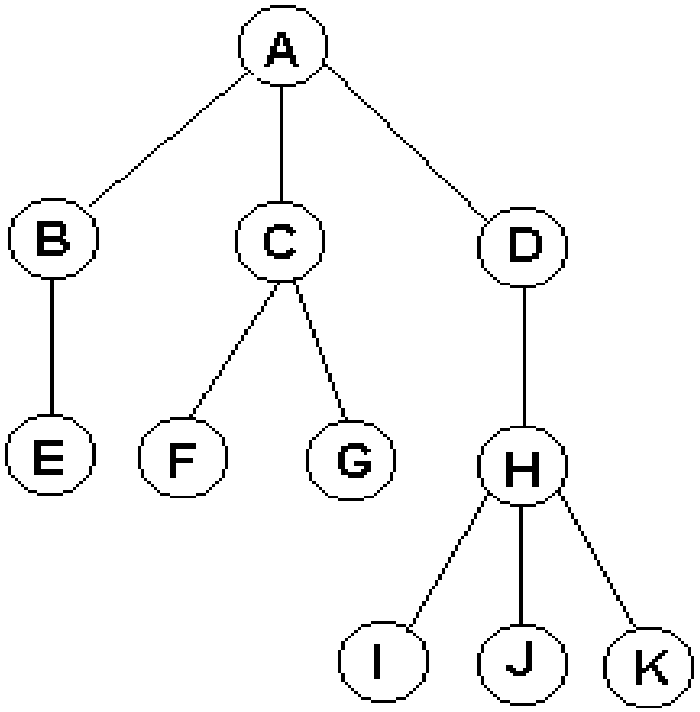
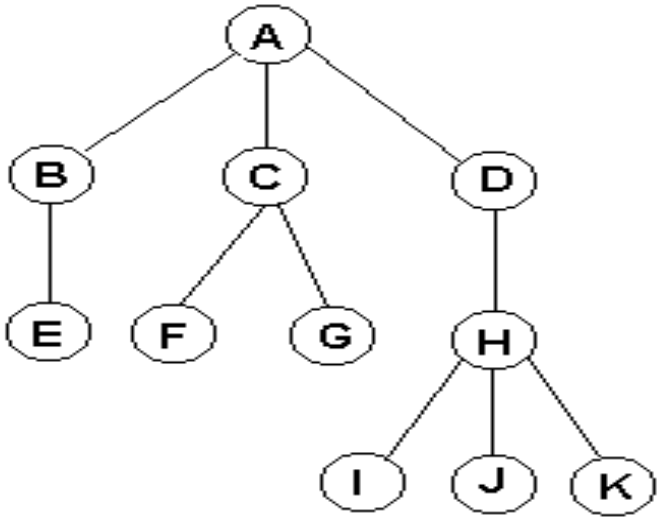
ÁRVORES

Uma árvore é uma estrutura não linear que representa relações de hierarquia e composição (um conjunto de dados é hierarquicamente subordinado a outro).



Definição formal: conjunto finito de um ou mais nós, tais que:

- Existe um nó denominado raiz da árvore;
- Os demais nós formam m ≥ 0 conjuntos disjuntos S_1, S_2, ... , S_m, onde cada um desses conjuntos é uma árvore.



As árvores S_i ($1 \leq i \leq m$) recebem a denominação de subárvores.

Terminologia:

GRAU de um nó: número de subárvores desse nó.

Nós TERMINAIS ou FOLHAS: aqueles que têm grau zero (não possuem subárvores).

GRAU DA ÁRVORE: grau máximo possível para todos os nós.

Obs: nem sempre haverá um grau máximo (em algumas árvores o número de filhos para um nó é indefinido)

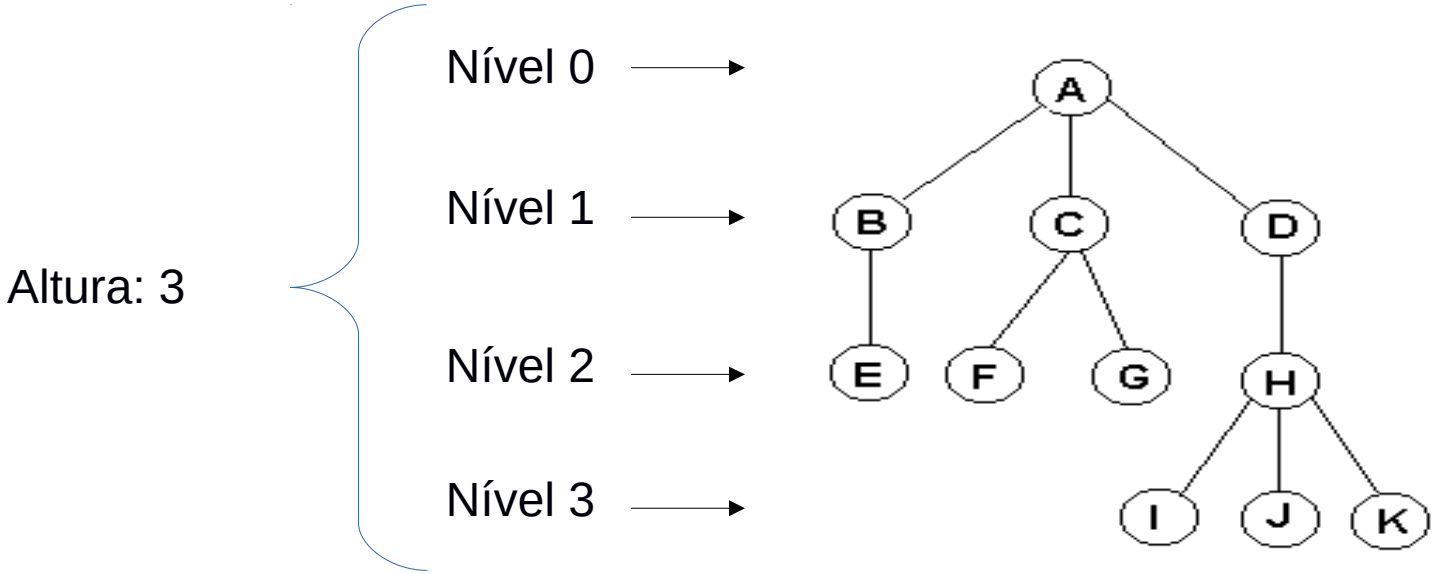
ÁRVORE HOMOGÊNEA: aquela em que todos os nós possuem as mesmas características (mesmo grau máximo e mesmo tipo de informação).

Obs: para identificar os nós de uma estrutura usam-se relações de parentesco, como: pai, filho, irmão, etc.

NÍVEL: distância de um nó até a raiz.

- a raiz por definição tem nível zero.
- para outro nó qualquer, o nível é o número de arcos que o liga à raiz.

ALTURA: o nível mais alto da árvore.



FLORESTA: conjunto de zero ou mais árvores disjuntas.

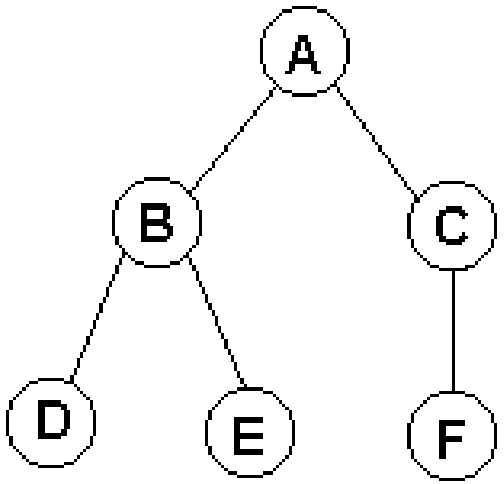
Quanto à ordenação, uma árvore pode ser:

- Não ordenada: a ordem dos filhos é irrelevante para a aplicação (apenas a hierarquia é importante).
- Ordenada: (leva-se em consideração a ordem dos filhos – 1o filho, segundo, etc.).

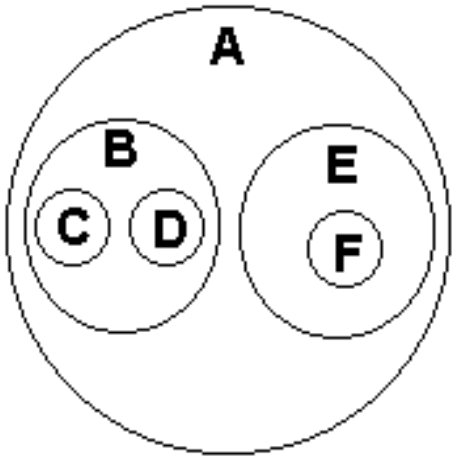
Representações:

Padrão mais usado:

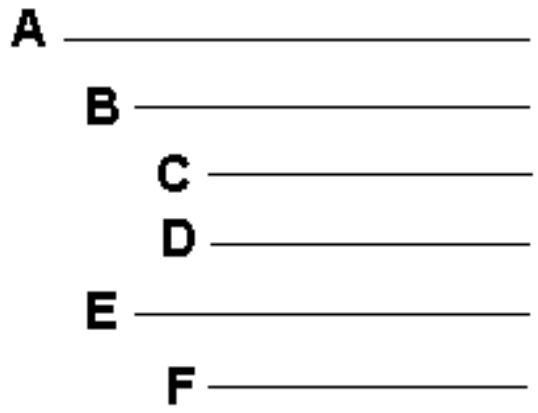
Representação cima-baixo (noção de detalhamento progressivo)



Outras representações possíveis:



(a)



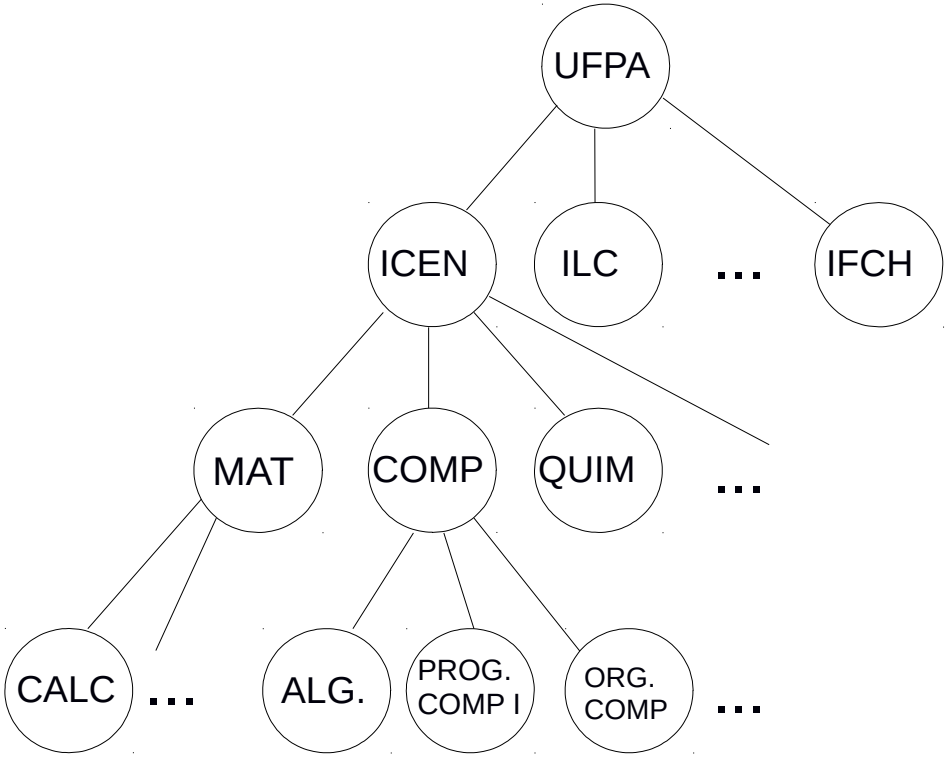
(b)

1 A; 1.1 B; 1.1.1 C; 1.1.2 D; 1.2 E 1.2.1 F

(C)

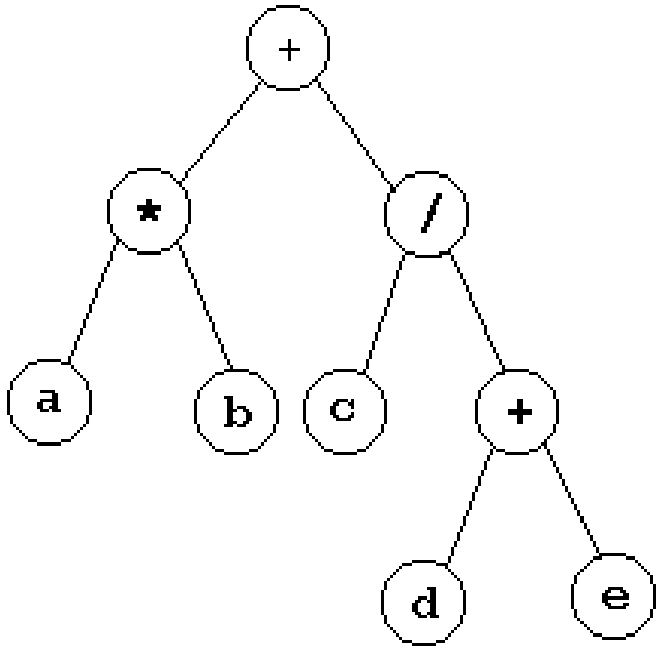
Aplicações de árvores: situações onde os dados a serem representados possuem relações hierárquicas entre si.

Ex.1: relações hierárquicas envolvendo instâncias administrativas.



Ex. 2: representação de uma expressão aritmética

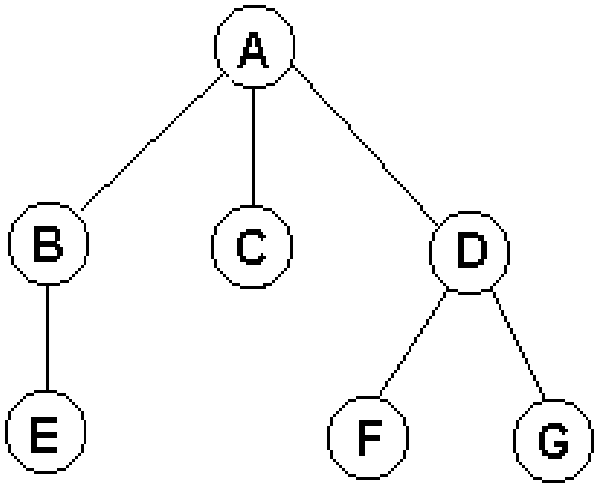
`a*b+c/(d+e)`



Alocação de árvores:

Assim como as listas lineares, as árvores podem ser alocadas por adjacência ou por encadeamento.

Por adjacência: os nós são alocados contiguamente na memória (array), segundo uma ordem convencionada. Ex:

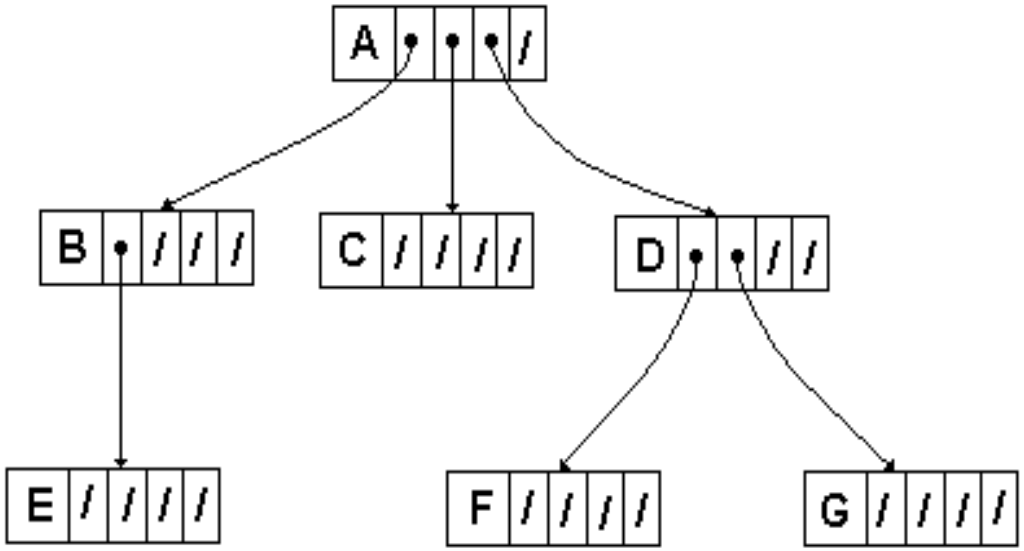


A	3	B	1	C	0	D	2	E	0	F	0	G	0

OBS: A alocação sequencial oferece dificuldades para manipulações (inserções, remoções, localizações de nós).

Alocação encadeada: alternativa em geral mais adequada.

Cada nó é um registro que pode ser alocado dinamicamente e possui espaço tanto para o dado quanto para as referências para todas as subárvores.



O tipo de dado **`No`** deve possuir tantas referências quanto o número máximo de subárvores que um nó possa ter.

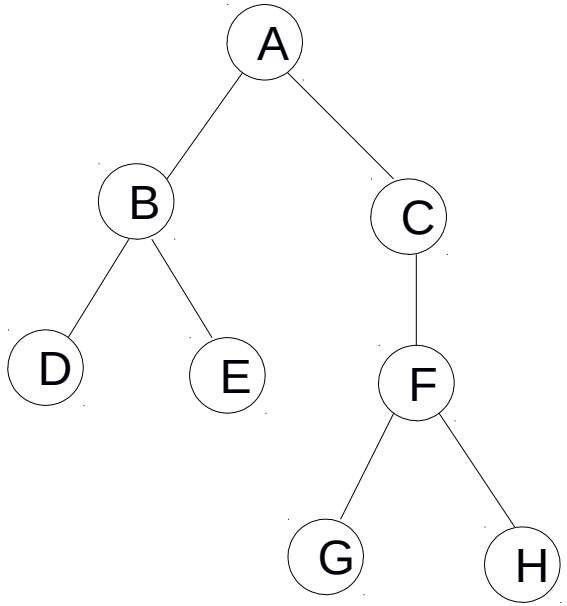
O problema é que assim haverá uma grande quantidade de referências não usadas.

A solução alternativa (mais econômica) é transformar a árvore a ser representada em uma árvore binária.

Árvores Binárias

Nestas árvores todos os nós têm no máximo duas subárvores (árvores de grau 2).

As duas subárvores são denominadas subárvore da esquerda e subárvore da direita. Exemplo:

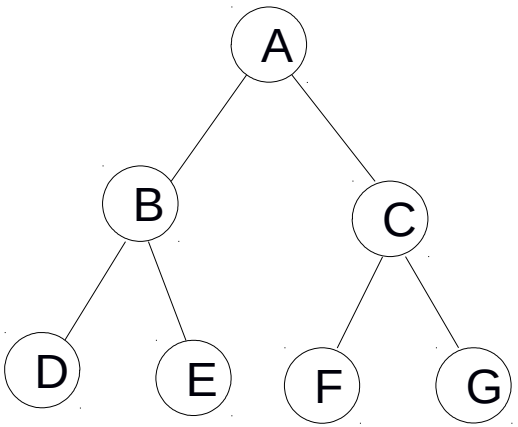


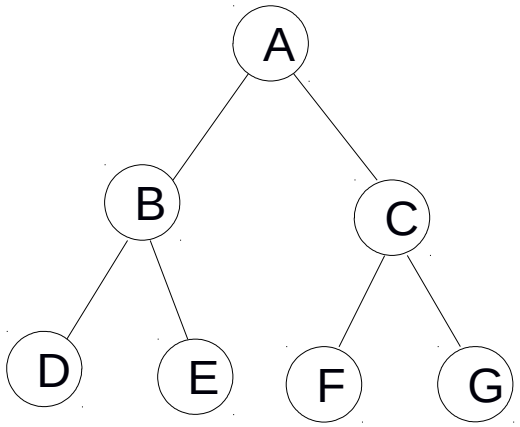
Formas de realização (implementação)

Alocação por Adjacência (contiguidade):

Alocação de espaço: uma árvore binária completa de altura **K**, possui $N = 2^{(K+1)}-1$ nós.

OBS: árvore binária **completa** é aquela em que todos os nós possuem dois filhos, exceto as folhas.





0	1	2	3	4	5	6
A	B	C	D	E	F	G

A partir da posição **`i`** de um nó:

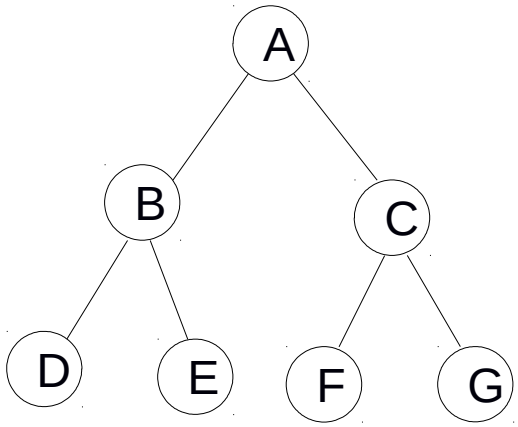
· O pai daquele nó está em `(i-1)div 2`, para `i>0`.

(se **`i=0`**, trata-se da raiz)

Ex: casa 3: (3-1) div 2 = 1

casa 4: (4-1) div 2 = 1

casa 5: (5-1) div 2 = 2



0	1	2	3	4	5	6
A	B	C	D	E	F	G

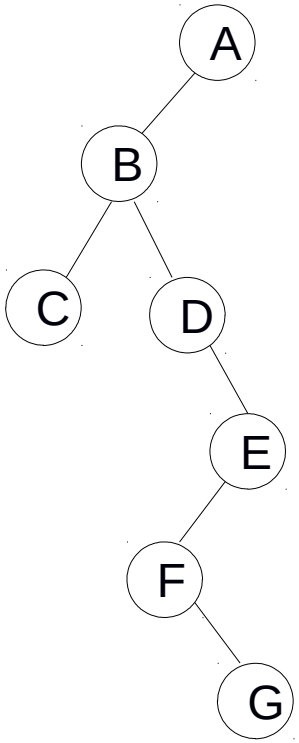
- O filho à esquerda do nó está na posição **`2i+1`**, se **`2i ≤ n-1`** (se **`2i+1 > n-1`**, o nó não tem filho à esquerda).
- O filho à direita do nó está na posição **`2i+2`**, se **`2i+2 ≤ n`** (se **`2i+2 > n-1`**, o nó não tem filho à direita).

Obs: para árvores binárias incompletas haverá, em alguma medida, desperdício de espaço.

Possível solução: subdividir o espaço de um nó (a célula do array) em 3 campos:

dado	pos. filho esq.	pos. filho dir.

```
typedef int tDado;
const int N = 10;
struct NoArv
{
   tDado val;
   unsigned filhEsq;
   unsigned filhDir;
};
typedef NoArv VetArv[N];
```



0	1	2	3	4	5	6
A	B	C	D	E	F	G
1	2	0	0	5	0	0
0	3	0	4	0	6	0

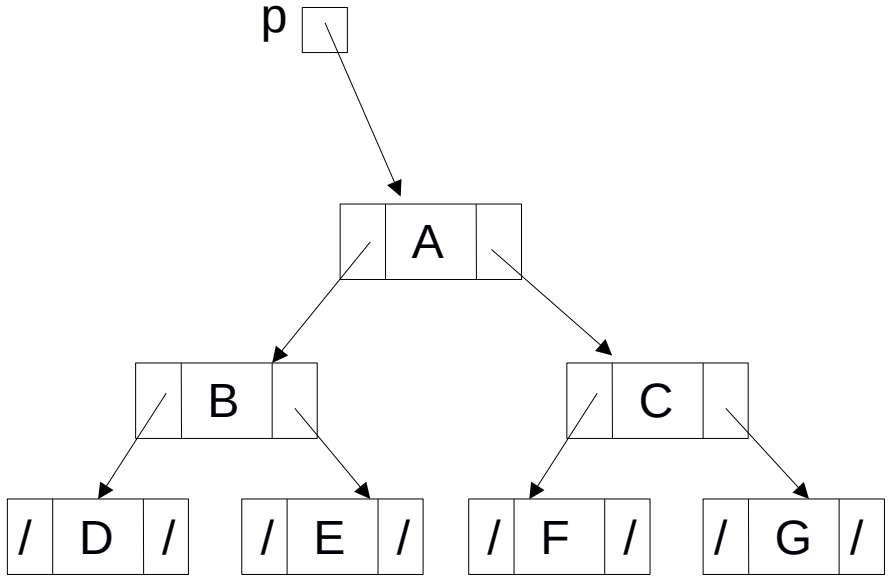
Alocação encadeada

Formato do nó:

PtEsq	Dado	PtDir

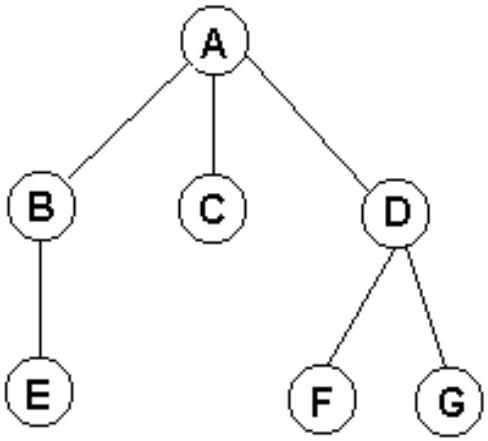
· **PtEsq**: endereço da subárvore da esquerda;
· **PtDir**: endereço da subárvore da direita;

```
struct NoArvBin
{
   NoArvBin * esq;
   tDado dado;
   NoArvBin * dir;
}
typedef NoArvBin * PtNo;
PtNo p;
```

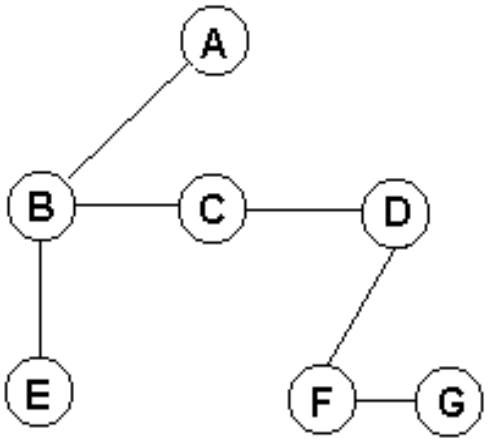


Transformação de árvore qualquer em árvore binária

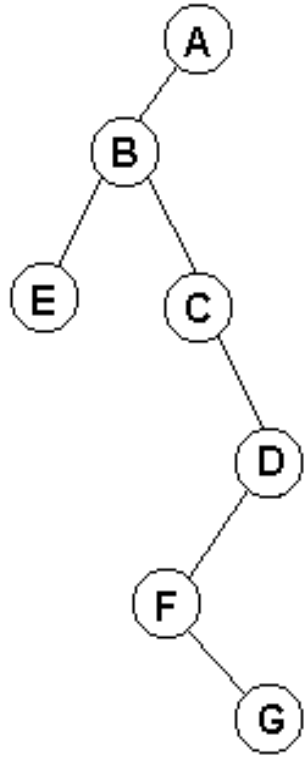
- Ligam-se os nós irmãos
- Removem-se as ligações entre um nó pai e os nós filhos, exceto a relativa ao primeiro filho.



(a)



(b)



(c)

Vantagem da transformação: não é necessário ter conhecimento prévio sobre a estrutura (grau máximo) para fins de alocação encadeada.

OBS: Para se interpretar corretamente a hierarquia em uma árvore transformada em binária, deve-se ter em mente a transformação havida:

- um filho à esquerda de um nó é filho de fato.
- um filho à direita de um nó é seu irmão

Percurso

Percorrer uma árvore é visitar de forma sistemática e ordenada cada nó, apenas uma vez.

OBS: visitar um nó significa ter acesso ao seu conteúdo, para uma operação qualquer (exibir os elementos, somá-los, etc.)

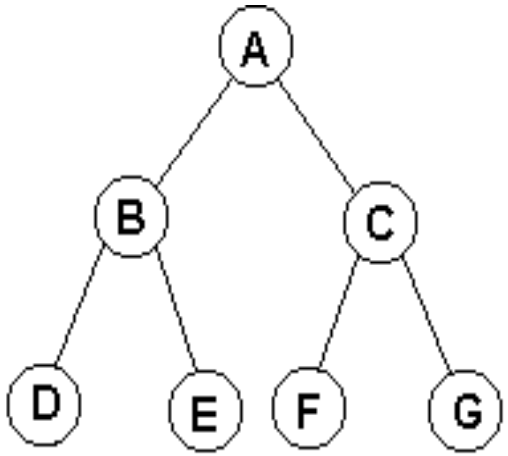
Principais percursos:

Pré-ordem:

· Visita-se a raiz

· Percorre-se a subárvore da esquerda (em pré-ordem)

· Percorre-se a subárvore da direita (em pré-ordem)



Nesse percurso, a ordem em que os nós são visitados é:

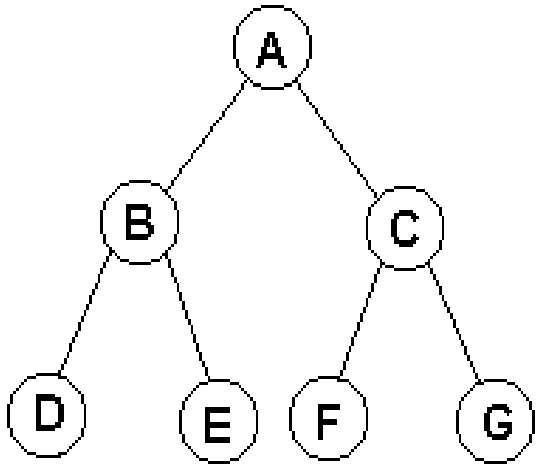
A B D E C F G

Ordem simétrica (ou ordem central):

· Percorre-se a subárvore da esquerda (em ordem simétrica)

· Visita-se a raiz

· Percorre-se a subárvore da direita (em ordem simétrica)

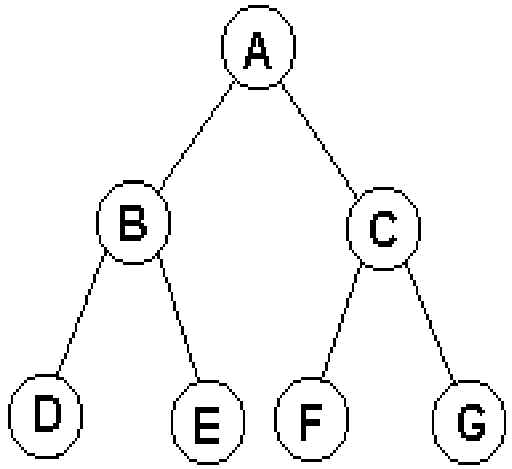


Nesse percurso, a ordem em que os nós são visitados é:

D B E A F C G

Ordem final (ou pós-ordem):

· Percorre-se a subárvore da esquerda (em ordem final)
· Percorre-se a subárvore da direita (em ordem final)
· Visita-se a raiz



Nesse percurso, a ordem em que os nós são visitados é:

D E B F G C A

Algoritmos de Percurso

Versões recursivas: ajustam-se muito bem a estruturas de árvores, pois exploram sua natureza estrutural recursiva (uma árvore é composta de subárvores)

```
void PercPreOrdem(PtNo PtRz)
{
   if (PtRz != NULL)
   {
      Visita(PtRz);
      PercPreOrdem(PtRz->esq);
      PercPreOrdem(PtRz→dir);
   }
}
```

```
void PercOrdemSimetr(PtNo PtRz)
{
   if (PtRz != NULL)
   {
      PercOrdemSimetr(PtRz->esq);
      Visita(PtRz);
      PercOrdemSimetr(PtRz->dir);
   }
}
```

```
void PercOrdemFinal(PtNo PtRz)
{
   if (PtRz != NULL)
   {
      PercOrdemFinal(PtRz->esq);
      PercOrdemFinal(PtRz→dir);
      Visita(PtRz);
   }
}
```

Versão não recursiva

Estrutura auxiliar de trabalho: Pilha
Operações

- Empilhar
- Desempilhar (retorna nil se a pilha estiver vazia)

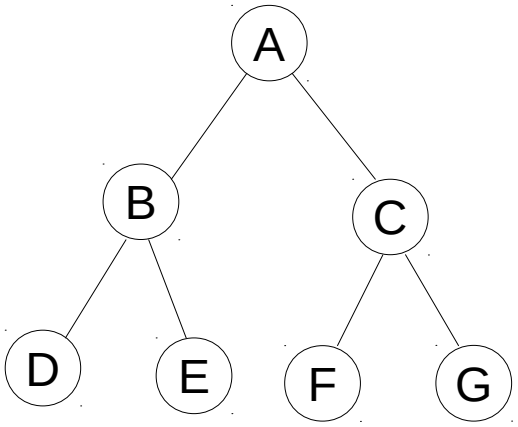
```
void PercursoPreOrdem(PtNo PtRz);
{
   PtNo PtTrab;
   bool fim;

   PtTrab := PtRz;
   fim = false;
   while (! fim)
   {
      while (PtTrab != NULL)
      {
         Utiliza(PtTrab);
         InserePilha(ptTrab);
         PtTrab = PtTrab->esq;
      }
      PtTrab = RetiraPilha();
      if (PtTrab == NULL)
         fim = true;
      else PtTrab = PtTrab->dir;
   }
}
```

Construção de árvores

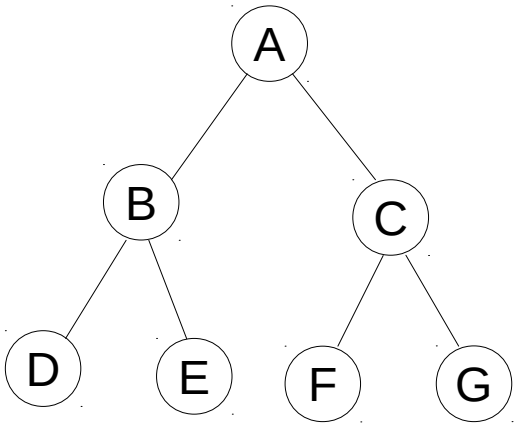
Caso particular:

- Árvore binária
- Alocação encadeada
- Nós serão fornecidos em ordem pré-fixada (raiz, subárvore da esquerda, subárvore da direita.



Ordem pré-fixada: raiz;
subárvore da esquerda;
subárvore da direita

Os dados serão fornecidos em ordem pré-fixada e um ponto (ou outro código, no caso de valores numéricos) indicará uma subárvore vazia. Exemplo:



Ordem de entrada dos dados:

A B D . . E . . C F . . G . .

Versão recursiva

```
void construir_arvore(ptNo & p )
{
   tDado resp;
   cout << "Forneca o valor: ";
   cin >> resp;
   if (resp == '.')    // não construir nó
   {
      p = NULL;
      return;
   }
   // alocar o nó
   p = new No;
   p->dado = resp;

   cout << "Inserção à esquerda do " << p->dado << ". " << endl;
   construir_arvore(p->esq );
   cout << "Inserção à direita do " << p->dado << ". " << endl;
   construir_arvore(p->dir );
}
```