8. ÁRVORES

Até .....

8.1 Introdução

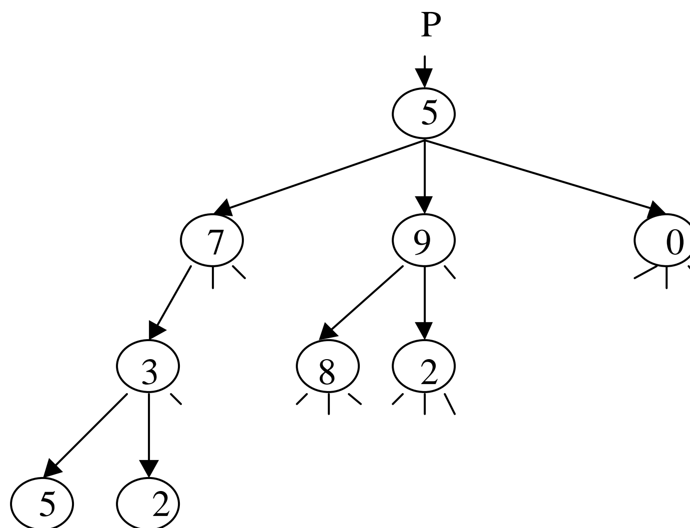
ÁRVORES

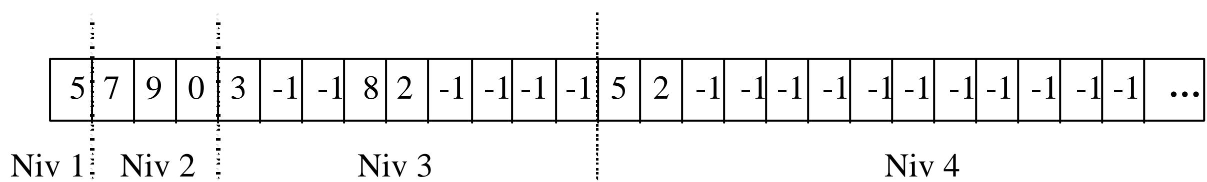
Formas de Armazenamento:

Árvores Estáticas:

Type Tipo_Árvore = Array[1..Max] of Tipo_Dado;

Por exemplo: árvore de grau 3.





Árvores Dinâmicas:

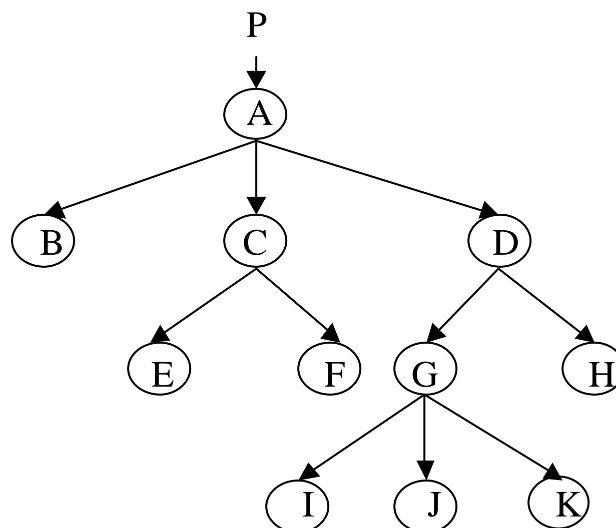
Declaração com Grau Fixo:

```
Type
    Arvore = ^Nodo;
    Nodo = Record
              Dado: Integer;
              Filho[1..3]: ^Nodo;
           End;
```

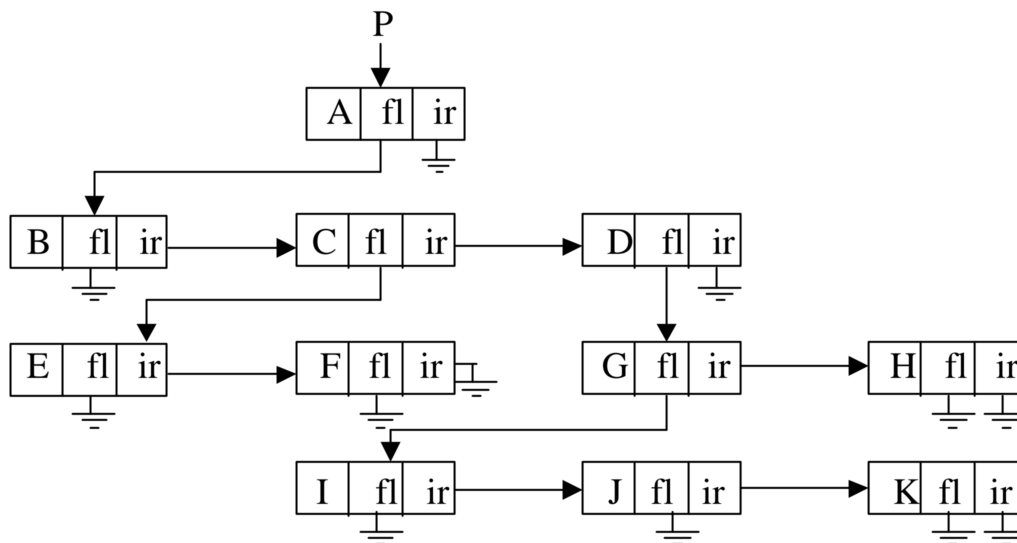
Declaração com Grau Dinâmico ou Arbitrário:

```
Type
    Nodo = Record
              Dado: Integer;
              Filho, Irmao: ^Nodo;
           End;
    Arvore = ^Nodo;

Var P: Arvore;
```

Exemplo de representação gráfica de árvore de grau arbitrário:



Exemplo da mesma representação com Lista Encadeada:



Inserção em Árvore de Grau Arbitrário: Deve ser informado quem é o pai. A inserção junto aos outros filhos deste mesmo Pai pode ser sempre na primeira posição ou ordenadamente. O exemplo abaixo insere o filho na primeira posição.

```
Procedure InsereArvoreGenerica(Var Pai: Arvore; Dado: integer);
Var No: ^Nodo;
Begin
   New(No);
   No^.Dado := Dado;
   No^.Filho := NIL;
   If (Pai = NIL)
     Then Begin
             No^.Irmao := NIL;
             Pai := No;
           End;
     Else Begin
             No^.Irmao := Pai^.Filho;
             Pai^.Filho := No;
          End;
End;
```

Exercício: Refazer o procedimento de inserção anterior para inserir os filhos ordenadamente. Suponha que os dados são nomes de pessoas.

Exercício 2: Faça uma função de busca que dado um nome retorna o ponteiro para o nó contendo o nome, na árvore genérica anterior.

```
function Busca(raiz: tipo_arvore; nome: tipo_nome): tipo_arvore;
var no : tipo_arvore
begin
```

```
  if raiz <> nil
    then begin
           no := Nil;
           if raiz^.nome = nome
             then no := raiz
             else begin
                    no := Busca(raiz^.ir);
                    if (no = Nil)
                      then no := Busca(raiz^.fs);
                  end
           Busca := no;
         end
   else Busca := Nil;
end;
```

Árvores Binárias

Uma árvore binária, considera qual é sua sub-árvore esquerda e direita, em cada nó. O valor **NIL** indica que não existe sub-árvore.

Declarações:

```
Type

Tipo_Arvore = ^Nodo;

Nodo = record
          Dado: integer;
          Esq, dir: Tipo_Arvore;
       End;
```

Algoritmo para Percorrer a Árvore Binária:

Três Tipos Básicos: Pré-Ordem, Em-Ordem e Pós-Ordem

Algoritmos recursivos de caminhamento:

```
Procedure  Pre_Ordem(Var Raiz: Tipo_Arvore);
     Begin
       If (Raiz <> Nil)
         Then Begin
                Writeln("Dado = ",Raiz^.Dado);
                Pre_Ordem(Raiz^.Esq);
                Pre_Ordem(Raiz^.Dir);
```

```
                End;
        End;

Procedure  Em_Ordem(Var Raiz: Tipo_Arvore);
        Begin
          If (Raiz <> Nil)
            Then Begin
                  Em_Ordem(Raiz^.Esq);
                  Writeln("Dado : ",Raiz^.Dado);
                  Em_Ordem(Raiz^.Dir);
            End;
        End;

Procedure  Pos_Ordem(Var Raiz: Tipo_Arvore);
        Begin
          If (Raiz <> Nil)
            Then Begin
                     Pos_Ordem(Raiz^.Esq);
                     Pos_Ordem(Raiz^.Dir);
                     Writeln("Dado : ",Raiz^.Dado);
                End;
        End;
```

Algoritmos não recursivos de caminhamento: (com Pilha)

```
Procedure  EM_ORDEM (Raiz : Tipo_Arvore);
Var  P: pilha; T: Arvore;  no: ^Nodo; Acabou: Boolean;
Begin
   Acabou := FALSE;   Criapilha(P);   T := Raiz;
   Repeat
     While (T <> NIL) Do
     Begin
        InserePilha (P, T);
        T := T^.Esq;
     End.
     If (PilhaVazia(P))
       Then Acabou = TRUE
       Else Begin
               RemovePilha (P, no);
               Writeln("Dado = ", no^.dado);
               T := no^.dir;
            End;
   Until Acabou;
End.
```

```
Procedure  POS_ORDEM (Raiz : Tipo_Arvore);
Var P: tipo_pilha;
      Acabou, Achou,
      Sentido: Boolean;    /* se FALSE, entao passou pelo nó e desceu pela esquerda  */
Begin                      /* se TRUE, entao passou pelo nó e desceu pela direita */
   Acabou := FALSE;  Criapilha(P);
   Repeat
      While (Raiz <> NIL) Do
      Begin
         InserePilha (P, Raiz, FALSE);
          Raiz := Raiz^.Esq;
      End.
      If (PilhaVazia(P))
        Then Acabou = TRUE
        Else Begin
                Achou := FALSE;
                While (NOT Achou) and (Not PilhaVazia(P))
                    Begin
                       Raiz := Topo(P, Sentido);
                       If  (Sentido <> TRUE)
                          Then Begin
                                   AtualizaTopo(P, TRUE)
                                    Raiz := Raiz^.Dir; Achou := TRUE;
                                 End
                          Else Begin
                                  RemovePilha (P, Raiz);
                                  Writeln("Dado = ", Raiz^.dado);
                                End;
                    End;
                Acabou := PilhaVazia(P);
   Until Acabou;
End.
```

Exercícios:

1) Fazer o Pre_Ordem usando Pilha.
2) Fazer o Pre_Ordem para contar o numero de elementos de uma Árvore de Grau Arbitrário
3)

Exemplo: Algoritmo para verificar a igualdade de duas árvores binárias:

```
Function Igual (A1, A2 : Arvore):  boolean;
Var result, acabou :  boolean;
     P : pilha;
     No1, no2 : ^nodo;
```

```
Begin
   CriaPilha(P);
   Result = True;
   Acabou = False;
   While ( (Result=True) and (~acabou) ) Do
   Begin
      While ( (A1 <> NIL) or (A2 <> NIL) ) Do
         Begin
            InserePilha (P, A1);
            InserePilha (P, A2);
            A1 := A1^.Esq;
            A2 := A2^.Esq;
         End.
     If ( (A1 <> NIL) or (A2 <> NIL) )
       Then result = False;
       Else if  PilhaVazia(P)
                Then acabou = true;
                Else Begin
                        RemovePilha (P, no2);
                        RemovePilha (P, no1);
                        Result = (no2^.dado = no1^.dado);
                        A1 = no1^.dir;
                        A2 = no2^.dir;
                     End;
   End;
   Igual := Result;
End;
```

Algoritmos para Inserção e Remoção em Árvore Binária (Árvore Binária de Busca):

Usando Recursividade:

```
Procedure  Ins_Arv_Bin (Var Raiz: Tipo_Arvore; Var X: Tipo_Elem);
     Begin
         If (Raiz = Nil)
              Then Begin
                          New(Raiz);
                          Raiz^.Dado := X;
                          Raiz^ .Esq := Nil;
                          Raiz^.Dir  := Nil;
                       End
              Else  Begin
                          If  (X < Raiz^.Dado)
                             Then Ins_Arv_Bin (Raiz^.Esq,X)
                             Else If  (X >Raiz^.Dado)
                                       Then Ins_Arv_Bin (Raiz^.Dir,X)
                                       Else Raiz^.Dado := X {Substituição}
```

```
                End;
    End;

Procedure Sucessor_Esq(SubArv: Tipo_Arvore; Var R: Tipo_Arvore);
Begin
        If  (R^.Dir <>Nil)
            Then Sucessor_Esq(SubArv,R^.Dir)
            Else Begin
                    SubArv^.Dado := R^.Dado;
                    SubArv:= R;
                    R := R^.Esq;
                    Dispose(SubArv);
                End;
End;

Function Rem_Arv_Bin( Var Raiz: Tipo_Arvore; Var X: Tipo_Elem): Boolean;
  Var
        Aux: Tipo_Arvore;
  Begin
        If  (Raiz = Nil)
           Then  Rem_Arv_Bin := FALSE
           Else    If  (X < Raiz^.Dado)
                     Then Rem_Arv_Bin (Raiz^.Esq, X)
                     Else If  (X > Raiz^.Dado)
                              Then  Rem_Arv_Bin (Raiz^.Dir, X)
                              Else If  (Raiz^.Dir = Nil)
                                      Then Begin
                                                X := Raiz^.Dado;
                                                Aux := Raiz;
                                                Raiz := Raiz^.Esq;
                                                Dispose(Aux);
                                            End
                                      Else If (Raiz^.Esq = Nil)
                                           Then Begin
                                                      X:=Raiz^.Dado;
                                                      Aux := Raiz;
                                                      Raiz := Raiz^.Dir;
                                                      Dispose(Aux);
                                                  End
                                           Else Begin
                                                     X := Raiz^.Dado;
                                                     Sucessor_Esq (Raiz, Raiz^.Esq);
                                                 End;
  End;
```

Inserção em Árvore Binária de Busca Sem Recursividade:

```
Procedure  Ins_Arv_Bin (Var Raiz: Tipo_Arvore; X: Tipo_Elem);
Var aux, ant, pos : Tipo_Arvore;
Begin
   new(aux);
   aux^.dado := X;
   aux^.dir := NIL;  aux^.esq := NIL;
   pos := Raiz;
   If Raiz <> NIL
     Then Begin
              While (pos <> NIL) Do
                Begin
                    ant := pos;
                    If (X <= pos^.dado)
                      Then  pos := pos^.esq
                      Else  pos := pos^.dir;
               End;
               If  (x <= ant^.Dado)
                  Then ant^.esq := aux
                  Else ant^.dir := aux;
            End
     Else Raiz := aux;
End;
```

Remoção em Árvore Binária de Busca sem Recursividade:

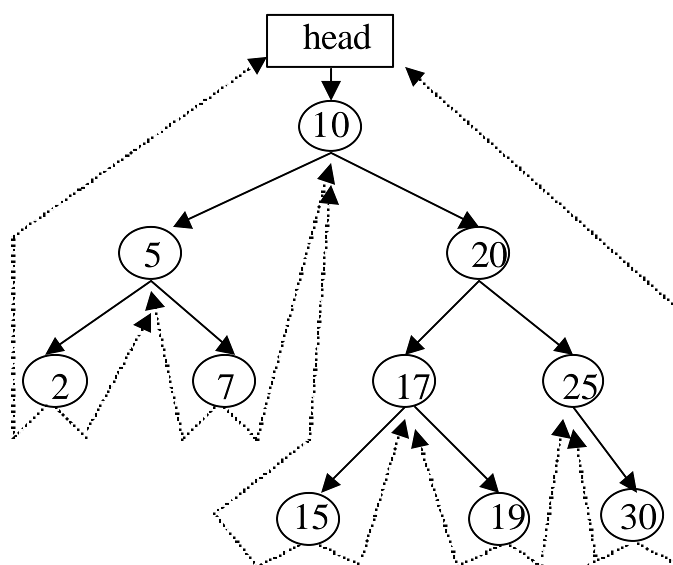
```
Procedure  Troca_Maior_Esq (pos: Arvore);
Var aux, ant: tipo_arvore;
Begin
   Aux := pos^.esq;  Ant := NIL;
   While (aux^.dir <> NIL) Do
          Begin
             Ant := aux;
             Aux := aux^.dir;
          End;
   Pos^.dado := aux^.dado;
   If (ant = NIL)
      Then pos^.esq := aux^.esq
      Else ant^.dir := aux^.esq;
   Dispose(ant);
End;

Function Rem_Arv_Bin( Var Raiz: Tipo_Arvore; Var X: tipo_elem): Boolean;
Var  pos, aux: tipo_arvore; achou, sentido: boolean;
Begin
   pos := Raiz; achou := FALSE;
   ant := NIL;
   while (pos <> NIL) Do
```

```
  Begin
     If (X < pos^.dado)
      Then Begin
              ant := pos;
              pos := pos^.esq;
              sentido := FALSE;
            End
      Else Begin
              If (X > pos^.dado)
                Then Begin
                        ant := pos;
                        pos := pos^.dir;
                        sentido := TRUE;
                      End
                Else Begin
                        achou := TRUE;
                        aux := pos;
                        X := pos^.dado;
                        If pos^.esq = NIL
                          Then begin
                                  If ant = NIL
                                     Then raiz := pos^.dir
                                     Else  If (sentido)
                                             Then ant^.dir := pos^.dir
                                             Else ant^.esq := pos^.dir;
                                 dispose(aux);
                              end
                          Else begin
                                  if pos^.dir = NIL
                                    then begin
                                            If  ant = NIL
                                               Then raiz := pos^.esq
                                               Else If  (sentido)
                                                       Then ant^.dir := pos^.esq
                                                        Else ant^.esq := pos^.esq;
                                          dispose(aux);
                                        end
                                    else  Troca_Maior_Esq (pos)
                                end;
                      End;
            End;
  End;
  RemoveArvore := achou;
End;
```

Exercício: Fazer o mesmo usando a substituição pelo nenor dos maiores.

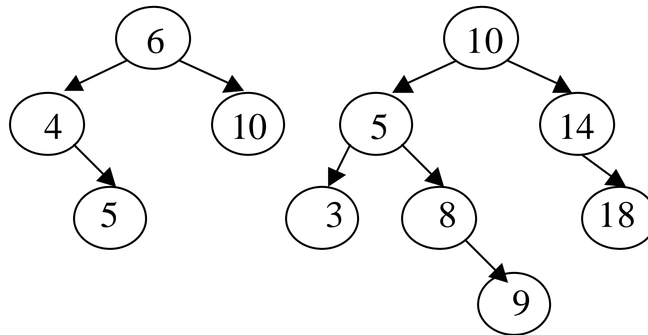
Árvore Binária com Reaproveitamento:

Existem mais nodos com ponteiros nulos doque nodos com ponteiros usados. Uma observação mais detalhada mostra **n-1** nodos com ponteiros usados e **n+1** nodos com ponteiros nulos. Uma forma de aproveitar estes ponteiros é usalos para apontar o predecessor e o sucessor conforme mostra a figura a seguir. Para controlar entre ponteiros normais e ponteiros reaproveitados devemos inserir dois bits a mais, o **BitEsq** e o **BitDir**, que deve conter 0 para o ponteiro normal e 1 para o ponteiro reaproveitado, ou vice-versa.



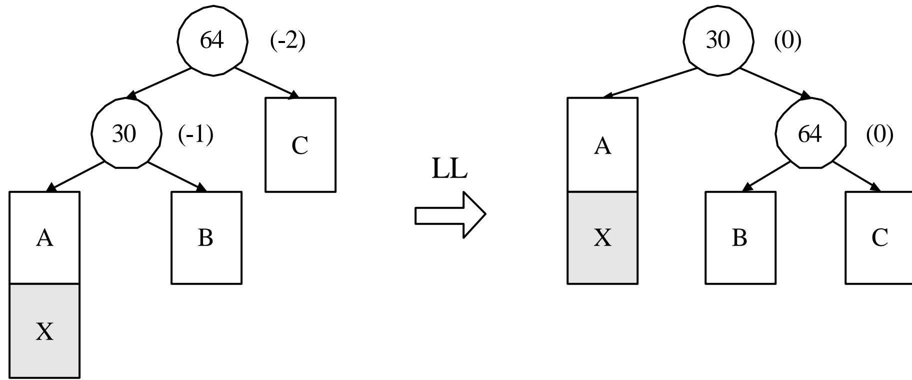
Árvores Balanceadas

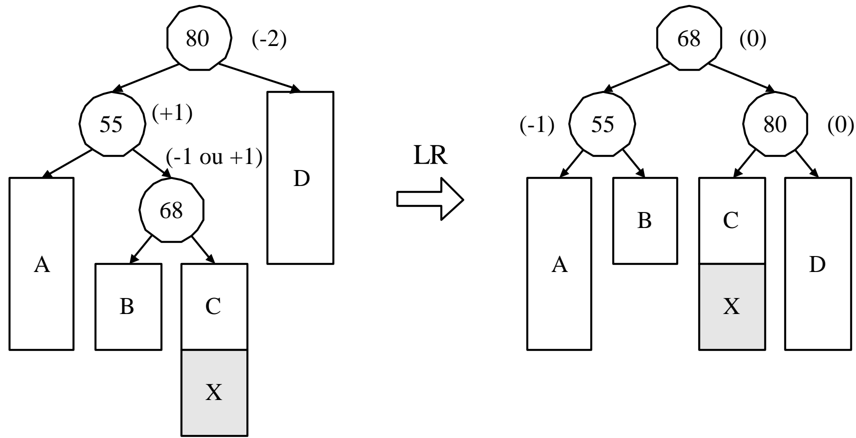
Também chamada de AVL em homenagem aos seus criadores Adelson-Velskii e Landis. A idéia é fazer com que os filhos de um pai, ou seja, as subárvores esquerda e direita de um nó, possuam alturas quase que iguais, diferenciando no máximo de um nível. Abaixo estão exemplos de árvores balanceadas.

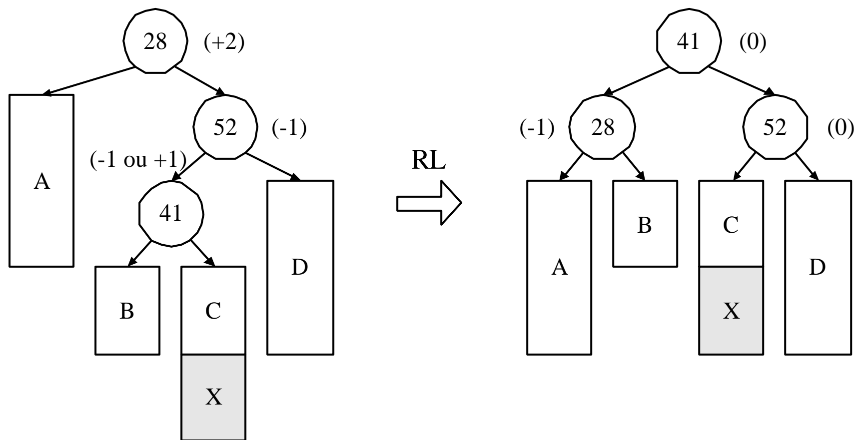


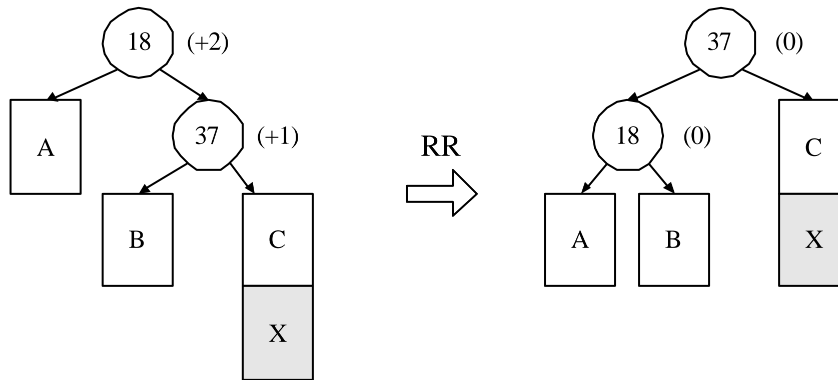
O balanceamento AVL não é totalmente perfeito, mas é uma solução aproximada que visa simplificação para que desempenho final seja melhorado. Cada nó da árvore possui um campo "bal", assim como o MinHeap, que guarda a informação de balanceamento dos filhos. Diferentemente do método do MinHeap, o "bal" de uma árvore AVL indica a diferença no número de níveis e não no número de nós. Assim, consideremos um determinado nó. Se "bal" = 1, isto significa que a subárvore direita possui um nível a mais que a sua subárvore esquerda. Se "bal" = 0, isto implica que ambas as subárvores esquerda e direita possuem o mesmo número de níveis. Se "bal"= -1, isto significa que a subárvore esquerda possui um nível a mais que a subárvore direita. Considera-se que determinado nó ficará desbalanceado se o seu "bal" atingir 2 ou -2. Para ajudar neste controle, usa-se a variável lógica "cresceu" para informar se determinada subárvore sofreu ou não aumento de nível.

A técnica de balanceamento AVL não permite que uma árvore permaneça desbalanceada, assim sendo, sempre durante uma inserção ou remoção, se a árvore ficar desbalanceada, ela deve ser imediatamente balanceada. Existem 4 operações básicas para balancear uma subárvore que tem o nó "p" como raiz. Estas operações são: LL, LR, RR e RL. A primeira letra indica o lado (left ou right) do nó "p" onde ocorreu o desbalanceamento => bal = -2 ou bal = +2. A segunda letra indica o lado (left ou right) do nó seguinte ao "p", que pode ser o nó P^.esq ou P^.dir, que possui um nível a mais (bal = -1 ou bal = +1). Por exemplo, a técnica LL deve ser usada para balancear um nó "p" que possui "bal = -2", ou seja, que tem desbalanceada a sua subárvore esquerda, cuja subárvore esquerda possui um nível a mais também no lado esquerdo (bal = -1). As figuras a seguir ilustram estes métodos, onde os retângulos identificados por letras representam sub-árvores de mesmo tamanho, podendo até serem nulas.


64
(-2)
30
(-1)
C
A
B
X
LL
30
(0)
A
64
(0)
X
B
C


80
(-2)
55
(+1)
(-1 ou +1)
D
68
A
B
C
X
LR
68
(0)
(-1)
55
80
(0)
A
B
C
X
D


28
(+2)
A
52
(-1)
(-1 ou +1)
41
D
B
C
X
RL
41
(0)
(-1)
28
52
(0)
A
B
C
X
D




```
type   tipo_avl = ^tipo_nodo;
         tipo_nodo = record
                          dado: tipo_dado;
                          esq, dir: tipo_avl;
                          count,
                          bal: integer;
                       end;
```

```
procedure  balanceia_esq;
var
begin
   case raiz^.bal of
     +1:  begin
              raiz^.bal := 0;
              cresceu := false;
          end;
     0: raiz^.bal := -1;
     -1:  begin
              p1 := raiz^.esq;
              if (p1^.bal = -1)
                then begin    { LL }
                         raiz^.esq := p1^.dir;
                         p1^.dir := raiz;
                         raiz^.bal := 0;
                         p1^.bal := 0;
                         raiz := p1;
                         cresceu := false;
                     end
                else begin   { LR }
                         p2 := p1^.dir;
                         raiz^.esq := p2^.dir;
```

```
                    p1^.dir := p2^.esq;
                    p2^.esq := p1;
                    p2^.dir := raiz;
                    if  (p2^.bal = -1)
                        then begin
                                raiz^.bal := +1;
                                p1^.bal := 0;
                              end
                        else begin
                                raiz^.bal := 0;
                                p1^.bal := -1;
                              end;
                    raiz := p2;
                    raiz^.bal :=0;
                    cresceu = false;
                  end
        end;
end;
```

```
procedure  balanceia_dir;
var
begin
   case raiz^.bal of
     -1: begin
            raiz^.bal := 0;
            cresceu := false;
          end;
      0: raiz^.bal := +1;
    +1:  begin
            p1 := raiz^.dir;
            if (p1^.bal = +1)
              then begin    { RR }
                      raiz^.dir := p1^.esq;
                      p1^.esq := raiz;
                      raiz^.bal := 0;
                      p1^.bal := 0;
                      raiz := p1;
```

```
                    cresceu := false;
                 end
             else begin   { RL }
                    p2 := p1^.esq;
                    raiz^.dir := p2^.esq;
                    p1^.esq := p2^.dir;
                    p2^.esq := raiz;
                    p2^.dir := p1;
                    if  (p2^.bal = +1)
                       then begin
                              raiz^.bal := -1;
                              p1^.bal := 0;
                           end
                       else begin
                              raiz^.bal := 0;
                              p1^.bal := +1;
                           end;
                    raiz := p2;
                    raiz^.bal :=0;
                    cresceu = false;
                 end
        end;
end;
```

```
procedure  insereAVL(var raiz: tipo_avl; x: tipo_dado; var cresceu: boolean);
var  p1, p2: tipo_avl;
begin
   if (raiz = NIL)
      then begin
                New(raiz);
                cresceu := true;
                raiz^.count := 1;
                raiz^.dado := x;
                raiz^.esq := NIL;  raiz^.dir :=NIL;
                raiz^.bal := 0;
             end
      else begin
```

```
              if (x < raiz^.dado)
                 then begin
                         insereAVL(raiz^.esq, x, cresceu);
                         if (cresceu)  then balanceia_esq;
                      end
                 else if (x > raiz^.dado)
                         then begin
                                 insereAVL(raiz^.dir, x, cresceu);
                                 if (cresceu)  then balanceia_dir;
                              end
                         else begin
                                 inc(raiz^.count);
                                 cresceu := false;
                              end;
          end;
end;
```

Remoção de Árvore balanceada de Busca:

Durante a remoção de um nodo de uma AVL, a subárvore que sofreu a remoção pode ser diminuída de um nível, ou não. Neste caso, o nodo pai desta subárvore poderá ficar desbalanceado, necessitando assim de balanceamento.

```
procedure  SubstituiMaiorEsq(var raiz, p: tipo_avl; var reduziu: boolean);
var aux: tipo_avl;
begin
   if (p^.dir <> NIL)
      then begin
               Troca(raiz, p^.dir, reduziu);
               if (reduziu)  then balanceia_esq(p, reduziu);
           end
      else begin
              raiz^.dado := p^.dado;
              raiz^.count := p^.count;
              aux :=p;    p := p^.esq;
              dispose (aux);
              reduziu := true;
           end;
end;
procedure  Remove_AVL (var raiz: tipo_avl; var x: tipo_dado; var reduziu: boolean);
var aux: tipo_avl;
begin
   if (raiz <> NIL)
      then begin
              if (x < raiz^.dado)
                then begin
                        Remove_AVL (raiz^.esq, x, reduziu);
```

```
                    if (reduziu)
                       then balanceia_dir (raiz, reduziu);
                  end
            else begin
                    if (x > A^.dado)
                       then begin
                                 Remove_AVL(raiz^.dir, x reduziu);
                                 if (reduziu)
                                    then balanceia_esq(raiz, reduziu);
                              end
                        else begin
                                 x := raiz^.count;    aux := raiz;
                                 if (raiz^.esq = NIL)
                                   then begin
                                             raiz := raiz^.dir;
                                             Dispose(aux);
                                             reduziu := true;
                                          end
                                   else begin
                                          if (raiz^.dir = NIL)
                                             then begin
                                                       raiz := raiz^.esq;
                                                       Dispose(aux);
                                                       reduziu := true;
                                                    end
                                             else begin
                                                       SubstituiMaiorEsq(raiz,          raiz^.esq,
reduziu);
                                                       if (reduziu)
                                                          then balanceia_dir(raiz,reduziu);
                                                    end;
                                          end;
                              end;
                  end;
         end;
end;
```