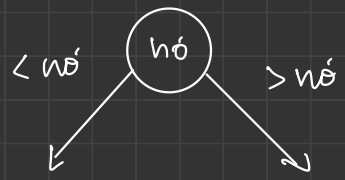
Árvore Binária de Pesquisa

Complexidade Temporal:

- **Inserção**: $O(\log n)$, n = nº de nós.
  pior caso: $O(n)$, caso esteja desbalanceada.
- **Pesquisa**: $O(\log n)$
  pior caso: $O(n)$
- **Remoção**: $O(\log n)$
  pior caso: $O(n)$

$O(\log n)$ se o numero de comparações for porporcional à altura da árvore

Complexidade espacial: $O(n)$, pq a cada novo elemento, cria um novo nó



AVL Tree

+ eficiente na pesquisa e inserção c balanceamento

Complexidade Temporal:

- Inserção: $O(\log n)$
- Pesquisa: $O(\log n)$
- Remoção: $O(\log n)$

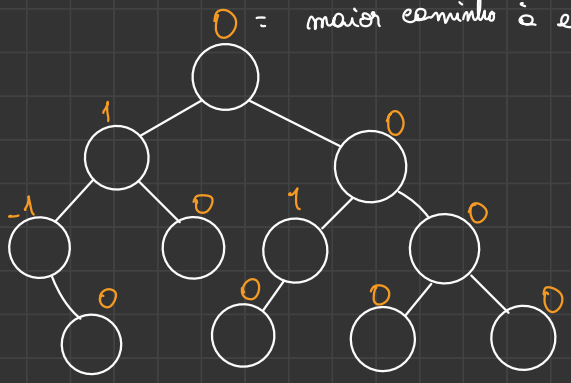
o balanceamento da árvore mantém a altura $\log n$, mantendo também o nº de comparações porporcional ao log de n elementos

busca, altura mantém o nº comparações pp. a $\log n$

a altura balanceada mantém o nº max de operações para a remoção porporcional ao log de n elementos

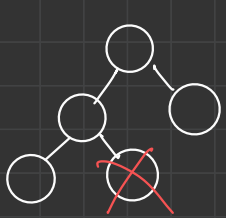
Complexidade espacial: $O(n)$

0 = maior caminho à esquerda - maior caminho à direita

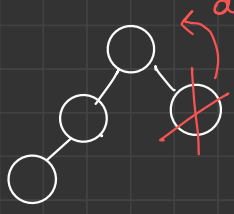


Complexidade temporal das rotações: $O(1)$

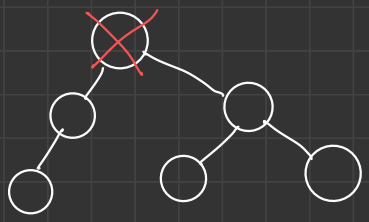
Delete:



como é folha é só apagar e verificar o balanceamento



dps o fator balanceamento fica a 2.



substituir pelo maior à esquerda ou menor à direita

Árvores Vermelha e Preta

Complexidade Temporal:

+ eficientes na inserção e remoção

- Inserção: $O(\log n)$
- Pesquisa: $O(\log n)$
- Remoção: $O(\log n)$

Complexidade espacial: $O(n)$

Regras na inserção:

1. a root é sempre preto
2. o novo nó é sempre vermelho
3. se o pai do novo nó for preto ✓
4. se o pai do novo nó for vermelho:

   se o irmão do pai for preto ou Null:
   → faz rotate e recolor

   se o irmão do pai for vermelho, recolor:
   o pai deste fica vermelho → se o pai do pai não for root, recolor

SPLAY Tree

+ eficiente para acesso a elementos mais frequentemente consultados

Complexidade Temporal:

- Inserção: $O(\log n)$
- Pesquisa: $O(\log n)$
- Remoção: $O(\log n)$

inserção → novo nó como root

Complexidade espacial: $O(n)$

função SPLAYing (trazer o elemento acedido mais perto da root)

1. se o node x é filho esquerdo ou direito do seu pai p
2. se p é root ou não
3. se o pai p é filho esquerdo ou direito do avô de x

- quando apaga, executa splaying nesse node, remove-o e divide a árvore em 2 e pega no maior do lado esquerdo e mete-o a root.