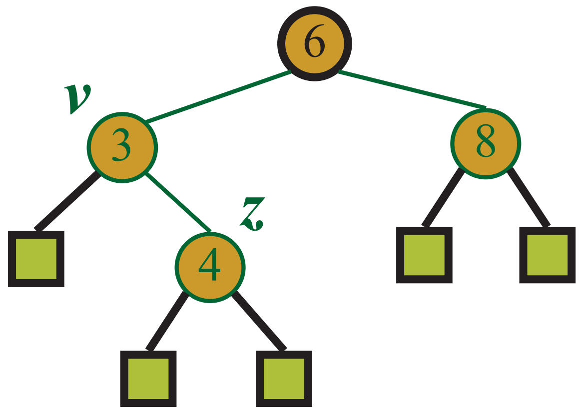
Árvores AVL



Adaptado de David

Árvore AVL

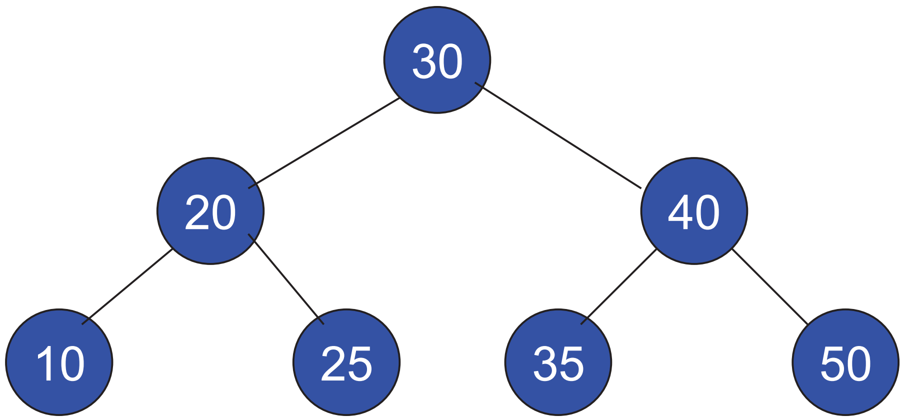
- Árvores binárias balanceadas
  - É fácil perceber que, após várias operações de inserção/remoção, a árvore binária de busca tende a ficar com as alturas de sae e sad muito desequilibradas (desiguais).
  - Pode-se obter inclusive uma árvore degenerada.
  - Nesse caso, o tempo de acesso a um nó pode ser linear em relação ao número de nós da árvore.
  - Para acessar qualquer nó de forma eficiente é necessário árvores binárias balanceadas.

Árvore AVL

- Árvores binárias balanceadas
  - **Árvores balanceadas** são uma classe de árvores binárias de busca tais que não existe desequilíbrio entre as sae e sad.
  - Assim, a altura da árvore é logarítmica em função do número de nós, o que garante um desempenho satisfatório para as operações de busca.
  - Todas as operações realizadas nessas árvores devem garantir que elas permaneçam balanceadas.
  - Ex: árvores AVL e árvores rubro-negra.

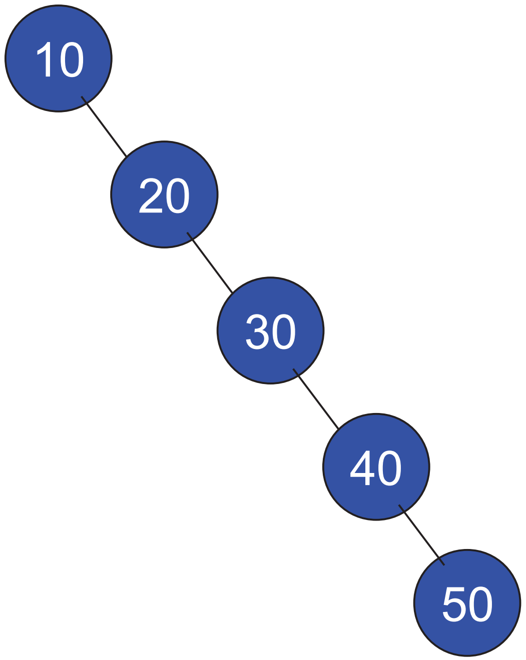
Árvores Binárias Balanceadas e AVL

Inserindo os nós 30, 20, 40, 10, 25, 35 e 50 nesta ordem, teremos:



Árvores Binárias Balanceadas e AVL

Inserindo os nós 10, 20, 30, 40 e 50 nesta ordem, teremos:



Árvores Binárias Balanceadas

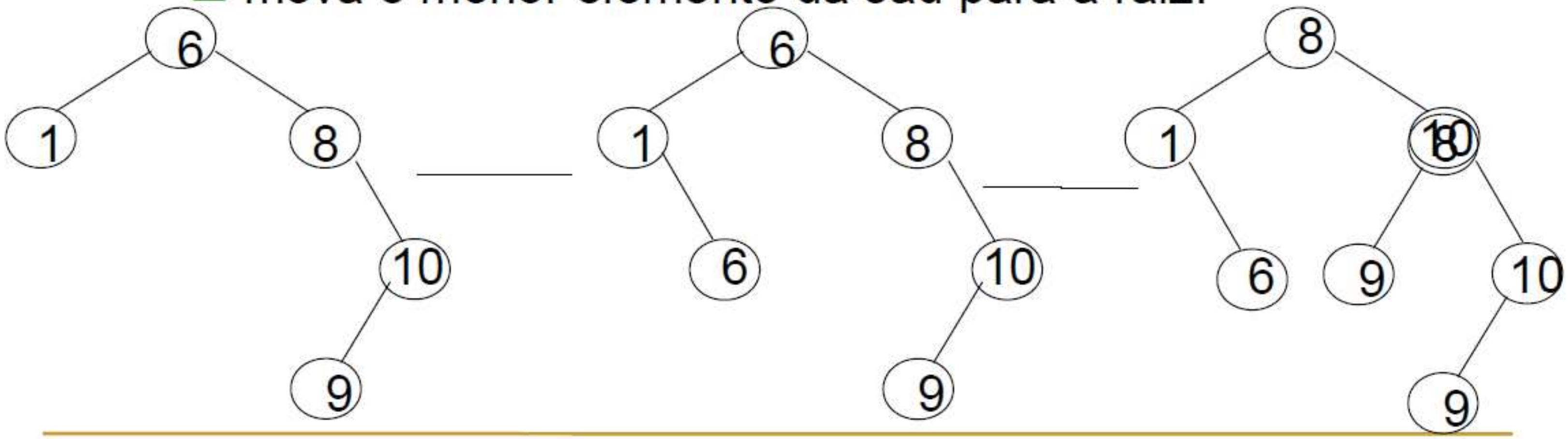
- Existem ordens de inserção de nós que conservam o balanceamento de uma árvore binária.
- Na prática é impossível prever essa ordem ou até alterá-la.
- Algoritmos para balanceamentos.

Árvores Binárias Balanceadas

- A vantagem de uma árvore balanceada com relação a uma degenerada está em sua eficiência.
- Por exemplo: numa árvore binária degenerada de 10.000 nós são necessárias, em média, 5.000 comparações (semelhança com arrays ordenados e listas encadeadas).
- Numa árvore balanceada com o mesmo número de nós essa média reduz-se a 14 comparações.

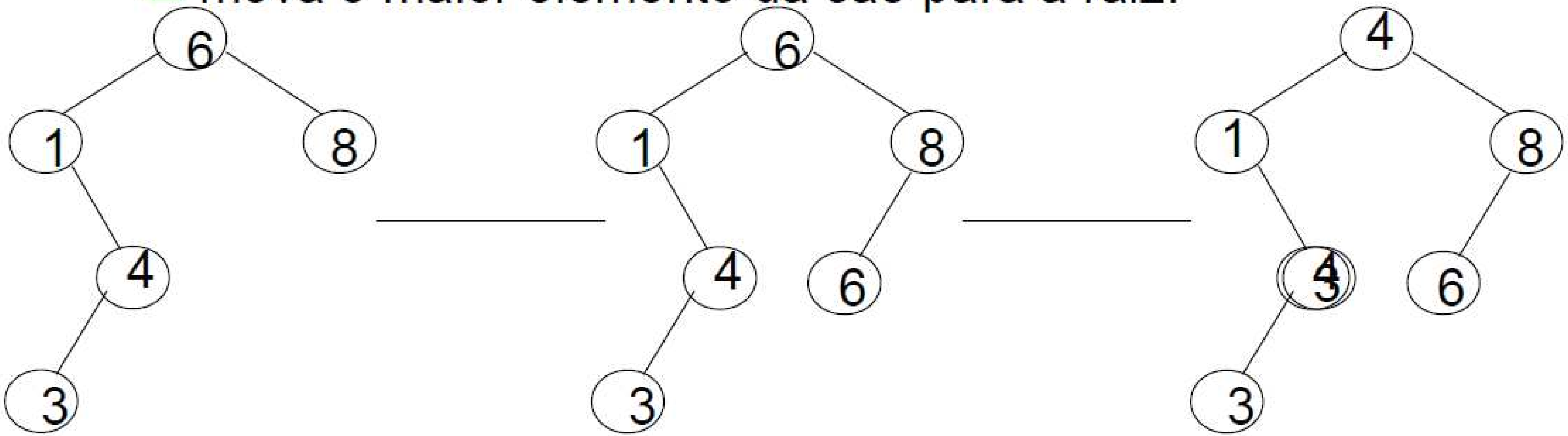
AVL

- Algoritmo para balancear árvore binária de busca:
  quando a sae possui m elementos e a sad possui
  n >= m + 2 elementos
  - mova o valor da raiz para a sae, onde ele se tornará o maior valor;
  - mova o menor elemento da sad para a raiz.



AVL

- Algoritmo para balancear árvore binária de busca:
  quando a sad possui m elementos e a sae possui
  n >= m + 2 elementos
  - mova o valor da raiz para a sad, onde ele se tornará o menor valor;
  - mova o maior elemento da sae para a raiz.



AVL

- Árvores binárias balanceadas

  - Dessa forma, a árvore continua com os elementos na mesma ordem.

  - Esse processo pode ser repetido até que a diferença entre os números de elementos das duas subárvores seja menor ou igual a 1. Naturalmente, o processo deve continuar(recursivamente) com o balanceamento das duas sub-árvores de cada árvore.

  - Um ponto a observar é que remoção do menor (ou maior) elemento de uma árvore é mais simples do que a remoção de um elemento qualquer.

AVL

- Árvores AVL

  - A primeira árvore binária de pesquisa com balanceamento foi proposta por **A**delson-**V**elskii e **L**andis (1962).

  - Uma árvore binária de pesquisa é uma árvore AVL se as alturas das sae e sad de qualquer nó N diferem de no máximo 1.

  - Em outras palavras, se T é uma árvore binária com sae e sad T_L e T_R , então T é uma árvore balanceada se e somente se:
    - T_L e T_R são árvores balanceadas;
    - As alturas h_L e h_R das subárvores T_L e T_R mantém a seguinte relação $| h_L - h_R | <= 1$

AVL

- Essa diferença é chamada de fator de balanceamento (FB).

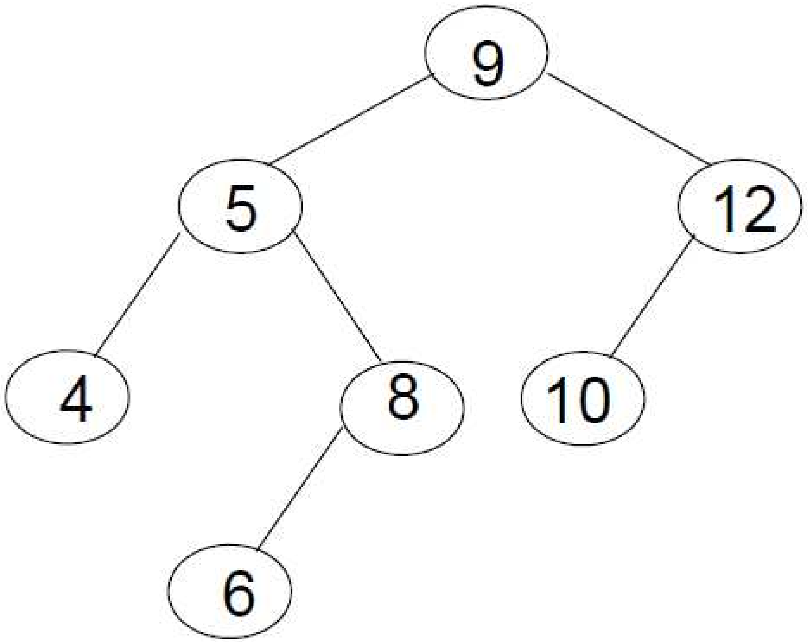
FB = altura da sae de N – altura da sad de N

  - Se FB = 0, sad e sae possuem mesma altura.
  - Se FB < 0, a sad possui altura maior do que sae.
  - Se FB > 0, a sae possui altura maior do que sad.

- Numa árvore AVL, o fator de balanceamento de qualquer nó deve ser -1, 0 ou 1.

AVL

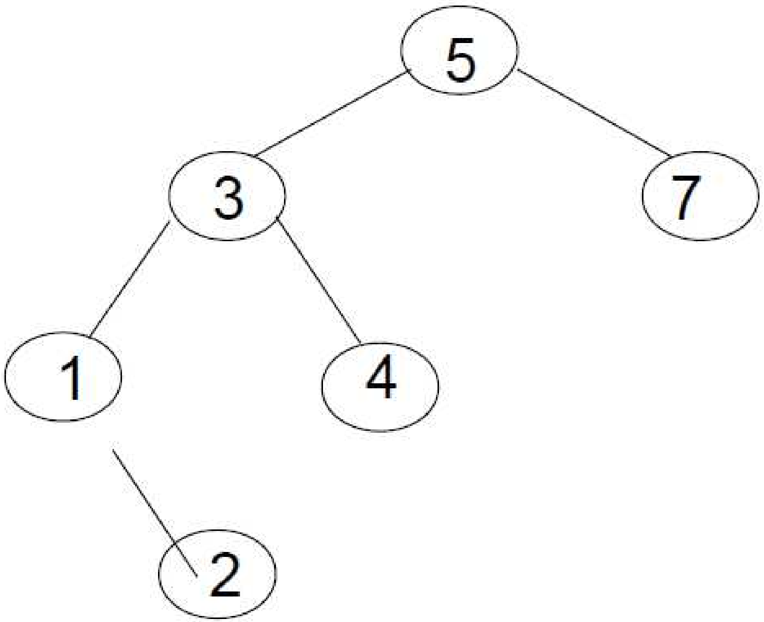
- Árvores AVL



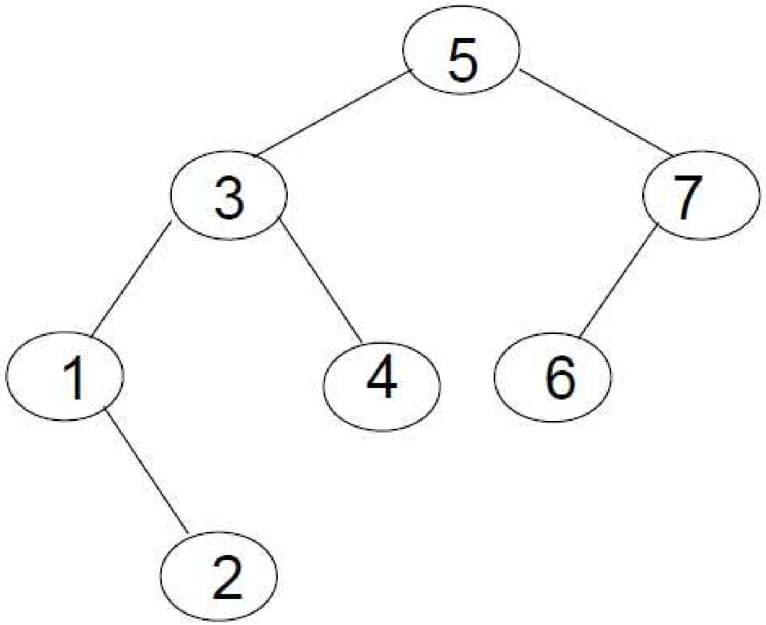
Exemplo de árvore AVL

AVL

- As árvores abaixo são árvores AVL?



Não é AVL



É AVL

Para ser AVL, a árvore deve ser uma árvore binária de busca e possuir a propriedade AVL

AVL

- Operações em Árvores AVL

  - As funções que vamos analisar são:
    - Insere
    - Retira

  - Após essas operações deve-se verificar se a árvore permanece AVL: se é uma árvore binária de busca e se a diferença entre as alturas das sae e sad de cada nó é, no máximo, igual a 1.

  - As outras funções de árvores (inclusive a de busca) são análogas às funções vistas para árvores binárias de busca.

AVL

Inserção numa Árvore AVL

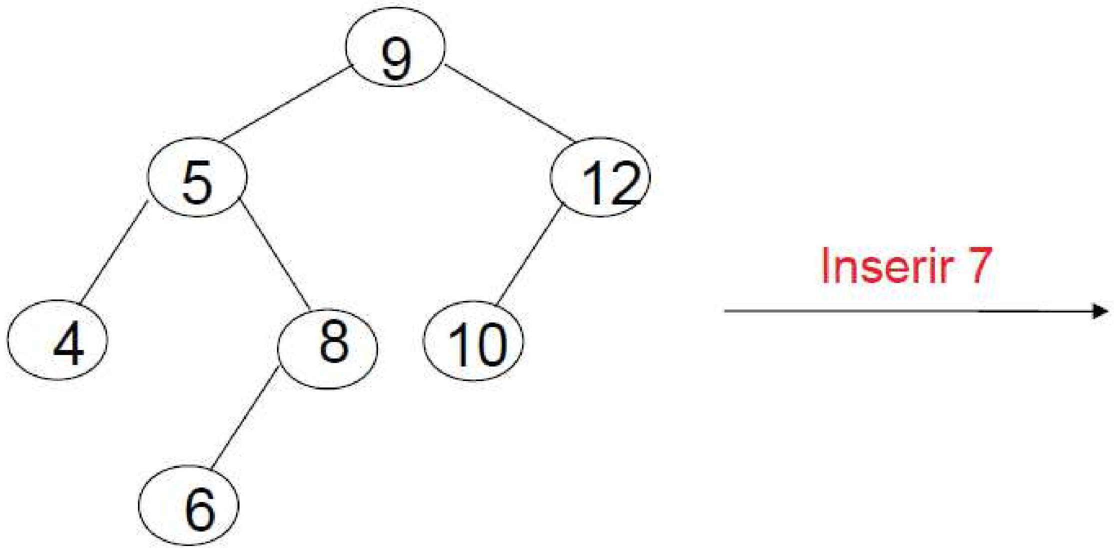
- Um novo nó é inserido exatamente como numa árvore binária de busca (como folha da árvore).

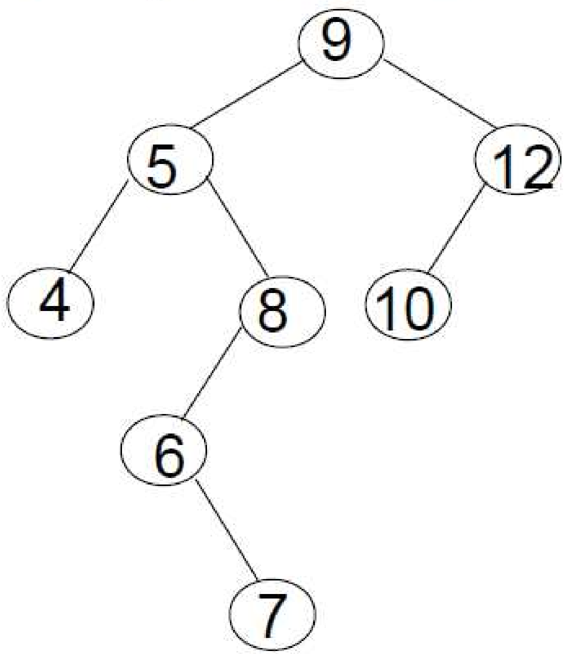
- Se a inserção implicar em |FB| > 1 para algum(s) ancestral (ais) do nó inserido, o desequilíbrio é corrigido através da rotação de nós.

- As rotações de rebalanceamento ocorrem em torno do nó ancestral mais próximo do nó incluído ou excluído tal que seu fator de desbalanceamento seja igual a +2 ou -2.

AVL

- Para que o resultado de uma inserção não tenha a propriedade AVL, é preciso ter pelo menos um nó da árvore inicial tal que uma subárvore tem altura superior à da outra, e que a inserção seja realizada nesta subárvore.

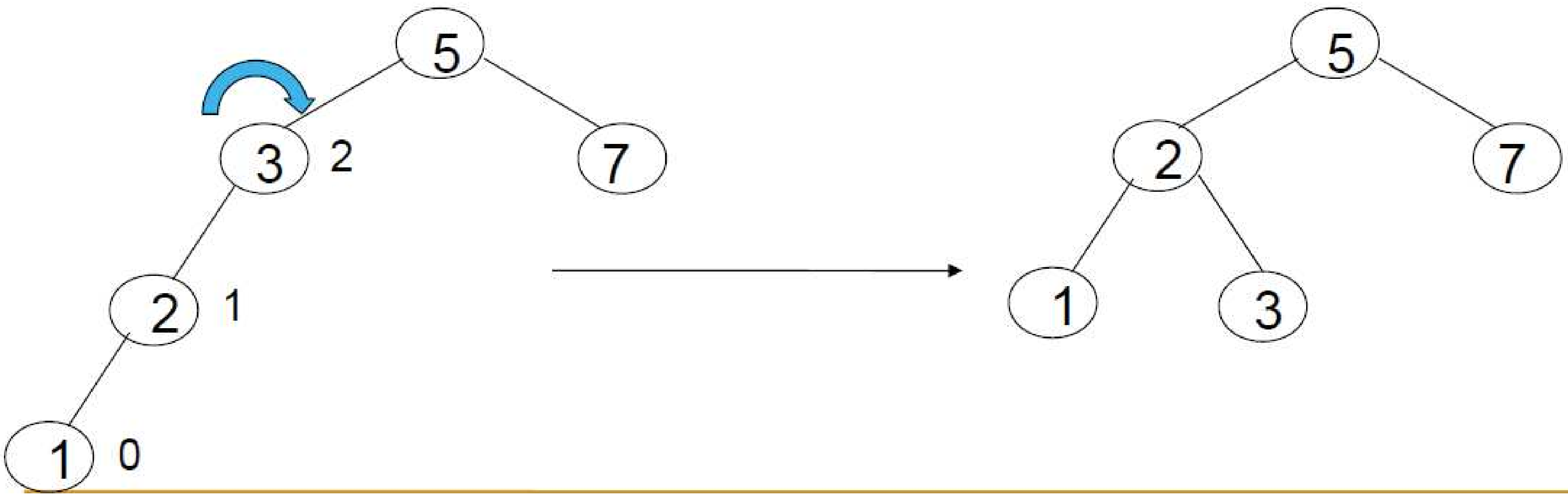




AVL

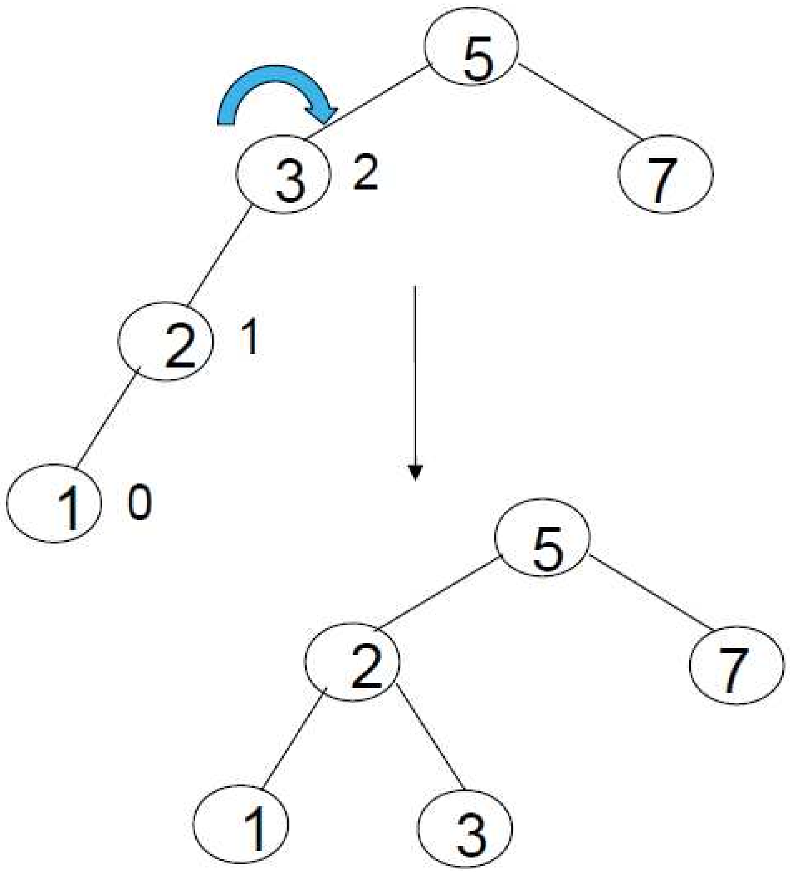
- **Rotações simples:** Pai e Filho têm desequilíbrio para o mesmo lado, ou fatores de balanceamento com o mesmo sinal.

Rotação à direita (RightRotation):



AVL

Rotação à direita (RightRotation):

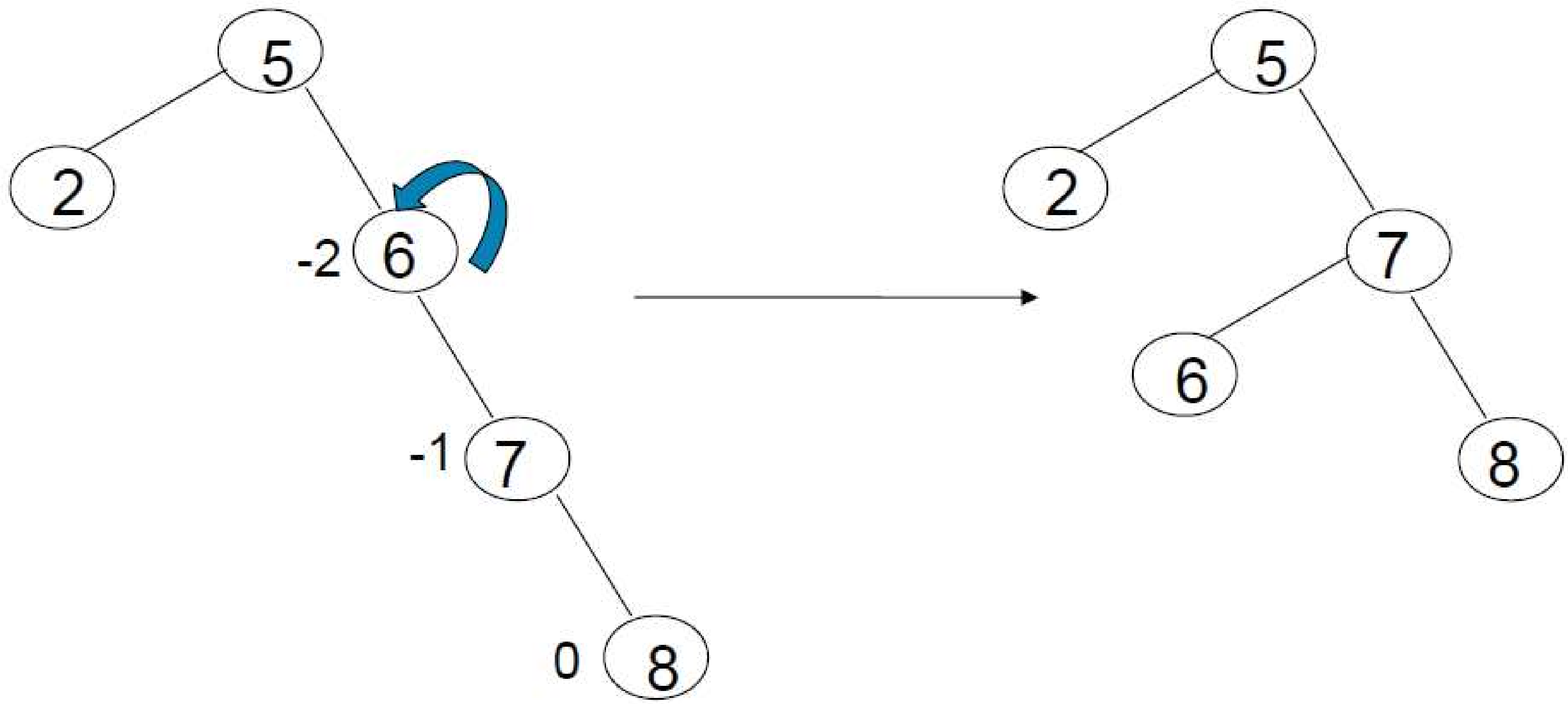


// L é o nó desbalanceado
temp = L;
L = temp -> E //filho esquerdo
temp ->E = L->D //como temp será filho à direita, ele herda os filhos à direita do seu pai.
L->D = temp
// Em seguida, atualiza-se os fatores de balanceamento

AVL

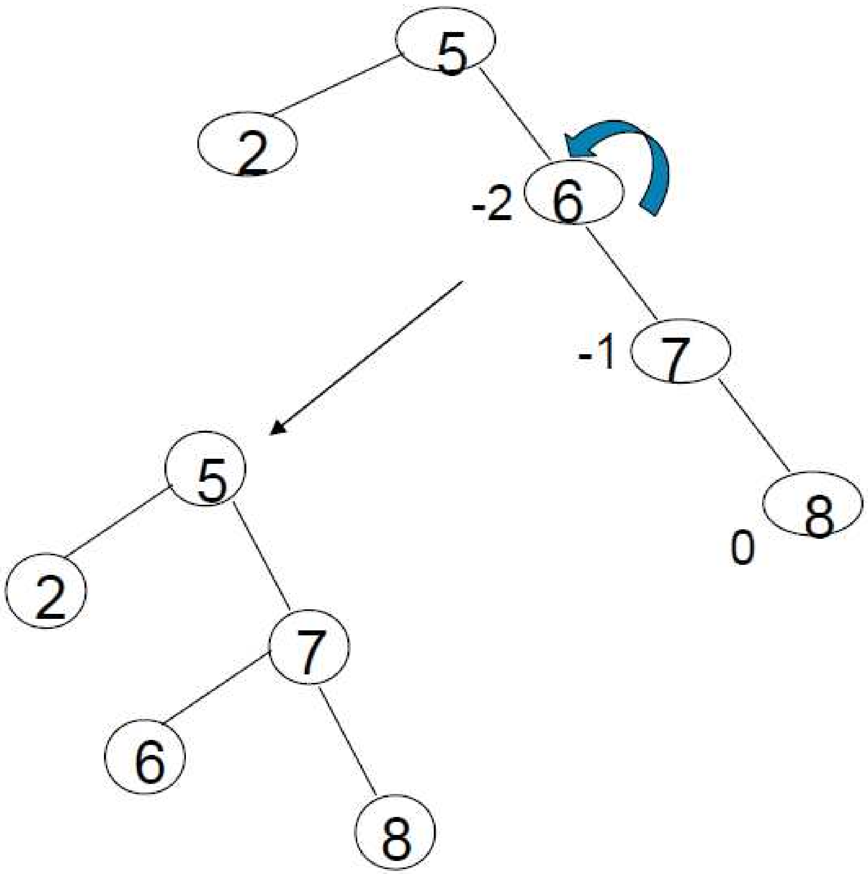
- Inserção numa Árvore AVL

Rotação à esquerda (LeftRotation):



AVL

Rotação à esquerda (LeftRotation):

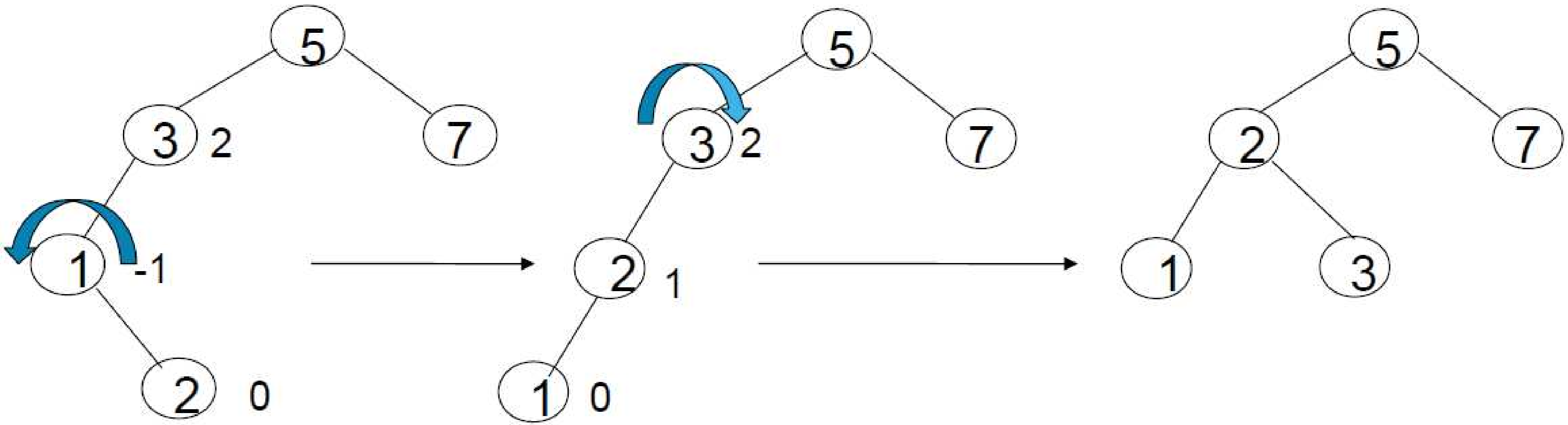


// L é o nó desbalanceado
temp = L;
L = temp -> D   //filho direito
temp ->D = L->E //como temp será filho à esquerda, ele herda os filhos à esquerda do seu pai.
L->E = temp
// Em seguida, atualiza-se os fatores de balanceamento

AVL

- **Rotações duplas (2 rotações simples, uma para cada lado):** Pai e Filho têm desequilíbrio para lados opostos, ou fatores de balanceamento com sinais opostos.

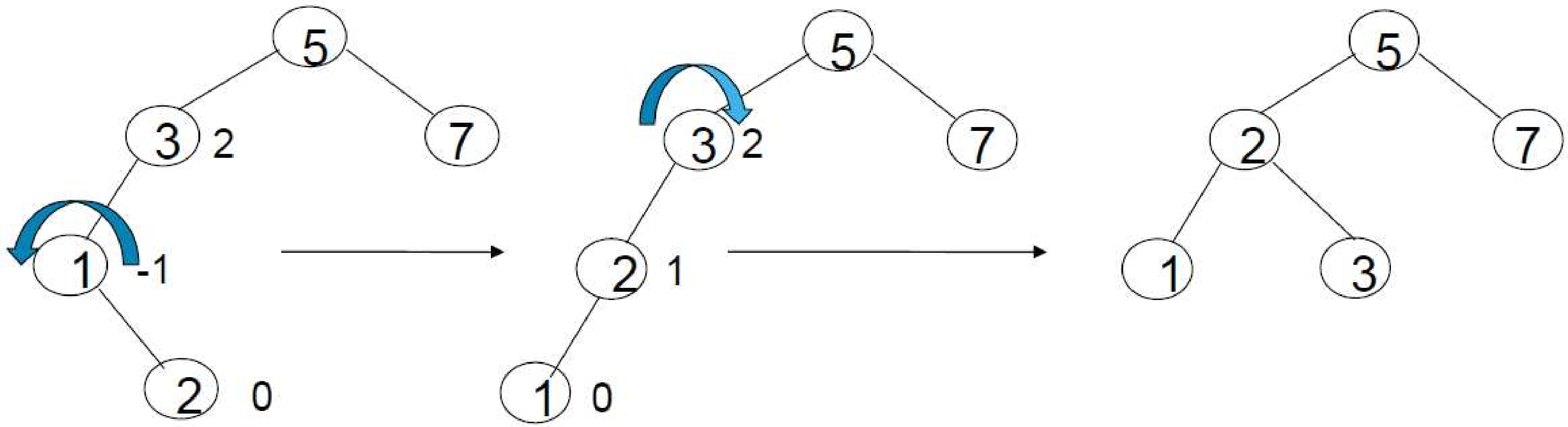
Dupla Rotação à Direita:



AVL

- Inserção numa Árvore AVL

Dupla Rotação à Direita:



//N é o nó desbalanceado

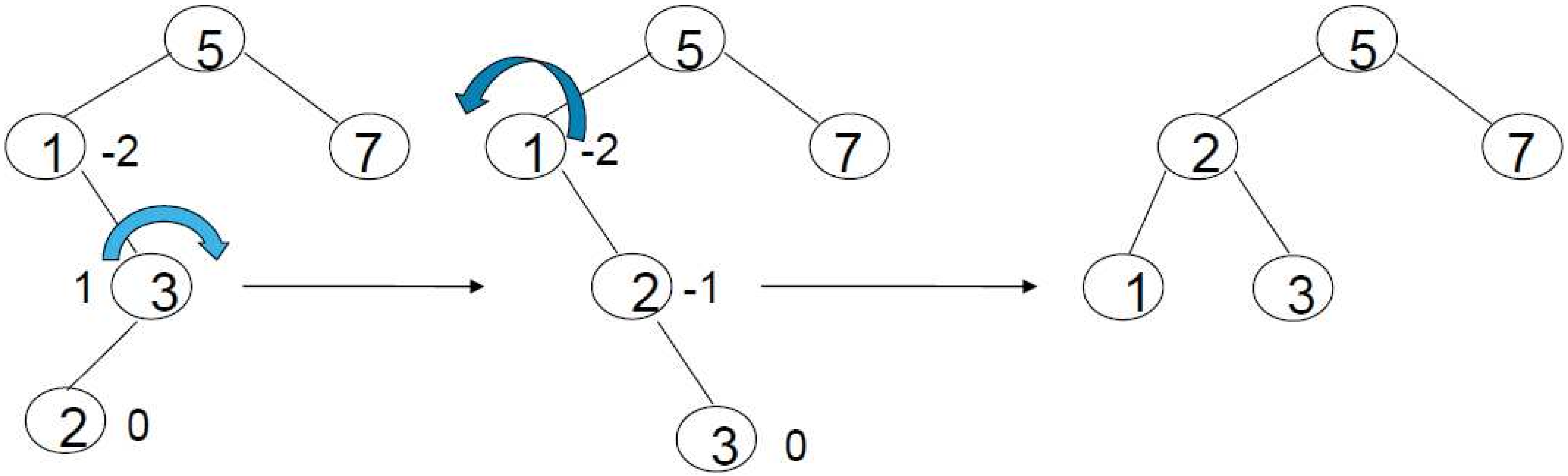
RotEsq(L->E)

RotDir(L)

AVL

- Inserção numa Árvore AVL

Dupla Rotação à Esquerda:



//N é o nó desbalanceado

RotDir(L->D)

RotEsq(L)

AVL

- Inserção numa Árvore AVL

  - Seja A uma árvore AVL, com raiz r e i nós, e c um valor a inserir. O procedimento a seguir é o seguinte:

    - c deve ser inserido da mesma forma que um elemento é inserido em uma árvore de busca binária.
    - Em seguida, deve-se verificar se a inserção do nó desequilibrou alguma subárvore. Caso negativo, o procedimento para. Caso positivo, é preciso reequilibrar a árvore usando a operação de rotação apropriada.

  - Como verificar se a inserção produz um desequilíbrio na árvore?

AVL

Inserção numa Árvore AVL

- Uma solução possível é, para cada nó n, ter um campo que armazena o fator de balanceamento (fatBal) entre as sae e sad.

- Logo, a estrutura de um nó na árvore AVL é:

```
struct arvore {
  int info;
  int fatBal;
  struct arvore* esq;
  struct arvore* dir;
};

typedef struct arvore Arvore;
```

Optamos por fazer todas as funções receberem o endereço da árvore como parâmetro pois algumas delas terão que retornar algum valor (e não poderão retornar á árvore atualizada).

AVL

- Inserção numa Árvore AVL

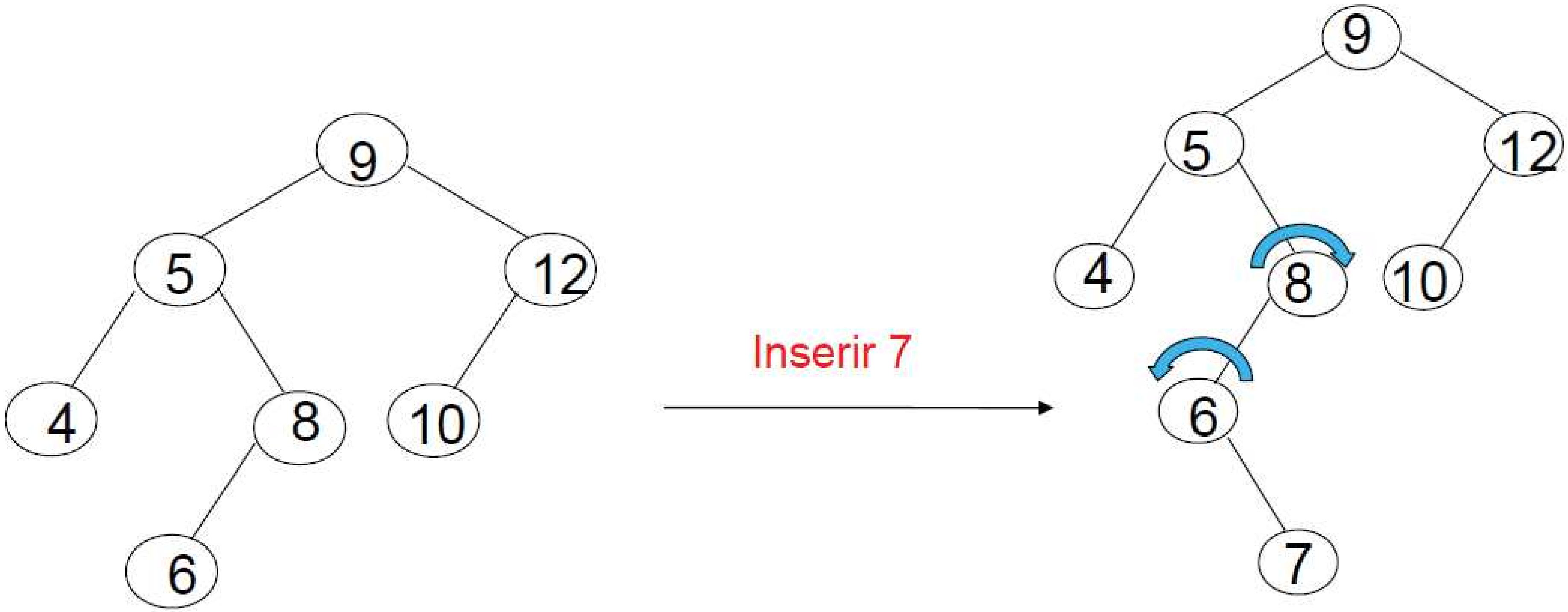
  - Quando um nó é inserido à esquerda de um nó n, e essa inserção aumenta a altura da sae, o FB é incrementado. Caso se torne igual a 2, uma rotação a direita é necessária.

  - Simetricamente, quando um nó é inserido à direita de n, e a altura da sad aumenta, o FB é decrementado. Caso se torne igual a -2, então uma rotação a esquerda é necessária.

  - Os algoritmos de rotação são responsáveis pela atualização dos fatores de balanceamento dos nós.

AVL

- Inserção numa Árvore AVL



Nome da rotação: Rotação dupla à direita

AVL

Exclusão numa Árvore AVL

- Um nó é excluído exatamente como numa árvore binária de busca.

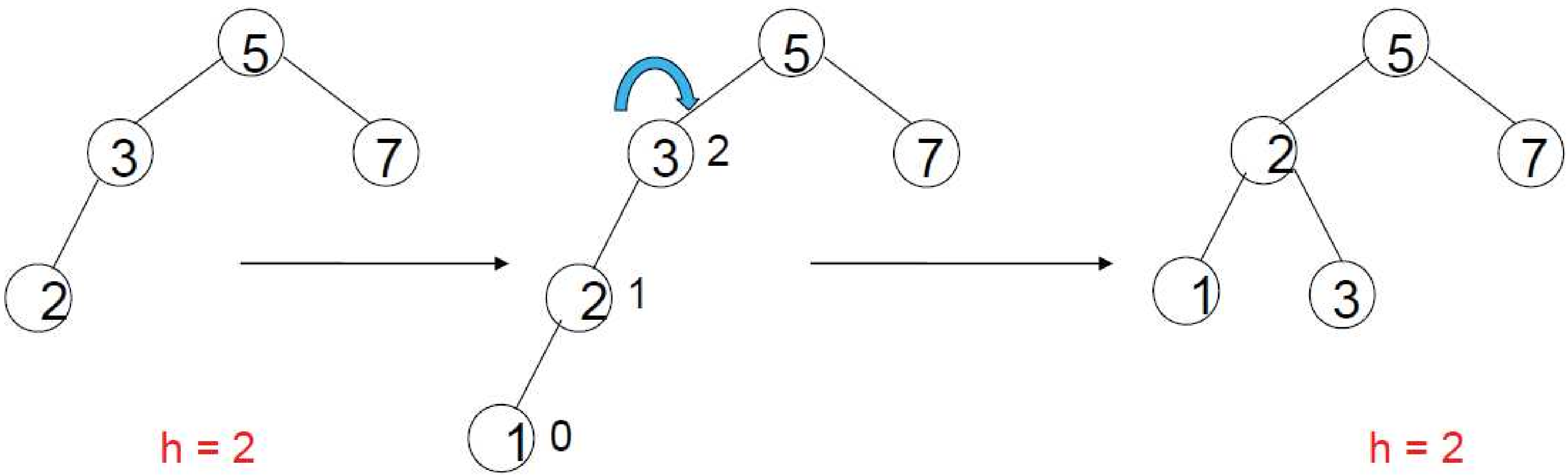
- Excluindo um nó pode ocorrer ou não a redução da sub árvore aonde o mesmo se encontrava. Havendo redução, há necessidade de proceder um dos rebalanceamentos estudados na inserção.

- Porém, ao contrário da inserção, pode ser preciso fazer mais de uma rotação pois a altura da subárvore remanejada pode ser alterada.

AVL

- Exclusão numa Árvore AVL

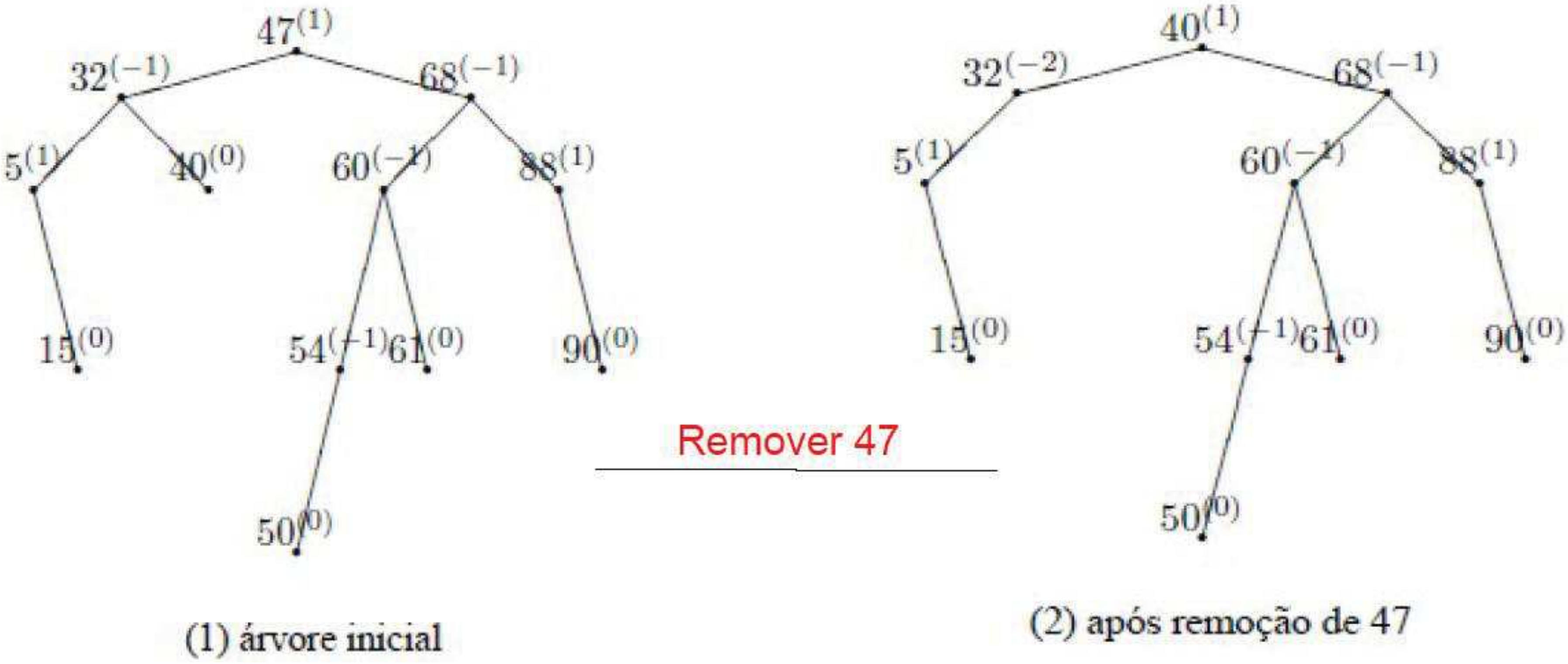
  - Quando uma inserção deve ser seguida de uma rotação, a altura da subárvore remanejada não é alterada.



AVL

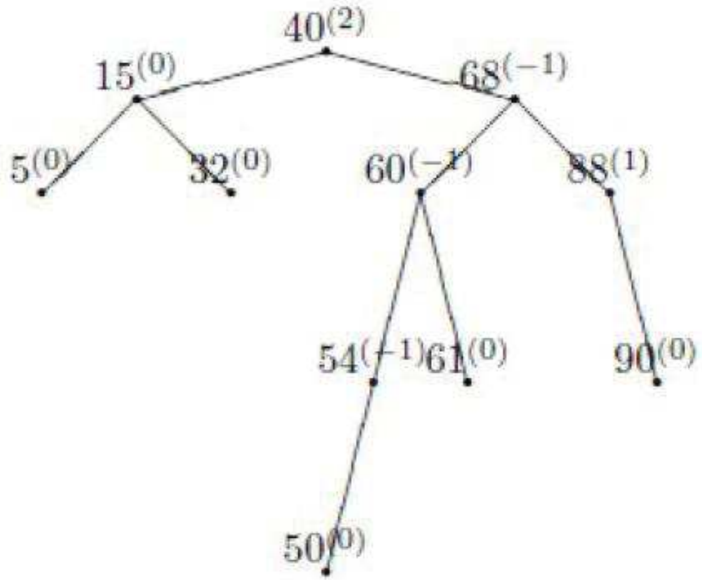
- Exclusão numa Árvore AVL

  - Essa propriedade não é verificada pela operação de remoção.

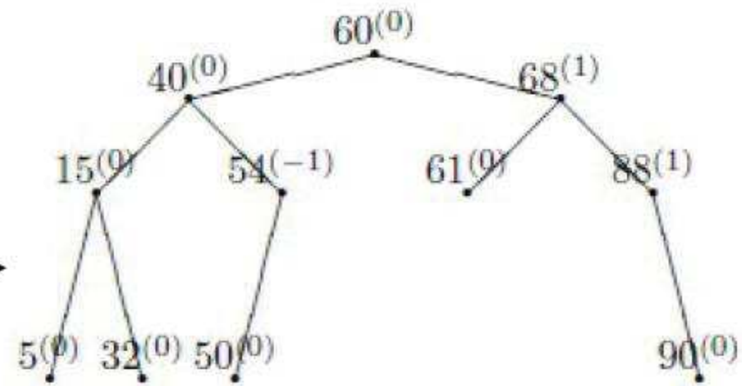


AVL

Exclusão numa Árvore AVL



(2) após rotação dupla a direita de 32



(3) após rotação dupla a esquerda de 40

A altura da subárvore resultante após essa rotação dupla é menor do que a altura da subárvore inicial, e a subárvore em 40 fica desbalanceada.

É necessária a rotação dupla a esquerda para reestabelecer o equilíbrio desta subárvore.

AVL

Exclusão numa Árvore AVL

- Seja A uma árvore AVL com raiz r e um valor v a remover. O procedimento a seguir é o seguinte:
  - Busca o valor v a remover na árvore. Se o valor não é encontrado o algoritmo termina.
  - Encontrado o nó n com o valor v a remover, verifique se ele possui subárvore à esquerda. Se possui, busque seu antecessor na subárvore à esquerda, faça a troca, atualize o fator de balanceamento e faça as rotações necessárias. Caso n não possua subárvore a esquerda, faça a árvore receber o filho da direita e remova o nó n.
  - O último passo é reequilibrar A depois da remoção. Esse passo pode ser realizado da mesma maneira que no algoritmo da inserção (utilizando rotações), mas para todo nó m entre n e a raiz r, tem que verificar se um desequilibro foi inserido, e neste caso operar a rotação correspondente.

AVL

Exclusão numa Árvore AVL

- Quando um nó é removido à esquerda de um nó n, e essa exclusão diminui a altura da sae, o FB é decrementado. Caso se torne igual a -2, uma rotação a esquerda é necessária.

- Simetricamente, quando um nó é removido à direita de n, e a altura da sad diminui, o FB é incrementado. Caso se torne igual a 2, então uma rotação a direita é necessária.

- Os algoritmos de rotação são responsáveis pela atualização dos fatores de balanceamento dos nós.

Complexidade de Tempo para árvores AVL

- uma única reestruturação é O(1)
  - usando uma árvore binária implementada com estrutura ligada
- pesquisa é O(log n)
  - altura de árvore é O(log n), não necesita reestruturação
- inserir é O(log n)
  - busca inicial é O(log n)
  - reestruturação para manter balanceamento é O(log n)
- remove é O(log n)
  - busca inicial é O(log n)
  - reestruturação para manter balanceamento é O(log n)

Aplicações

- Para que servem as Árvores Binárias?

- Exemplos de aplicações:
  - Redes de Comunicação de Dados
    - Envio de pacotes ordenados e/ou redundantes
  - Codificação de Huffman
    - Compressão e Descompressão de arquivos

1) Redes de Comunicação

- A maioria dos protocolos de comunicação fragmenta as mensagens em pacotes que são numerados e enviados através da rede
- Não há garantia da chegada em ordem dos pacotes
- Perdas de pacotes geram novos envios e estes podem causar duplicatas dos mesmos

Reconstrução da Mensagem

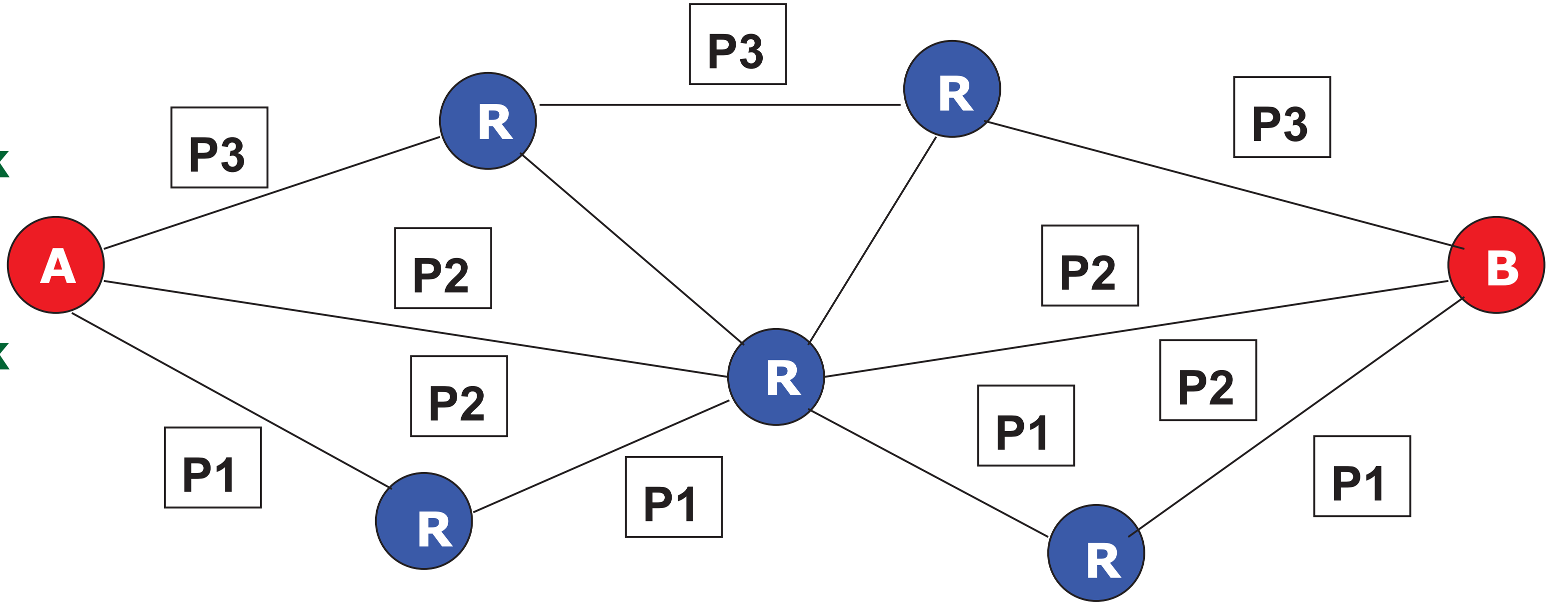
- Como reconstruir a mensagem corretamente?
  - Descartar os pacotes repetidos
  - Ordenar os pacotes
- Como implementar tal algoritmo?
  - Utilizando Árvores Binárias

Exemplo:

P1 Ok

P2 ?

P3 Ok



Ordem de Chegada:

P3 P1 P2

Confirmação de envio: P1 e P3.

Reenvio de P2.

Problemas: ordens e redundância dos pacotes

Algoritmo

- O primeiro pacote é colocado na raiz da árvore. Cada pacote sucessivo é comparado com o da raiz
- Se for igual, descarta-se a réplica. Se for menor ou maior, percorre-se os lados esquerdo ou direito da árvore
- Sub-árvore vazia implica inserção do novo pacote
- Sub-árvore não vazia implica comparação dos pacotes com a mesma

Problemas resolvidos?

- Problema da ordenação
  - A ordenação dos pacotes pode ser feita trivialmente com apenas uma chamada ao método inOrder() da árvore binária
- Problema da redundância
  - Solucionado com o algoritmo de inserção na árvore, visto que o pacote, antes de ser inserido, é comparado com os demais que já se encontram na árvore binária

2) Codificação de Huffman

- Algoritmo utilizado para comprimir arquivos
- Todo o algoritmo é baseado na criação de uma Árvore Binária
- Programas como **Winzip** e **WinRAR** utilizam este algoritmo
- Criado por **David Huffman** em 1952

Códigos e Caracteres

- Caracteres são letras, números e símbolos
- Códigos são seqüências de bits que podem representar de maneira ÚNICA um caracter
- b bits para representar c caracteres:
- Exemplos:

$$c = 2^b$$

ASCII (7 bits)	Extended ASCII (8 bits)
2^7 = 128 caracteres	2^8 = 256 caracteres

Como comprimir arquivos?

- No código ASCII, todos os caracteres têm um número fixo de bits
- Números variáveis de bits implica menor capacidade de armazenamento
- Associações com bits variáveis podem comprimir consideravelmente o arquivo

☝ Como comprimir arquivos desta maneira?

👍 Utilizando a Codificação de Huffman!

Exemplo:

- Considere o arquivo com o seguinte texto:

AAAAAAAAAABBBBBBBBCCCCCCDDDDDEE

- Freqüências: A = 10; B = 8; C = 6; D = 5; E = 2
- Construção da Árvore Binária
- Comparação do número de bits
  - Tamanho Fixo (8 bits) → Total = 248 bits
  - Tamanho Variável → Total = 69 bits

Compressão

- Depois da geração da árvore, o arquivo é percorrido novamente e cada caracter do arquivo é substituído pelo código binário contido na árvore, gerando uma cadeia de bits
- Criação da tabela de caracteres e códigos binários
- O que é armazenado?
  - Cadeia de bits gerada
  - Tabela de caracteres e códigos

Descompressão

- Regeneração da árvore binária através da tabela de caracteres e códigos
- A cadeia de bits é percorrida e, à medida que uma sub-cadeia é encontrada na tabela de caracteres e códigos, a mesma é substituída pelo caracter correspondente

Conclusões

- As árvores binárias são uma das estruturas de dados mais importantes devido a grande aplicabilidade das mesmas.
- A maioria dos algoritmos das árvores binárias são de simples entendimento, facilitando sobremaneira o desenvolvimento de sistemas.