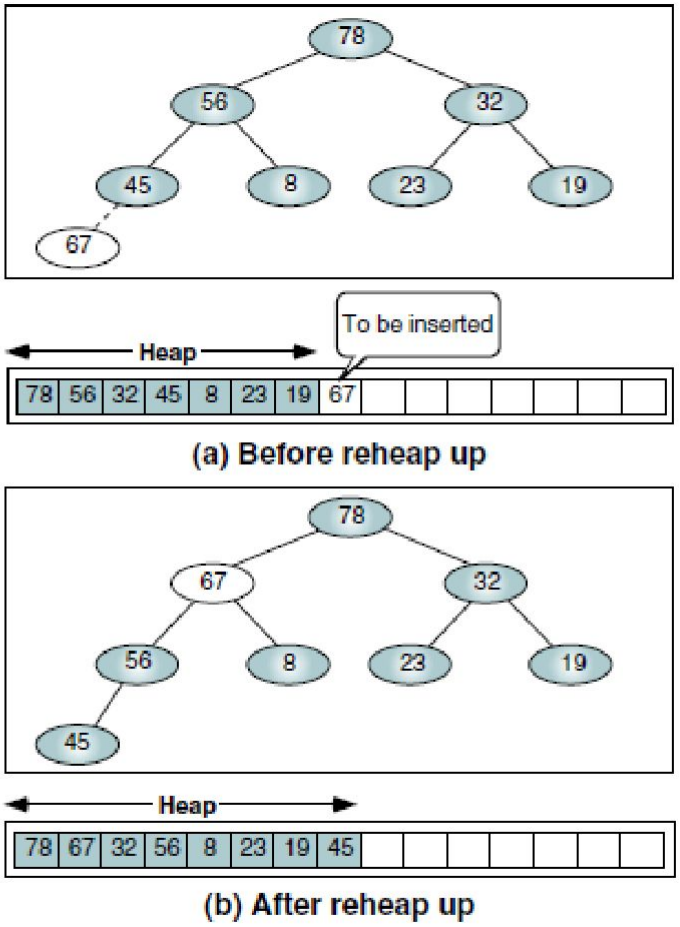
Building a Heap

Algoritmos
Inserir um nó

```
Algorithm insertHeap (heap, last, data)
Inserts data into heap.
   Pre    heap is a valid heap structure
          last is reference parameter to last node in heap
          data contains data to be inserted
   Post   data have been inserted into heap
   Return true if successful; false if array full
1  if (heap full)
   1  return false
2  end if
3  increment last
4  move data to last node
5  reheapUp (heap, last)
6  return true
end insertHeap
```

Algoritmos

Inserir um nó



Insert Node

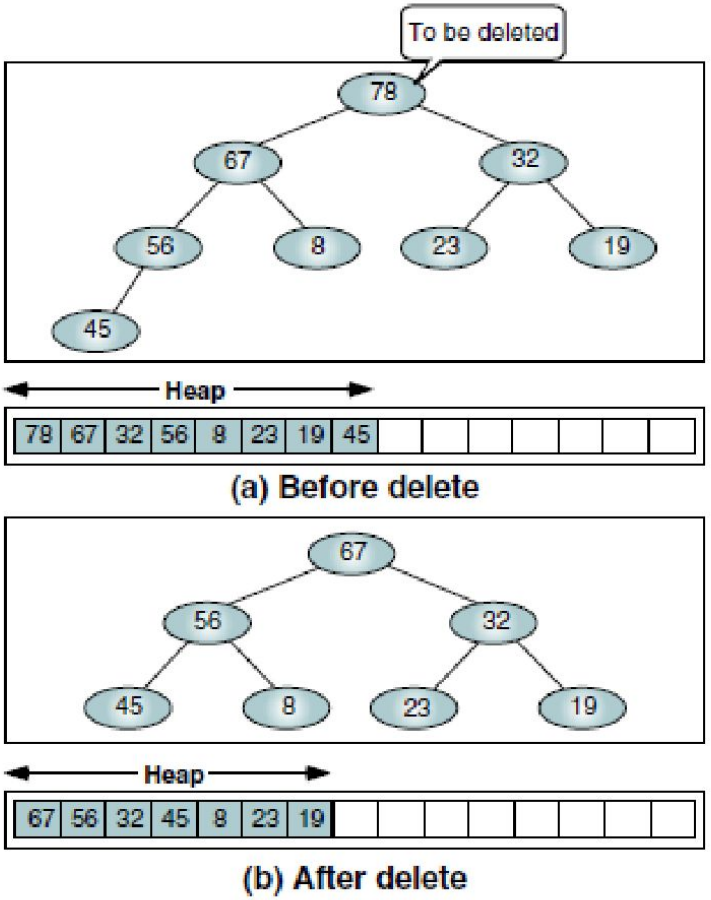
Algoritmos

Remover um nó

```
Algorithm deleteHeap (heap, last, dataOut)
Deletes root of heap and passes data back to caller.
   Pre    heap is a valid heap structure
          last is reference parameter to last node in heap
          dataOut is reference parameter for output area
   Post   root deleted and heap rebuilt
          root data placed in dataOut
   Return true if successful; false if array empty
1  if (heap empty)
   1  return false
2  end if
3  set dataOut to root data
4  move last data to root
5  decrement last
6  reheapDown (heap, 0, last)
7  return true
end deleteHeap
```

Algoritmos

Remover um nó



`deleteHeap` Node

Referências

- Gilberg, R.F. e Forouzan, B. A. Data Structures_A Pseudocode Approach with C. Capítulo 9. Heaps. Segunda Edição. Editora Cengage, Thomson Learning, 2005.