UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS
Centro de Ciências Exatas e Tecnológicas - CENTRO 6

Curso: Informática | Disciplina: Laboratório II | Horário: 31
Prof.: Marinho Barcellos | E-Mail: marinho@exatas.unisinos.br | Data: 19/10/1999

AULA 7 – ÁRVORES

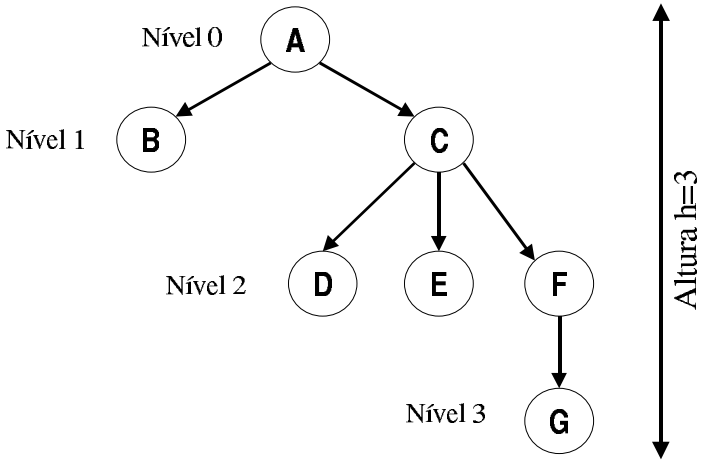
Representação usando árvores

Árvore é a estrutura de dados não linear mais importante no âmbito de algoritmos computacionais. Uma árvore é definida formalmente como um conjunto *T* com um ou mais nodos tal que:

a. existe um nodo especial $\Re$ chamado **RAIZ.**
b. os nodos restantes são particionados em $n \geq 0$ conjuntos disjuntos $T_1..T_n$, sendo que cada subconjunto destes é também é uma árvore (denominada *sub-árvore*)
c. além disso, existe uma aresta ligando o nodo raiz $\Re$ à raiz $\Re_i$ de suas sub-árvores 1..*n*.

Note que esta definição é **RECURSIVA**: uma árvore é definida em termos de suas sub-árvores, sendo que sub-árvores também são árvores. Uma sub-árvore possui seu próprio nodo raiz, que é dito um nodo ***FILHO*** da raiz inicial $\Re$. Não é à toa que os algoritmos eficientes para este tipo de estrutura são também recursivos.

Na figura abaixo temos a representação gráfica de uma árvore:



A árvore acima tem $\Re$ =A; $\Re$ é pai de dois *filhos*, B e C, que por sua vez são também raízes de sub-árvores. A sub-árvore da esquerda tem $\Re$ =B e nenhum *filho*. A árvore da direita, por sua vez, tem $\Re$ =C e três sub-árvores como filhos.

O número de sub-árvores de um nodo é chamado de ***GRAU*** deste nodo. Na árvore acima, o nodo C possui grau 3 . Um nodo com grau 0 é denominado ***FOLHA.*** Além do nodo raiz e dos nodos folha, há os nodos ***INTERNOS***. Nodos internos têm um nodo pai e grau maior do que 0.

Árvore Ordenada x Orientada

Se a ordem relativa das sub-árvores $T_1..T_n$ **é** importante, diz-se que a árvore é ***ordenada***. Se a ordem em um mesmo nível **não** é importante a árvore é dita ser ***orientada***.

Floresta

Uma ***floresta*** é um conjunto (normalmente ordenado) de zero ou mais árvores disjuntas.

Pais e Filhos

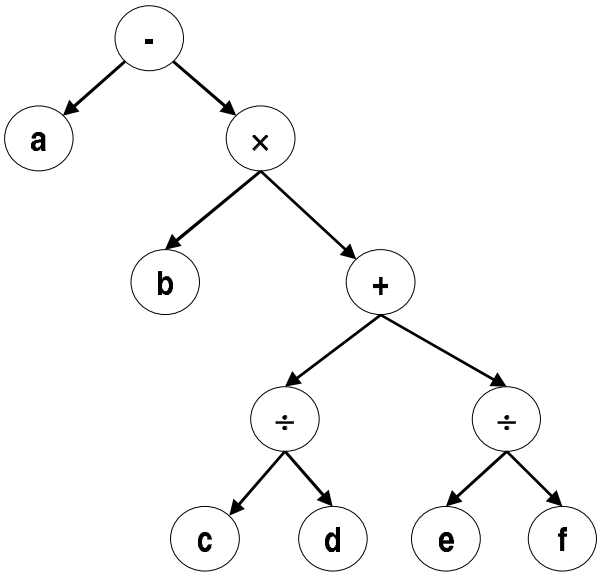
Nós chamamos de **filhos** de um nodo as raízes de suas sub-árvores. Formalmente,

> Seja $\Re$ um nodo raiz de uma sub-árvore qualquer, $\aleph$ o conjunto de todos os nodos da árvore, sejam $\rho_1..\rho_n \in \aleph$ suas sub-árvores, tal que existe uma aresta interconectando $\Re$ a cada ρ_i ($\Re \rightarrow \rho_i$), diz-se que cada ρ_i é *filho* ou descendente de $\Re$, e $\Re$ é dito ser o nodo *pai* ou *ancestral* de ρ_i.

Árvore Binária

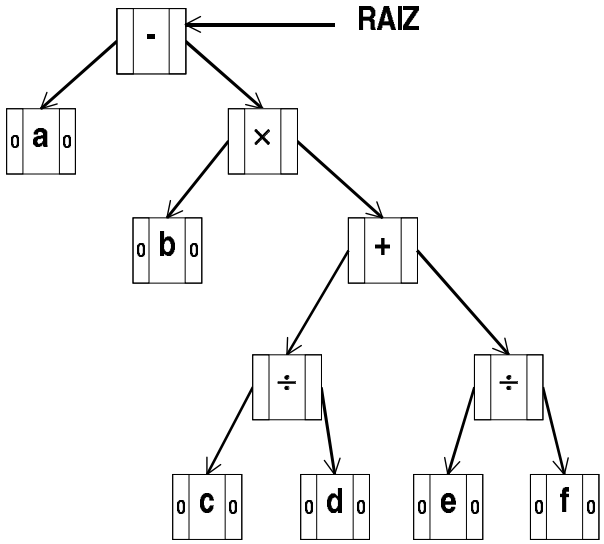
Em uma árvore binária, cada nodo pai tem até no máximo dois filhos (ou grau 2), e os filhos são distintos: *filho ou ramo* ***esquerdo*** e *filho ou ramo* ***direito***. Portanto árvores binárias são ordenadas.

Exemplo de uma fórmula representada usando uma árvore: a - b(c/d + e/f)



Representação de uma árvore binária usando ponteiros

Na prática, a implementação de cada nodo da árvore é feita usando um registro (RECORD) para cada nodo. Cada nodo possui um ou mais campos de informação, mais dois campos tipo ponteiro que apontam para os filhos do nodo. A mesma idéia pode ser usada para árvores que não são binárias, onde ao invés de termos dois ponteiros, têm-se uma lista de filhos (na forma de array ou lista encadeada), cujo tamanho variará de acordo com o número (máximo) de nodos filhos daquele nodo. A figura abaixo ilustra a implementação da árvore binária mostrada acima.



Caminhamento em árvore binária

Há três tipos de caminhamento em árvores binárias: pré ordem, em ordem, pós ordem.

Pré ordem (VED):

- **V**isita a raiz
- Caminha árvore da **E**squerda
- Caminha árvore da **D**ireita

Em ordem (EVD):

- Caminha a árvore da **E**squerda
- **V**isita a raiz
- Caminha a árvore da **D**ireita

Pós ordem (EDV):

- Caminha a árvore da **E**squerda
- Caminha a árvore da **D**ireita
- **V**isita a raiz

Exemplo de algoritmo e implementação para *Caminhamento Em Ordem*:

```
PROCEDURE CaminhaEmOrdem(p : tipo_p);
   PROCEDURE visita(p : tipo_p);
   BEGIN
      { executa ação sobre elemento apontado por p }
   END;
BEGIN
    IF p < NIL THEN BEGIN
       p := p^.esq;
      CaminhaEmOrdem(p);
       visita(p);
       p := p^.dir;
      CaminhaEmOrdem(p);
    END;
END.
```

O algoritmo acima é acionado no corpo principal do programa passando-se como argumento o ponteiro que aponta para a raiz. Os outros dois algoritmos e suas implementações são similares ao caso acima, mudando-se apenas a ordem em que visita(p) é chamada.

Exercício 7-1

Escreva um programa que abra e leia um **arquivo texto contendo a descrição de uma árvore binária** onde as linhas têm o seguinte formato:

nodo-pai <espaço> *nodo-filho*

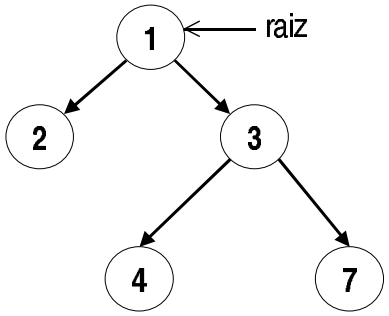
e monte a árvore.

O seu programa **deve se assegurar** que o nodo **sendo incluído não existe** e que o nodo informado como **pai existe**. Há três condições a serem testadas (em qualquer caso o programa apresenta mensagem de erro e continua montando a árvore):

1. se um nodo a ser incluído já aparece na árvore, não é possível incluí-lo uma segunda vez.
2. se o nodo pai informado no arquivo não existe na árvore;
3. se o nodo pai, encontrado na árvore, já tem dois filhos, não é possível incluir um terceiro filho (porque a árvore é binária);

Por exemplo, se o seu programa tomar como entrada o arquivo abaixo:

1 2
1 3
3 4
3 7
3 6
6 9



ele realizará as seguintes ações:

(a) cria nodo **raiz** 1
(b) cria nodo 2 como filho de 1
(c) cria nodo 3 como filho de 1
(d) cria nodo 4 como filho de 3
(e) cria nodo 7 como filho de 3
(f) não consegue criar nodo 6 porque nodo 3 já tem dois filhos
(g) não consegue criar nodo 9 porque nodo pai 6 não existe

Exercício 7-2

Usando a rotina de montagem de árvore acima, implemente um programa que imprima uma seqüência de identificadores de nodos a partir do tipo de caminhamento realizado. O programa deve solicitar o nome do arquivo contendo a árvore a ser criada, e apresentar um menu de opções para apresentar os três tipos de caminhamento:

- pós ordem
- pré ordem
- em ordem

Por exemplo, pós-ordem, para o arquivo acima: 2 4 7 3 1