1. Introdução

Devemos inicialmente definir:

a) Árvore:
   - É uma lista na qual cada elemento possui dois ou mais sucessores, porém, todos os elementos possuem apenas um antecessor. Forbellone & Eberspacher (2000, p167)
   - Uma estrutura de árvore com tipo básico T pode ser definida recursivamente conforme uma das duas situações abaixo: (1)A estrutura vazia; (2) Um nó do tipo T, associado a um número finito de estruturas disjuntas de árvores de mesmo tipo base T, denominada subárvores. Wirth (1986, p165)
   - Uma árvore enraizada T, ou simplesmente árvore, é um conjunto finito de elementos denominados nós ou vértices tais que: (1) T = 0, e a árvore é dita vazia, ou (2) existe um nó especial, r, chamado raiz de T; os restantes constituem um único conjunto vazio ou são divididos em m >=1 conjuntos disjuntos não vazios, as subárvores de r, ou simplesmente subárvores, cada qual por sua vez uma árvore. Szwarcfiter & Markenzon (1994, p.62)

b) Raiz: primeiro elemento, que dá origem aos demais
c) Nó: qualquer elemento da árvore
d) altura de uma árvore: quantidade de níveis a partir do nó-raiz até o nó mais distante (a raiz está no nível zero)
e) grau de uma árvore: número máximo de ramificações da árvore
f) grau de um nó: número máximo de ramificações a partir desse nó
g) filho: sucessor de um determinado nó
h) pai: único antecessor de um dado elemento
i) folha: elemento final (sem descendentes, sem filhos)
j) nível de um nó: é distância que um nó tem em relação ao nó raiz (a raiz está no nível 0)

Exemplo: Seja a árvore abaixo

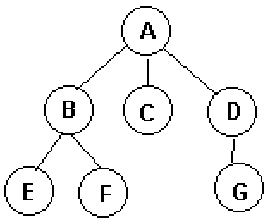

Figura 3.1. Uma exemplo de árvore

raiz: A
nós: A, B, C, D, E, F, G.
altura da árvore: 2
grau da árvore: 3
folhas: E, F e G

grau do nó-B: 2	grau do nó-C: 0	grau do nó-D: 1
filhos do nó-B: E e F	filhos do nó-C: 0	filhos do nó-D: G
pai do nó-B: A	pai do nó-C: A	pai do nó-D: A
nível do nó-B: 1	nível do nó-C: 1	nível do nó-D: 1

2. Representações:

Pode-se representar uma árvore de várias formas. Seja a Figura 3.1 dada acima.

a) Grafo
Repare que a Figura 3.1 é uma representação usando grafo.

b) Conjuntos aninhados

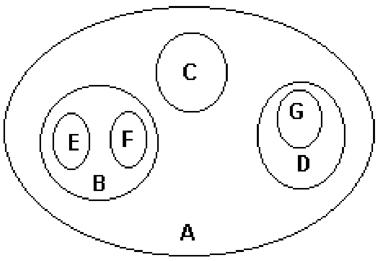

Figura 3.2. Representação por conjuntos aninhados

c) Parênteses aninhados
(A (B (E,F), C, D (G)))

d) Denteação
A
    B
        E
        F
    C
    D
        G

Podemos implementar uma árvore usando estruturas estáticas ou dinâmicas.

2.1. Representação Estática

A representação estática pode ser feita de várias formas. Vamos ver as sugestões de alguns autores:

a) Primeira representação:

Sabendo-se que a árvore da figura 1 tem grau 3 e altura 2, teremos:

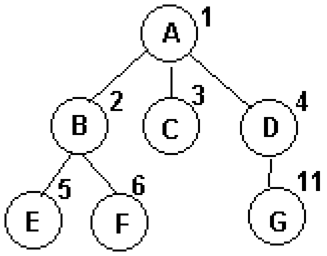

Figura 3.3. A árvore com seus nós enumerados

E poderemos construir o vetor:

```
typedef tipo_no Arvore [Max];
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
A	B	C	D	E	F					G		

b) Segunda representação:

Olhando a figura 1 podemos construir:

```
struct elemento{
        tipo_no valor;
        int nro_filhos;}
typedef struct elemento Arvore [Max]
```

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

c) Terceira representação:

Olhando a figura 1 podemos construir:

```
typedef int vetor_filhos[3];
struct elemento{
        tipo_no valor;
        vetor_filhos filhos;}
typedef struct elemento Arvore [Max]
```

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

5				6			
F	0	0	0	G	0	0	0

OBS:

Um dos problemas a ser resolvido em árvores é a questão de busca por um determinado dado. Em Forbellone & Eberspacher (2000, p.170) são mostrados dois tipos de busca usando a terceira representação acima.

As buscas apresentadas são: a busca em profundidade e a busca em Largura.

A busca em profundidade percorre todos os nós de um ramo até atingir os nós terminais (folhas) repetindo o processo em todos os ramos.

A busca em largura (ou em amplitude) visita todos os filhos de mesmo nível dos diversos ramos antes de passar para o próximo nível.

Dessa forma, podemos implementar uma busca em profundidade usando pilhas e a busca em largura usando filas.

Uma outra forma é fazer a busca recursiva.

2.2. Representação dinâmica:

A representação dinâmica usa os ponteiros para a implementação. Poderíamos ter a seguinte configuração para uma árvore de grau 3:

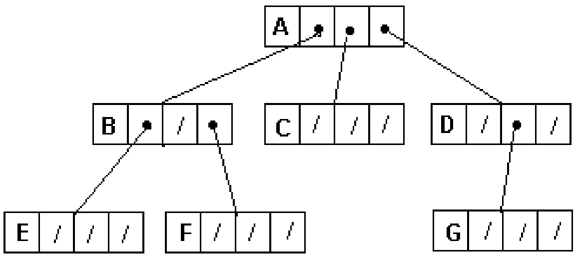

Figura 3.4. Uma representação de implementação de uma árvore

```
typedef struct no{
        tipo_no info;
        struct no* primeiro;
        struct no* segundo;
        struct no* terceiro;
        } *def_arvore;
```

3. Árvores binárias

Definição segundo Tenenbaum et al.(1994, p.303):

> "Uma árvore binária é um conjunto finito de elementos que está vazio ou é particionado em três subconjuntos disjuntos. O primeiro subconjunto contém um único elemento, chamado raiz da árvore. Os outros dois subconjuntos são em si mesmos árvores binárias, chamadas subárvores esquerda e direita da árvore original. Uma subárvore esquerda ou direita pode estar vazia. Cada elemento de uma árvore binária é chamado nó da árvore."

Assim, uma árvore binária é aquela que possui no máximo grau 2 e onde temos a idéia de filhos a esquerda e filhos a direita de um nó. Veja um exemplo de árvore binária na figura 3.5 abaixo.

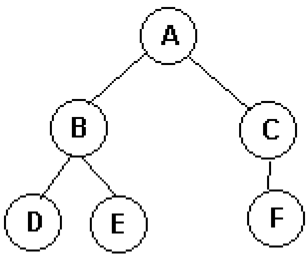

Figura 3.5. Árvore Binária

Outras definições:

- árvore estritamente binária: se todo nó que não for folha tiver subárvores esquerda e direita não vazias. Veja figura 3.6.

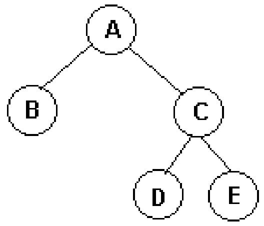

Figura 3.6. Uma árvore estritamente binária

- árvore binária completa de profundidade d é a árvore estritamente binária em que todas as folhas estejam no nível d. Veja figura 3.7.

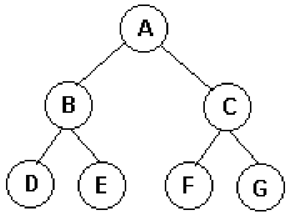

Figura 3.7. Uma árvore binária completa de profundidade 2

- árvore binária quase completa de profundidade d é uma árvore binária de profundidade d se (1) cada folha na árvore estiver no nível d ou no nível d-1. (2) para cada nó nd na árvore com descendente direito no nível d, todos os descendentes esquerdos de nd que forem folhas estiverem também no nível d. Veja figura 3.8.

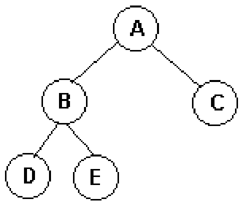

Figura 3.8. Uma árvore binária quase completa de profundidade 2

3.1 Inserção em Árvore Binária

Na inserção de elementos numa árvore binária temos de saber se o filho de um nó é a esquerda ou a direita da raiz e fazer a inserção.

3.2. Percurso em Árvore Binária

Uma operação comum em árvores binárias é percorrer cada um de seus nós uma vez. Pode-se simplesmente querer imprimir o conteúdo de cada nó ao passar por ele, ou podemos processá-lo de alguma maneira.

Para percorrer a árvore binária, temos três possibilidades:

- Pré-ordem
- Em-ordem ou ordem simétrica
- Pós-ordem

Seja uma árvore binária com três nós: raiz (R), nó a esquerda (E) e nó a direita (D). Então nos percursos citados acima teríamos:

- pré-ordem : R, E, D (visita a raiz, a subárvore esquerda e por fim a subárvore direita)
- em-ordem: E, R, D (visita a subárvore esquerda, a raiz e por fim a direita)
- pós-ordem: E, D, R (visita a subárvore esquerda depois a direita e por fim a raiz)

Olhando a árvore da figura 3.9 abaixo, teremos:

- pré-ordem: 14, 4, 3, 9, 7, 5, 15, 18, 16, 17, 20
- em-ordem: 3, 4, 5, 7, 9, 14, 15, 16, 17, 18, 20
- pós-ordem: 3, 5, 7, 9, 4, 17, 16, 20, 18, 15, 14

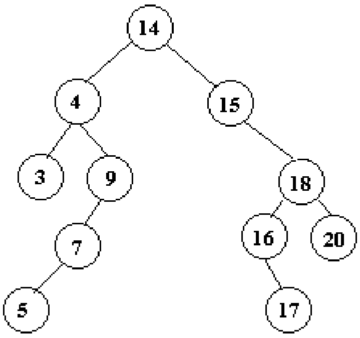

Figura 3.9. Uma árvore binária

3.2.1. Algoritmos recursivos

Pré-ordem:

Mostra_raiz(arvore);
Mostra_lado_esquerdo (arvore);
Mostra_lado_direito (arvore);

Em-ordem:

Mostra_lado_esquerdo (arvore);
Mostra_raiz(arvore);
Mostra_lado_direito (arvore);

Pós-ordem:

Mostra_lado_esquerdo (arvore);
Mostra_lado_direito (arvore);
Mostra_raiz(arvore);

3.2.2. Algoritmos não recursivos

Para resolver o percurso sem usar recursão teremos de fazer uso de estruturas de pilhas e filas para poder percorrer a árvore visitando o nó uma única vez..

3.3. Busca em Árvore Binária

Como a árvore não possui nenhum tipo de ordenação, não há como otimizar o processo. Então temos de percorrer todos os elementos para encontrar o que procuramos.