Balanceamento de Árvores Binárias de Busca

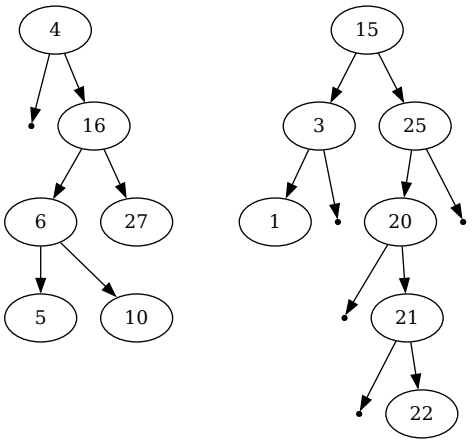
Algoritmos e Estruturas de Dados - 2025/01

Última atualização: 08/01/2025 14:31

Revisão

Vamos começar relembrando conceitos importantes para essa aula.

Exercício: Qual é a altura das árvores abaixo?



Exercício: Desenhe a árvore criada quando inserimos os seguintes valores em uma *ABB*. Note que estamos criando **duas árvores com os mesmos valores, mas inseridos em ordens diferentes**.

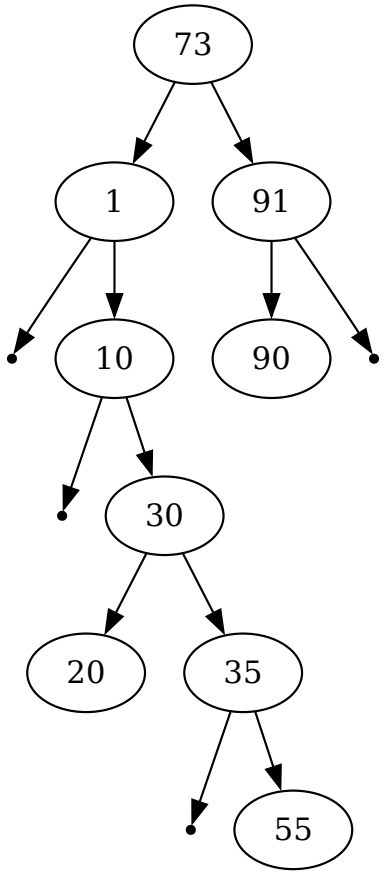
- `5 7 11 33 3 15 2`
- `2 7 5 11 3 33 15`

Balanceamento usando array auxiliar

Vamos relembrar o algoritmo visto na parte expositiva:

1. passe por todos elementos em-ordem, guardando o valor em um array
2. insira como raiz o elemento do meio do array
3. faça o passo *2* recursivamente para cada lado do array

Podemos interpretar esse algoritmo como um que encontra a melhor ordem de inserção. Vamos praticar com a árvore abaixo e simular a execução desses 3 passos.



Iremos fazer a simulação passo a passo.

Passo 1: Passe pelos nós da árvore em-ordem e coloque-os ao lado.

Com este resultado na mão, vamos começar a determinar uma ordem de inserção para que a árvore fique balanceada. Se ficar em dúvida, reveja o passo 2 do algoritmo acima.

Passo 2: Qual é o primeiro elemento a ser inserido?

Vamos agora inserir todos valores menores que a esse elemento. No caso, serão os elementos `1 10 20 30`.

Passo 3: Encontre novamente o elemento do meio e coloque-o ao lado. (Como estamos com 4 elementos, selecionamos o "do meio à esquerda")

Passo 4: Continue o algoritmo até processar todos os elementos *menores que a raiz*. Desenhe a árvore abaixo.

Faz diferença pegar o elemento à esquerda ou à direita?

Refaça os passos 2 a 4 pegando sempre o elemento da direita e desenha a àrvore abaixo. A altura da árvore resultante é igual ao seu desenho acima?

Passo n: Repita o processo para os elementos maiores que `35` (a raiz), usando o critério de escolher a mediana à esquerda. Desenhe abaixo o resultado final.

Agora vamos formalizar as duas partes desse algoritmo.

Parte 1: `TREE-TO-ARRAY(R, AUX, IDX)` recebe uma árvore `R`, um array `AUX` que com tamanho suficiente para todos os nós de `R` e `IDX` sendo o índice atual que iremos preencher em `AUX`. O algoritmo devolve o valor atualizado de `IDX`.

Algoritmo `TREE-TO-ARRAY`

Parte 2: `REBALANCEIA(R)` recebe uma árvore `R` e devolve uma nova árvore balanceada `R2` contendo os mesmos valores que `R`. Você pode chamar `TREE-TO-ARRAY`.

Algoritmo `REBALANCEIA`