Estrutura de Dados Heap 1

Usando uma Fila de Prioridades para Ordenação

Prof. Mateus M. Luna[1] **Prof. André L. Moura**[2]

1. mateus_m_luna@ufg.br
2. andre_moura@ufg.br

Material elaborado com base do trabalho das professoras Telma e Juliana Félix (INF-UFG) e do professor João Luís Garcia Rosa (ICMC-USP)

2022

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Sumário

A estrutura de dados *Heap*

Dando continuidade aos nossos estudos de ordenação, veremos o *Heap Sort*. Mas antes, precisamos entender o que é uma heap!

A Heap é uma estrutura de dados implementada como um **array visualizado como árvore**, mais especificamente em **árvores binárias quase completas**. Cada nó desta árvore precisa obedecer uma **propriedade max-heap ou min-heap** (um conjunto de regras). Se obedecer estas condições, poderemos realizar operações de **inserção** e **remoção** de forma relativamente eficiente.

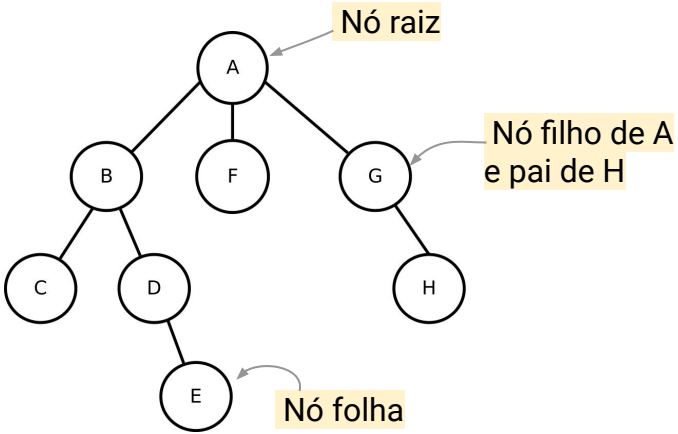
Heap são em particular usadas para **criar filas de prioridade**, por isso também são referidas como tal.

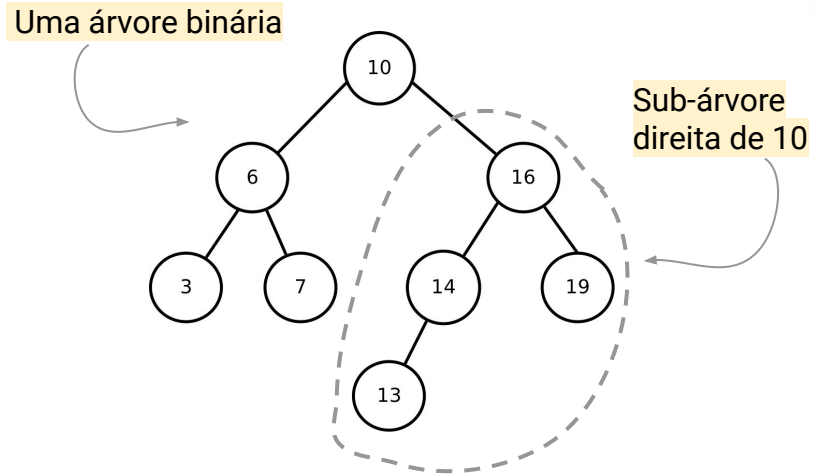
Árvores Binárias Quase Completas

Árvores Binárias Quase Completas?

Veremos árvores em detalhes mais adiante no curso. Por hora, nos interessa que:

- **Árvore** é uma *estrutura de dados em grafo não dirigida acíclica*, com *nós* conectados;
- Todo **nó** em uma árvore pode ser/conter um nó chamado de **pai** (ancestral) e múltiplos nós chamados de **filhos** (descendentes);
- **Árvores binárias,** são árvores onde cada nó possui de *0 a 2 nós no máximo*.





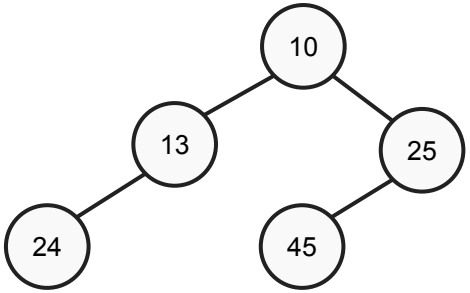
Árvores Binárias *Quase Completas?*

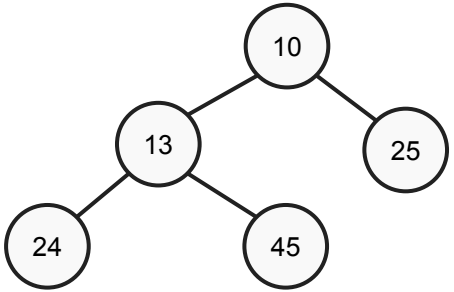
A **profundidade** de um nó é a distância deste nó até a raiz.

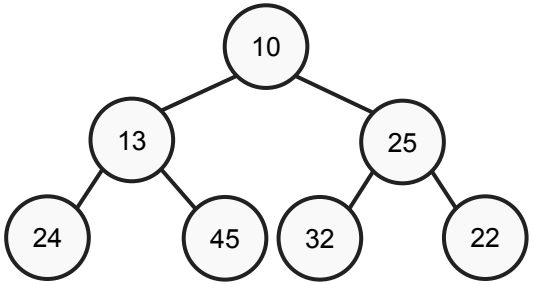
Um conjunto de nós com a mesma profundidade é denominado **nível da árvore**.

A maior profundidade de um nó, dentre todos, é a **altura da árvore**.

Uma **Árvore Binária Completa** é uma árvore binária em que cada nível da árvore binária está completamente preenchido, exceto o último nível. Se todos os níveis estão completamente preenchidos, temos uma **Árvore Binária Cheia** (alguns chamam esta de completa e a anterior de *quase completa*).





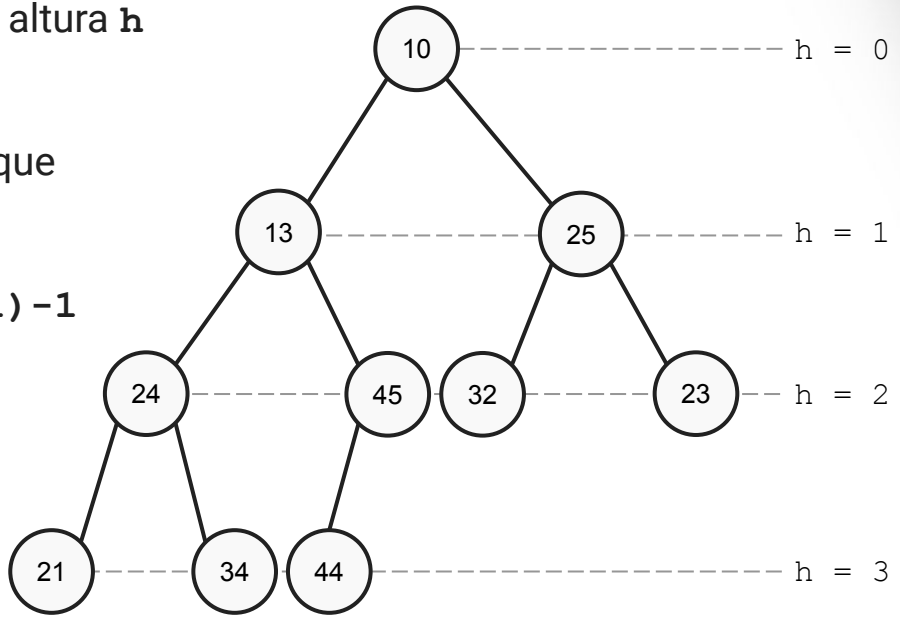


Árvores Binárias *Quase Completas?*

A definição de árvores binárias completas nos permite assumir alguns dados. Dada a altura **h** e a quantidade máxima de nós **n**:

Qual o número máximo de elementos que cabem nela? $n(h) = 2^{h+1}-1$

Qual a altura dela? $h(n) = \log_2(n+1)-1$

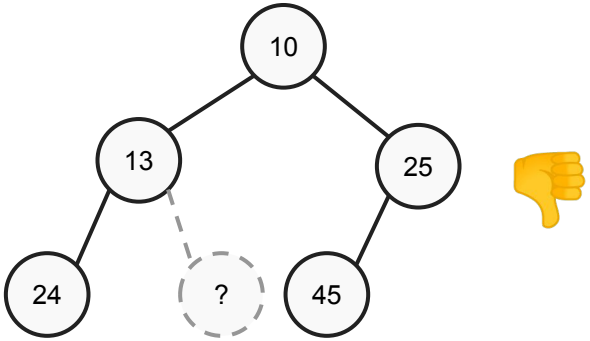


Uma árvore binária *quase completa* com **10 nós**, de **altura 3**.
Quando estiver completa e cheia, ela terá 15 nós.

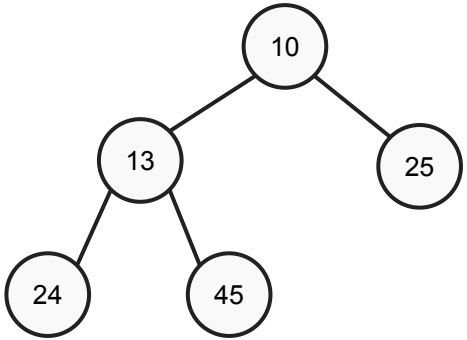
Representação de Arrays como Árvores

Representação de Arrays Multidimensionais

Além de visualizados como *Árvores Binárias Quase Completas*, Heaps tem mais uma particularidade: Seu último nível está sempre preenchido **em ordem da esquerda para a direita**:



👎

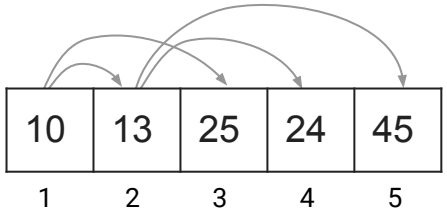


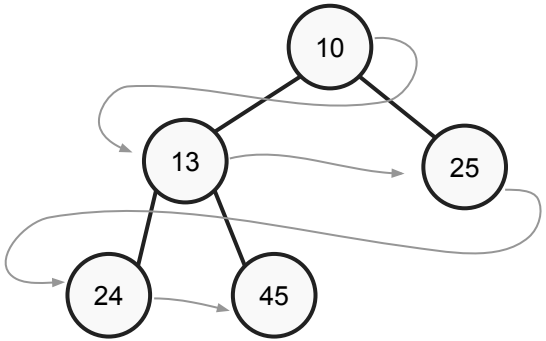
👍

A importância de todas essas restrições é que isso nos permite armazenar essa estrutura de dados em **um simples array**, onde os elementos são lidos fazendo uma **Busca em Largura**.

Representação de Arrays Multidimensionais

Esta leitura via Busca em Largura funciona olhando nível a nível da árvore, de cima para baixo, da esquerda para a direita:





Mesmo guardada em um array, podemos fazer perguntas como "quem é o pai" deste nó?" e "quem é o filho esquerdo deste nó?" com uma simples aritmética em torno do índice `i` (note que estamos usando base 1 ao invés de 0):

Nó atual: `i`

Filho esquerdo deste nó: `2*i`

Pai deste nó: `⌊i/2⌋`

Filho direito deste nó: `2*i + 1`

Propriedades
max-heap & min-heap

Propriedades *max-heap & min-heap*

Há dois tipos de Heaps:

- **Heaps máximos**: onde todo nó obedece à propriedade *max-heap*, estando o maior elemento no topo;
- **Heaps mínimos**: onde todo nó obedece à propriedade *min-heap*, estando o menor elemento no topo;

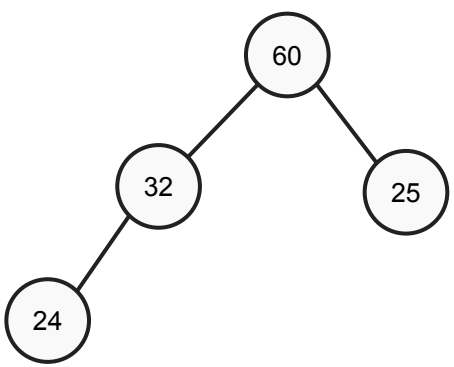
A propriedade *max-heap* diz que: *Dado um nó* `n` *da árvore, este sempre deve ser* ***menor que o seu pai e maior que os seus filhos***. *Deste modo, o nó raiz é maior que todos os demais nós da árvore.*

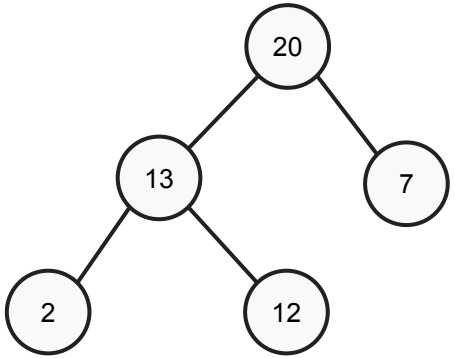
A propriedade *min-heap* diz que: Dado um nó `n` da árvore, este sempre deve **ser maior que o seu pai e menor que os seus filhos**. *Deste modo, o nó raiz é sempre menor que todos os demais nós da árvore.*

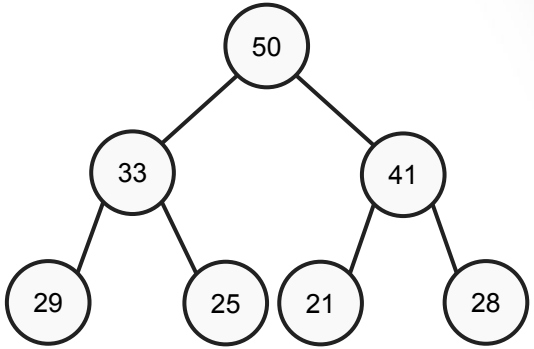
Para este conteúdo, daremos exemplos usando Heaps máximos, mas as propriedades são análogas para os mínimos.

Propriedades *max-heap & min-heap*

Exemplos de Heaps máximos:







60	32	25	24

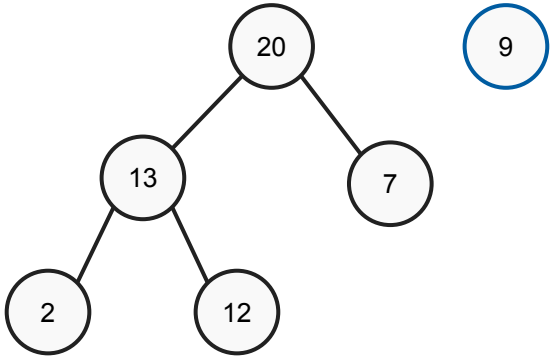
20	13	07	02	12

50	33	41	29	25	21	28

Inserção e Remoção

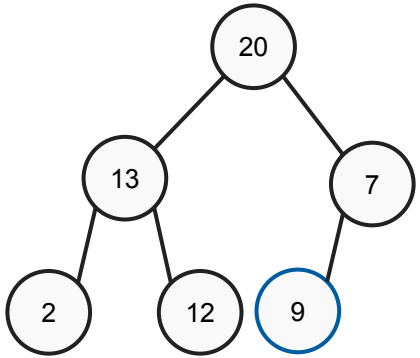
Inserção em Heaps

Considere a inserção do número **9**:



20	13	07	02	12	**09**

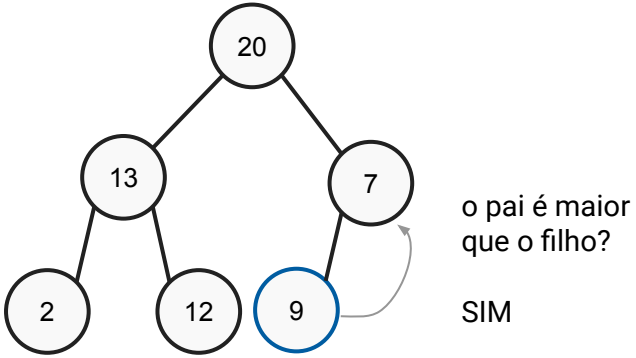
De início, colocamos ele no primeiro espaço vago, no final do array.



20	13	07	02	12	**09**

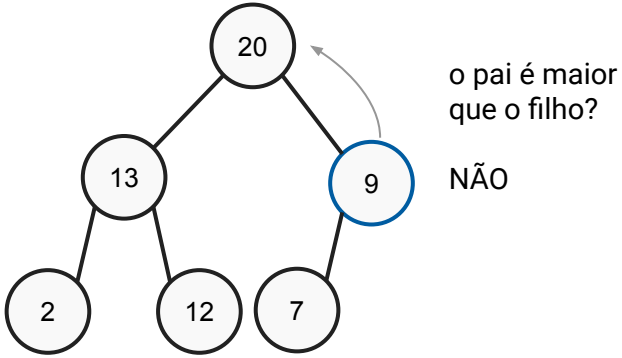
Inserção em Heaps

Em seguida, checamos a propriedade max-heap, **de baixo para cima**.



20	13	07	02	12	**09**

Havendo violação, troca-se pai e filho. Em seguida, checa-se novamente.



20	13	**09**	02	12	07

Inserção em heaps

Qual o melhor caso para esta lógica de inserção?

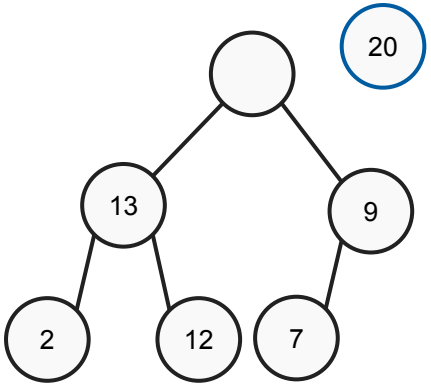
Se o elemento inserido for o menor de todos, ele não precisará subir em momento algum na lista, já estará no seu lugar ao ser inserido. Torna-se uma operação constante **O(1)**.

E qual o pior caso para esta lógica de inserção?

Se o elemento inserido for o maior de todos, serão feitas trocas até que o elemento chegue na raiz. Isto equivale a percorrer a altura completa da árvore, o que nos indica que a complexidade desta operação é logarítmica **O(logn)**.

Remoção em Heaps

A remoção na heap ocorre **sempre pelo topo**. Removemos então o **20**:

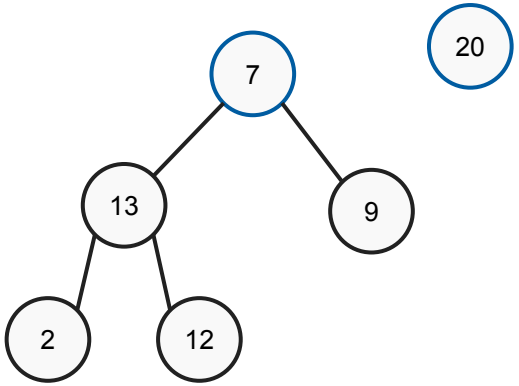


o pai é maior que o filho?

SIM

20		13	09	02	12	07

Escolhemos o último elemento do array e o promovemos para raiz:

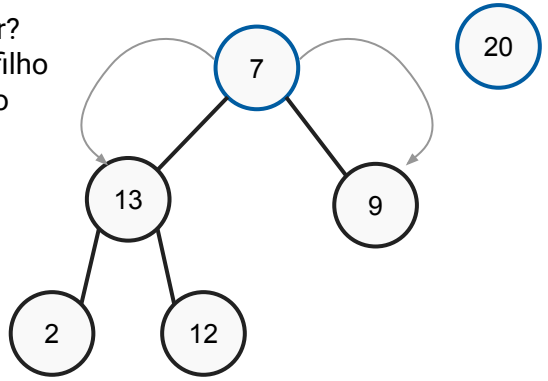


20

Remoção em Heaps

Com o novo nó raiz, verificamos se a propriedade max-heap é mantida:

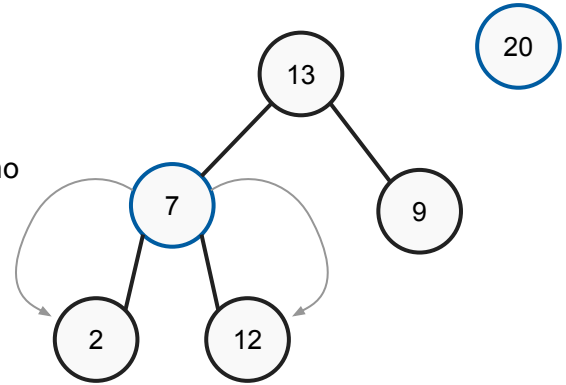
Quem é maior?
O nó atual, o filho esquerdo ou o direito?



20	07	13	09	02	12

Decidido o maior, trocamos o elemento, e percorremos o lado onde a troca ocorreu com a mesma lógica, **de cima para baixo**:

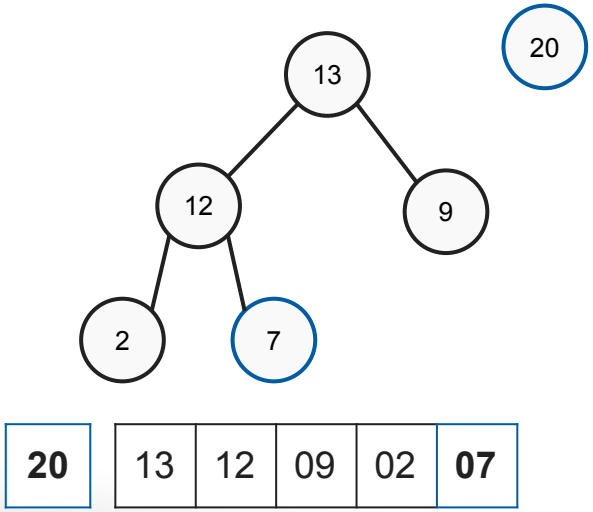
Quem é maior?
O nó atual, o filho esquerdo ou o direito?



20	13	07	09	02	12

Remoção em Heaps

Ao final, termos descido com o elemento o quanto for necessário para se manter a propriedade max-heap.



Nota: Sabemos que o 13 agora é o maior elemento, porém ele ainda é menor que 20. Se inserirmos todos os elementos que removermos sequencialmente em uma nova lista, teremos...

...um vetor ordenado 👀

Remoção em heaps

Qual o custo desta lógica de remoção?

A remoção do elemento mais no topo é de custo constante. Re-posicionar a nova raiz com o topo, porém, exige *novamente* o percurso de **no máximo a altura da árvore**. Portanto, a remoção também é **O(logn)**.

Na próxima aula

- Ordenação por Heap;
- Heapify: construção de uma heap a partir de um vetor já existente;

Obrigado!

Dúvidas ou sugestões:

mateus_m_luna@ufg.br ou
andre_moura@ufg.br

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Estrutura de Dados Heap 1

Usando uma Fila de Prioridades para Ordenação

Prof. Mateus M. Luna[1] **Prof. André L. Moura**[2]

1. mateus_m_luna@ufg.br
2. andre_moura@ufg.br

Material elaborado em parceria com as professoras Telma e Juliana Félix

2022

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS