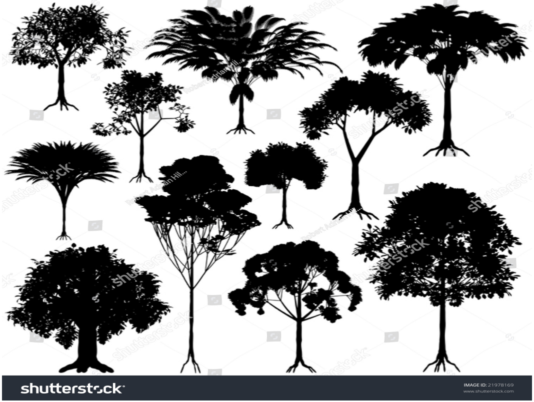
Onde estamos ...

Capítulo 06 – Árvores

Pontos fundamentais a serem cobertos:

1. Contexto e motivação
2. Definição
3. Implementações
4. Exercícios



Definição

- Uma árvore é uma estrutura hierárquica composta por nós e ligações entre eles
- Pode ser vista como um grafo acíclico
- Cada nó possui somente um pai e zero ou mais filhos
- Muitas definições ...

Estrutura

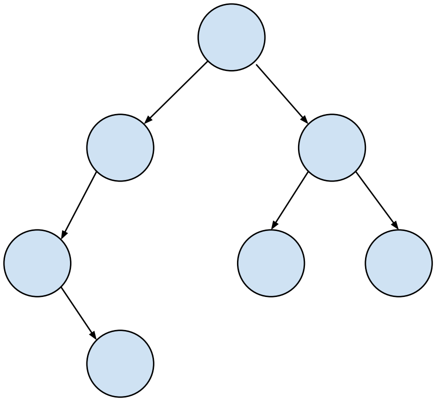


Figura 1: Exemplo de uma árvore

Roadmap para estudo

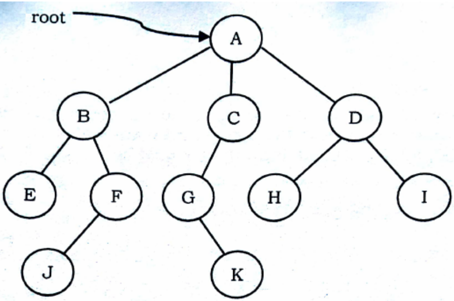
mantendo um *foco*:

1. Conceitos de árvores genéricas etc ...
2. Árvores Binárias
3. Árvore Binária de Busca
4. Árvore AVL (iniciais dos nomes: *Georgii Adelson-Velsky et Evguenii Landis (en), qui l'ont publié en 1962 sous le titre An Algorithm for the Organization of Information*)
5. Projeto 10% :
   - Implemente uma AVL;
   - Leia um conjunto de dados numéricos contidos no arquivo fornecido (um valor por linha: string, int, float, char), inserindo-os sequencialmente na AVL implementada;
   - Imprima o percurso (valores dos nós) em pré-ordem, em-ordem e pós-ordem, além da altura da árvore
   - Com a altura dará para ver se a AVL está OK!
6. Vídeos bem legais no Youtube da UNIVEST

Características – Requisitos

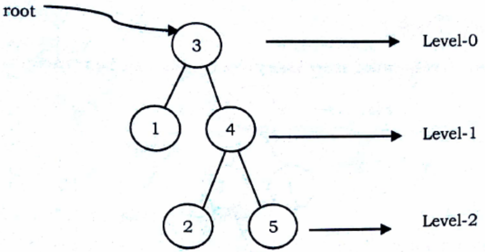
- Como é um nó?
- Qual o grau de um nó?
- Como devem estar estruturados os valores dos nós?
- O que é uma chave do nó?
- O que é a altura?
- Nível?
- Caminhos

Glossário – 01

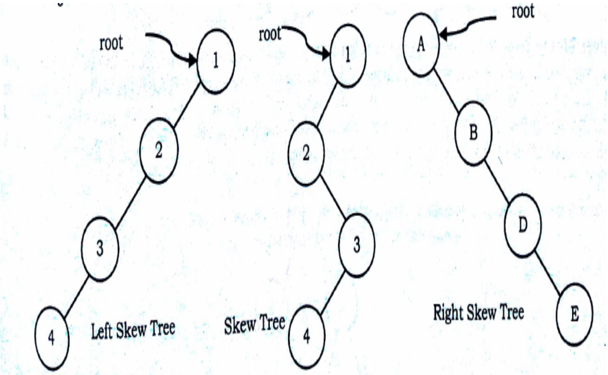


- The *root* of a tree is the node with no parents. There can be at most one root node in a tree (node above example).
- An *edge* refers to the link from parent to child (all links in the figure).
- A node with no children is called *leaf* node (E, J, K, H and I).
- Children of same parent are called *siblings* (B, C, D are siblings of A, and E, F are the siblings of B).
- A node p is an *ancestor* of node q if there exists a path from *root* to q and p appears on the path. The is called a *descendant* of p. For example, A, C and G are the ancestors of K.
- The set of all nodes at a given depth is called the *level* of the tree (B, C and D are the same level). T node is at level zero.

Glossário – 02



- The *depth* of a node is the length of the path from the root to the node (depth of G is 2, $A - C - G$).
- The *height* of a node is the length of the path from that node to the deepest node. The height of a tree is the length of the path from the root to the deepest node in the tree. A (rooted) tree with only one node (the root) has a height of zero. In the previous example, the height of B is 2 ($B - F - J$).
- *Height of the tree* is the maximum height among all the nodes in the tree and *depth of the tree* is the maximum depth among all the nodes in the tree. For a given tree, depth and height returns the same value. But for individual nodes we may get different results.
- The size of a node is the number of descendants it has including itself (the size of the subtree C is 3).
- If every node in a tree has only one child (except leaf nodes) then we call such trees *skew trees*. If every node has only left child then we call them *left skew trees*. Similarly, if every node has only right child then we call them *right skew trees*.


root
1
2
3
4
Left Skew Tree
root
1
2
3
4
Skew Tree
root
A
B
D
E
Right Skew Tree

Árvores Genéricas

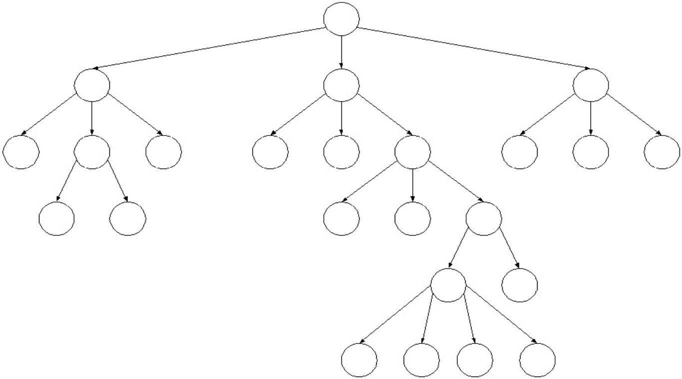


Figura 2: Usando os problemas de árvores genéricas para apresentar Árvores Binárias (AB)

Transformando uma Árvore Genérica em Binária

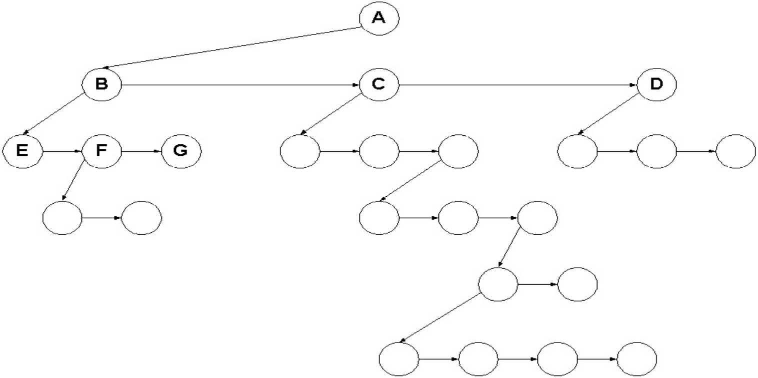


Figura 3: Usando os problemas de árvores genéricas para apresentar Árvores Binárias(AB)

Representação Computacional de uma Árvore Genérica

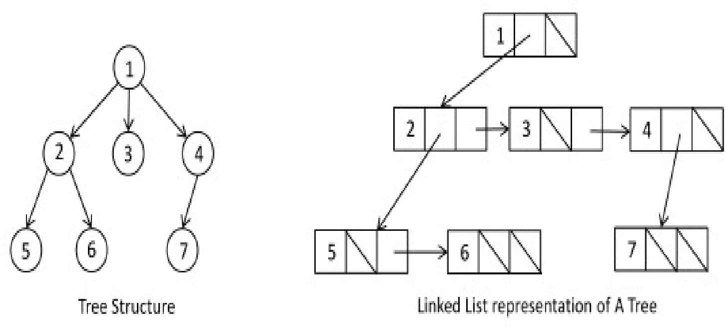
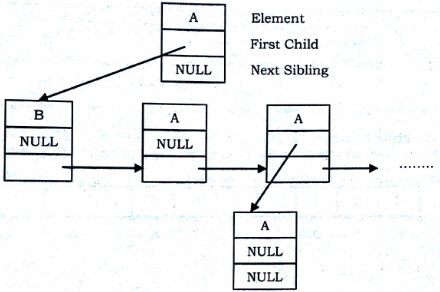


Figura 4: Felizmente há um algoritmo que transforme Árvores Genéricas em Binárias (AB)

Representação Computacional de uma Árvore Genérica



Based on this discussion, the tree node declaration for general tree can be given as:

```
struct TreeNode {
    int data;
    struct TreeNode *firstChild;
    struct TreeNode *nextSibling;
};
```

Note: Since we are able to convert any generic tree to binary representation; in practice we use binary trees. We can treat all generic trees with a first child/next sibling representation as binary trees.

Figura 5: Veja a *struct* ... lembra o quê?

Aplicações

- Área de compiladores: análise sintática
- Buscas com complexidade na ordem de: $O(\log n)$
- Na área de IA para construção de árvores de decisão: mineração de dados (*big data*)
- Organização de taxonomias de conhecimento
- Estruturas hierárquicas em geral

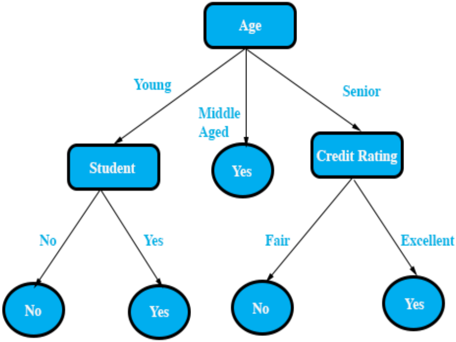


Figura 6: Árvore de Decisão

Aplicação de Árvores

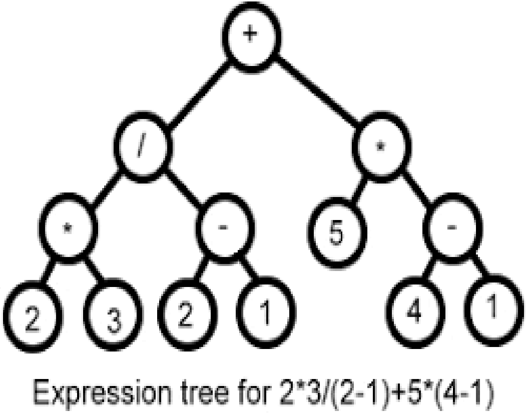


Figura 7: Árvores de expressões – binárias – novamente

Árvores Binárias de Buscas - ABBs

- Árvores Genéricas (AGs): foram utilizadas para motivação as ABBs
- Computacionalmente, as ABBs tem um interesse maior que as AGs
- Assim, se inicia com ABBs, e seus algoritmos serão adaptados as AGs

Árvores Binárias de Buscas - ABBs

- Árvores Genéricas (AGs): foram utilizadas para motivação as ABBs
- Computacionalmente, as ABBs tem um interesse maior que as AGs
- Assim, se inicia com ABBs, e seus algoritmos serão adaptados as AGs

Definição:

Árvore onde cada nó possui até 2 filhos. O filho da esquerda só pode conter chaves menores do que a do pai, enquanto que o filho da direita só comporta chaves maiores do que a do pai.

Árvore Binária de Busca

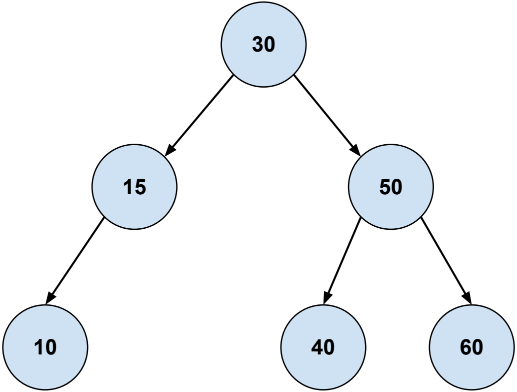
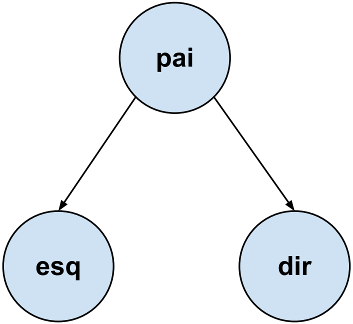


Figura 8: Exemplo de árvore binária de busca

Árvore Binária de Busca




```
struct ArvoreBinaria {
    struct ArvoreBinaria* esq;
    struct ArvoreBinaria* dir;

   // contém a chave e os dados satélite
    TipoItem valor;
};
```

Figura 9: Estrutura básica / nó

Operações Básicas

Operações Básicas

- Inserção
- Busca
- Remoção – faltando

Usos Comuns

- Dicionários / vetores associativos
- Filas de prioridades

Complexidade Computacional

Quando a árvore está balanceada todas as três operações podem ser implementadas com complexidade computacional igual a $O(\log n)$.

No pior caso (desbalanceamento) estas operações possuem complexidade $O(n)$ [**?**].

Árvore Binária de Busca - Inserção

```
INSERÇÃO(ARVORE, ITEM) {
    SE ARVORE->CHAVE == NULO
       ARVORE->ITEM = ITEM
       return      //e SE CHAVE jah existente?

    SE ITEM->CHAVE < ARVORE->CHAVE
        SE ARVORE->ESQ = NULO ENTÃO
            ARVORE->ESQ = ARVORE(ITEM)
        SENÃO
            INSERÇÃO(ARVORE->ESQ, ITEM)
    SENÃO
        SE ARVORE->DIR = NULO ENTÃO
            ARVORE->DIR = ARVORE(ITEM)
        SENÃO
            INSERÇÃO(ARVORE->DIR, ITEM)
}
```

Árvore Binária de Busca - Inserção

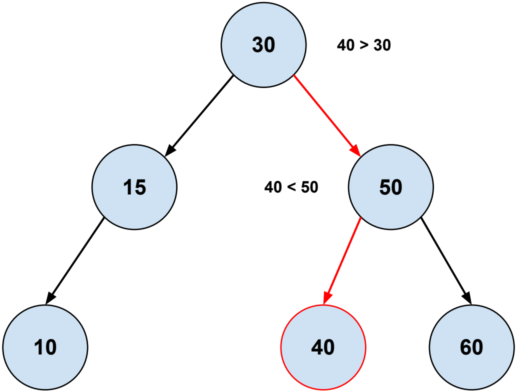


Figura 10: Exemplo de inserção da chave 40

Árvore Binária de Busca - Inserção

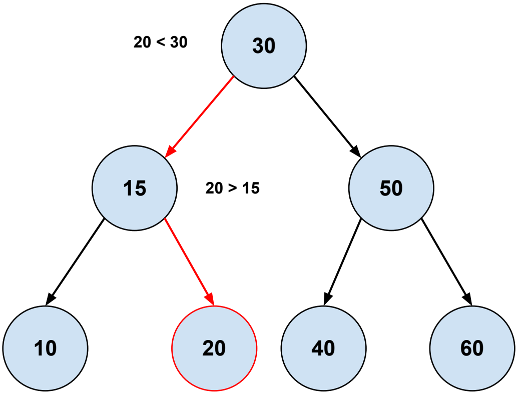


Figura 11: Exemplo de inserção da chave 20

Árvore Binária de Busca - Busca

```
BUSCA(ARVORE, CHAVE) {
    SE ARVORE = NULO
        return NULO

    SE ARVORE->CHAVE = CHAVE
        return ARVORE

    SE CHAVE < ARVORE->CHAVE
        return BUSCA(ARVORE->ESQ, CHAVE)
    SENÃO
        return BUSCA(ARVORE->DIR, CHAVE)
}
```

Árvore Binária de Busca – Remoção

A remoção de um nó se enquadra em um dos seguintes casos:

1. Remoção de um nó folha (nenhum filho)
2. Remoção de um nó com somente um filho
3. Remoção de um nó com dois filhos
4. Faltam as figuras ainda ....

Balanceamento

Uma árvore binária de busca balanceada garante operações de busca, inserção e remoção com complexidade $O(\log n)$, onde n é o número de nós, o que a torna atrativa para diversas aplicações.

Determinadas sequências de inserções ou remoções podem fazer com que uma ABB fique desbalanceada, tornando suas operações $O(n)$.

Cálculo da Altura

```
ALTURA(ARVORE) {
    SE ARVORE = NULO
        return -1

    A1 = ALTURA(ARVORE->DIR)
    A2 = ALTURA(ARVORE->ESQ)

    return maior(A1, A2) + 1
}
```

Cálculo da Altura

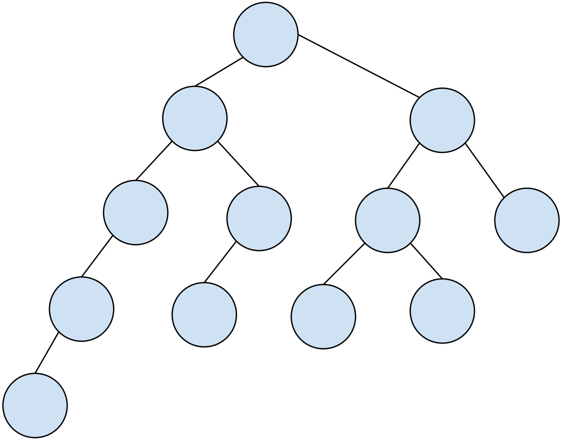


Figura 12: Exercício: determine a altura de cada subárvore.

Cálculo da Altura

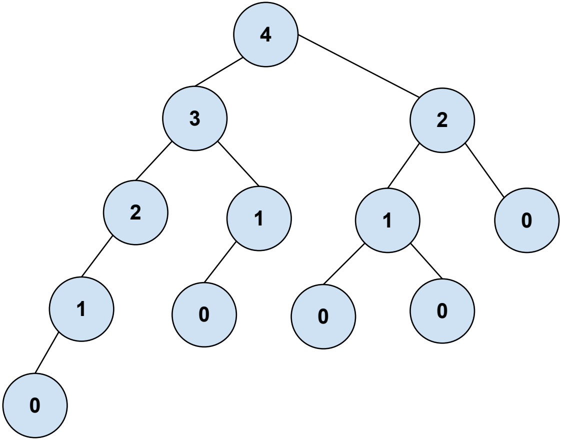


Figura 13: Resposta do exercício.

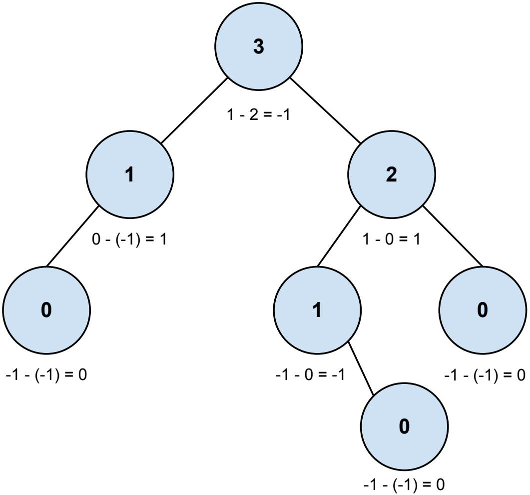
Cálculo do Fator de Balanceamento

```
FB(ARVORE) {
    A1 = ALTURA(ARVORE->ESQ)
    A2 = ALTURA(ARVORE->DIR)
    return A1 - A2
}
```

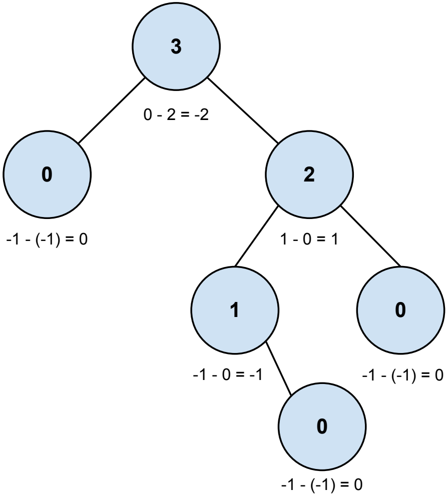
Balanceamento

- Uma ABB está balanceada quando cada nó possui um FB igual a -1, 0 ou 1
- Uma inserção ou remoção pode tornar uma árvore desbalanceada, necessitando de rotações para o seu balanceamento

Exemplo de ABB Balanceada



Exemplo de ABB Desbalanceada

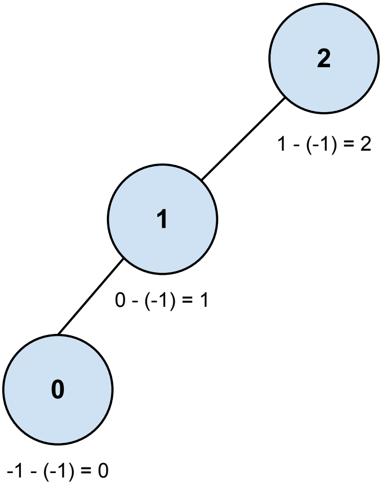


Operação de rotação

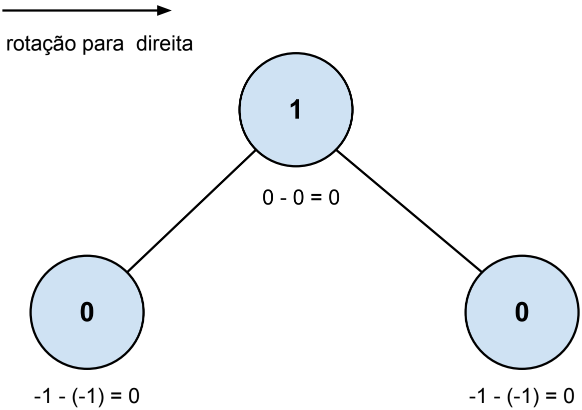
```
ROTACAO_DIREITA(RAIZ) {
    PIVO      = RAIZ->ESQ
    RAIZ->ESQ = PIVO->DIR
    PIVO->DIR = RAIZ
    RAIZ      = PIVO
}

ROTACAO_ESQUERDA(RAIZ) {
    PIVO      = RAIZ->DIR
    RAIZ->DIR = PIVO->ESQ
    PIVO->ESQ = RAIZ
    RAIZ      = PIVO
}
```

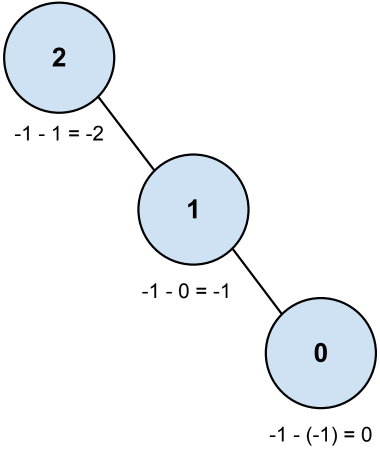
Rotação para Direita



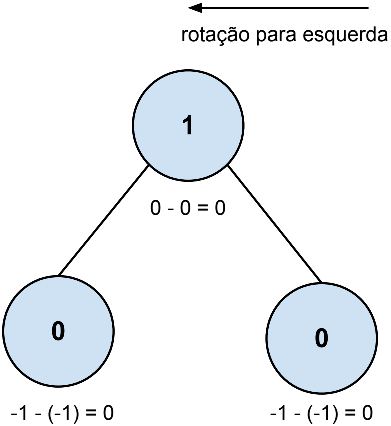
Rotação para Direita



Rotação para Esquerda



Rotação para Esquerda



Árvores AVL

- **AVL** desenvolvida por G. M. **A**delson-**V**elskii and E. M. **L**andis
- Garante o balanceamento da árvore ao realizar rotações após cada inserção ou remoção na ABB

Balanceamento - Inserção

```
BALANCEAMENTO(RAIZ) {
    SE FB(RAIZ) = -2 ENTÃO
        SE FB(RAIZ->DIR) = -1 ENTÃO
            ROTACAO_ESQUERDA(RAIZ)
        SENÃO
            ROTACAO_DIREITA(RAIZ->DIR)
            ROTACAO_ESQUERDA(RAIZ)
    SENÃO SE FB(RAIZ) = 2 ENTÃO
        SE FB(RAIZ->ESQ) = 1 ENTÃO
            ROTACAO_DIREITA(RAIZ)
        SENÃO
            ROTACAO_ESQUERDA(RAIZ->DIR)
            ROTACAO_DIREITA(RAIZ)
}
```

Balanceamento - Inserção

- Para que a árvore tenha um bom desempenho, é essencial que o balanceamento seja calculado eficientemente, isto é, sem a necessidade de percorrer toda a árvore após cada modificação
- Manter a árvore estritamente balanceada após cada modificação tem seu preço (desempenho). Árvores AVL são utilizadas normalmente onde o número de consultas é muito maior do que o número de inserções e remoções e quando a localidade de informação não é importante

Árvore de Espalhamento

- Reestrutura a árvore em cada operação de inserção, busca ou remoção por meio de operações de rotação
- Nome original: *splay tree* [**?**]. Não confundir com a Árvore N-Ária de Espalhamento (ANE) criada por professores da UDESC

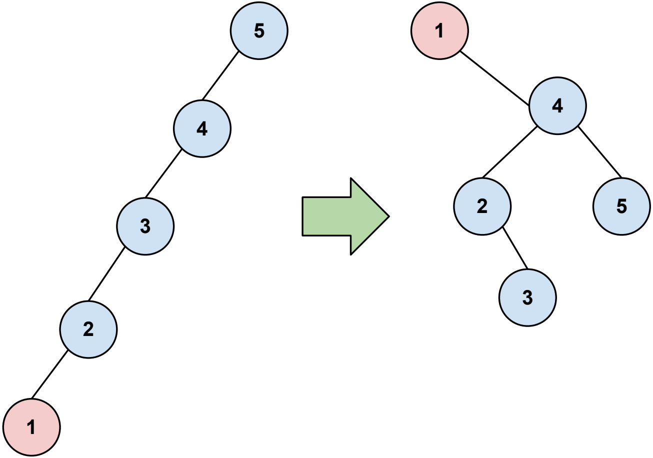
Árvore de Espalhamento

- Evita a repetição de casos ruins [O(n)] devido ao seu rebalanceamento natural
- Não realiza o cálculo de fatores de balanceamento, simplificando sua implementação
- Pior caso para uma operação se mantém O(n), mas, ao considerar uma cadeia de operações, *garante* uma complexidade amortizada de O(logn) para suas operações básicas

Árvore de Espalhamento

- Se baseia na operação de espalhamento, que utiliza rotações para mover uma determinada chave até a raiz
- A sua complexidade O(log n) em uma análise amortizada é garantida pelas rotações efetuadas, o que a difere do uso simples de heurísticas como o *mover para a raíz*

Exemplo - Espalhamento pela chave 1



Operações Básicas

Espalhamento Move a chave desejada para a raiz por uma sequência bem definida de operações de rotação

Busca Busca uma chave na árvore

Inserção Insere uma nova chave na árvore

Remoção Remove uma chave da árvore

- Uma árvore de espalhamento é uma árvore binária de busca válida, logo operações como os percursos (pré-em-pós) são idênticas as operações em uma ABB
- As operações de inserção, busca e remoção podem ser definidas com base na operação de espalhamento

Árvore de Espalhamento - Busca

```
BUSCA(RAIZ, CHAVE) {
    return ESPALHAMENTO(RAIZ, CHAVE)
}
```

Árvore de Espalhamento - Inserção

```
INSERE(RAIZ, CHAVE) {
    INSERE_ABB(RAIZ, CHAVE)
    return ESPALHAMENTO(RAIZ, CHAVE)
}
```

Árvore de Espalhamento - Remoção

```
REMOVE(RAIZ, CHAVE) {
    RAIZ = ESPALHAMENTO(RAIZ, CHAVE)

    SE RAIZ->DIR ENTÃO
        AUX = ESPALHAMENTO(RAIZ->DIR, CHAVE)
        AUX->ESQ = RAIZ->ESQ
    SENÃO
        AUX = RAIZ->ESQ

    return AUX
}
```

Estratégias de Espalhamento

Duas estratégias:

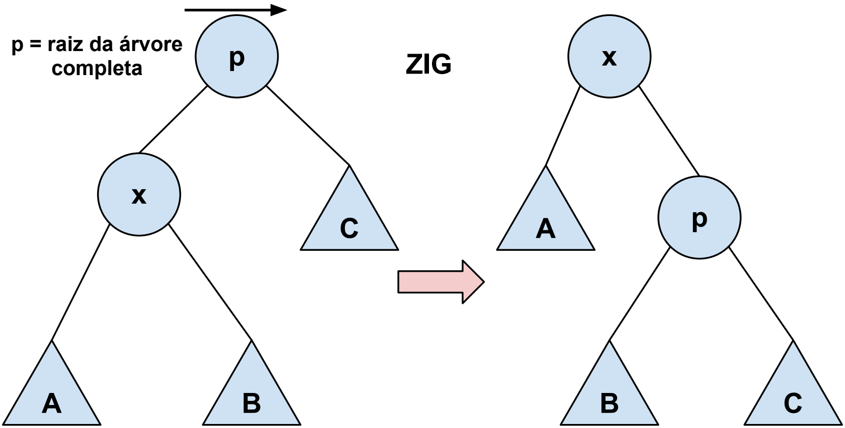
Bottom-Up Parte do nó acessado e o movimenta para a raiz da árvore por meio de rotações

Top-Down Parte do nó raiz, rotacionando e *removendo do caminho* os nós entre a raiz e o nó desejado, armazenando-os em duas árvores auxiliares, remontando a árvore completa na sua etapa final.

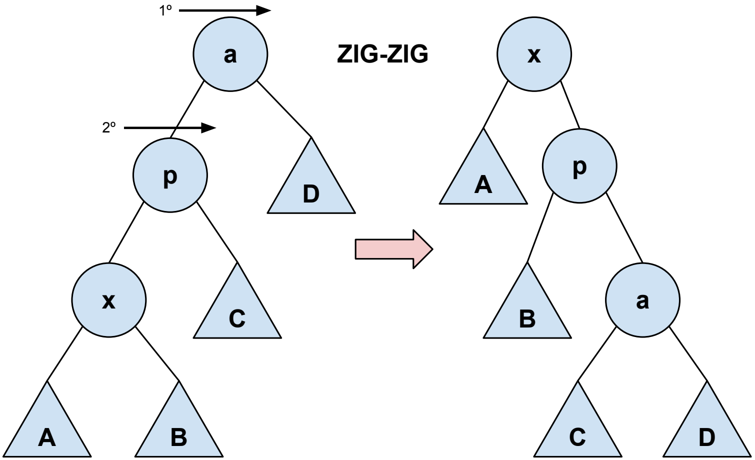
Espalhamento Bottom-Up

- Na estratégia Bottom-Up, a operação de espalhamento realiza rotações subindo gradativamente de níveis, a partir da chave desejada
- Enquanto a chave não estiver na raiz, deve-se verificar qual o caso aplicável (ZIG, ZIG-ZIG ou ZIG-ZAG) e realizar as rotações necessárias

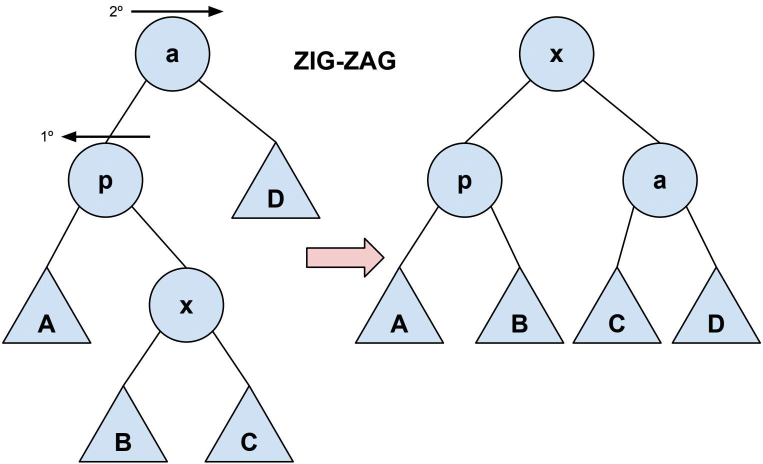
Caso 1: ZIG



Caso 2: ZIG-ZIG



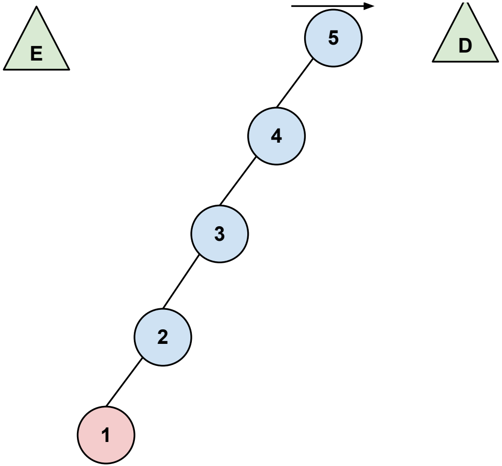
Caso 3: ZIG-ZAG



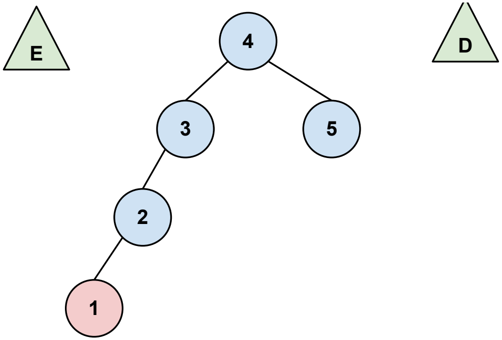
Espalhamento Top-Down

- Na estratégia Top-Down as chaves que estão no caminho da chave desejada para a raiz são rotacionadas e removidas para árvores auxiliares seguindo uma sequência de operações bem definidas
- Quando a chave desejada chega até a raiz, a árvore é remontada pelo retorno das chaves removidas

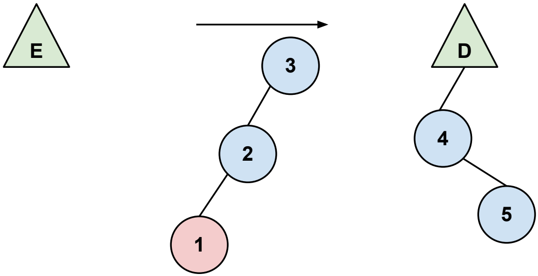
Exemplo: Top-Down 1/6



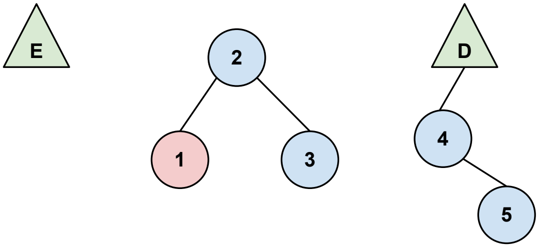
Exemplo: Top-Down 2/6



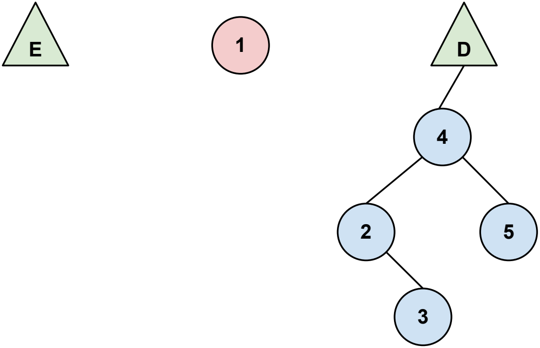
Exemplo: Top-Down 3/6



Exemplo: Top-Down 4/6



Exemplo: Top-Down 5/6



Exemplo: Top-Down 6/6