árvores de busca: árvores binárias de busca, árvore AVL, árvore rubro negra, e árvore-B.

link do doc :
Definição de árvores e BSTs
https://algoritmosempython.com.br/cursos/algoritmos-python/estruturas-dados/arvores/

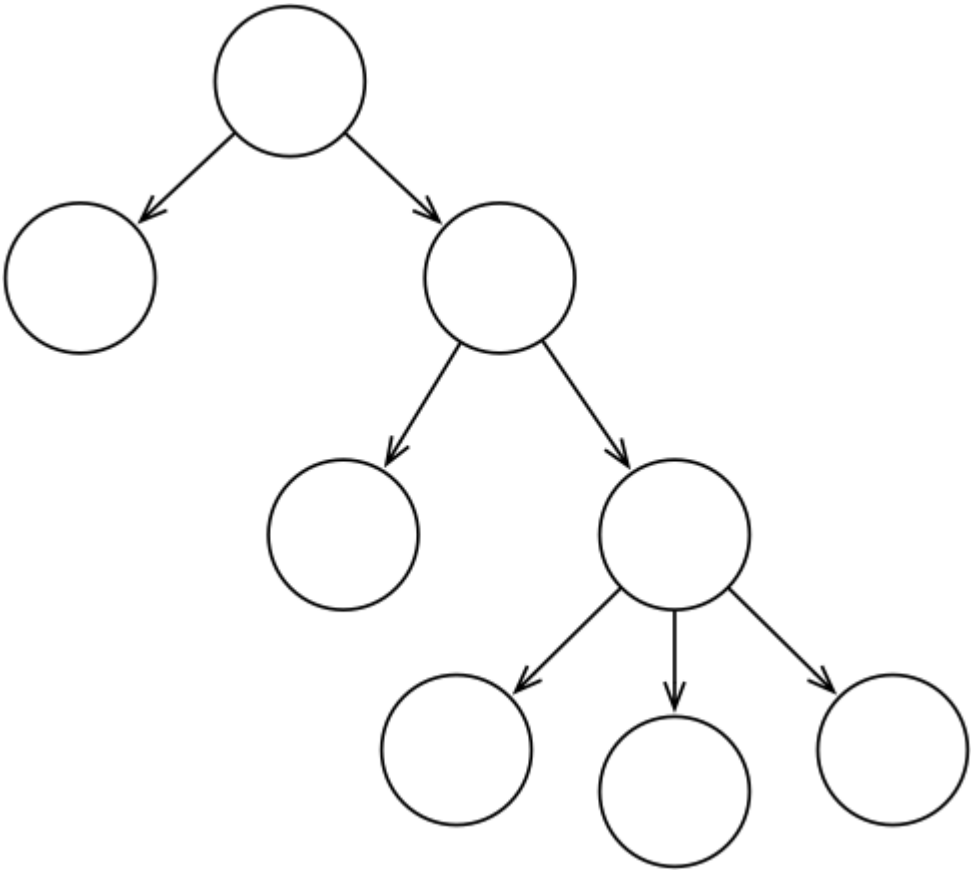
Árvores AVL
http://tiagodemelo.info/wp-content/uploads/2020/09/arvores-avl10.pdf

Árvores rubro negra

Árvores-B

-Árvores:

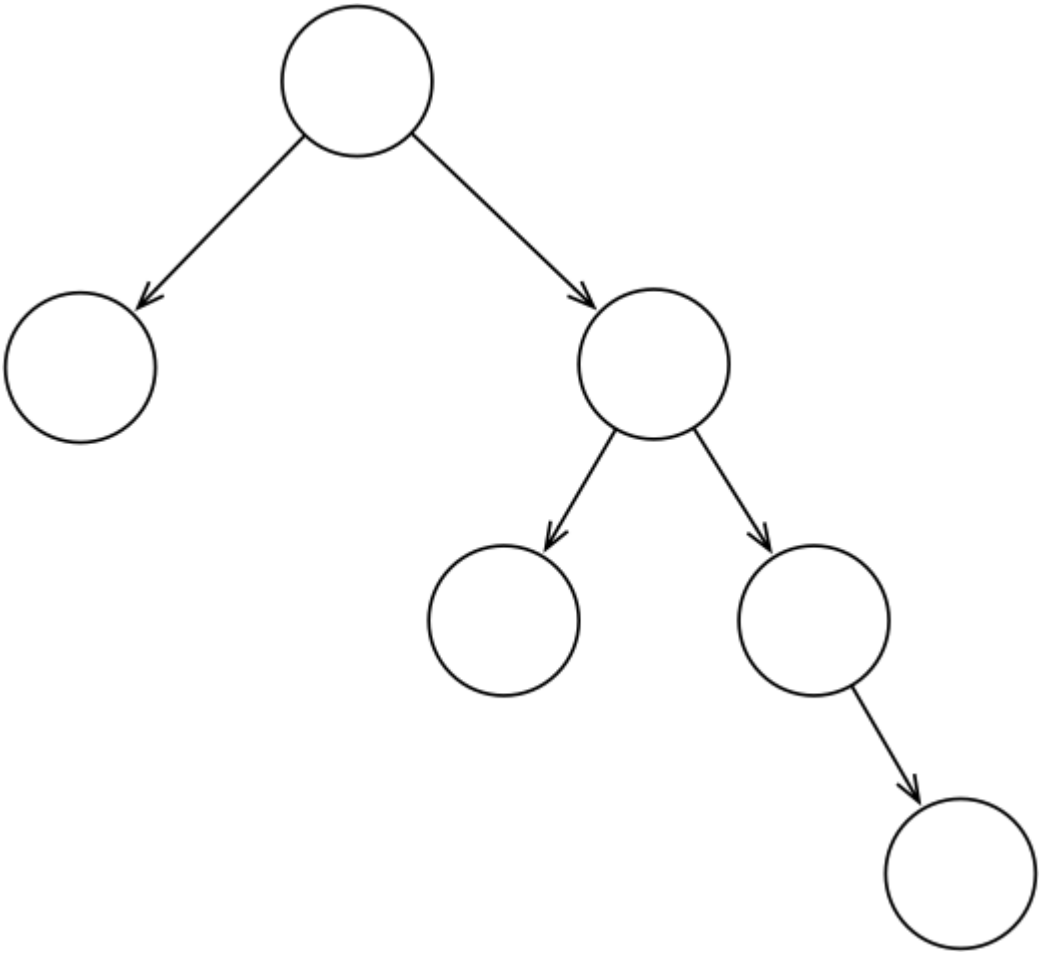
- São estruturas de dados hierárquicas
- Formada por um conjunto de elementos chamados de nós ou vértices
- Os nós são conectados por arestas
- A árvore tem nível de nós, sendo a Raiz, nó que cria a árvore, nível 0



- Apresentam relação de pais e filhos, por exemplo os nós do nivel 1 são filhos do nó do nível 0(raiz), e esta não tem pais.

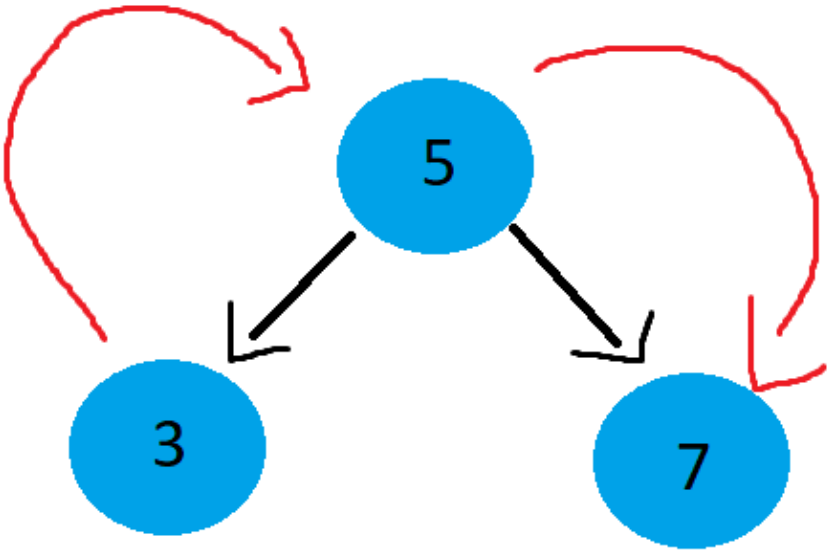
-Árvore binárias:

- Cada nodo pode ter no máximo dois filhos
- Pode ser dividida em várias sub-árvores
- “A raiz da árvore possui dois filhos, um à direita e outro à esquerda, que por sua vez são raízes de duas sub-árvores. Cada uma dessas sub-árvores possui uma sub-árvore esquerda e uma sub-árvore direita, seguindo esse mesmo raciocínio.”
- Na prática, nós tem um valor chamado chave e um apontador para cada filho
- Esses apontadores representam as arestas

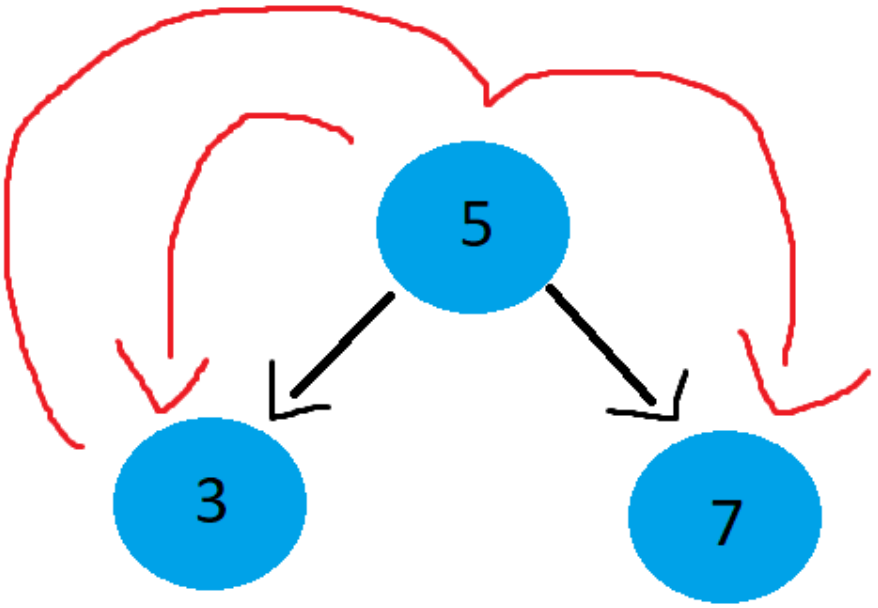


-Árvore binárias de busca(BST):

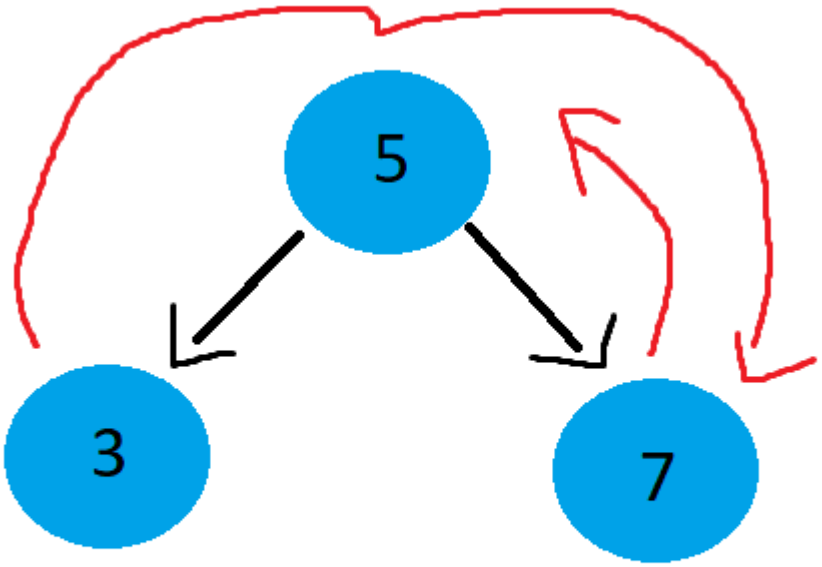
- Tem duas propriedades que as distinguem, e isso é em relação ao valor de seus nós
- Dado qualquer nó, seu filho a esquerda será menor que o pai e o filho à direita será maior
- 3 caminhamentos : em ordem , pós-ordem, pré-ordem



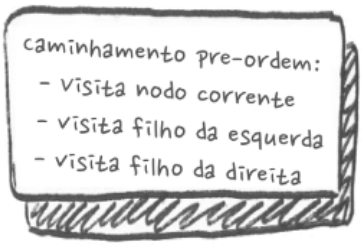
Caminhamento em Ordem (ordem crescente dos nós)

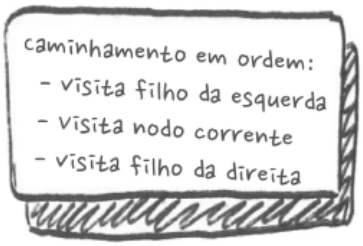


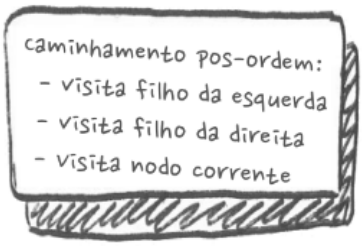
Caminhamento pré-ordem (Corrente, esquerda,direita)



Caminhamento pós-ordem (Esq,Direita,Corrente)

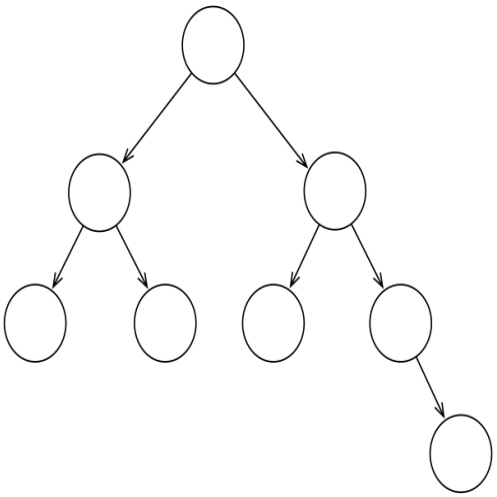




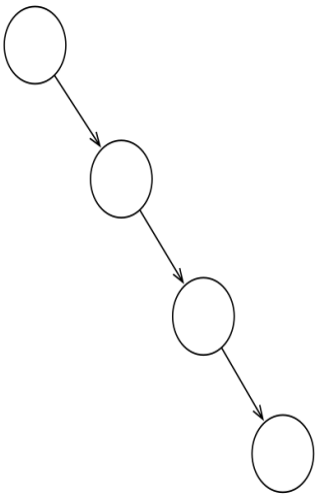


- Tem algoritmos de inserção e deleção
- Possui algoritmos de busca, para achar chaves específicas(valor dos nós)
- A profundidade de um nodo é o número de níveis da raiz até aquele nodo
- Uma BST é balanceada se a diferença da profundidade de duas folhas quaisquer é no máximo 1.
- (a) está balanceada, (b) não está

a)



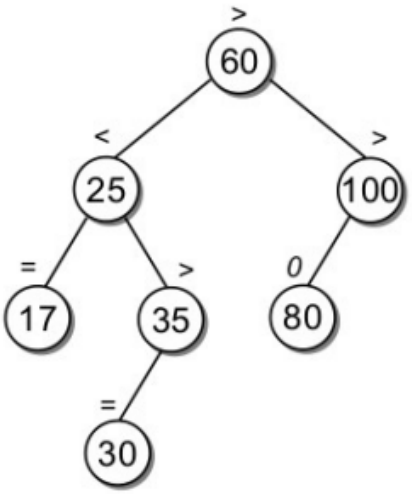
b)

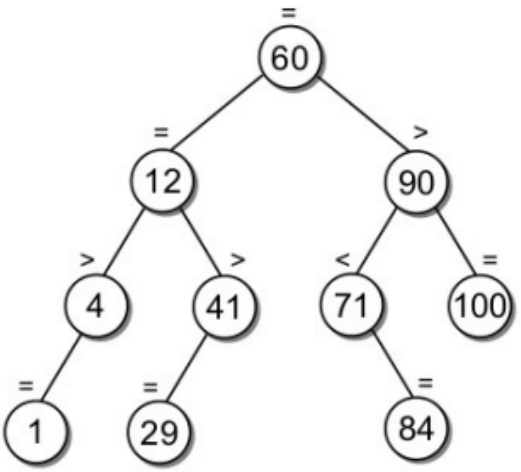


- B funciona com se fosse uma lista encadeada, pela falta de complexidade na hora da busca
- A complexidade de A é O(Log n), no melhor caso; no pior caso será n
- A complexidade de B é O(n)

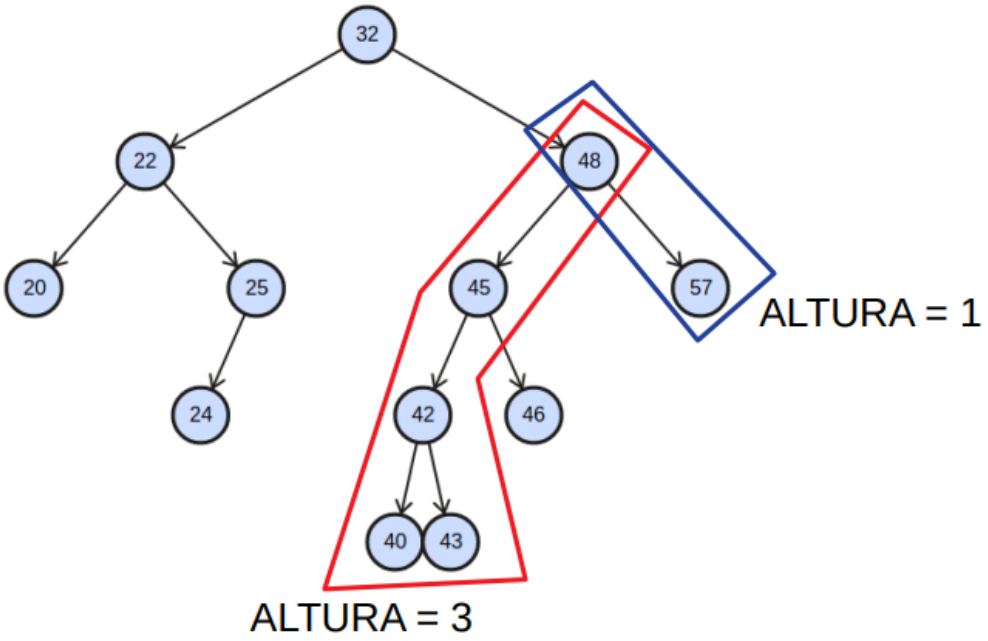
-Árvore AVL:

- Em uma árvore desbalanceada de 10 mil nós, 500 comparações são necessárias para efetuar uma busca, já na AVL a média desce para 14.
- A árvore AVL foi inventada por G. M. Adelson Velskii e Y. M. Landis em 1962.
- O objetivo dessa árvore é garantir que a altura sempre esteja balanceada, ou seja é AVL se a diferença das alturas nas folhas forem apenas de 1





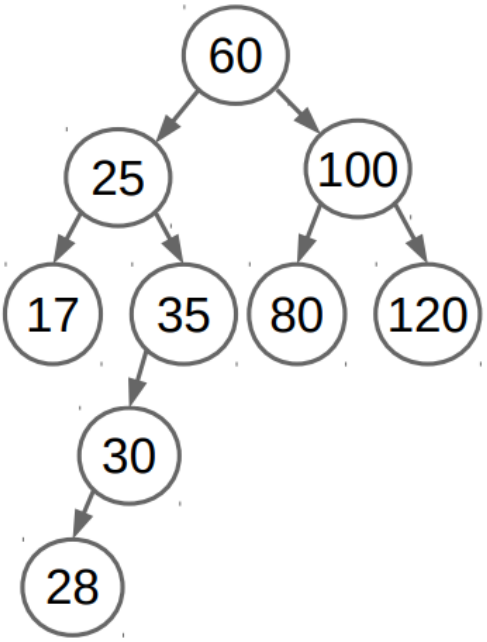
Exemplo de árvore que **não** é AVL:

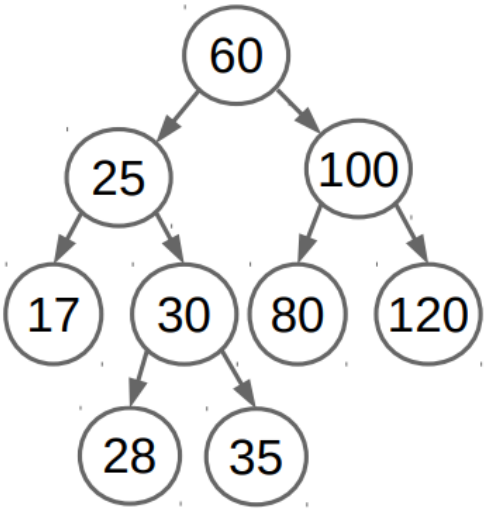


- Repare que deve-se contar as alturas para confirmar se é AVL ou não.
- Inserção e remoção devem ser modificadas na AVL para manter a árvore balanceada
- Se a inserção de um novo elemento causar um desbalanceamento, a árvore deve ser rebalanceada

Inserção

- Como a árvore ficaria balanceada?



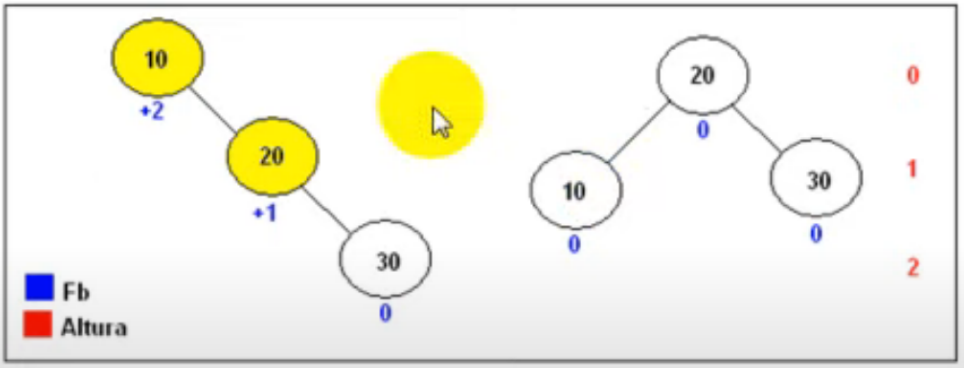


13 / 40

- Após a inserção de um novo elemento que desbalancearia as árvores, para resolver o problema basta rotacionar o nó raiz mais próximo do novo elemento (no caso acima, o 35). Esse nó é chamado de nó pivot
- Fator de Balanceamento: Cada nó tem um valor que deve ser -1, 0 ou 1; usamos ele para determinar se a árvore está desbalanceada.
- FatBal = h(sub-árvore direita) - h(sub-árvore esquerda)
- FatBal = 0 quando as duas sub-árvores têm a mesma altura

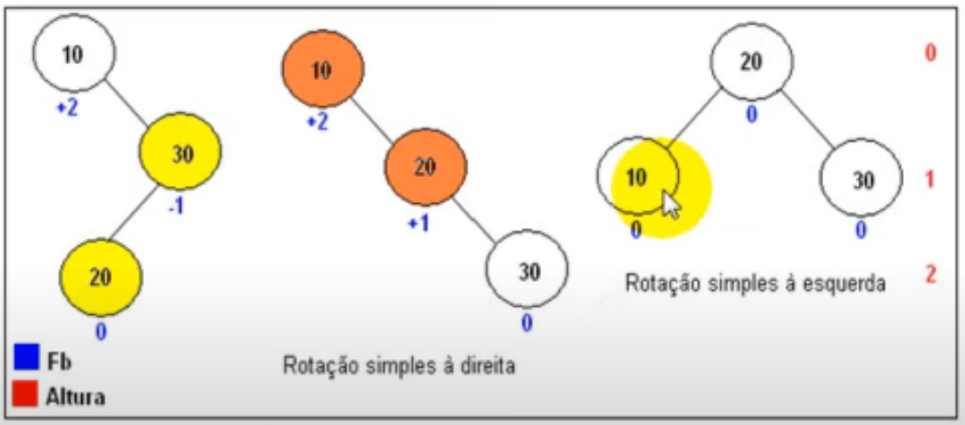
Rotações

- Rotação simples à esquerda



Rotações

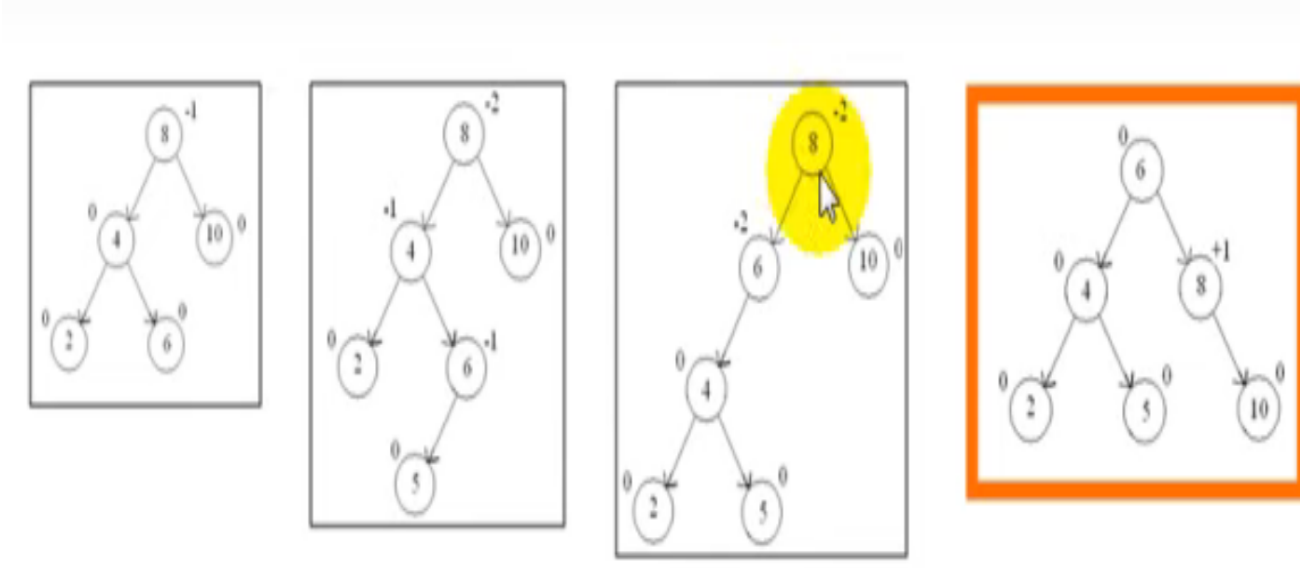
- Rotação dupla à esquerda



(rotação simples à direita + rotação simples à esquerda)

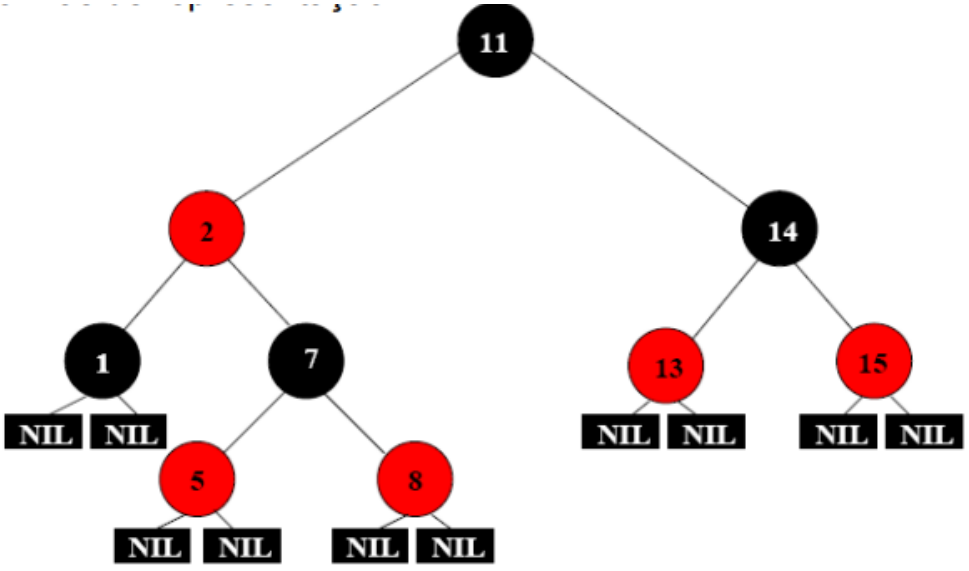
- Para saber quando fazer uma rotação simples ou dupla, basta ver o FatBal dos nós; se eles estão no mesmo módulo, então simples; caso alterne entre positivo e negativo (como no exemplo acima), então é uma dupla
- Se FatBal for positivo , a rotação é para a esquerda
- Se FatBal for negativo , a rotação é para a direita
- Rotação dupla à direita : -2 , +1 , 0

- Rotação dupla à esquerda : +2 , -1 , 0



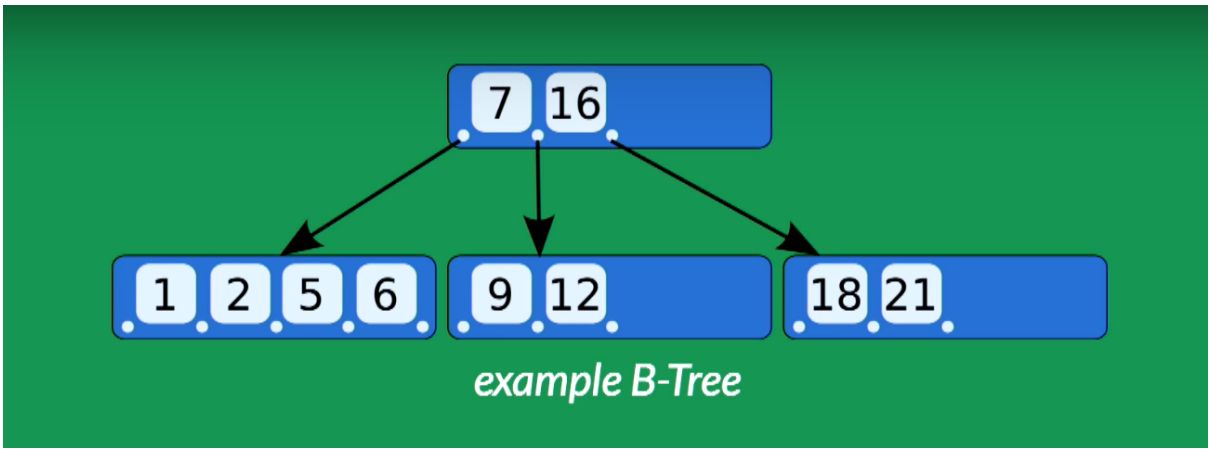
-Árvore Rubro-Negra:

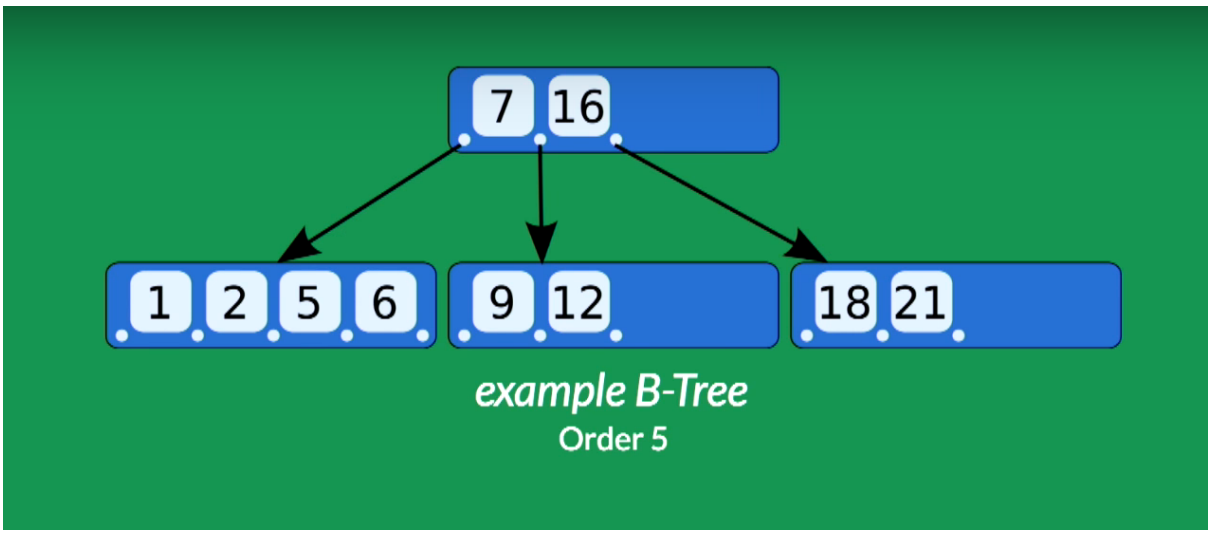
- Tem um bit a mais pra armazenar a informação de se o nó é vermelho ou preto
- Complexidade de O (log n)
- A raiz é preta (Nil)
- Toda folha é preta
- Se um nó é vermelho, seus filhos são pretos
- Não pode existir dois nós rubros consecutivos



- Caso não esteja balanceada, são realizadas rotações e / ou ajustes de cores
- Altura negra é o número de nós negros encontrados até qu8alquer nó folha descendente
- Todo nó inserido sempre é vermelho
-

-Árvore B:





COMPLEXIDADE É O(log(n))

*ver balanceamento (inserção e deleção) em AVL
*rubro negra e B
*talvez implementar o código no google colab
*revisao da matéria