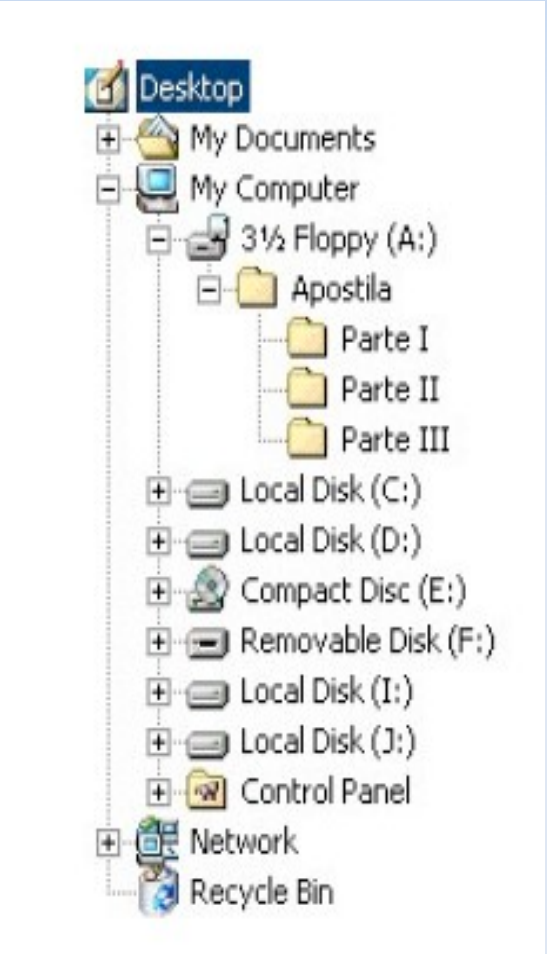
Estruturas de Dados

Árvores

Prof. Eduardo Alchieri

Árvores
(introdução)

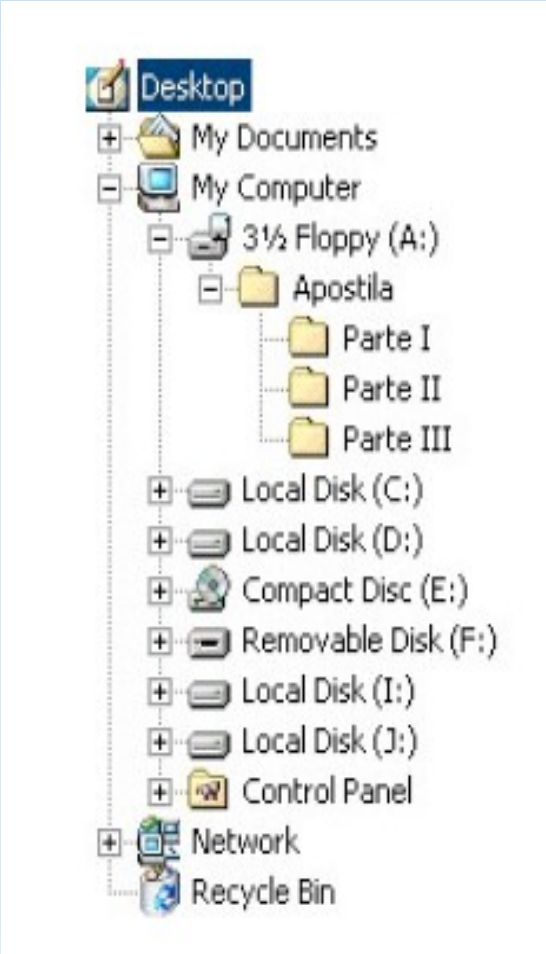
- Importância de estruturas unidimensionais ou lineares (vetores e listas) é inegável
- Porém, estas estruturas não são adequadas para representar dados que devem ser dispostos de maneira **hierárquica**
  - Por exemplo, **diretórios** criados em um computador



Um exemplo de estrutura de diretório no Windows 2000

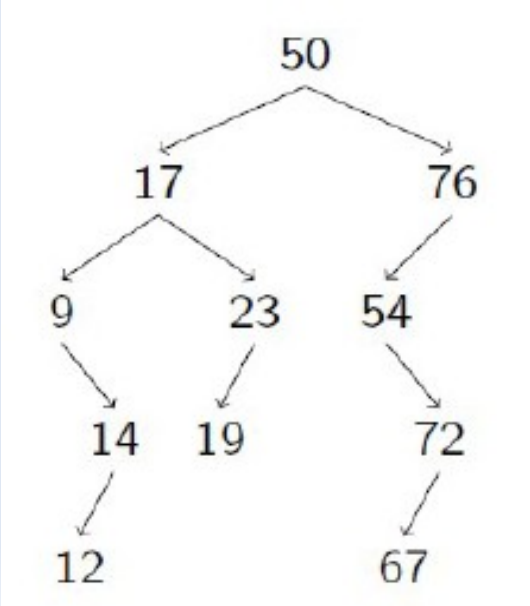
Árvores
(introdução)

- **Árvore** é uma estrutura de dado não linear adequada para representar **hierarquias**



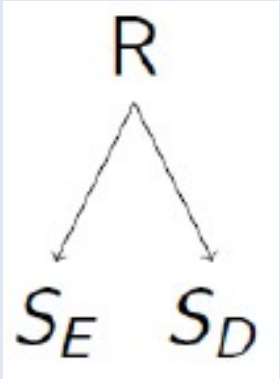
Árvores
(definição)

- Árvores
  - Dados são dispostos de forma hierárquica
  - Elementos (nós)
    - Raiz (pai) - [ancestrais]
    - Galhos (filhos) – [ancestrais/descendentes]
    - Folhas (terminais) - [descendentes]



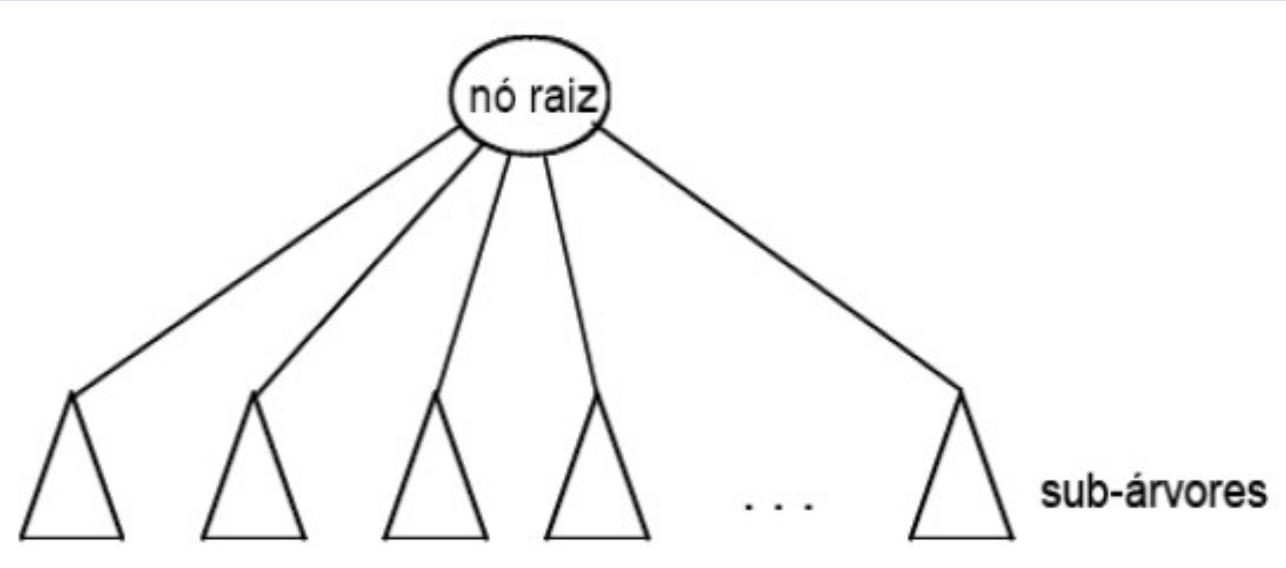
Árvores
(definição)

- Forma mais natural de definirmos uma estrutura de árvore é usando **recursividade**
  - Definição recursiva de árvores
    - Uma árvore é uma coleção de nós
    - A coleção pode estar vazia, ou consistir de um nó raiz R
    - Existe um arco direcionado de R para a raiz de cada subárvore: a raiz de cada subárvore é chamada de filho de R, da mesma forma R é chamado de pai da raiz de cada subárvore



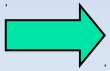
Árvores
(definição)

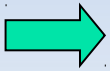
- Definição recursiva de árvores (outra forma de representar uma árvore)



Árvores
(exemplos)

- Exemplo de árvore





Quantas subárvores existem na árvore acima?

Quais são as subárvores?

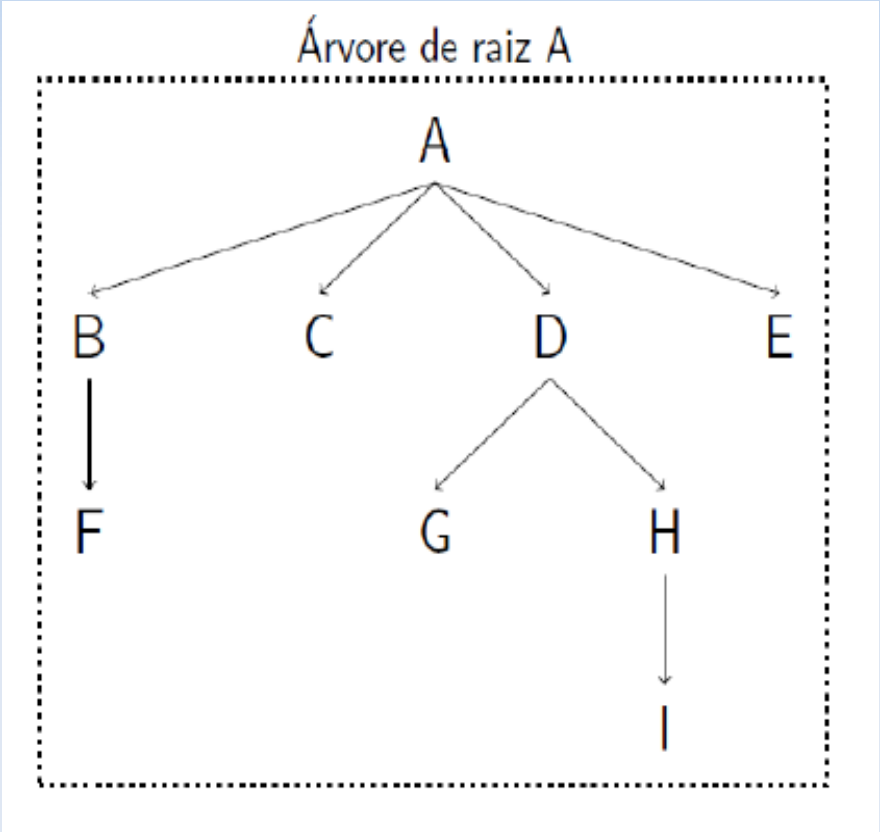
Quais nós são as raízes das subárvores da árvore acima?

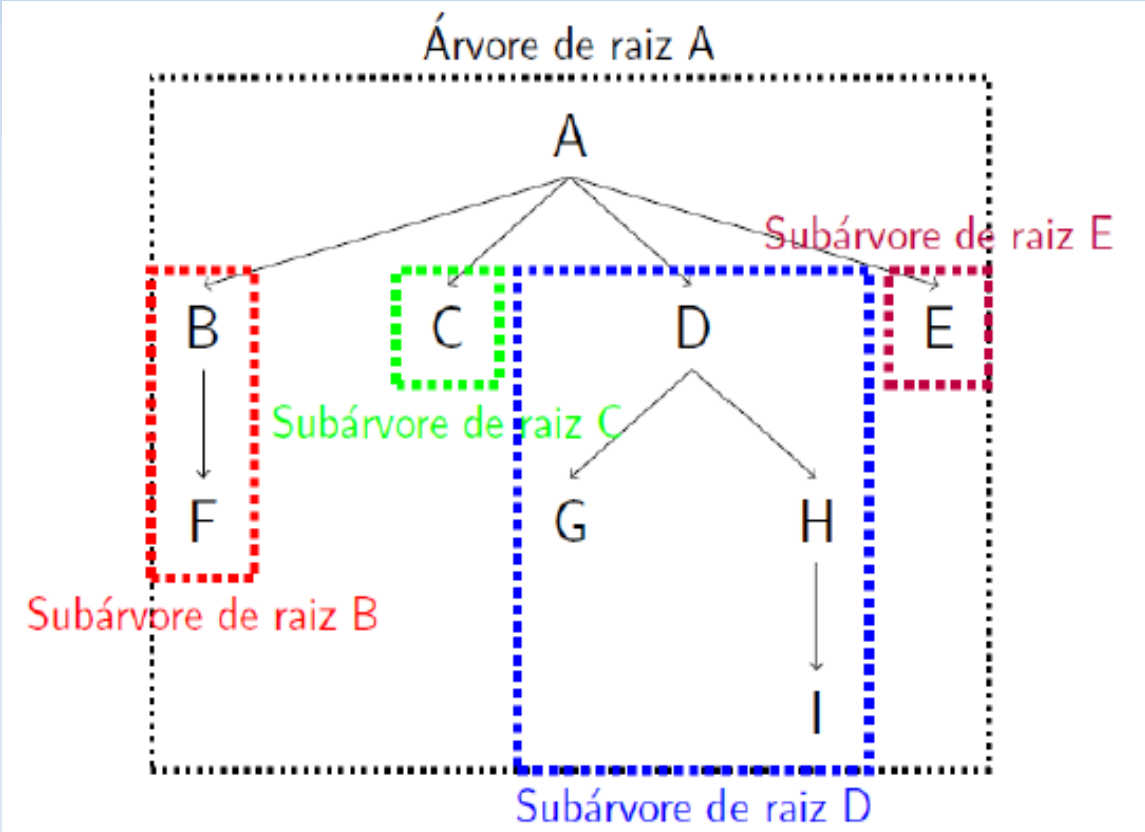
Quais nós são considerados nós internos?

Quais nós são considerados nós externos (folhas)?

Árvores
(subárvores)

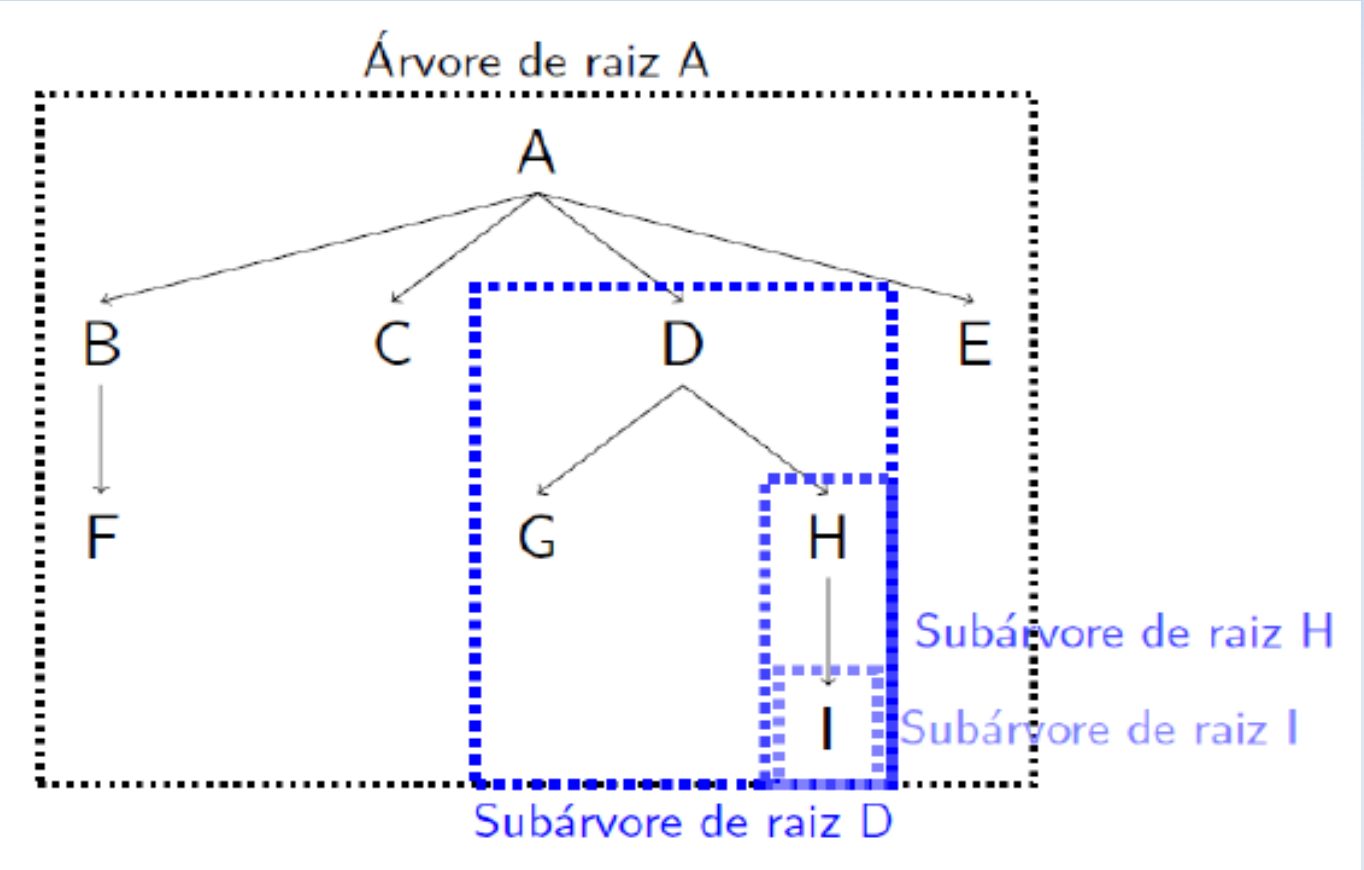
- Subárvores (visualização da definição recursiva)





Árvores
(subárvores)

- Subárvores (visualização da definição recursiva)

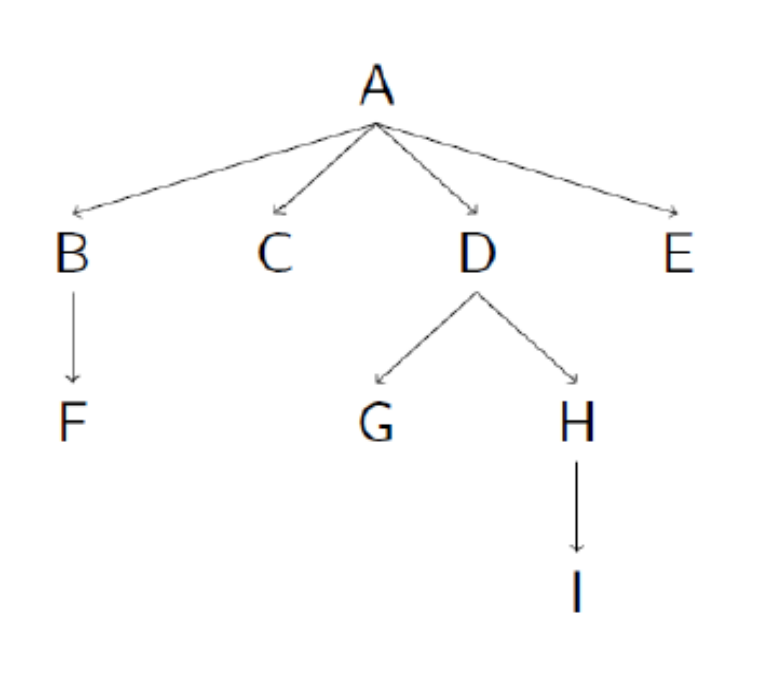


Árvores
(terminologia)

- Terminologia
  - Grau de um nó: número de subárvores relacionadas com o nó
  - Folha: um nó de grau zero
  - Ordem: número máximo de galhos em um elemento
  - Caminho: sequência única de arcos que leva a um nó a partir da raiz
  - Comprimento do Caminho: número de arcos no caminho
  - Nível de um nó: o comprimento do caminho da raiz até o nó, que é o número de arcos no caminho
  - Altura: raiz mais o máximo número de descendentes
    - Caminho entre a raiz e a(s) folhas(s) mais distante(s) + 1

Árvores
(exemplo)

- Exemplo



Nível 0

Nível 1

Nível 2

Nível 3

Ordem: 4

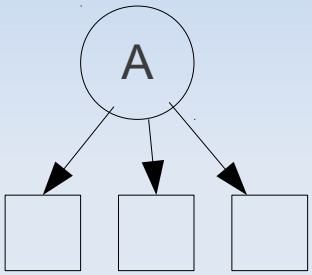
Altura: 4

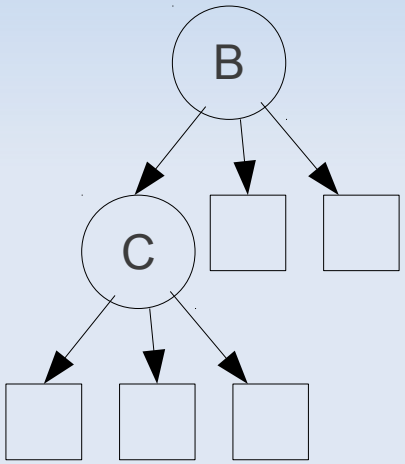
Árvores N-árias
(definição)

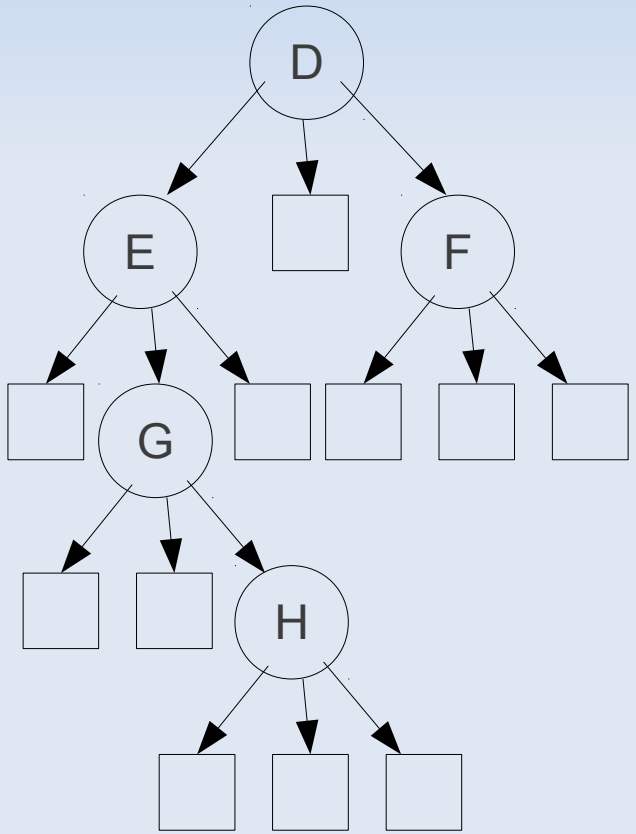
- Todos os nós da árvore possuem o mesmo grau (N), i.e., mesmo número de filhos
- Definição
  - Uma árvore N-ária T é um conjunto finito de nós com as seguintes propriedades:
    - O conjunto é vazio; ou
    - O conjunto consiste de uma raiz R, e exatamente N árvores N-árias distintas, que são subárvores de R

Árvores N-árias
(exemplos)

- Exemplos de árvores N-árias (N=3)



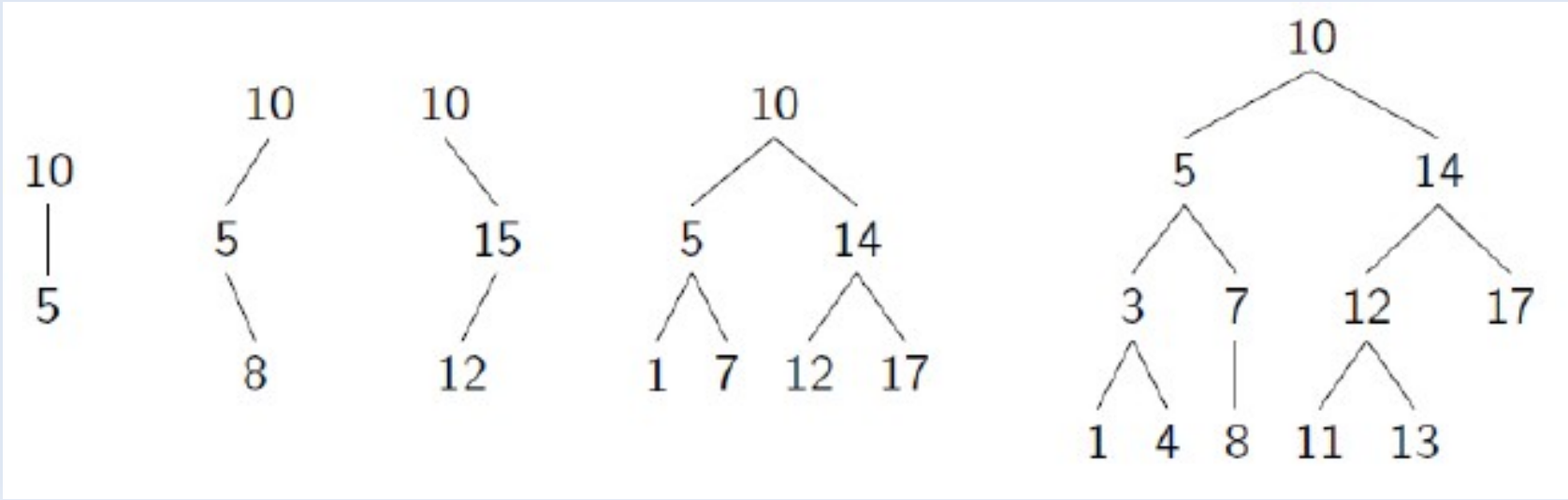




Uma árvore N-ária com n ≥ 0 nós internos contém (N – 1)n + 1 nós externos

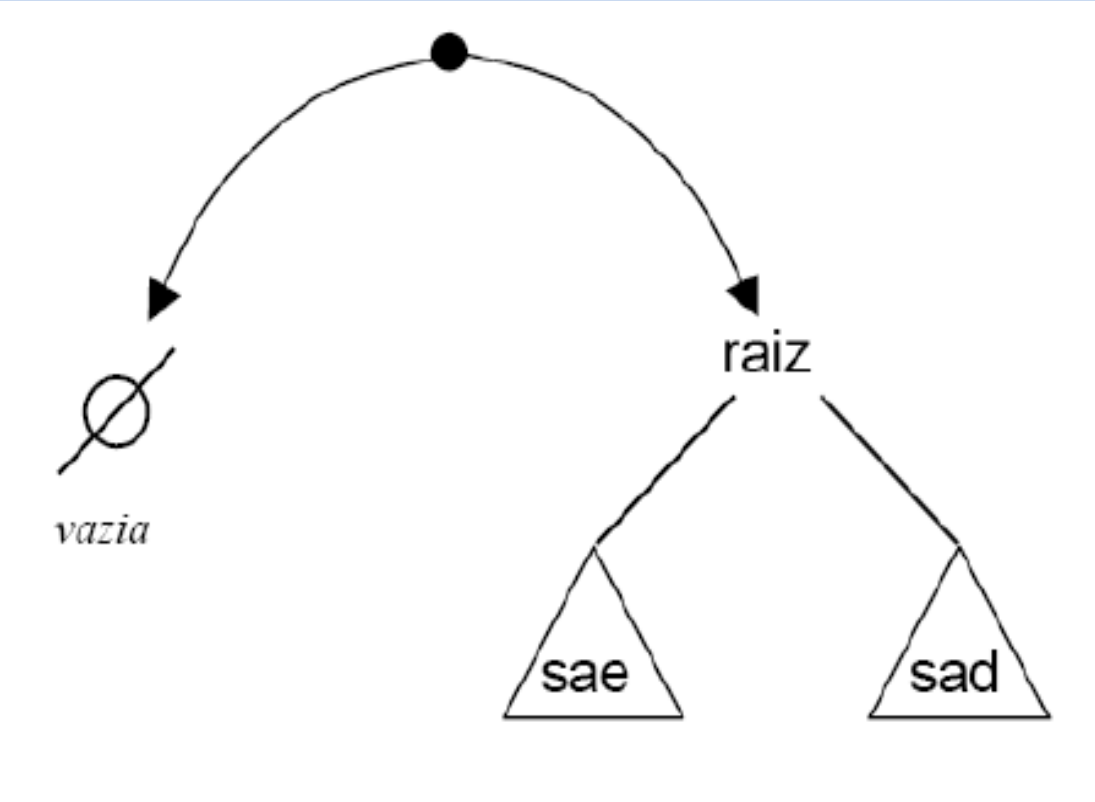
Árvores Binárias
(definição)

- Definição
  - Conjunto finito de elementos que está vazio ou pode ser particionado em três subconjunto disjuntos:
    - Raiz, um subconjunto que possui um único elemento
    - Subárvore esquerda, que é uma árvore binária
    - Subárvore direita, que também é uma árvore binária



Árvores Binárias
(definição)

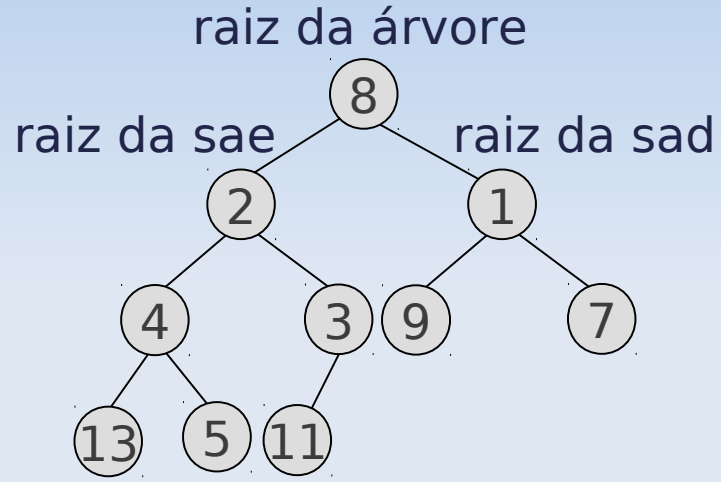
- Representação esquemática da estrutura de árvores binárias



Árvores Binárias

(exemplo)

- Exemplo:



Percursos em Árvores
(percursos)

- Existem muitas aplicações de árvores
  - Existem muitos algoritmos diferentes para manipulá-las
  - No entanto, estes algoritmos têm a característica comum de visitar sistematicamente todos os nós da árvore
    - O algoritmo caminha através da estrutura de dados e faz algumas computações em cada nó da árvore
- Dois métodos essencialmente diferentes
  - Percurso em profundidade
    - Percurso pré-ordem (a raiz é visitada antes)
    - Percurso em ordem (árvores binárias – a raiz é visitada na ordem, entre as subárvores)
    - Percurso em pós-ordem (a raiz é visitada depois)
  - Percurso em largura

Percursos em Árvores
(percursos)

- Percurso pré-ordem (a raiz é visitada antes)
  - Visite/processe a raiz, e depois
  - Realize um percurso em pré-ordem em cada uma das subárvores da raiz na ordem definida
  - Algoritmo

```
pre_ordem(No raiz){
        Se (raiz é null) então retorne;
        processa(raiz);
        para cada subárvore sa da raiz, faça:
                pre_ordem(sa);
```

Percursos em Árvores
(percursos)

- Percurso em-ordem (a raiz é visitada na ordem, entre as subárvores – só faz sentido para Árvores Binárias)
  - Percorra em ordem a subárvore da esquerda
  - Visite/processe a raiz;
  - Percorra em ordem a subárvore da direita
  - Algoritmo

```
em_ordem(No raiz){
        Se (raiz é null) então retorne;
        em_ordem(raiz.esquerda);
        processa(raiz);
        em_ordem(raiz.direita);
```

Percursos em Árvores
(percursos)

- Percurso Pós-ordem (a raiz é visitada depois)
  - Realize um percurso em pós-ordem em cada uma das subárvores da raiz na ordem definida
  - Visite/processe a raiz;
  - Algoritmo

```
pos_ordem(No raiz){
        Se (raiz é null) então retorne;
        para cada subárvore sa da raiz, faça:
                pos_ordem(sa);
        processa(raiz);
```

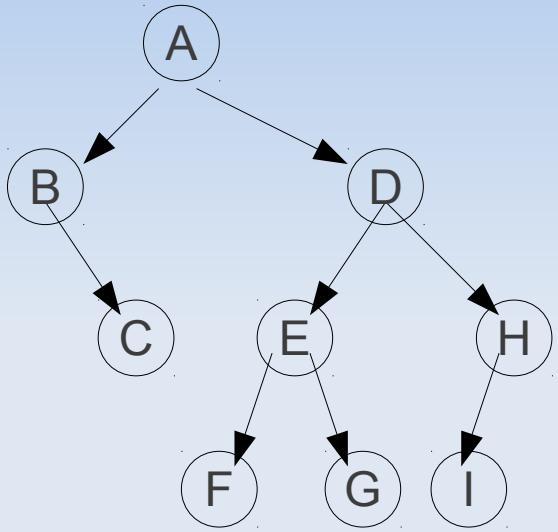
Percursos em Árvores
(percursos)

- Percurso em largura
  - Também chamado de percurso em ordem de nível
  - Utiliza uma fila para armazenar a ordem em que os nós devem ser acessados
  - Algoritmo

```
largura(No raiz){
        f.enfileire(raiz)
        Enquanto f não estiver vazia, faça:
                no = f.desenfileire();
                processa (no);
                para cada filho fi de no (da esquerda para a
                                                direita), faça:
               f.enfileire(fi);
```

Percursos em Árvores
(percursos)

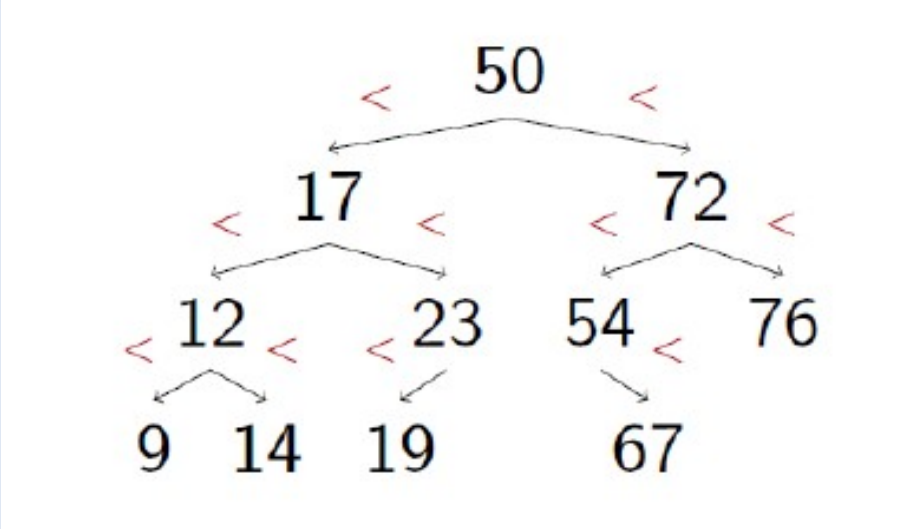
- Exemplo



- Pré-ordem: A,B,C,D,E,F,G,H,I
- Ordem: B,C,A,F,E,G,D,I,H
- Pós-ordem: C,B,F,G,E,I,H,D,A
- Largura: A,B,D,C,E,H,F,G,I

Árvores Binárias de Busca
(definição)

- Árvore Binária de Busca
  - É uma árvore binária ordenada
  - A árvore binária de busca tem os filhos ordenados segundo um certo critério



- Árvore Binária (de Busca) vs Árvores
  - Os nós de uma árvore binária não podem ter mais de dois filhos, enquanto não há limites para o número de filhos de uma árvore

Árvores Binárias de Busca
(operações)

- Operações
  - Criar
  - Esvaziar
  - Inserir
  - Remover
  - Buscar
  - Etc.

Árvores Binárias de Busca
(percursos)

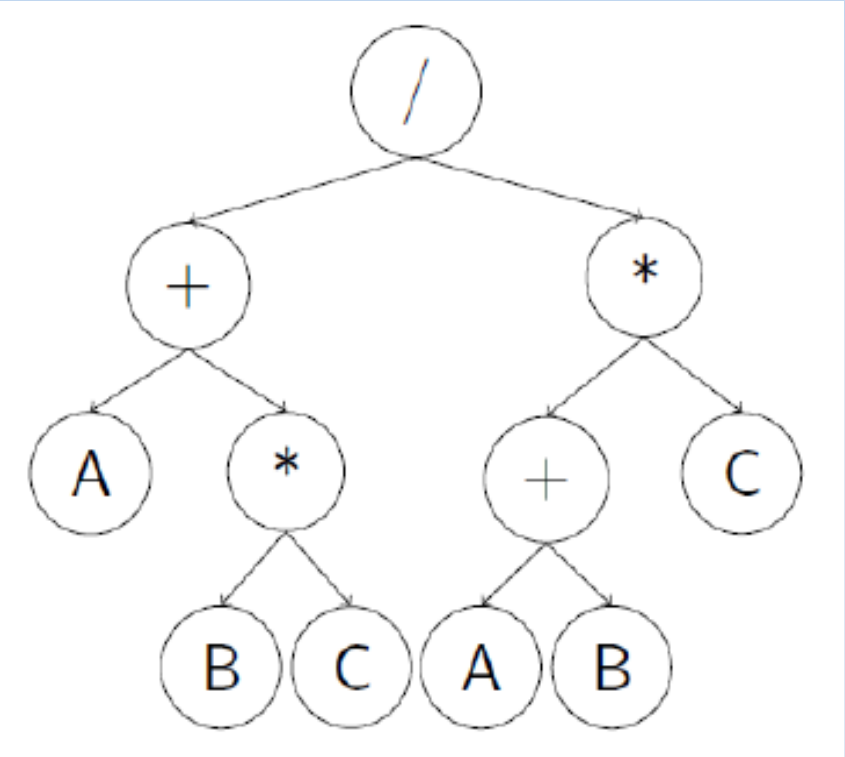
- Percurso pré-ordem
  - Visite/percorra a raiz, e depois
  - Percorra em pré-ordem a subárvore da esquerda, e depois
  - Percorra em pré-ordem a subárvore da direita
- Percurso em ordem
  - Percorra em ordem a subárvore da esquerda, e depois
  - Visite/percorra a raiz, e depois
  - Percorra em ordem a subárvore da direita
- Percurso pós-ordem
  - Percorra em pós-ordem a subárvore da esquerda, e depois
  - Percorra em pós-ordem a subárvore da direita, e depois
  - Visite/percorra a raiz, e depois

Árvores Binárias de Busca
(percursos)

- Percurso em largura
  - Enfileire o nó raiz
  - Enquanto a fila não estiver vazia
    - Desenfileire o nó n
    - Visite/ processa n
    - Enfileire o filho da esquerda de n
    - Enfileire o filho da direita de n

Árvores Binárias de Busca
(árvores de expressões)

- Expressões contendo operadores binários possuem inerentemente uma estrutura de árvore



- Pós-ordem: A B C * + A B + C * / (forma pós-fixada)
- Pré-ordem: / + A * B C * + A B C (forma pré-fixada)
- Em ordem: A + B * C / A + B * C (forma infixada)

Árvores Binárias de Busca
(busca)

- Os elementos da árvore binária estão ordenados, então a busca na árvore faz uso de um algoritmo simples
  - Compare o elemento com a raiz
    - Se for igual, pare a busca
    - Se for menor, busque na subárvore da esquerda
    - Se for maior, busque na subárvore da direita

- Custo médio de uma busca binária: O(log n)

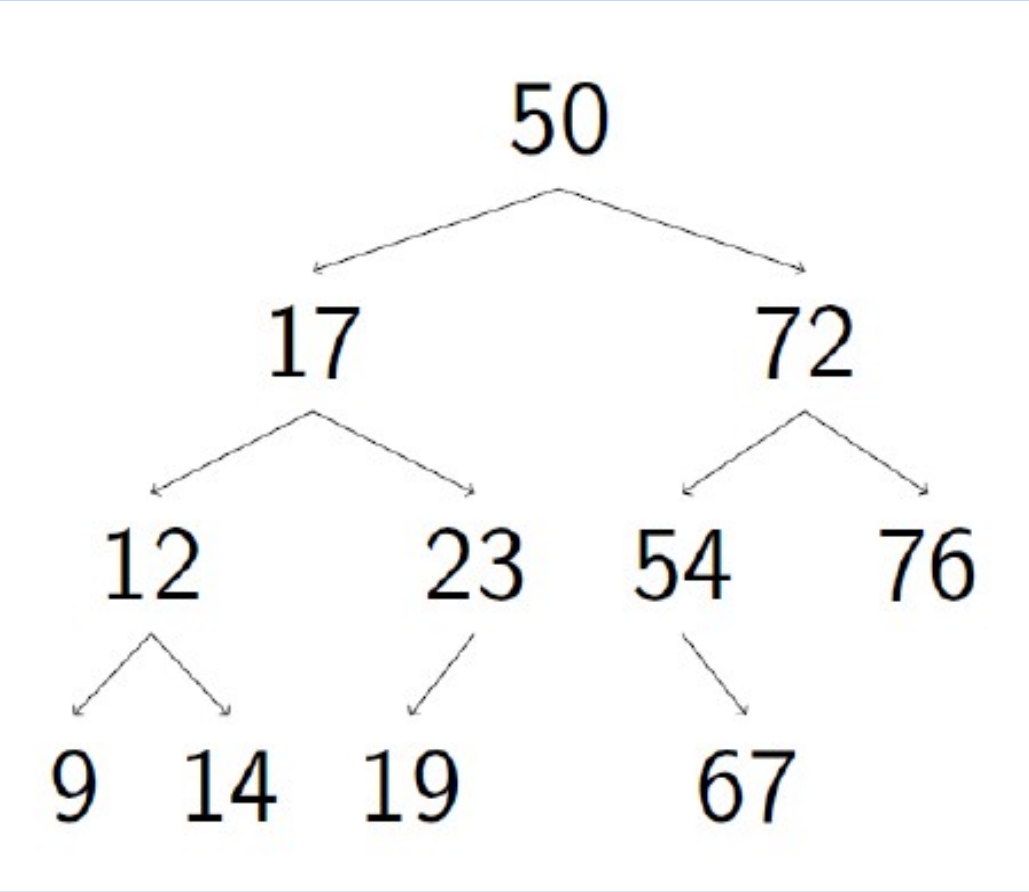
Árvores Binárias de Busca
(busca)

- Algoritmo:

```
busca(raiz,dado)
        if (raiz == null)
                return null;
        if(raiz.dado > dado)
                return busca(raiz.esquerda,dado);
         if(raiz.dado < dado)
                return busca(raiz.direita,dado);
         return raiz;
```

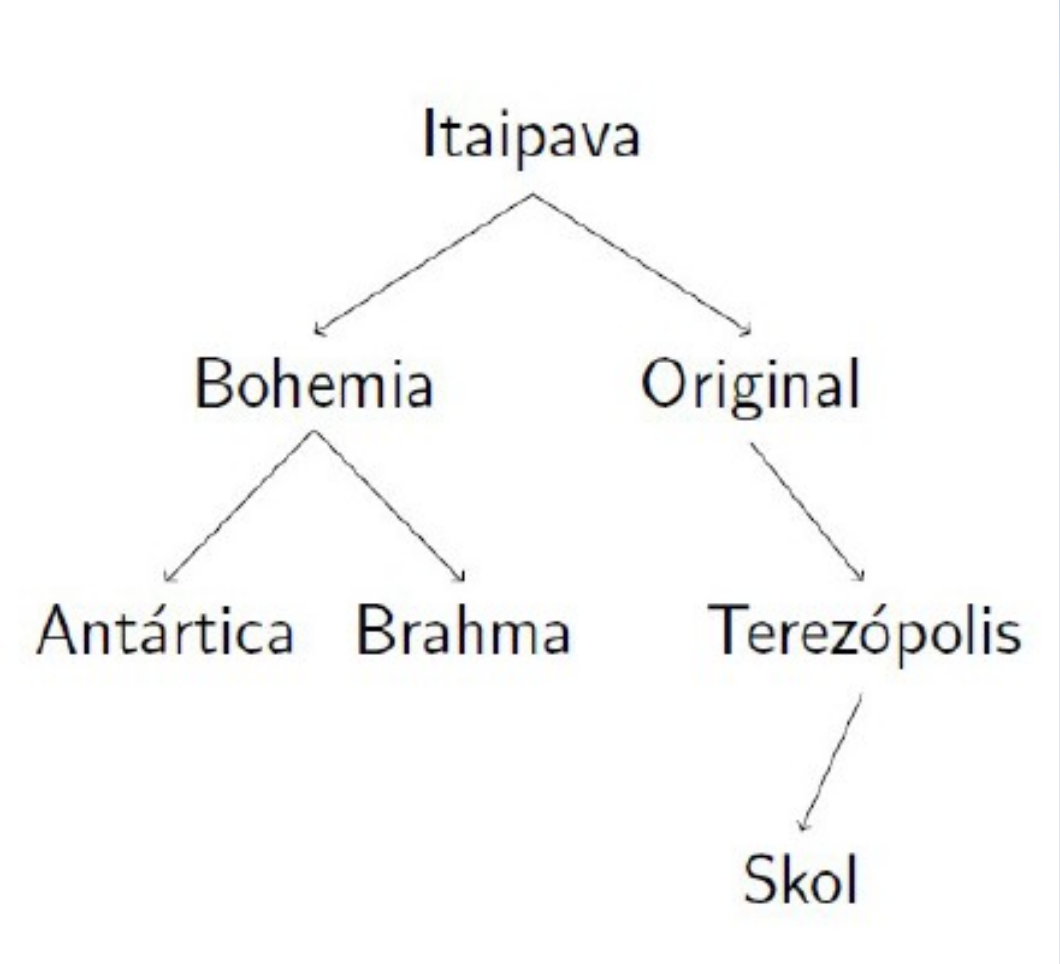
Árvores Binárias de Busca
(busca)

- Procurando um número



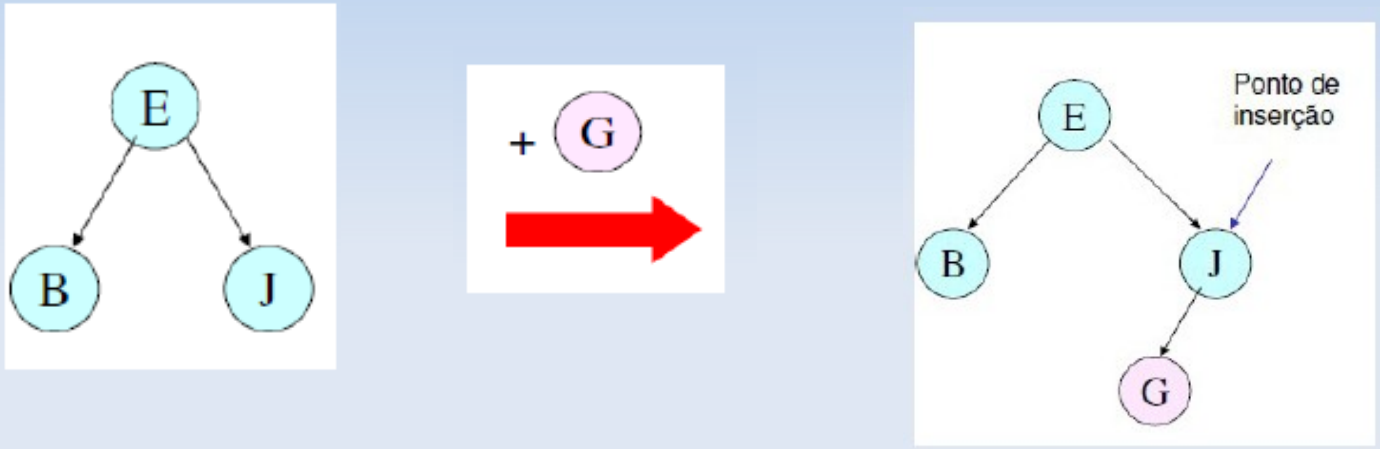
Árvores Binárias de Busca
(busca)

- Procurando uma palavra



Árvores Binárias de Busca
(inserção)

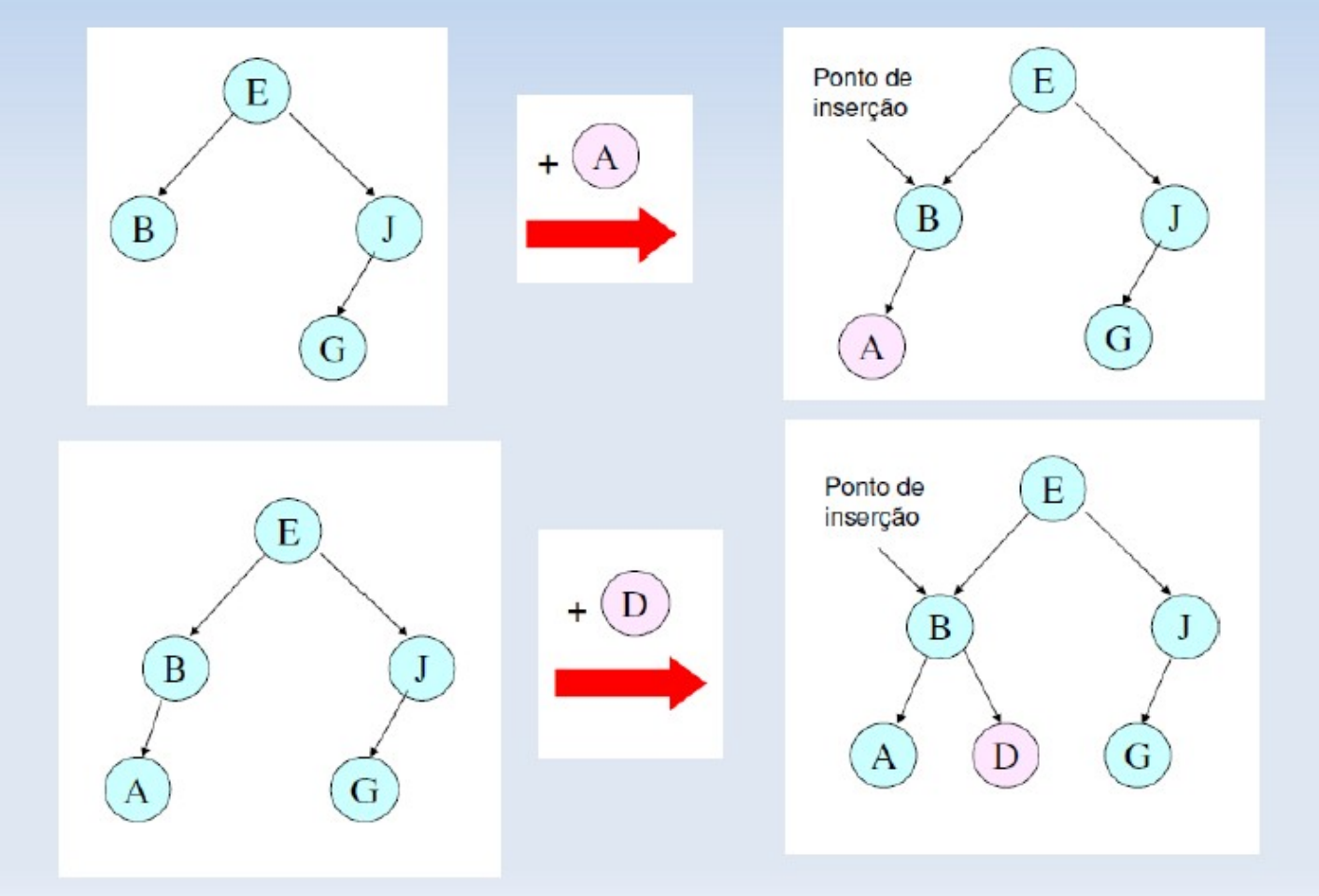
- Inserção de um novo nó



- A localização do ponto de ”inserção” é semelhante à busca por um valor na árvore
- Após a inserção do novo elemento, a árvore deve manter as propriedades de árvore binária de busca
- O nó inserido é sempre uma folha

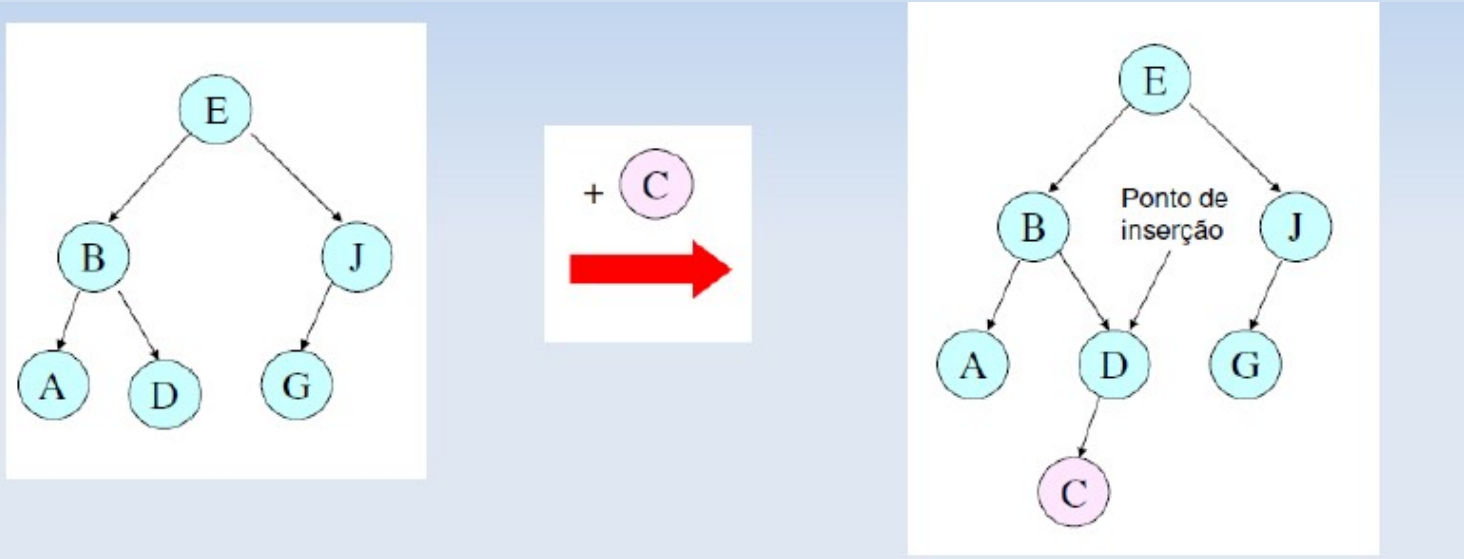
Árvores Binárias de Busca
(inserção)

- Inserção de um novo nó



Árvores Binárias de Busca
(inserção)

- Inserção de um novo nó



Árvores Binárias de Busca
(inserção)

- Algoritmo para inserir um nó

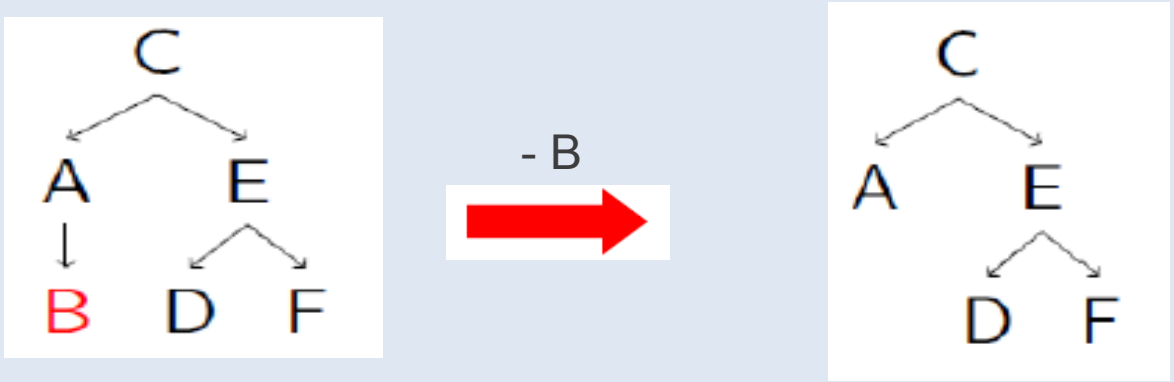
```
inserir_na_AB (raiz, dado)
    if(raiz.dado > dado){
        if(raiz.esquerda == null){
            raiz.esquerda = novo elemento com dado
        }else{
            inserir_na_AB(raiz.esquerda,dado);
        }
    else{
        if(raiz.direita == null){
            raiz.direita = novo elemento com dado
        }else{
            inserir_na_AB(raiz.direita,dado);
        }
    }
```

Árvores Binárias de Busca
(remoção)

- Remoção de um nó
- Existem três situações na remoção de um nó
  - Nó com 0 filhos
  - Nó com 1 filho
  - Nó com 2 filhos
    - Fusão
    - Cópia

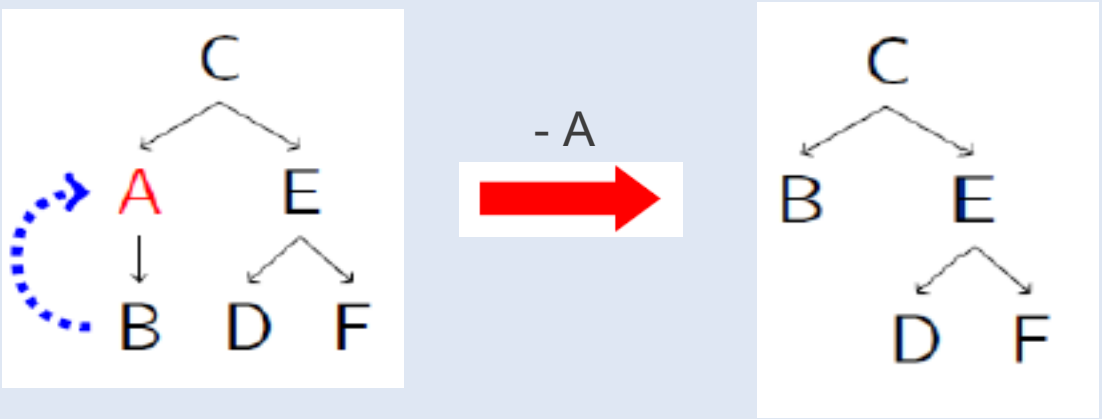
Árvores Binárias de Busca
(remoção)

- Remoção de um nó com 0 filhos (folha)
  - O nó simplesmente é retirado, e seu pai recebe nulo no lugar do ponteiro para aquele filho



Árvores Binárias de Busca
(remoção)

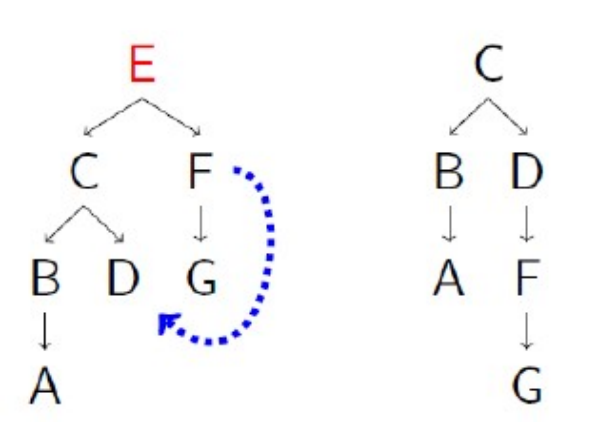
- Remoção de um nó com 1 filho
  - O nó é retirado e em seu lugar toda a subárvore cuja raiz é seu filho toma o lugar
    - O pai do nó a ser retirado aponta para o fillho do nó a ser retirado



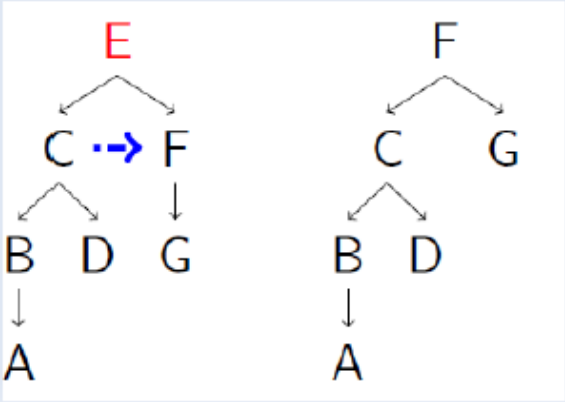
Árvores Binárias de Busca
(remoção)

- Remoção de um nó com 2 filhos (Fusão)
  - Extrai uma árvore das duas subárvores do nó a ser eliminado: essa árvore vai substituir o nó e seus descendentes
    1) O maior nó da subárvore esquerda passa a ser a raiz da subárvore direita; **ou**
    2) O menor nó da subárvore direita passa a ser a raiz da subárvore esquerda

Solução 1)

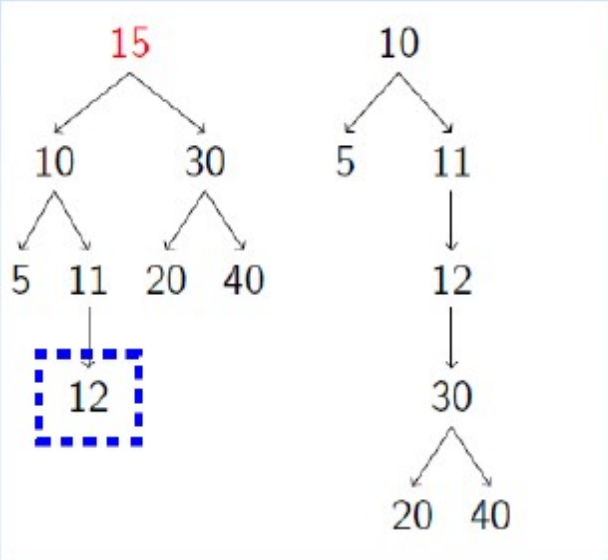


Solução 2)

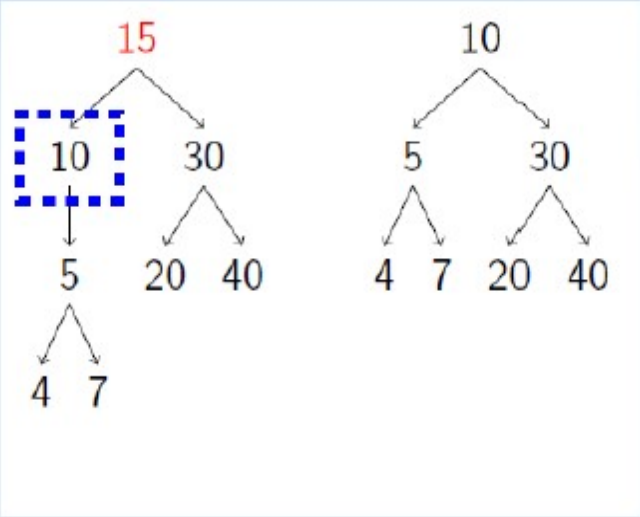


Árvores Binárias de Busca
(remoção)

- (Des)vantagens da remoção por fusão



A altura aumenta.



A altura diminui.

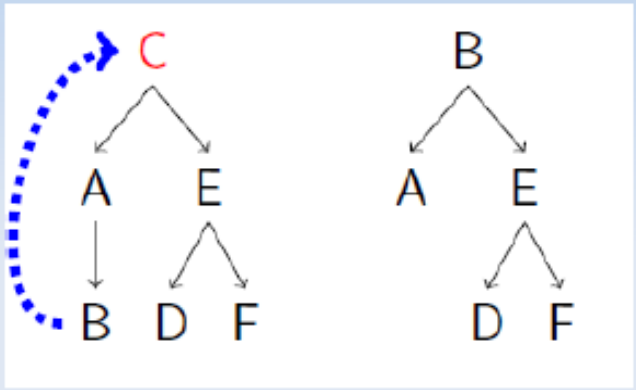
Árvores Binárias de Busca
(remoção)

- Remoção de um nó com 2 filhos (Cópia)
  - O nó não é retirado, mas tem seu conteúdo alterado
    - É substituído pelo elemento **antecessor ou sucessor**
      1) Para encontrar o nó antecessor, desce para a subárvore da esquerda do nó a ser retirado e caminhe até o final da subárvore da direita
      2) Para encontrar o nó sucessor, desce para a subárvore da direita do nó a ser removido e caminhe até o final da subárvore da esquerda
    - Após isso, o substituto é removido conforme o número de filhos (0 ou 1)

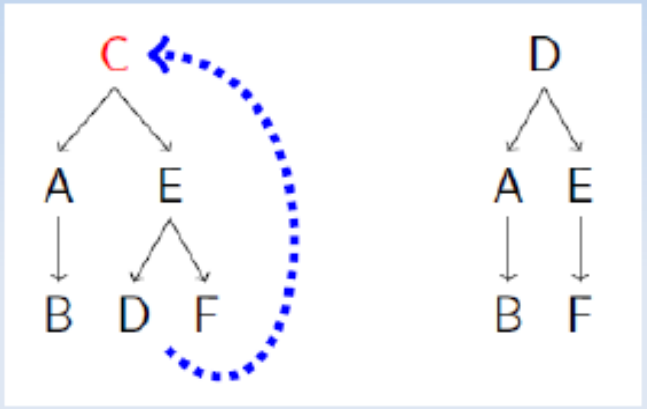
Árvores Binárias de Busca
(remoção)

- Remoção de um nó com 2 filhos (Cópia)

Solução 1)



Solução 2)



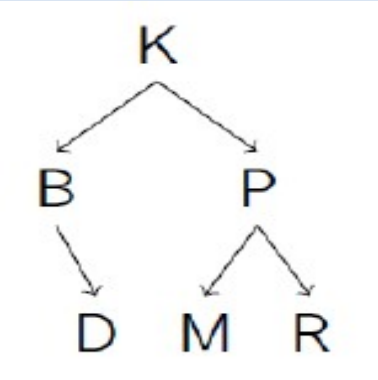
- Com a ocorrência de muitas adições e remoções em uma árvore de busca, a mesma terá um dos lados maior que o outro
  - Ficará desbalanceada, diminuindo sua eficiência

Árvores Binárias de Busca
(outras operações)

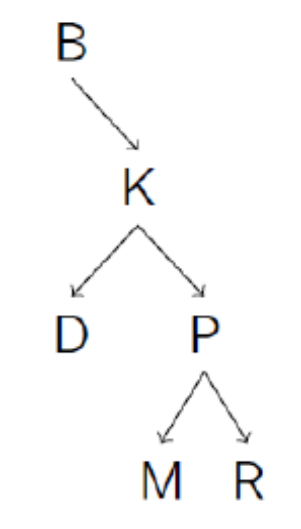
- Outras operações em árvores de busca:
  - Algoritmo para encontrar o maior elemento
  - Algoritmo para encontrar o menor elemento
  - Contar o número de elementos
  - Somar os valores dos elementos
  - Imprimir os elementos em ordem crescente
  - Imprimir os elementos em ordem decrescente
  - Etc...

Árvores Binárias de Busca
(eficiência)

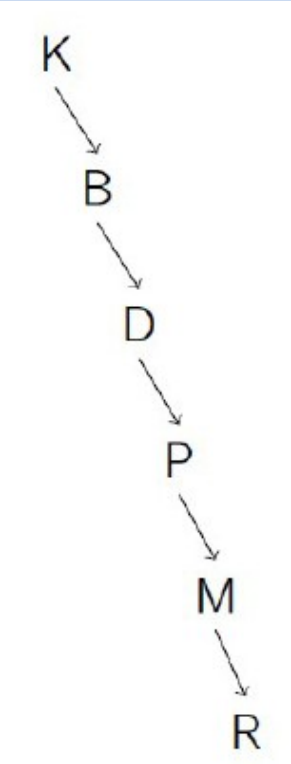
- A eficiência de uma árvore binária como estrutura de busca depende da disposição de seus elementos
- Qual o pior caso ?



3 comparações



4 comparações



árvore degenerada em lista! $\Leftarrow$ 6 comparações

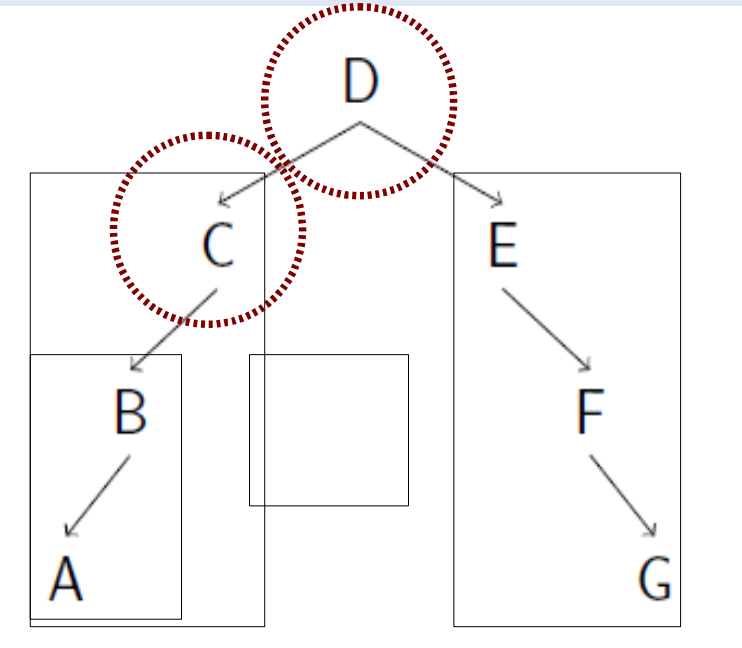
Árvores Binárias de Busca
(eficiência)

- Tempo médio de uma busca, inserção ou remoção: O(log n)
  - Já no pior caso: O(n)

- O problema com árvores de busca é que mesmo o tempo médio sendo O(log n), não sabemos sabemos nada sobre a forma da árvore
  - Solução: balanceamento

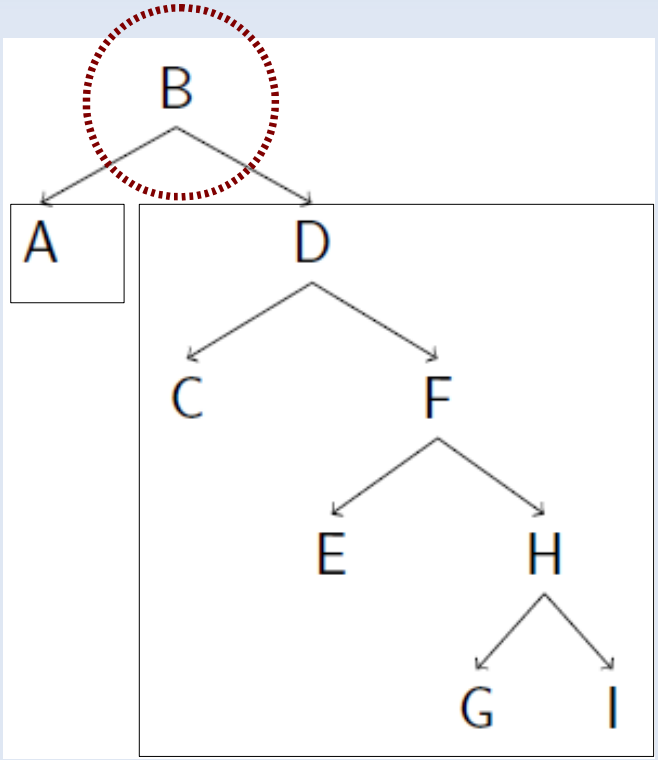
Árvores Binárias Balanceadas
(definição)

- Árvore binária balanceada: para cada nó, as alturas de suas subárvores diferem de, no máximo, 1.
  - É a árvore com a menor altura para o seu número de nós.



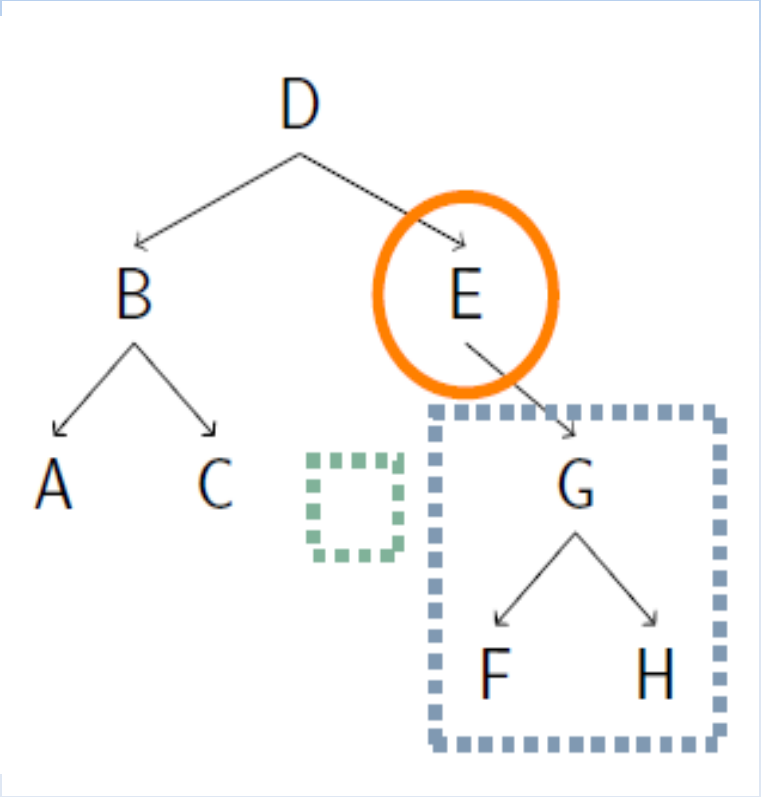
Árvores Binárias Balanceadas
(definição)

- Árvore binária balanceada: para cada nó, as alturas de suas subárvores diferem de, no máximo, 1.
  - É a árvore com a menor altura para o seu número de nós.



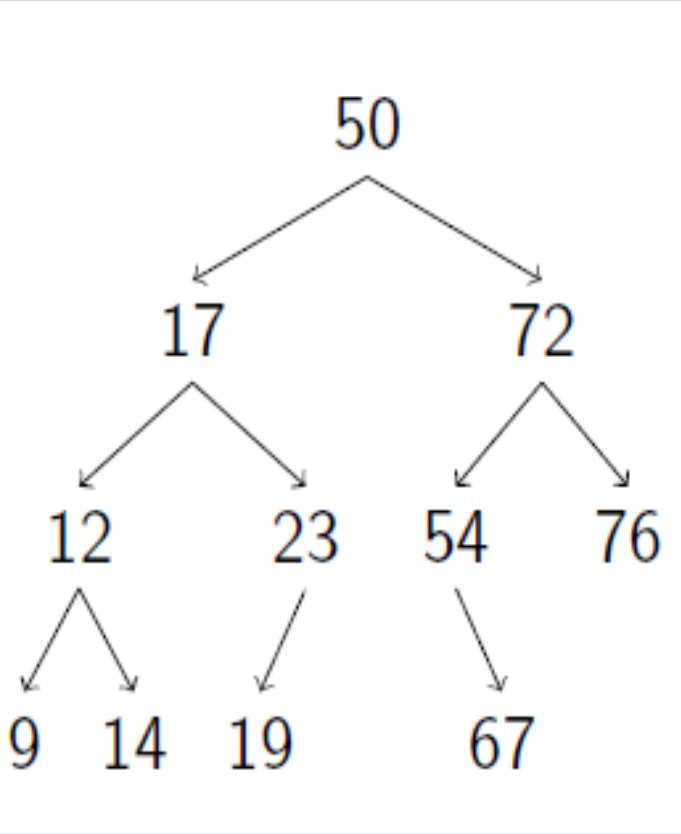
Árvores Binárias Balanceadas
(algoritmo)

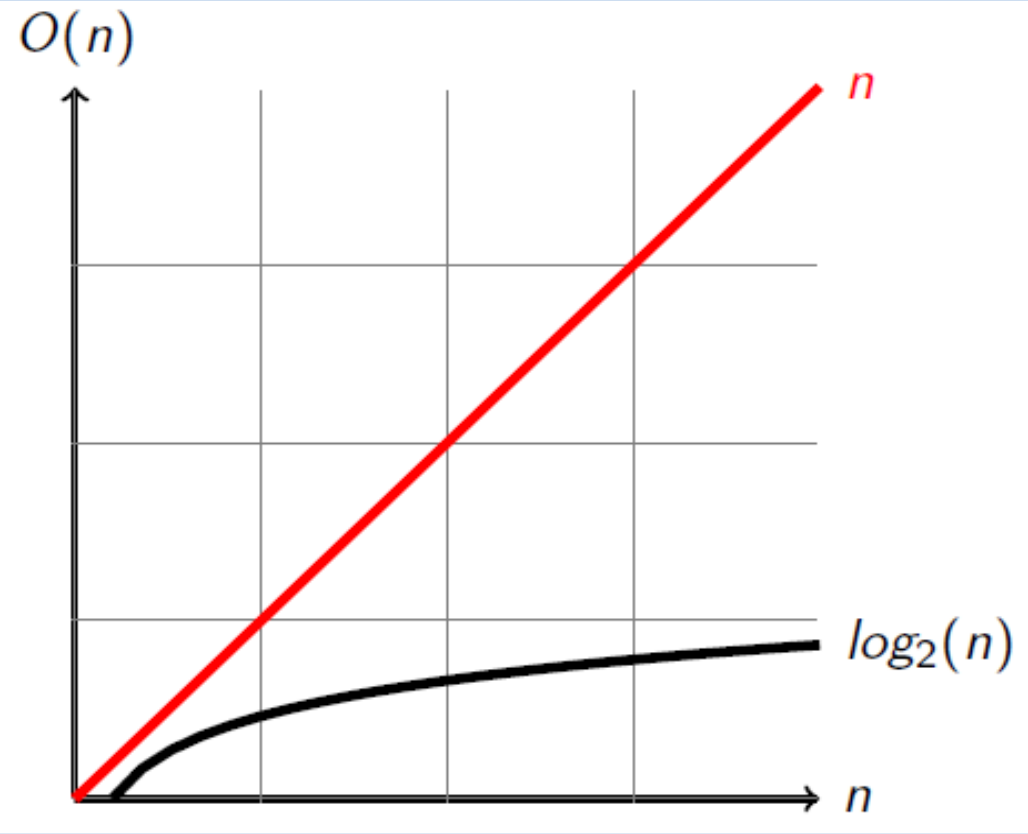
```
boolean balanceada (No raiz){
    if(raiz == null){ retorne verdadeiro; }
    if(!balanceada(raiz.esq)){ retorne falso; }
    if(!balanceada(raiz.dir)){ retorne falso; }
    if(abs(altura(raiz.esq) –
                    altura(raiz.dir)) > 1){
        retorne falso;
    }
    retorne verdadeiro;
}
```



Árvores Binárias Balanceadas
(custo)

- O objetivo desta árvore é estruturar os dados de forma que a pesquisa binária seja eficiente





custo de tempo para pesquisa: $O(log_2(n))$

Árvores Binárias Balanceadas
(custo)

- O custo da maioria das operações depende diretamente da altura da árvore, por isso o desejo de se ter a menor altura possível

Altura	Nós em um nível	Total de nós
1	1	1
2	2	3
3	4	7
4	8	15
11	1024	2047
14	8192	16383
h	2^{h-1}	$2^h - 1$

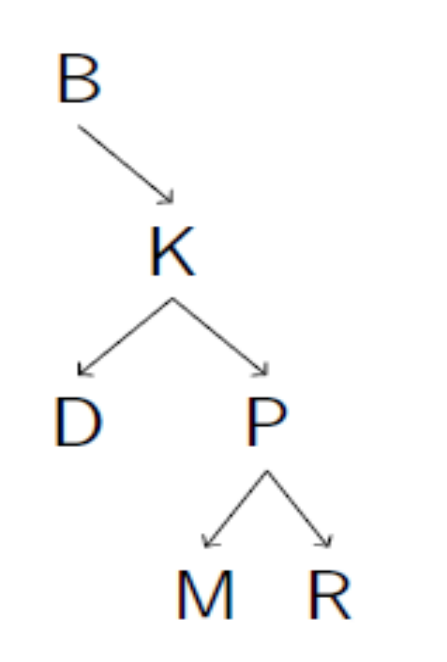
h é o número máximo de testes a serem feitos

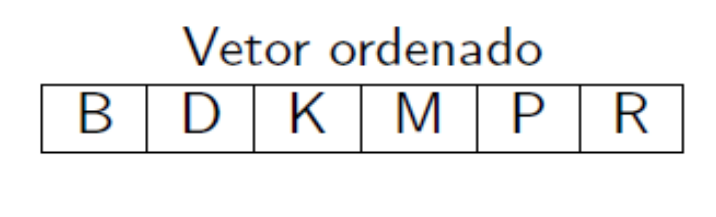
Árvores Binárias Balanceadas
(balanceamento)

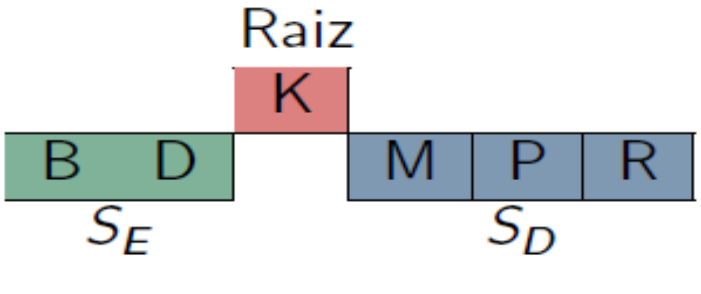
- Algoritmos para balanceamento
  - Estático: destruir a estrutura da árvore e contrui-lá balanceada
    - Vetor
    - DSW (Day/Stout/Warren)
  - Dinâmico: balanceamento junto as operações
    - AVL (Adelson-Velskii e E.M. Landis)
    - Rubro-negra

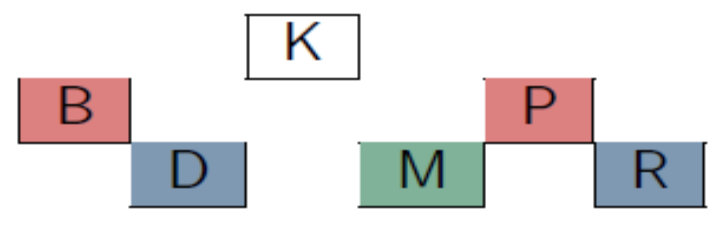
Árvores Binárias Balanceadas
(vetor)

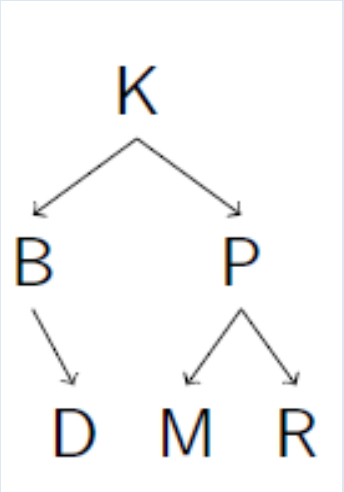
- Os dados da árvore são armazenados em um vetor (ou lista), ordenados, e outra árvore é construda a partir deste vetor











Árvores Binárias Balanceadas
(AVL)

- AVL (Adelson-Velskii e E.M. Landis)
- Se dissermos que uma árvore binária é balanceada se as subárvores esquerda e direita de cada nó tiverem a mesma altura, então as únicas árvores balanceadas serão as árvores binárias perfeitas.
- Condição de balançeamento AVL: Uma árvore binária vazia é balanceada AVL. Uma árvore não-vazia, T = {r, Tl, Tr}, é balanceada AVL se tanto Tl quanto Tr forem balanceadas AVL e |Hl – Hr| <= 1, onde Hl é a altura de Tl e Hr é a altura de Tr.
- Idéia básica: cada nó mantém uma informação adicional, chamada fator de balanceamento, que indica a diferença de altura entre as subárvores esquerda e direita.

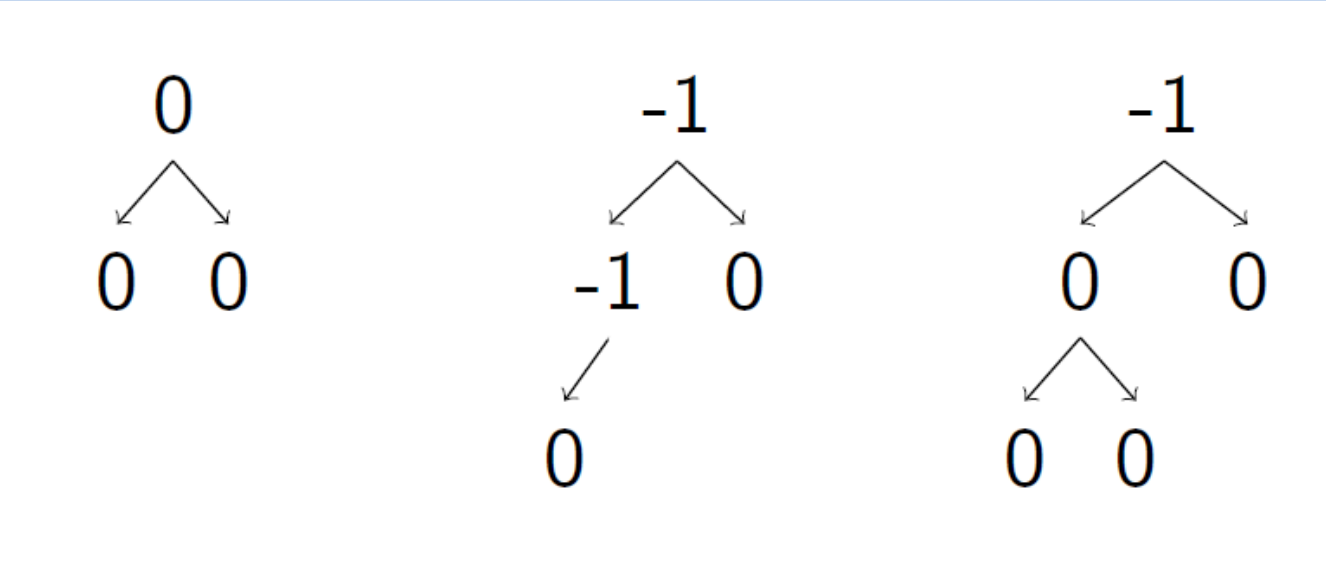
< 0 se a subárvore da esquerda for maior

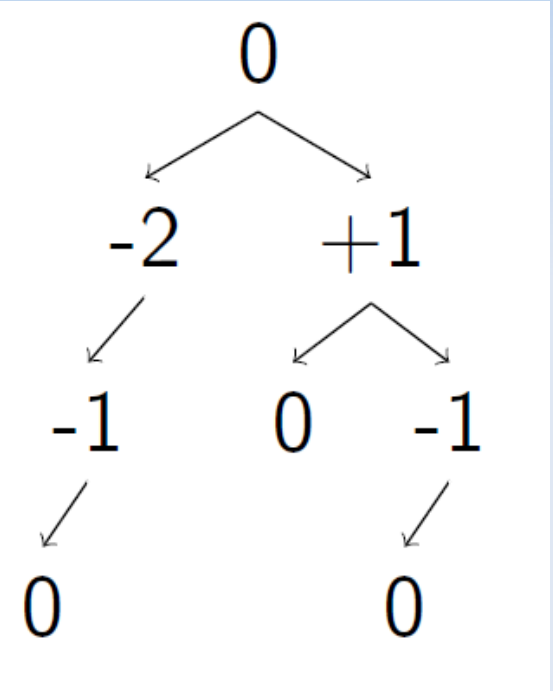
0 se forem do mesmo tamanho

> 0 se a subárvore da direita for maior

Árvores Binárias Balanceadas
(AVL)

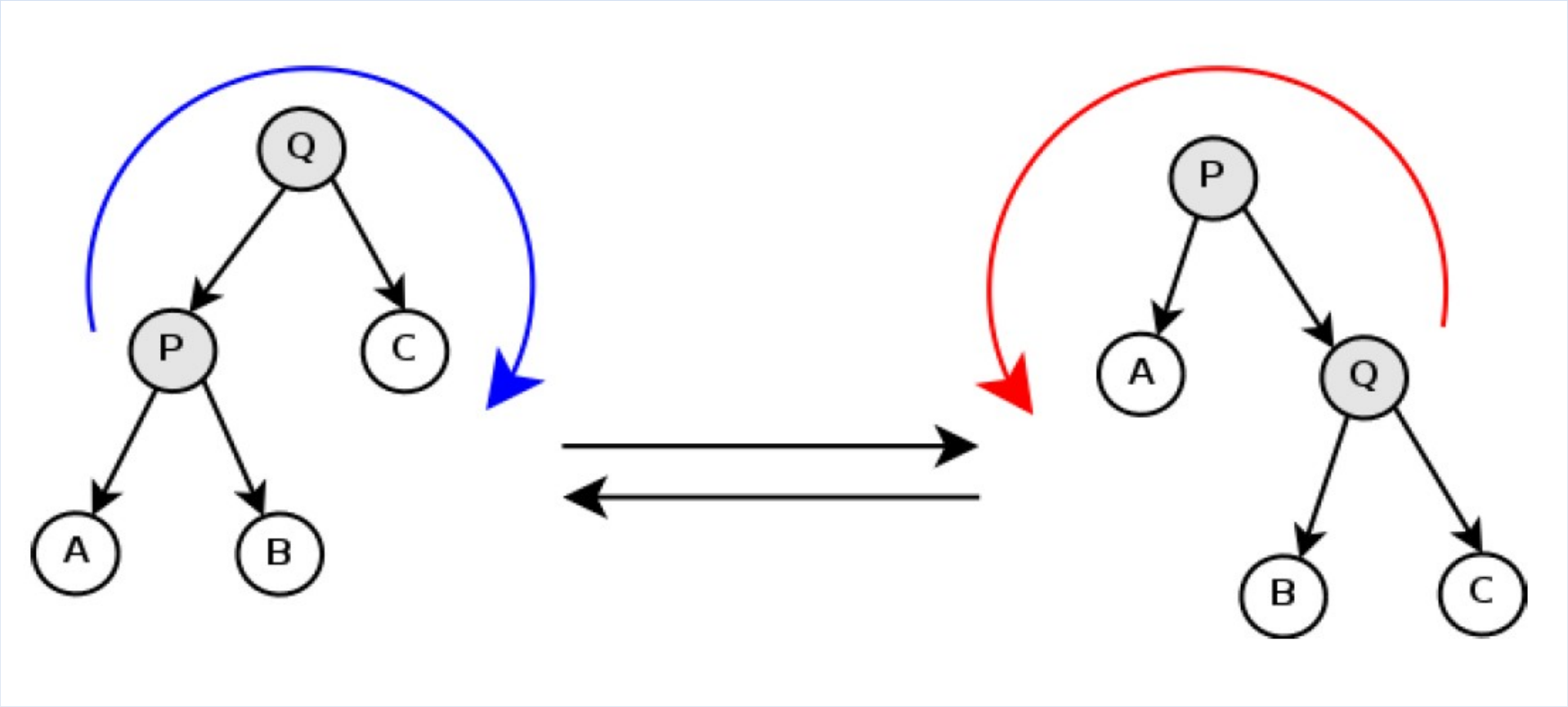
- Exemplos:





Árvores Binárias Balanceadas
(AVL)

- Nas operações de inserção e remoção de elementos, o balanceamento da árvore resultante é ajustado através da operação de rotação, que preserva a ordenação da árvore.



Árvores Binárias Balanceadas
(AVL)

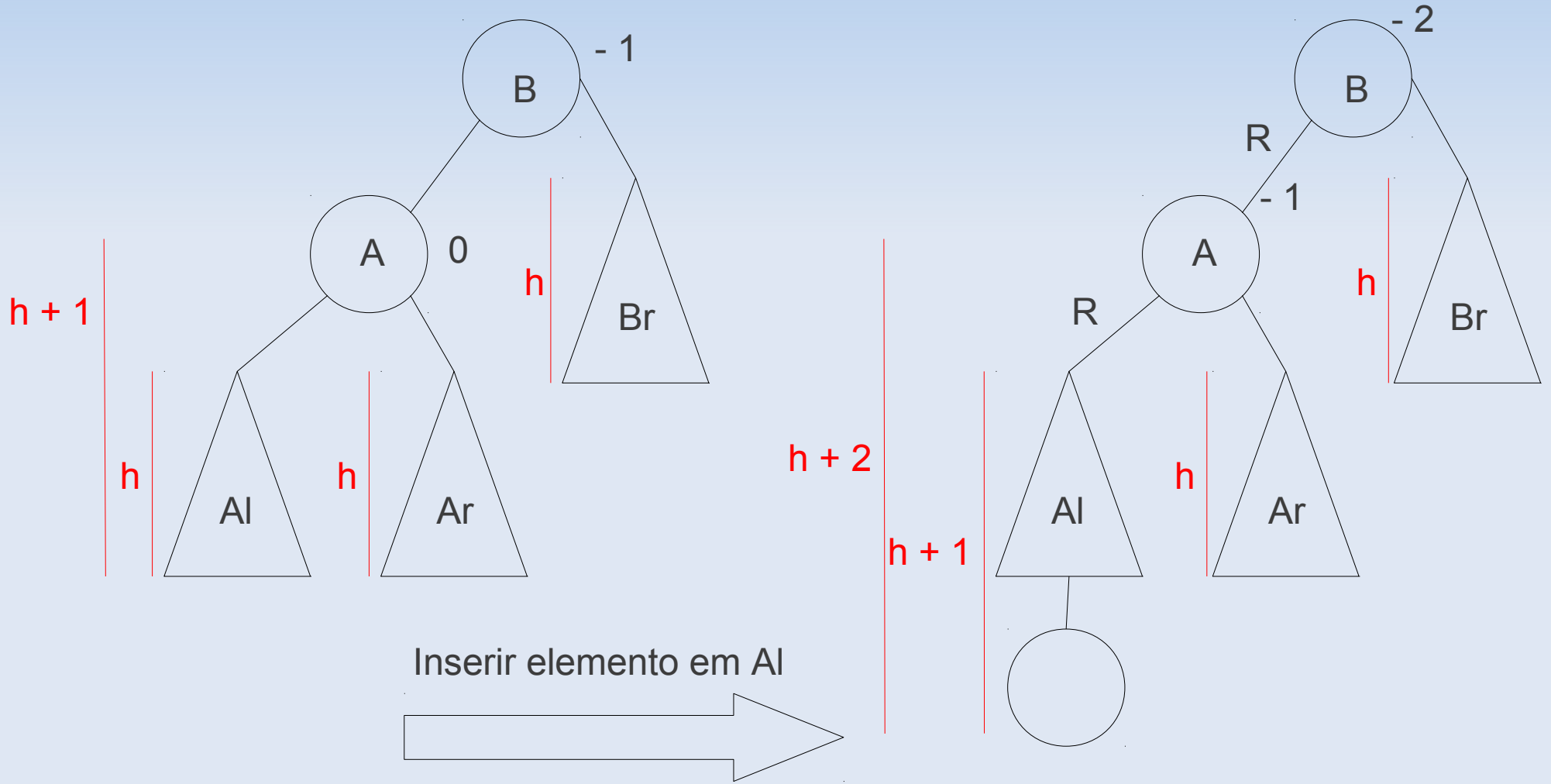
- Propriedades da rotação:
  - A rotação não destrói a propriedade de ordenação dos dados
  - Depois da rotação, os nós rotacionados ficam com fator de balanço zero
  - Depois da rotação, a árvore continua com a mesma altura que tinha anteriormente (antes da inserção que desbalanceou a árvore)

Árvores Binárias Balanceadas
(AVL)

- Quatro tipos de rotações
  - Rotação RR simples: as duas primeiras arestas no caminho da inserção vão para a direita (nó desbalanceado com um fator de balanço negativo e subárvore da esquerda com fator negativo)
  - Rotação LL simples: oposto do anterior - nó desbalanceado com um fator de balanço positivo e subárvore da direita com fator positivo
  - Rotação RL dupla: nó desbalanceado com um fator de balanço negativo e subárvore da esquerda com fator positivo
  - Rotação LR dupla: nó desbalanceado com um fator de balanço positivo e subárvore da direita com fator negativo

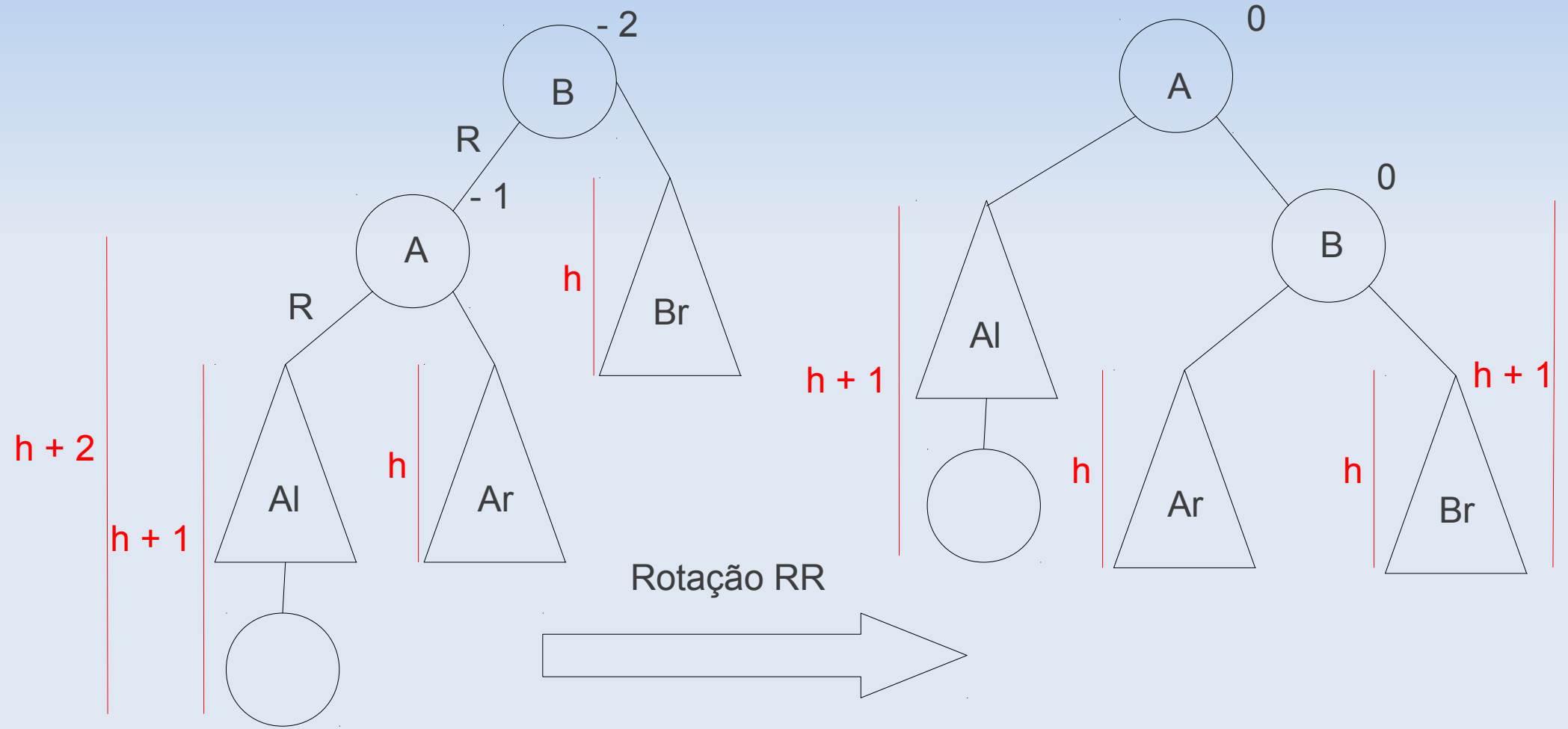
Árvores Binárias Balanceadas
(AVL)

- Rotação RR simples (Al, Ar e Br com a mesma altura h)



Árvores Binárias Balanceadas
(AVL)

- Rotação RR simples (Al, Ar e Br com a mesma altura h)



Árvores Binárias Balanceadas
(AVL)

- Rotação RR simples (Algoritmo)

```
rotacionaRR(No n){
        No temp = n.direita;
        n.direita = n.esquerda;
        n.esquerda = n.direita.esquerda;
        n.direita.esquerda = n.direita.direita;
        n.direita.direita = temp;
        Dado tempD = n.dado;
        n.dado = n.direita.dado;
        n.direita.dado = tempD;
}
```

Árvores Binárias Balanceadas
(AVL)

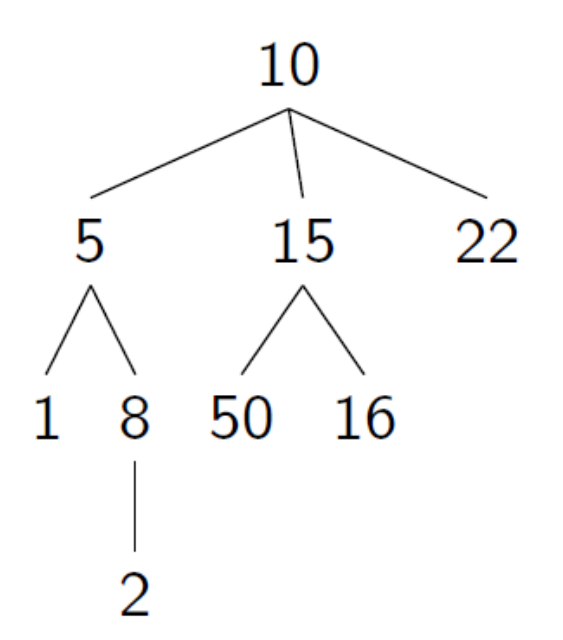
- Rotação LL simples
  - É o oposto da rotação RR
- Rotação RL
  - Primeiro faz-se uma rotação LL na subárvore da esquerda do nó desbalanceado
  - Depois uma rotação RR no nó
- Rotação LR (é o oposto da RL)
  - Primeiro faz-se uma rotação RR na subárvore da direita do nó desbalanceado
  - Depois uma rotação LL no nó

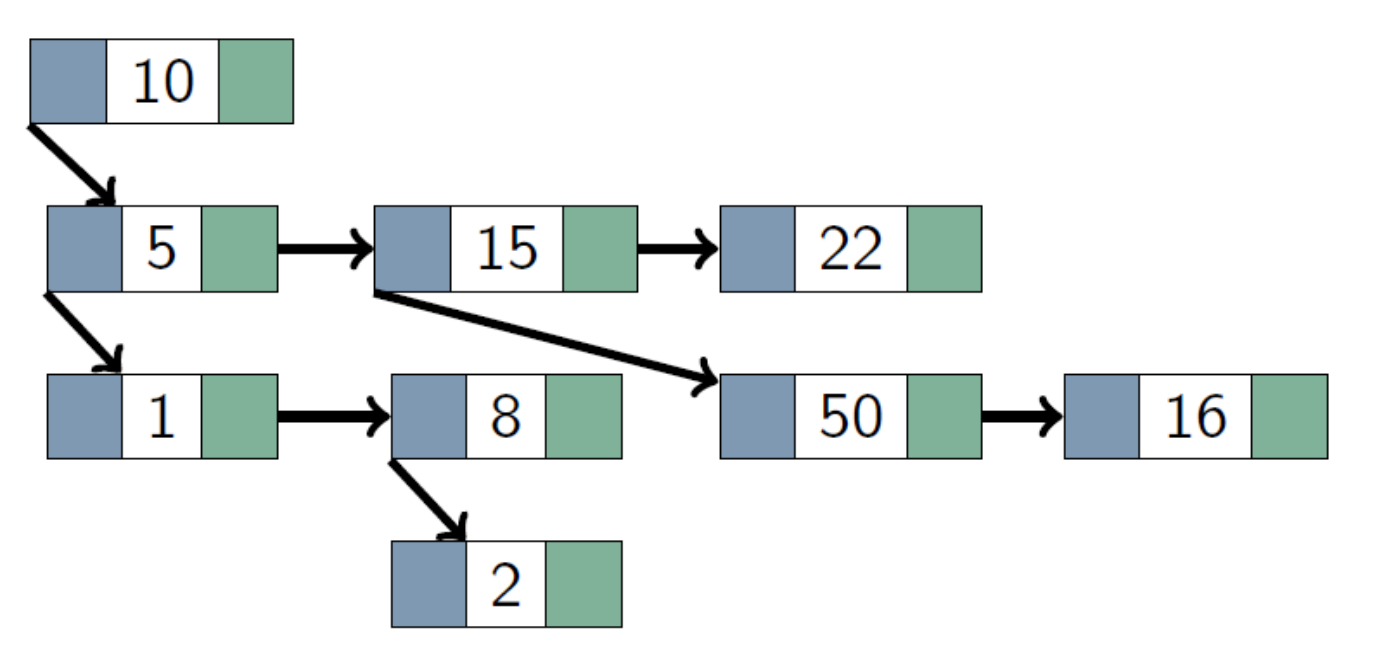
Árvores Binárias Balanceadas
(AVL)

- Como vimos, após uma inserção pode ser necessário fazer alguma rotação na árvore para mantê-la balanceada.
  - ”Sobe” a árvore (pelo caminho da insserção) atualizando os fatores de balanceamento e fazendo as rotações necessárias
- No caso da remoção, basta atualizar os fatores de balanceamento e verificar se precisa alguma rotação, da seguinte forma:
  - Remove por cópia
  - ”Sobe” a árvore atualizando o fator de balanceamento dos nós:
    - Se for +1 ou -1, esta balanceada, pode-se parar.
    - Se for 0, a altura diminuiu, é preciso continuar.
    - Se for +2 ou -2, está desbalanceada, é preciso uma rotação e atualização.

Árvores Genéricas

- Uma arvore generica pode possuir um número arbitrário de filhos por nó
- Como implementar se não sabemos a quantidade de filhos?
  - Usando vetores, listas
  - Ou ainda usando a mesma estrutura da árvore binaria com diferentes signicados



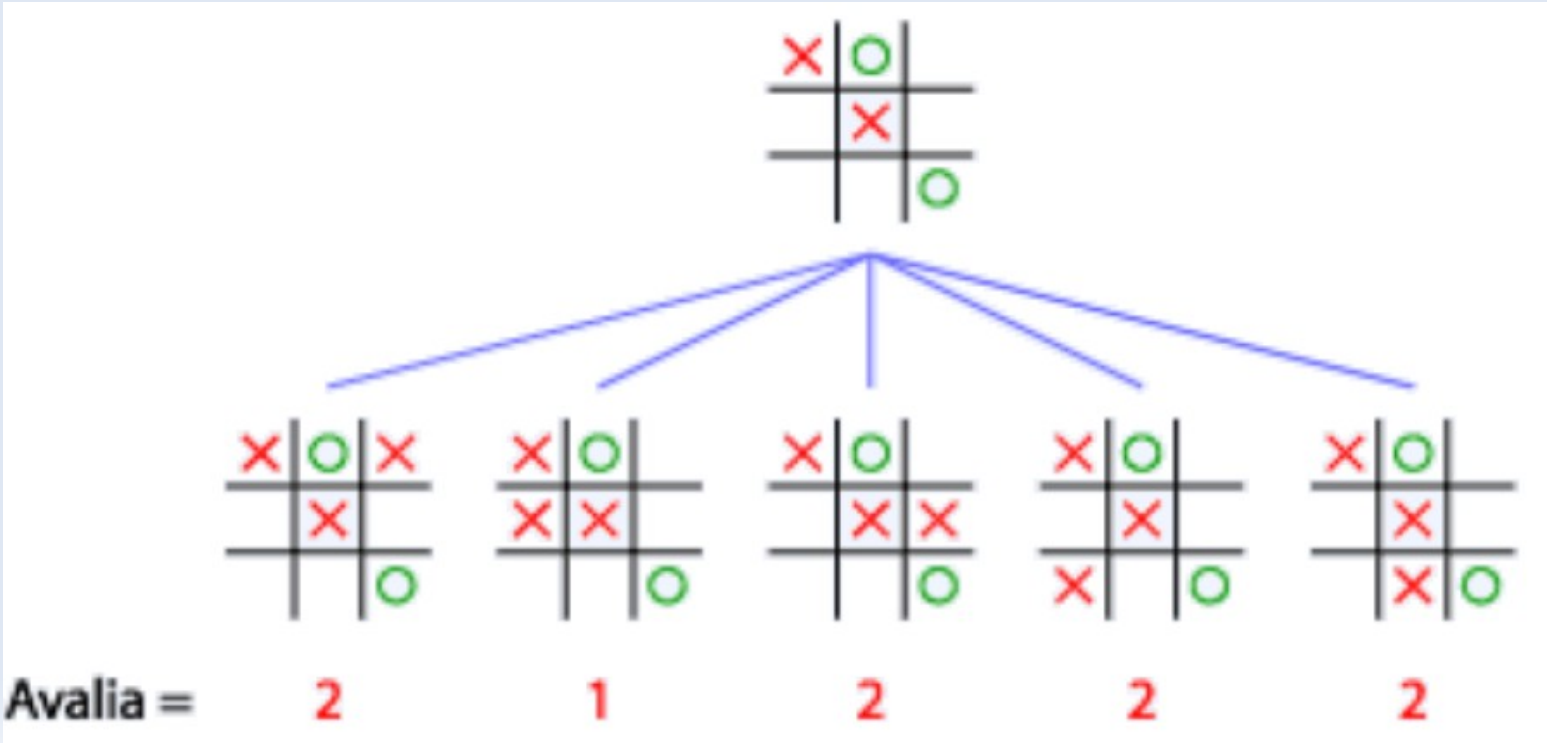


Árvores Genéricas

- Aplicação: Game Trees
  - Game Trees são árvores que representam as possibilidades de jogadas para um jogador a partir de um estado do jogo
    - Primeiro nível: estado atual do jogo
    - Segundo nível: jogadas possíveis do jogador (computador)
    - Terceiro nível: jogadas possíveis do oponente
    - Quarto: jogadas do jogador; Quinto: jogadas do oponente; e assim por diante.
  - Os fillhos de um nó representam todas as possibilidades a partir daquela situação do jogo
  - Função de avaliação que retorna um valor representando o quao bom está um estado do jogo (configuração do tabuleiro)
  - Altura da árvore indica o número de jogadas adiante que se deseja prever

Árvores Genéricas

- Aplicação: Game Trees
  - Game tree do jogo da velha
    - A função de avaliação retorna o valor do numero de linhas, colunas e diagonais abertas para o jogador (computador) menos o núumero de linhas, colunas e diagonais abertas ao adversário

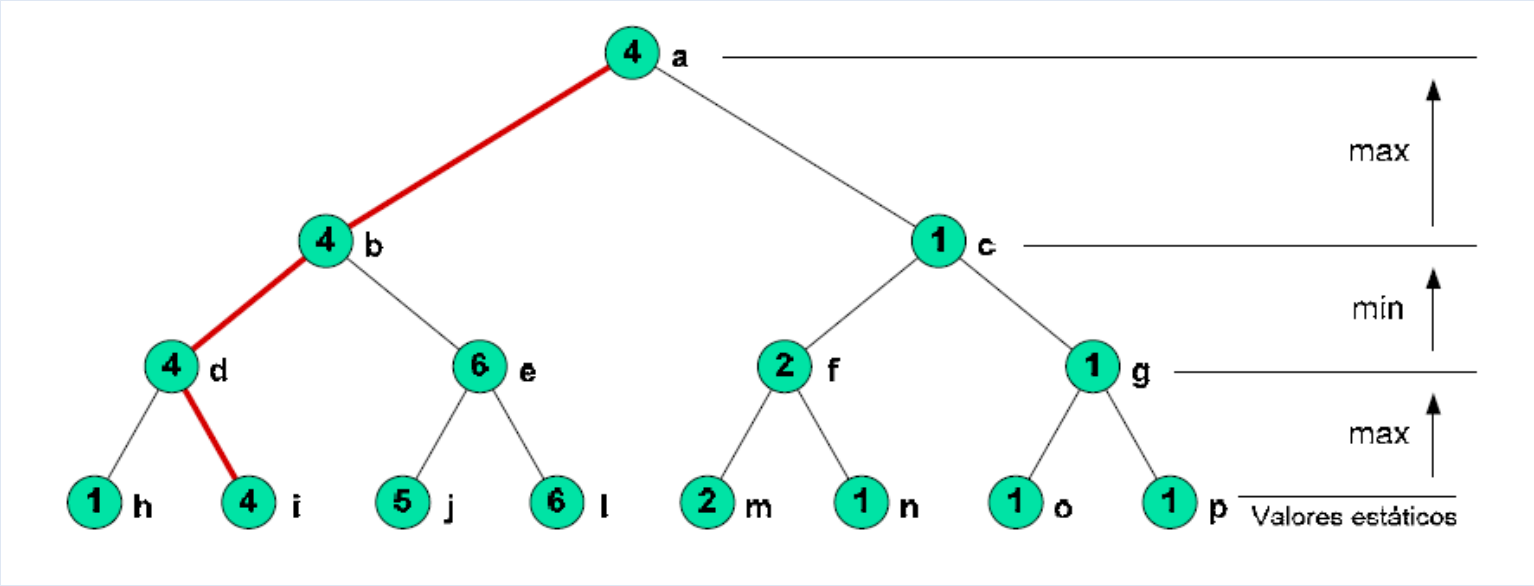


Árvores Genéricas

- Aplicação: Game Trees
  - Para determinar a melhor jogada, utiliza-se o método minimax
    - Quanto maior for o valor da função de avaliação, maior serão as chances do jogador vencer
    - Quanto menor for o valor, maior serão as chances do oponente vencer
    - O objetivo é tentar maximizar o valor dado pela função avaliação, i.e., selecionar a jogada que garanta a melhor situação ao fim de n jogadas
    - O objetivo é alcançado propagando o valor correspondendo ao melhor estado até ao nó raiz
    - Este valor corresponde ao ganho mínimo que se obtém se optarmos pela jogada correta

Árvores Genéricas

- Aplicação: Game Trees
  - Consideremos um problema genérico, onde os nós representam estados e os ramos representam as jogadas possíveis a partir de cada estado. Os valores associados aos nós folha são obtidos por uma função de avaliação



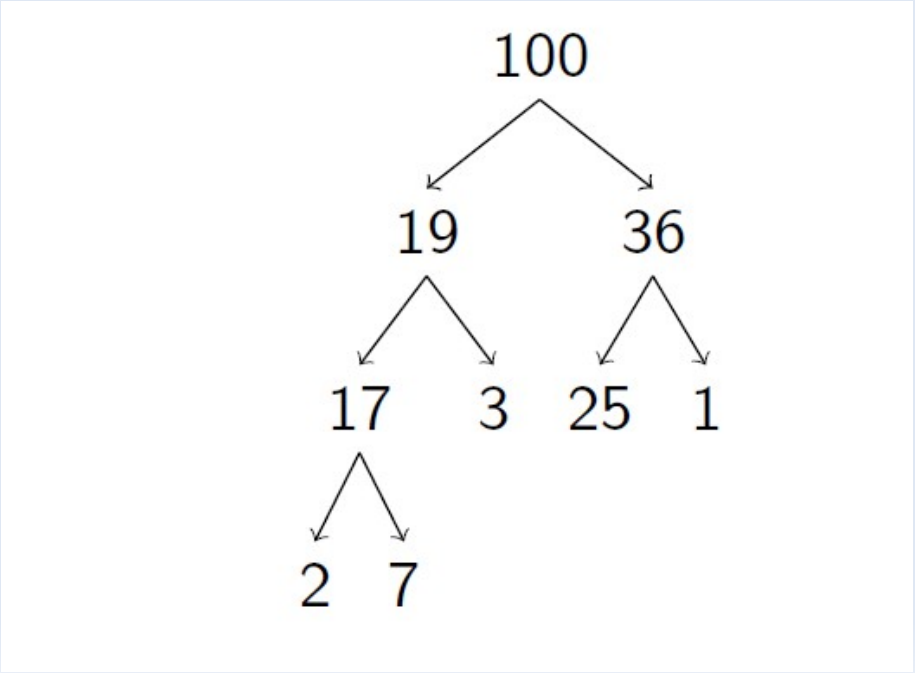
Árvores Genéricas

- Aplicação: Game Trees
  - Algoritmo minimax:
    - Se nó for folha: retorna o valor da função de avaliação
    - Se nó representa jogada do oponente: executa-se minimax recursivamente em cada filho e retorna o menor valor encontrado
    - Se nó representa jogada do jogador: executa-se minimax recursivamente em cada filho e retorna o maior valor encontrado

Heaps

- Estrutura de dados abstrata, derivada da árvore, que satisfaz a propriedade:
  - Se B é filho de A, então B.chave <= A.chave (heap de máximo)
  - Se B é filho de A, então B.chave >= A.chave (heap de mínimo)
- Pode ser construda em tempo linear.
- Não há restrições quanto ao número de filhos por nó
  - Na prática: 2

    (heaps binárias)

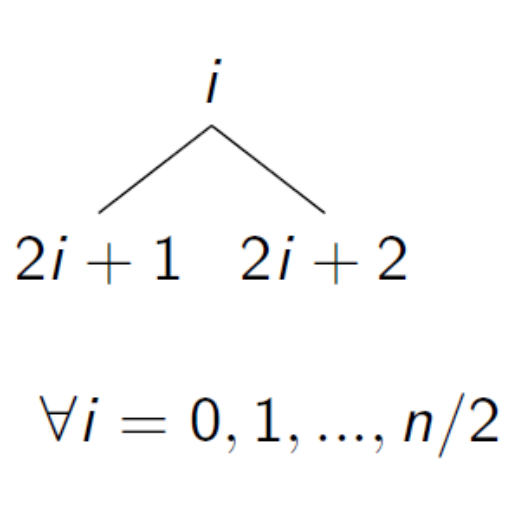


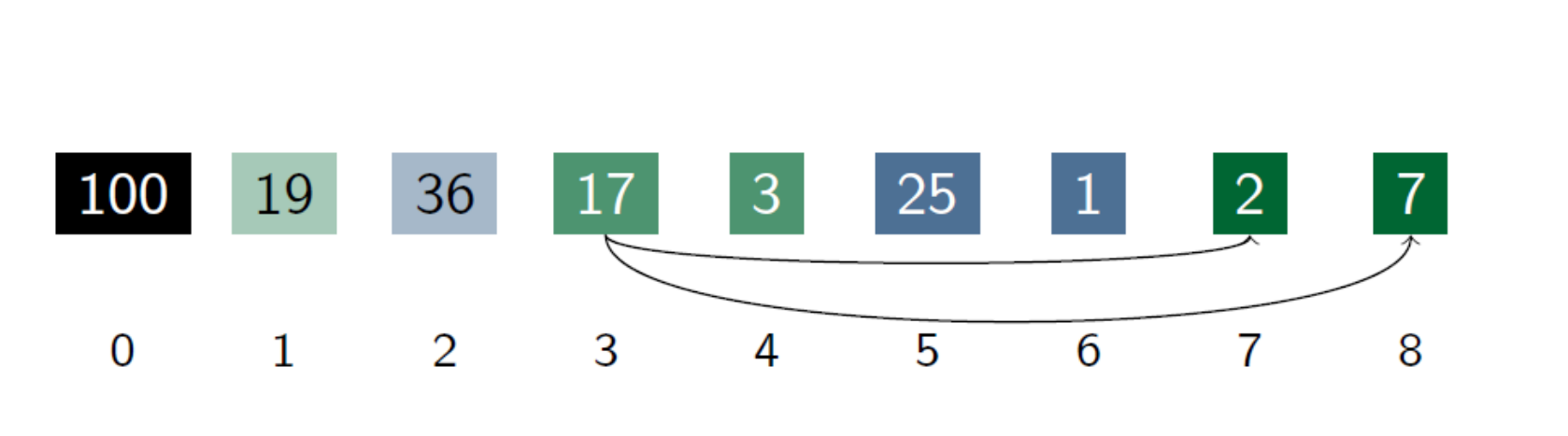
Heaps

- Operações comuns:
  - busca-max: encontra o máximo item (ou busca-min)
  - remove-max: remove a raiz (ou remove-min)
  - insere: insere um novo valor
  - fusão: une duas heaps (como uma heap)

Heaps

- Implementação
  - A forma mais eficiente de implementar uma heap é usando um vetor





Filas de Prioridade

- Uma heap é a estrutura mais eficiente para implementar uma fila de prioridade
- Uma fila de prioridade é uma lista de itens na qual cada item está associado a uma prioridade
  - Em geral, itens distintos possuem prioridades diferentes
- Os itens são inseridos na fila de prioridade em uma órdem arbitrária qualquer, mas são removidos de acordo com sua prioridade
- Operações: enfileirar, encontraMaiorPrioridade, removeMaiorPrioridade.
- Exemplo de utilização: software de gerenciamento da impressora