# Splaying Tree (Árvore espalhada)

Estrutura de Dados II

Jairo Francisco de Souza

# Introdução

- Inventada por Adelson Velskii e Landis - 1962.
- Também chamada de Árvores Auto-Ajustadas ou Árvore de Afunilamento.
- É um tipo de Árvore Binária de Pesquisa (BST)
  - máximo 2 filhos: TE e TD;
  - todos nós à esquerda contem subárvores com valores menores ao nó raiz da subárvore;
  - todos nós à direita contem valores maiores ao nó raiz da subárvore;

# Introdução

- Árvores mais simples que AVL:
  - não forçam o equilíbrio;
  - não mantém informação de altura.
- É uma árvore auto-ajustável, alterações tendem ao equilíbrio.
- É utilizada para aplicações específicas, onde se realizam uma seqüência de operações em um universo ordenado.
- Possui três operações básicas: pesquisa, inserção e remoção.
- Em todas operações a árvore faz SPLAY.

# Introdução

- Permite pior caso para uma só operação O(n)
- Garante pior caso amortizado O(logn)
  - Uma sequência qualquer de m operações demora, no pior caso O(m log n)
  - reestruturar a árvore para impedir repetição de operações O(n)

# Espalhamento

- O que é fazer SPLAY?
  - É trazer um elemento X para a raiz da árvore (BTT-Bring To Top), utilizando sucessivas rotações e tantas quanto necessárias.

- Objetivos:
  - Minimizar o número de acessos para achar a chave requerida.
  - Otimizar a eficiência das operações, através da freqüência com que cada nó é acessado, mantendo estes nós na parte superior da árvore.

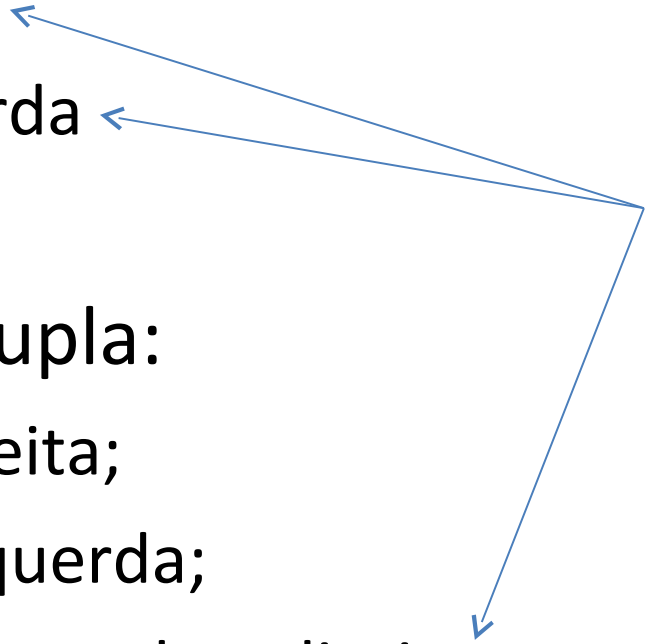
# Espalhamento

- Torna mais acessível o que é mais usado;
  - Ajusta a estrutura da árvore à frequência de acesso aos dados;
  - Junto à raiz estão os elementos
    - mais usados
    - mais recentes
  - Os elementos mais inativos ficam mais “longe” da raiz;
- O recurso é implementado por meio de rotações nos nós.

# Rotações

Rotação Simples:

- Zig direita
- Zig esquerda

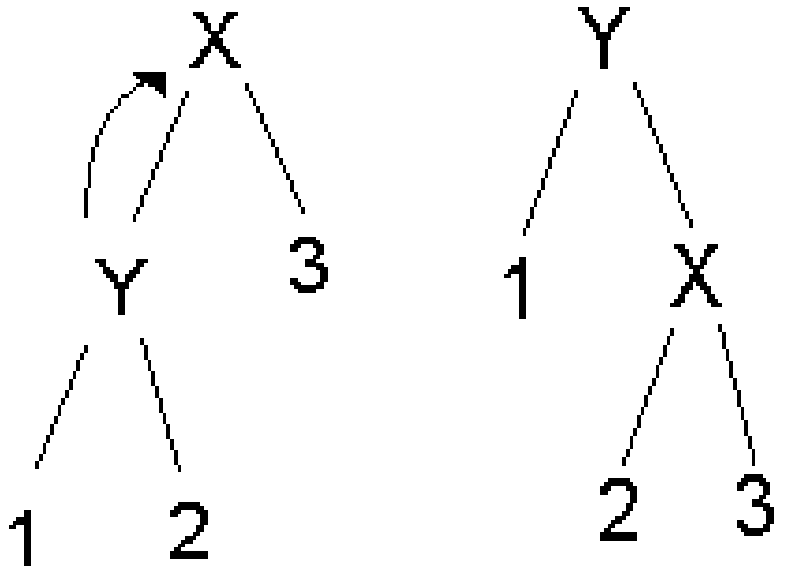



Rotação Dupla:

- Zig-zig direita;
- Zig-zig esquerda;
- Zig-zag esquerda e direita.

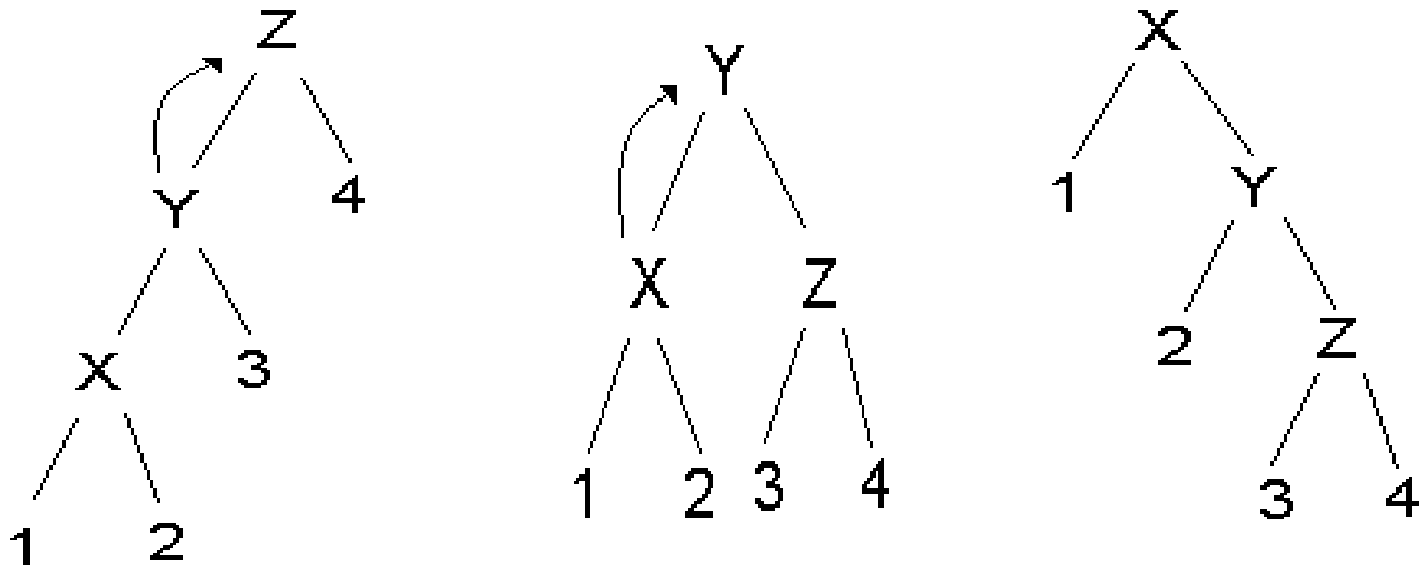
# Rotação Zig

- Nesta rotação, o filho direito do elemento y, ficará o filho esquerdo do elemento x, que era pai de y.
- É a rotação de um nó sobre seu pai, permanecendo a árvore com a mesma altura.



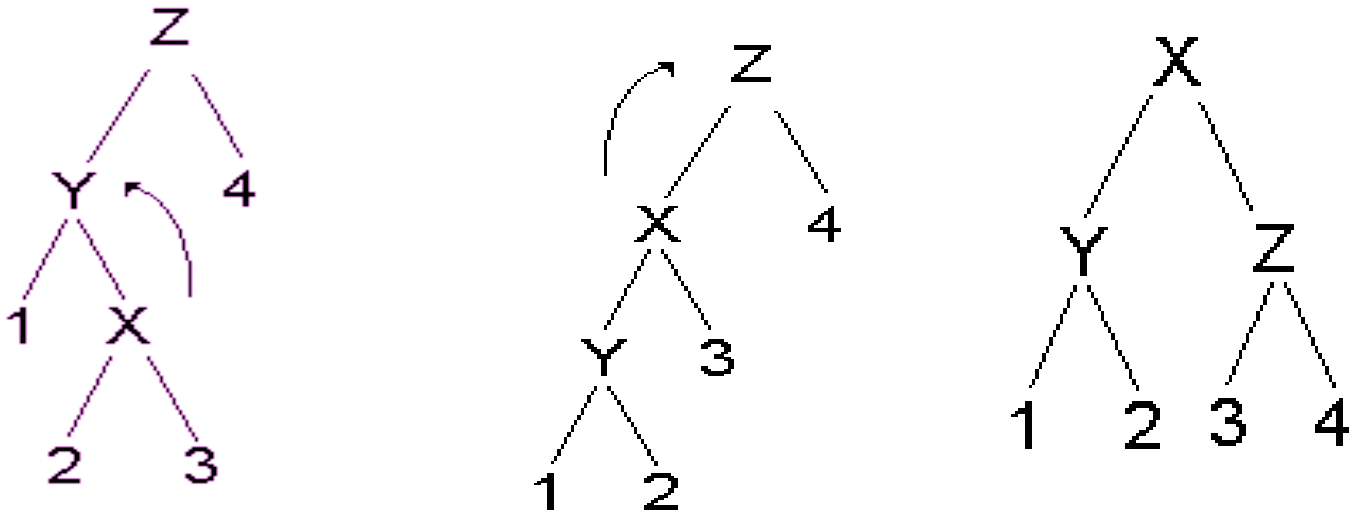
# Rotação Zig-Zig

- Para fazer o zig-zig de x, primeiro temos que fazer o zig do pai de x (que é y).
- Depois, fazemos o zig de x.
- Conclusão: no fundo é feito zig (y) e zig (x), respectivamente.

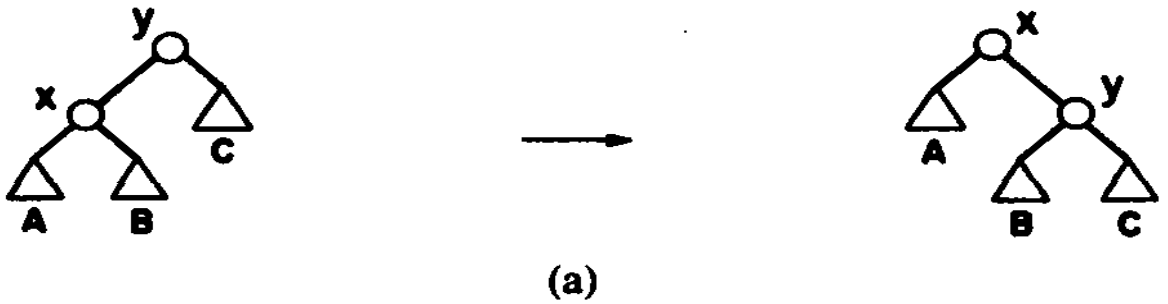


# Rotação Zig-Zag

- Ao contrário de ZIG-ZIG, primeiro devemos fazer o ZIG de X com o pai de X (que é o y).
- Depois fazer o ZIG de X com avô de X (que é o Z).
- Conclusão: No fundo corresponde a fazer ZIG (X) e ZIG (X).



a) ZIG: rotação simples.

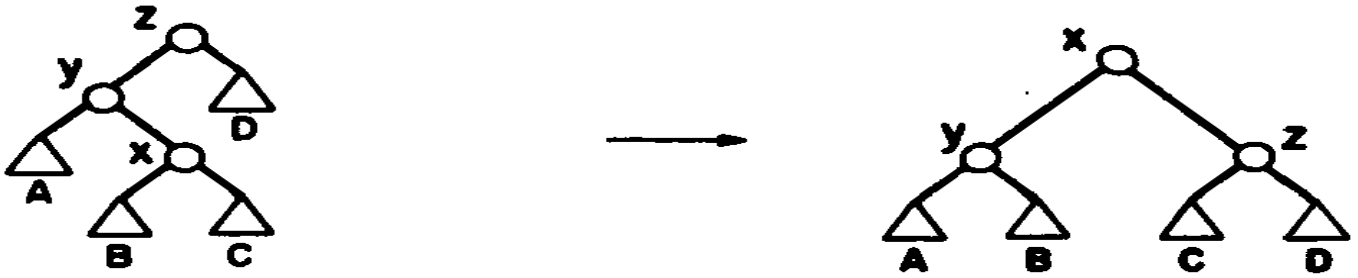



(a)

(b) Zig-zig: duas rotações simples


(b)

(c) Zig-zag: rotação dupla.

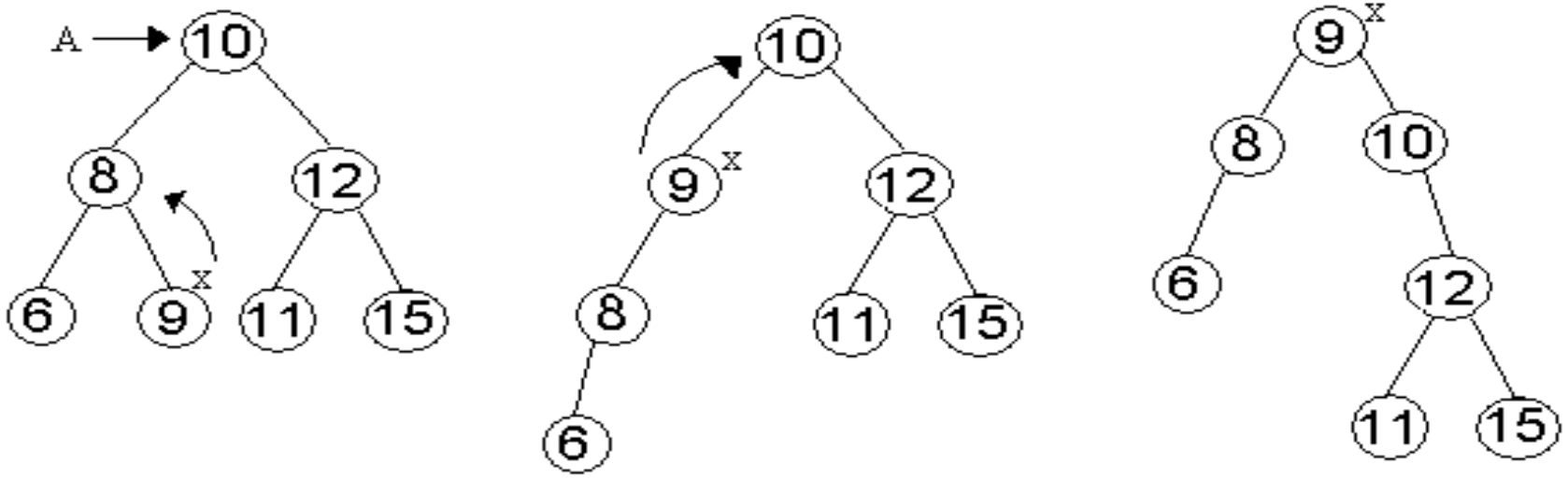



(c)

# Exemplo de Pesquisa(9, A):

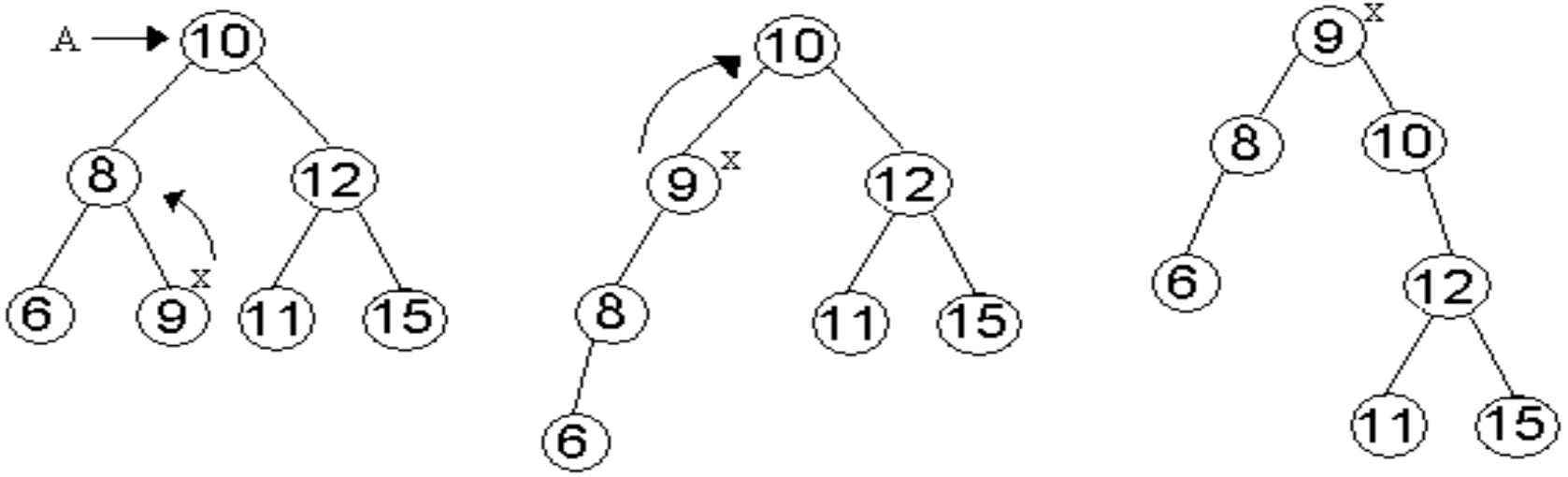
1) Pesquisa tipo BST até encontrar o nó
2) Se nó existe então fazer SPLAY do x (neste caso 9).

   Caso não exista, fazer SPLAY do último nodo não nulo encontrado na busca (elemento menor ou maior, mais próximo de 9.
2) Neste caso, para fazer SPLAY, utilizar o ZIG-ZAG(x).
3) Árvore final com o x na raiz.



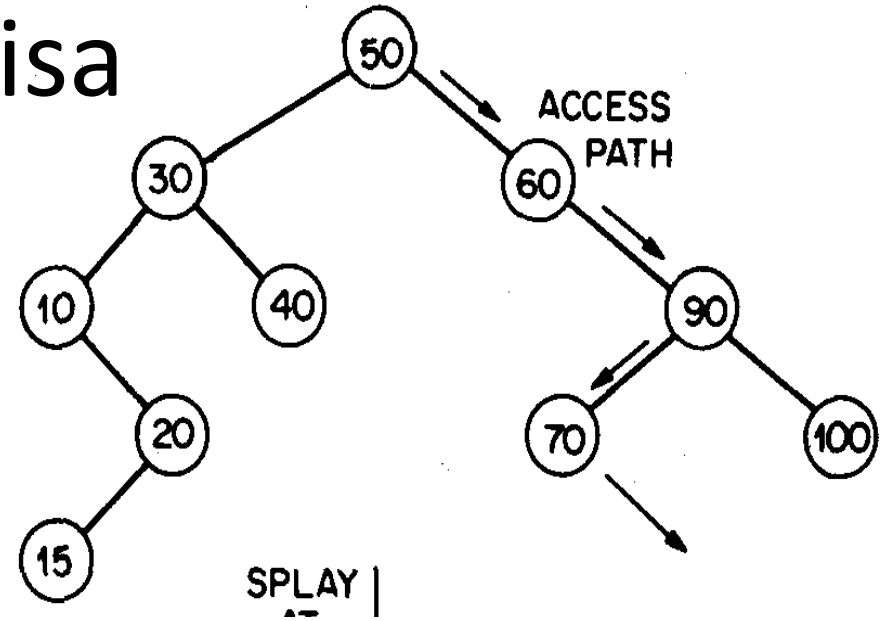
# Exemplo de Pesquisa(x, A):

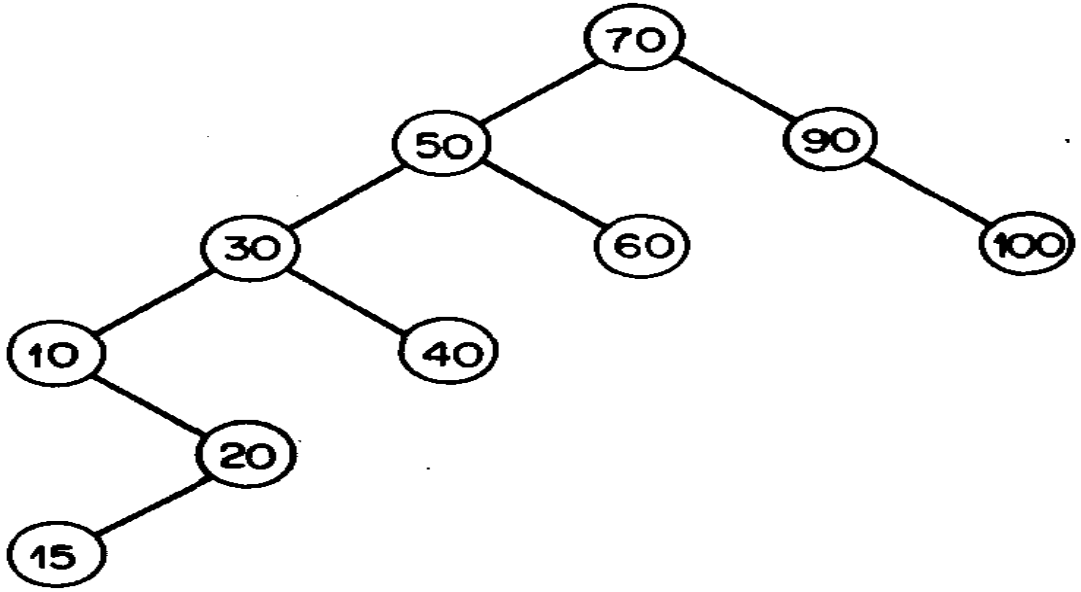
**Início**
  **Faz percurso BST até encontrar nó.**
  **Se existe o elemento x na árvore A**
    **Faz SPLAY do elemento x**
  **Senão**
    **Faz SPLAY do sucessor esquerdo ou direito mais próximo de x**
**Fim**



# Exemplo de Pesquisa

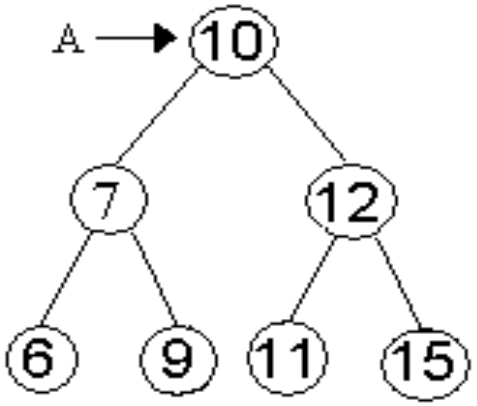
- Pesquisar o valor 80 (não existe na árvore):
- Como a pesquisa parou em 70, é este o elemento que vai para a raiz.
- Se a primeira árvore não tivesse o nodo com valor 70, a pesquisa pararia em 90 e o valor 90 iria para a

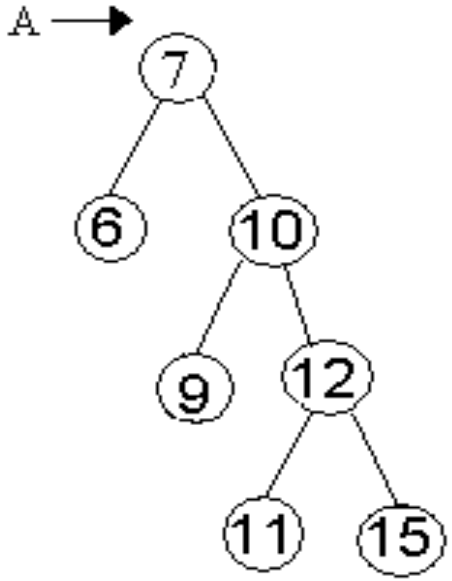


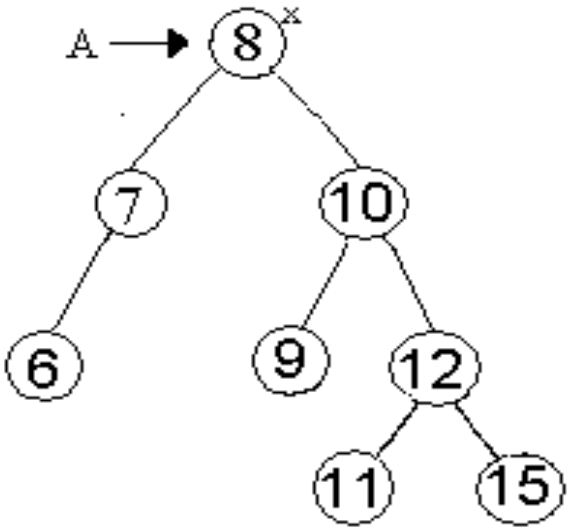



# Exemplo de Inserção(8, A)

1) SPLAY(8).
2) 7 vai para a raiz (primeiro elemento menor que 8).
3) Inserir 8, 7 ficará o seu filho esquerdo, e 10 o seu filho direito.



# Exemplo de Inserção(x, A)

OBS: Essa abordagem é interessante caso queira garantir que valores próximos a *x* fiquem perto da raiz.

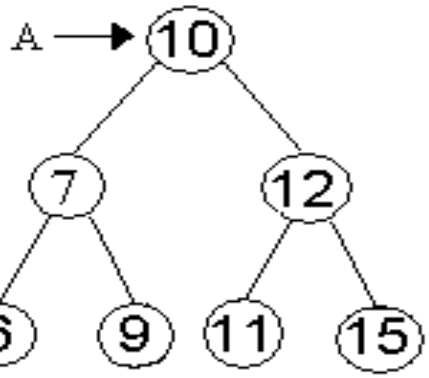
**Início**

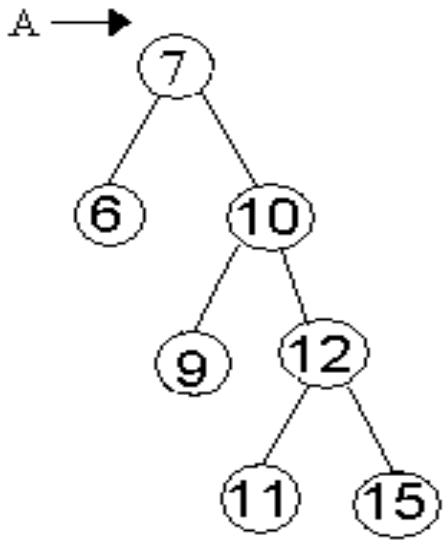
**Faz SPLAY do elemento x .**

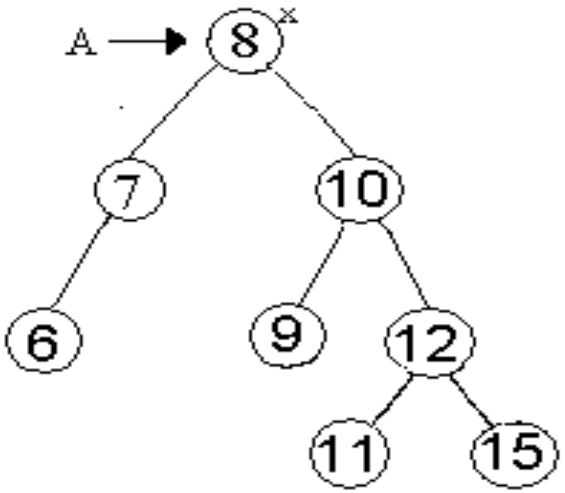
**Como não existe, o elemento menor mais próximo de x, fica na raiz.**

**Agora é só inserir x, que passa a ser a raiz da árvore.**

**Fim**

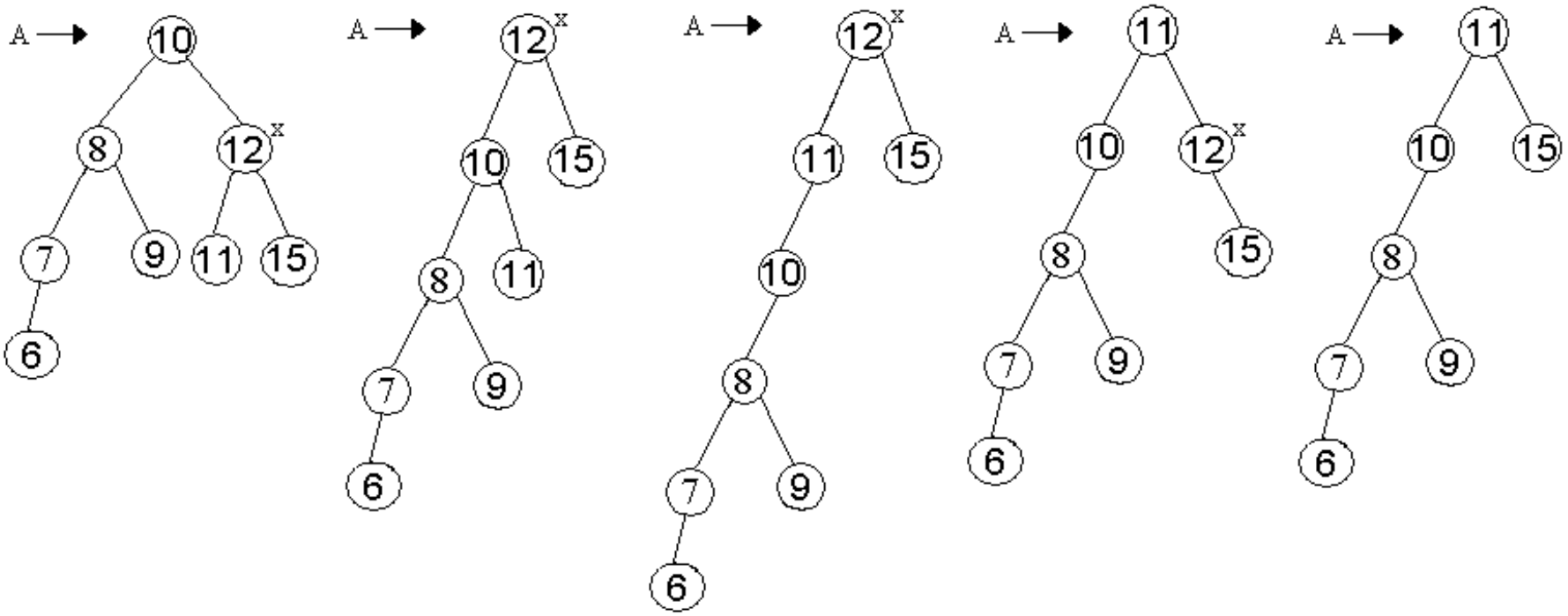


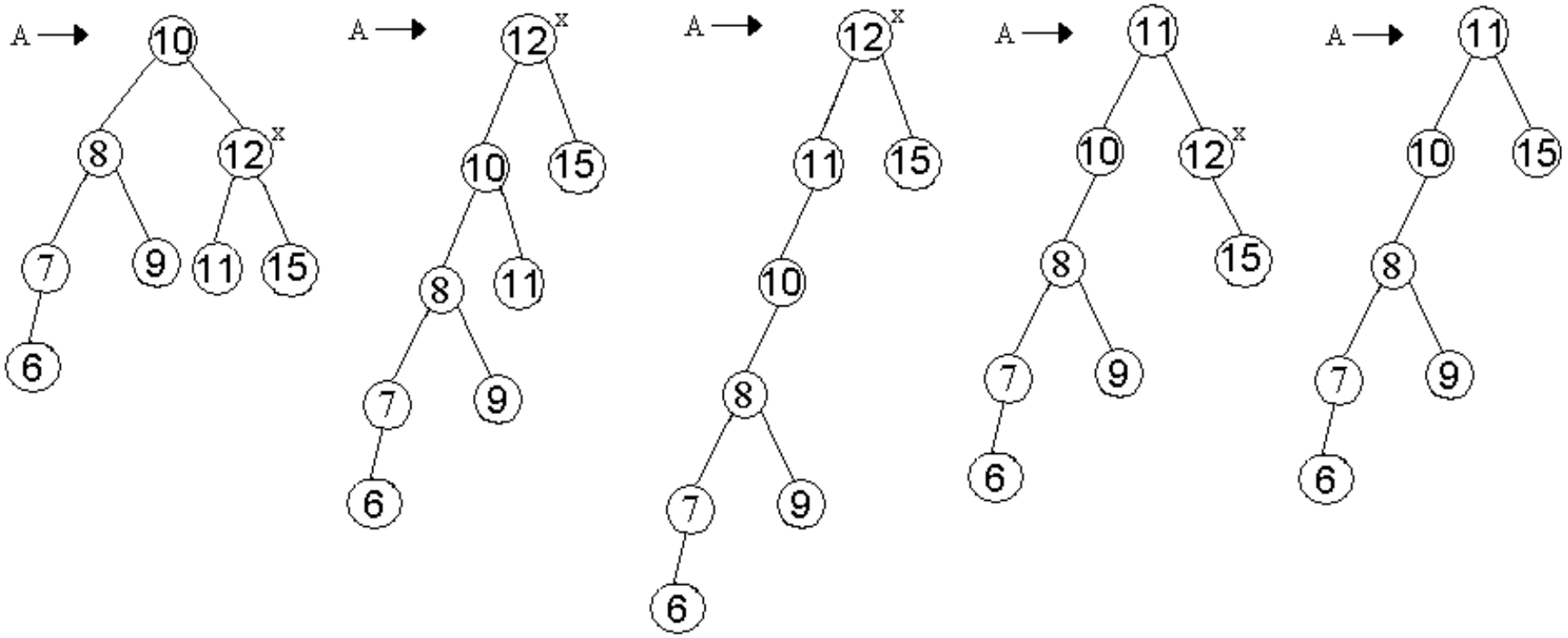
# Exemplo de Remoção(12, A)

1) Pesquisa tipo BST.
2) SPLAY (12,A).
3) SPLAY (12,A') onde A' é a subárvore esquerda do 12.
4) O elemento menor mais próximo de 12 vai para a raiz e sem filho esquerdo, podemos então remover.
5) Eliminar o 12 e ligar o pai de x (12) com seu filho direito (15).



# Exemplo de Remoção(x, A)

**Início**
**Se existe o elemento x na árvore A.**
**Faz SPLAY do elemento x.**
**Faz SPLAY do elemento x, na sua subárvore esquerda.**
**Traz o elemento menor mais próximo de x para a raiz.**
**Remove o x.**
**Senão**
**Faz SPLAY do elemento menor mais próximo de x.**
**Fim**



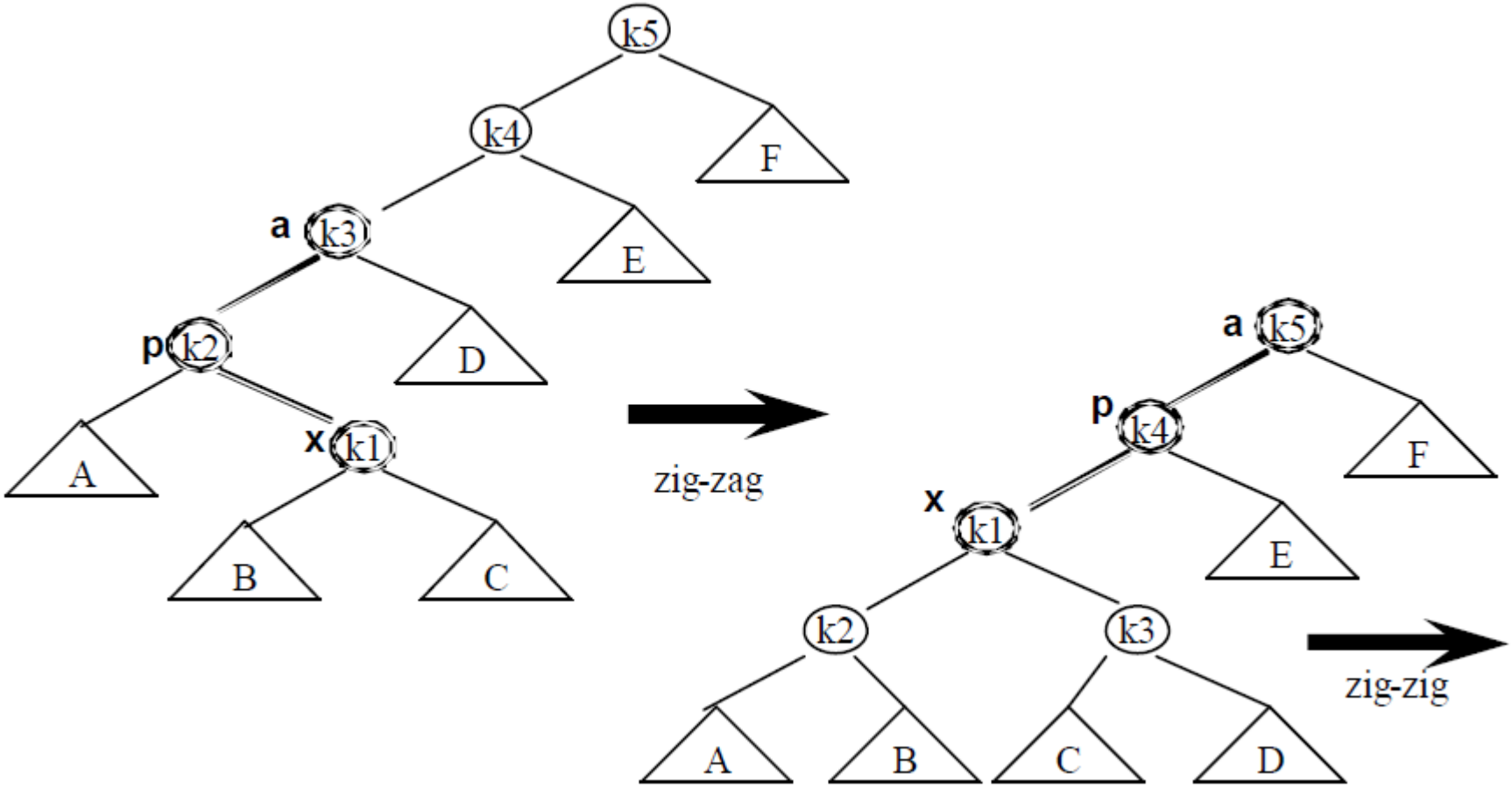
# Como fazer o espalhamento?!

- Duas abordagens
  - Bottom-up
  - Top-down

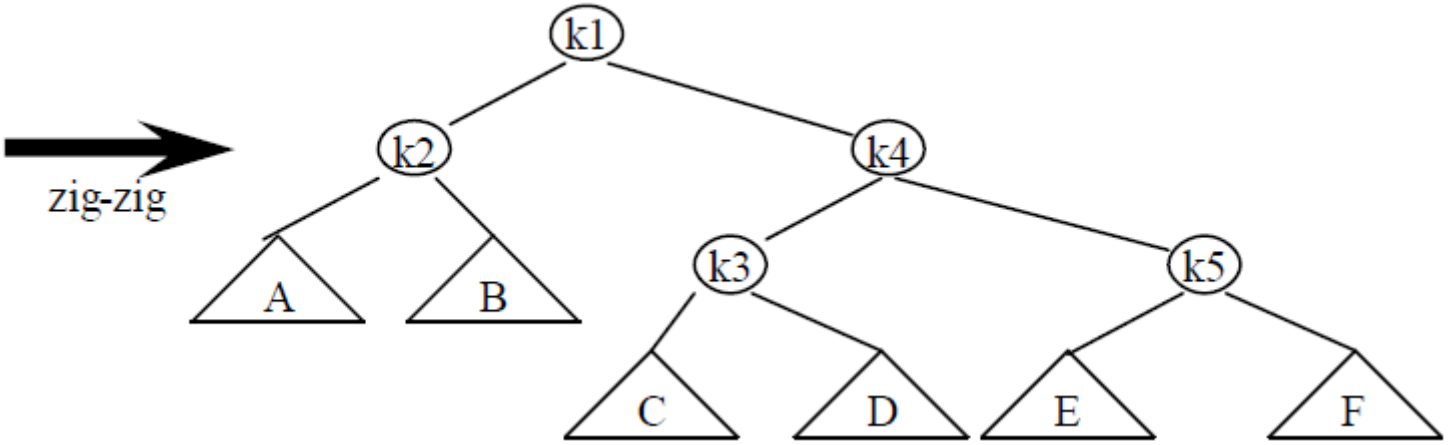
# Espalhamento bottom-up

- rotações ascendentes desde o nó x acessado até à raiz
- se p (pai de x) é a raiz: rotação simples (zig)
- senão, existe um a (avô de x) e
  - se x é filho direito (esquerdo) de p e p filho direito (esquerdo) de a faz rotação zig-zig
    - Rodar primeiro o pai e avô, depois o filho
  - se x é filho direito (esquerdo) de p e p filho esquerdo (direito ) de a faz rotação zig-zag
    - Rodar pai e filho, depois avô

# Espalhamento em k1



# Espalhamento em k1

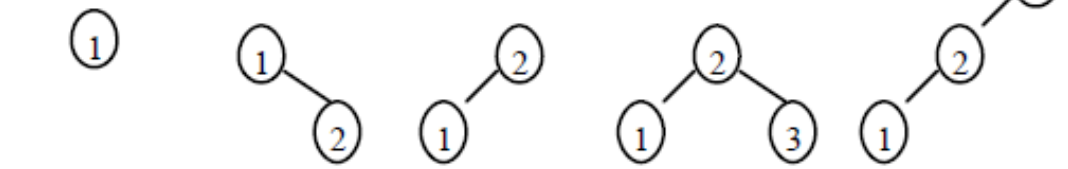



- Fazer uma pesquisa de k1 extrai a respectiva informação e tem como efeito lateral reestruturar a árvore, tornando-a mais equilibrada
- Não só o nó k1 veio para raiz como os que estavam no seu caminho ficaram mais perto dela

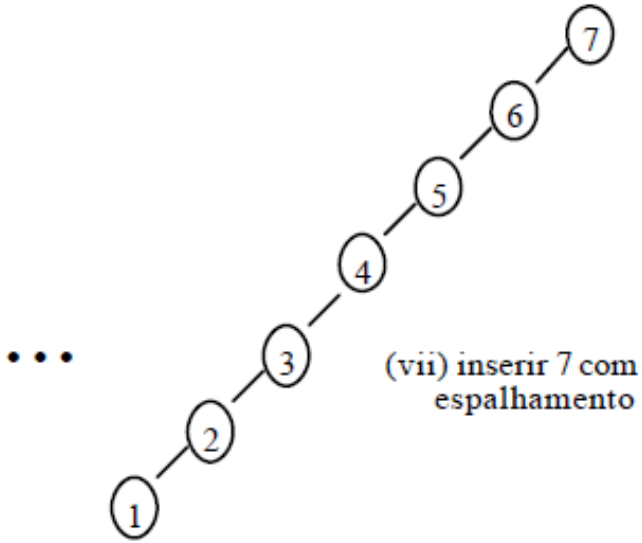
# Exemplo

OBS: Nesse *slide*, está sendo realizado o *splay* **depois** da inserção, somente para ilustração.

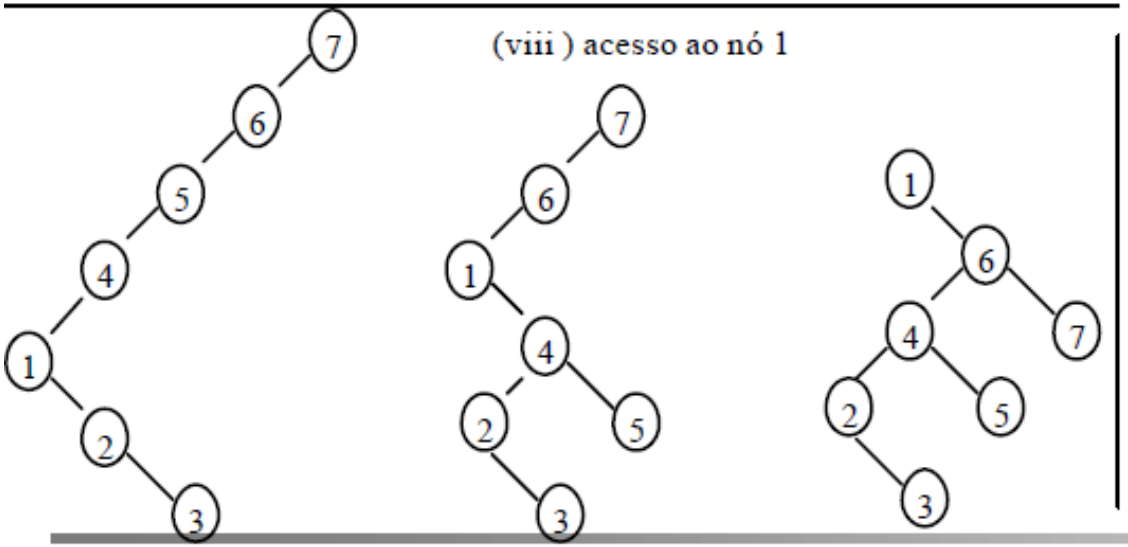
- começar com lista vazia e inserir sucessivamente nós com chave de 1 a 7
- consultar toda a árvore pela mesma ordem

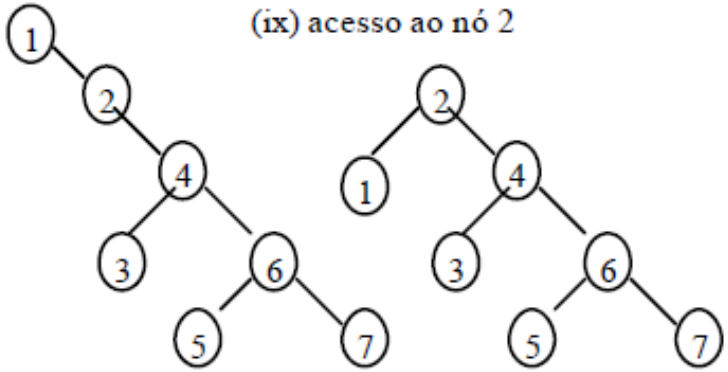



(i) inserir 1 (ii) inserir 2 com espalhamento (iii) inserir 3 com espalhamento




• • • (vii) inserir 7 com espalhamento



# Análise do exemplo

- Inserções
  - cada inserção é feita em tempo constante (insere filho à direita da raíz e faz uma rotação)
  - total das n inserções é O(n)
  - a árvore resultante é muito ruim (lista) mas o processo encontra-se "adiantado“ relativamente a n * O(log n)
- Acessos
  - acesso ao nó 1 é O(n) (nº de rotações relacionado com distância à raíz) – acesso muito ruim que faz reduzir a vantagem acumulada
  - acesso ao nó 2 já é O(n/2)
  - globalmente: calcula-se tempo amortizado
- Conclusão
  - consegue-se sempre, para as primeiras m operações, manter o tempo amortizado, mesmo no pior caso, abaixo de O(m log n)

# Exercício

- Executar a sequência de operações em uma árvore splay tree utilizando a abordagem bottom-up:
  - Inserir 40, 30, 15, 50, 45, 13, 80, 71, 20
  - Remover 50
  - Remover 30
  - Remover 13
  - Inserir 50
  - Remover 71

# Aplicação

- Consultas bancárias
  - Ao fazer uma consulta, o registro de um cliente será levado para o topo da árvore, diminuindo o tempo de acesso para as próximas consultas. Assim, clientes que fazem consultas freqüentemente, terão acesso mais rápido. Os registros de clientes que não utilizam estes serviços, por sua vez, ficarão nos níveis mais inferiores da árvore.
- Sistemas de Arquivos
  - Microsoft Windows utiliza na sua indexação de arquivos por árvore Splay.

# Aplicação

- Hospital
  - Registro de pacientes recentes vai para a raiz no momento da internação e permanece por algum tempo. Vai afundando, caso não seja acessado.

- Particularmente útil na implementação de caches.
  - Proxy Squid utiliza árvores Splay para manipular o cache interno de páginas Web.

# Considerações finais

- Árvores Splay podem ficar desequilibradas, mas garantem uma complexidade O(log n) ao longo do tempo de utilização (complexidade amortizada).
- Ajusta a árvore à freqüência de acesso aos dados.
  - Junto à raiz estão os elementos mais usados / mais recentes (mais disponíveis). Os mais inativos ficam mais longe da raiz.
- É importante notar que em um acesso uniforme, a performance da Árvore Splay será considerada pior que em outro tipo de árvore.
- Há estudos empíricos, que defendem que, em muitos casos reais, se verifica que 90% dos acessos são feitos apenas à 10% dos elementos.