Aula 06 - Conteúdo

1) Árvore Binária de Busca
2) Algoritmos para árvore binária de busca
3) Exercícios

Árvore Binária de Busca

A árvore binária de busca é uma árvore binária que satisfaz as seguintes propriedades:

a) cada nó **w** na subárvore esquerda de um nó **v**, satisfaz w.info < v.info
b) cada nó **w** na subárvore direita de um nó **v** satisfaz w.info > v.info
c) para cada elemento **x** da árvore, existe um e só um nó **v** tal que v.info = x

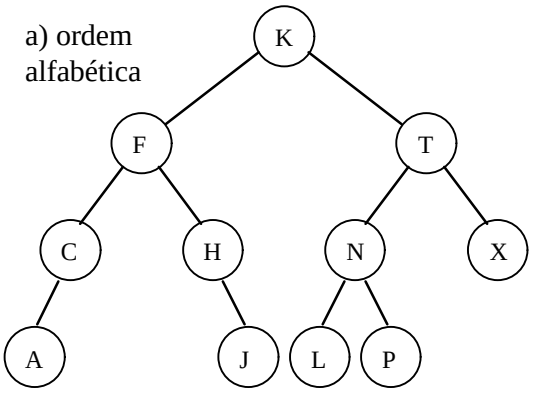
Em outras palavras a definição acima indica que todos os nós a esquerda de um determinado nó são menores (informação/chave) que este nó e todos os nós a direita de um dado nó são maiores (informação/chave) que este nó.

Observe que as condições **a** e **b** acima acarretam o fato de que ao se percorre a árvore segundo o algoritmo in order (EPD) as informações contidas nos nós serão apresentadas de maneira ordenada.

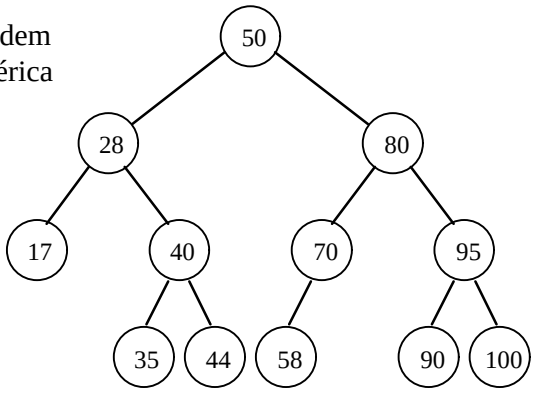
Este tipo de árvore é apropriada para realizarmos operações de inserção, remoção e busca.

Exemplos

a) ordem alfabética

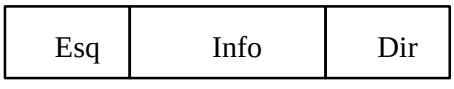


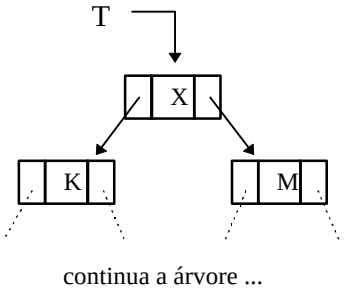
b) ordem numérica



Os algoritmos de inserção, busca e remoção utilizarão a mesma estrutura de dados baseada em alocação dinâmica utilizada para árvores binárias.

Esquema de estrutura ligada com alocação dinâmica





continua a árvore ...

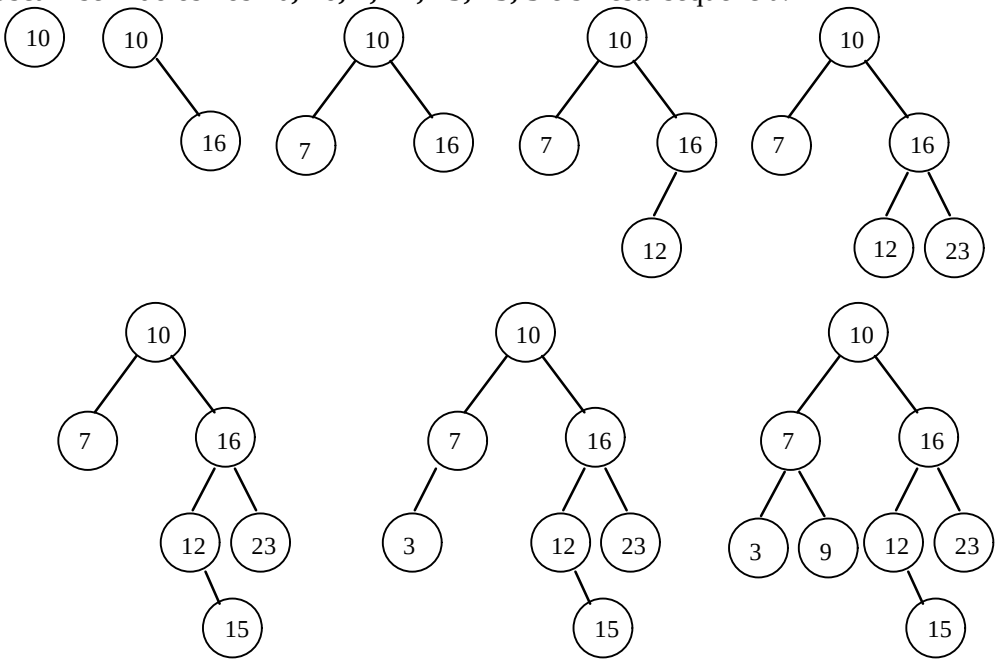
Algoritmo de inserção numa árvore binária de busca

A inserção numa árvore binária de busca é realizada obedecendo as suas propriedades, isto é, os nós de uma subárvore esquerda de um nó são sempre menores que este nó e os nós de uma subárvore direita de um nó são sempre maiores que este nó.

Observe que ao inserimos um nó numa árvore binária de busca, o mesmo torna-se uma folha da árvore.

```
Algoritmo InsABB(T,Elem)
  Se T == NULL então
    T = alocar espaço
    T->Info = Elem
    T->Esq = NULL
    T->Dir = NULL
  senao
    Se Elem < T->Info então InsABB(T->Esq,Elem)
    senao
    Se Elem > T->Info então InsABB(T->Dir,Elem)
  fimse
```

Considere a utilização do algoritmo acima para montar uma árvore binária de busca inserindo os nós **10, 16, 7, 12, 23, 15, 3** e **9** nesta seqüência.



Algoritmo de busca numa árvore binária de busca

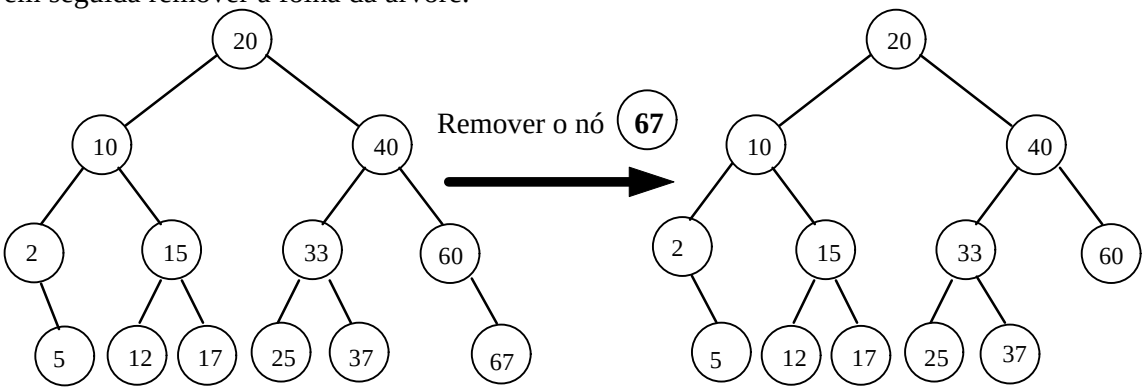
```
Algoritmo BuscaABB(T,Elem)
  Se T == NULL então
    escreva(“Nó contendo Elem não encontrado”)
  senão
    Se ( T->Info > Elem) então BuscaABB(T->Esq,Elem)
    senão
    Se ( T->Info < Elem) então BuscaABB(T->Dir,Elem)
    senão
    escreva(“Elemento desejado”,T->Info)
  fimse
```

Algoritmo de remoção numa árvore binária de busca

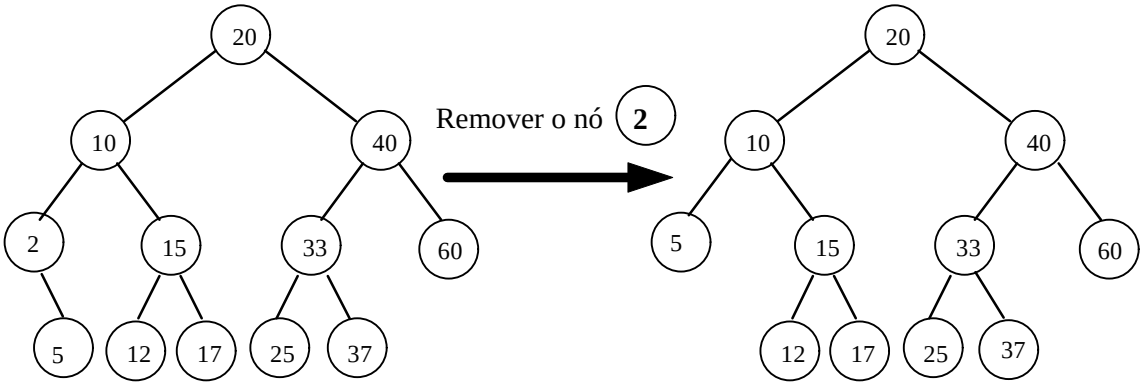
Na remoção de um nó de uma árvore binária de busca devemos nos certificar da manutenção do seu ordenamento (subárvore esquerda menor e subárvore direita maior).

O grau de dificuldade na operação de remoção de um nó de uma árvore binária de busca depende de quantos filhos tem este nó. Podemos dividir em três casos: nós sem filhos (folhas), nós com um único filho e nós com dois filhos.

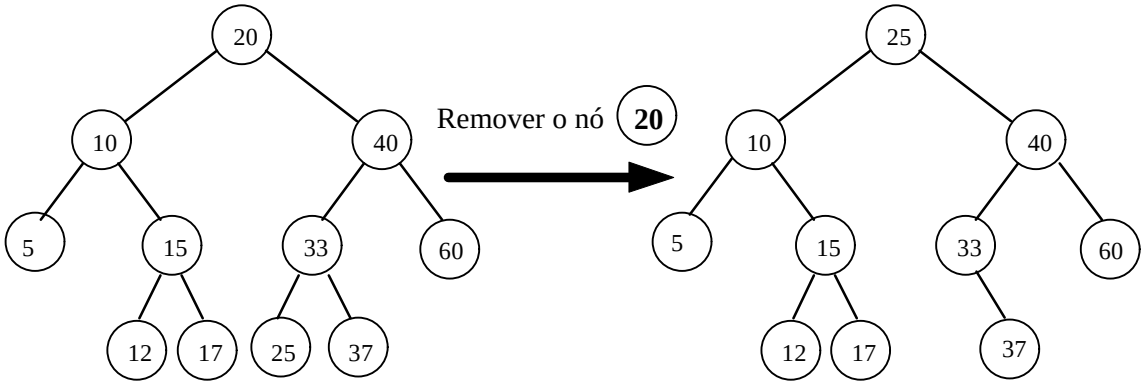
a) nós sem filhos (folhas): devemos ajustar o apontador do pai correspondente, e em seguida remover a folha da árvore.

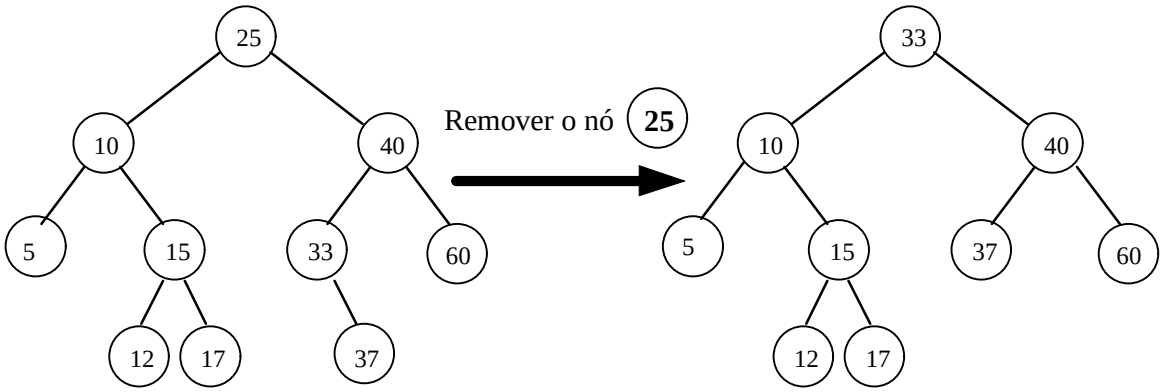


b) nó com um único filho: devemos ajustar o apontador do pai correspondente ao nó, fazendo-o apontar para o filho do nó que vai ser removido, e em seguida remover o nó da árvore.



c) nó com dois filhos: precisamos ser cuidadosos para preservar a ordenação da árvore. Para se efetuar uma remoção e ainda preservar a ordenação, devemos substituir o nó a ser removido pelo seu **Sucessor Imediato**. O sucessor imediato de um nó é o seu descendente mais a esquerda a partir da sua subárvore direita.





Algoritmo de remoção numa árvore binária de busca

```
Algoritmo Substitue(T,suc)
  Se suc->Esq == NULL então
    T->Info = suc->Info
    aux = suc
    suc = suc->dir
    Liberar espaço(aux)
  senão
    Substitue(T,suc->Esq)
  fimse

Algoritmo Remove(T,Elem)
  T = Busca(Raiz,Elem) //encontra o nó a ser removido
  Se T->Esq = NULL     //o nó não tem filho ou só tem filho dir
    aux = T
    T = T->Dir
    Liberar espaço(aux)
  senão
    Se T->Dir == NULL  //o nó só tem filho esq
      aux = T
      T = T->Esq
      Liberar espaço(aux)
    senão
      Substitue(T,T->Dir)
    fimse
  fimse
```

Algoritmos para inserção, busca e remoção em árvores binárias de busca utilizando representação ligada com alocação dinâmica (arquivo ProjEx601.dpr e Ex601.pas).

```
TNo = class
  protected
    Info: string;
    Dir, Esq: TNo;
  public
    Constructor InicNo(str: string);
    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;
    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;

var
  FormABB: TFormABB;
  T01: TABB;
  Elem: TNo;
  StrGlobal: String;

Function TABB.InsABB(var T:TNo;x: string):boolean;
begin
  if ( T = NIL ) then
  begin
    T := TNo.InicNo(x);
    InsABB := TRUE;
  end
  else
  begin
    if ( x > T.Info ) then InsABB(T.Dir,x)
```

```
    else if ( x < T.Info ) then InsABB(T.Esq,x)
    else InsABB := FALSE;
  end
end;

Function TABB.Inserir(x:string):boolean;
begin
  Inserir := InsABB(Raiz,x);
end;

Function TABB.BuscaABB(var T:TNo; x: string):TNo;
begin
  if ( T = NIL ) then BuscaABB := NIL
  else
  begin
    if ( x > T.Info ) then BuscaABB(T.Dir,x)
    else if ( x < T.Info ) then BuscaABB(T.Esq,x)
    else BuscaABB := T;
  end
end;

Function TABB.Buscar(x:string):TNo;
begin
  Buscar := BuscaABB(Raiz,x);
end;


Procedure TABB.Substitue(var T,suc: TNo);
var aux: TNo;
begin
  if ( suc.Esq = NIL ) then
  begin
   T.Info := suc.Info;
   aux := suc;
   suc := suc.Dir;
   aux.Free;
  end
  else Substitue(T,suc.Esq);
end;

Function TABB.DelABB(var T: TNo;x: string):boolean;
var aux: TNo;
begin
  if ( T <> NIL ) then
  begin
    if ( x < T.Info ) then DelABB := DelABB(T.Esq,x)
    else if ( x > T.Info ) then DelABB := DelABB(T.Dir,x)
    else
    begin
      if ( T.Esq = NIL ) then
      begin
        aux := T;
        T := T.Dir;
        aux.Free;
      end
      else if ( T.Dir = NIL ) then
      begin
        aux := T;
        T := T.Esq;
        aux.Free;
      end
```

```
      else Substitue(T,T.Dir);
      DelABB := TRUE;
    end
  end
  else DelABB := FALSE;
end;

Function TABB.Remover(x:string):boolean;
begin
  Remover := DelABB(Raiz,x);
end;
```

Os algoritmos para inserção, busca e remoção desenvolvidos acima utilizam-se da abordagem recursiva, entretanto podemos implementá-los de forma não recursiva como mostrado abaixo (arquivo ProjEx602.dpr e Ex602.pas):

```
Function TABB.InsABB(var T:TNo;x: string):boolean;
var p,q,v: TNo;
    ExisteNo: boolean;
begin
  q := NIL;
  p := T;
  ExisteNo := FALSE;
  while ( p <> NIL ) do
  begin
    if ( x = p.Info ) then
    begin
      ExisteNo := TRUE;
      break;
    end
    else
    begin
      q := p;
      if ( x < p.Info ) then p := p.Esq
      else p := p.Dir;
    end
  end;
  if ( not ExisteNo ) then
  begin
    v := TNo.InicNo(x);
    if ( q = NIL ) then T := v
    else
    begin
     if ( x < q.Info ) then q.Esq := v
     else q.Dir := v;
    end
  end;
  InsABB := not ExisteNo;
end;

Function TABB.Inserir(x:string):boolean;
begin
  Inserir := InsABB(Raiz,x);
end;

Function TABB.BuscaABB(T:TNo; x: string):boolean;
var p: TNo;
    achou: boolean;
begin
  achou := FALSE;
```

```
  p := T;
  while ( (p <> NIL) and (not achou) ) do
  begin
    if ( x < p.Info) then p := p.Esq
    else if (x > p.Info) then p := p.Dir
    else
      achou := TRUE;
  end;
  BuscaABB := achou;
end;

Function TABB.Buscar(x:string):boolean;
begin
  Buscar := BuscaABB(Raiz,x);
end;


Function TABB.DelABB(var T: TNo;x: string):boolean;
var p,q,rp,f,s: TNo;
    achou: boolean;
begin
  q := NIL;
  p := T;
  achou := FALSE;
  while ( (p<>NIL) and (not achou) ) do
  begin
    if ( x < p.Info) then
    begin
      q := p;
      p := p.Esq
    end
    else if ( x > p.Info) then
    begin
      q := p;
      p := p.Dir
    end
    else achou := TRUE;
  end;
  if ( p = NIL ) then DelABB := FALSE
  else
  begin
    if ( p.Esq = NIL ) then rp := p.Dir
    else if ( p.Dir = NIL ) then rp := p.Esq
    else
    begin
      f := p;
      rp := p.Dir;
      s := rp.Esq;
      while ( s <> NIL ) do
      begin
        f := rp;
        rp := s;
        s := rp.Esq;
      end;
      if ( f <> p ) then
      begin
        f.Esq := rp.Dir;
        rp.Dir := p.Dir;
      end;
      rp.Esq := p.Esq;
      if ( p = T ) then T := rp;
```

```
    end;
    if ( q = NIL ) then T := rp
    else
    begin
      if ( p = q.Esq) then q.Esq := rp
      else q.Dir := rp;
    end;
    p.Free;
    DelABB := TRUE;
  end;
end;

Function TABB.Remover(x:string):boolean;
begin
  Remover := DelABB(Raiz,x);
end;
```

Exercícios

6.01) Monte uma árvore binária de busca inserindo os nós abaixo na ordem apresentada:

a) 8, 26, 6, 12, 20, 30, 10, 5, 2, 7
b) 80, 90, 95, 85, 60, 55, 65, 20, 30, 40, 25, 10, 70, 62, 87
c) 20, 15, 50, 40, 12, 17, 45, 35, 13, 14, 47, 46

6.02) Monte uma árvore binária de busca inserindo os nós 50, 25, 80, 10, 30, 60, 90, 5, 27, 70, 85, 95, 26, 29, 65 e 82, 97. Após montada a árvore efetue as seguintes remoções:

a) nó 5 b) nó 30
c) nó 80 d) nó 27
e) nó 50 f) nó 90

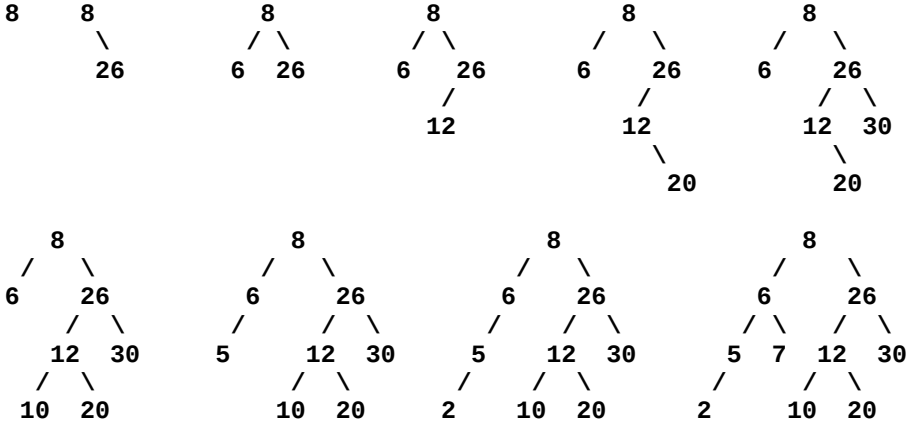
6.03) Escreva um algoritmo para encontrar o menor e o maior elemento de uma árvore binária de busca.

6.04) Elabore um programa, que implemente uma agenda utilizando árvore binária de busca (apontadores e alocação dinâmica). Cada nó da árvore deve conter os campos Nome, Telefone e Idade. A árvore deve ser ordenada pelo campo Nome.

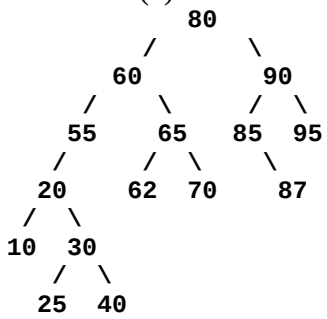
6.05) Modifique a função de deleção de árvore binária de busca para utilizar o antecessor imediato (maior elemento da subárvore esquerda do nó a ser removido) ao contrário do sucessor imediato.

Respostas

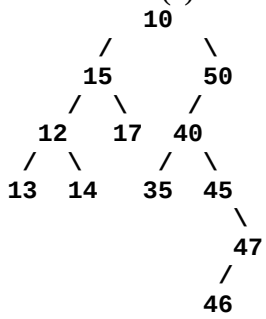
6.01) item (a) - passo a passo



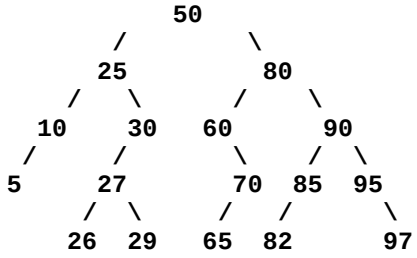
item (b) árvore final



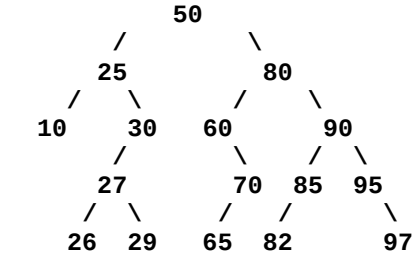
item (c) árvore final



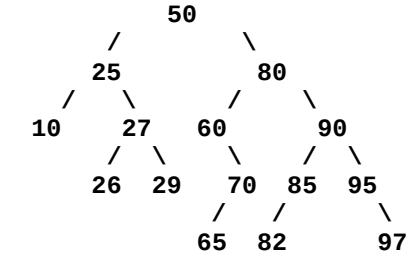
6.02) árvore binária após inserir os nós



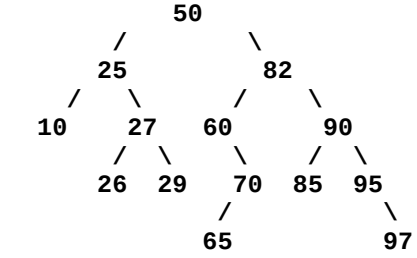
a)remoção do nó 5



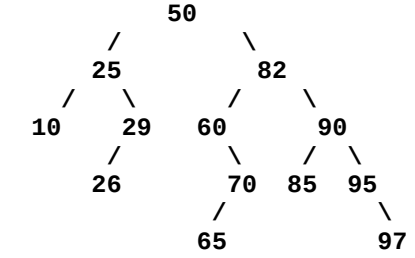
b) remoção do nó 30



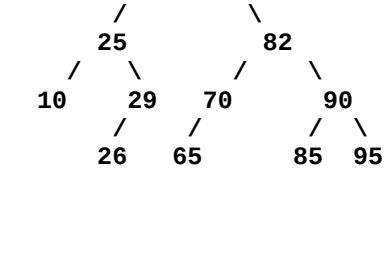
c)remoção do nó 80



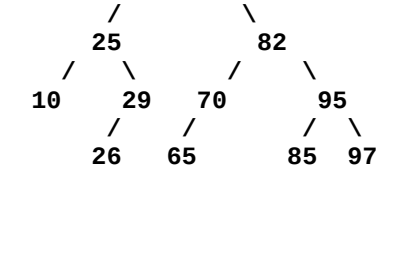
d) remoção do nó 27



e)remoção do nó 50



f) remoção do nó 90



```
\
97
```

6.03) algoritmo p/ encontrar o menor elemento de uma árvore binária de busca (o algoritmo para encontrar o maior elemento é similar ao apresentado abaixo).

```
Algoritmo MenorAbb
  Se T != NULL então
    aux = T
    Enquanto aux->Esq != NULL
      aux = aux->Esq
    fimEnquanto
    Escreva("Menor: ",aux->Info)
  fimse
```

6.04)Basta alterar a estrutura do programa de inserção, deleção e busca já apresentado e reescrever os algoritmos baseando-se na nova estrutura.

Sugestão

```
TNo = class
  protected
    Nome, Fone: string;
    Idade: integer;
    Dir, Esq: TNo;
  Public
    //métodos
  end;

TArvBin = Class
  protected
    Raiz: TNo;
  Public
    //métodos
  end;

TABB = Class(TArvBin)
    //métodos
  end;
```

6.05) Este é por sua conta. Divirta-se !