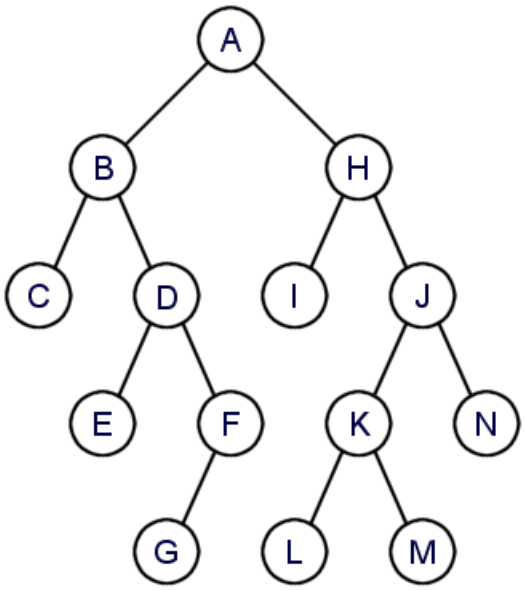
- **Travessias de uma Árvore Binária**:



Pré-Ordem:
1º raiz;
2º sub-árvore esquerda;
3º sub-árvore direita.

{ A B C D E F G H I J K L M N}

Em-Ordem:
1º sub-árvore esquerda;
2º raiz;
3º sub-árvore direita.

{ C B E D G F A I H L K M J N}

Pos-Ordem:
1º sub-árvore esquerda;
2º sub-árvore direita;
3º raiz.

{ C E G F D B I L M K N J H A}

Travessia por Níveis: **{ A B H C D I J E F K N G L M }**

- **Versões Recorrentes**

```
procedure TravessiaPreOrdem (t : ArvoreBinaria);
          begin if not vazia(t)
                then begin visitar(t);
                           TravessiaPreOrdem(esquerda(t));
                           TravessiaPreOrdem(direita(t))
                     end
          end; { TravessiaPreOrdem }
```

```
procedure TravessiaEmOrdem (t : ArvoreBinaria);
          begin if not vazia(t)
                then begin TravessiaEmOrdem(esquerda(t));
                           visitar(t);
                           TravessiaEmOrdem(direita(t))
                     end
          end; { TravessiaEmOrdem }
```

```
procedure TravessiaPosOrdem (t : ArvoreBinaria);
          begin if not vazia(t)
                then begin TravessiaPosOrdem(esquerda(t));
                           TravessiaPosOrdem(direita(t));
                           visitar(t);

                     end
          end; { TravessiaPosOrdem }
```

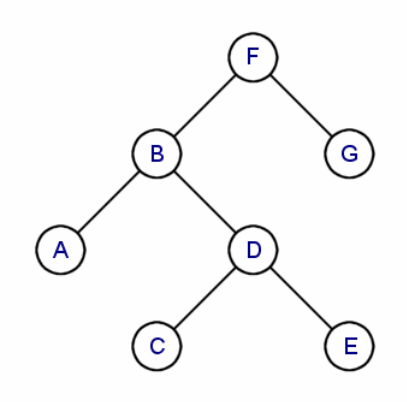
- **Versões Iterativas**

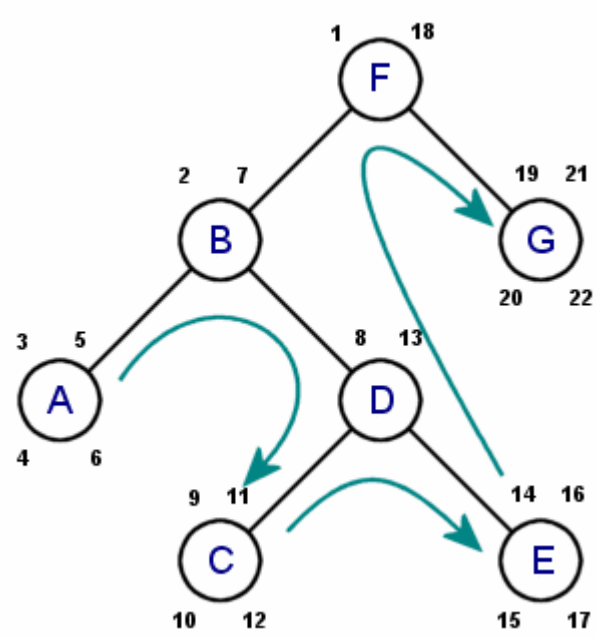
```
procedure TravessiaPreOrdemIterativa (raiz : ArvoreBinaria);
        var   p : ArvoreBinaria;
              S : PilhadeArvoresBinarias;

        begin  criar(S);

               { começar por guardar a raiz se existir }
               if raiz <> nil
               then por(raiz, S);


               while not vazia(S) do

                    begin  { visitar o topo da Pilha }
                           p:= topo(S);
                           visitar(p);
                           { e remover }
                           tirar(S);

                           { guardar direita se existir }
                           if direita(p) <> nil
                           then por(direita(p), S);

                           { guardar esquerda se existir }
                           if esquerda(p) <> nil
                           then por(esquerda(p), S)

                    end


        end; { TravessiaPreOrdemIterativa  }
```

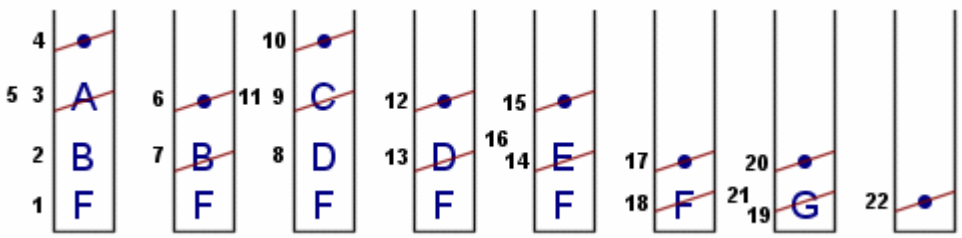
```
procedure TravessiaPorNiveisIterativa (raiz : ArvoreBinaria);
        var   p : ArvoreBinaria;
              Q : FiladeArvoresBinarias;

        begin  criar(Q);

               { começar por guardar a raiz se existir }
               if raiz <> nil
               then juntar(raiz, Q);


               while not vazia(Q) do

                    begin  { visitar a frente da fila }
                           p:= frente(Q);
                           visitar(p);
                           { e remover }
                           remover(Q);

                           { juntar esquerda se existir }
                           if esquerda(p) <> nil
                           then juntar(esquerda(p), Q);

                           { juntar direita se existir }
                           if direita(p) <> nil
                           then juntar(direita(p), Q)

                    end


        end; { TravessiaPorNiveisIterativa  }
```

```
procedure TravessiaEmOrdemIterativa (raiz : ArvoreBinaria);
        var   p : ArvoreBinaria;
              S : PilhadeArvoresBinarias;
              acabou : boolean;

        begin  criar(S);
               p:= raiz;
               { começar por guardar a raiz }
               por(p, S);
               acabou:= false;

               while not acabou do

                    if p <> nil
                    then  begin  { descer para a esquerda e guardar }
                                 p:= esquerda(p);
                                 por(p, s)
                          end

                    else  begin { subir a partir da sub-árvore vazia }
                                 tirar(S); { remover a sub-árvore vazia }

                                 { consultar a Pilha }
                                 if vazia(S)
                                 then  { toda a árvore foi visitada }
                                       acabou:= true

                                 else  begin  { visitar o topo da Pilha }
                                              p:= topo(S);
                                              visitar(p);
                                              { e remover }
                                              tirar(S);

                                              { descer para a direita }
                                              p:= direita(p);
                                              { e guardar }
                                              por(p, S)
                                       end
                          end

        end; { TravessiaEmOrdemIterativa }
```

Por exemplo,

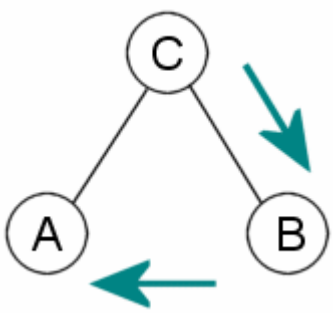






visita: A B C D E F G vazia(S)

```
procedure TravessiaPosOrdemIterativa (raiz : ArvoreBinaria);
var  p, ant : ArvoreBinaria;
     S : PilhadeArvoresBinarias;

begin  criar(S);
       { começar por guardar a raiz }
       por(raiz, S);
       ant:= nil;

       while  not vazia(S) do
              begin  p:= topo(S);
                     tirar(S);

                     if (direita(p) = nil ) and (esquerda(p) = nil)
                     then  begin  { é uma folha: visitar e marcar  }
                                  ant:= p;
                                  visitar(p)
                           end
                     else  if (direita(p) = ant) or (esquerda(p) = ant)
                           then  begin  { subiu na árvore: visitar e marcar  }
                                        ant:= p;
                                        visitar(p)
                                 end
                           else  begin  { não subiu nem é uma folha }
                                        { guardar este outra vez }
                                        por(p, S);

                                        { guardar a sua direita se existir }
                                        if direita(p) <> nil
                                        then por(direita(p), S);

                                        { guardar a sua esquerda se existir }
                                        if esquerda(p) <> nil
                                        then por(esquerda(p), S)
                                 end
              end

end; { TravessiaPosOrdemIterativa }
```



Exercícios:

1. Construir uma árvore binária, sendo dada a sua travessia pré-ordem com informação do tipo {0, 1} sobre as sub-árvores de cada vértice.

2. Construir uma árvore binária, sendo dadas as suas travessias pré-ordem e em-ordem.

3. Construir uma árvore binária, sendo dadas as suas travessias pré-ordem e pos-ordem.

4. Construir uma árvore binária, sendo dadas as suas travessias em-ordem e pos-ordem.

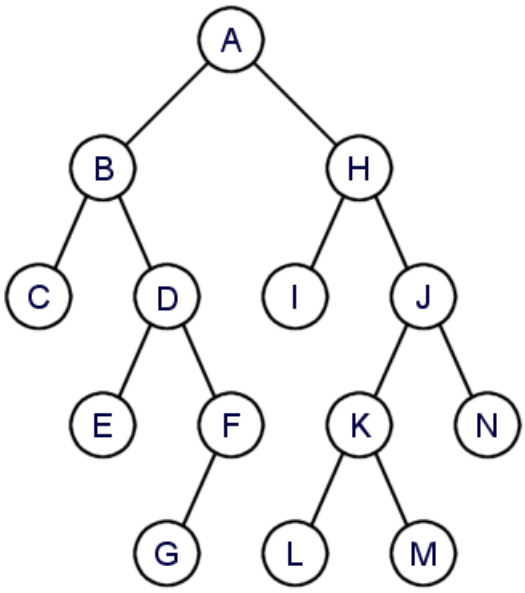
5. Calcular o sucessor de um dado vértice, em:
   a) pré-ordem
   b) em-ordem
   c) pos-ordem

6. Calcular o predecessor de um dado vértice, em:
   a) pré-ordem
   b) em-ordem
   c) pos-ordem

7. Calcular a maior sub-árvore perfeita de uma dada árvore binária.

8. ...

- **Construção de uma árvore binária, sendo dadas as suas travessias pré-ordem e em-ordem.**



dados: pre[1..n]
em[1..n]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
pre	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**
em	**C**	**B**	**E**	**D**	**G**	**F**	**A**	**I**	**H**	**L**	**K**	**M**	**J**	**N**

- Identificar casos particulares (base da recorrência):

  n = 0 (?)
  n = 1 (?)

- Identificar a **raiz** : em[raiz] = pre[1]

	1	2	3	4	5	6	**7**	8	9	10	11	12	13	14
pre	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**
em	**C**	**B**	**E**	**D**	**G**	**F**	**A**	**I**	**H**	**L**	**K**	**M**	**J**	**N**

- Identificar as **sub-árvores**

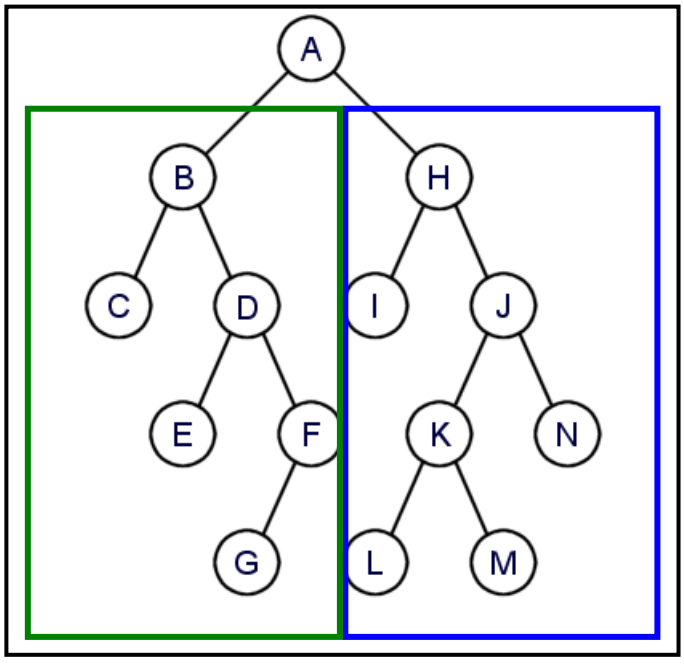
	1	2	3	4	5	6	**7**	8	9	10	11	12	13	14
pre	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**
em	**C**	**B**	**E**	**D**	**G**	**F**	**A**	**I**	**H**	**L**	**K**	**M**	**J**	**N**

esquerda: pre[2..raiz]
em[1..raiz-1]

direita: pre[raiz+1..n]
em[raiz+1..n]

- Construir as **chamadas** recorrentes:

	1	2	3	4	5	6	**7**	8	9	10	11	12	13	14
pre	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**
em	**C**	**B**	**E**	**D**	**G**	**F**	**A**	**I**	**H**	**L**	**K**	**M**	**J**	**N**



- **Algoritmo recorrente:**

construir (t , pre[1..n], em[1..n]):

se n = 0 | se n = 1
então t ← nil | então criar(t)

senão calcular raiz : em[raiz] = pre[1]

criar(t)
colocar pre[1]

construir (esquerda , pre[2..raiz], em[1..raiz-1])

construir (direita , pre[raiz+1..n], em[raiz+1..n])

- Decisões de implementação:

- caso(s) base

- função ou procedimento?

- escolha dos parâmetros

- ...

- **Construção de uma árvore binária, sendo dadas as suas travessias pré-ordem e pos-ordem.**

	1	2	3	4	5	**6**	7	8	9	10	11	12	13	14
pre	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**
pos	**C**	**E**	**G**	**F**	**D**	**B**	**I**	**L**	**M**	**K**	**N**	**J**	**H**	**A**

construir (t , pre[1..n], pos[1..n]):

se n = 0
então t ← nil

se n = 1
então criar(t)

senão calcular raiz : pos[raiz] = pre[2]

criar(t)
colocar pre[1]

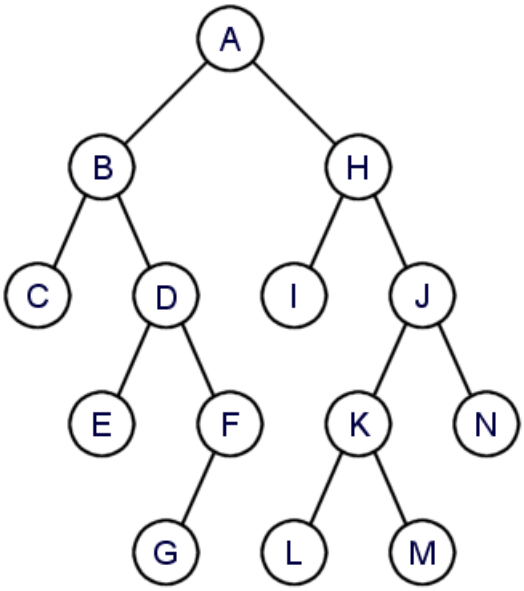
construir (esquerda , pre[2..raiz+1], pos[1..raiz])

construir (direita , pre[raiz+2..n], pos[raiz+1..n-1])

- **Construção de uma árvore binária, sendo dadas as suas travessias em-ordem e pos-ordem.**

	1	2	3	4	5	6	**7**	8	9	10	11	12	13	14
em	**C**	**B**	**E**	**D**	**G**	**F**	**A**	**I**	**H**	**L**	**K**	**M**	**J**	**N**
pos	**C**	**E**	**G**	**F**	**D**	**B**	**I**	**L**	**M**	**K**	**N**	**J**	**H**	**A**

```
type  AB  =  ^no;
      no  =  record  elemento : tipo;
                     esq, dir : AB
             end;
```



Exercícios:

1. Dados dois elementos x e y, verificar se x é um **antepassado** de y.

2.
   a) Listar todos os elementos, indicando o **nível** em que cada um se encontra.
   b) Construir uma tabela x[1 .. h] com o número de elementos que se encontram em cada nível.
   c) Identificar o nível em que se encontra um dado elemento.

3. Verificar se os elementos estão ou não dispostos por **ordem** não-decrescente em:
   a) pre-ordem.
   b) em-ordem.
   c) pos-ordem.

4.
   a) Calcular o **tamanho** (número de elementos) de uma AB.
   b) Listar todos os elementos, indicando o tamanho de cada uma das sub-árvores correspondentes.
   c) Calcular o tamanho da sub-árvore correspondente a um dado elemento.