Aula 07 - Conteúdo

1) Árvore AVL ou árvore de altura equilibrada
2) Algoritmos para árvores AVL
3) Exercícios

Árvores AVL ou Arv. Bin. de Altura Equilibrada.

Adelson Velsky e Landis introduziram em 1962 uma estrutura de árvore binária que é equilibrada com respeito as alturas das sub árvores. Devido a natureza equilibrada das árvores AVL, as recuperações, inserções e deleções podem ser efetuadas num tempo da ordem de log N onde N é o nº de nós.

Definição 1: a árvore vazia é de altura equilibrada. Seja T uma árvore binária de busca não vazia com T_E e T_D como sub árvores esquerda e direita, então T terá altura equilibrada se:

i) T_E e T_D forem equilibradas na altura

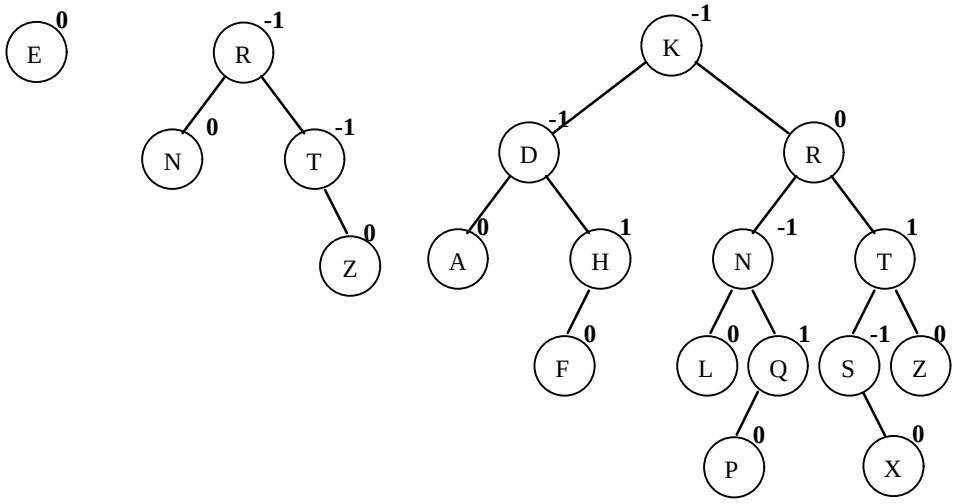
ii)$| h_E - h_D | \leq 1$ onde h_E e h_D são as alturas de T_E e T_D respectivamente

Definição 2: uma árvore AVL é uma árvore binária de busca que é vazia ou cuja diferença de pesos entre as profundidades das sub árvores esquerda e direita é, no máximo, igual a 1, e as sub árvores direita e esquerda, por sua vez, também são árvores AVL.

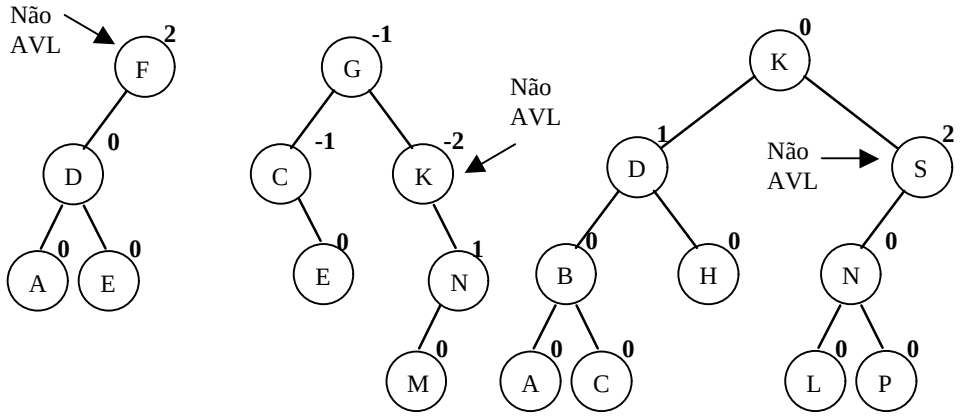
Uma árvore AVL sempre tem um certo grau de balanço, mas não precisa estar perfeitamente balanceada. Para restaurar o balanço após uma inserção ou remoção temos que reorganizar a árvore.

A principal vantagem das árvores AVL é que o balanceamento permite que as buscas **não** se tornem seqüenciais, porém devemos considerar o trabalho extra gasto no balanceamento.

Exemplos de árvores AVL.

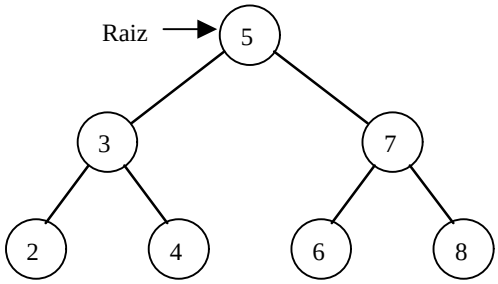


Exemplos de árvores **NÃO** AVL.

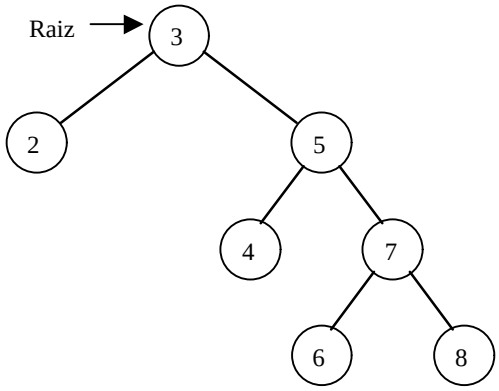


Antes de estudarmos os algoritmos de balanceamento para árvores AVL devemos conhecer as operações de **rotação direita** e **rotação esquerda** que podem ser aplicadas em qualquer árvore binária.

Considera a árvore binária abaixo:

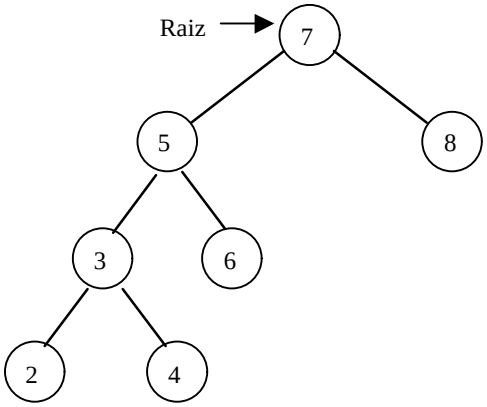


Após efetuarmos uma **Rotação Direita** temos:




```
Algoritmo:
p = Raiz
q = FilhoEsq(p)
h = FilhoDir(q)
FilhoDir(q) = p
FilhoEsq(p) = h
Raiz = q;
```

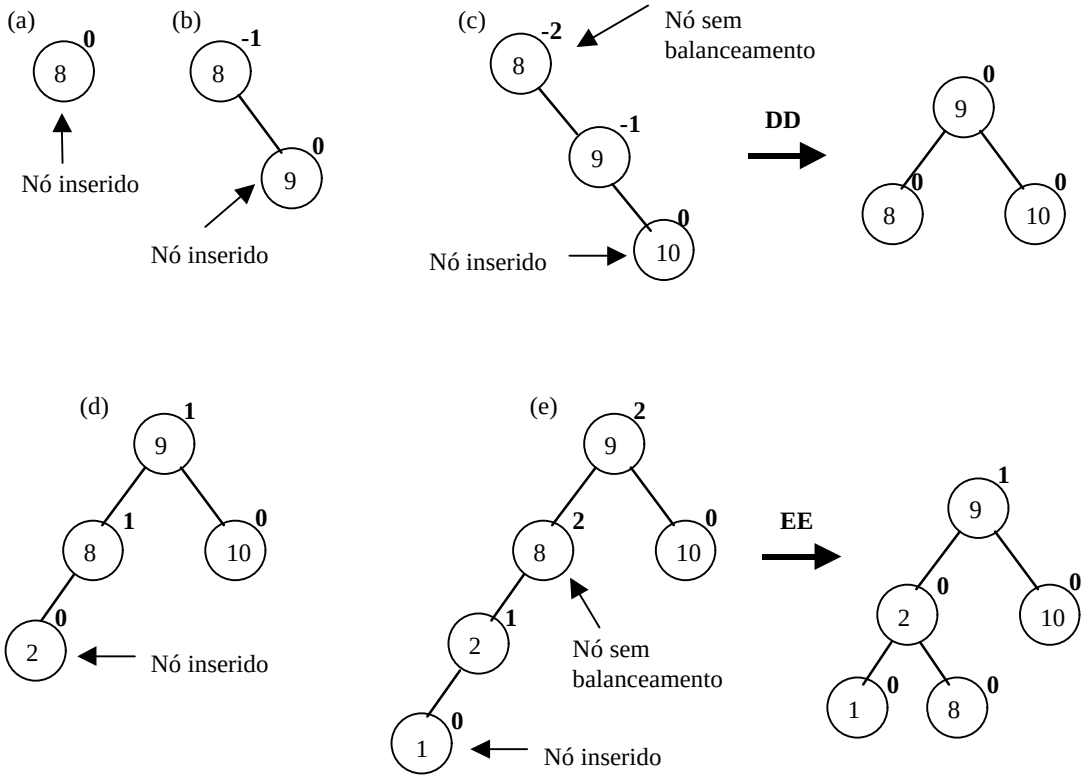
Após efetuarmos uma **Rotação Esquerda** temos:



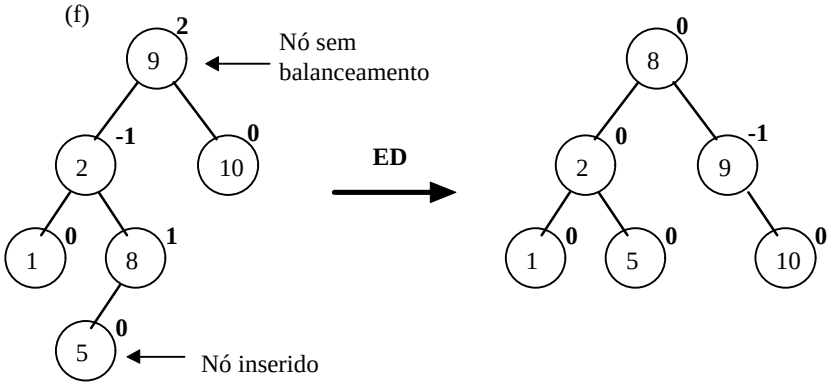

```
Algoritmo:
p = Raiz
q = FilhoDir(p)
h = FilhoEsq(q)
FilhoEsq(q) = p
FilhoDir(p) = h
Raiz = q;
```

Devemos observar que aos percorrermos a árvore original ou as árvores rotacionadas, utilizando o algoritmo in order, teremos a mesma seqüência de nós visitados. Por exemplo: para qualquer uma das três árvores anteriores o percurso in order é 2, 3, 4, 5, 6, 7 e 8.

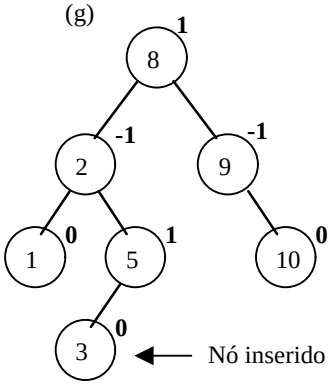
Para ilustrar os processos envolvidos na manutenção de uma árvore AVL vamos construir uma árvore inserindo os nós 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, 11 e 12 na seqüência dada



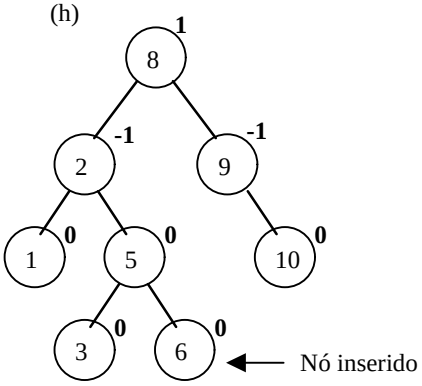
(f)


9
2
Nó sem
balanceamento
2
-1
10
0
1
0
8
1
5
0
Nó inserido
ED
8
0
2
0
9
-1
1
0
5
0
10
0

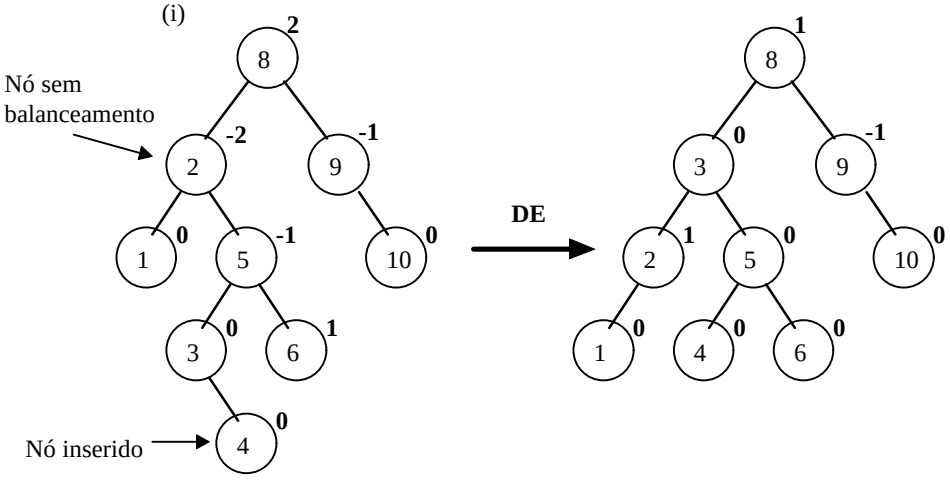
(g)


8
1
2
-1
9
-1
1
0
5
1
10
0
3
0
Nó inserido

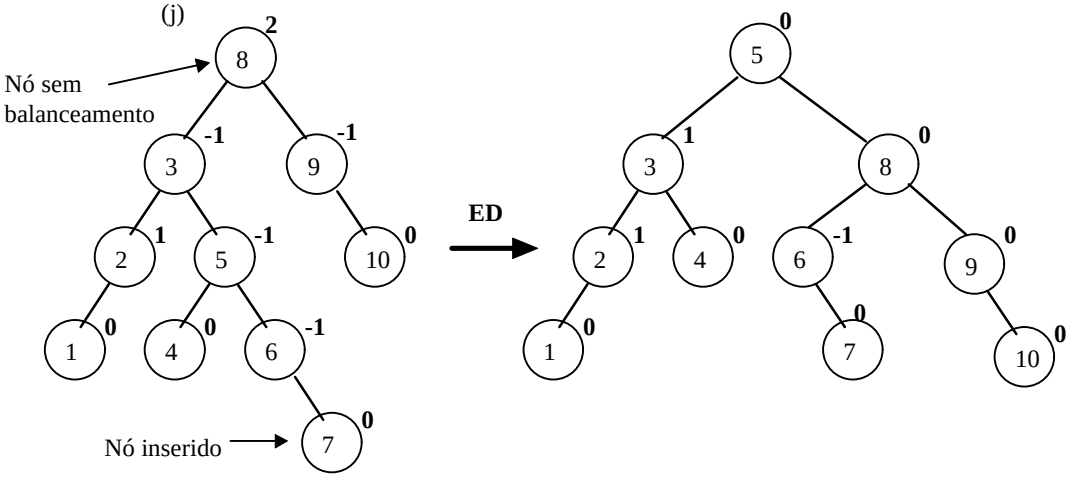
(h)


8
1
2
-1
9
-1
1
0
5
0
10
0
3
0
6
0
Nó inserido

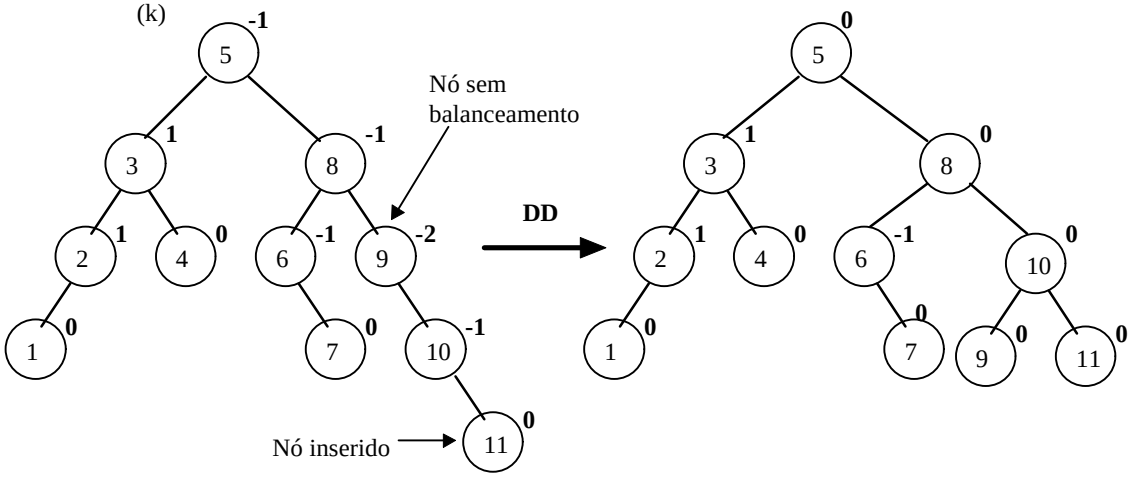
(i)


Nó sem
balanceamento
8
2
2
-2
9
-1
1
0
5
-1
10
0
3
0
6
1
Nó inserido
4
0
DE
8
1
3
0
9
-1
2
1
5
0
10
0
1
0
4
0
6
0

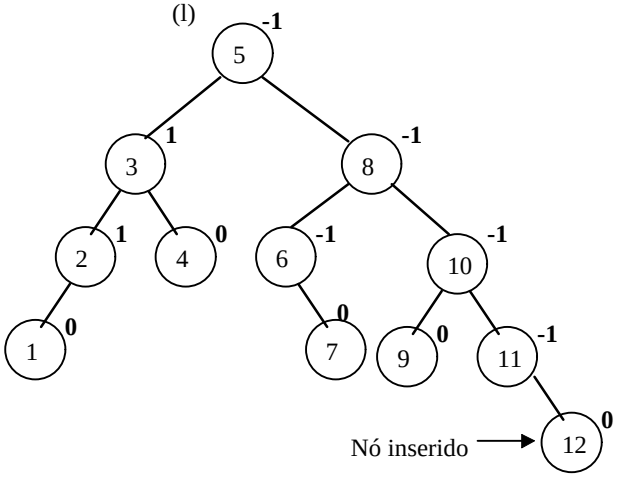
(j)


Nó sem balanceamento
Nó inserido
ED

(k)


Nó sem balanceamento
Nó inserido
DD

(l)


Nó inserido

No exemplo anterior vimos como o acréscimo de um nó é capaz de desequilibrar a árvore AVL. O processo de reequilibrar é conduzido essencialmente usando 4 tipos de rotação: EE, DD, ED e DE. Essas rotações são caracterizadas pelo ancestral mais próximo, **A**, do nó inserido, **Y**, cujo fator de equilíbrio passa a ser +2 ou -2. Assim sendo, temos:

EE : novo nó Y inserido na sub árvore esquerda da sub árvore esquerda de A.
ED : novo nó Y inserido na sub árvore direita da sub árvore esquerda de A.
DD : novo nó Y inserido na sub árvore direita da sub árvore direita de A.
DE : novo nó Y inserido na sub árvore esquerda da sub árvore direita de A.

O algoritmo abaixo, elaborado em C, insere um elemento numa árvore AVL e realiza o seu balanceamento, caso necessário (arquivo ProjEx701.dpr e Ex701.pas).

```
TNo = class
  protected
    Info: string;
    Bal: integer;
    Dir, Esq: TNo;
  public
    Constructor InicNo(str: string;b:integer = 0);
    Procedure SetNoInfo(str:string);
    Procedure GetNoInfo(var str:string);
  end;

TArvBin = Class
  protected
    Raiz: TNo;
    Procedure LiberaNos(var T:TNo);
  public
    Constructor InicArvBin; overload;
    Constructor InicArvBin(x:string);overload;
    Destructor DelArvBin;
    Function InsDir(p: TNo; x: string):boolean;
    Function InsEsq(p: TNo; x: string):boolean;
    Function FilhoDir(p: TNo):TNo;
    Function FilhoEsq(p: TNo):TNo;
    Function Altura: integer;
    Procedure ImpPorNivel;
    Procedure InOrder;
    Procedure PreOrder;
    Procedure PosOrder;
  end;

TABB = Class(TArvBin)
  protected
    Function InsABB(var T:Tno; x: string): boolean;
    Function BuscaABB(var T:Tno ; x: string): TNo;
    Procedure Substitue(var T,suc: TNo);
    Function DelABB(var T: TNo;x: string):boolean;
  public
    Function Inserir(x: string): boolean;
    Function Buscar(x: string): TNo;
    Function Remover(x: string): boolean;
  end;

TAVL = Class(TABB)
  protected
    Function InsAVL(var T:Tno; x: string): boolean;
```

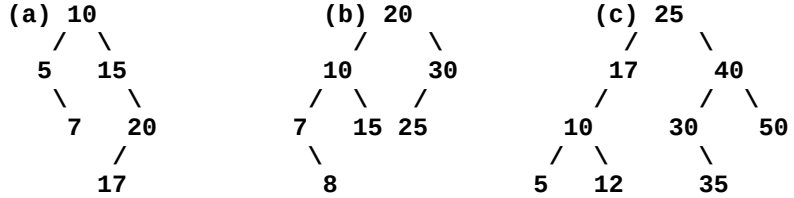
```
    Procedure RotacaoDir(var T:TNo);
    Procedure RotacaoEsq(var T:TNo);
  private
    Function Inserir(x: string): boolean;
  end;

var
  T01: TAVL;

//As implementações das classes Tno, TarvBin e TABB foram efetuadas
//nas aulas 06 e 07

Procedure TAVL.RotacaoDir(var T:TNo);
var q,h: TNo;
begin
  q := T.Esq;
  h := q.Dir;
  q.Dir := T;
  T.Esq := h;
end;

Procedure TAVL.RotacaoEsq(var T:TNo);
var q,h: TNo;
begin
  q := T.Dir;
  h := q.Esq;
  q.Esq := T;
  T.Dir := h;
end;

Function TAVL.InsAVL(var T:Tno; x: string): boolean;
var p,q,fp,ya,fya,s: TNo;
    imbal:integer;
begin
  if ( T = NIL ) then
  begin
    T := TNo.InicNo(x,0);
    InsAVL := TRUE;
    Exit;
  end
  else
  begin
    ya := T;   p := T;
    fp := NIL; fya := NIL;
    while (p <> NIL) do
    begin
      if (x = p.Info) then
      begin
        InsAVL := FALSE;
        Exit;
      end
      else
      begin
        if ( x < p.Info ) then q := p.Esq
        else q := p.Dir;
      if (q <> NIL) then
      begin
        if (q.Bal <> 0 ) then
        begin
          fya := p;
          ya := q;
```

```
      end;
    end;
    fp := p;
    p := q;
    end;
  end;
end;

//aloca espaco e inicializa um no´
InsAVL:=TRUE;
q := TNo.InicNo(x,0);
if ( x < fp.Info) then fp.Esq := q
else fp.Dir := q;

//Alterar coeficientes de balanceamento entre ya e q
if ( x < ya.Info ) then p := ya.Esq
else p := ya.Dir;
s := p;
while (p <> q) do
begin
  if (x < p.Info) then
  begin
    p.Bal := 1;
    p := p.Esq;
  end
  else
  begin
    p.Bal := -1;
    p := p.Dir;
  end
end;
//Determina se a árvore está desbalanceada, sendo q o nó
//inserido, ya o nó desbalanceado com maior altura, fya o
//pai de ya e s o filho de ya na direção do desbalanceamento
if ( x < ya.Info) then imbal := 1
else imbal := -1;
if (ya.Bal = 0) then //arv. não desbalanceada
begin
  ya.Bal := imbal;
  Exit;
end
else
if (ya.Bal <> imbal) then //arv. não desbalanceada
begin
  ya.Bal := 0;
  Exit;
end
else
begin
  if (s.Bal = imbal) then //arv. desbal. na mesma direção (EE,DD)
  begin
    p := s;
    if (imbal = 1) then RotacaoDir(ya)
    else RotacaoEsq(ya);
    ya.Bal := 0; s.Bal := 0;
  end
  else  //arv. desbal. Em direções opostas (ED,DE)
  begin
    if (imbal = 1) then
    begin
    p := s.Dir;
```

```
      RotacaoEsq(s);
      ya.Esq := p;
      RotacaoDir(ya);
      end
      else
      begin
      p := s.Esq;
        RotacaoDir(s);
        ya.Dir := p;
        RotacaoEsq(ya);
      end;
      if (p.Bal = 0) then
      begin
        ya.Bal := 0;
        s.Bal := 0;
      end
      else if (p.Bal = imbal) then
      begin
       ya.Bal := -imbal;
         s.Bal := 0;
      end
      else
      begin
        ya.Bal := 0;
      s.Bal := imbal;
      end;
      p.Bal := 0;
    end;
  end;
  //ajusttar o apontador da subárvore rotacionada
  if (fya = NIL) then T := p
  else
  begin
    if ( ya = fya.Dir) then fya.Dir := p
    else fya.Esq := p;
  end;
end;

Function TAVL.Inserir(x: string): boolean;
begin
  Inserir := InsAVL(Raiz,x);
end;
```

Exercícios

7.01) Verifique se as árvores abaixo são AVL. Caso a árvore não seja AVL indique o nó sem balanceamento.



7.02) Monte uma árvore AVL inserindo os nós na seqüência dada:

a) 50, 30, 20, 70, 40, 35, 37, 38, 10, 32, 45, 42, 25, 47, 36.

b) 100, 80, 60, 40, 20, 70, 30, 50, 35, 45, 55, 75, 65, 73, 77

c) 200, 170, 125, 85, 42, 138, 61, 100, 69, 90, 111, 150, 133, 146, 155

d) 95, 82, 60, 40, 22, 67, 32, 52, 34, 46, 53, 75, 62, 70, 78

Respostas

7.01)

a) Não AVL - nó desequilibrado: 15.

b) É AVL.

c) Não AVL - nó desequilibrado 17

7.02) item (a)