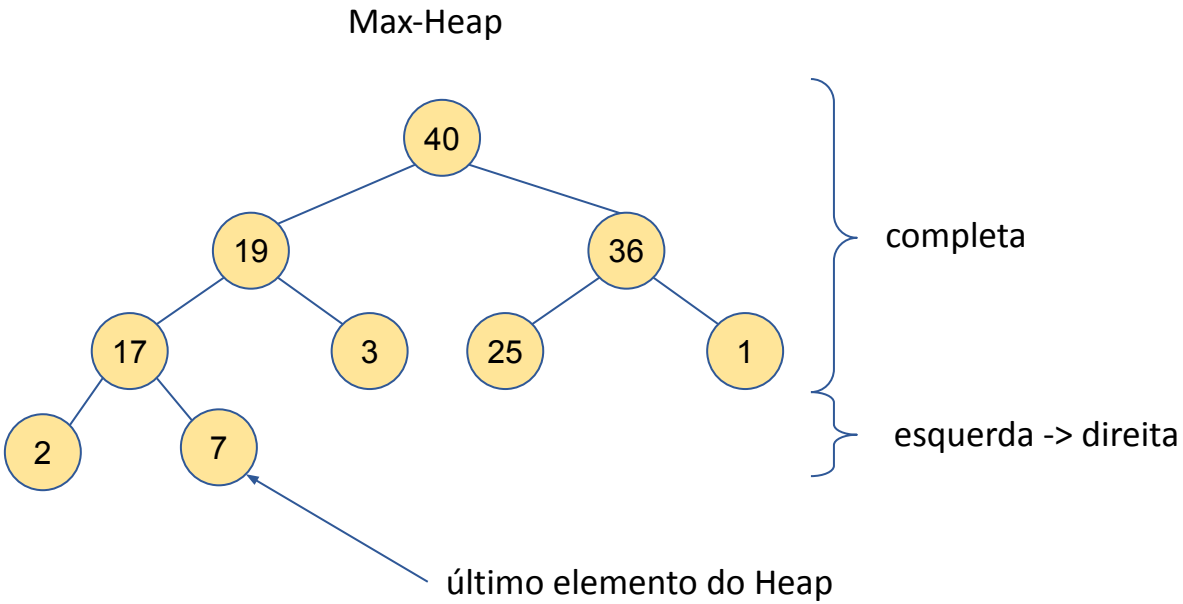
HEAPs

Prof. Hélder Almeida

DEFINIÇÃO

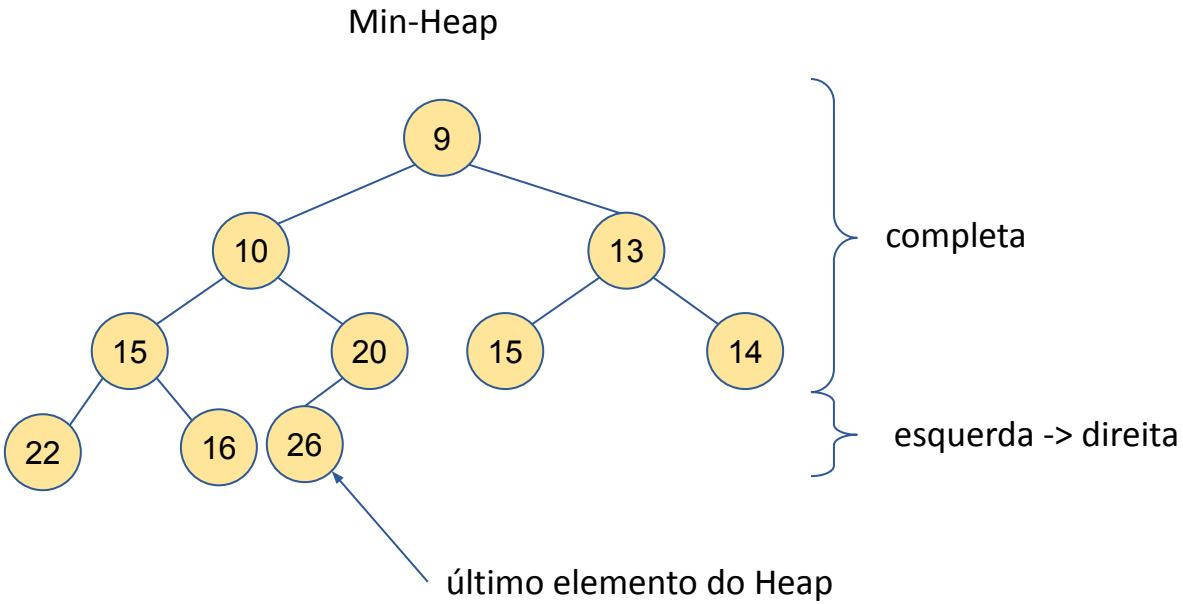
- É uma estrutura de dados **organizada** como uma **árvore binária** seguindo as seguintes **regras:**
- Heap-Order**:**
  - **Max-Heap**: todo nó interno ***v***, que não a raiz, ***key(v) ≤ key(parent(v))***
    - A **raiz** possui o **maior elemento;**
  - **Min-Heap**: todo nó interno ***v***, que não a raiz, ***key(v) ≥ key(parent(v))***
    - A **raiz** possui o **menor elemento;**
- Estar organizada da seguinte forma:
  - Estar **completa** até pelo menos seu **penúltimo nível;**
  - Se o seu último nível não estiver completo, todos os nós do **último nível** deverão estar **agrupados à esquerda**

Exemplo



Dica: um heap armazenando **n** chaves tem altura ***O*(log *n*)**

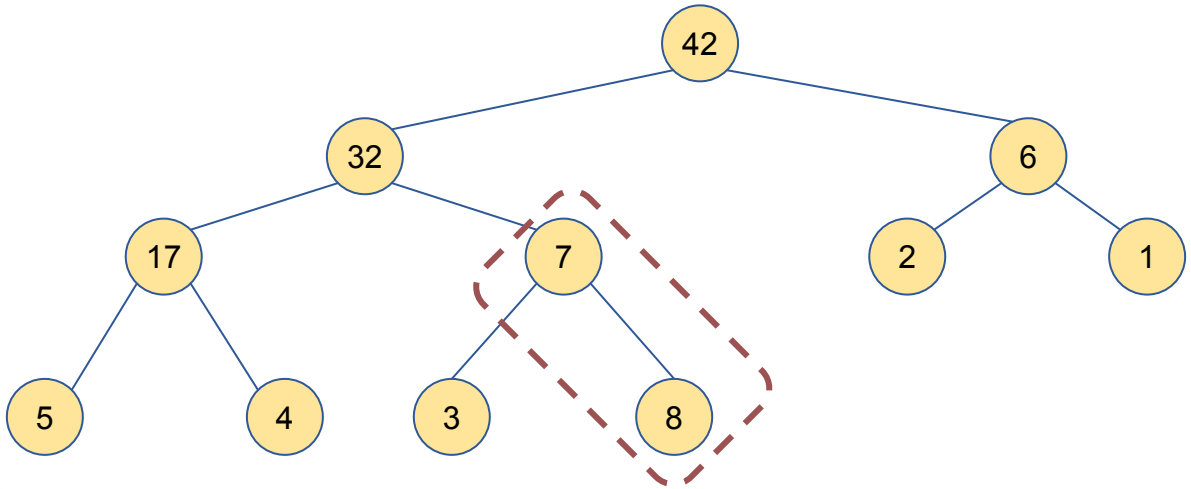
Exemplo



Dica: um heap armazenando **n** chaves tem altura ***O*(log *n*)**

Perguntas

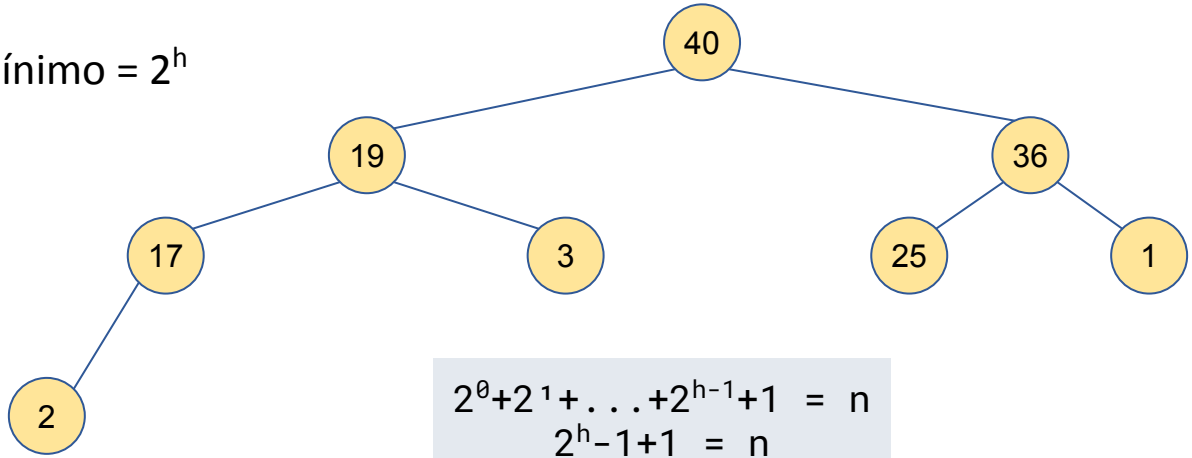
- A árvore binária a seguir pode representar uma Max-Heap? Justifique



Perguntas

- Qual o número mínimo de elementos de uma heap com altura h?

Mínimo = 2^h

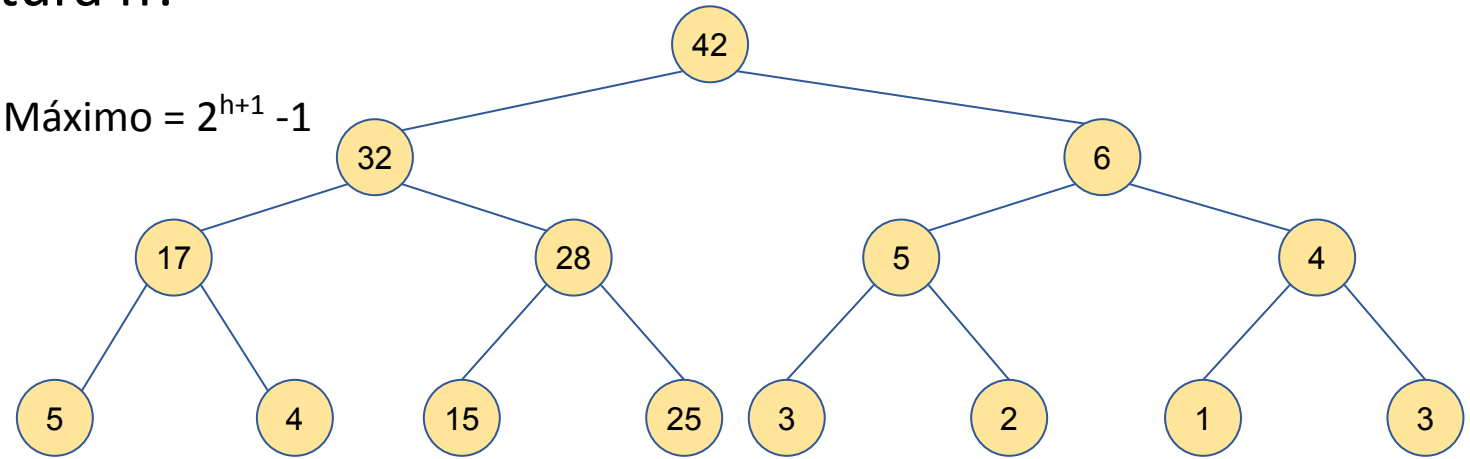


$$2^0+2^1+...+2^{h-1}+1 = n$$
$$2^h-1+1 = n$$
$$2^h = n$$

Perguntas

- Qual o número máximo de elementos de uma heap com altura h?

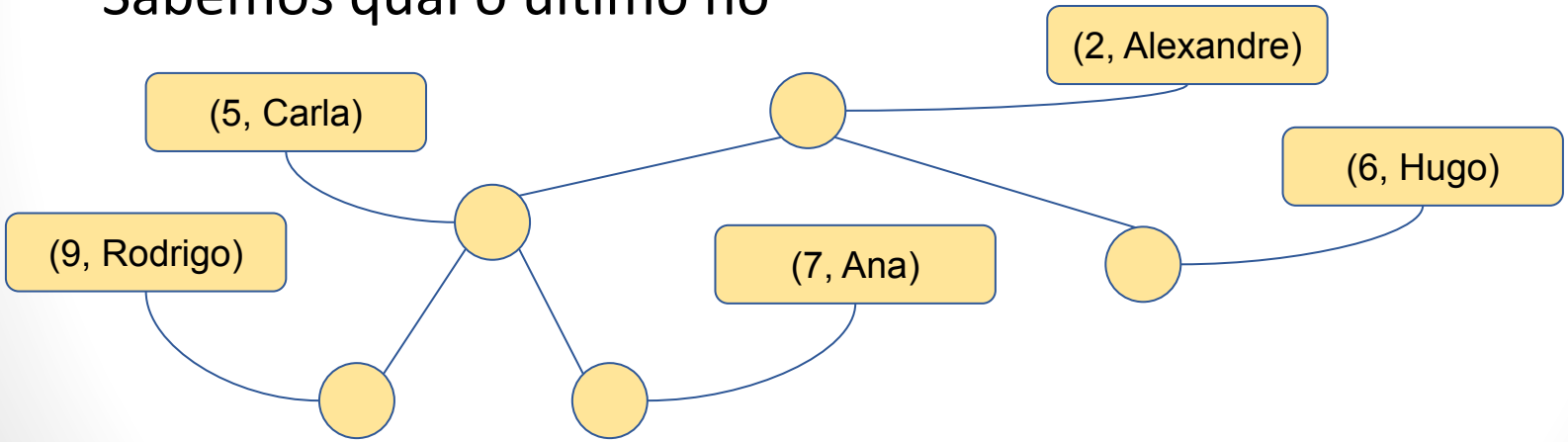
Máximo = $2^{h+1} - 1$



$$2^0 + 2^1 + \ldots + 2^h = n$$
$$2^{h+1} - 1 = n$$

Heaps e filas de prioridade

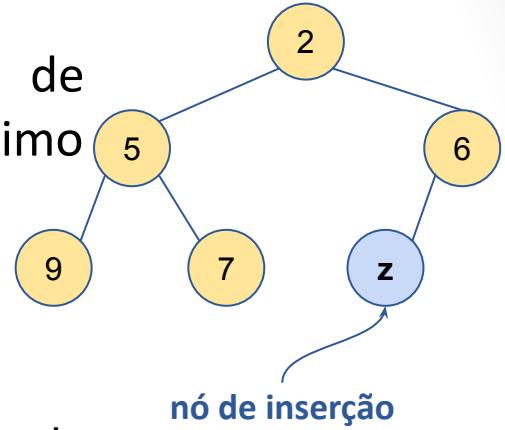
- Podemos usar um heap para implementar uma **fila de prioridade**
- Armazenamos um item **(chave, elemento)** em cada nó
- Sabemos qual o último nó



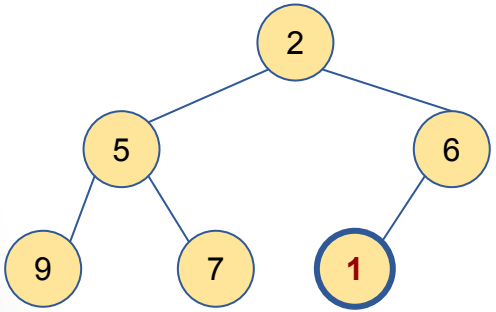
Heaps - Inserção

O algoritmo de inserção consiste de 3 passos:

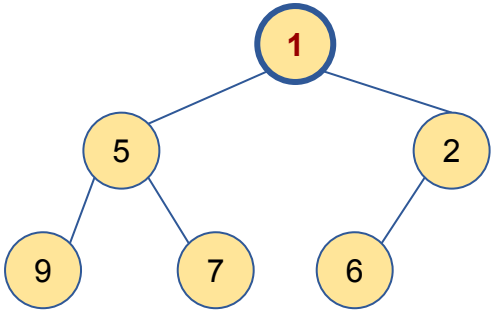
a. Encontrar o nó de inserção z (próximo último nó)



b. Armazenar **k** em **z**



c. Restaurar a heap-order



Restaurar a Heap-Order

Basicamente existem 2 algoritmos para organizar um Heap:

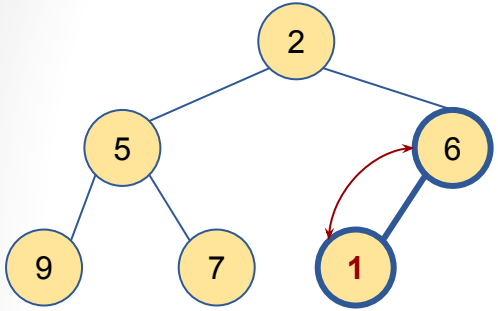
- **Upheap** ou "Corrige-Subindo" e
- **Downheap** ou Heapify ou "Corrige-Descendo".

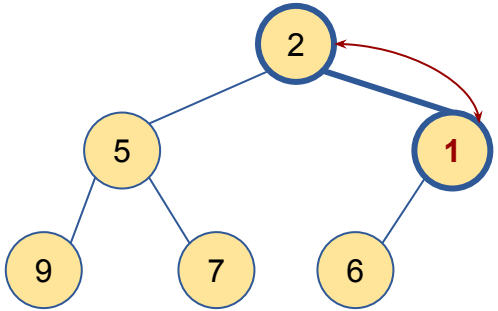
No caso de uma inserção, onde sabemos a localização do possível nó fora de ordem, é mais eficiente e simples utilizar o **Upheap**.

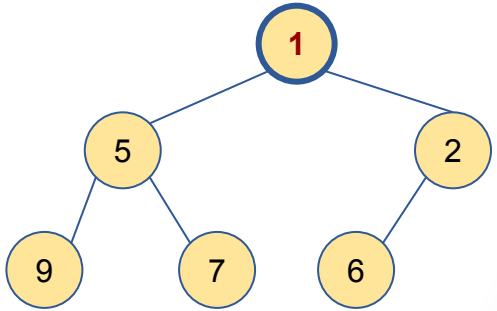
Upheap

- Após a inserção de uma nova chave ***k***, a propriedade heap-order pode estar violada
- O algoritmo upheap restaura a propriedade heap-order trocando ***k*** sobre o “caminho acima” a partir do nó de inserção
- Upheap termina quando a chave ***k*** encontra o nó raiz ou um nó cujo pai possua uma chave menor ou igual a ***k***, para Min-heaps, ou maior ou igual a ***k*** para Max-heaps.
- Como um heap tem altura **O(log n)**, upheap roda em tempo **O(log n)**

Upheap



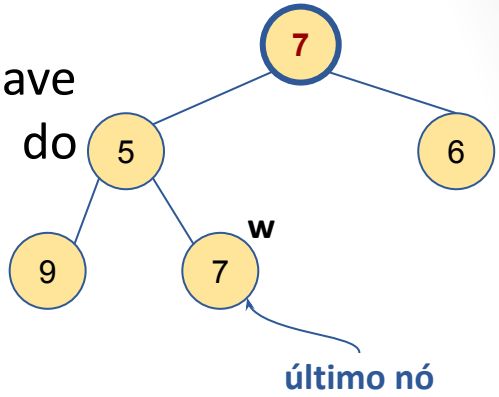




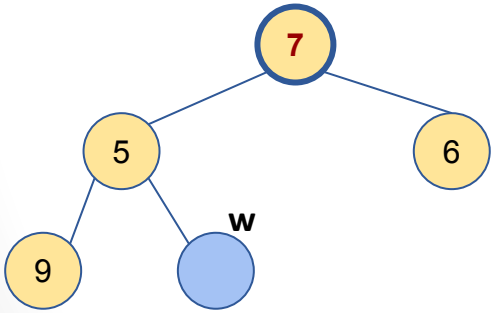
Heaps - Remoção

O algoritmo de remoção elimina a raiz em 3 passos:

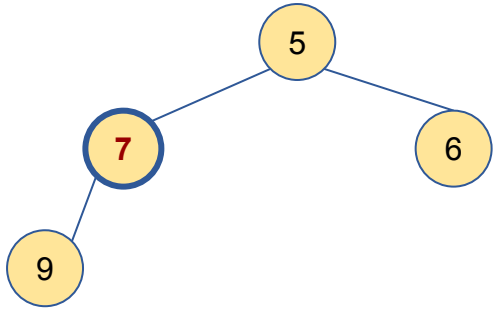
a. Substituir a chave da raiz pela do último nó **w**



b. Eliminar **w**



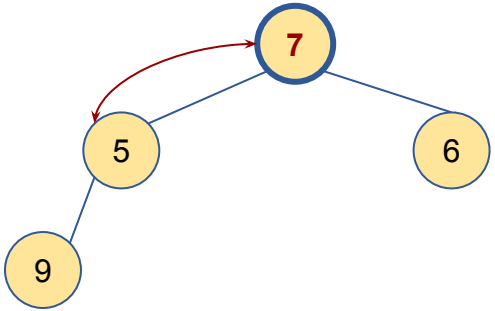
c. Restaurar a heap-order

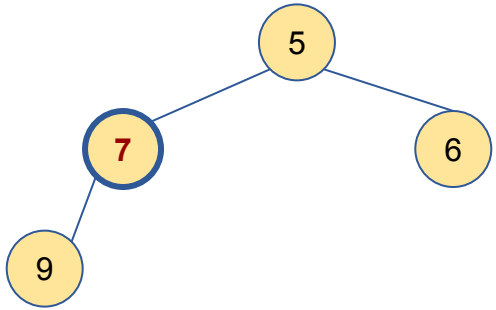


Downheap

- Depois de substituir a chave da raiz com a chave do último nó, a propriedade heap-order pode estar violada
- O algoritmo downheap restaura esta propriedade trocando a chave ***k*** sobre o "caminho abaixo" da raiz
- Downheap termina quando a chave ***k*** encontra uma folha ou um nó cuja chave é maior do que ***k***
- Como um heap tem altura **O(log n)**, downheap roda em tempo **O(log n).**

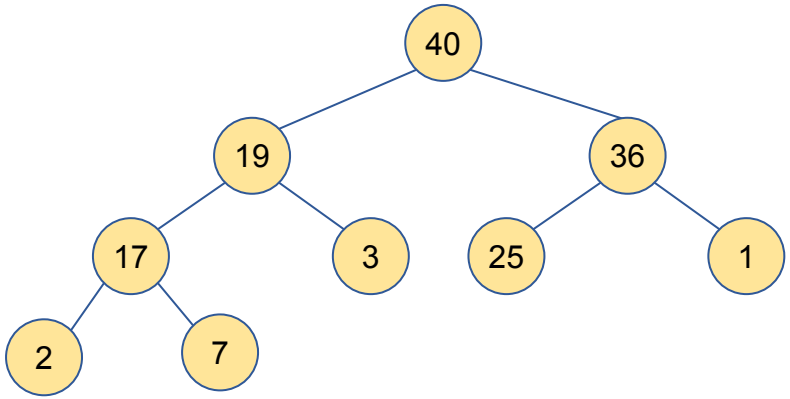
Downheap





Representação em Vetor

índice	0	1	2	3	4	5	6	7	8	9
valor		40	19	36	17	3	25	1	2	7



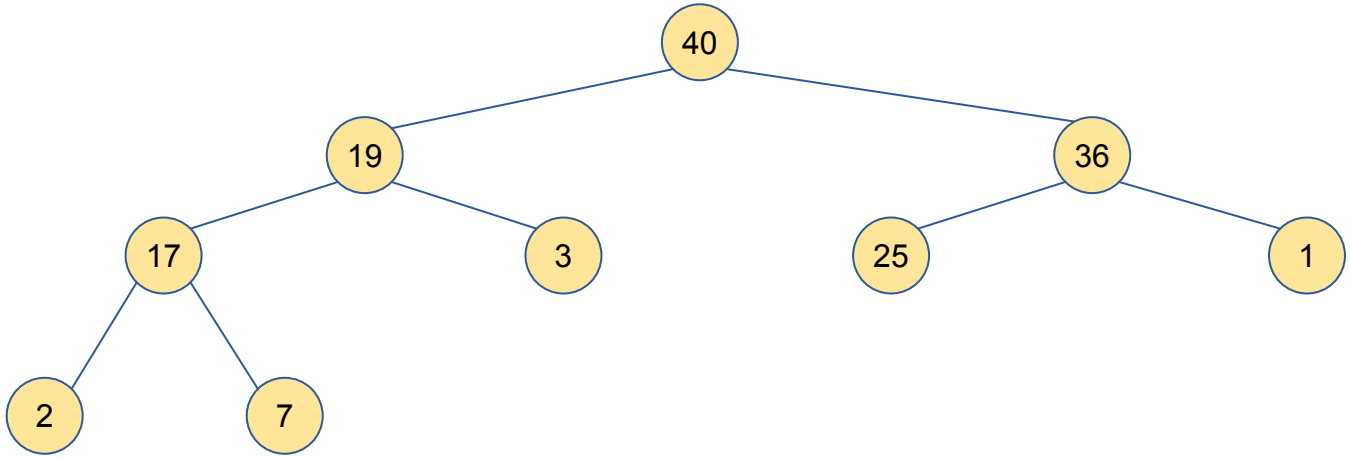

```
parent(i):
    return i/2

left(i):
    return 2*i

right(i):
    return 2*i+1
```

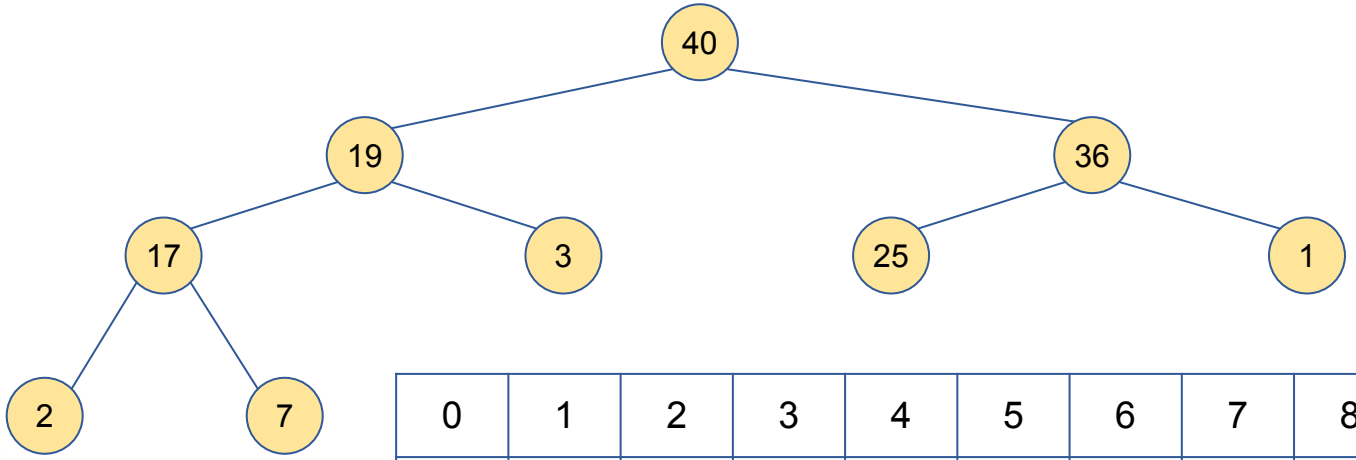
Exercícios

Inserir o elemento 20



Exercícios

Inserir os elementos 32 45 17 2 5 na heap abaixo. Mostre como a heap fica após cada operação, graficamente e na representação de vetor.



0	1	2	3	4	5	6	7	8	9
	40	19	36	17	3	25	1	2	7

Conteúdo adicional

- https://people.ksp.sk/~kuko/gnarley-trees/Intro.html
- https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/heap.html
-