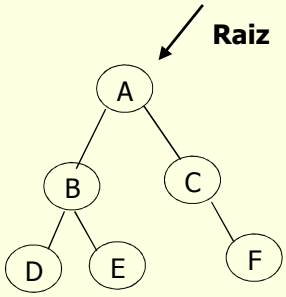
Árvores binárias de busca

SCC-214 – Projeto de Algoritmos

Thiago A. S. Pardo

Árvore binárias

- Árvores de grau 2, isto é, cada nó tem dois filhos, no máximo



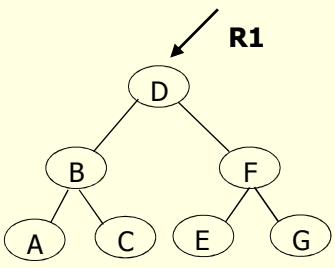
Terminologia:
- filho esquerdo
- filho direito
- informação

Árvores binárias de busca (ABB)

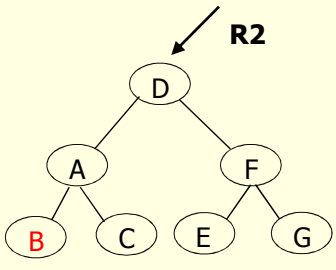
- Também chamadas "árvores de pesquisa" ou "árvores ordenadas"
- *Definição*
  - Uma árvore binária com raiz R é uma ABB se:
    - a chave (informação) de cada nó da subárvore esquerda de R é menor do que a chave do nó R (em ordem alfabética, por exemplo)
    - a chave de cada nó da subárvore direita de R é maior do que a chave do nó R
    - as subárvores esquerda e direita também são ABBs

ABB

- Exemplos



é ABB



não é ABB

ABB

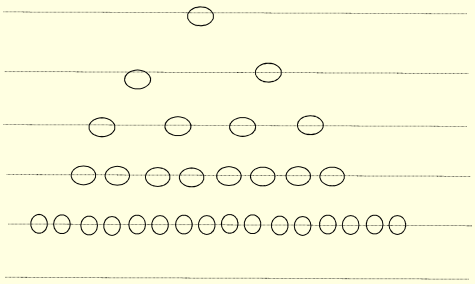
- Por que uma ABB é boa?
- Imagine a situação
  - Sistema de votação por telefone (“Você decide”)
    - Cada número só pode votar uma vez
    - Um sistema deve armazenar todos os números que já ligaram
    - A cada nova ligação, deve-se consultar o sistema para verificar se aquele número já votou; o voto é computado apenas se o número ainda não votou
    - A votação deve ter resultado on-line

ABB

- Por que uma ABB é boa?
- Solução com ABBs
  - Cada número de telefone é armazenado em uma ABB
  - Suponha que em um determinado momento, a ABB tenha 1 milhão de telefones armazenados
  - Surge nova ligação e é preciso saber se o número está ou não na árvore (se já votou ou não)

ABB

- Por que uma ABB é boa?
- Considere uma ABB com chaves uniformemente distribuídas (árvore cheia)

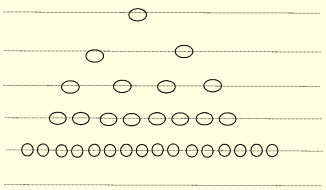


ABB

- Por que uma ABB é boa?
- *Responda*
  - Quantos elementos cabem em uma árvore de N níveis, como a anterior?
  - Como achar um elemento em uma árvore assim a partir da raiz?
  - Quantos nós se tem que visitar, no máximo, para achar o telefone na árvore, ou ter certeza de que ele não está na árvore?

ABB

- Por que uma ABB é boa?



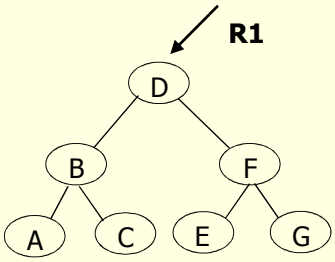
Nível	Quantos cabem
1	1
2	3
3	7
4	15
...	...
N	**$2^N - 1$**
10	1.024
13	8.192
16	65.536
18	262.144
20	1 milhão
30	1 bilhão
	...

ABB

- Por que uma ABB é boa?
- Para se **buscar** em uma ABB
  - Em cada nó, compara-se o elemento buscado com o elemento presente
    - Se menor, percorre-se a subárvore esquerda
    - Se maior, percorre-se subárvore direita
  - Desce-se verticalmente até as folhas, no pior caso, sem passar por mais de um nó em um mesmo nível
  - Portanto, no pior caso, a busca passa por tantos nós quanto for a altura da árvore

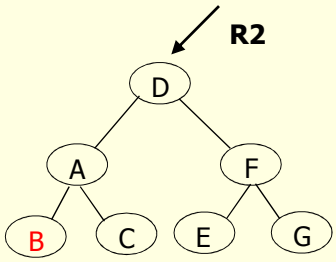
ABB

- Exemplo: busca pelo elemento E nas árvores abaixo



é ABB

3 consultas



não é ABB

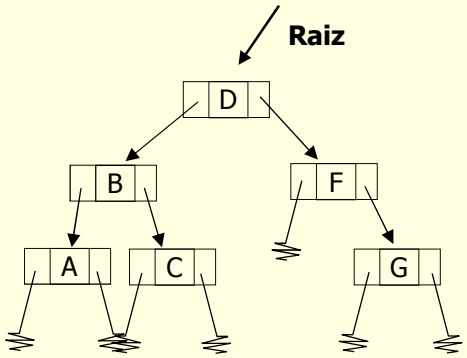
6 consultas

ABB

- Por que uma ABB é boa?
  - Buscas muito rápidas!!!

ABB

- Representação



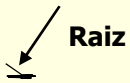


ABB vazia

ABB

- Declaração similar às árvores binárias convencionais
  - Manipulação diferente

```
typedef struct no {
        int info;
        struct no *esq, *dir;
} no;

no *raiz;
```

ABB

- Operações sobre a ABB
  - Devem considerar a ordenação dos elementos da árvore
    - Por exemplo, na inserção, deve-se procurar pelo local certo na árvore para se inserir um elemento
- Exercício
  - Construa a partir do início uma ABB com os elementos K, E, C, P, G, F, A, T, M, U, V, X, Z

ABB

- Operações básicas
  - Está na árvore?
  - Inserção
  - Remoção

ABB

- Está na árvore?
  - Comparando o parâmetro “chave” com a informação no nó “raiz”, 4 casos podem ocorrer
    - A árvore é vazia => a chave não está na árvore => fim do algoritmo
    - Elemento da raiz = chave => achou o elemento (está no nó raiz) => fim do algoritmo
    - Chave < elemento da raiz => chave pode estar na subárvore esquerda
    - chave > info(raiz) => chave pode estar na subárvore direita
  - Pergunta: quais os casos que podem ocorrer para a subárvore esquerda? E para a subárvore direita?

ABB

- Exercício
  - Implementação da sub-rotina de busca de um elemento na árvore

ABB

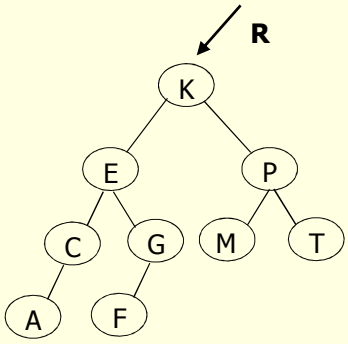
- Inserção
  - Estratégia geral
    - Inserir elementos como nós folha (sem filhos)
    - Procurar o lugar certo e então inserir
  - Comparando o parâmetro “chave” com a informação no nó “raiz”, 4 casos podem ocorrer
    - A árvore é vazia => insere o elemento, que passará a ser a raiz; fim do algoritmo
    - Elemento da raiz = chave => o elemento já está na árvore; fim do algoritmo
    - Chave < elemento da raiz => insere na subárvore esquerda
    - Chave > elemento da raiz => insere na subárvore direita

ABB

- Exercício
  - Implementação da sub-rotina de inserção de um elemento na árvore

ABB

- Remoção
  - Para a árvore abaixo, remova os elementos T, C e K, nesta ordem



ABB

- Remoção
  - Caso 1 (remover T): o nó a ser removido (R) não tem filhos
    - Remove-se o nó
    - R aponta para NULL
  - Caso 2 (remover C): o nó a ser removido tem 1 único filho
    - Remove-se o nó
    - “Puxa-se” o filho para o lugar do pai
  - Caso 3 (remover K): o nó a ser removido tem 2 filhos
    - Acha-se a maior chave da subárvore esquerda
    - R recebe o valor dessa chave
    - Remove-se a maior chave da subárvore esquerda

ABB

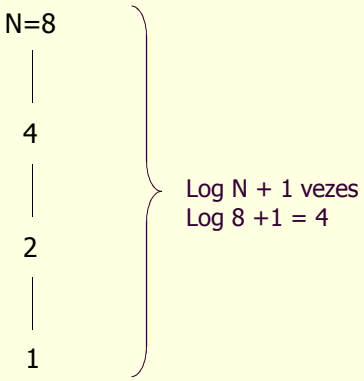
- Exercício para casa
  - Implementação da sub-rotina de remoção de um elemento da árvore

ABB

- Vantagens
  - Se **nós espalhados uniformemente**, consulta rápida para grande quantidade de dados
    - Divide-se o espaço de busca restante em dois em cada passo da busca
    - O(Log N)

ABB

- Análise do algoritmo: O(Log N)



ABB

- Contra-exemplo
  - Inserção dos elementos na ordem em que aparecem
    - A, B, C, D, E, ..., Z
    - 1000, 999, 998, ..., 1

ABB

- O desbalanceamento da árvore pode tornar a busca tão ineficiente quanto a busca seqüencial (no pior caso)
  - O(N)
- Solução?

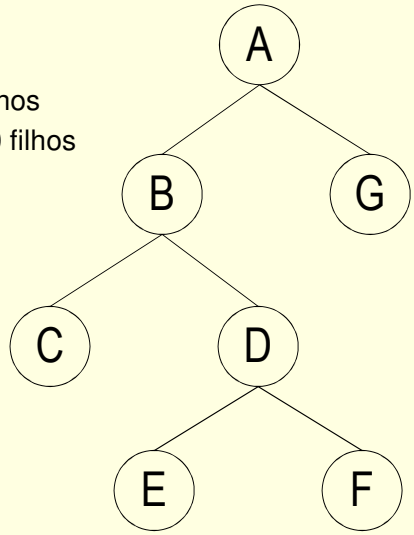
ABB

- O desbalanceamento da árvore pode tornar a busca tão ineficiente quanto a busca seqüencial (no pior caso)
  - O(N)
- Solução?

  Balanceamento da árvore quando necessário!

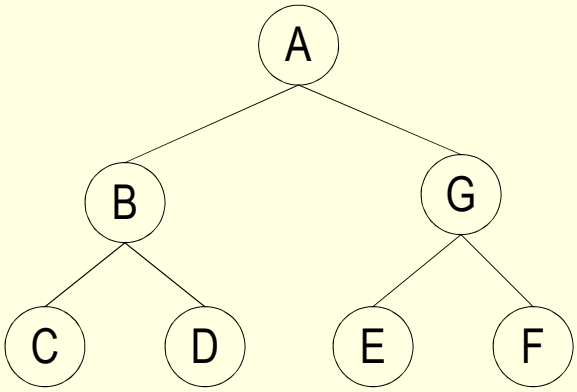
Conceitos

- Árvore estritamente binária
  - Os nós tem 0 ou 2 filhos
    - Todo nó interno tem 2 filhos
    - Somente as folhas têm 0 filhos



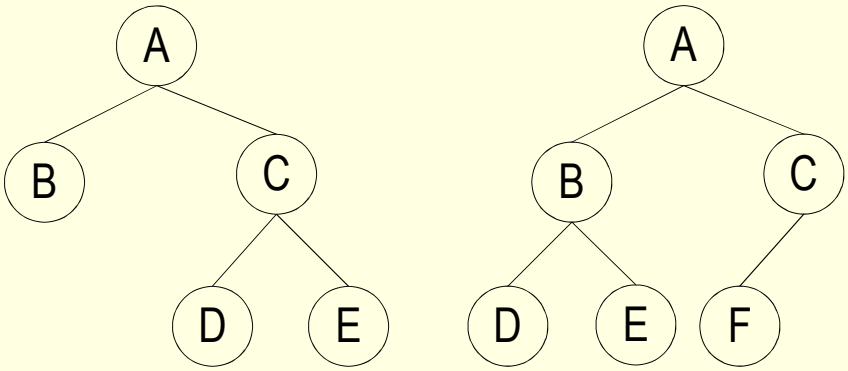
Conceitos

- Árvore binária completa (ou cheia)
  - Árvore estritamente binária
  - Todos os nós folha no mesmo nível



Conceitos

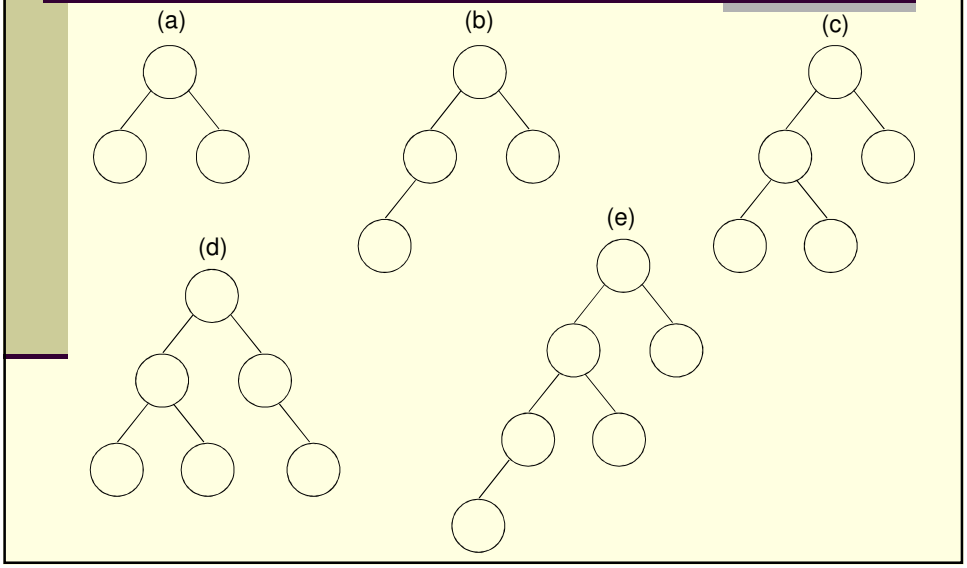
- Uma árvore binária é dita balanceada se, para cada nó, as alturas de suas duas subárvores diferem de, no máximo, 1



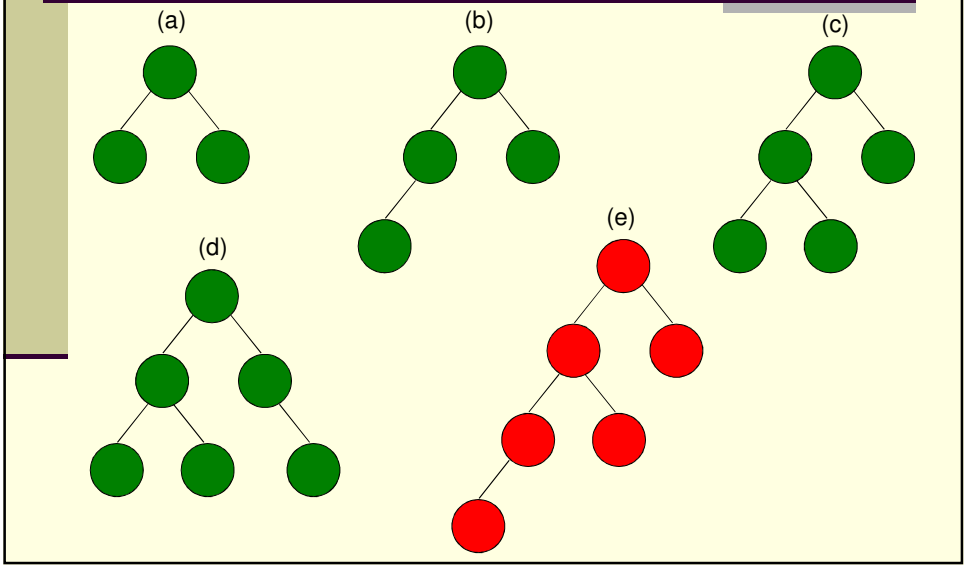
AVL

- Árvore binária de busca balanceada
  - Para cada nó, as alturas das subárvores diferem em 1, no máximo
  - Proposta em 1962 pelos matemáticos russos G.M. Adelson-Velskki e E.M. Landis
    - Métodos de inserção e remoção de elementos da árvore de forma que ela fique balanceada

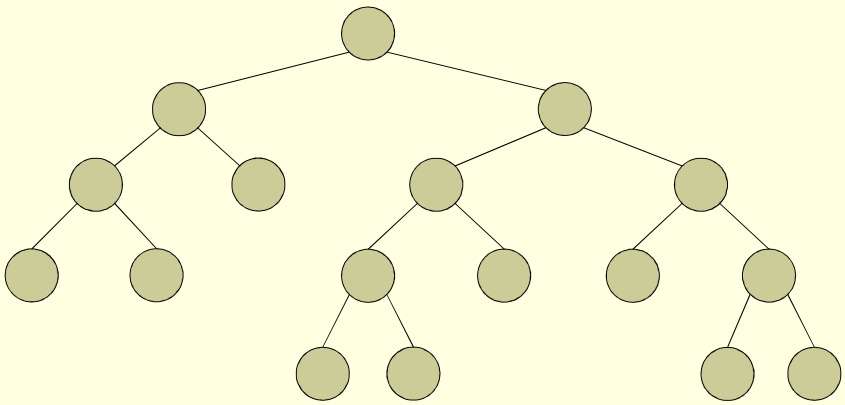
AVL: quem é e quem não é?



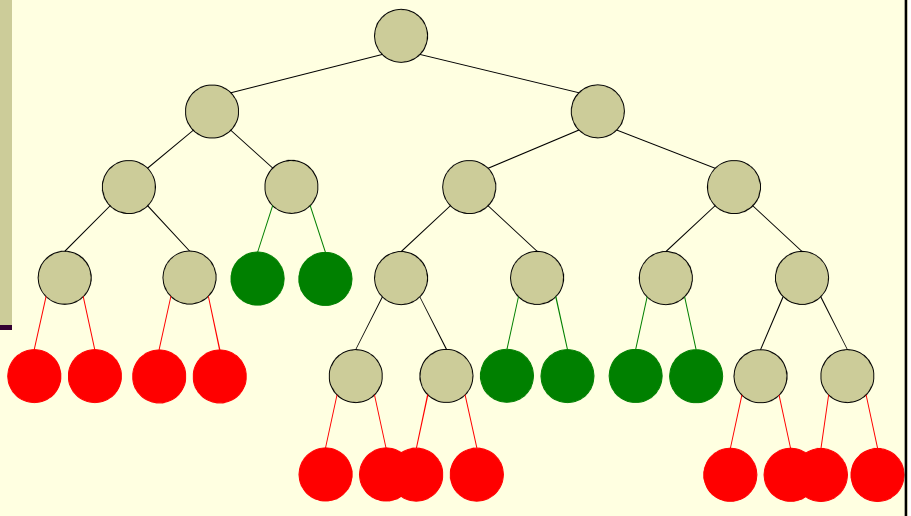
AVL: quem é e quem não é?



Pergunta: a árvore abaixo é AVL?



Exercício: onde se pode incluir um nó para a AVL continuar sendo AVL?



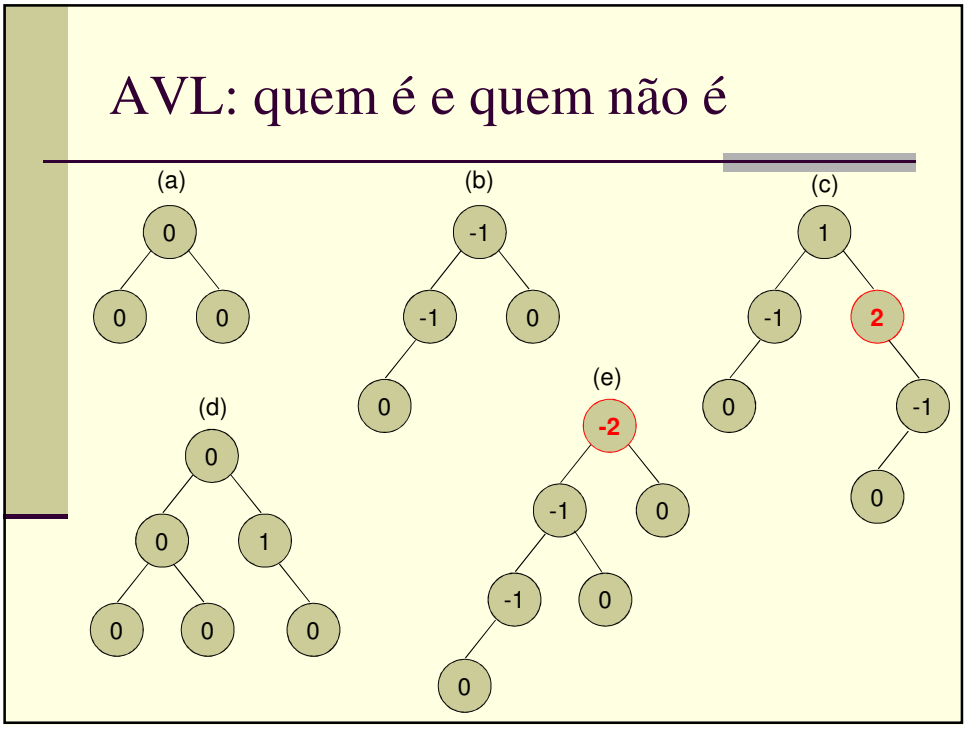
AVL

- Como é que se sabe quando é necessário balancear a árvore?
  - Se a diferença de altura das subárvores deve ser 1, no máximo, então temos que procurar diferenças de altura maior do que isso
  - Possível solução: cada nó pode manter a diferença de altura de suas subárvores
    - Convencionalmente chamada de fator de balanceamento do nó

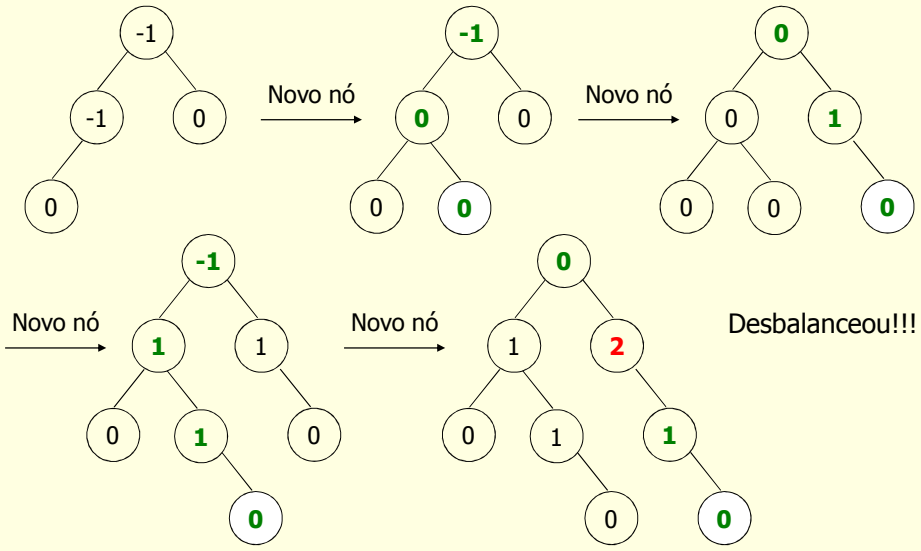
AVL

- Fatores de balanceamento dos nós
  - Altura da subárvore direita menos altura da subárvore esquerda
    - Hd-He
  - Atualizados sempre que a árvore é alterada (elemento é inserido ou removido)
  - Quando um fator é 0, 1 ou -1, a árvore está balanceada
  - Quando um fator se torna 2 ou -2, a árvore está desbalanceada
    - Operações de balanceamento!

AVL: quem é e quem não é



AVL: exemplo de desbalanceamento

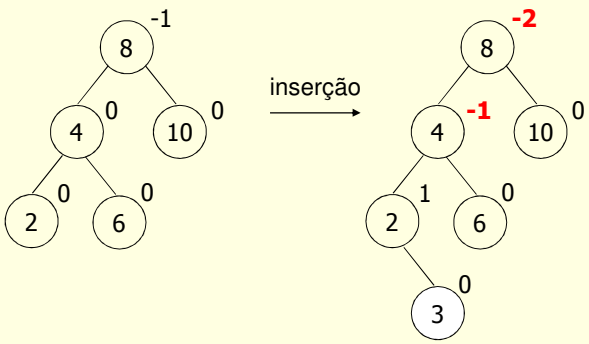


AVL

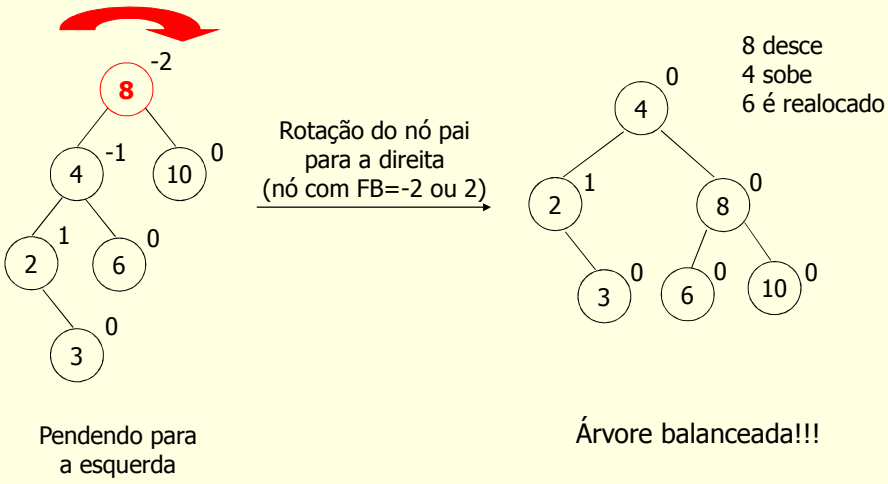
- Controle do balanceamento
  - Altera-se o algoritmo de inserção para balancear a árvore quando ela se tornar desbalanceada após uma inserção (nó com FB 2 ou -2)
    - Rotações
      - Se árvore pende para esquerda (FB negativo), rotaciona-se para a direita
      - Se árvore pende para direita (FB positivo), rotaciona-se para a esquerda
    - 2 casos podem acontecer

AVL: primeiro caso

- Raiz de uma subárvore com FB -2 (ou 2) e um nó filho com FB -1 (ou 1)
  - Os fatores de balanceamento têm sinais iguais: subárvores de nó raiz e filho pendem para o mesmo lado



AVL: primeiro caso



Pendendo para a esquerda

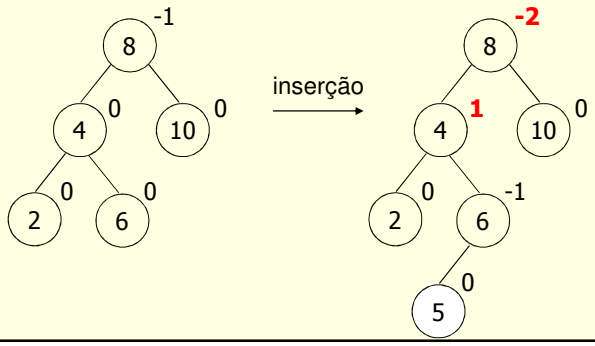
Árvore balanceada!!!

AVL: primeiro caso

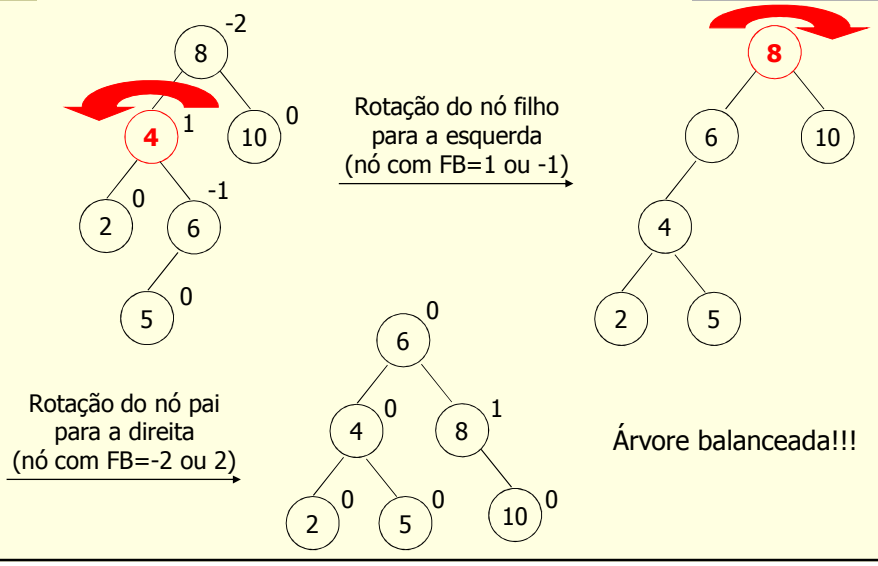
- Quando subárvores do pai e filho pendem para um mesmo lado
  - Rotação simples para o lado oposto
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

AVL: segundo caso

- Raiz de uma subárvore com FB -2 (ou 2) e um nó filho com FB 1 (ou -1)
  - Os fatores de balanceamento têm sinais opostos: subárvore de nó raiz pende para um lado e subárvore de nó filho pende para o outro (ou o contrário)



AVL: segundo caso



AVL: segundo caso

- Quando subárvores do pai e filho pendem para lados opostos
  - Rotação dupla
    - Primeiro, rotaciona-se o filho para o lado do desbalanceamento do pai
    - Em seguida, rotaciona-se o pai para o lado oposto do desbalanceamento
  - Às vezes, é necessário realocar algum elemento, pois ele perde seu lugar na árvore

AVL

- As transformações dos casos anteriores diminuem em 1 a altura da subárvore com raiz desbalanceada p
- Assegura-se o rebalanceamento de todos os ancestrais de p e, portanto, o rebalanceamento da árvore toda

AVL

- Novo algoritmo de inserção
  - A cada inserção, verifica-se o balanceamento da árvore
    - Se necessário, fazem-se as rotações de acordo com o caso (sinais iguais ou não)
  - Em geral, armazena-se uma variável de balanceamento em cada nó para indicar o FB

AVL

- Declaração

```
typedef struct no {
        int info;
        struct no *esq, *dir;
        int FB;
} no;

no *raiz;
```

AVL

- Exercício
  - Inserir os elementos 10, 3, 2, 5, 7 e 6 em uma árvore e balancear quando necessário

AVL

- Exercício
  - Inserir os elementos A, B, C, ..., J em uma árvore e balancear quando necessário