Tipos Especiais de Listas

Árvores AVL

Sumário

- Implementação

- Remoção em Árvores Binárias de Busca

Relembrando sobre ponteiros

```
#include <stdio.h>
void insere1(int *x){
  int b = 10;
  int *c = &b;
  x = c;
  printf("\nDentro Insere 1 - Valor Apontado %d Valor %p Endereço
%p",*x,x,&x);
}
void insere2(int *x){
  int b = 10;
  int *c = &b;
  *x = *c;
  printf("\nDentro Insere 2 - Valor Apontado %d Valor %p Endereço
%p",*x,x,&x);
}
void insere3(int **x){
   int b = 10;
   int *c = &b;
   *x = c;
   printf("\nDentro Insere 3 - Valor Apontado %d Valor %p Endereço
%p",**x,*x,x);
}
```

```
int main(){
 int *a, b;
 b=2;
 a=&b;
 printf("\nAntes Insere  1 - Valor Apontado %d Valor %p Endereço
%p",*a,a,&a);
 insere1(a);
 printf("\nApos Insere   1 - Valor Apontado %d Valor %p Endereço
%p\n",*a,a,&a);
 b=2;
 a=&b;
 printf("\nAntes Insere  2 - Valor Apontado %d Valor %p Endereço
%p",*a,a,&a);
 insere2(a);
printf("\nApos Insere   2 - Valor Apontado %d Valor %p Endereço
%p\n",*a,a,&a);
 b=2;
 a=&b;
 printf("\nAntes Insere  3 - Valor Apontado %d Valor %p Endereço
%p",*a,a,&a);
 insere3(&a);
 printf("\nApos Insere   3 - Valor Apontado %d Valor %p Endereço
%p\n",*a,a,&a);
return 1;
}
```

Implementação

- Implementação

Relembrando

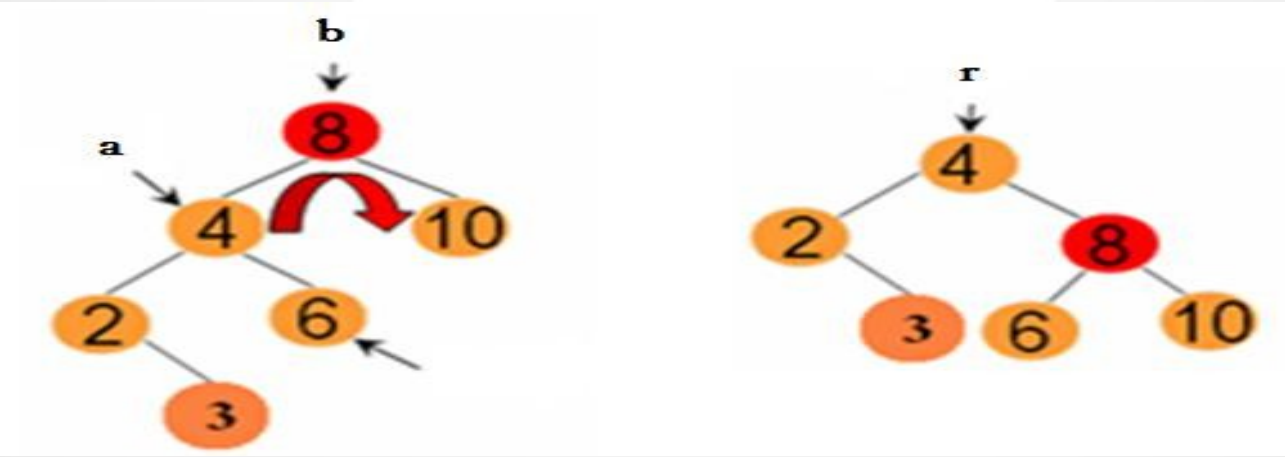
```
typedef struct {
 int chave;
 int valor;
} INFO;
```

```
typedef struct {
   tNO *raiz;
} ARVORE;
```

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

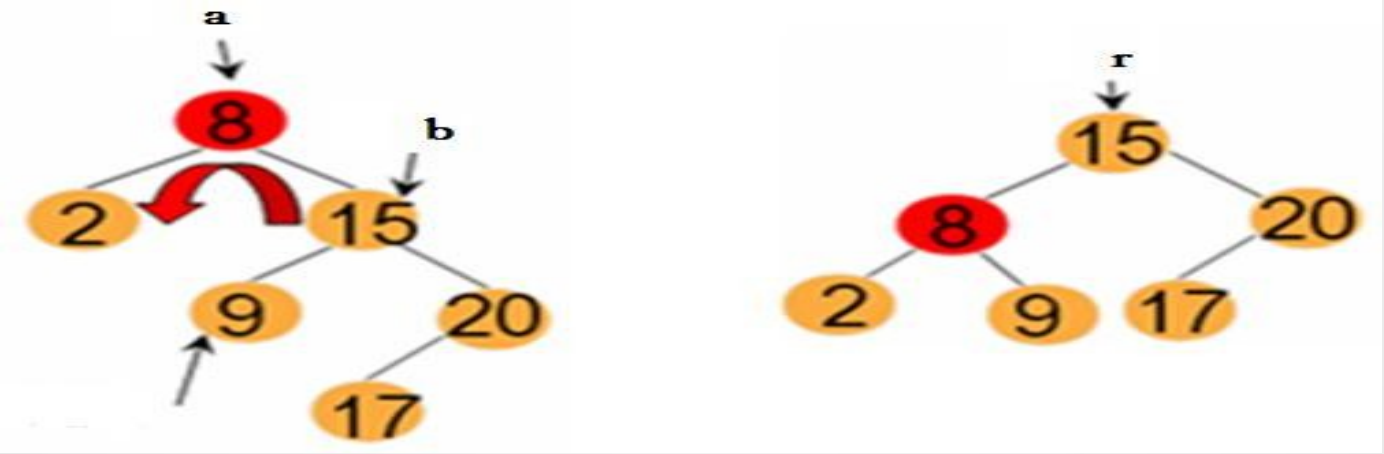
Algoritmo - Rotação Direita

```
void rot_dir(tNO **r) {//realiza rotacao a direita
   tNO *b= *r;
   tNO *a=b->fesq;
   b->fesq=a->fdir;
   a->fdir=b;
   a->fb=0;
   b->fb=0;
   *r=a;//atualiza a raiz
   }
```



Algoritmo - Rotação Esquerda

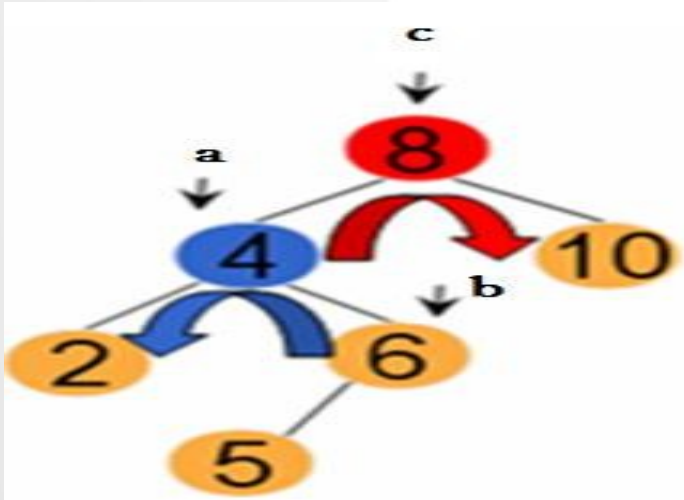
```
void rot_esq(tNO **r) {//realiza rotacao a esquerda
   tNO *a =*r;
   tNO *b=a->fdir;
   a->fdir=b->fesq;
   b->fesq=a;
   a->fb=0;
   b->fb=0;
   *r = b;//atualiza a raiz
   }
```

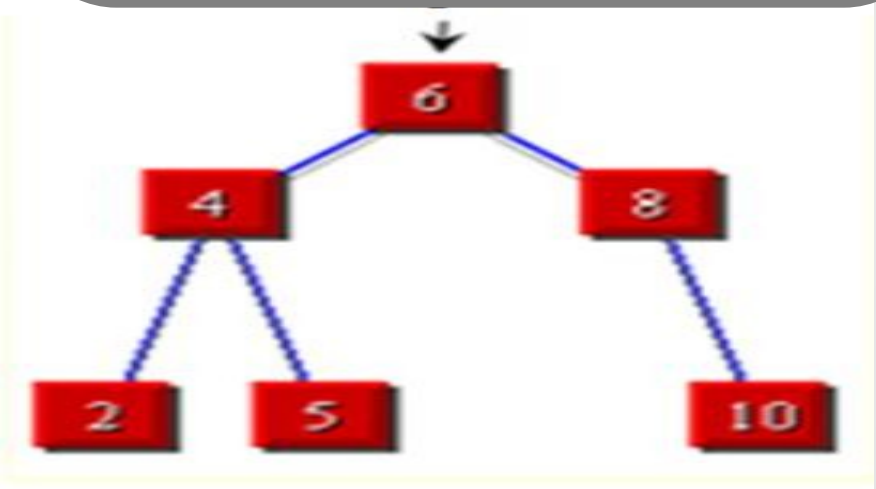


Algoritmo - Rotação Dupla Esquerda/Direita

```
void rot_esq_dir(tNO **r)
{   tNO *c=*r;
   tNO *a=c->fesq;
   tNO *b=a->fdir;
   c->fesq=b->fdir;
   a->fdir=b->fesq;
   b->fesq = a;
   b->fdir=c;
   ...
```

```
switch (b->fb){
   case -1://no a direita
   a->fb=1;c->fb=0;break;
   case 0: //ambos
   a->fb=0;c->fb=0;break;
   case 1://no a esq
   a->fb=0;c->fb=-1;break;
   };
   b->fb=0; *r = b;
   }
```

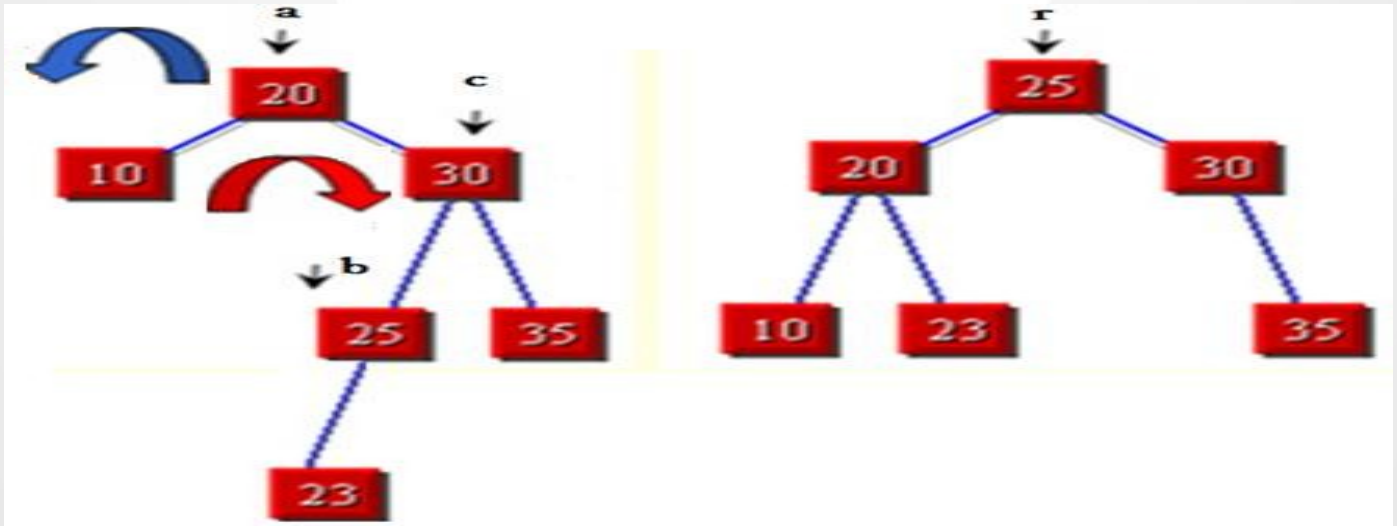




Algoritmo - Rotação Dupla Direita/Esquerda

```
void rot_dir_esq(tNO **r)
{ tNO *a=*r;
  tNO *c=a->fdir;
  tNO *b=c->fesq;
  c->fesq=b->fdir;
  a->fdir=b->fesq;
  b->fesq = a;
  b->fdir=c;
  ...
```

```
switch (b->fb){
   case -1://no a direita
   a->fb=1;c->fb=0;break;
   case 0: //ambos
   a->fb=0;c->fb=0;break;
   case 1://no a esq
   a->fb=0;c->fb=-1;break;
   }
   b->fb=0; *r = b;}
```



Inserções em Árvores AVL

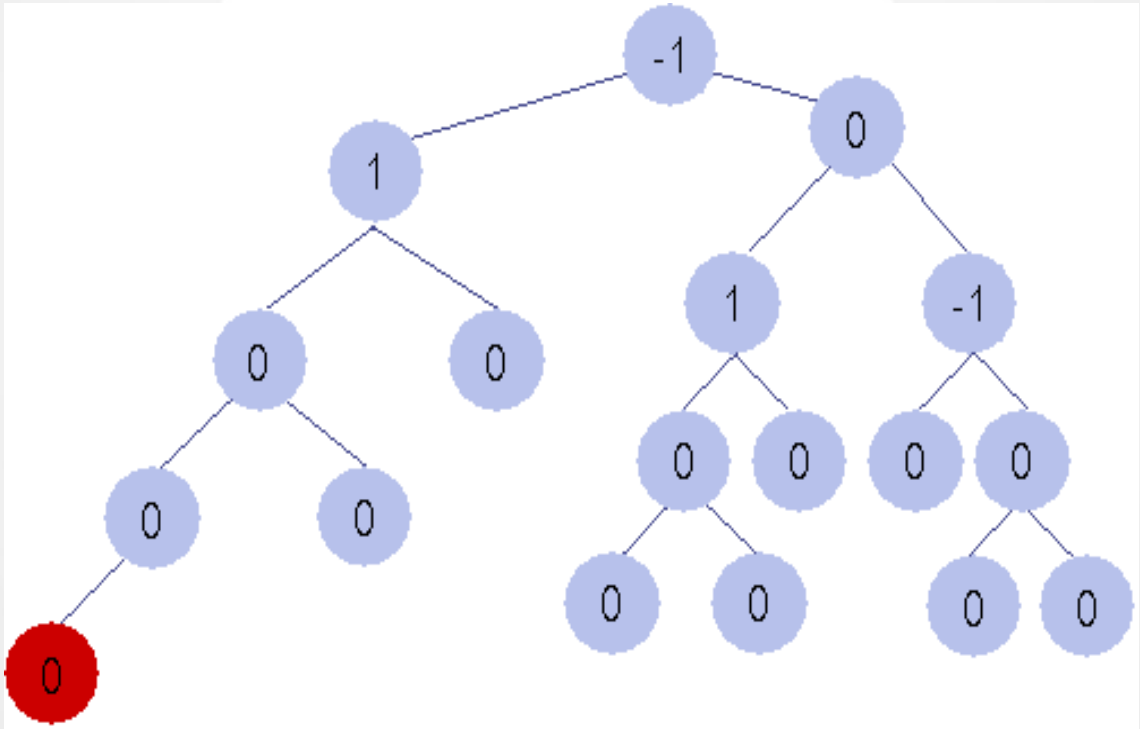
Algoritmo de Inserção

- 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

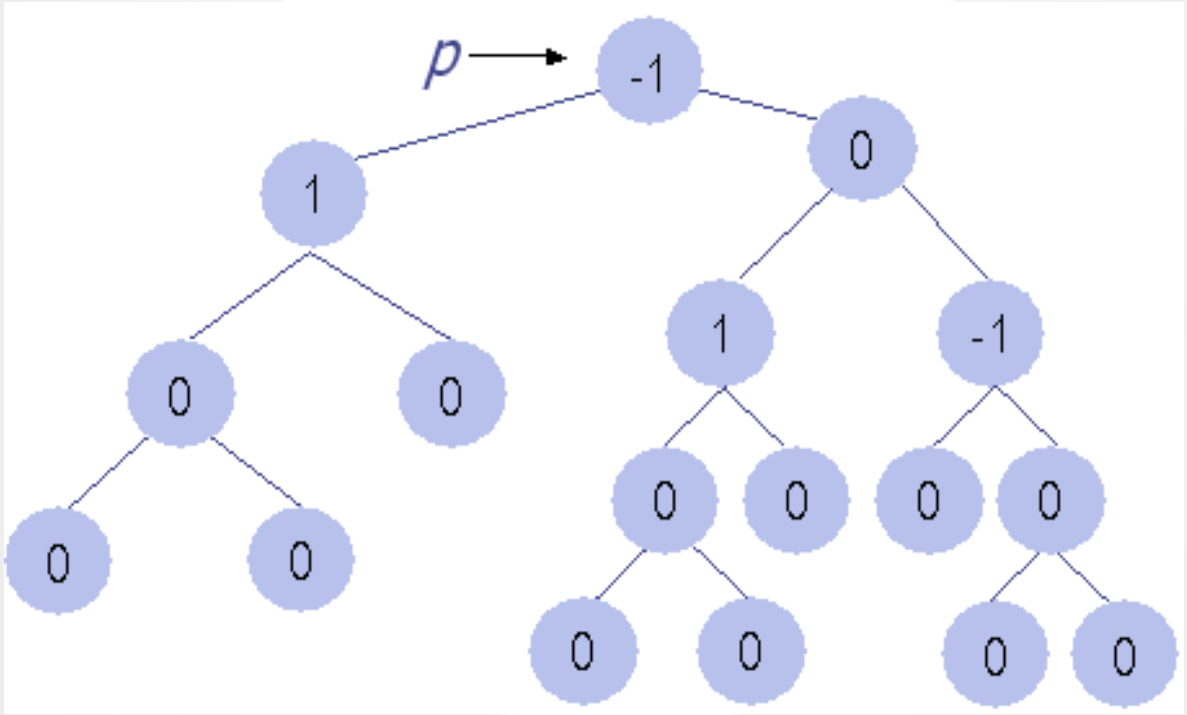
Algoritmo de Inserção

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



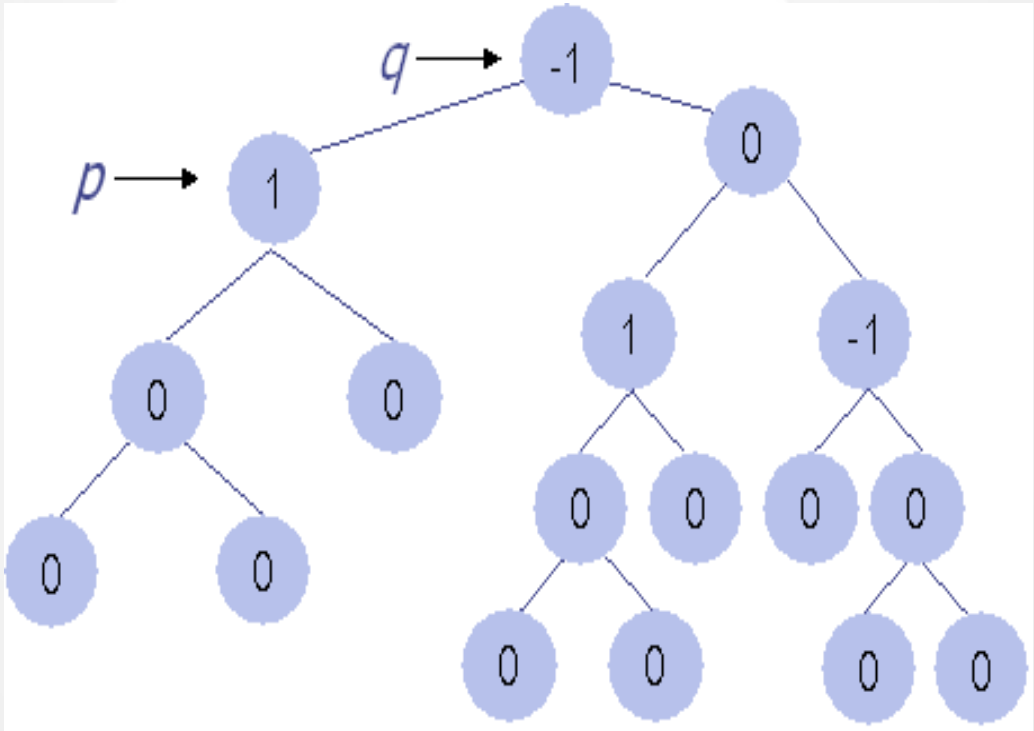
Algoritmo de Inserção

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



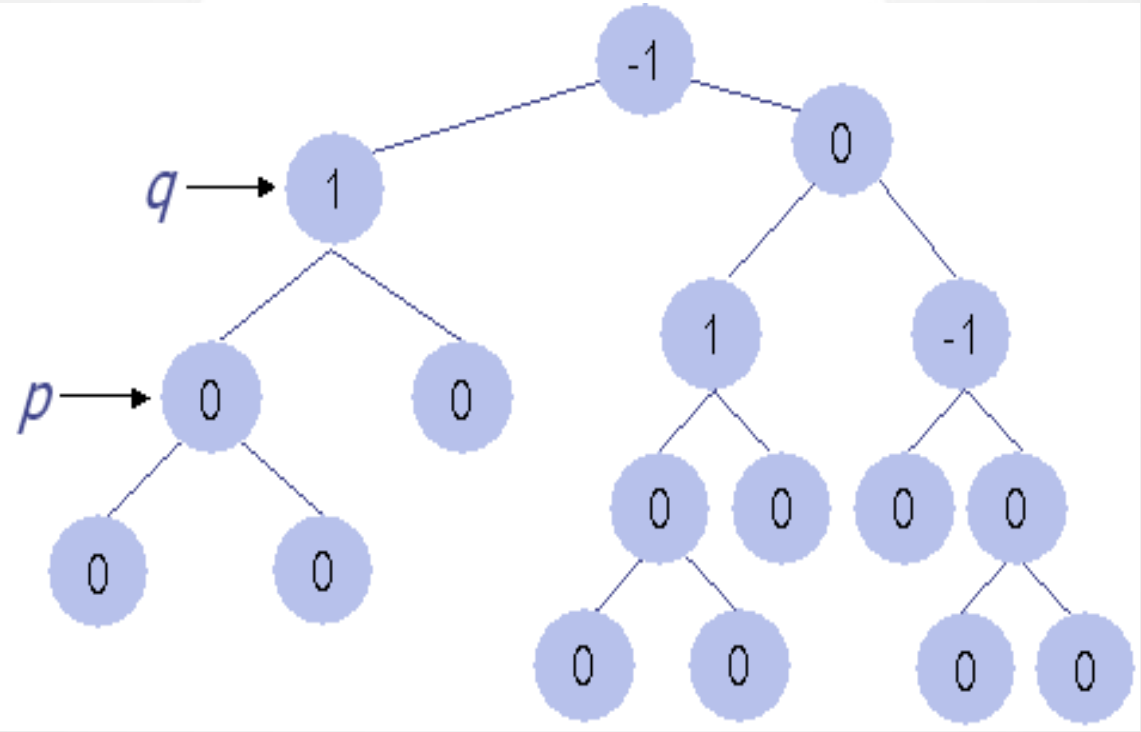
Algoritmo de Inserção

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



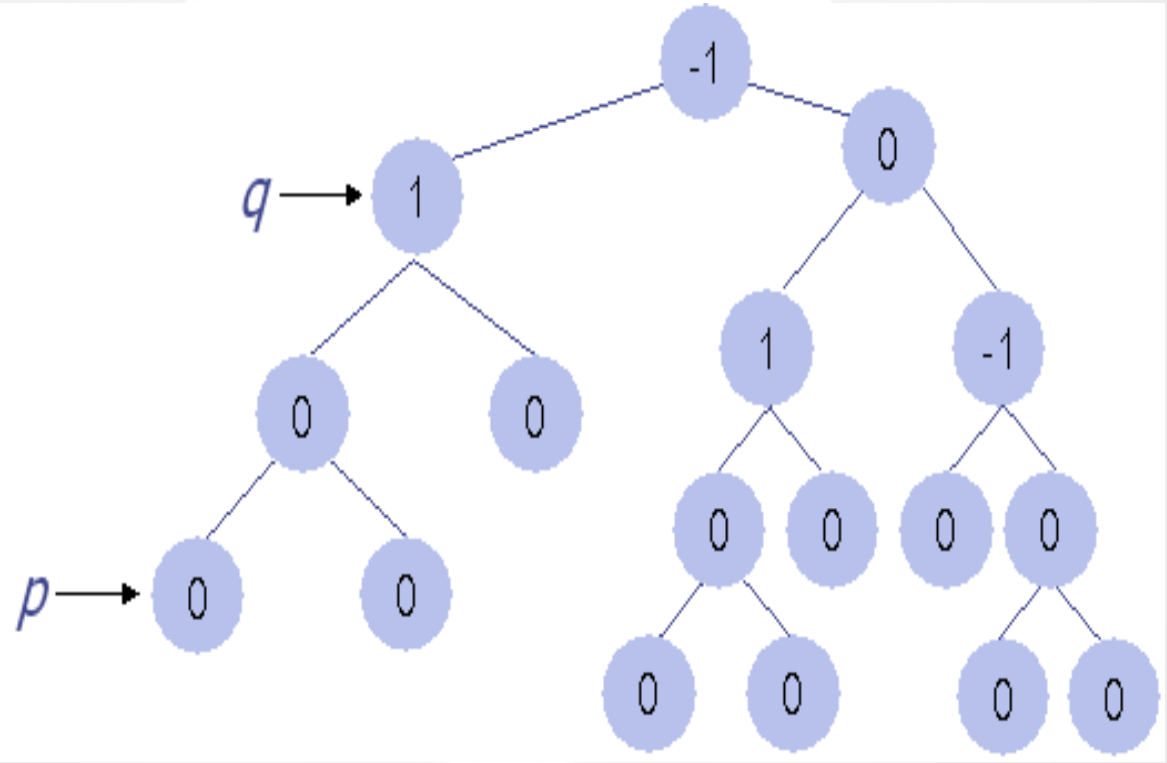
Algoritmo de Inserção

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



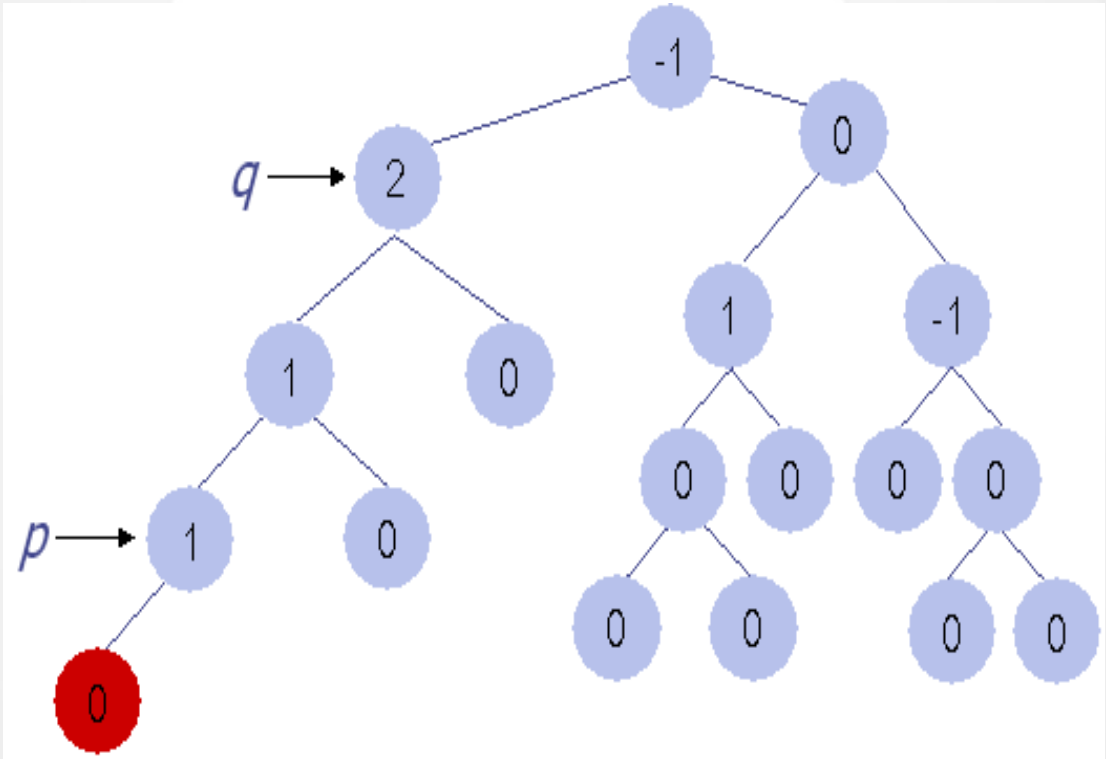
Algoritmo de Inserção

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



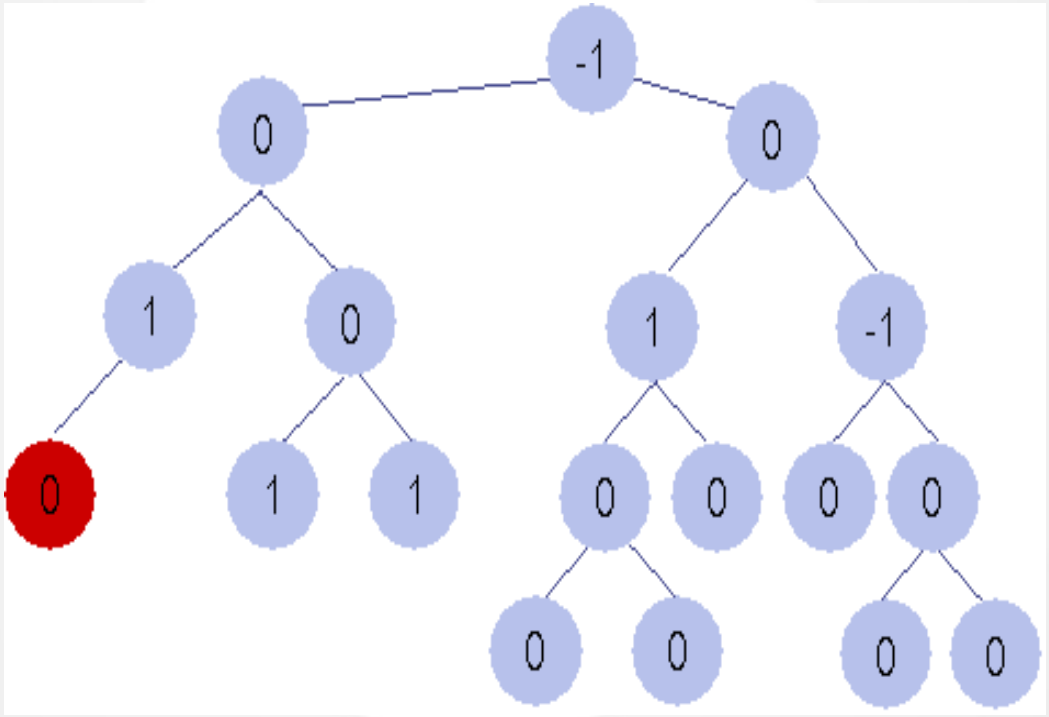
Algoritmo de Inserção

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



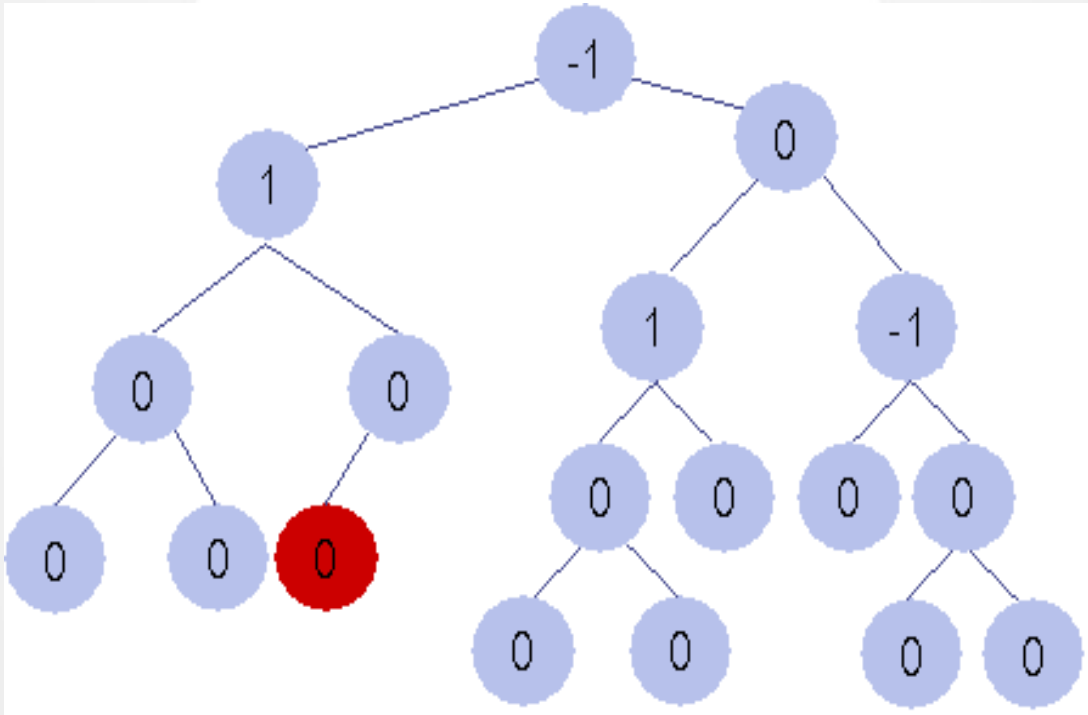
Algoritmo de Inserção

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



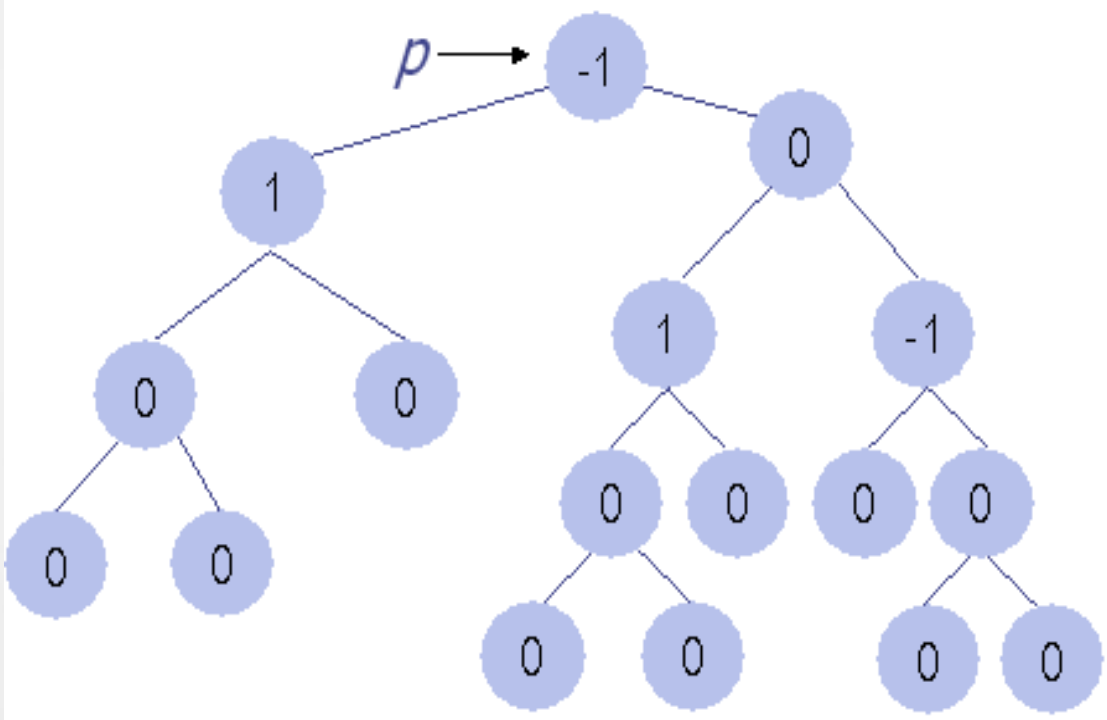
Algoritmo de Inserção

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



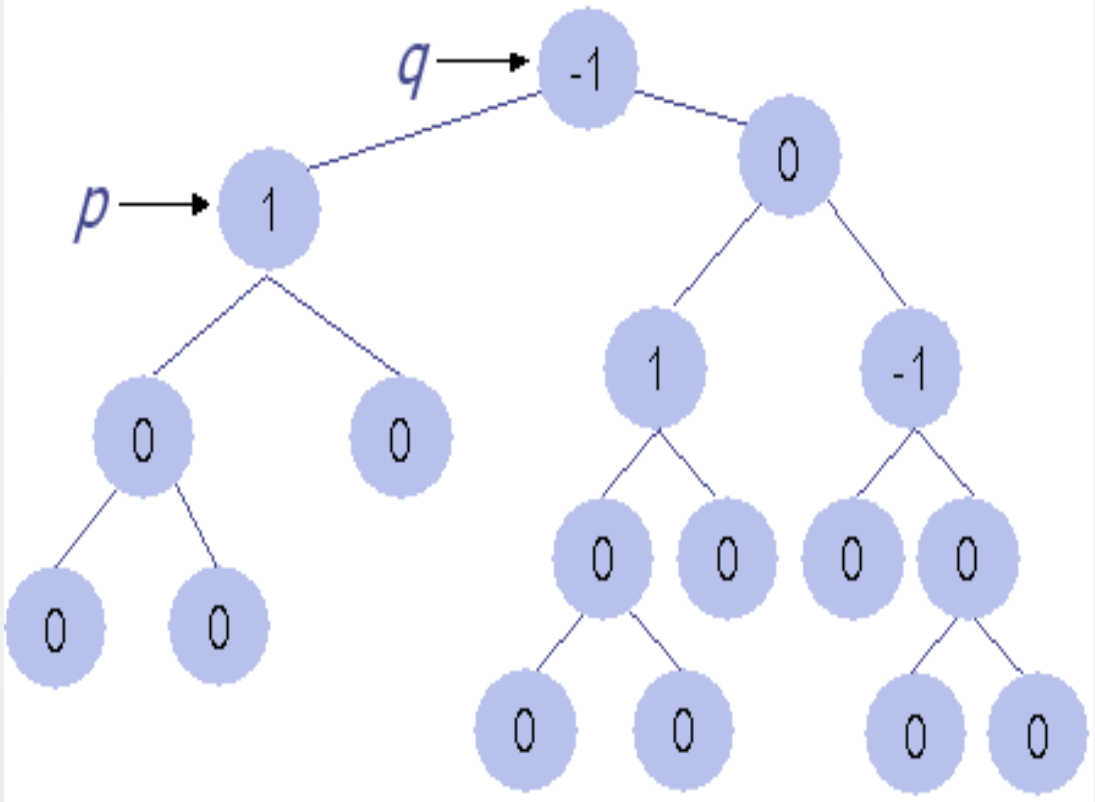
Algoritmo de Inserção

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



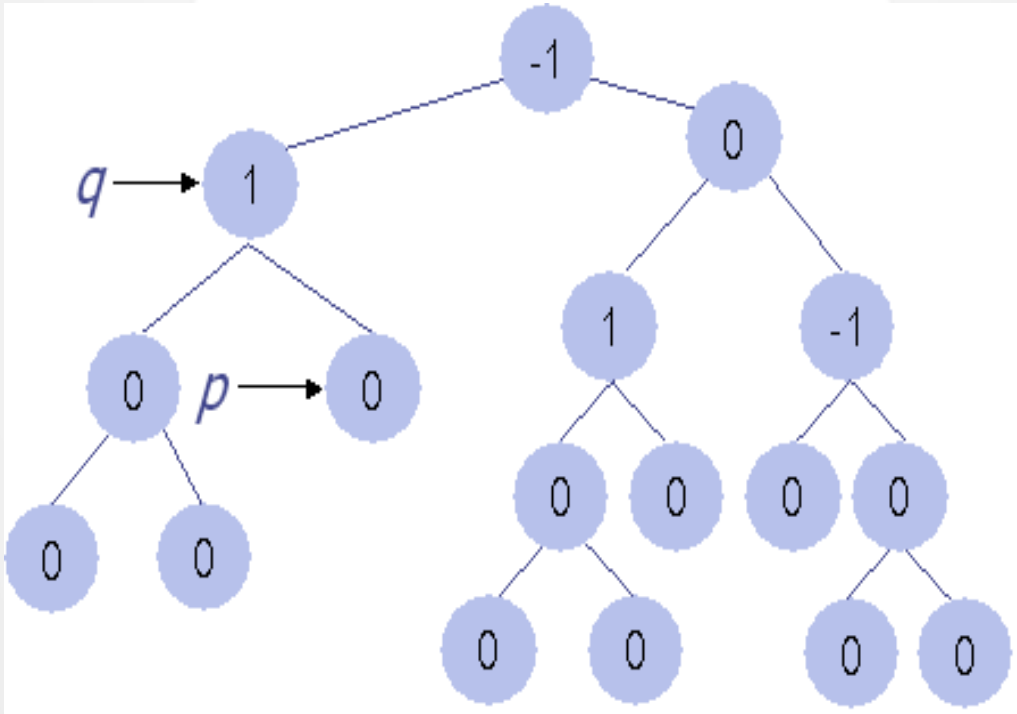
Algoritmo de Inserção

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



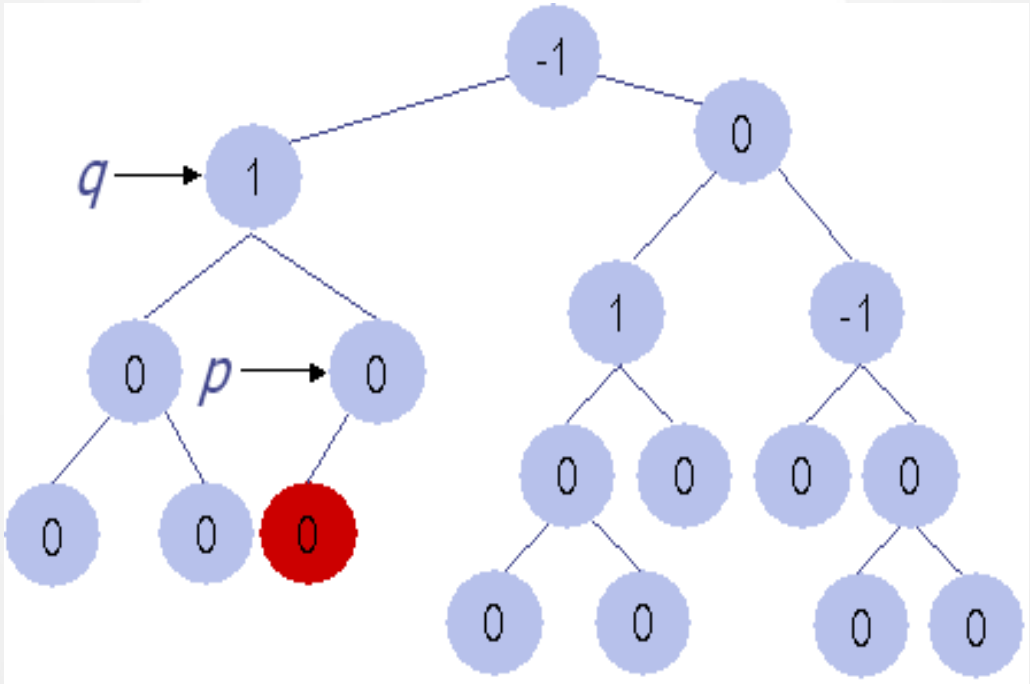
Algoritmo de Inserção

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



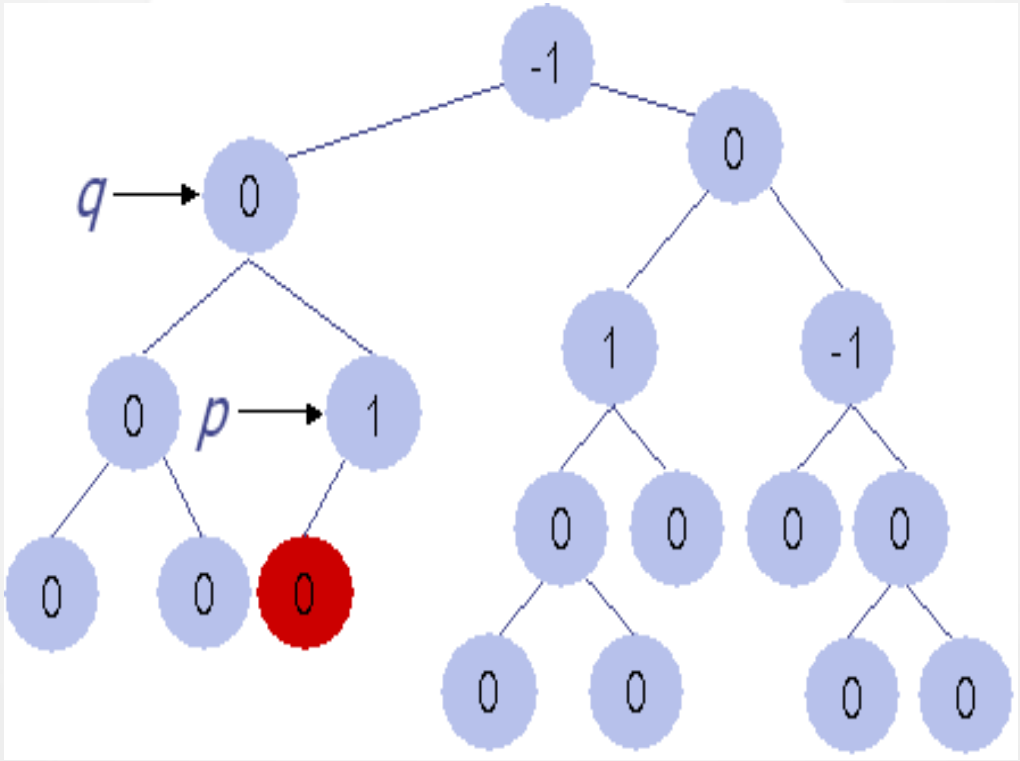
Algoritmo de Inserção

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



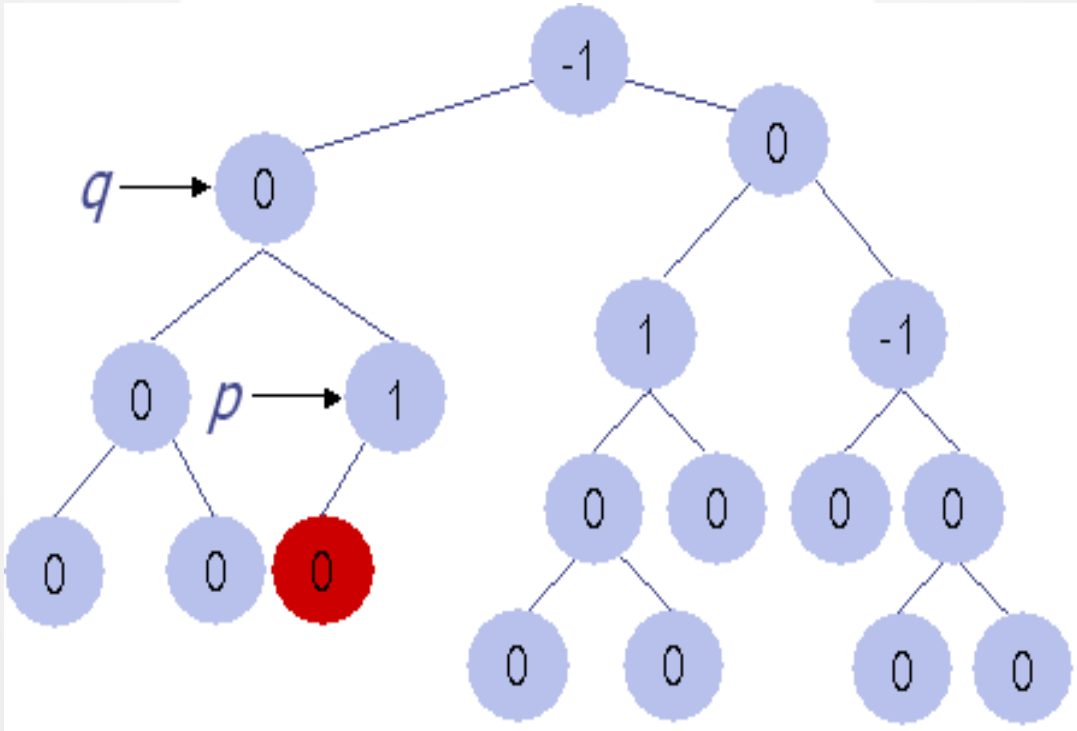
Algoritmo de Inserção

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



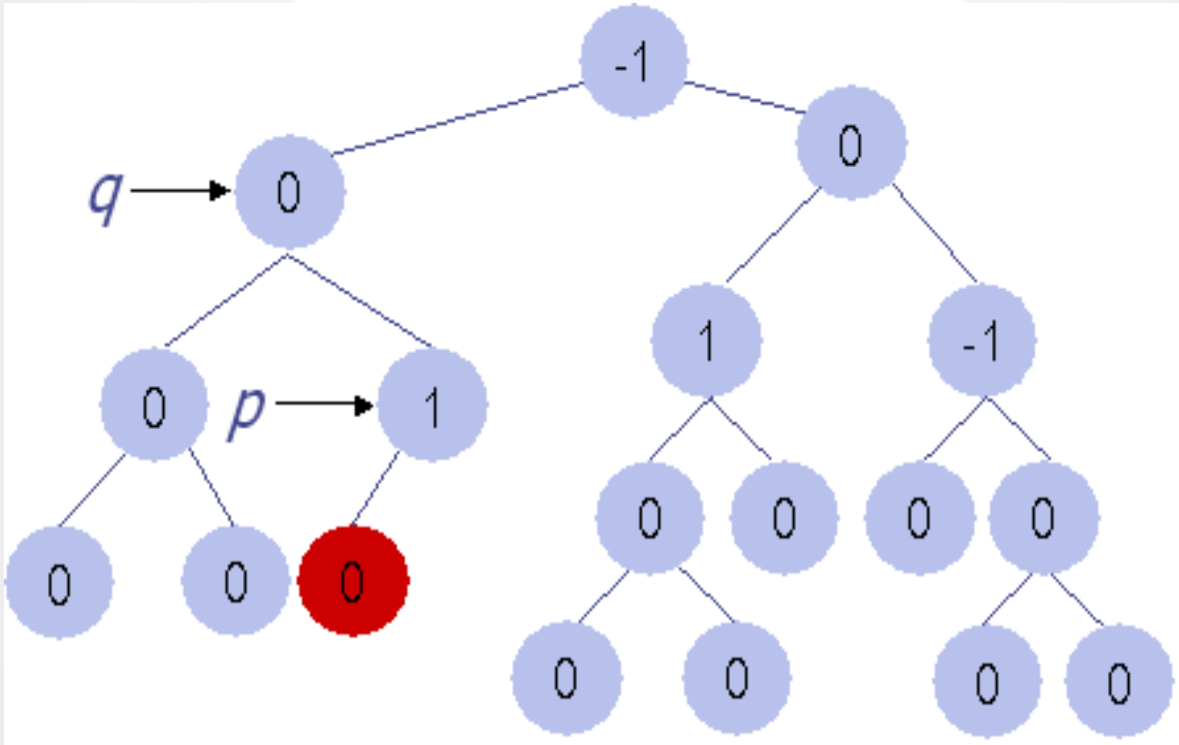
Algoritmo de Inserção

- 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) {
   int mudouH =0;
   arv->raiz = inserir_arvore_avl_aux(arv->raiz,
info,&mudouH);
   }
```

```
tNO *inserir_arvore_avl_aux(tNO *p, INFO info, int
*mudouH) {
 if (p == NULL) {//No folha
   p = (tNO *) malloc(sizeof(tNO));
   if (p==NULL) return NULL; //memoria insuficiente
   else{
     printf("\nInserindo chave %d",info.chave);
     p->fesq = p->fdir =  NULL;//inicializa
     p->info = info;//guarda a informação
     p->fb = 0; //fator de balanceamento
     *mudouH = 1;//indica que a arvore cresceu
     return p;
    } //fecha else
  } else if(p->info.chave == info.chave) //No já existe
        return p;
```

```
else if (p->info.chave>info.chave) {
    p->fesq = inserir_arvore_avl_aux(p->fesq,info,mudouH);
    if (p->fesq!=NULL){//condição para memoria insuficiente
        if (*mudouH){//arvore cresceu
         printf("\nRealizando bal. chave = % d fp %d",p->info.chave,p->fb);
         switch (p->fb){
             case 1: {//BAL(p) sera 2
                if (p->fesq->fb==1){//No a esq
                  printf("\nRealizando Rotação Direita");
                  rot_dir(&p); }//realiza rotação
                 else {
                  printf("\nRealizando Rotação Esquerda/Direita");
                  rot_esq_dir(&p);
                 }
                *mudouH=0; break;
              }
              case 0: {//arvore desbalanceada
                 p->fb = 1;*mudouH =1; break;
             }
              case -1: {//arvore maior a direita
                 p->fb = 0;*mudouH =0; break;
              }
           }//fecha switch
      return p;
      }//fecha if
      else return p; //arvore não cresceu
    }
  else return NULL; //se p->fesq == NULL
  }
```

```
else  {//descendo a direita
    p->fdir = inserir_arvore_avl_aux(p->fdir,info,mudouH);
    if (p->fdir!=NULL){
       if (*mudouH){//arvore cresceu
        printf("\nRealizando bal chave = %d  fp = %d",p->info.chave,p->fb);
             switch (p->fb){
              case 1: {//maior a esquerda
                  p->fb =0;*mudouH=0; break;}
              case 0: {//arvore balanceada
                  p->fb = -1; *mudouH =1; break;}
               case -1: {//Bal(p) = -2 arvore maior a direita
                  if (p->fdir->fb==-1) {
                    printf("\nRealizando Rotação Esquerda");
                    rot_esq(&p);}
                  else{
                    rot_dir_esq(&p);
                    printf("\nRealizando Rotação Direita/Esquerda");}
                    *mudouH =0; break;}
             }//fecha switch
         return p;
        }//fecha if
        else return p; //arvore não cresceu
      }
      else return NULL; //p->fdir == NULL
    }
}
```

AVL

- Exercicio
  - Inserir os elementos 10, 3, 2, 5, 7 e 6 em uma árvore e balancear quando necessário

AVL

- Os percursos in-ordem da árvore original e da balanceada permanecem iguais
  - Exercício: prove para um dos exemplos anteriores!

AVL

- Exercício casa: teste a sub-rotina de inserção inserindo alguns elementos na árvore abaixo

Remoção em Árvores Binárias de Busca

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

Remoção em AVLs

- A altura máxima de uma ABB AVL é 1.44*log2 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 log2 n + 0.25
- Na média, é necessária uma rotação em 46.5% das inserções

Remoção em Árvores AVL

- Considere que o elemento a ser removido encontra-se na raiz de uma árvore T:
  - 1) A raiz não possui filhos: remover a raiz e anular T;
  - 2) A raiz possui um único filho: remover a raiz e substituí-la por seu filho;
  - 3) A raiz possui dois filhos: escolher o nó que armazena o maior elemento na subárvore esquerda e substituir a raiz por ele.

Remoção em Árvores AVL

- Pode ocasionar:
  - A diminuição da altura da árvore, e/ou;
  - A alteração dos fatores de balanceamento de seus nós.

Remoção em Árvores AVL

- Algoritmo de Remoção
  - Efetuar a remoção;
  - Ajustar os fatores de balanceamento;
  - Verificar a quebra do equilíbrio;
  - Se a árvore não estiver balanceada, corrigir a estrutura através de movimentações dos nós (Rotações).

Remoção em Árvores AVL

- Rotação
- Tipos de Rotação
  - Simples
  - Dupla
- Principais nós envolvidos:
  - Nó A: Nó ancestral mais próximo do ponto de remoção que possuía fator de balanceamento diferente de zero antes da remoção ser efetuada.
  - Nó B: Filho de A na subárvore de maior altura.
  - Nó C: Filho de B na subárvore de maior altura (utilizado apenas em rotações duplas).

Remoção em Árvores AVL

- Observação:
  - Enquanto a inserção de uma única chave pode resultar em uma rotação de dois ou três nós no máximo, a remoção pode exigir uma rotação em cada um dos nós ao longo de toda a trajetória de busca.
  - Ver exercício a seguir.

Algoritmo de Inserção

```
void remove_arvore_avl(ARVORE_AVL *arv, INFO info) {
 int mudouH = 0;
 arv->raiz=remove_arvore_avl_aux(arv->raiz,info, &mudouH);
}
```

```
tNO *remove_arvore_avl_aux(tNO *p, INFO info, int *mudouH) {
   if (p == NULL) {//Não existe a chave
     printf("Chave não encontrada !");
     *mudouH =0;
     return NULL;
   }
  ...
```

```
else {// p não é nulo
     if (p->info.chave>info.chave){ //descer a esquerda
        p->fesq = remove_arvore_avl_aux(p->fesq, info,mudouH);
        if (*mudouH) p = balanceamento_esquerdo(p,mudouH);
     }else if (p->info.chave<info.chave){ //descer a direita
         p->fdir = remove_arvore_avl_aux(p->fdir, info,mudouH);
        if (*mudouH)p = balanceamento_direito(p,mudouH);
     }else{//encontrou a chave
        printf("Chave encontrada igual %d",p->info.chave);
        if (p->fdir == NULL){//Nao tem filho a direita
            printf("\nNao tem filho a direita");
            printf("\nRemovendo a informação %d",p->info.chave);
            p = p->fesq;*mudouH =1;
         }
         else if(p->fesq==NULL){//Nao tem filho a esquerda
          printf("\nNao tem filho a esquerda");
          printf("\nRemovendo a informação %d",p->info.chave);
          p=p->fdir;*mudouH=1;
          }
         else{//tem filho a esquerda e direita
            p->fesq=busca_remove_avl(p->fesq,p,mudouH);
            if (*mudouH) p=balanceamento_esquerdo(p,mudouH);
         }}}
  return p;}
```

```
tNO *balanceamento_esquerdo(tNO *p,int *mudouH){
  int pfdir_fb;
  printf("\nRealizando Balanceando Esquerdo" );
  switch (p->fb){
  case 1:{//arvore maior a esquerda
    p->fb = 0;*mudouH = 1; //não precisa ja estava em 1
    break; }
  case 0:{//arvore balanceada
    p->fb = -1; //tende para direita
    *mudouH = 0;break;}
  case -1:{
     pfdir_fb = p->fdir->fb; //guarda o fator de bal. da direita
     if (pfdir_fb<=0){
       printf("\nRotacao Esquerda chave %d",p->info.chave );
       rot_esq(&p);
       if (pfdir_fb==0){
            p->fesq->fb=-1; //tende para direita
             p->fb=1; //tende para esquerda
            *mudouH = 0;}
        else{*mudouH = 1; //nao precisa}
     }else{
         printf("\nRotacao Direita/Esquerda" );
        rot_dir_esq(&p);
         *mudouH = 1; //nao precisa}
    break;}}
return (p);
}
```

```
tNO *balanceamento_direito(tNO *p,int *mudouH){
 int pfesq_fb;
 printf("\nRealizando Balanceando Direito" );
 switch (p->fb){
        case -1:{//arvore maior a direita
          printf("\nArvore era maior a direita %d" ,p->info.chave);
          p->fb = 0;*mudouH = 1; //não precisa ja estava em 1
          break;}
        case 0:{//arvore balanceada
          p->fb = 1; //tende para esquerda
          *mudouH = 0; break;}
        case 1:{//arvore maior a esquerda
          printf("\nArvore maior a esquerda %d %d" ,p->info.chave,p->fb);
           pfesq_fb = p->fesq->fb;
           if (pfesq_fb>=0){
              printf("\nRotacao Direita" );
              rot_dir(&p);
              if (pfesq_fb==0){
                 p->fdir->fb=1; //tende para esq
                 p->fb=-1; //tende para dir
                 *mudouH = 0;}
              else{*mudouH = 1; //nao precisa}
          }else{
              printf("\nRotacao Esquerda/Direita" );
              rot_esq_dir(&p);
             *mudouH = 1; //nao precisa}
           break;}}
    return (p);
}
```

```
tNO *busca_remove_avl(tNO *no,tNO *no_chave,int *mudouH){
  tNO *no_removido;
   if (no->fdir!=NULL){
      no->fdir =busca_remove_avl(no->fdir,no_chave,mudouH);
        if (*mudouH) no = balanceamento_direito(no,mudouH);
   }
  else{//nao tem filho a direita
    no_chave->info = no->info; //substitui a chave
    no_removido = no ;//guarda o endereço do no
    no = no->fesq;//se tiver filho refaz a ligação
    *mudouH = 1;
    free(no_removido);
    }
  return no;
}
```

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
  - a; z; b; y; c; x
  - a; z; b; y; c; x; d;w; e; v; f
  - a; v; l; t; r; e; i; o; k
  - 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

- EMostre 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