Algoritmo e Estrutura de Dados II
COM-112

Aula 11

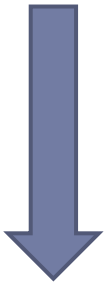
Vanessa Souza

Universidade Federal de Itajubá – Instituto de Matemática e Computação

Remoção em árvores binária de pesquisa

Remoção

- Em uma árvore binária de pesquisa, para removermos um nó *X* de uma árvore de busca binária preservando suas propriedades, precisamos considerar três casos:
  - *X* não possui nenhum filhos (folha).
  - *X* possui apenas um filho.
  - *X* possui dois filhos.



+ complexidade

Remoção

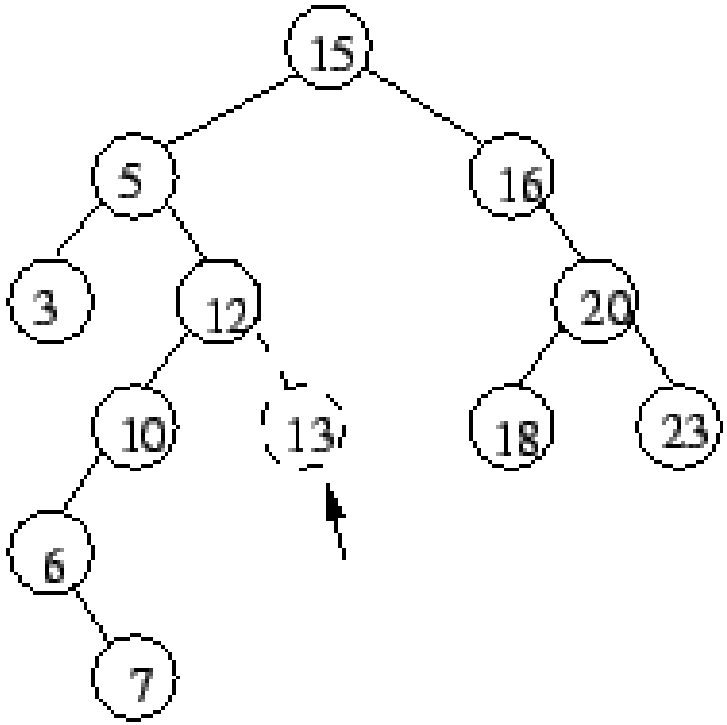
- Caso 3 : O nó tem dois filhos
  - O caso mais complicado ocorre quando o nó a ser removido tem dois filhos (ou seja, é pai de uma subárvore esquerda e de uma subárvore direita).
  - Um algoritmo possível é conhecido como remoção por cópia.

Remoção

- Caso 3 : O nó tem dois filhos
  - O algoritmo consiste em encontrar o sucessor (ou predecessor) imediato do nó a ser removido e substituir este nó por seu sucessor (ou predecessor).
    - O sucessor de um nó X é o nó com a menor chave maior que X
    - Filho mais à esquerda, da subárvore da direita de X
  - Se o sucessor for folha, basta copiar a chave e os dados satélites do sucessor para o lugar do nó X e remover o sucessor.
  - Se o sucessor tiver um filho, faz a remoção do sucessor e copia os dados do sucessor para o lugar do nó X.

Remoção

- Exercício
  - Dada a árvore abaixo, remover sucessivamente os nós 13, 16, 5 e 20.



Remoção

- Algoritmo
  - void removeNo (no *A, chave);
  - void removeFolha (no *A);
  - void remove1Filho (no *A);

Árvores Adelson-Velskii and Landis (AVL)

REMOÇÃO EM ÁRVORES AVL

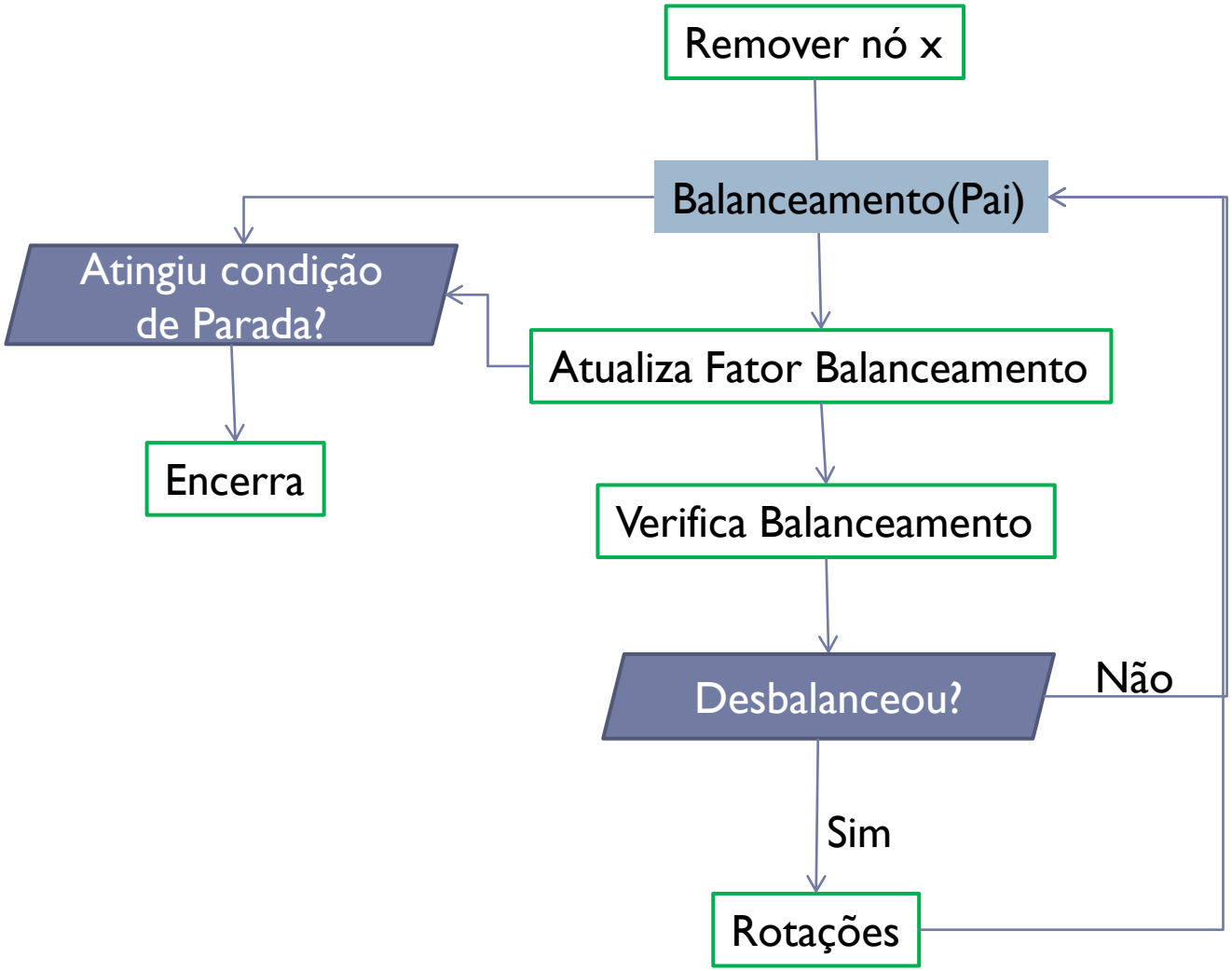
Árvores AVL – Remoção

- Considere que o elemento a ser removido encontra-se na raiz de uma árvore T:
  - A raiz não possui filhos
    - remover a raiz e anular T;
  - A raiz possui um único filho
    - remover a raiz e substituí-la por seu filho;
  - A raiz possui dois filhos
    - escolher o nó que armazena o menor elemento na subárvore direita (sucessor) e substituir a raiz por ele

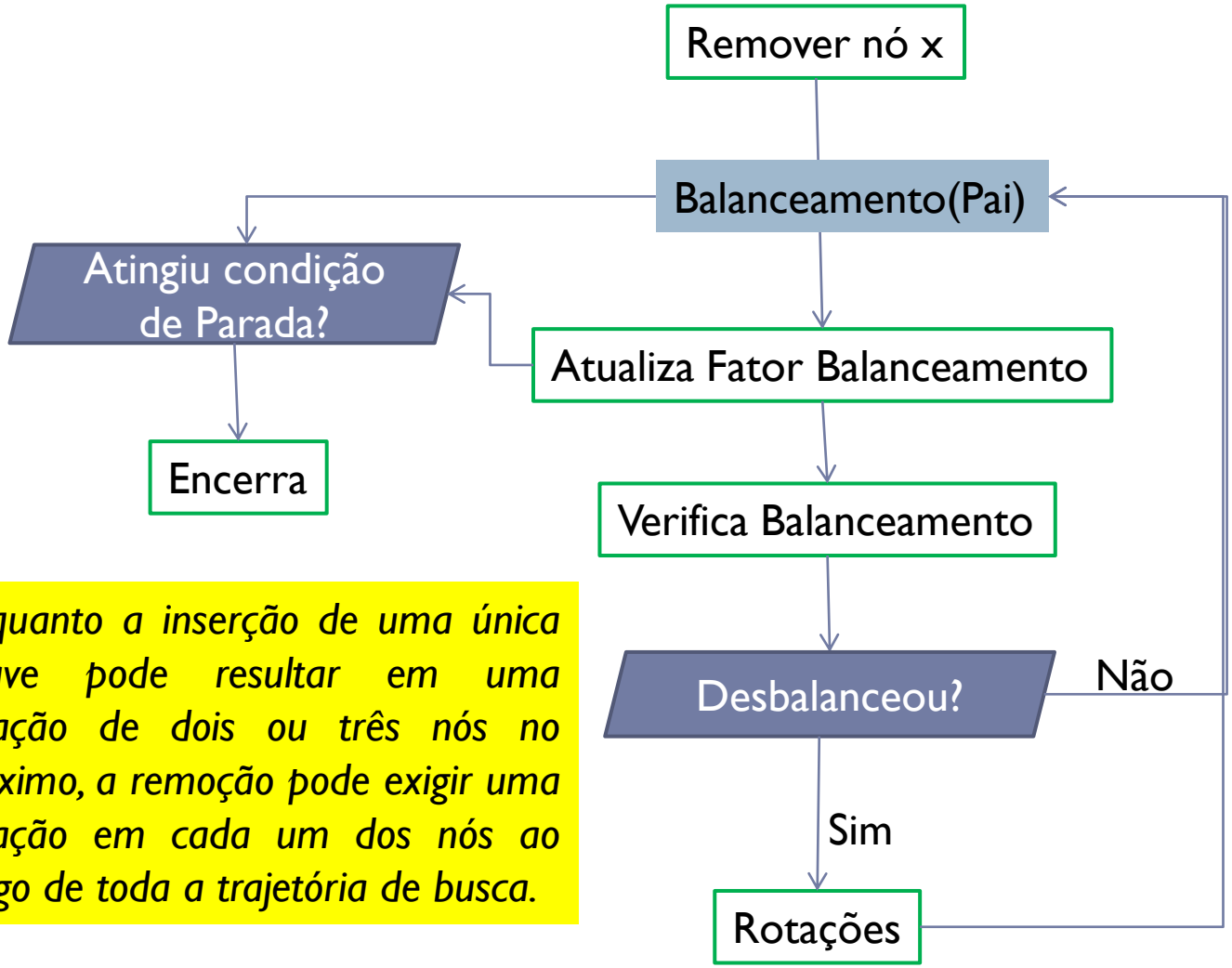
Árvores AVL – Remoção

- Assim como na inserção, a remoção pode ocasionar:
  - A diminuição da altura da árvore e possível desbalanceamento da mesma
  - Alteração dos fatores de balanceamento de seus nós

Implementação da AVL – REMOÇÃO



Implementação da AVL – REMOÇÃO



Enquanto a inserção de uma única chave pode resultar em uma rotação de dois ou três nós no máximo, a remoção pode exigir uma rotação em cada um dos nós ao longo de toda a trajetória de busca.

Exercícios

- Mostre (desenhe) uma árvore AVL após a inserção dos seguintes elementos, em ordem: 20, 15, 25, 12, 17, 30, 26, 16

- Mostre como ficará a árvore acima após a remoção dos seguintes elementos, na ordem abaixo
  - 25, 30, 26

Exercícios

- Mostre (desenhe) uma árvore AVL após a inserção dos seguintes elementos, em ordem: 20, 15, 25, 12, 17, 30, 26, 16, 18

- Mostre como ficará a árvore acima após a remoção dos seguintes elementos, na ordem abaixo
  - 12

Remoção em AVL

- Algoritmo
  - void removeNoBalanceado (no *A, chave);
  - void BalanceamentoRemocao (no *A);

Remoção - Implementação

- void removeNoBalanceado (no *A, chave);
  - Remove um nó da árvore
  - Chama a função BalanceamentoRemocao para o pai do nó removido

Remoção - Implementação

- void BalanceamentoRemocao (no *A);
  - Função recursiva que atualiza o FB dos nós da árvore após uma remoção
    - Possui 2 critérios de parada
      - Chegou à raiz da árvore
      - Após o balanceamento encontrou um nó cujo FB seja igual a 1 ou -1
    - No caso de a árvore ter sofrido desbalanceamento com a remoção, chama as rotações apropriadas
    - Ajusta o Fator de Balanceamento

Remoção - Implementação

- void BalanceamentoRemocao (no *A);
  - Ajusta o Fator de Balanceamento

Rotação	Ajuste do FB
Simples à Esquerda	Se (filho->fb == 0) Filho->fb = -1 No->fb = 1
Direita – Esquerda	Se (neto->fb == 1) → no->fb = -1 Se (neto->fb == -1) → filho->fb = 1

  - Simétrico para rotações à direita

Remoção em AVL

- Note que a operação de remoção pode ser realizada em tempo ***O(log(n))***.

- Na remoção de um elemento em uma árvore AVL, pode haver a necessidade de realizar mais de duas rotações (o que não acontece na inserção), podendo se estender para uma rotação em cada nível (***O(log(n))***) no pior caso.

AVL

- Há um custo adicional para manter uma árvore balanceada, mesmo assim garantindo $O(\log_2 n)$, mesmo no pior caso, para todas as operações

- Testes empíricos provaram que:
  - Uma rotação é necessária a cada duas inserções
  - Uma rotação é necessária a cada cinco remoções

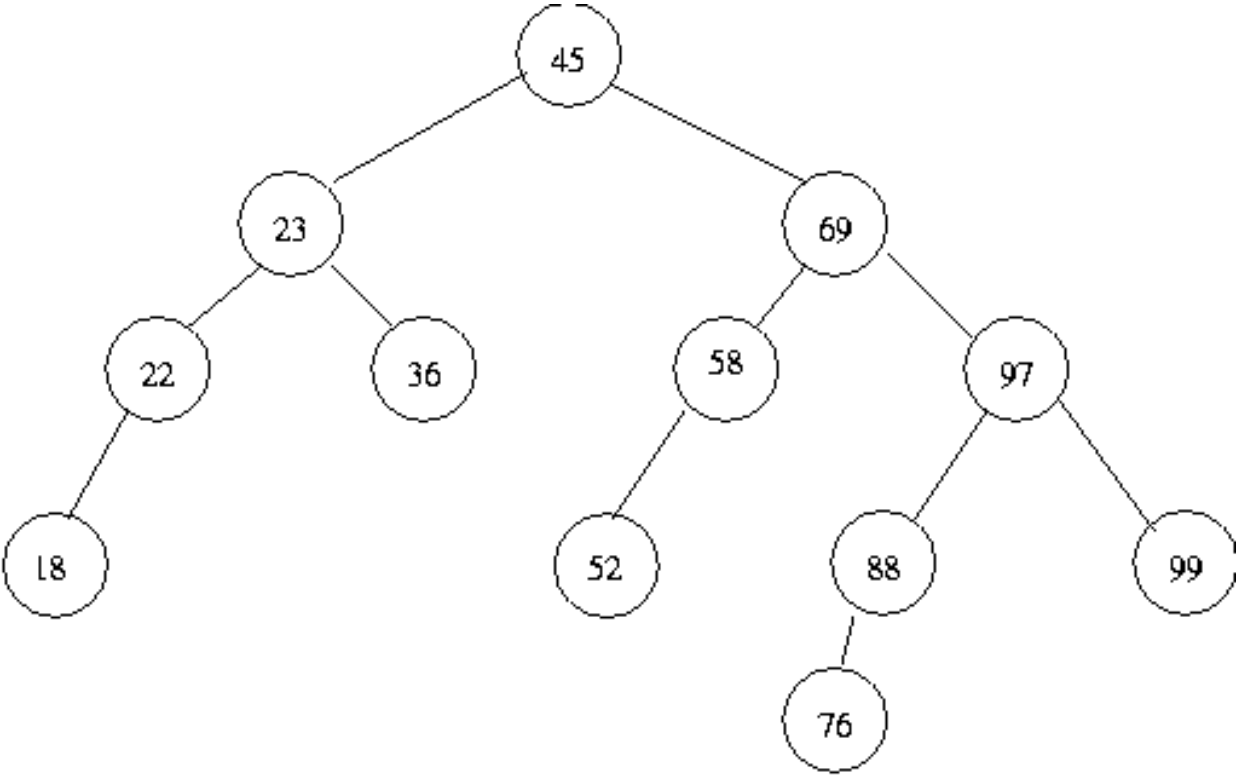
- A remoção em árvore balanceada é tão simples (ou tão complexa) quanto a inserção

AVL

- Árvore AVL é uma boa opção como ED para buscas de chaves, SE a árvore é balanceada => tempo proporcional a $O(\log_2 n)$.

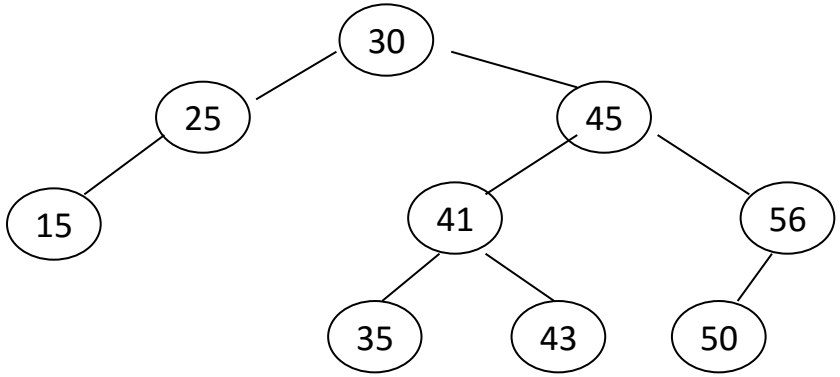
- Inserções e Eliminações causam desbalanceamento.
  - Melhor se aleatórias (não ordenadas) para evitar linearizações

Exercícios



- Insira 105
- Insira 20
- Remova 45
- Remova 58

Exercícios



- Realize, na árvore ao lado, as inserções das seguintes chaves 49, 60, 65, e em seguida a remoção das chaves 45 e 41, escolhendo necessariamente o predecessor para a posição da chave removida.

- Mostre todas as rotações e o formato da árvore após cada operação.

Exercício

- Apresente duas maneiras distintas de inserir as chaves 10, 20, 30, 40 e 50 em uma árvore AVL inicialmente vazia de modo que só ocorram exatamente 2 rotações duplas no mesmo sentido. Justifique sua resposta, realizando as inserções.