

# Estruturas de Dados

## Árvores AVL

Departamento de Computação

Prof. Martín Vigil

Adaptado de prof. Jean Martina e Aldo Wangenheim

2020.1



UNIVERSIDADE FEDERAL  
DE SANTA CATARINA

# Histórico

## Árvore AVL

Inventada em por Georgy Maximovich Adelson-Velskii e Yevgeniy Mikhailovich Landis ) e publicada em: "An algorithm for the organization of information. Trans. of Akademiya Nauk SSSR. Doklady, 1962, v. 146, no. 2, pp. 263-266 "



Georgy Maximovich  
Adelson-Velskii (Samara,  
Rússia, 1922)



Yevgeniy Mikhailovich Landis  
(Kharkov, Ucrânia, 1921)

# Características

- ▶ Manter uma árvore binária de busca balanceada sob a presença de constantes inserções e deleções é **ineficiente**;
- ▶ Para contornar esse problema foi criada a árvore **AVL (Adelson-Velskii e Landis)**;
- ▶ A árvore AVL é uma árvore binária com uma **condição de balanço**, porém não completamente balanceada;
- ▶ Árvores AVL permitem inserção / deleção e **rebalanceamento** aceitavelmente rápidos.

# Características

- ▶ Critério de balanceamento:
  - ▶ Dado um nodo qualquer, uma árvore está **AVL-balanceada** se as alturas das duas subárvores deste nodo diferem de, no máximo, 1;
- ▶ Para rebalancear uma árvore após uma inserção, são utilizadas **rotações** de subárvores:
  - ▶ Rotações simples em muitos casos;
  - ▶ Rotações duplas em alguns.

# Características

- Estrutura de um nodo na árvore AVL

```
estrutura nodoAVL {  
    T*      _dado;  
    nodoAVL* _esq;  
    nodoAVL* _dir;  
    int     _alt;  
};
```

- O campo **`_alt`** deve ser atualizado recursivamente quando um nodo for inserido ou deletado.

# Características

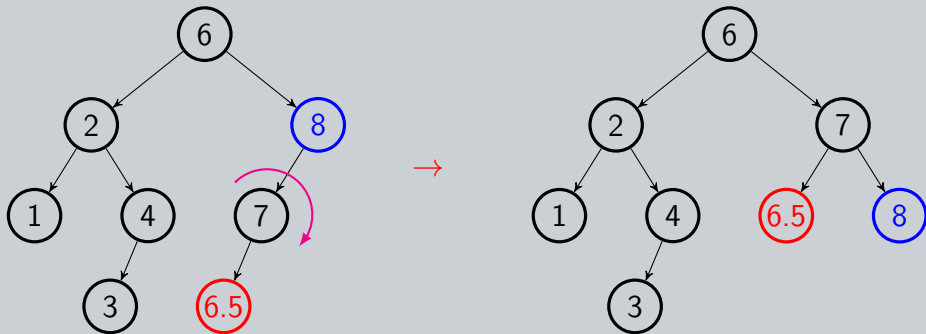
- ▶ Para o retorno da altura de uma subárvore necessitamos de um método especial;
- ▶ Um nodo pode ter 0, 1 ou 2 filho(s);
- ▶ Lembre-se que a pesquisa para garantir a condição-AVL é realizada perguntando-se a altura das duas subárvores.

```
int altura(nodoAVL* subárvore)
início
    se (subárvore == NULO)
        retorne -1; /* A altura de uma subárvore
                     inexistente é definida como -1 */
    senão
        retorne subárvore->_alt;
    fim se
fim
```

# Inclusão com rotação

Exemplo de rebalanceamento:

- O nodo com chave **6.5** desequilibrou a árvore no nodo **8**. Com a **rotação** da subárvore em torno do nodo 7 (**rotação simples à esquerda**), rebalanceamos.



# Inclusão com rotação

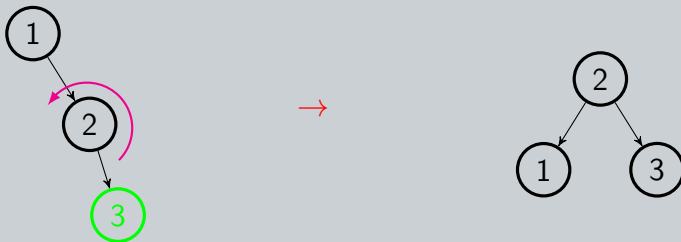
Algoritmo básico:

- ▶ Partimos do nodo inserido e subimos a árvore;
- ▶ Atualizamos a informação do balanceamento em cada nodo (na árvore AVL, cada nodo conhece a sua altura);
- ▶ Se chegamos à raiz sem encontrar algo errado, terminamos;
- ▶ Caso encontremos um nodo desbalanceado ( $|\text{alt}_{\text{esq}} - \text{alt}_{\text{dir}}| > 1$ ), realizamos a rotação no primeiro nodo desbalanceado encontrado;
- ▶ No exemplo anterior, isto significa que, depois da inserção de 6.5, o nodo 8 ficou desbalanceado.



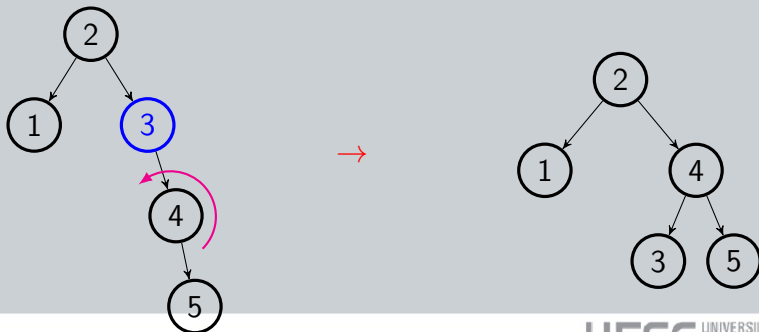
# Exemplo: Adicionando de 1 a 7

- ▶ O primeiro problema ocorre quando inserimos o 3;
- ▶ A condição-AVL foi violada;
- ▶ Executamos uma **rotação simples** entre a raiz (cuja condição-AVL está violada) e seu filho da direita (**rotação simples à direita.**)



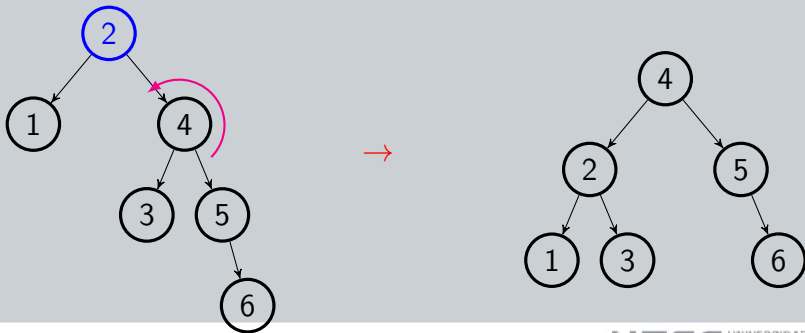
# Exemplo: Adicionando de 1 a 7

- ▶ Inserimos o elemento 4 sem problemas;
- ▶ Inserimos o elemento 5: violação em 3;
- ▶ Mesmo processo da rotação anterior será seguido;
- ▶ Importantíssimo: observe-se que 2 precisa ser notificado que seu filho agora é o nodo com chave 4.



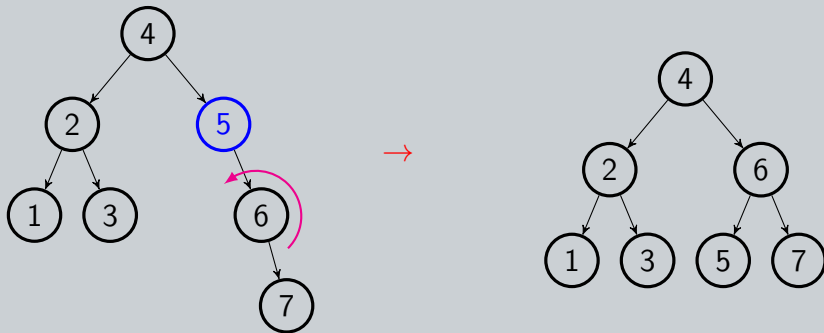
# Exemplo: Adicionando de 1 a 7

- ▶ A inclusão do elemento 6 desequilibra a raiz:
  - ▶ subárvore direita tem altura 0;
  - ▶ subárvore esquerda tem altura 2;
- ▶ Rotacionamos na raiz entre 2 e 4 (*rotação simples à direita.*)



# Exemplo: Adicionando de 1 a 7

- ▶ A inclusão de um nodo com chave 7 causa uma rotação como já havíamos visto antes:
  - ▶ O 5 fica desequilibrado;
  - ▶ Rotacionamos em 6 (**rotação simples à direita**).



# Inclusão com rotação simples à esquerda

```
nodoAVL *simp_roda_esq(k2 *nodoAVL)
variáveis locais
  k1 : *nodoAVL;
início
  k1 <- k2->esq;
  k2->esq <- k1->dir;
  k1->dir <- k2;
  /* atualize alturas */
  k2->alt <- max(altura(k2->esq), altura(k2->dir)) + 1;
  k1->alt <- max(altura(k1->esq), k2->alt) + 1;
  retorne k1; /* nova raiz da subárvore */
fim
```

# Inclusão com rotação simples à direita

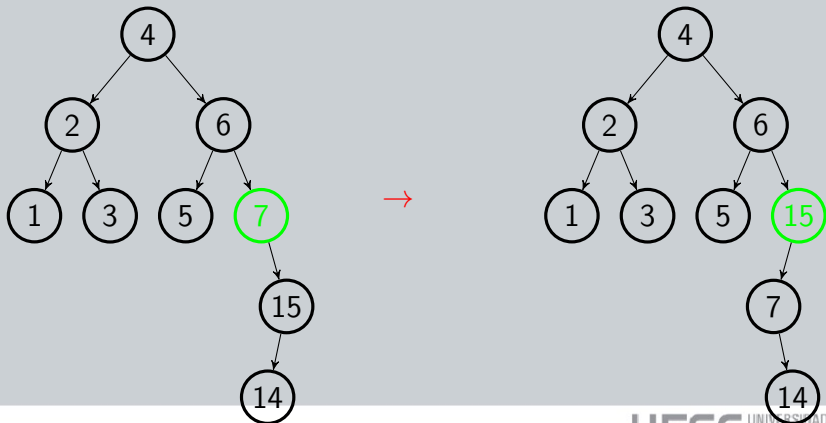
```
nodoAVL *simp_roda_dir(k2 *nodoAVL)
variáveis locais
  k1 : *nodoAVL;
início
  k1 <- k2->dir;
  k2->dir <- k1->esq;
  k1->esq <- k2;
  /* atualize alturas */
  k2->alt <- max(altura(k2->dir), altura(k2->esq)) + 1;
  k1->alt <- max(altura(k1->dir), k2->alt) + 1;
  retorne k1; /* nova raiz da subárvore */
fim
```

# Inclusão com rotação dupla

- ▶ O algoritmo descrito até agora tem um problema:
  - ▶ Existem casos onde ele não basta para consertar a árvore após uma inclusão ou exclusão;
- ▶ Exemplo: inserimos as chaves 15 e 14, nesta ordem, na árvore anterior:
  - ▶ A inserção de 15 não provoca problemas, nem desbalanceia a árvore;
  - ▶ A inserção de 14, por sua vez, faz o rebalanceamento falhar.

# Rotação Simples Não é Suficiente

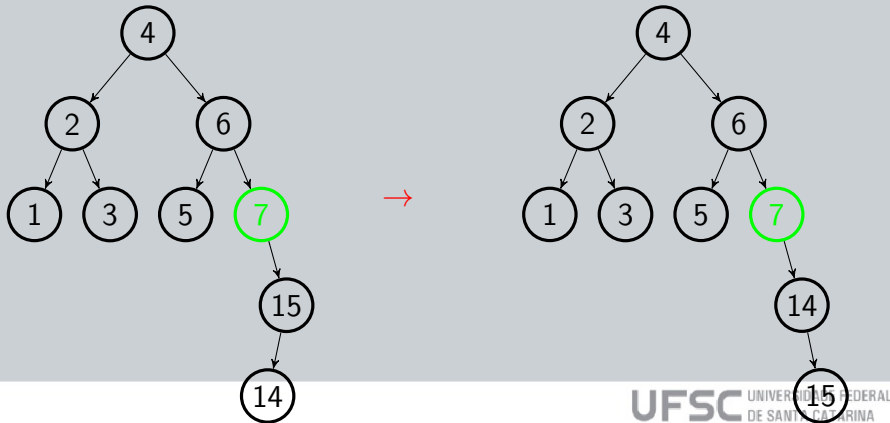
- ▶ O problema que surge aqui é que tanto 7 como 14 são candidatos à subárvore esquerda de 15.
- ▶ Neste caso, precisamos de uma dupla rotação.





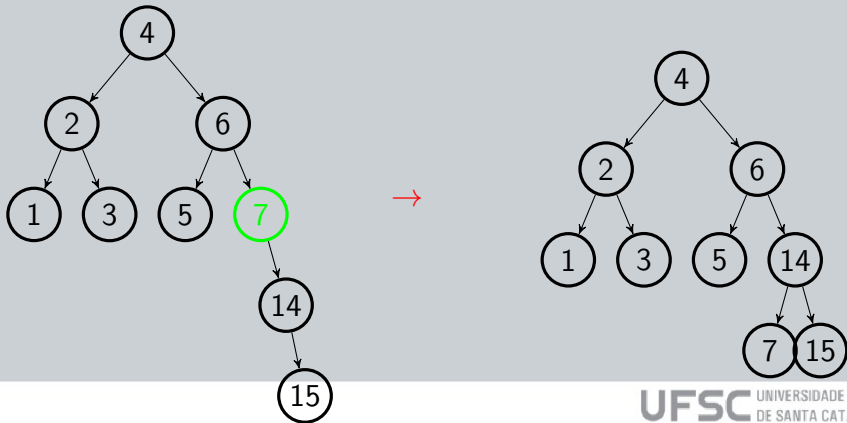
# Inclusão com Rotação Dupla

- ▶ No exemplo, a rotação direita-esquerda envolve o 7, o 15 e o 14;
- ▶ Primeiro rotacionamos 14-15 à direita, depois 7-14 à esquerda.



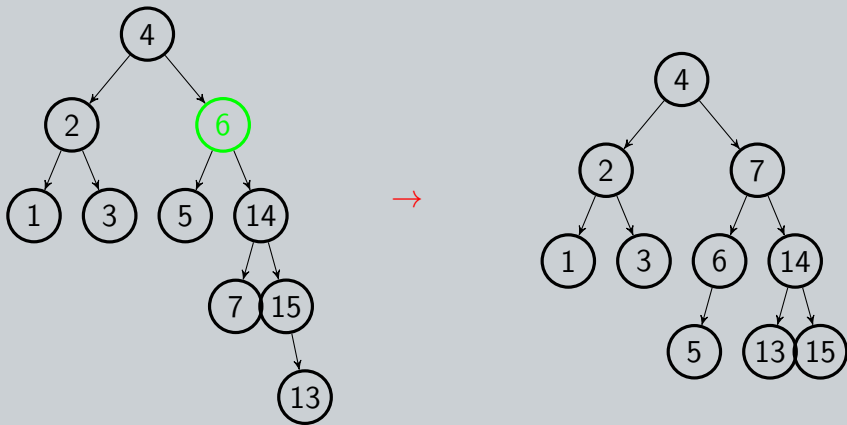
# Inclusão com Rotação Dupla

- ▶ No exemplo, a rotação direita-esquerda envolve o 7, o 15 e o 14;
- ▶ Primeiro rotacionamos 14-15 à direita, depois 7-14 à esquerda.



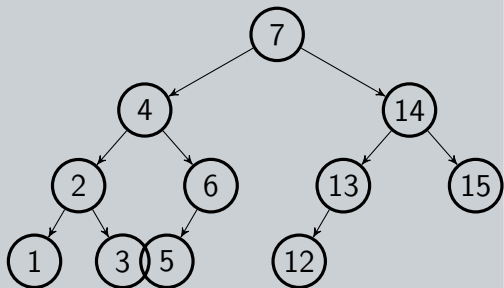
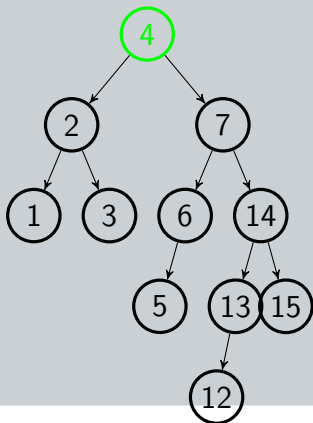
# Inclusão do Elemento 13

- Novamente uma dupla rotação direita-esquerda.



# Inclusão do Elemento 12

- ▶ A inclusão do elemento 12 provoca um desequilíbrio na raiz;
- ▶ Mas agora é simples.



# Rotação Dupla à Esquerda

- Função chamada somente se k3 possuir um filho à esquerda e se o filho à esquerda de k3 possuir um filho à direita. Ela realiza a rotação e atualiza as alturas das subárvores rotacionadas.

```
nodoAVL *dup_roda_esq(k3 : *nodoAVL)
início
    /* Rotacione entre k1 e k2 */
    k3->esq <- simp_roda_dir( k3-> esq );
    /* Rotacione entre k3 e k2 */
    retorne ( simp_roda_esq( k3 );
fim
```

- Da mesma forma que com a rotação simples, a rotação dupla à direita é a operação simétrica da rotação à esquerda e é facilmente implementada.

# Inserção em Árvore AVL

```
nodoAVL *inserçãoAVL(T* info, *nodoAVL arv, *nodoAVL pai)
variáveis
    arv_rodada : *nodoAVL;
início
    se (arv é NULO) então /* Folha: aloca novo nodo */
        arv <- aloca novo nodo;
        se (arv é NULO) então retorne ERRO;
        arv->info <- info; arv->alt <- 0;
        arv->esq <- NULO; arv->dir <- NULO;
    senão
        se (info < arv->info) então
            arv->esq <- inserçãoAVL(info, arv->esq, arv);
            se ((altura(arv->esq) - altura(arv->dir) > 1)
                se (info < arv->esq->info) então
                    arv_rodada <- simp_rodada_esq(arv);
                senão arv_rodada <- dup_rodada_esq(arv);
            fim se
            se (pai->esq = arv) então pai->esq <- arv_rodada;
            senão pai->dir <- arv_rodada;
        fim se
    senão
        arv->alt <- max( altura(arv->esq), altura(arv->dir)) + 1;
    fim se
    senão // (continua)
```

# Inserção em Árvore AVL

```
nodoAVL *inserçãoAVL(T* info, *nodoAVL arv, *nodoAVL pai)
variáveis
    arv_rodada : *nodoAVL;
início
    se (arv é NULO) então /* Folha: aloca novo nodo */
        arv <- aloca novo nodo;
        se (arv é NULO) então retorne ERRO;
        arv->info <- info; arv->alt <- 0;
        arv->esq <- NULO; arv->dir <- NULO;
    senão
        se (info < arv->info) então
            arv->esq <- inserçãoAVL(info, arv->esq, arv);
            se ((altura(arv->esq) - altura(arv->dir) > 1)
                se (info < arv->esq->info) então
                    arv_rodada <- simp_roda_esq(arv);
                senão arv_rodada <- dup_roda_esq(arv);
            fim se
            se (pai->esq = arv) então pai->esq <- arv_rodada;
            senão pai->dir <- arv_rodada;
            fim se
        senão
            arv->alt <- max( altura(arv->esq), altura(arv->dir)) + 1;
        fim se
    senão // (continua)
```

# Inserção em Árvore AVL

```
nodoAVL *inserçãoAVL(T* info, *nodoAVL arv, *nodoAVL pai)
variáveis
    arv_rodada : *nodoAVL;
início
    se (arv é NULO) então /* Folha: aloca novo nodo */
        arv <- aloca novo nodo;
        se (arv é NULO) então retorne ERRO;
        arv->info <- info; arv->alt <- 0;
        arv->esq <- NULO; arv->dir <- NULO;
    senão
        se (info < arv->info) então
            arv->esq <- inserçãoAVL(info, arv->esq, arv);
            se ((altura(arv->esq) - altura(arv->dir) > 1) //Rotaciona?
            se (info < arv->esq->info) então
                arv_rodada <- simp_rodada_esq(arv);
            senão arv_rodada <- dup_rodada_esq(arv);
            fim se
            se (pai->esq = arv) então pai->esq <- arv_rodada;
            senão pai->dir <- arv_rodada;
            fim se
        senão
            arv->alt <- max( altura(arv->esq), altura(arv->dir)) + 1;
        fim se
    senão // (continua)
```



# Inserção em Árvore AVL

```
nodoAVL *inserçãoAVL(T* info, *nodoAVL arv, *nodoAVL pai)
variáveis
    arv_rodada : *nodoAVL;
início
    se (arv é NULO) então /* Folha: aloca novo nodo */
        arv <- aloca novo nodo;
        se (arv é NULO) então retorne ERRO;
        arv->info <- info; arv->alt <- 0;
        arv->esq <- NULO; arv->dir <- NULO;
    senão
        se (info < arv->info) então
            arv->esq <- inserçãoAVL(info, arv->esq, arv);
            se ((altura(arv->esq) - altura(arv->dir) > 1)
                se (info < arv->esq->info) então //Qual Rotacão?
                    arv_rodada <- simp_roda_esq(arv);
                    senão arv_rodada <- dup_roda_esq(arv);
                fim se
            se (pai->esq = arv) então pai->esq <- arv_rodada;
            senão pai->dir <- arv_rodada;
        fim se
    senão
        arv->alt <- max( altura(arv->esq), altura(arv->dir)) + 1;
    fim se
    senão // (continua)
```

# Inserção em Árvore AVL

```
nodoAVL *inserçãoAVL(T* info, *nodoAVL arv, *nodoAVL pai)
variáveis
    arv_rodada : *nodoAVL;
início
    se (arv é NULO) então /* Folha: aloca novo nodo */
        arv <- aloca novo nodo;
        se (arv é NULO) então retorne ERRO;
        arv->info <- info; arv->alt <- 0;
        arv->esq <- NULO; arv->dir <- NULO;
    senão
        se (info < arv->info) então
            arv->esq <- inserçãoAVL(info, arv->esq, arv);
            se ((altura(arv->esq) - altura(arv->dir) > 1)
                se (info < arv->esq->info) então
                    arv_rodada <- simp_roda_esq(arv);
                senão arv_rodada <- dup_roda_esq(arv);
            fim se // Acerta o Pai
            se (pai->esq = arv) então pai->esq <- arv_rodada;
            senão pai->dir <- arv_rodada;
            fim se
        senão
            arv->alt <- max( altura(arv->esq), altura(arv->dir)) + 1;
        fim se
    senão // (continua)
```

# Inserção em Árvore AVL

```
nodoAVL *inserçãoAVL(T* info, *nodoAVL arv, *nodoAVL pai)
variáveis
    arv_rodada : *nodoAVL;
início
    se (arv é NULO) então /* Folha: aloca novo nodo */
        arv <- aloca novo nodo;
        se (arv é NULO) então retorne ERRO;
        arv->info <- info; arv->alt <- 0;
        arv->esq <- NULO; arv->dir <- NULO;
    senão
        se (info < arv->info) então
            arv->esq <- inserçãoAVL(info, arv->esq, arv);
        se ((altura(arv->esq) - altura(arv->dir) > 1)
            se (info < arv->esq->info) então
                arv_rodada <- simp_rodada_esq(arv);
            senão arv_rodada <- dup_rodada_esq(arv);
        fim se
        se (pai->esq = arv) então pai->esq <- arv\_rodada;
        senão pai->dir <- arv\_rodada;
        fim se
    senão // Ajusta a Altura.
        arv->alt <- max( altura(arv->esq), altura(arv->dir)) + 1;
    fim se
senão // (continua)
```

# Inserção em Árvore AVL

```
senão //(continuado de cima)
  se (info > arv->info) então
    /* caso simétrico para árvore direita */
    arv->dir <- inserçãoAVL(info, arv->dir, arv);
    se ((altura(arv->dir) - altura(arv->esq) > 1)
      se (info < arv->dir->info) então
        arv_rodada <- simp_roda_dir(arv);
      senão
        arv_rodada <- dup_roda_dir(arv);
    fim se
    se (pai->dir = arv) então pai->dir <- arv_rodada;
    senão pai->esq <- arv_rodada;
    fim se
  senão
    arv->alt <- max( altura(arv->esq), altura(arv->dir)) +
      1;
  fim se
senão
  retorne ERRO: "chave já está na árvore"
fim se
fim se
fim se
retorne arv;
fim
```

# Deleção em Árvore AVL

- ▶ Busque a chave a deletar na árvore;
- ▶ Delete utilizando o algoritmo **recursivo** de Deleção da Árvore Binária de Busca (DABB) visto em aula;
- ▶ Modifique DABB para:
  - ▶ Logo após retorno da chamada recursiva **atualizar & verificar** alturas das subárvores filhas;
  - ▶ Aplicar "regra do zigue-zague" se necessário.

# Deleção em Árvore AVL

- ▶ "regra do zigue-zague":
  - ▶ Olhar subárvore que desequilibrou e seus filhos;
- ▶ Desequilíbrio é à esquerda:
  - ▶ Sub-subárvore esquerda é a mais funda?
  - ▶ Esquerda-esquerda: rotação simples à esquerda;
  - ▶ Sub-subárvore direita é a mais funda?
  - ▶ Esquerda-direita: rotação dupla à esquerda;
- ▶ Desequilíbrio é à direita:
  - ▶ Sub-subárvore direita é a mais funda?
  - ▶ Direita-direita: rotação simples à direita
  - ▶ Sub-subárvore esquerda é a mais funda?
  - ▶ Direita-esquerda: rotação dupla à direita

# Deleção em Árvore AVL

```
tNodo* delete(info: tInfo, arv: *tNodo)
    tmp, filho: *tNodo;
início
    se (arv = NULO) então
        retorne arv
    senão
        se (info < arv->info) // Vá à esquerda.
            arv->filhoÀEsquerda <- delete(info, arv->filhoÀEsquerda)
                ;
            retorne arv;
        senão
            se (info > arv->info) // Vá à direita.
                arv->filhoÀDireita <- delete(info, arv->filhoÀDireita)
                    ;
                arv <- atualiza-zigue-zague(arv);
            retorne arv;
        senão // Encontrei elemento que quero deletar.
            (CONTINUA)
```

# Trabalho

- ▶ Implemente uma classe Lista todas as operações vistas;
- ▶ Implemente a lista usando Templates;
- ▶ Use as melhores práticas de orientação a objetos;
- ▶ Documente todas as classes, métodos e atributos;
- ▶ Aplique os testes unitários disponíveis no moodle da disciplina para validar sua estrutura de dados;
- ▶ Entregue até a data definida no moodle.



# Perguntas????



UNIVERSIDADE FEDERAL  
DE SANTA CATARINA



Este trabalho está licenciado sob uma Licença Creative Commons Atribuição 4.0 Internacional. Para ver uma cópia desta licença, visite

<http://creativecommons.org/licenses/by/4.0/>.



UNIVERSIDADE FEDERAL  
DE SANTA CATARINA