

# Capítulo 8

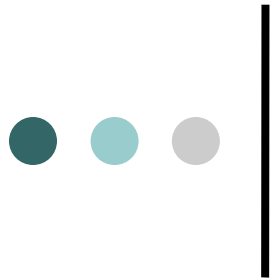
Árvores B

Parte 1



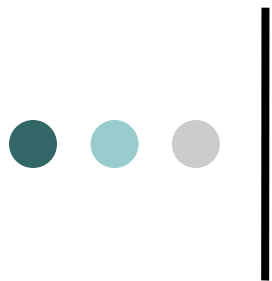
# Programa

- Introdução
- Operações básicas sobre arquivos
- Armazenagem secundária
- Conceitos básicos sobre estrutura de arquivo
- Organizando arquivos para desempenho
- Indexação
- Processamento co-seqüencial e ordenação
- **B-Tree e outras organizações em árvores**
- B+Tree e acesso seqüencial indexado
- Hashing
- Hashing estendido



# O problema

- Arquivo de índices muito grande que não cabem na memória:
  - Mantém-se o arquivo de índice em memória secundária.
  - PROBLEMA: O acesso a memória secundária é lento
- 2 problemas mais específicos:
  - Problema 1 - Pesquisa binária requer muitos seeks.
  - Problema 2 - Pode se tornar muito caro manter um arquivo de índices ordenado tal que a pesquisa binária possa ser feita.



# Armazenagem e Recuperação em Árvores e Blocos

- Objetivos
  - Resolver problemas de:
    - ordenação (evita a ordenação global).
    - pesquisa por chave.
    - acesso ordenado por chave.
    - acesso sequencial.



# Principais Metodologias

- Árvore Binária
- Árvores AVL
- Árvores paginadas
- Árvores B



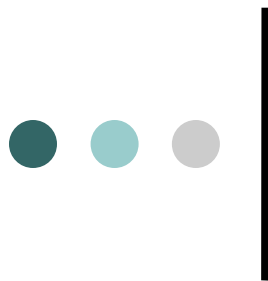
## Pesquisa Binária

### Mantendo as chaves ordenadas

- Para se resolver o problema da ordenação (*problema - 2*), pode-se criar uma estrutura que faça com que a eliminação e a inserção de registros tenham efeito local.

Árvore de Pesquisa Binária é uma solução:

- os elementos da subárvore à esquerda têm chave menor que a da raiz.
- os elementos da subárvore à direita têm chave maior ou igual a da raiz.



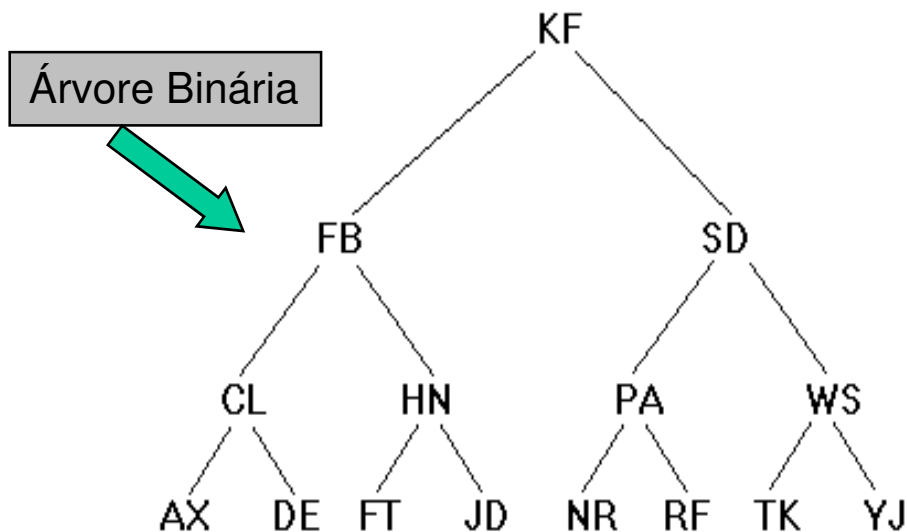
# Árvore de Pesquisa Binária

(como uma solução)

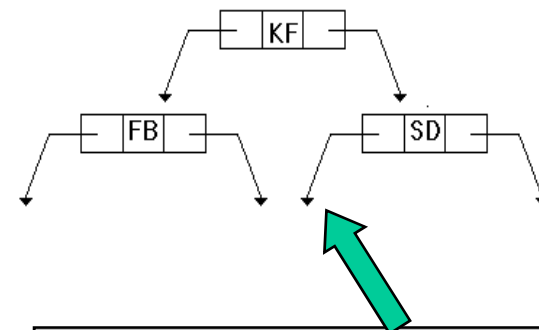
- Dada a lista ordenada, podemos expressar a pesquisa binária desta lista como uma árvore binária.

Lista de chaves ordenadas:

AX CL DE FB FT HN JD KF NR PA RF SD TK WS YJ



Cada nó representa um reg. Contém a chave e a PRR.



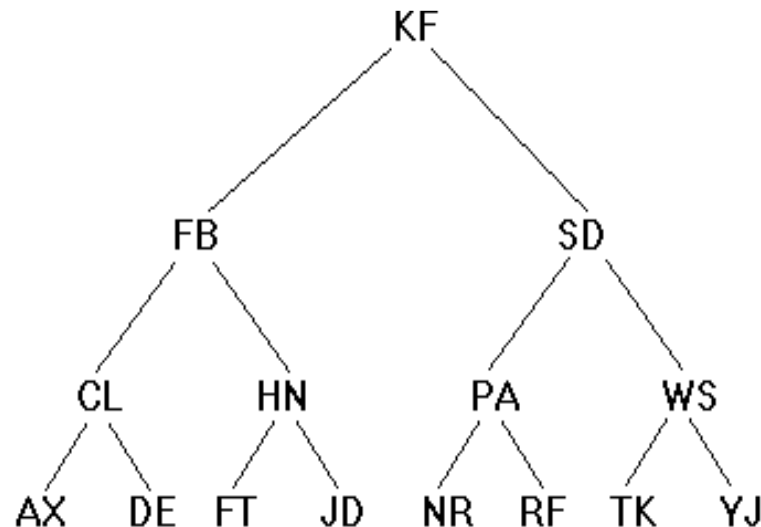
Representação parcial dos links dentro do nós da árvore binária.



# Árvore de Pesquisa Binária

(como uma solução)

- Um arquivo de índices linear pode ser obtido percorrendo a árvore de pesquisa binária “in order”, isto é:
  - subárvore da esquerda, visita raiz, subárvore da direita.



**AX CL DE FB FT HN JD KF NR PA RF SD TK WS YJ**

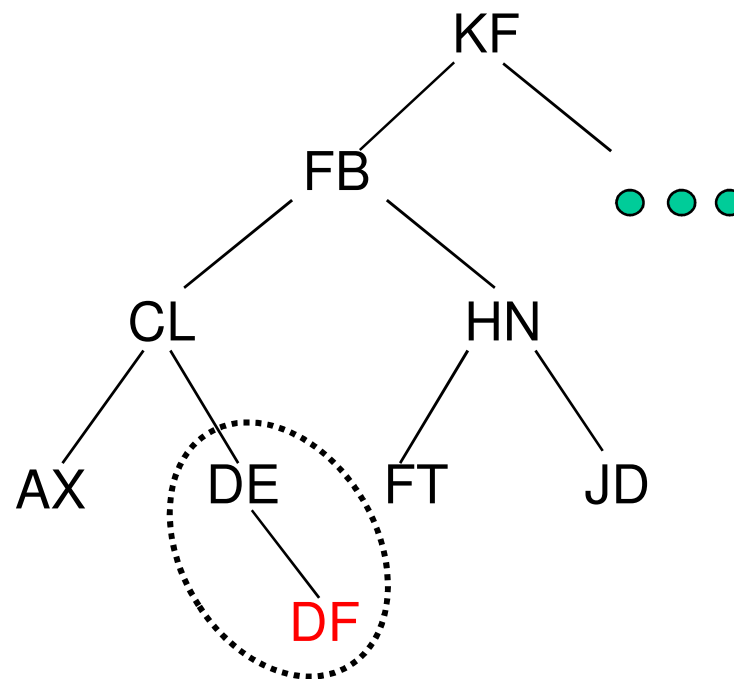


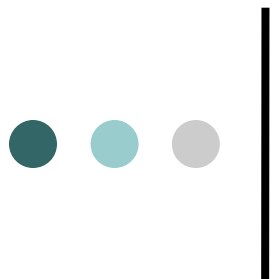


# Árvore de Pesquisa Binária

(como uma solução)

- A inserção de um elemento, por exemplo DF, afeta a árvore apenas localmente.





# Árvore de Pesquisa Binária

(como uma solução)

- Com arquivo de índices (linear), a inserção do elemento DF:
  - causaria um “deslocamento” dos elementos de DE em diante.

Antes:      AX CL DE FB FT HN JD KF NR PA RF SD TK WS YJ

Depois:     AX CL DE **DF** FB FT HN JD KF NR PA RF SD TK WS YJ

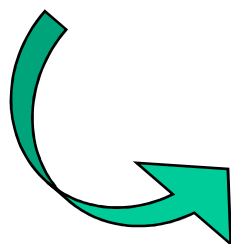
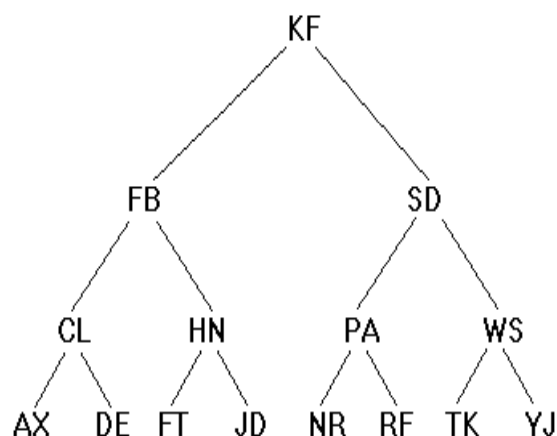
Por este motivo, o uso da árvore binária é mais indicado do que um índice linear para arquivos que sofrem muita inserção e retirada de registros.



# Árvore de Pesquisa Binária

(como uma solução)

- Representação em matriz de adjacências



Raiz = 9

	key	filho esq.	filho dir.
0	FB	10	8
1	JD		
2	RF		
3	SD	6	13
4	AX		
5	YJ		
6	PA	11	2
7	FT		

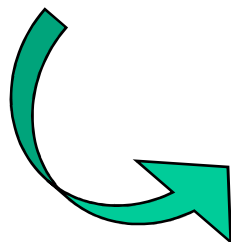
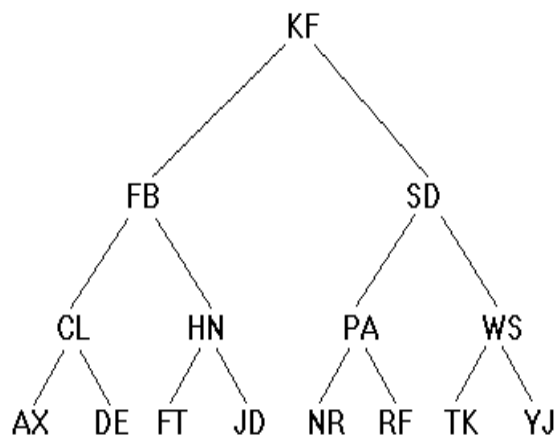
	key	filho esq.	filho dir.
8	HN	7	1
9	KF	0	3
10	CL	4	12
11	NR		
12	DE		
13	WS	14	5
14	TK		

OBS: pode-se usar (-1) para indicar ausência de filho.

# Árvore de Pesquisa Binária

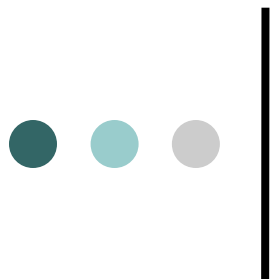
(como uma solução)

- Representação via ponteiros



	chave	esq	dir		chave	esq	dir
1	KF	2	3	9	DE	-	-
2	FB	4	5	10	FT	-	-
3	SD	6	7	11	JD	-	-
4	CL	8	9	12	NR	-	-
5	HN	10	11	13	RF	-	-
6	PA	12	13	14	TK	-	-
7	WS	14	15	15	YJ	-	-
8	AX	-	-				

Representação Encadeada.



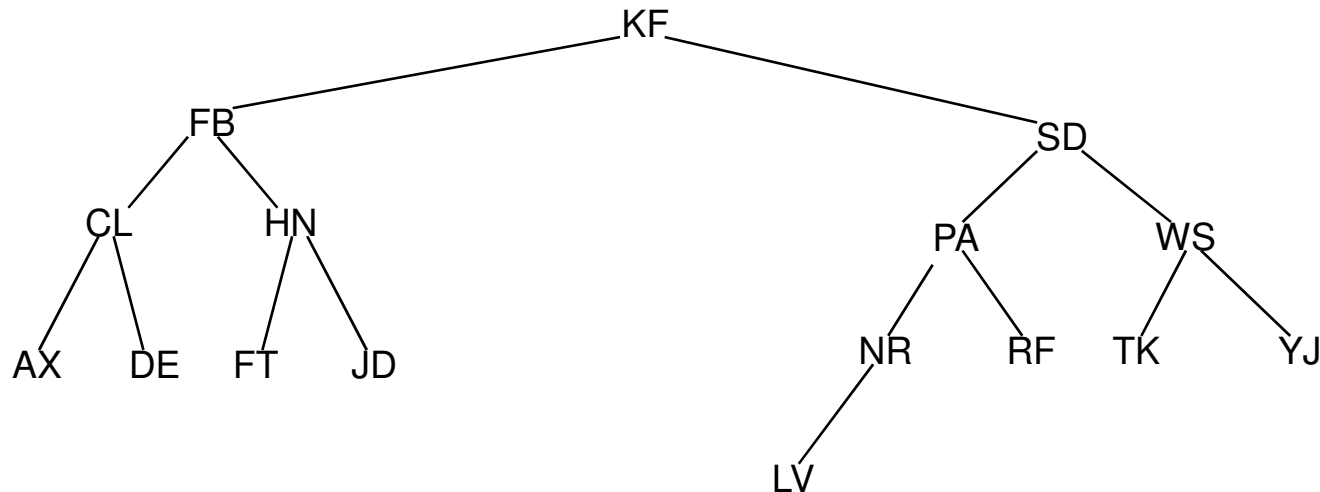
# Árvore de Pesquisa Binária

(como uma solução)

- **VANTAGENS:**
  - A ordem dos registros não está associada à ordem no arquivo físico.
  - O arquivo físico não precisa estar ordenado.

# Árvore de Pesquisa Binária

(como uma solução)



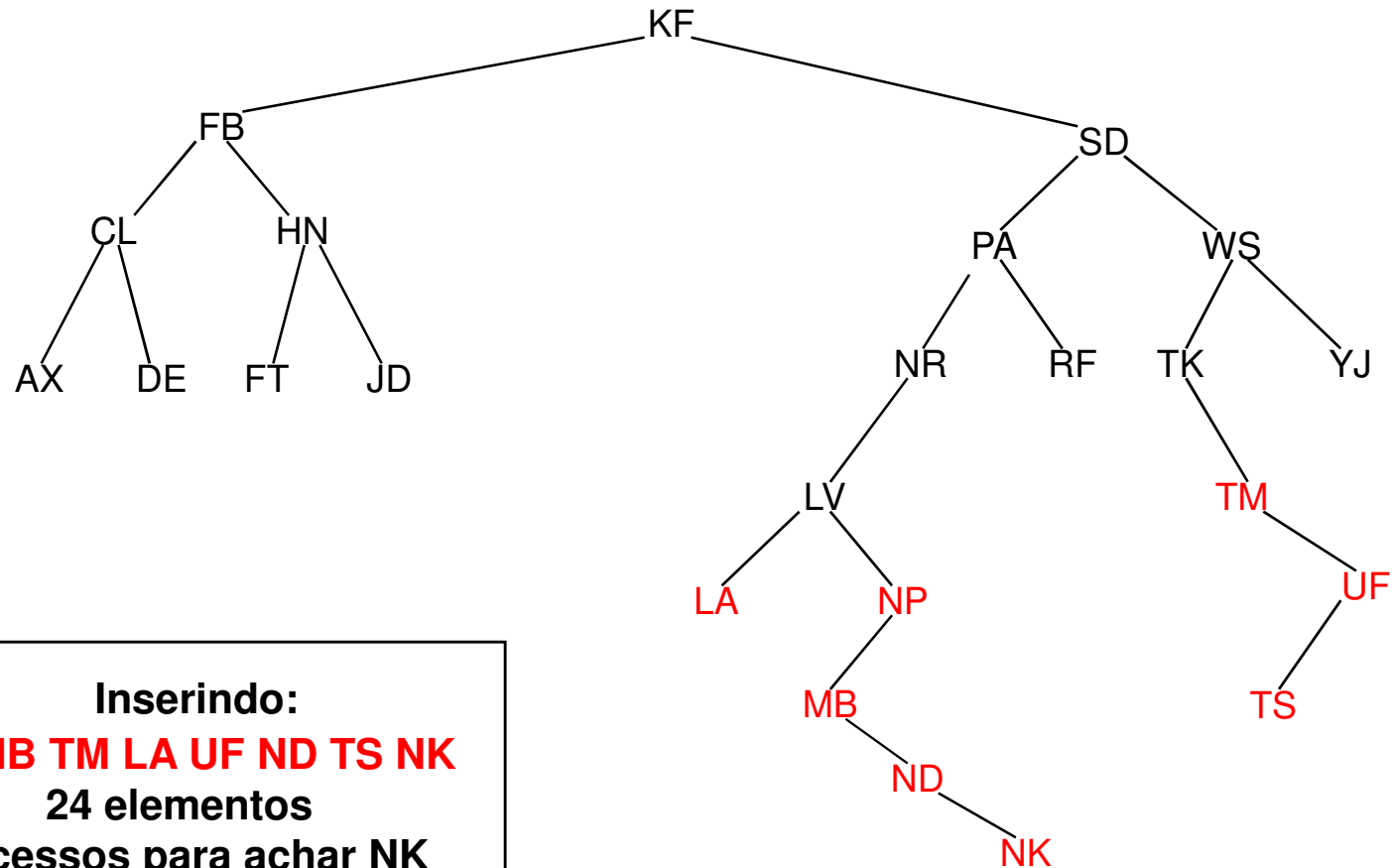
- Incluir as chaves NP, MB, TM, LA, UF, ND, TS e NK na sequência em que elas aparecem:
  - gera desbalanceamento da árvore.

**Árvore balanceada:** altura do **menor** caminho para uma folha difere da altura do **maior** caminho por apenas um nível.

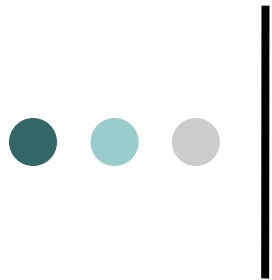


# Árvore de Pesquisa Binária

(como uma solução)



**Inserindo:**  
**NP MB TM LA UF ND TS NK**  
24 elementos  
9 acessos para achar NK



# Árvore de Pesquisa Binária

(como uma solução)

- **DESVANTAGENS:**

- A inclusão de novos elementos pode levar a um desbalanceamento da árvore.
  - Nesse caso, o esforço de busca por uma dada chave pode aumentar muito, ficando maior que  $O(\log_2 N)$ .
    - Para localizar a chave NK foram necessários 9 acessos ( $\log_2 24 = 4,585$ ).
- Busca com mais que 4 ou 5 acessos em disco é intolerável.





# Árvores Especiais

- Árvores AVL
  - Solução para manter índices ordenados
    - Mantém a árvore balanceada.
- Árvores Paginadas
  - Solução para diminuir custo de seek
    - Um nó é composto por uma página com diversos índices.



# Árvores AVL

- Proposta em 1962 pelos russos:  
G.M. Adel'son-Vel'skii e E.M. Landis
  - O método nunca usa mais que  $O(\log_2 N)$  operações para pesquisar ou inserir um item na árvore.
  - Uma árvore AVL é uma árvore de altura balanceada.
    - membro da classe mais geral de árvores de altura balanceadas conhecidas como HB(k) – “*Height Balanced*”.

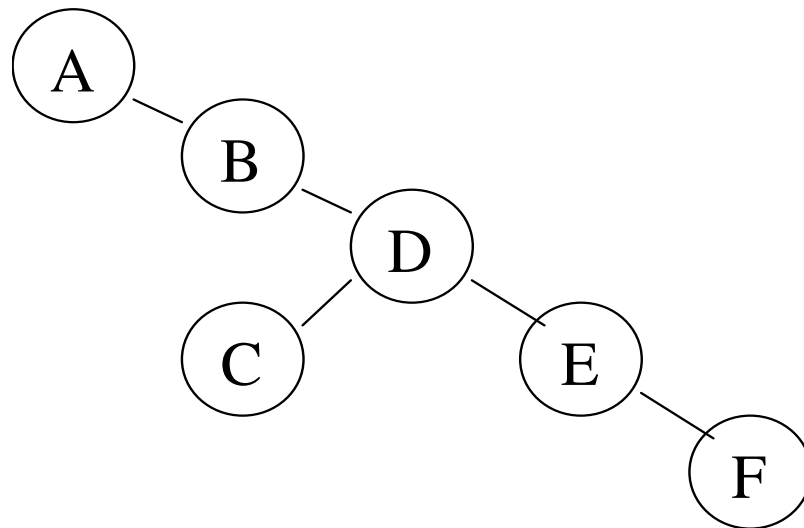


# Árvores AVL

- É uma árvore binária de pesquisa construída de tal modo que, para qualquer nó, a altura da sua subárvore à direita difere da altura da sua subárvore à esquerda em no máximo um nível.
- Essa limitação visa garantir que a busca seja a mais rápida possível.
- Uma árvore AVL é uma árvore de altura balanceada 1 ou seja uma HB(1).

# Árvores AVL

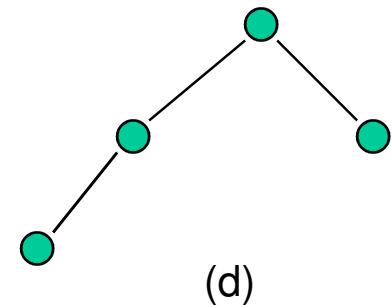
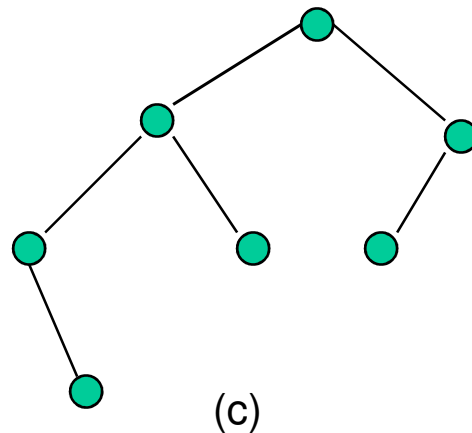
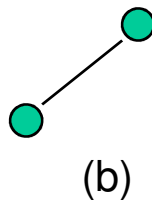
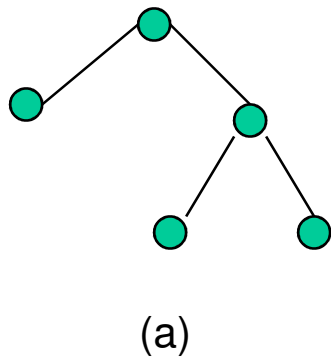
- Exemplo de árvore não balanceada (**não AVL**)



- O percorrimento nessa árvore será muito próximo do acesso sequencial para achar a chave F.

# Árvores AVL

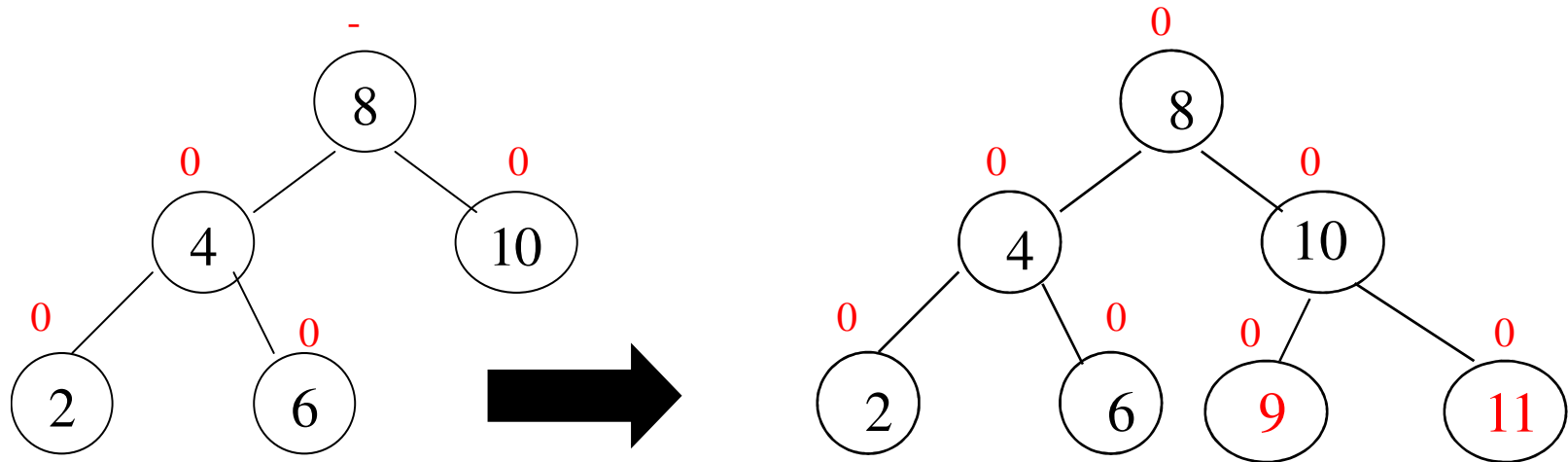
- São árvores AVL?



- Todas são exemplos de árvores AVL pois as alturas da subárvores da esquerda diferem das alturas das subárvores da direita por no máximo um nível.

# Árvores AVL

- Inserção preservando o balanceamento:



**- , 0 , + = fator de balanço**  
**= altura da subárvore direita – altura da subárvore esquerda**

# ● ● ● | Árvores AVL

- O problema surge quando:
  - A inserção é feita numa subárvore à direita de um nó com fator de balanço  $+1$
  - Dualmente, se a inserção é feita numa subárvore à esquerda de um nó com fator de balanço  $-1$ .

Manutenção de altura balanceada pode ser garantida através de rotações (de complexidade limitada)

# ● ● ● | Árvores AVL

- Operações de balanceamento dinâmico:
  - rotação simples à esquerda.
  - rotação simples à direita.
  - rotação dupla à direita e à esquerda.
  - rotação dupla à esquerda e à direita.

## Notação:

P: raiz desbalanceada mais próxima.

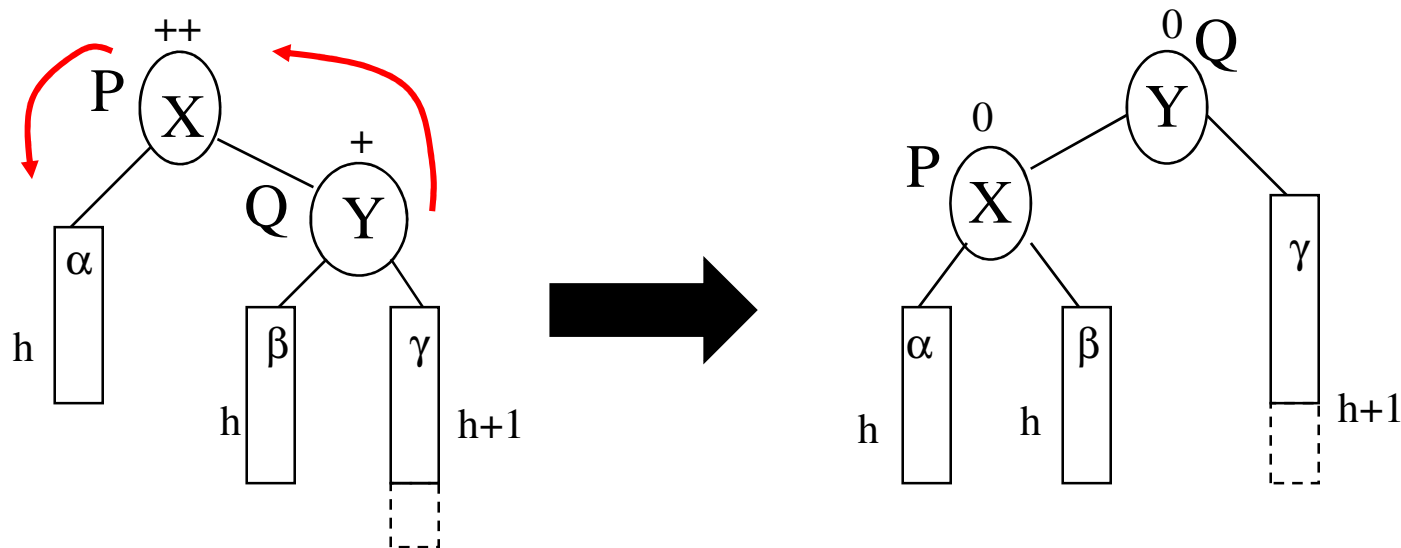
Q: filho de P, raiz da subárvore alterada.

retângulos: subárvores dos nós P ou Q.

h: altura da subárvore.



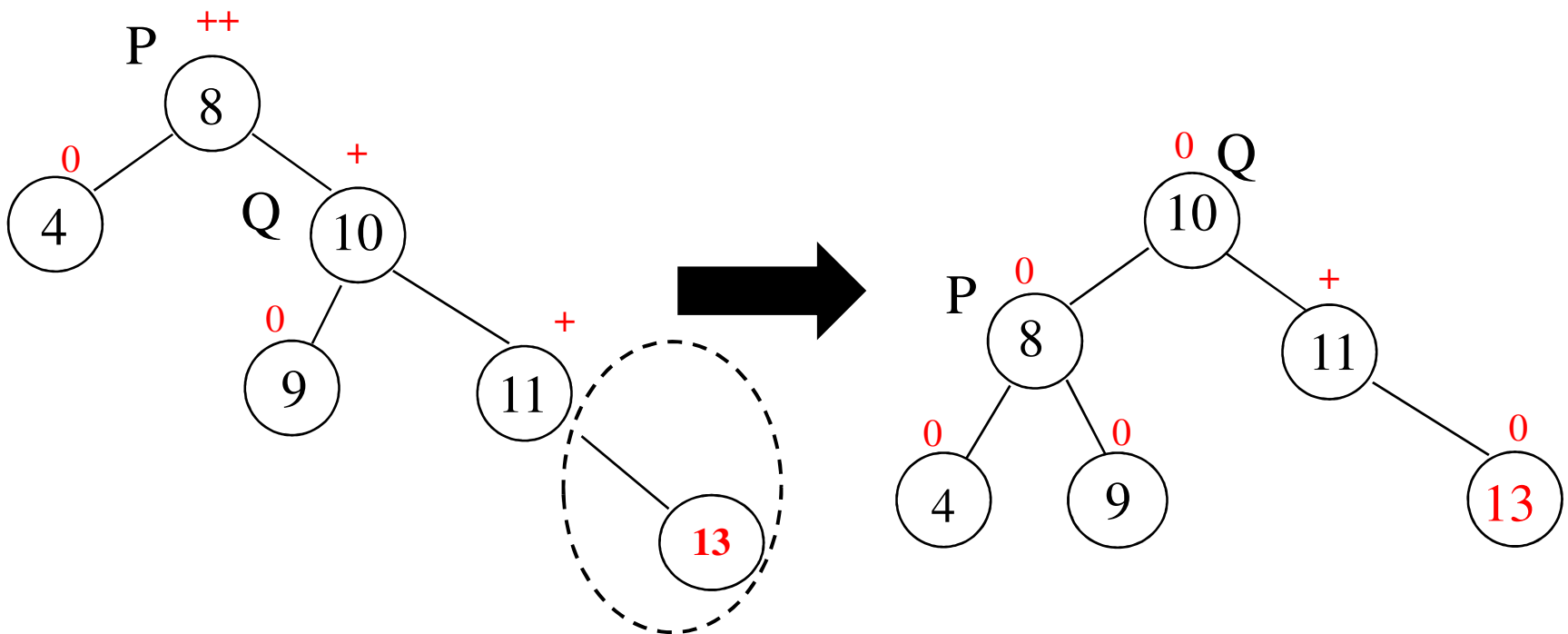
# Árvores AVL: Rotação Simples à Esquerda



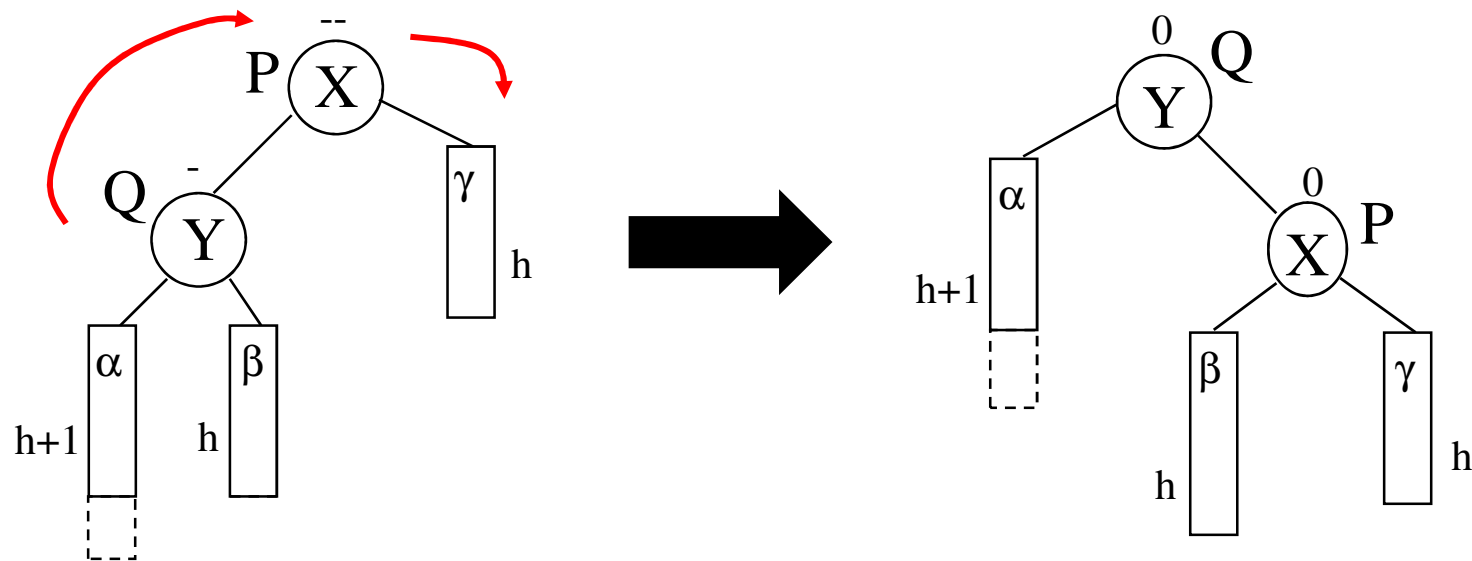
- Um novo elemento é incluído à direita, na subárvore à direita de P, desbalanceando P. Uma rotação simples de Q à esquerda, em torno de P:
  - Q torna-se a nova raiz no lugar de P.
  - $\beta$  é vinculada à direita de P.
  - P é vinculado à esquerda de Q, nó antes ocupado lugar por  $\beta$ .

# Árvores AVL: Rotação Simples à Esquerda

- Exemplo:



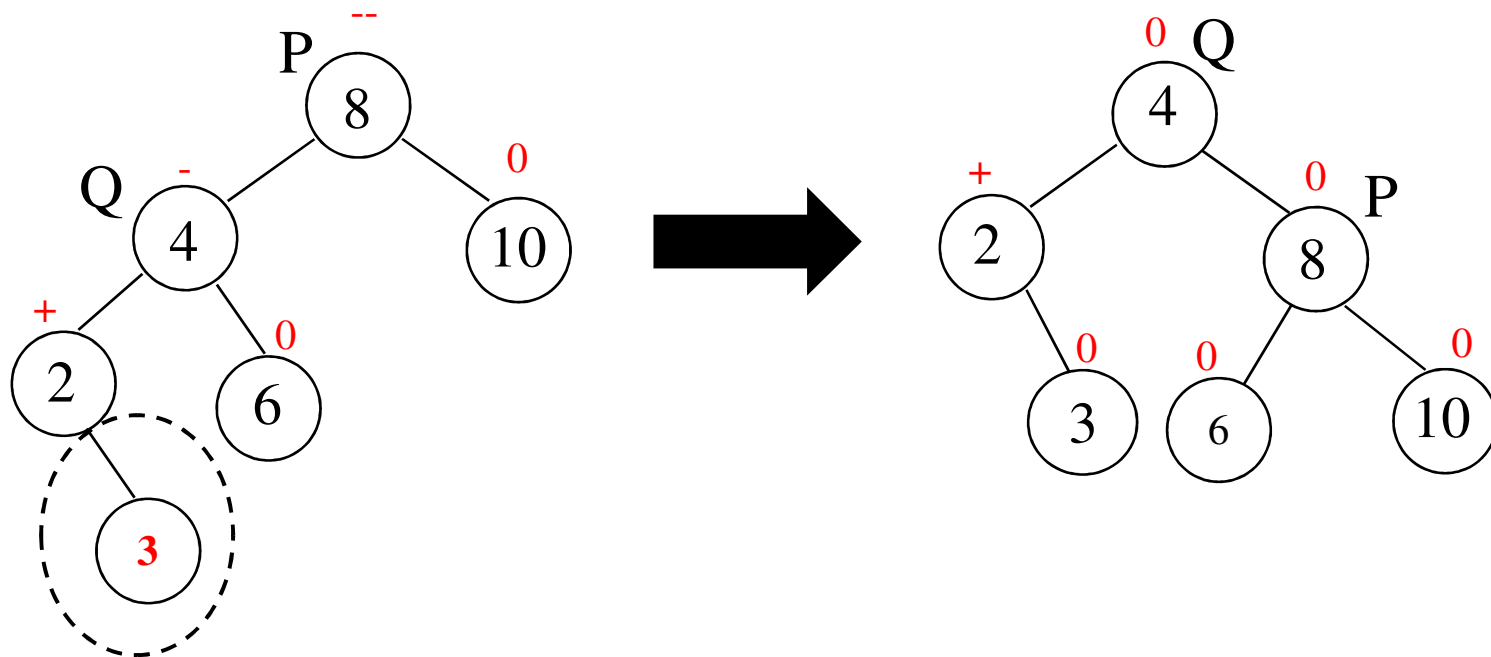
# Árvores AVL: Rotação Simples à Direita



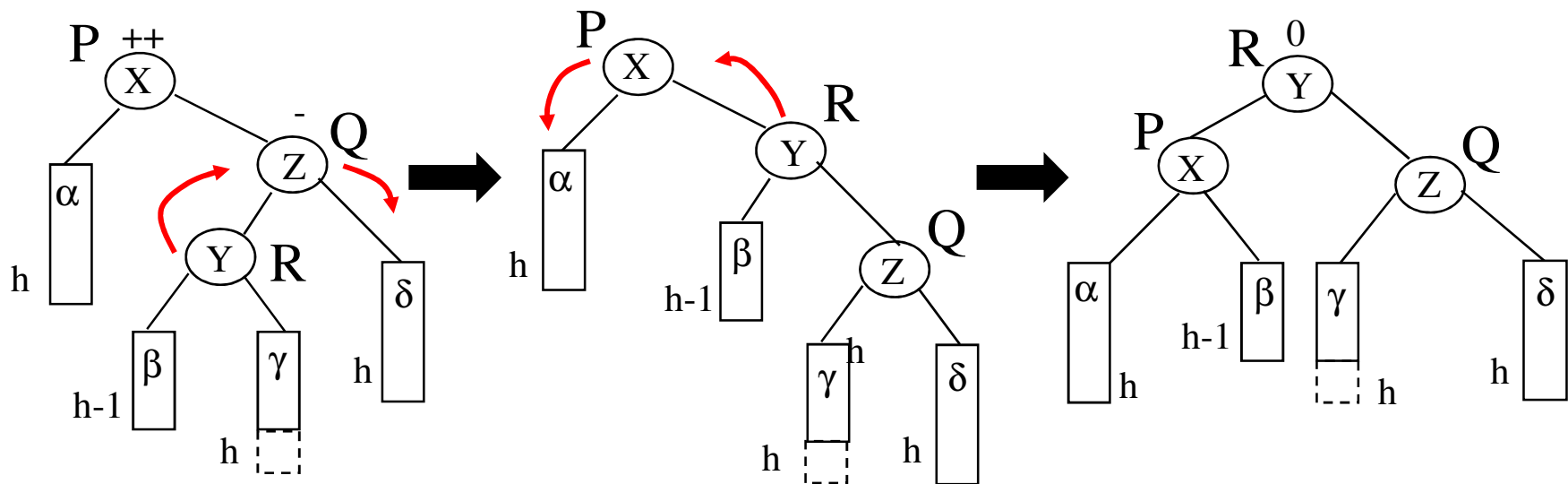
- Um novo elemento é incluído à esquerda, na subárvore à esquerda de P, desbalanceando P. Uma rotação simples de Q à direita, em torno de P:
  - Q torna-se a nova raiz no lugar de P.
  - $\beta$  é vinculada à esquerda de P.
  - P é vinculado à direita de Q, no lugar antes ocupado por  $\beta$ .

# Árvores AVL: Rotação Simples à Direita

- Exemplo:



# Árvores AVL: Rotação Dupla: Direita e Esquerda

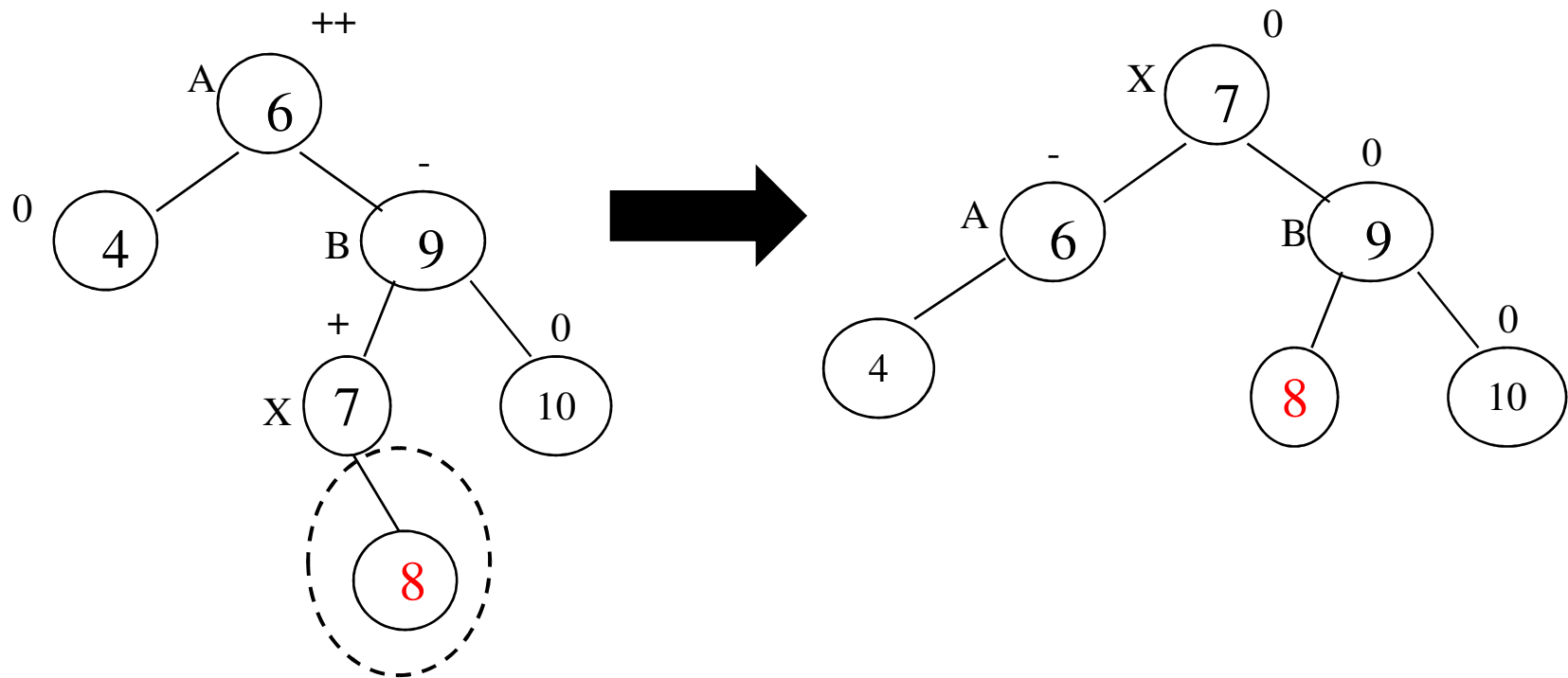


- Rotação à direita de  $R$  em torno de  $Q$ .
- Rotação à esquerda de  $R$  em torno de  $P$ .

# Árvores AVL:

## Rotação Dupla: Direita e Esquerda

- Exemplo:





# Árvores AVL

- Os algoritmos de busca, inserção e retirada podem ser vistos em:
  - Knuth, Donald E. The Art of Computer Programming. Volume 3: Sorting and Searching. Reading Massachusetts: Addison-Wesley, 1973.
  - Piccolo, Homero. Estrutura de Dados. Editora MSD, 2000.



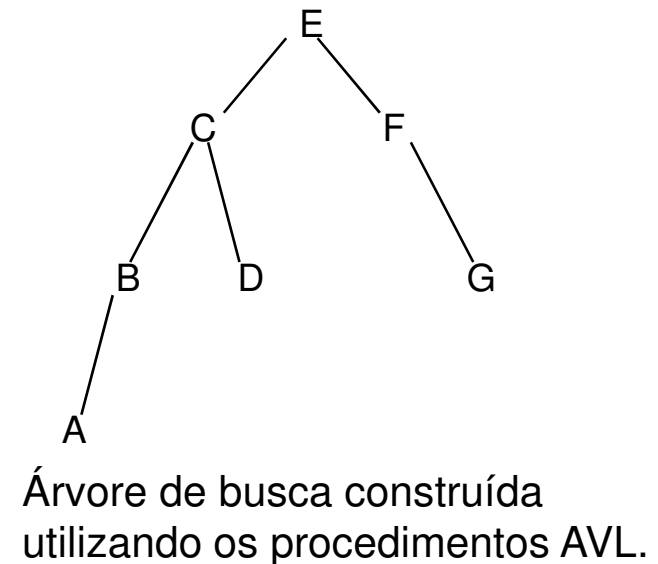
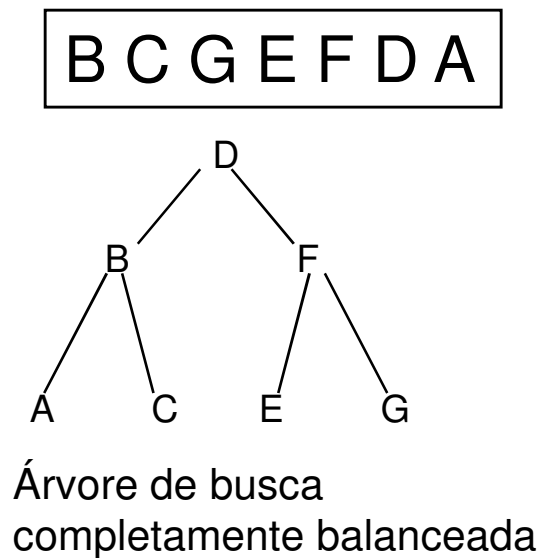
# Árvores AVL

- Árvores AVL não são aplicadas diretamente a maioria dos problemas de estruturas de arquivos, porque como todas as árvores binárias, elas tem muitos níveis (são muito profundas).
- Entretanto, no contexto que estamos discutindo (problema de acessar arquivos de índices que não cabem na memória), as árvores AVL são interessantes porque sugerem que é possível definir procedimentos para manter o balanço da árvore.



# Árvores AVL

- O fato de uma árvore AVL ser de altura balanceada 1 ( $HB(1)$ ), garante que a performance de busca se aproxima da performance em uma árvore completamente balanceada.





# Árvores AVL

- Efetuar uma busca, dadas N chaves:
  - Árvore completamente balanceada
    - No máximo:  $\log_2 (N+1)$
  - Árvore AVL
    - No máximo:  $1.44 \log_2 (N+2)$
- Dadas 1.000.000 chaves:
  - Árvore completamente balanceada
    - Requer procurar no máximo 20 níveis, nunca 21.
  - Árvore AVL
    - O número máximo de níveis a serem procurados aumenta apenas para 28.



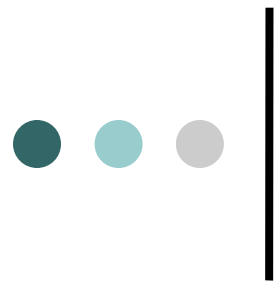
# Árvores AVL

Balanceamento da árvore utilizando os métodos AVL garante obter uma performance aproximada da performance de uma árvore binária ótima, a um custo aceitável na maioria das aplicações que utilizam memória primária.



# Árvores AVL

- Vantagens:
  - Resolve o problema do desbalanceamento, verificado nas árvores de pesquisa binárias
  - A árvore é balanceada e mantém o máximo de acessos na  $O(\log_2 N)$
  - O tempo de busca/inserção de uma chave é da ordem de  $O(\log_2 N)$ .
- Desvantagens:
  - Para 68.585.259.008 chaves vamos precisar 36 seeks a disco
    - $\log_2 68.585.259.008 = 35,997...$
  - Mais que 4 acessos em disco é inaceitável para achar uma chave.



# Próxima aula...

- Árvores Binárias Paginadas