UNISINOS - **UNIVERSIDADE DO VALE DO RIO DOS SINOS**
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS (C6/6)
PIP/CA - Programa Interdisciplinar de Pós-Graduação em Computação Aplicada

ALGORITMOS
&
ESTRUTURAS DE DADOS

Professor responsável: *Fernando Santos Osório* **E-mail:** ***osorio@exatas.unisinos.br***
Semestre: 2000/1 **Web:** ***http://www.inf.unisinos.br/~osorio/***
Horário: Noite - de 21/02 à 25/02/2000

Estruturas de Dados com Alocação Dinâmica

1. Alocação de Dinâmica de Memória:

Podemos trabalhar com listas encadeadas de diferentes tipos, resultando nas estruturas de dados descritas a seguir.

- Conjuntos de estruturas de dados com alocação dinâmica a serem estudados:

A) *Listas encadeadas simples* ✓
B) *Listas duplamente encadeadas* ✓
C) *Filas* ✓
D) *Pilhas* ✓
E) *Árvores* ⇦

---

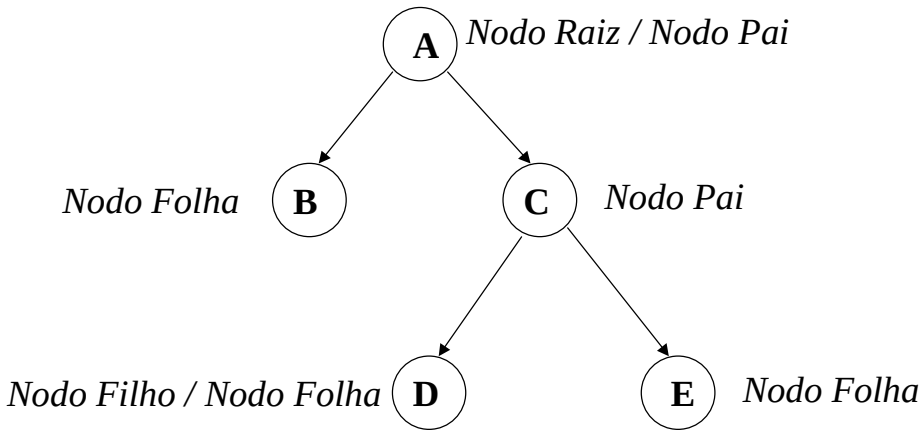
2. Árvores Binárias:

As árvores são estruturas de dados criadas usualmente através do uso de alocação dinâmica de memória, baseadas em listas encadeadas, que possuem um nodo superior (raiz / pai), apontando para os seus nodos filhos (folhas / filho). Por sua vez cada nodo pai pode possuir nodos filhos, e assim chegamos a definição “recursiva” de uma árvore: um nodo pai aponta para nodos filhos, onde estes nodos filhos também podem ser nodos pais.

Uma árvore binária é um tipo específico de árvore que possui apenas 2 nodos filhos ligados a cada nodo pai: o nodo da esquerda e o nodo da direita. As árvores binárias permitem uma busca mais eficiente quando os dados são inseridos de forma ordenada, podendo ser percorridas de três maneiras:

- Percorrer de modo prefixado: Pré-Ordem (VED = Visita/Esquerda/Direita);
- Percorrer de modo infixado : Em Ordem (EVD = Esquerda/Visita/Direita);
- Percorrer de modo pósfixado: Pós-Ordem (EDV = Esquerda/Direita/Visita);

A figura abaixo mostra um exemplo de uma árvore binária:



Exemplo de criação de uma árvore binária simples (apenas 3 nodos):

```
typedef    int    Tipo_Dado;

typedef    struct  bloco_ab {
                              Tipo_Dado      Dado;
                              struct  bloco_ab     *FilhoEsq, *FilhoDir;
                              struct  bloco_ab     *Pai;   /* opcional */
                            }   Nodo_AB;

main ( )
{
    Nodo_AB  *Arvore_Binaria;
    Nodo_AB  *Novo_Nodo;

    /*  Criando uma árvore binária com 3 nodos apenas...  */

    /* Alocação dinâmica: cria o nodo raiz (pai)     */
    Novo_Nodo = (Nodo_AB *) calloc ( 1, sizeof (Nodo_AB) );
    Novo_Nodo -> Dado = 10;
    Novo_Nodo -> Pai = NULL;     /* Ainda não possui nodos pai e filhos */
    Novo_Nodo -> FilhoEsq = NULL;
    Novo_Nodo -> FilhoDir = NULL;

   /* Arvore_Binaria aponta para a Raiz da árvore */
    Arvore_Binaria = Novo_Nodo;

    /* Alocação dinâmica: cria o nodo que será o filho da esquerda     */
    Novo_Nodo = (Nodo_AB *) calloc ( 1, sizeof (Nodo_AB) );
    Novo_Nodo -> Dado = 5;
    Novo_Nodo -> Pai = Arvore_Binaria;    /*  O pai do novo nodo é o nodo raiz */
    Novo_Nodo -> FilhoEsq = NULL;   /* Não tem filho a esquerda  */
    Novo_Nodo -> FilhoDir = NULL;   /* Não tem  filho a direita      */
```

```
        /* Ajusta ponteiros: o filho da esquerda da raiz é o novo nodo        */
         Arvore_Binaria -> FilhoEsq = Novo_Nodo;

         /* Alocação dinâmica: cria o nodo que será o filho da direita     */
         Novo_Nodo = (Nodo_AB *) calloc ( 1, sizeof (Nodo_AB) );
         Novo_Nodo -> Dado = 15;
         Novo_Nodo -> Pai = Arvore_Binaria;    /*  O pai do novo nodo é o nodo raiz */
         Novo_Nodo -> FilhoEsq = NULL;   /* Não tem filho a esquerda  */
         Novo_Nodo -> FilhoDir = NULL;   /* Não tem  filho a direita      */

        /* Ajusta ponteiros: o filho da direita da raiz é o novo nodo              */
         Arvore_Binaria -> FilhoDir = Novo_Nodo;
    }
```

- As árvores binárias ordenadas permitem que se faça a chamada pesquisa binária, pois os nodos estão ordenados, onde sempre a esquerda de um nodo se encontram valores menores que o valor deste nodo, e a sua direita sempre temos valores maiores que o valor deste nodo. Note também que a árvore acima foi criada usando uma lista duplamente encadeada (pai aponta para os filhos – esq./dir., e filho aponta para o pai). As listas duplamente encadeadas permitem que se percorra a lista/árvore nos dois sentidos, o que pode ser bastante interessante de acordo com o problema a ser tratado... entretanto, este tipo de encadeamento duplo pode não ser necessário em muitas situações. Nas rotinas que serão implementadas a seguir, podemos percorrer toda a árvore de forma recursiva, sem precisar usar o ponteiro de "nodo pai".

Rotinas básicas de manipulação de árvores binárias:

- Estruturas de dados com alocação dinâmica
- Inserção: nodo pai, nodo filho ou inserção em uma ***árvore binária ordenada***
  Árvore binária ordenada: existe uma relação explícita entre os nodos
  Exemplo: *nodo da esquerda < pai < nodo da direita*
- Remoção: qualquer elemento da árvore (ajusta estrutura para manter coerente)
- Listagem: exibir em pré-ordem, em ordem infixada, e em pós-ordem
- Balanceamento: equilibrar os ramos da árvore em relação a profundidade dos mesmos

Aplicação típica:
- Árvore binária de busca (arvore binária ordenada e balanceada), árvores de decisão, ...

- Estrutura de dados:

```
typedef    int    Tipo_Dado;

typedef    struct  bloco_ab {
                                Tipo_Dado          Dado;
                                struct  bloco_ab   *FilhoEsq, *FilhoDir;
                                struct   bloco_ab   *Pai;
                           }   Nodo_AB;
```

- Rotinas:

```
void inicializa_arvbin          (Nodo_AB **AB);
int insere_ord_arvbin           (Nodo_AB **AB, Tipo_Dado Dado);
int  remove_dado_arvbin         (Nodo_AB **AB, Tipo_Dado Dado);
void exibe_ab_infixado          (Nodo_AB  *AB);
void exibe_ab_prefixado         (Nodo_AB  *AB);
void exibe_ab_posfixado         (Nodo_AB  *AB);
void apaga_arvbin               (Nodo_AB **AB);
int conta_nodos_arvbin          (Nodo_AB  *AB);
Nodo_AB *pesquisa_arvbin        (Nodo_AB  *AB, Tipo_Dado Dado);
Nodo_AB *maior_arvbin           (Nodo_AB  *AB, Tipo_Dado *Dado);
Nodo_AB *menor_arvbin           (Nodo_AB  *AB, Tipo_Dado *Dado);
Nodo_AB *sucessor_arvbin        (Nodo_AB  *AB, Tipo_Dado *Dado);
Nodo_AB *predecessor_arvbin(Nodo_AB  *AB, Tipo_Dado *Dado);
```

- Exemplo: inserção de dados na árvore binária de forma ordenada

```
int insere_ord_arvbin (AB, Dado)
Nodo_AB **AB;
Tipo_Dado Dado;
{
        /* Árvore binária onde os nodos são inseridos de maneira ordenada:      */
        /* - Os nodos a esquerda de um nodo pai são sempre menores que ele  */
        /* - Os nodos a direita de um nodo pai são sempre maiores que ele       */

        Nodo_AB *novo,  *aux, *temp;

        novo = (Nodo_AB *) calloc ( 1, sizeof (Nodo_AB) );
        if ( novo == NULL ) return (ERRO);    /* Erro na alocação */

        novo -> Dado = Dado;
        novo -> FilhoEsq = NULL;
        novo -> FilhoDir = NULL;
        aux = *AB;

        while ( aux != NULL )     /* Acha o ponto onde vai inserir */
        {
                temp = aux;
                if ( Dado > (aux -> Dado) )
                    aux = aux -> FilhoDir;
                else
                    aux = aux -> FilhoEsq;
        }

        if ( aux == *AB)
        {
            novo -> Pai = NULL;
            *AB = novo;
        }
```

```
        else
        {
            novo -> Pai = temp;
            if  ( Dado > temp->Dado)
                temp -> FilhoDir = novo;
            else
                temp -> FilhoEsq = novo;
        }
        return (OK);
}
```

- Exemplo: listar a árvore binária em pré-ordem

```
void exibe_ab_infixado (AB)
Nodo_AB *AB;
{
        if  ( AB != NULL )
        {
            exibe_ab_infixado ( AB -> FilhoEsq );
            printf ("%d\n", AB -> Dado);
            exibe_ab_infixado ( AB -> FilhoDir);
        }
}
```

- Exemplo: listar a árvore binária em modo infixado

```
void exibe_ab_prefixado (AB)
Nodo_AB *AB;
{
        if  ( AB != NULL )
        {
            printf ("%d\n", AB -> Dado);
            exibe_ab_prefixado ( AB -> FilhoEsq );
            exibe_ab_prefixado ( AB -> FilhoDir);
        }
}
```

- Exemplo: listar a árvore binária em pós-ordem

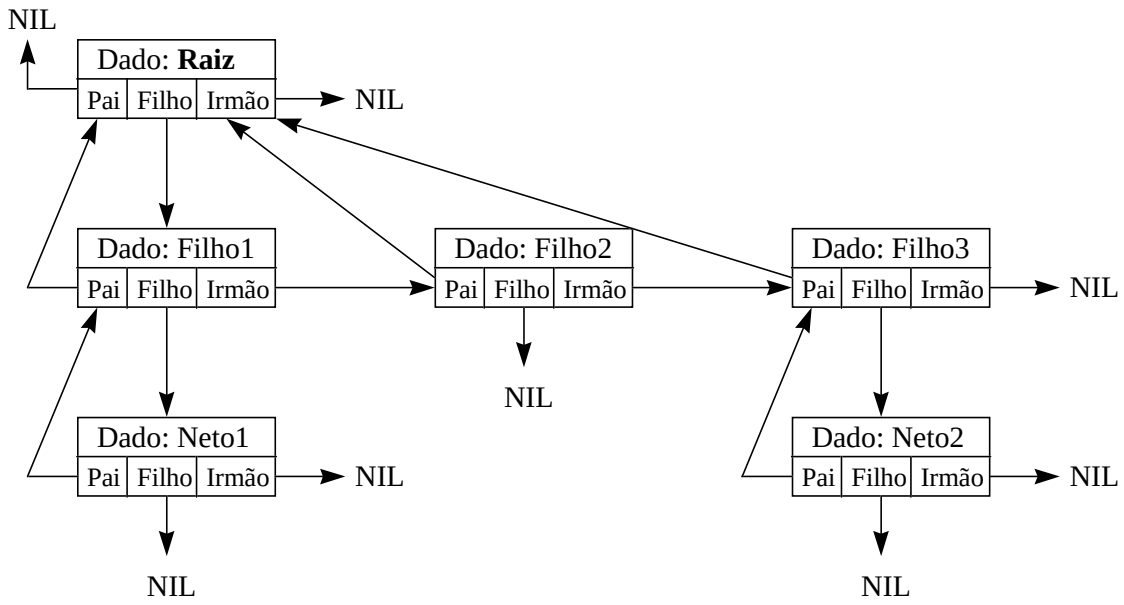
```
void exibe_ab_posfixado (AB)
Nodo_AB *AB;
{
        if  ( AB != NULL )
        {
            exibe_ab_posfixado ( AB -> FilhoEsq );
            exibe_ab_posfixado ( AB -> FilhoDir);
            printf ("%d\n", AB -> Dado);
        }
}
```

3. Árvores Genéricas:

As árvores são estruturas de dados criadas usualmente através do uso de alocação dinâmica de memória, baseadas em listas encadeadas, que possuem um nodo superior (raiz / pai), apontando para os seus nodos filhos (folhas / filho). Por sua vez cada nodo pai pode possuir nodos filhos, e assim chegamos a definição "recursiva" de uma árvore: um nodo pai aponta para nodos filhos, onde estes nodos filhos também podem ser nodos pais.

Uma árvore genérica é um tipo especial de árvore onde podemos ter um número variável de nodos filhos associados a um nodo pai. Um exemplo de implementação de uma árvore genérica pode ser dado por uma estrutura onde cada nodo possui um ponteiro para o seu nodo p ai, um ponteiro para o seu nodo filho, e um ponteiro para o seu nodo irmão. Assim sendo, cada nodo pode possuir um número ilimitado de filhos, pois o seu nodo filho aponta para uma lista ilimitada de nodos no mesmo nível na hierarquia da árvores (irmãos). Além disto, continuamos tendo uma estrutura em árvore, pois cada nodo possui seu nodo pai e seu(s) nodo filho(s).

A figura abaixo mostra um exemplo de uma árvore genérica:



- Estrutura de dados:

```
typedef    int    Tipo_Dado;

typedef    struct  bloco_ag {
                                Tipo_Dado          Dado;
                                struct  bloco_ag   *Pai, *Filho, *Irmao;
                             }  Nodo;
```

- Rotinas:

```
void inicializa_arvgen      (Nodo_AG **AG);
int insere_filho_arvgen     (Nodo_AG **AG, Tipo_Dado Dado);
int insere_irmao_arvgen     (Nodo_AG **AG, Tipo_Dado Dado);
int insere_pai_arvgen       (Nodo_AG **AG, Tipo_Dado Dado);
void exibe_arvgen           (Nodo_AG  *AG);
int salva_disco_arvgen      (Nodo_AG  *AG, char *NomeArq);
int le_disco_arvgen         (Nodo_AG **AG, char *NomeArq);
void apaga_arvgen           (Nodo_AG **AG);
int remove_dado_arvgen      (Nodo_AG **AG, Tipo_Dado Dado);
Nodo_AG *pesquisa_arvgen    (Nodo_AG  *AG, Tipo_Dado Dado);
```

Atenção: Endereços de memória não servem para nada se estes forem salvos em disco... um ponteiro para uma posição de memória ***NÃO*** deve ser salvo em disco como sendo um endereço. Não podemos garantir que, quando um ponteiro(endereço) for lido do disco, a variável apontada continue exatamente na mesma posição de memória!

⇨ Outros tópicos importantes:

- ⇨ Balanceamento de árvores ordenadas – Árvores AVL e Red-Black
- ⇨ Pesquisa binária X Pesquisa Seqüencial (Análise de complexidade / Performance)
- ⇨ Ordenação: Insertion Sort, Bubble Sort, Shaker Sort, Quick Sort, Merge Sort, ...
- ⇨ Grafos, Redes de Petri, Cadeias de Markov, ...