ÁRVORES AVL

Prof. Antonio Carlos Sobieranski

UFSC – DEC – ENC

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Contextualização

No início dos anos 60, GM. Adelson-Velsky e E.M.Landis inventaram a primeira estrutura de árvore de busca binária auto-balanceável, chamada de AVL-Trees.

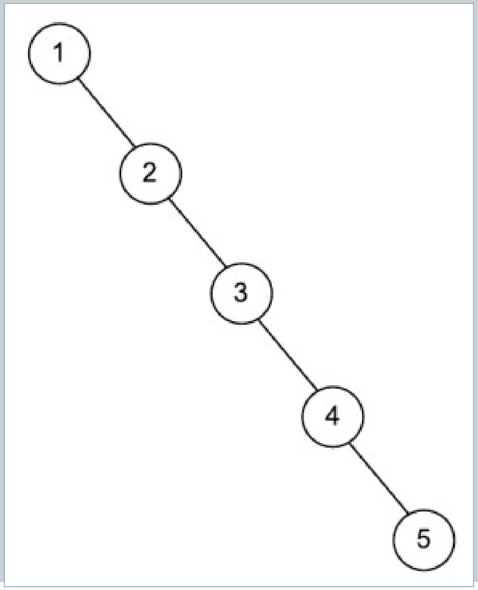
Uma árvore AVL é uma árvore binária de busca (simples BST) com uma condição de auto-balanceamento, baseado na premissa de que a altura das sub-árvores esquerda e direita não seja uma diferença maior que 1.

Esta condição, restaurada depois de cada modificação da árvore, força a forma geral de uma árvore AVL.

Contextualização

Podemos exemplificar a importância do balanceamento em BSTs através da construção a partir de uma árvore vazia, inserindo: 1,2,3,4,5.

Neste exemplo, podemos verificar que a natureza do problema de inserção faz adições sucessivas para o lado direito da árvore. Estamos degradando a performance da árvore de busca para O(n).



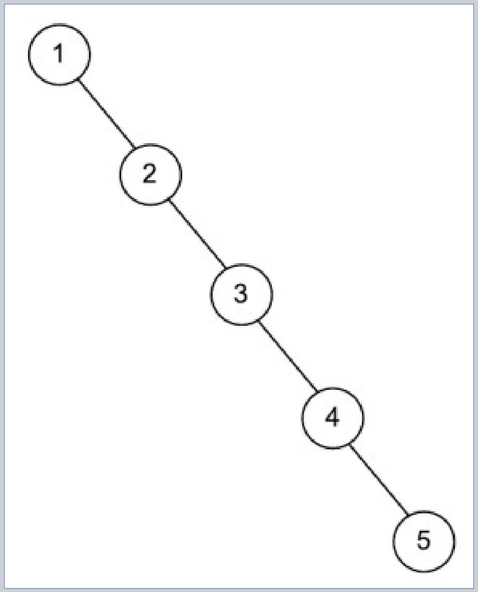
Contextualização

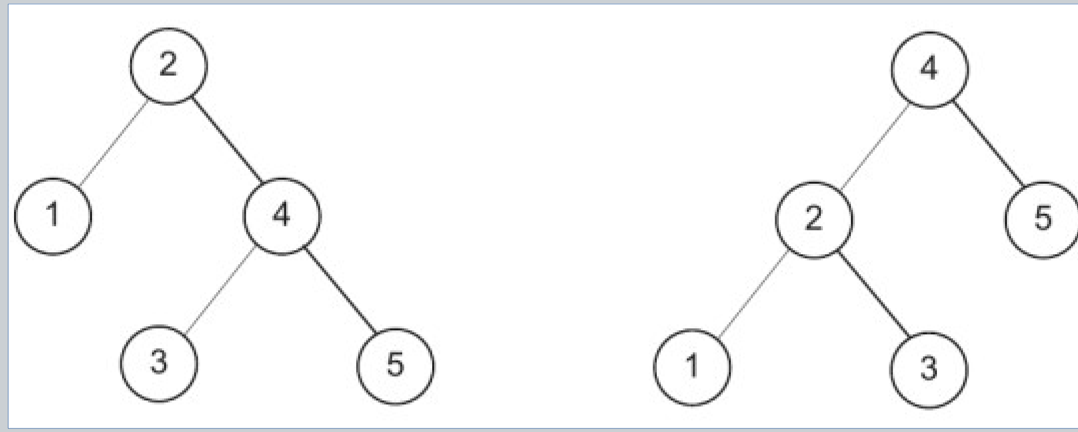
A árvore de busca binária da Figura à esquerda representa o pior caso de cenário onde o tempo de execução para todas as operações comuns, tais como busca, inserção, e remoção são O(n).

No entanto, é sábido que a complexidade de tempo bem como a profundidade (número de níveis da árvore) para busca em árvores cheias / completas é O(log n), para n nodos.

A aplicação de uma condição de balanceamento assegura novamente a performance da árvore para o pior caso, independente da ordem em que os elementos são inseridos

Contextualização





Solução

A solução é simples: realizar o balanceamento da árvore, com o objetivo de tornar cheia, ou pelo menos completa

(árvore quase cheia) = árvore de busca binária balanceada.

A condição de balanceamento de uma AVL é baseada nas diferenças de alturas das sub-árvores, também conhecida como fator de balanceamento de nodos.

Através de um conjunto de operações eficientes de rotações, determinadas pelo fator de balanceamento, a condição de balanceamento da árvore é restaurado.

Definições preliminares

Definição de árvore de busca: é um grafo conexo acíclico com um nó especial, denominado raiz da árvore.

Uma árvore binária de busca é uma árvore binária em que todo nodo, além de armazenar um elemento, possui 2 nós filhos: esquerdo e direito, que pela sua definição recursiva, são também sub-árvores da esquerda e direita.

É o tipo de árvore mais utilizado na computação. A principal utilização de árvores binárias são para busca. Sendo um tipo especial de grafo, podem ser representadas por tabelas (binária) ou ponteiros.

Definições preliminares

Uma árvore binária apresenta as seguintes definições:

- Cada nó possui 2 filhos no máximo (esquerdo e direito).
- Árvore binária cheia: possui todos os nós internos com 2 filhos e todas as folhas a mesma profundidade (F 5.34).
- Árvore binária completa é quase cheia (F 5.35).
- Profundidade de um nó: comprimento do caminho da raíz ao nó (raíz profundidade = 0).

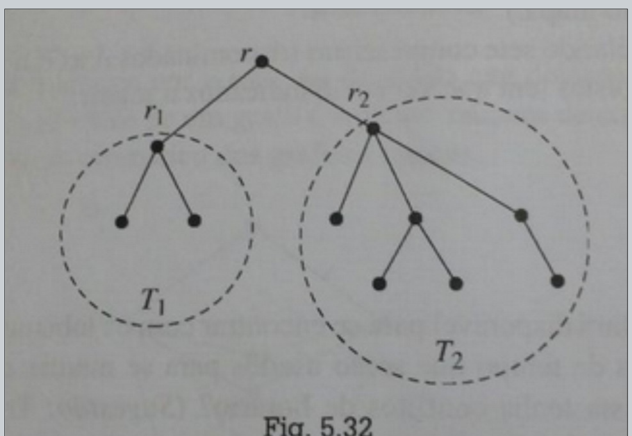


Fig. 5.32

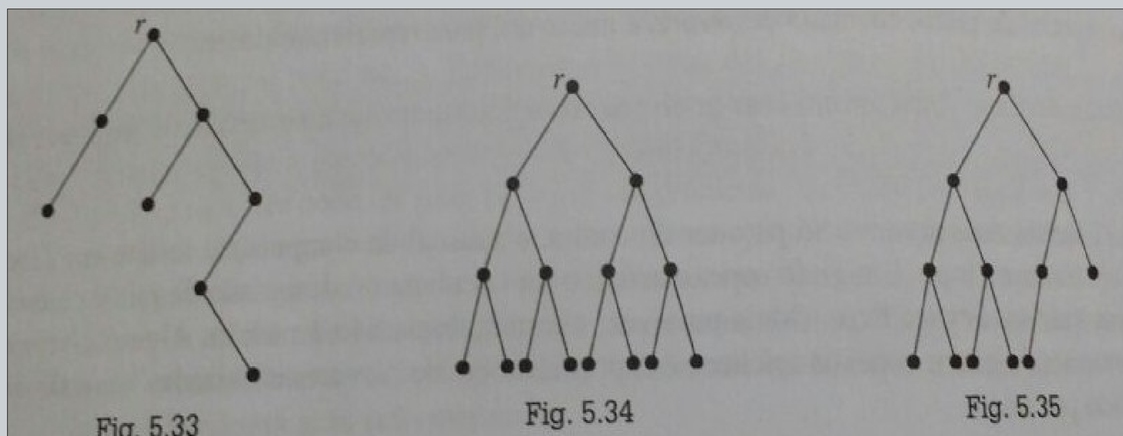


Fig. 5.33
Fig. 5.34
Fig. 5.35

Definições preliminares

Uma árvore binária apresenta as seguintes definições:

- Altura de uma árvore: maior profundidade dos nós, caminho mais longo, a maior profundidade de um nodo é a altura da árvore, e idealmente, uma árvore binária cheia possui altura d = O(log n), assim como a complexidade de tempo para realizar buscas na árvore.
- Nó sem filhos é chamado folha, possui grau 0.
- Os nós de uma árvore binária possuem grau 0, 1 ou 2.
- Floresta: grafo acíclico não necessariamente conexo, coleção de árvores distintas.
- Nodo cheio é aquele que possui 2 nodos filhos.

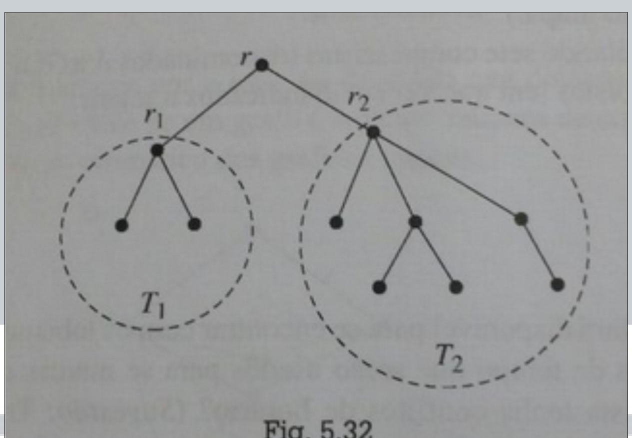


Fig. 5.32

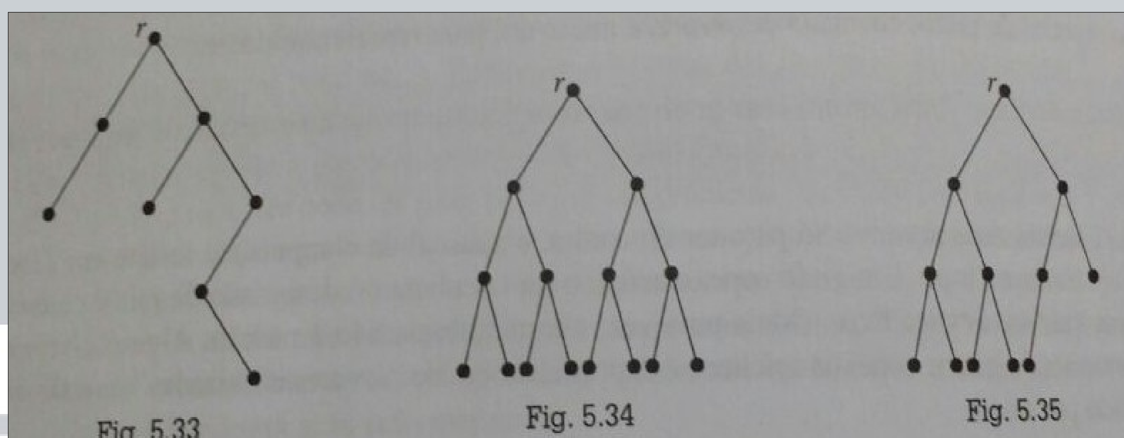


Fig. 5.33 Fig. 5.34 Fig. 5.35

Definições preliminares

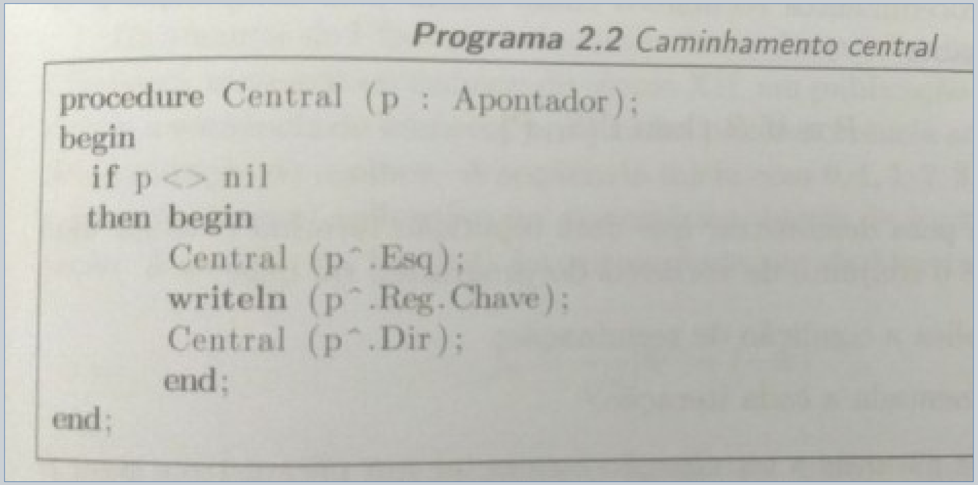
Realização da Busca, Inserção, Remoção:

- Perceba que a definição é recursiva e, devido a isso, muitas operações sobre árvores binárias utilizam recursão. A implementação desta estrutura é bastante direta, e algoritmos recursivos podem ser utilizados para realizar diversos cálculos tal como determinar o tamanho e a altura da árvore.
- Existe mais de uma ordem de caminhamento em árvores, sendo in-order, pre-order e post-order.
- Em geral, para árvores de pesquisa binária a mais utilizada é o caminhamento central (in-order).

Definições preliminares

Realização da Busca, Inserção, Remoção:

- (in-order): Esse caminhamento é expresso em termos recursivos a saber: caminha na sub-árvore da esquerda na ordem central; visita a raiz; caminha na sub-árvore da direita na ordem central.

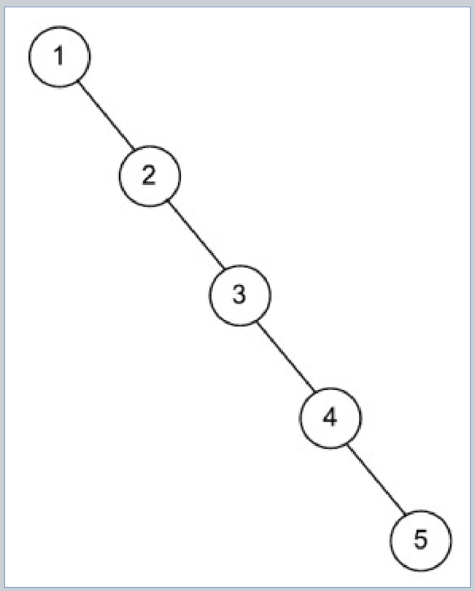

Programa 2.2 Caminhamento central

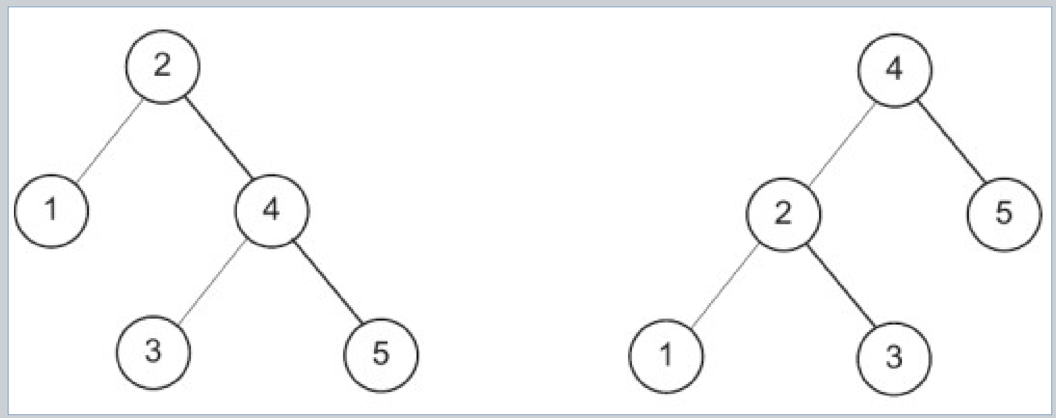
```
procedure Central (p : Apontador);
begin
  if p <> nil
  then begin
        Central (p^.Esq);
        writeln (p^.Reg.Chave);
        Central (p^.Dir);
       end;
end;
```

Definições preliminares

Realização da Busca, Inserção, Remoção:

- Qual a saída de cada uma das árvores utilizando caminhamento *in-order* ?





AVL – Balanceamento

Árvores AVL levam em consideração o fator de balanceamento da árvore, que é atualizado a cada operação de inserção ou remoção.

O fator de balanceamento é aplicado a cada nodo da árvore, sendo realizada a verificação das alturas das sub-árvores da direita e da esquerda, para cada nodo.

Formalmente, o fator de balanceamento pode ser definido como a diferença entre a altura a sub-árvore da esquerda e da direita:

$$balanceFactor = height(leftSubTree) - height(rightSubTree)$$

AVL – Balanceamento

Utilizando o fator de balanceamento dado, diz-se que uma sub-árvore é *left-heavy* se o fato de balanceamento > 0.

Se o fator de balanceamento < 0, diz-se que a sub-árvore é *right-heavy*.

Caso o fator de balanceamento=0, a sub-árvore está perfeitamente balanceada.

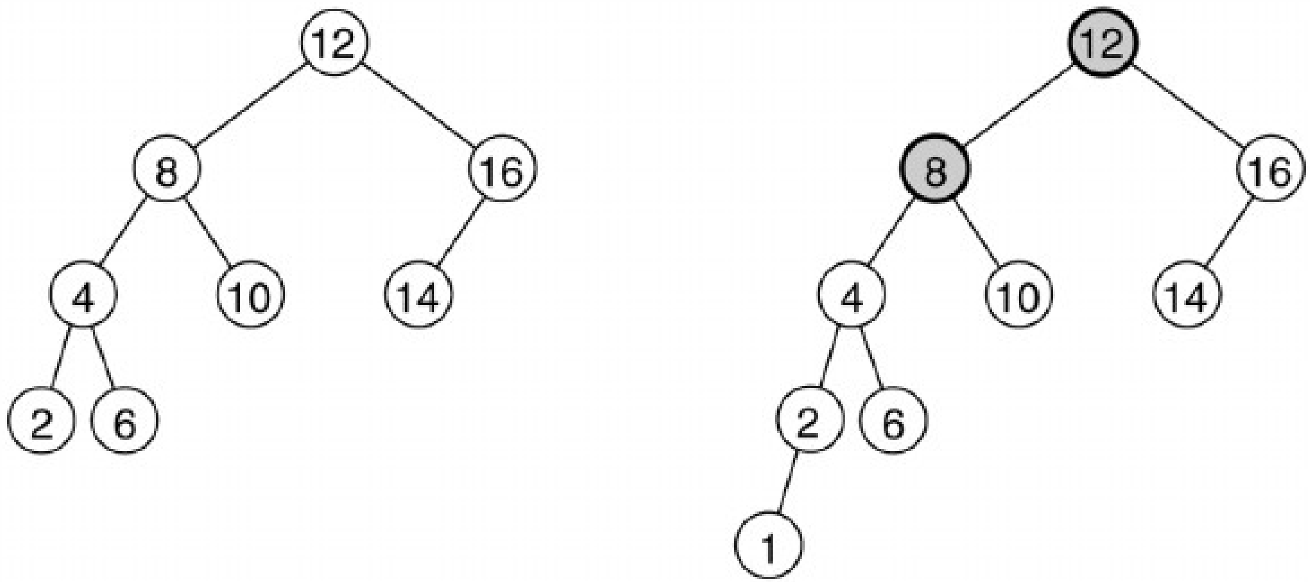
Para propósitos de implementação, consideramos a árvore balanceada quando os fatores de balanceamento forem

{-1,0,1}.

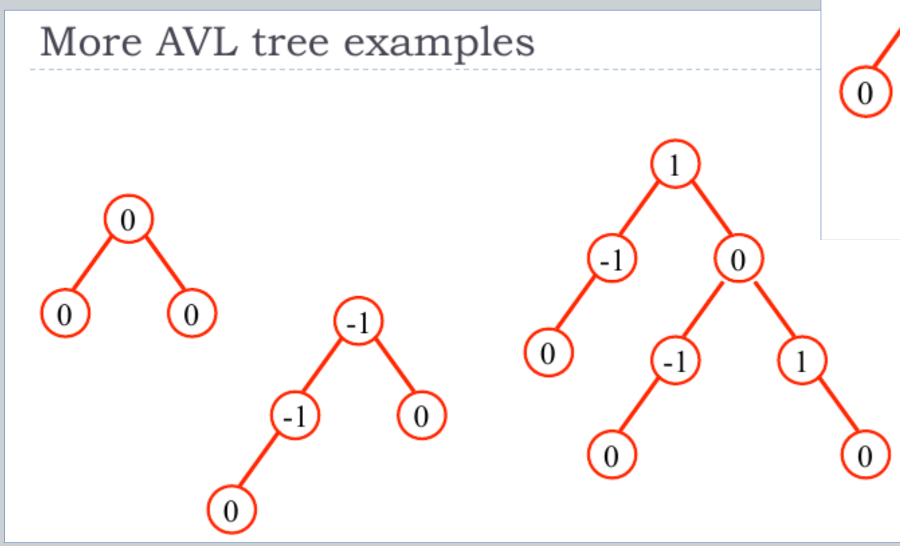
$$balanceFactor = height(leftSubTree) - height(rightSubTree)$$

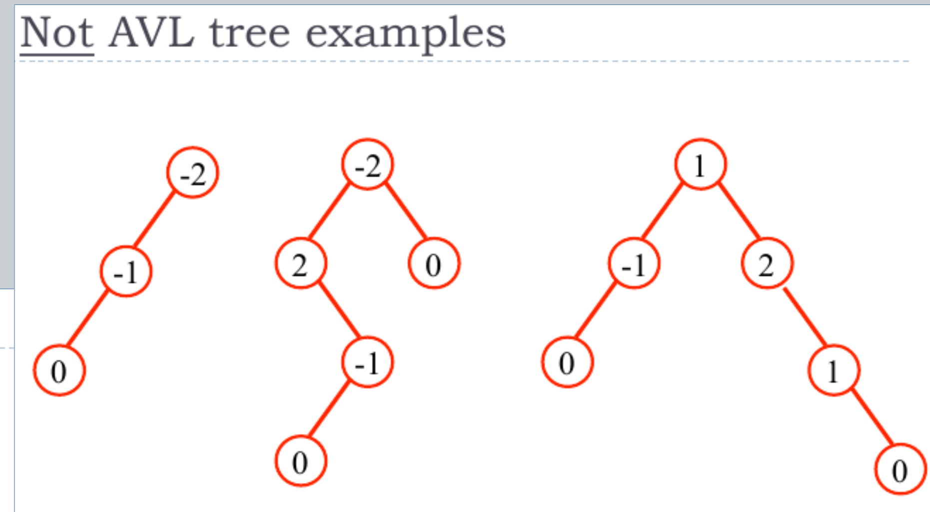
AVL – Balanceamento

- (a) an AVL tree
- (b) not an AVL tree (unbalanced nodes are darkened)



AVL – Balanceamento





AVL – Rotações

O balanceamento é realizado utilizando rotações, o qual altera a forma da árvore mas com o objetivo de preservar as propriedades da árvore.

A ideia é localmente reorganizar os nodos de uma sub-árvore não balanceada até ela se tornar balanceada. Cada operação de rotação realizada eficientemente em tempo constante, e envolve um trio de nodos, sendo: ***parent, left, right***.

AVL – Rotações

As **operações** envolvem **rotações** à **esquerda** e **rotações** à direita, objetivando **reduzir** a **altura/profundidade** da árvore pela movimentação de :

- Sub-árvores pequenas para baixo em função da altura total da árvore.
- Sub-árvores grandes para cima.

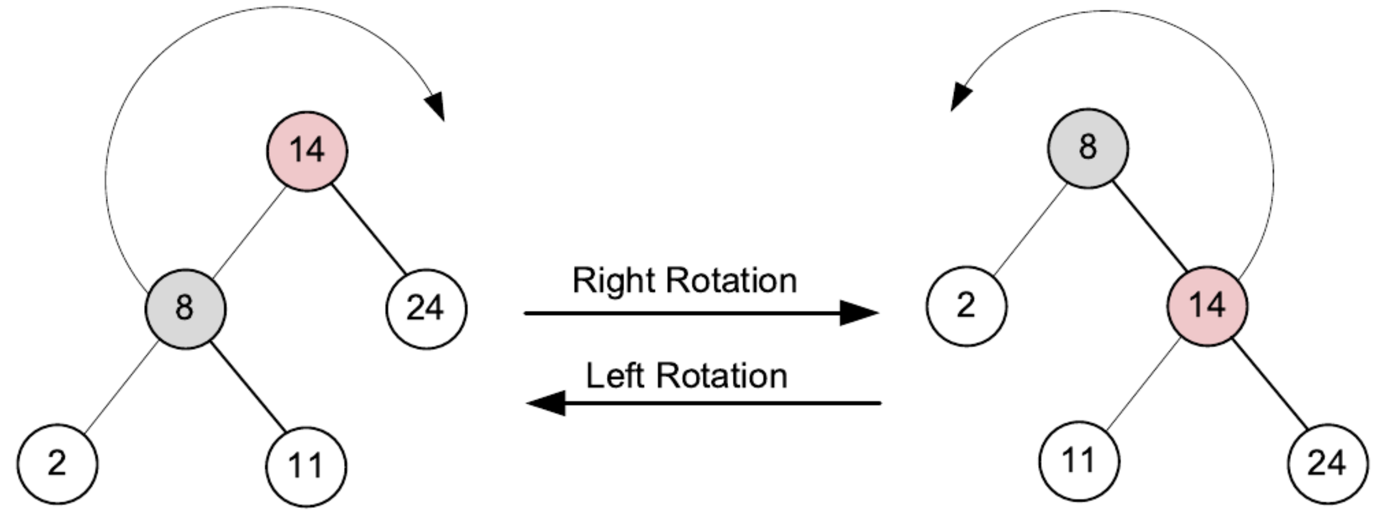


Figure 7.4: Tree left and right rotations

AVL – Rotações

Temos ainda 2 formas de rotação, que são aplicadas de acordo com os fatores de balanceamento dos nodos, juntamente com a análise dos filhos de cada nodo:

- **Rotação simples** é aplicada quando um nó está **desbalanceado** e seu filho estiver no **mesmo sentido** da **inclinação**, **formando** uma **linha reta**.
- **Rotação dupla** é aplicada quando um nó estiver **desbalanceado** e seu **filho** estiver **inclinado** no **sentido inverso** ao **pai**, formando um "**joelho**". A rotação dupla é simplesmente uma rotação, seguida de outra para o lado inverso da primeira.

AVL – Rotações

Para garantirmos as propriedades da árvore AVL rotações devem ser feitas conforme necessário após operações de remoção ou inserção. Seja **P** o nó pai, **FE** o filho da esquerda de **P** e **FD** o filho da direita de **P** podemos definir 4 tipos diferentes de rotação:

- **Rotação Simples à Direita**
- **Rotação Simples à Esquerda**
- **Rotação Dupla à Direita**
- **Rotação Dupla à Esquerda**

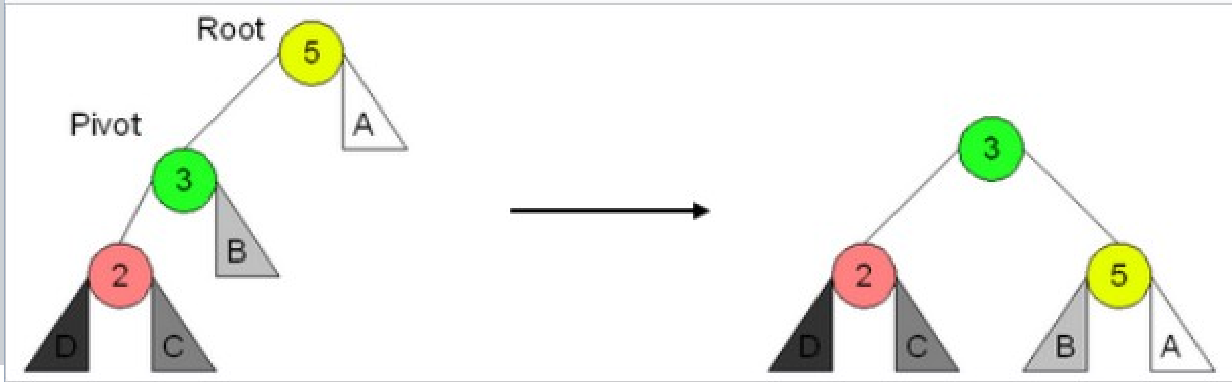
AVL – Rotações

Rotação Simples à Direita (+,+) → linha reta

Deve ser efetuada quando a diferença das alturas ***h*** dos filhos de ***P*** é igual a 2 (*supondo que já se inicie com a AVL, esses são os valores máximos*) e a diferença das alturas ***h*** dos filhos de ***FE*** é igual a 1. O nó ***FE*** deve tornar o novo pai e o nó ***P*** deve se tornar o filho da direita de ***FE***.

Segue pseudocódigo:

- Seja Y o filho à esquerda de X
- Torne o filho à direita de Y o filho à esquerda de X.
- Torne X o filho à direita de Y



Caso 1.1 - Rotação Simples à Direita

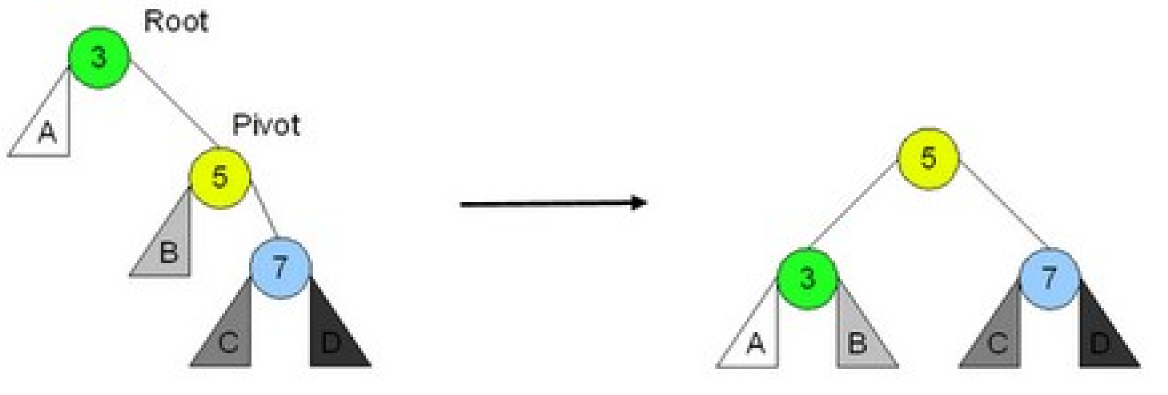
AVL – Rotações

Rotação Simples à Esquerda (–, –) → linha reta

Deve ser efetuada quando a diferença das alturas ***h*** dos filhos de ***P*** é igual a -2 e a diferença das alturas ***h*** dos filhos de ***FD*** é igual a -1. O nó ***FD*** deve tornar o novo pai e o nó ***P*** deve se tornar o filho da esquerda de ***FD***.

Segue pseudocódigo:

- Seja Y o filho à direita de X
- Torne o filho à esquerda de Y o filho à direita de X.
- Torne X filho à esquerda de Y

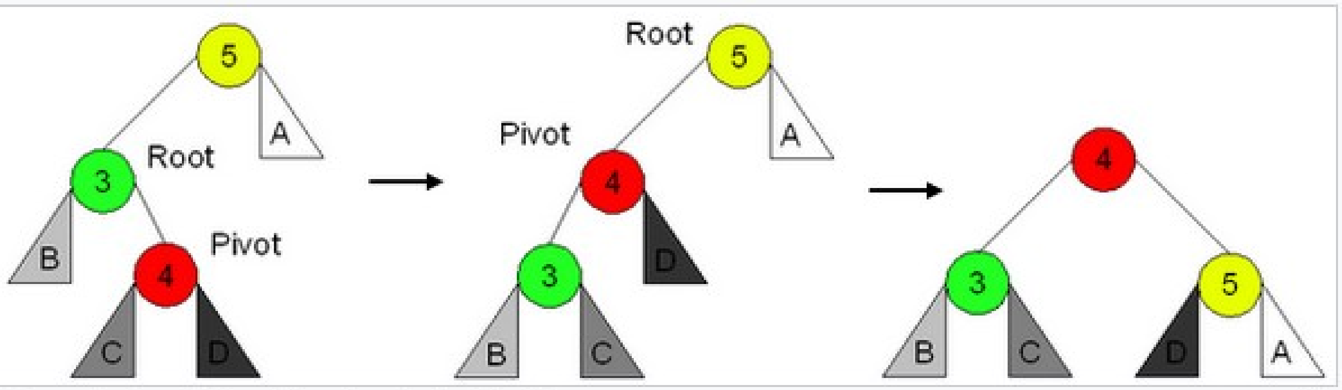


Rotação simples a esquerda

AVL – Rotações

Rotação Dupla à Direita (+, –) → joelho

Deve ser efetuada quando a diferença das alturas ***h*** dos filhos de ***P*** é igual a 2 e a diferença das alturas ***h*** dos filhos de ***FE*** é igual a -1. Nesse caso devemos aplicar uma **rotação** à **esquerda** no nó ***FE*** e, em seguida, uma rotação à **direita** no nó ***P***.

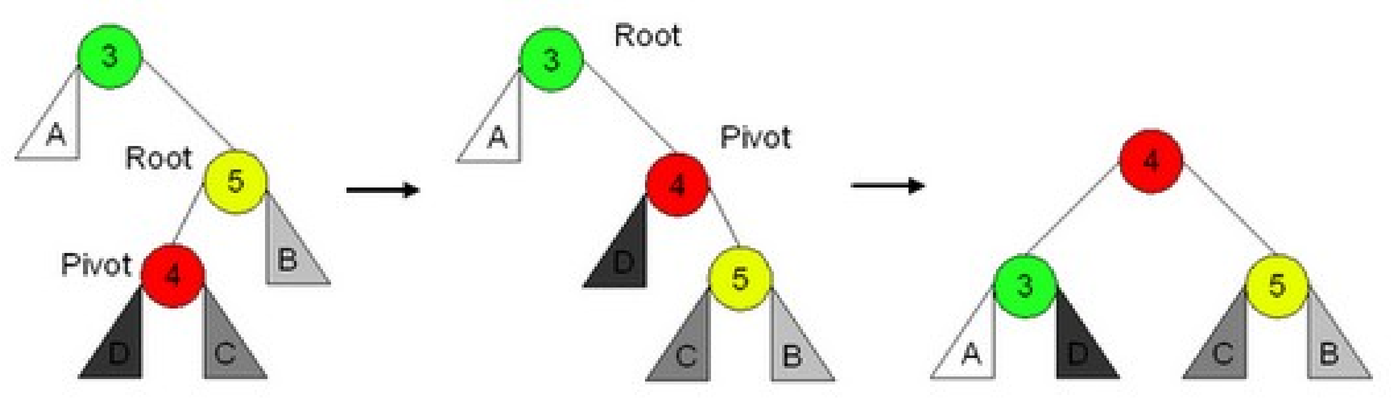


Caso 1.2 - Rotação dupla à direita

AVL – Rotações

Rotação Dupla à Esquerda (–, +) → joelho

Deve ser efetuada quando a diferença das alturas ***h*** dos filhos de ***P*** é igual a -2 e a diferença das alturas ***h*** dos filhos de ***FD*** é igual a 1. Nesse caso devemos aplicar uma **rotação** à **direita** no nó **FD** e, em seguida, uma **rotação** à esquerda no nó **P**.



Rotação dupla à esquerda

AVL – Rotações

A rotação a **direita** e a **esquerda** são **simétricas**. Apenas **ponteiros** são **alterados** por uma rotação resultando em complexidade de tempo **O(1)**. **Outros campos** apresentados nos nodos **não** são **alterados**.

```
algorithm RightRotation(node)
    Pre:  node.Left != ∅
    Post: node.Left is the new root of the subtree,
          node has become node.Left's right child and,
          BST properties are preserved
    LeftNode ← node.Left
    node.Left ← LeftNode.Right
    LeftNode.Right ← node
end RightRotation
```

```
algorithm LeftRotation(node)
    Pre:  node.Right != ∅
    Post: node.Right is the new root of the subtree,
          node has become node.Right's left child and,
          BST properties are preserved
    RightNode ← node.Right
    node.Right ← RightNode.Left
    RightNode.Left ← node
end LeftRotation
```

Exercício

Demonstre a construção e balanceamento de uma BST considerando os seguintes números em sequência:

15 27 49 10 8 67 59 9 13 20 14

Operações e Complexidades

A operação básica em uma árvore AVL geralmente envolve os mesmos algoritmos de uma árvore de busca binária convencional não-balanceada.

- **AVL search:** mesmo da BST padrão. O(log n) Mantendo a árvore balanceada todas as vezes, garantimos que o método get() irá rodar O(logn).
- **AVL insert:** mesmo BST insert, exceto que é necessário checkar o balanceamento e pode ser necessário balancear a árvore depois da inserção. O(log n)
- **AVL delete:** remover elemento, checar balanceamento, e corrigir. O(log n)

Operações e Complexidades

Mas a questão é, qual o custo para o método put() ?

Uma vez que **novos nodos** irão ser inseridos como **folhas**, atualizar os fatores de balanceamento de todos os **parents** irá requerir no máximo l**og n operações**, um para cada nível da árvore.

Se uma sub-árvore for encontrada fora de balanceamento um máximo de 2 rotações serão requeridas para que a árvore esteja novamente balanceada. Mas, cada uma das rotações trabalha em ***(1)*** tempo, então a operação de put() também executará em tempo ***O(log n)***.

Por definição, todos os nós da AVL devem ter **fb = -1, 0 ou 1**. Para garantir essa propriedade, a cada inserção ou remoção o fator de balanço deve ser atualizado a partir do pai do nó inserido até a raiz da árvore.

Conclusões

- **Árvores binárias** de **busca auto-balanceáveis** (AVL) foram inventada em 1962 pelos matemáticos **Adelson-Velskii** e **Landis**. Utiliza operações de *Add* e *Remove* modificadas para manter a árvore balanceada conforme operações são realizadas.
- Do ponto de vista operacional, Árvore AVL é um tipo de árvore binária que automaticamente torna a árvore balanceada conforme operações de inserção e remoção são efetuadas.
- Uma árvore **AVL implementa** as mesmas **propriedades** de uma **árvore** de busca **binária convencional**, e a única diferença é como a árvore se **comporta**, que eficientemente **reorganiza-se** de maneira que a propriedade de árvore binária é preservada e a profundidade restringe-se a **d=O(log n).**

Antonio Carlos Sobieranski

E-mail: a.sobieranski@ufsc.br

Sala C112 – Jd.Avenidas

dec.ufsc.br

UNIVERSIDADE FEDERAL
DE SANTA CATARINA