Árvores AVL

Estruturas de Dados I

Departamento de Computação

Universidade Federal de São Carlos (UFSCar)

Sumário

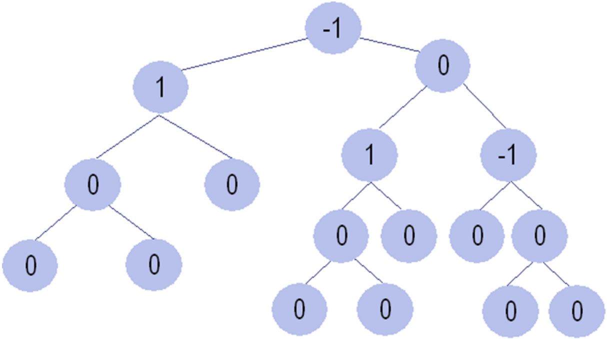
Árvores Binárias de Busca

- Altura de uma árvore binária (AB): igual à profundidade, ou nível máximo, de suas folhas
- A eficiência da busca em árvore depende do seu balanceamento
- Algoritmos de inserção e remoção em ABB não garantem que a árvore gerada a cada passo seja balanceada

Árvores AVL

- Árvore AVL: ABB na qual as alturas das duas sub-árvores de todo nó nunca diferem em mais de 1
  - Fator de balanceamento de nó: a altura de sua sub-árvore esquerda menos a altura de sua sub-árvore direita
    - $FB(p) = h(T_E(p)) - h(T_D(p))$
  - Em uma árvore AVL todo nó tem fator de balanceamento igual a 1, -1 ou 0

Árvores AVL

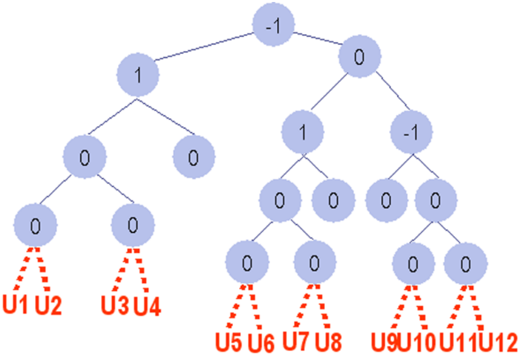


Árvores AVL

- O problema das árvores AVL e das árvores balanceadas de uma forma geral é como manter a estrutura balanceada após operações de inserção e remoção
- As operações de inserção e remoção sobre ABBs não garantem o balanceamento

Árvores AVL

- As seguintes inserções tornam a árvore desbalanceada



Árvores AVL

- As seguintes situações podem levar ao desbalaceamento de uma árvore AVL
  - O nó inserido é descendente esquerdo de um nó que tinha $FB = 1$ ($U1$ e $U8$)
  - O nó inserido é descendente direito de um nó que tinha $FB = -1$ ($U9$ e $U12$)

Árvores AVL

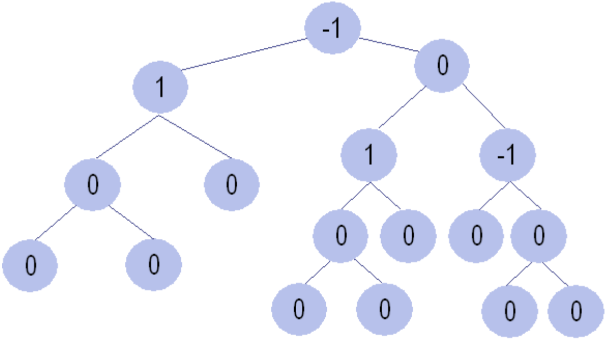
- Para manter uma árvore balanceada é necessário aplicar uma transformação na árvore tal que
  1. O percurso em-ordem na árvore transformada seja igual ao da árvore original (isto é, a árvore transformada continua sendo uma ABB)
  2. A árvore transformada fique balanceada

Árvores AVL

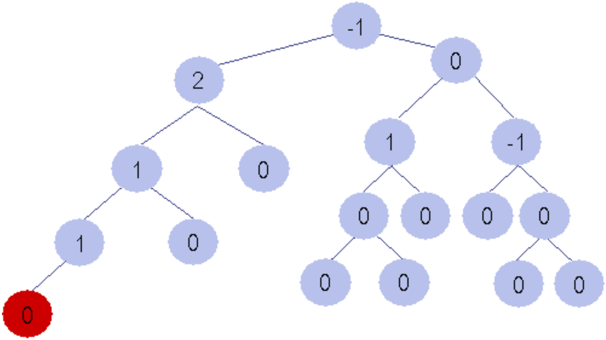
- A transformação que mantém a árvore balanceada é chamada de **rotação**
- A rotação pode ser feita à esquerda ou à direita, dependendo do desbalanceamento a ser tratado
- A rotação deve ser realizada de maneira a respeitar as regras 1 e 2 definidas no slide anterior
- Dependendo do desbalanceamento a ser tratado, uma única rotação pode não ser suficiente

Sumário

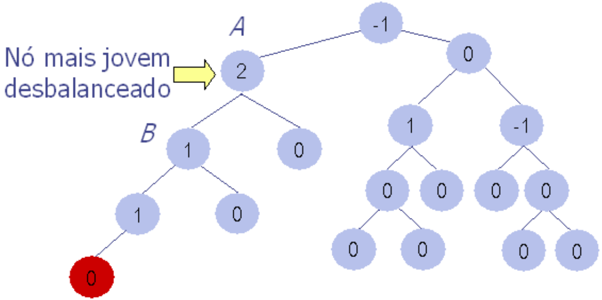
Árvores AVL - Rotação Direita



Árvores AVL - Rotação Direita



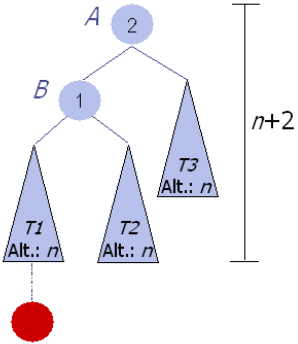
Árvores AVL - Rotação Direita



- A rotação direita consiste em subir o nó B para o lugar de A. A desce para ser sub-árvore direita de B

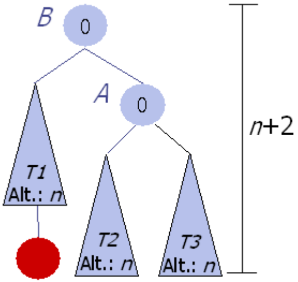
Árvores AVL - Rotação Direita

- A rotação direita tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado

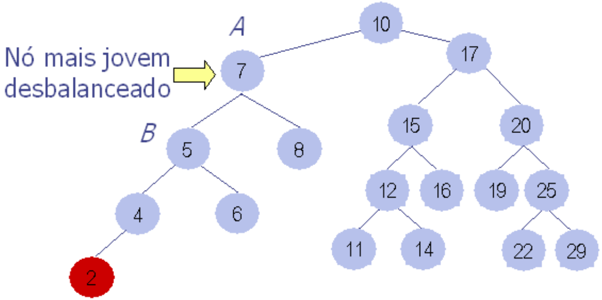


Árvores AVL - Rotação Direita

- A rotação direita tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado

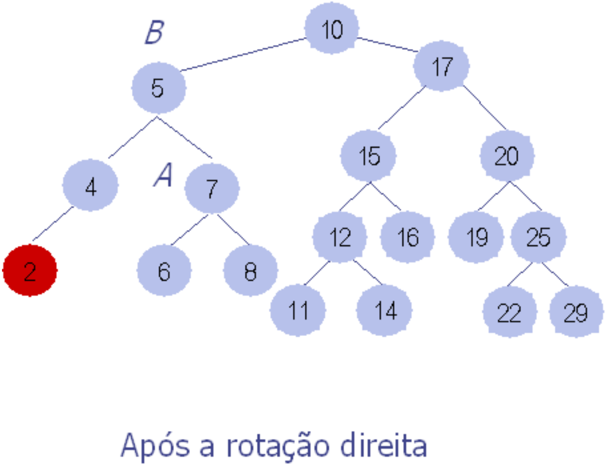


Árvores AVL - Rotação Direita



Antes da rotação direita

Árvores AVL - Rotação Direita



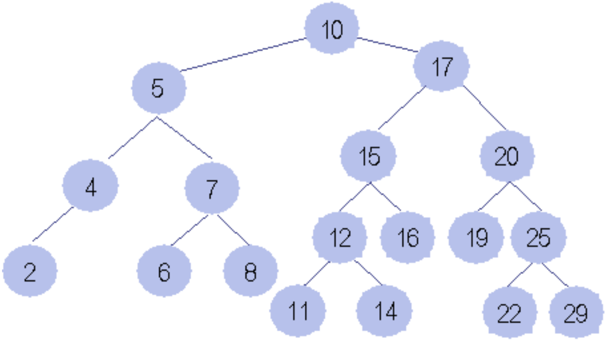
Árvores AVL - Rotação Direita

Exercício

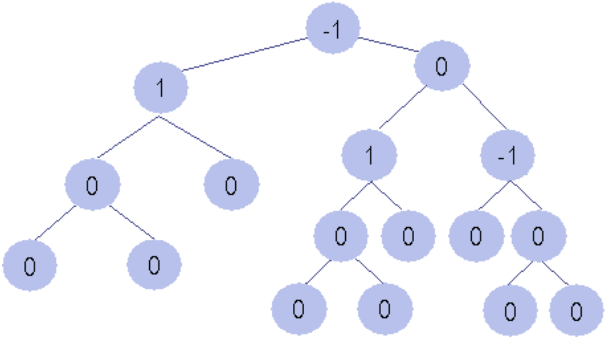
- Insira em uma árvore AVL a seqüência de valores: $5, 4, 3, 2, 1$. Na ordem que os valores foram listados

Sumário

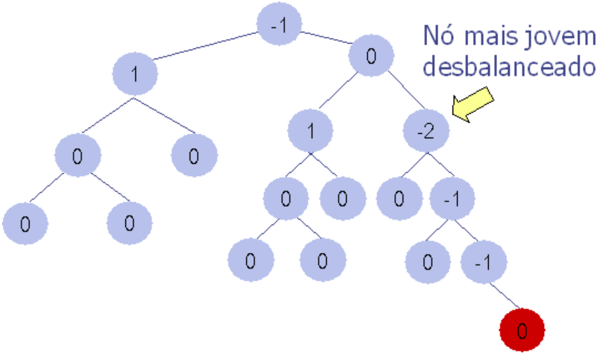
Árvores AVL - Rotação Esquerda



Árvores AVL - Rotação Esquerda

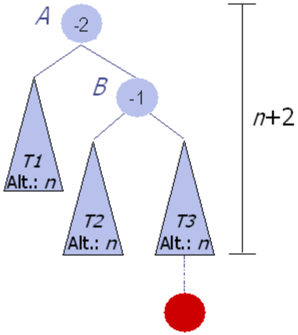


Árvores AVL - Rotação Esquerda



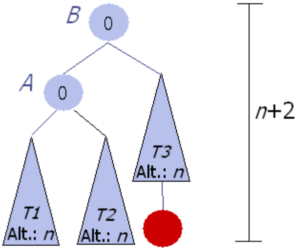
Árvores AVL - Rotação Esquerda

- A rotação esquerda tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado

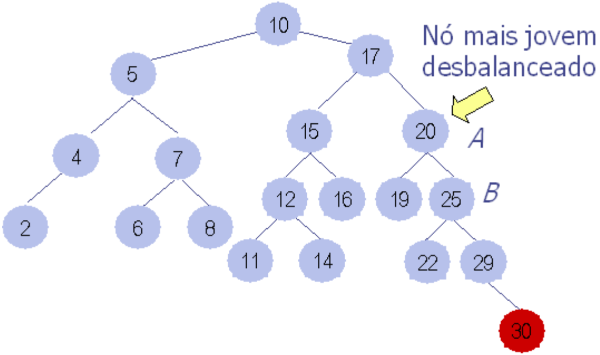


Árvores AVL - Rotação Esquerda

- A rotação esquerda tem formato geral ilustrado à direita
- $T1$, $T2$ e $T3$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado

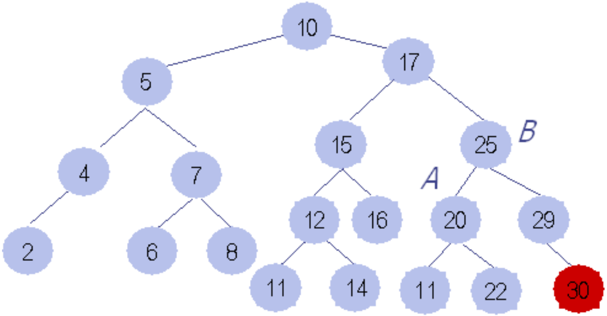


Árvores AVL - Rotação Esquerda



Antes da rotação esquerda

Árvores AVL - Rotação Esquerda



Após a rotação esquerda

Árvores AVL - Rotação Esquerda

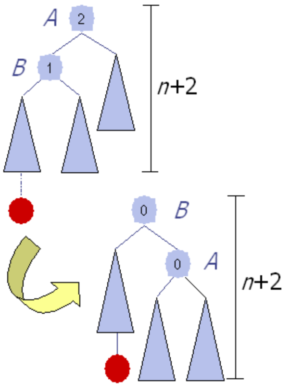
Exercício

- Insira em uma árvore AVL a seqüência de valores: $1, 2, 3, 4, 5$. Na ordem que os valores foram listados

Sumário

Rotações Simples

- Tanto para a rotação direita quanto para a rotação esquerda, a sub-árvore resultante tem como altura a mesma altura a sub-árvore original
- Isso significa que o fator de balanceamento de nenhum nó acima de A é afetado

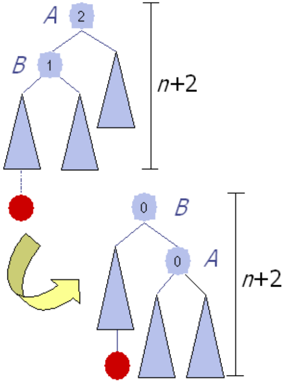


Rotações Simples

- Quando se deve utilizar a rotação direita ou esquerda?

Rotações Simples

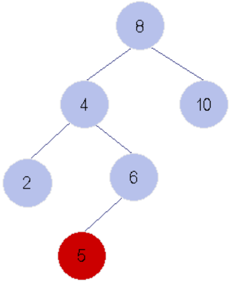
- Quando se deve utilizar a rotação direita ou esquerda?
  - Quando o fator de balanceamento do nó A é positivo, a rotação é direita. Se for negativo a rotação é esquerda



Sumário

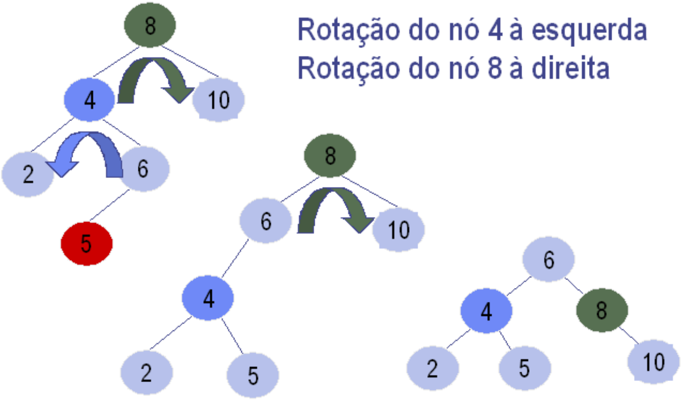
Rotações Duplas

- Será que as rotações simples solucionam todos os tipos de desbalanceamento?
  - Infelizmente, não
- Existem situações nas quais é necessário uma rotação dupla



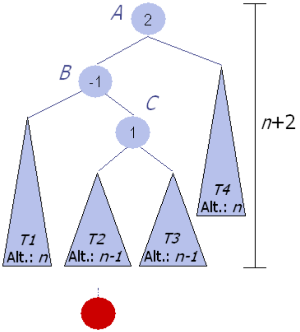
Rotações Duplas

Rotação do nó 4 à esquerda
Rotação do nó 8 à direita



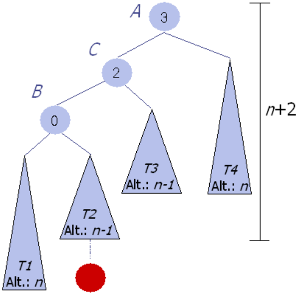
Árvores AVL - Rotação Esq./Dir.

- A rotação dupla esquerda/direita tem formato geral ilustrado à direita
- $T1$, $T2$, $T3$ e $T4$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado



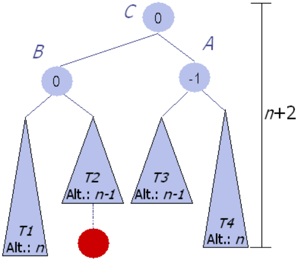
Árvores AVL - Rotação Esq./Dir.

- Passo 1: rotação esquerda em B
- A princípio a rotação esquerda parece deixar a árvore ainda mais desbalanceada
- Entretanto...



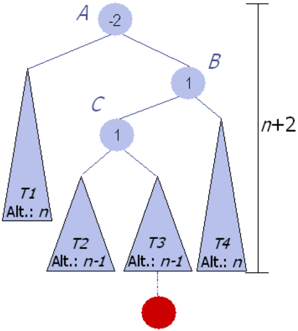
Árvores AVL - Rotação Esq./Dir.

- Passo 2: rotação direita em A
- Repare que a altura final da sub-árvore é $n+2$
- Funciona também se o novo nó tivesse sido inserido em $T3$



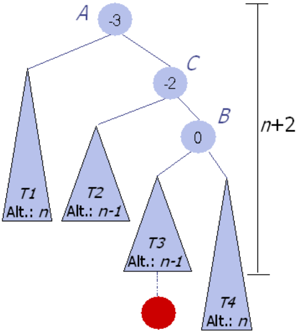
Árvores AVL - Rotação Esq./Dir.

- A rotação dupla direita/esquerda tem formato geral ilustrado à direita
- $T1$, $T2$, $T3$ e $T4$ podem ser sub-árvores de qualquer tamanho, inclusive 0
- A é o nó mais jovem a se tornar desbalanceado



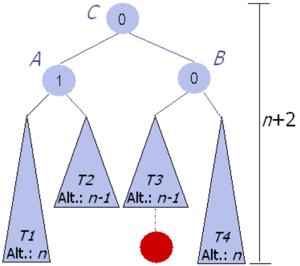
Árvores AVL - Rotação Dir./Esq.

- Passo 1: rotação direita em B
- A princípio a rotação direita parece deixar a árvore ainda mais desbalanceada
- Entretanto...



Árvores AVL - Rotação Dir./Esq.

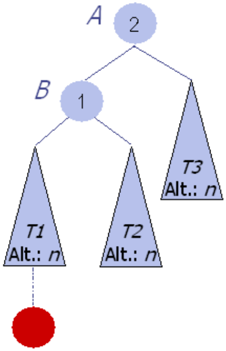
- Passo 2: rotação esquerda em A
- Repare que a altura final da sub-árvore é $n+2$
- Funciona também se o novo nó tivesse sido inserido em $T2$



Sumário

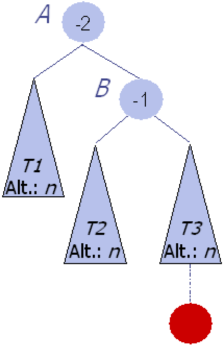
Como decidir qual rotação usar?

- Se o **sinal** do nó A e do nó B forem **iguais** então a rotação é **simples**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **positivo**, então a rotação é **direita**



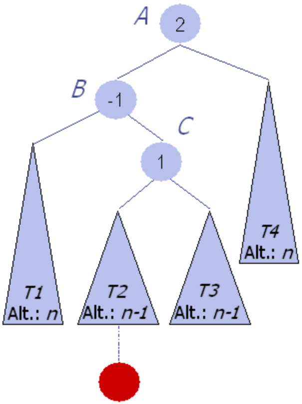
Como decidir qual rotação usar?

- Se o **sinal** do nó A e do nó B forem **iguais** então a rotação é **simples**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **negativo**, então a rotação é **esquerda**



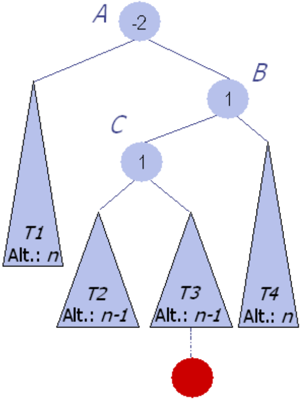
Como decidir qual rotação usar?

- Se o *sinal* do nó A e do nó B forem **diferentes** então a rotação é **dupla**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **positivo**, então a rotação é **esquerda/direita**



Como decidir qual rotação usar?

- Se o **sinal** do nó A e do nó B forem **diferentes** então a rotação é **dupla**
- Se o fator de balanceamento nó A (nó mais jovem a se tornar desbalanceado) for **negativo**, então a rotação é **direita/esquerda**



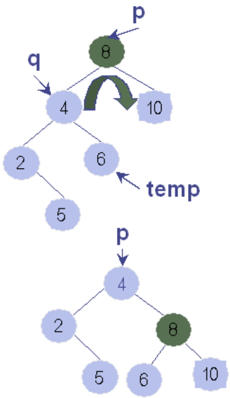
Sumário

Definição de Tipos

```
typedef struct {
  int chave;
  int valor;
} INFO;

typedef struct NO {
  INFO info;
  int fb; //fator de balanceamento
  struct NO *pai; //ponteiro para o pai
  struct NO *fesq; //ponteiro para o filho da esquerda
  struct NO *fdir; //ponteiro para o filho da direita
} NO;

typedef struct {
  NO *raiz;
} ARVORE_AVL;
```

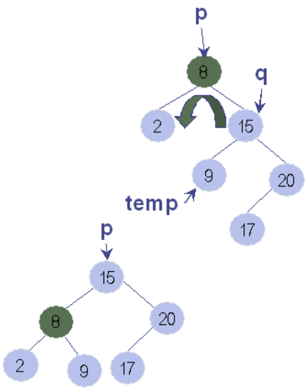
Algoritmo - Rotação Direita

```
NO *rot_dir(NO *no) {
  NO *aux = no->fesq;

  if (no->pai) { //verifica se no não é a raiz
    if (no->pai->fesq == no)
      no->pai->fesq = aux;
    else
      no->pai->fdir = aux;
  }

  aux->pai = no->pai;
  no->fesq = aux->fdir;
  if (no->fesq) no->fesq->pai = no;
  aux->fdir = no;
  no->pai = aux;

  return(aux);
}
```



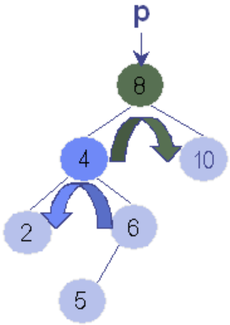
Algoritmo - Rotação Esquerda

```
NO *rot_esq(NO *no) {
  NO *aux = no->fdir;

  if (no->pai) { //verifica se no não é a raiz
    if (no->pai->fesq == no)
      no->pai->fesq = aux;
    else
      no->pai->fdir = aux;
  }

  aux->pai = no->pai;
  no->fdir = aux->fesq;
  if (no->fdir) no->fdir->pai = no;
  aux->fesq = no;
  no->pai = aux;

  return(aux);
}
```



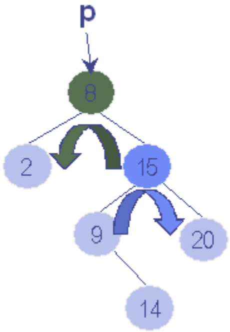
Algoritmo - Rotação Esq./Dir.

```
NO *rot_esq_dir(NO *no) {
  rot_esq(no->fesq);
  return(rot_dir(no));
}
```



Algoritmo - Rotação Dir./Esq.

```
NO *rot_dir_esq(NO *no) {
  rot_dir(no->fdir);
  return(rot_esq(no));
}
```



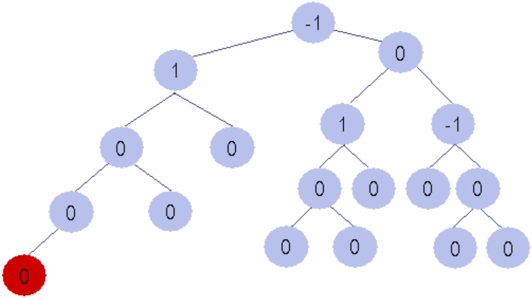
Sumário

Algoritmo de Inserção

- Utilizando as rotinas de rotação pode-se definir um algoritmo de inserção em árvores AVL
- Nessa operação é importante saber
  - O balanceamento de cada nó da árvore
  - O nó ancestral mais jovem do nó inserido que pode se tornar desbalanceado
  - A inserção é feita em dois passos: o primeiro é uma inserção em ABBs e o segundo é o rebalanceamento, se necessário

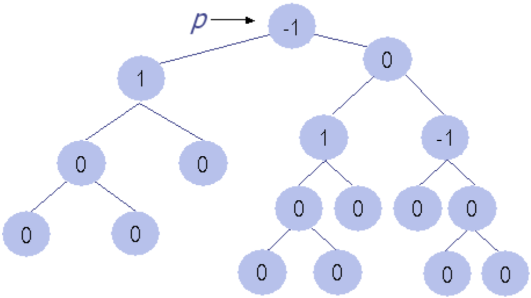
Algoritmo de Inserção

- Vamos supor que um novo nó será inserido na posição marcada em vermelho



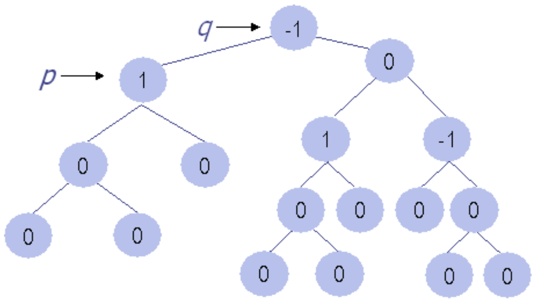
Algoritmo de Inserção

- Um ponteiro p marca a posição que se está procurando...



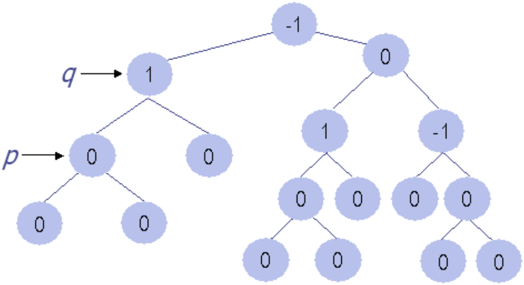
Algoritmo de Inserção

- E q aponta para o ancestral mais jovem que possui balanceamento diferente de 0...



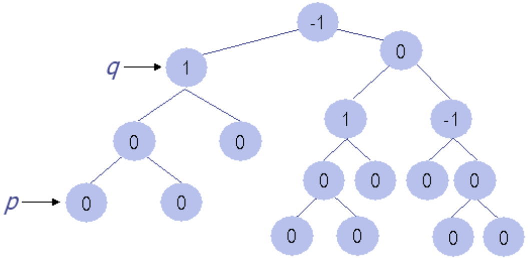
Algoritmo de Inserção

- E q aponta para o ancestral mais jovem que possui balanceamento diferente de 0...



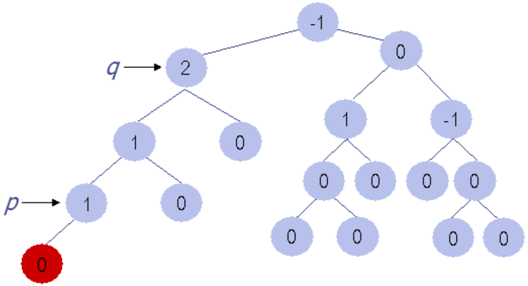
Algoritmo de Inserção

- E q aponta para o ancestral mais jovem que possui balanceamento diferente de 0...



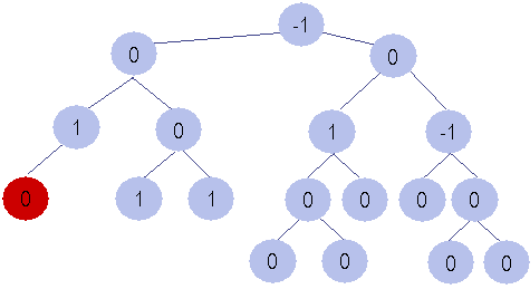
Algoritmo de Inserção

- O balanceamento entre q e p é atualizado...



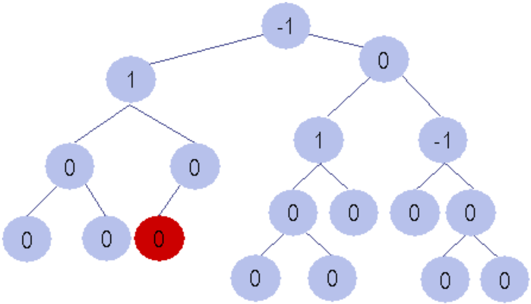
Algoritmo de Inserção

- A rotação apropriada é realizada, os fatores de balanceamento são atualizados



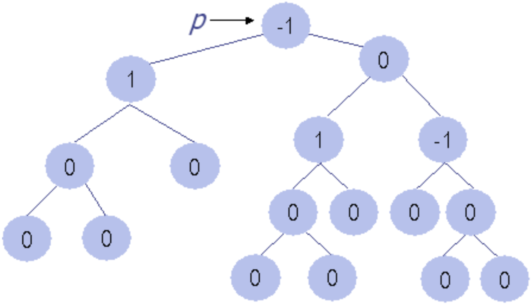
Algoritmo de Inserção

- Vamos supor que um novo nó será inserido na posição marcada em vermelho



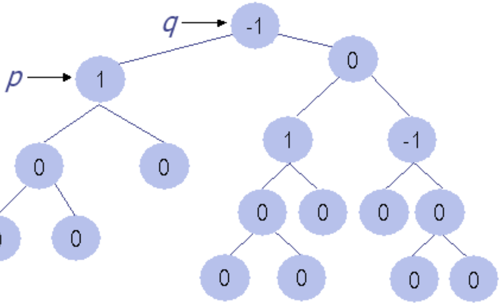
Algoritmo de Inserção

- Um ponteiro p marca a posição que se está procurando...



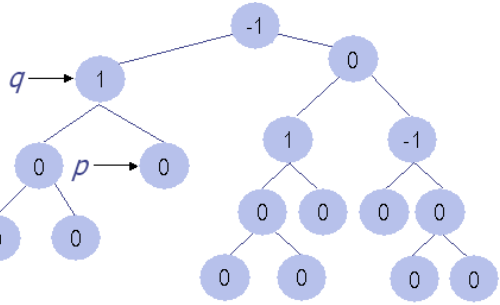
Algoritmo de Inserção

- E q aponta para o ancestral mais jovem que possui balanceamento diferente de 0...



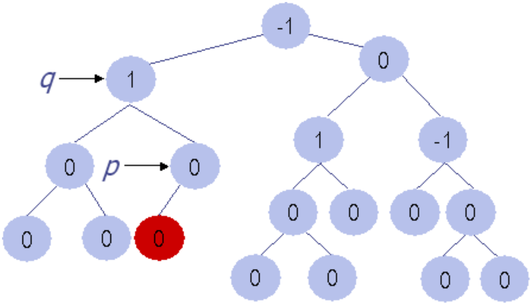
Algoritmo de Inserção

- E q aponta para o ancestral mais jovem que possui balanceamento diferente de 0...



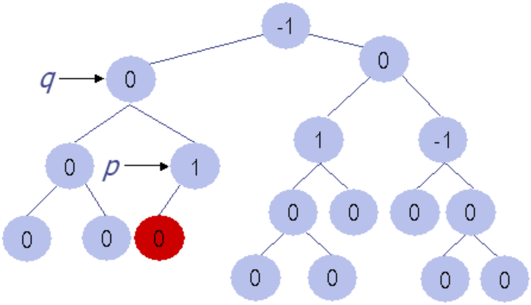
Algoritmo de Inserção

- E q aponta para o ancestral mais jovem que possui balanceamento diferente de 0...



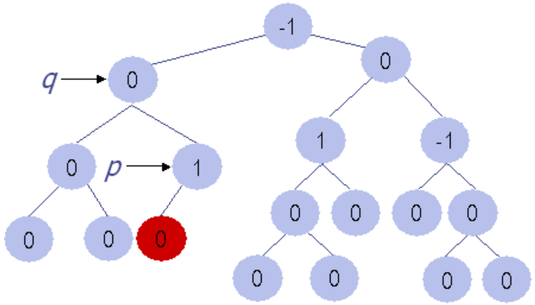
Algoritmo de Inserção

- O balanceamento entre q e p é atualizado...



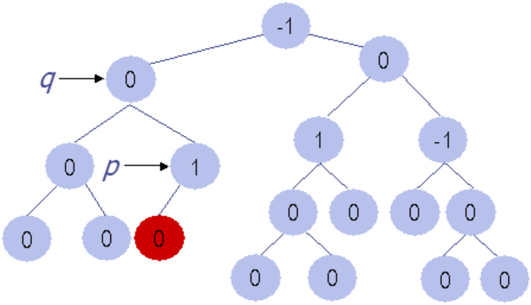
Algoritmo de Inserção

- Não há necessidade de ajustar o fator de balanceamento acima de q (porque?)



Algoritmo de Inserção

- Como não houve desbalanceamento, o algoritmo termina



Algoritmo de Inserção

```
void inserir_arvore_avl(ARVORE_AVL *arv, INFO info) {
  arv->raiz = inserir_arvore_avl_aux(arv->raiz, info);
}

struct NO *inserir_arvore_avl_aux(NO *raiz, INFO info) {
  if (raiz == NULL) {
    raiz = (NO *) malloc(sizeof(NO));
    raiz->fesq = raiz->fdir = raiz->pai = NULL;
    raiz->info = info;
    raiz->fb = 0;
  } else {
    if (raiz->info.chave > info.chave) { //desce pela esquerda
      raiz->fesq = inserir_arvore_avl_aux(raiz->fesq, info);
      raiz->fesq->pai = raiz;

      //adicionar as rotações...
    } else { //desce pela direita
      raiz->fdir = inserir_arvore_avl_aux(raiz->fdir, info);
      raiz->fdir->pai = raiz;

      //adicionar as rotações...
    }
  }

  return(raiz);
}
```

Algoritmo de Inserção

```c
void inserir_arvore_avl(ARVORE_AVL *arv, INFO info) {
  int atualiza_fb = 0;
  arv->raiz = inserir_arvore_avl_aux(arv->raiz, info, &atualiza_fb);
}

struct NO *inserir_arvore_avl_aux(NO *raiz, INFO info, int *atualiza_fb) {
  if (raiz == NULL) {
    //...
    *atualiza_fb = 1; //inseriu, atualiza os fbs
  } else {
    if (raiz->info.chave > info.chave) { //desce pela esquerda
      raiz->fesq = inserir_arvore_avl_aux(raiz->fesq, info, atualiza_fb);
      raiz->fesq->pai = raiz;

      if (*atualiza_fb) {
        //...
      }
    } else { //desce pela direita
      raiz->fdir = inserir_arvore_avl_aux(raiz->fdir, info, atualiza_fb);
      raiz->fdir->pai = raiz;

      if (*atualiza_fb) {
        //...
      }
    }
  }

  return(raiz);
}
```

Remoção em AVLs

- Para eliminar um nó de uma árvore AVL, o algoritmo é um pouco mais complicado
- Enquanto que a inserção pode requerer no máximo uma rotação (simples ou dupla), a remoção pode requerer mais de uma rotação
    - No pior caso, pode-se fazer uma rotação a cada nível da árvore
    - Ou seja, no pior caso O($\log\ n$) rotações
    - Na prática, são necessárias apenas $0,214$ rotação por eliminação, em média

Complexidade das AVLs

- A altura máxima de uma ABB AVL é $1,44\ \log_2\ n$
  - Dessa forma, uma pesquisa nunca exige mais do que 44% mais comparações que uma ABB totalmente balanceada.
- Na prática, para n grande, os tempos de busca são por volta de $\log_2\ n\ +\ 0,25$
- Na média, é necessária uma rotação em $46,5\%$ das inserções

Exercícios

- Simule a inserção da seguinte seqüência de valores em uma árvore AVL: $10, 7, 20, 15, 17, 25, 30, 5, 1$
- Em cada opção abaixo, insira as chaves na ordem mostrada de forma a construir uma arvore AVL. Se houver rebalanceamento de nós, mostre qual o procedimento a fazer
  1. a, z, b, y, c, x
  2. $a, z, b, y, c, x, d, w, e, v, f$
  3. $a, v, l, t, r, e, i, o, k$
  4. m, te, a, z, g, p

Exercícios

- Escreva uma função que retorna a altura da árvore AVL. Qual é a complexidade da operação implementada? Ela é mais eficiente que a implementação para ABBs?
- Implemente o TAD AVL com as operações de inserção e busca e demais operações auxiliares

Exercícios

- Mostre a árvores AVL gerada passo-a-passo pelas inserções das seguintes chaves na ordem fornecida
  - $10, 5, 20, 1, 3, 4, 8, 30, 40, 35, 50, 45, 55, 51, 100$

Leitura complementar e Créditos

Leitura Complementar

1. *AVL tree*

   `http://en.wikipedia.org/wiki/AVL_tree`

2. *AVL tree - NIST*

   `http://xlinux.nist.gov/dads//HTML/avltree.html`

3. *AVL Trees - Introduction*

   `http://pages.cs.wisc.edu/~ealexand/cs367/NOTES/AVL-Trees/index.html`

4. *AVL Trees - Applet*

   `http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html`

Créditos: aula baseada nos tópicos de aula e slides criados pelo prof. Fernando Vieira Paulovich, publicamente disponíveis em: `http://wiki.icmc.usp.br/index.php/Scc-202(paulovich)`