

Sistema Aberto de Educação



Guia de Estudo

Estrutura de Dados

2ª Parte



Instituição Credenciada pelo MEC
Centro Universitário do Sul de Minas



SABE – Sistema Aberto de Educação

**Av. Cel. José Alves, 256 - Vila Pinto
Varginha - MG - 37010-540
Tele: (35) 3219-5204 - Fax - (35) 3219-5223**

Instituição Credenciada pelo MEC – Portaria 4.385/05

**Centro Universitário do Sul de Minas - UNIS/MG
Unidade de Gestão da Educação a Distância – GEaD**

Mantida pela
Fundação de Ensino e Pesquisa do Sul de Minas - FEPESMIG

Varginha/MG

SILVA, Lázaro Eduardo.

Guia de Estudo – Estrutura de Dados.
Lázaro Eduardo da Silva. Varginha: GEaD-
UNIS/MG, 2008.
22p.

1. Recursividade. 2. Árvore. 3. Classificação













REITOR**Prof. Ms. Stefano Barra Gazzola****GESTOR****Prof. Ms. Wanderson Gomes de Souza****Supervisora Técnica****Prof^a. Ms. Simone de Paula Teodoro Moreira****Coord. do Núcleo de Recursos Tecnológicos****Lúcio Henrique de Oliveira****Coord. do Núcleo de Desenvolvimento Pedagógico****Prof^a. Vera Lúcia Oliveira Pereira**

Revisão ortográfica / gramatical

Prof^a. Maria José Dias Lopes Grandchamp**Design/diagração e Equipe de Tecnologia Educacional****Prof. Celso Augusto dos Santos Gomes****Prof^a. Débora Cristina Francisco Barbosa****Danúbia Pinheiro Teixeira****Jacqueline Aparecida Silva****Autor****LÁZARO EDUARDO DA SILVA**

Bacharel em Ciência da Computação (2004), Especialista em Redes de Computadores (2005), Especialista em Docência na Educação a Distância (2008) – UNIS, Varginha - MG. Mestrando em Engenharia Elétrica – USP, São Carlos – SP. Professor desde 2005 foi também suplente de coordenação da pós-graduação em Redes de Computadores. Atualmente é professor no curso Ciência da Computação, Sistemas de Informação, pós-graduação em Redes de Computadores e pós-graduação em Banco de Dados - UNIS - MG.

TABELA DE ÍCONES

	REALIZE. Determina a existência de atividade a ser realizada. Este ícone indica que há um exercício, uma tarefa ou uma prática para ser realizada. Fique atento a ele.
	PESQUISE. Indica a exigência de pesquisa a ser realizada na busca por mais informação.
	PENSE. Indica que você deve refletir sobre o assunto abordado para responder a um questionamento.
	CONCLUSÃO. Todas as conclusões, sejam de idéias, partes ou unidades do curso virão precedidas desse ícone.
	IMPORTANTE. Aponta uma observação significativa. Pode ser encarado como um sinal de alerta que o orienta para prestar atenção à informação indicada.
	HIPERLINK. Indica um link (ligação), seja ele para outra página do módulo impresso ou endereço de Internet.
	EXEMPLO. Esse ícone será usado sempre que houver necessidade de exemplificar um caso, uma situação ou conceito que está sendo descrito ou estudado.
	SUGESTÃO DE LEITURA. Indica textos de referência utilizados no curso e também faz sugestões para leitura complementar.
	APLICAÇÃO PROFISSIONAL. Indica uma aplicação prática de uso profissional ligada ao que está sendo estudado.
	CHECKLIST ou PROCEDIMENTO. Indica um conjunto de ações para fins de verificação de uma rotina ou um procedimento (passo a passo) para a realização de uma tarefa.
	SAIBA MAIS. Apresenta informações adicionais sobre o tema abordado de forma a possibilitar a obtenção de novas informações ao que já foi referenciado.
	REVENDO. Indica a necessidade de rever conceitos estudados anteriormente.

Sumário

1. RECURSIVIDADE.....	8
1.1. O PROBLEMA DAS TORRES DE HANOI	9
2. ÁRVORE.....	10
2.1. ÁRVORE BINÁRIA	11
2.1.1. Caminhamento em Árvore Binária	12
2.1.2. Construção de Árvore Binária:.....	12
2.1.3. Inserção, Remoção e Consulta ordenada em Árvore Binária:.....	12
3. MÉTODOS DE CLASSIFICAÇÃO	18
3.1. CLASSIFICAÇÃO POR TROCA	18
3.2. CLASSIFICAÇÃO POR SELEÇÃO	19
3.3. CLASSIFICAÇÃO POR INSERÇÃO	20
4. BIBLIOGRAFIA	22

Lista de Figuras

<i>Figura 1 – Recursividade: programa fatorial para o valor 4.....</i>	<i>8</i>
<i>Figura 2 - Exemplo das Torres de Hanói.....</i>	<i>9</i>
<i>Figura 3 - Representação esquemática da estrutura árvore.....</i>	<i>10</i>
<i>Figura 4 - Árvore binária</i>	<i>11</i>
<i>Figura 5 - Construção da árvore binária</i>	<i>12</i>
<i>Figura 6 - Inserção em árvore binária.....</i>	<i>13</i>
<i>Figura 7 - Remoção de nó com subárvore vazia.....</i>	<i>13</i>
<i>Figura 8 - Remoção de nó sem subárvores vazias.....</i>	<i>13</i>
<i>Figura 9 - Estrutura de implementação da árvore binária.....</i>	<i>14</i>

Lista de Programas

<i>Programa 1 - Recursão</i>	<i>8</i>
<i>Programa 2 - Torres de Hanói</i>	<i>9</i>
<i>Programa 3 - Declaração do tipo para implementação da estrutura árvore</i>	<i>14</i>
<i>Programa 4 - Procedimento que constrói uma árvore</i>	<i>14</i>
<i>Programa 5 - Procedimento que destrói uma árvore.....</i>	<i>15</i>
<i>Programa 6 - Procedimento de inserção ordenada na árvore binária</i>	<i>15</i>
<i>Programa 7 - Rotina de encontrar o menor dos maiores em uma subárvore.....</i>	<i>16</i>
<i>Programa 8 - Procedimento que retira um nó da árvore.....</i>	<i>17</i>
<i>Programa 9 - Rotinas de percurso em árvore binária.....</i>	<i>17</i>
<i>Programa 10 - Estrutura base para implementação dos algoritmos de ordenação</i>	<i>18</i>
<i>Programa 11 - Rotina de classificação por troca</i>	<i>19</i>
<i>Programa 12 - Rotina de classificação por seleção.....</i>	<i>19</i>
<i>Programa 13 - Rotina de classificação por inserção</i>	<i>20</i>

Lista de Tabelas

<i>Tabela 1 - Acompanhamento da classificação por troca</i>	<i>19</i>
<i>Tabela 2 - Acompanhamento da classificação por seleção.....</i>	<i>20</i>
<i>Tabela 3 - Acompanhamento da classificação por inserção</i>	<i>20</i>

1. Recursividade

Segundo Tenenbaum, um procedimento ou função é chamado recursivo quando na sua implementação existe uma chamada do próprio procedimento ou função. Esta chamada recebe o nome de chamada recursiva.

Em termos gerais, ao enfrentar a tarefa de escrever um programa para resolver um problema, não se preocupe em procurar uma solução recursiva. A maioria dos problemas pode ser solucionada de maneira simples, usando métodos não-recursivos. Entretanto, alguns problemas podem ser resolvidos em termos lógicos e com mais elegância por meio da recursividade. A recursividade é utilizada para simplificar a lógica de programação. Quase sempre substitui uma repetição, e como tal deve ter uma condição que determine a interrupção da repetição.

Podemos dividir uma rotina recursiva em duas partes básicas:

- A condição de parada da recursividade
- chamada(s) recursiva(s)

Exemplo: Cálculo de Fatorial.

Esta operação matemática é definida de forma recursiva:

$$0! = 1$$

$$n! = (n-1)! * n \text{ p/ } n > 0$$

Então:

$$4! = 3! * 4 = 2! * 3 * 4 = 1! * 2 * 3 * 4 = 0! * 1 * 2 * 3 * 4 = 1 * 1 * 2 * 3 * 4 = 24$$

```
program exemplo_recursao;  
function fat (n : integer) : integer;  
begin  
  if (n <= 0) then fat := 1  
  else fat := fat(n-1) * n;  
end;  
begin  
  writeln ('Fatorial de 4 = ', fat(4));  
  writeln ('Fatorial de 6 = ', fat(6));  
  readln;  
end.
```

Programa 1 - Recursão

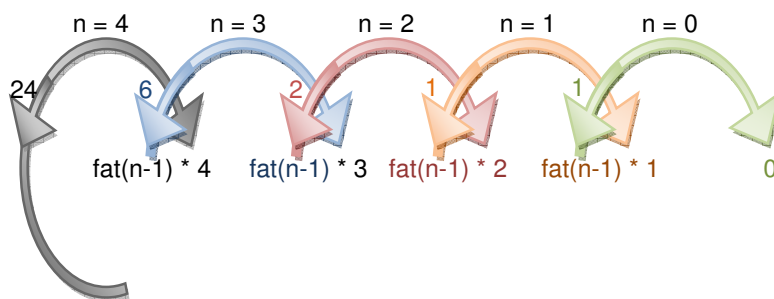


Figura 1 – Recursividade: programa fatorial para o valor 4

No exemplo acima, quando o programa faz a chamada da função fat enviando o valor 4, na função o valor irá realizar o else com uma chamada da mesma função enviando o valor 3. Quando esta chamada é realizada, o retorno da função fica pendente, visto que ela precisa do retorno da chamada com valor 3 para realizar a multiplicação por 4. Isso se repete, até que o valor enviado a função seja zero, enviando como retorno o valor 1. Quando este retorno acontece, as multiplicações

pendentes são realizadas, tendo como retorno total da chamada inicial o resultado da multiplicação $1 * 1 * 2 * 3 * 4$.

1.1. O problema das Torres de Hanoi

Um problema muito famoso que pode simplificar sua implementação utilizando a técnica de recursividade é as torres de Hanói.

“Há muito e muito tempo atrás, no alto das montanhas de Hanoi, havia um mosteiro onde habitavam sacerdotes brâmanes; entre eles, era praticado um ritual para prever o fim do mundo. Conta a lenda, que no mosteiro havia três torres, sendo que na primeira delas estavam empilhados 64 discos de ouro em tamanhos decrescentes. Os sacerdotes acreditavam que quando eles terminassem de transferir todos os discos da primeira torre para a terceira (usando a segunda), sem nunca colocar um disco maior sobre um menor, então, neste dia, o mundo acabaria!”

Para solucionar o problema das Torres de Hanoi, usando recursão, considere um caso geral em que n discos devem ser transferidos. Para mover n discos da torre A para a torre C, usando a torre B como auxiliar, fazemos:

- Se $n = 1$, transfira o disco da torre A para torre C e pare;
- Caso Contrário:
 1. Transfira $n-1$ discos da torre A para B, usando C como auxiliar;
 2. Transfira o último disco da torre A para a torre C;
 3. Transfira $n-1$ discos da torre B para torre C, usando A como auxiliar;

```
program recursao_hanoi;  
procedure hanoi (n:integer; origem, auxiliar, destino: char);  
begin  
  if (n = 1) then  
    writeln ('Mova disco 1 da torre ', origem, ' para ', destino)  
  else begin  
    hanoi (n-1, origem, destino, auxiliar);  
    writeln ('Mova disco ', n, ' da torre ', origem, ' para ', destino);  
    hanoi (n-1, auxiliar, origem, destino);  
  end;  
end;  
begin  
  hanoi (5, 'A', 'B', 'C');  
  readln;  
end.
```

Programa 2 - Torres de Hanói

Vamos verificar o funcionamento das torres de Hanoi para cinco discos de diferentes diâmetros na torre A, de modo que um disco maior fique sempre abaixo de um disco menor. O objetivo é deslocar os cinco discos para a torre C, usando a torre B como auxiliar. Somente o primeiro disco de toda a torre pode ser deslocado para outra torre, e um disco maior não pode nunca ficar posicionado sobre um disco menor (Tenenbaum, 2005).

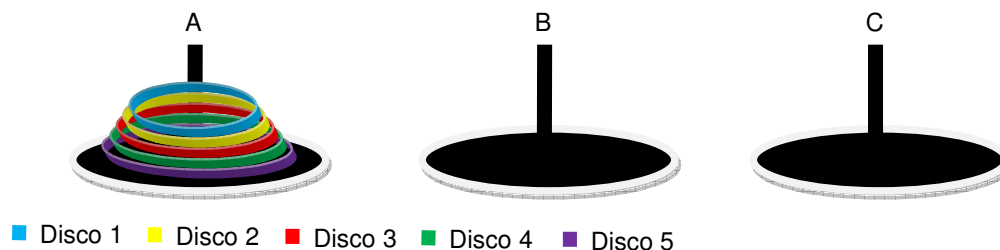


Figura 2 - Exemplo das Torres de Hanói

Para deslocarmos os dois primeiros discos da torre A para a torre B usando C como auxiliar, basta deslocarmos o disco 1 da torre A para a torre C, deslocarmos o disco 2 da torre B e

deslocarmos o disco 1 para a torre B em cima do disco 2. Poderíamos deslocar o disco 3 da torre A para a torre C e, aplicar novamente a solução para dois discos, movendo-os de B para C usando A como auxiliar.

Como parágrafo acima, sabemos como deslocar os três primeiros discos da torre A para a torre C usando B como auxiliar. Poderíamos deslocar o quarto disco da torre A para a torre B e, aplicar novamente a solução de três discos, movendo-os de C para B, usando a torre A como auxiliar.

Como parágrafo acima, sabemos como deslocar os quatro primeiros discos da torre A para a torre B usando C como auxiliar. Poderíamos deslocar o maior disco de A para C e, por último, aplicar novamente a solução aos quatro discos, movendo-os de B para C, usando a torre A como auxiliar.

Perceba a recursividade nos parágrafos acima. Cada parágrafo caracteriza as instruções da rotina conforme programa apresentado.



Acesse a implementação em flash (.swf) disponibilizada na Midiateca que simula o funcionamento das Torres de Hanói. Caso não consiga realizar a transferência dos discos, execute o programa acima no Pascal e siga as instruções de resposta do programa para verificar o funcionamento da implementação.

Em geral, uma versão não-recursiva de um programa executará com mais eficiência, em termos de tempo e espaço, do que uma versão recursiva. Isso acontece porque o trabalho extra dispendido para entrar e sair de um bloco é evitado na versão não-recursiva.

Contudo, verificamos que uma solução recursiva é o método mais natural e lógico de solucionar um problema.

Dessa forma, ocorre um conflito entre a eficiência da máquina e a do programador. Com o custo da programação aumentando consideravelmente e o custo da computação diminuindo, chegamos ao ponto em que, na maioria dos casos, não compensa para o programador construir exaustivamente uma solução não-recursiva (Tenenbaum, 2005).



Acesse o ambiente virtual de aprendizagem no item atividades e faça os exercícios propostos sobre recursividade. Utilize o fórum Tira-dúvidas para colocar suas dificuldades e socializar a implementação realizada. Apesar de não avaliativo, o fórum é muito importante para construirmos uma seqüência lógica de pensamento e amadurecermos na implementação de problemas computacionais.

2. Árvore

A árvore é uma estrutura de dados que caracteriza a hierarquia entre seus dados. Devido a isto, é a estrutura indicada para aplicações onde é necessário representar ordem e hierarquia. Uma árvore é uma estrutura de dados em que cada elemento tem um ou mais elementos associados, sendo que não existem árvores vazias, no mínimo haverá um nó raiz, cada árvore tem apenas uma raiz.

Representação esquemática:

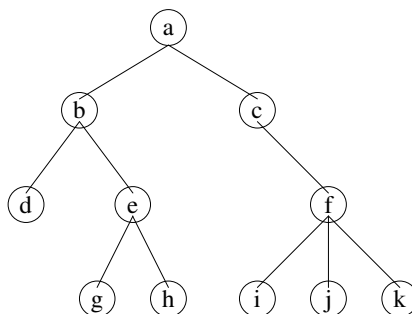


Figura 3 - Representação esquemática da estrutura árvore



Terminologia:

A **raiz** desta árvore é o nó **a**. Mas todo nó é raiz de uma **subárvore**. Então **b** é a raiz de uma subárvore que tem dois nós **d** e **e**, **d** é a raiz de uma subárvore vazia.

O número de subárvores de um nó é o **grau** daquele nó. Então **a** tem grau 2.

Um nó de grau zero é um nó **folha** ou nó terminal. Então **d** é uma folha desta árvore.

O **nível** do nó raiz é zero. Os demais níveis são definidos pelo número de ramos que o ligam com o nó raiz. O nó **e**, por exemplo, tem nível 2.

Floresta é um conjunto de duas ou mais árvores.

Quando a **ordem** das subárvores é relevante, dizemos que a árvore é **ordenada**. Caso contrário, dizemos que é uma árvore **orientada**.

2.1. Árvore Binária

São estruturas do tipo árvore, onde o grau de cada nó é menor ou igual a dois.

Exemplo:

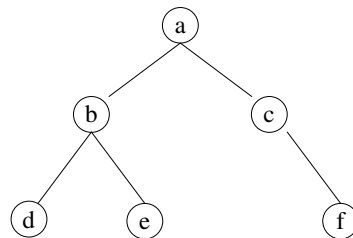


Figura 4 - Árvore binária

Existe numa árvore binária, distinção entre a subárvore direita e esquerda. Então para o exemplo anterior de árvore binária, a subárvore direita do nó **a** começa com o nó **c** e a subárvore esquerda começa com o nó **b**.

Embora as árvores naturais cresçam com suas raízes fincadas na terra e suas folhas no ar, em computação universalmente as estruturas de dados em árvore possuem a raiz no topo e as folhas no chão. O sentido da raiz para as folhas é “para baixo” e o sentido oposto é “para cima”. Quando você percorre uma árvore a partir das folhas na direção da raiz, diz-se que você está “subindo” a árvore, e se partir da raiz para as folhas, você está “descendo” a árvore (Tenenbaum, 2005).



Se todo nó que não é folha numa árvore binária tiver subárvores esquerda e direita não-vazias, a árvore será considerada **árvore estritamente binária**. Portanto a árvore binária da Figura 4 não é estritamente binária. Para ela se tornar estritamente binária, é necessário que a subárvore esquerda do nó **c** seja não vazia.

A **profundidade** de uma árvore binária significa o nível máximo de qualquer folha na árvore. Isso equivale ao tamanho do percurso mais distante da raiz até qualquer folha. Sendo assim, a profundidade da árvore da Figura 4 é 2.

Uma **árvore binária completa** é uma árvore estritamente binária em que todas as folhas estejam no nível da profundidade. Se for inserido um nó na subárvore esquerda da Figura 4 ela se tornará árvore binária completa.

2.1.1. Caminhamento em Árvore Binária

Caminhar em uma árvore binária significa percorrer todos os nós da árvore de forma sistemática de modo que cada nó seja visitado uma única vez. Existem 3 formas básicas de caminhamento em árvore binária:

1. Caminhamento pré-fixado (pré-ordem)

- Processar o nó
- Percorrer a subárvore esquerda
- Percorrer a subárvore direita

2. Caminhamento central (em-ordem)

- Percorrer a subárvore esquerda
- Processar o nó
- Percorrer a subárvore direita

3. Caminhamento pós-fixado (pós-ordem)

- Percorrer a subárvore esquerda
- Percorrer a subárvore direita
- Processar o nó

Exemplo:

Os nós da árvore binária anterior são processados da seguinte forma segundo cada caminhamento:

pré-ordem: a, b, d, e, c, f

em-ordem: d, b, e, a, c, f

pós-ordem: d, e, b, f, c, a

2.1.2. Construção de Árvore Binária:

Usaremos uma forma seqüencial dos nós para facilitar o processo de construção de uma árvore binária. Suponha que os elementos da árvore aparecem em pré-ordem e que um ponto (.) representa uma subárvore vazia. Assim a seqüência **abg..c.de.f....** representa a seguinte árvore binária:

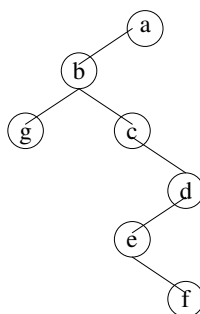


Figura 5 - Construção da árvore binária

2.1.3. Inserção, Remoção e Consulta ordenada em Árvore Binária:

A estrutura de árvore binária é indicada para manutenção ordenada de dados. Assim, se usarmos o critério que dados maiores devem ser inseridos no ramo direito e que os dados menores devem ser inseridos no ramo esquerdo, a árvore quando percorrida em-ordem terá todos os seus dados ordenados.

Inserção em Árvore Binária:

A inserção dos valores 4, 2, 1, 3, 7, 5 e 8 usando o critério citado anteriormente dá como resultado a seguinte árvore binária:

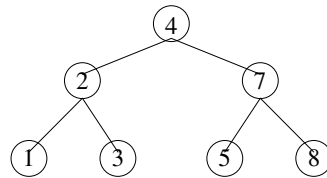


Figura 6 - Inserção em árvore binária

Esta árvore percorrida em ordem dá como resultado estes dados ordenados: 1, 2, 3, 4, 5, 7 e 8. Observe que para toda subárvore temos os dados maiores que o nó raiz à direita e os dados menores que o nó raiz à esquerda. Assim, os valores maiores que 4 (5, 7 e 8) estão à direita. O valor 1 que é menor que 2 está à esquerda de 2 e assim por diante, para todos os nós.

Remoção da Árvore Binária:

Para remover um nó de uma árvore temos que analisar dois casos:

Caso 1: O nó a ser removido possui uma ou ambas subárvores vazias. Neste caso a remoção é feita com a ligação do subramo esquerdo ou direito com o próximo nó. Exemplo:

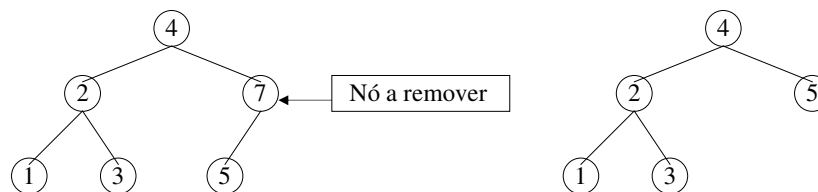


Figura 7 - Remoção de nó com subárvore vazia

Caso 2: Nenhuma das subárvores do nó é vazia. Neste caso o nó a ser removido deve ser substituído pelo símbolo seguinte na ordem a qual a árvore está organizada. O símbolo seguinte pode ser, por exemplo, o menor dos maiores, ou seja, aquele que contém o menor valor da subárvore direita do nó a ser removido. Exemplo:

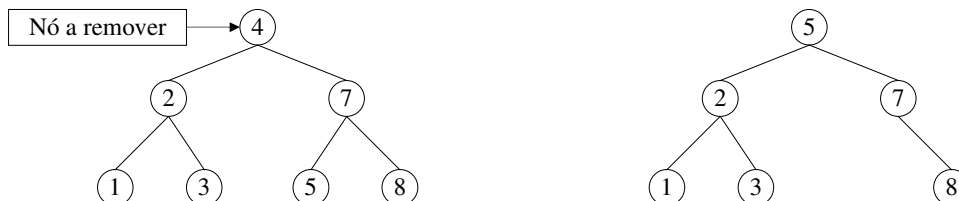


Figura 8 - Remoção de nó sem subárvores vazias

Consulta da Árvore Binária:

Para procurar um nó na árvore basta acompanhar a ordenação dos elementos em relação ao valor procurado. Então, por exemplo, para procurar o valor 3 na árvore binária ordenada anterior, compara-se 3 com a raiz (4), 3 é menor que 4, logo o valor 3 se estiver presente na árvore binária estará na subárvore esquerda do nó 4. Esta verificação acontece até que o valor procurado seja encontrado ou que seja alcançada uma subárvore vazia.



Acesse o ambiente virtual de aprendizagem no item atividades e faça os exercícios propostos sobre árvore.

Implementação da Árvore Binária:

Para a implementação da árvore, utilizaremos o tipo declarado abaixo, que segue o mesmo encadeamento proposto na implementação da lista duplamente encadeada.

```
type arvore = ^no;
  no = record
    esq : arvore;
    info : char;
    dir : arvore;
  end;
```

Programa 3 - Declaração do tipo para implementação da estrutura árvore

O tipo árvore declarado é um ponteiro que guarda endereço de nó. O nó é composto por um campo chamado info que irá guardar o conteúdo do nó da árvore, no exemplo acima é do tipo char, ou seja, esta árvore será construída com caracteres. Ela possui ainda, dois ponteiros esq e dir que também são do tipo árvore, portanto o ponteiro esq estará apontado para o nó da subárvore a esquerda e o nó dir estará apontando para o nó da subárvore a direita, conforme Figura 9

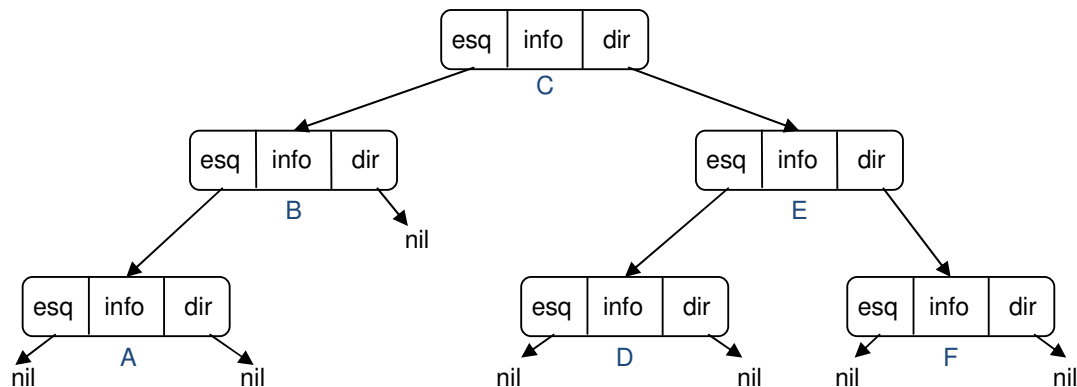


Figura 9 - Estrutura de implementação da árvore binária

Na construção da árvore, será utilizado o conceito já apresentado, onde os valores são apresentados de forma seqüencial, aparecendo em pré-ordem e o ponto representa subárvore vazia.

```
procedure constroi (var a : arvore);
var c: char;
begin
  c := readkey;
  write (c);
  if c = '.' then
    a := nil
  else begin
    new (a);
    a^.info := c;
    constroi (a^.esq);
    constroi (a^.dir);
  end;
end;
```

Programa 4 - Procedimento que constrói uma árvore

Observe que este procedimento utiliza recursividade para construir a árvore a partir de uma sequência de valores digitados. Perceba que cada nó alocado recebe o valor digitado e precisa passar por uma chamada recursiva da subárvore esquerda e depois da subárvore direita para então estar totalmente construído. Para alterar a sequência lógica de construção da árvore binária, basta alterar as posições do comando de criação do nó, inserção da informação e das chamadas recursivas, conforme rotina de percurso em árvore binária que será apresentada mais a frente.

```
procedure destroi (var a : arvore);
begin
  if a <> nil then
    begin
      destroi (a^.esq);
      destroi (a^.dir);
      dispose (a);
      a := nil;
    end;
end;
```

Programa 5 - Procedimento que destrói uma árvore

A implementação de destruir uma estrutura árvore proposta, percorre recursivamente a subárvore esquerda e direita desalocando os nós. Note que assim, nenhum espaço de memória alocado para o nó, deixará de ser devolvido ao sistema operacional e que os nós serão desalocados das folhas para a raiz.

```
procedure insere (var a : arvore; c : char);
begin
  if a <> nil then
    begin
      if c > a^.info then
        insere (a^.dir, c)
      else insere (a^.esq, c);
    end
  else
    begin
      new (a);
      a^.info := c;
      a^.dir := nil;
      a^.esq := nil;
    end;
end;
```

Programa 6 - Procedimento de inserção ordenada na árvore binária

A rotina de inserção ordenada na árvore binária guarda os valores, de forma que ao percorrer a árvore em-ordem, os valores listados estejam ordenados. Perceba que antes de inserir o valor na árvore, a rotina verifica se o valor a inserir é maior que o valor do nó atual, se for ela faz uma chamada recursiva enviando o nó da direita, senão ela faz a chamada recursiva enviando o nó da esquerda. Isso se repete até que ela encontre uma subárvore vazia. Este procedimento de verificar se o valor a inserir é maior ou menor que o nó atual é utilizado para manter o conceito de ordem apresentado acima, onde os valores a esquerda de um nó são sempre menores que o nó atual e os valores a direita deste mesmo nó são sempre maiores que ele.

```
function inter (p : arvore) : arvore;
var r,s : arvore;
begin
  r := p^.dir;
  s := r;
  while r <> nil do
    begin
      s := r;
      r := r^.esq;
    end;
end;
```

```

end;
inter := s;
end;

```

Programa 7 - Rotina de encontrar o menor dos maiores em uma subárvore

No caso de remoção de um nó que não possua subárvore vazia é necessário encontrar o menor dos maiores ou o maior dos menores de uma subárvore conforme explicitado acima. A função acima realiza esta busca, onde a variável "r" declarada localmente para a rotina inicialmente recebe o nó a direita do enviando, indicando que a subárvore a ser percorrida será a direita, onde se encontra os valores maiores que o nó raiz enviado dessa subárvore. A variável "s" estará sempre um nó anterior a variável "r" que enquanto não for nula, estará sendo direcionada para a subárvore esquerda, ou seja, caminhando para o menor valor dessa subárvore. Quando esta variável "r" receber o valor nulo indica que um nó anterior a ela é o nó folha da subárvore que contém o menor valor da subárvore. Este valor estará armazenado na variável "s" e retornará seu conteúdo para a chamada da função.

```

procedure retira (var a : arvore; c : char);

```

```

var p, q : arvore;

```

```

begin

```

```

    p := a;

```

```

    q := nil;

```

```

    while (p^.info <> c) and (p <> nil) do

```

```

    begin

```

```

        q := p;

```

```

        if c < p^.info then

```

```

            p := p^.esq

```

```

        else p := p^.dir;

```

```

    end;

```

```

    if p = nil then

```

```

    begin

```

```

        writeln ('No nao encontrado');

```

```

    end

```

```

    else

```

```

    begin

```

```

        if p^.esq = nil then

```

```

        begin

```

```

            if q = nil then

```

```

                a := p^.dir

```

```

            else

```

```

            begin

```

```

                if q^.esq = p then

```

```

                    q^.esq := p^.dir

```

```

                else

```

```

                    q^.dir := p^.dir;

```

```

                    dispose (p);

```

```

            end

```

```

        end

```

```

    end

```

```

    else

```

```

    begin

```

```

        if p^.dir = nil then

```

```

        begin

```

```

            if q = nil then

```

```

                a := p^.esq

```

```

            else

```

```

            begin

```

```

                if q^.esq = p then

```

```

                    q^.esq := p^.esq

```

```

                else q^.dir := p^.esq;

```

```

                    dispose (p);

```

```

                end

```

```

            end

```

```

        end

```

```

    end

```

```

end

```

Busca na árvore o valor enviado na variável c

Remoção do nó raiz

Faz a remoção do nó quando a subárvore esquerda está vazia

Remoção do nó raiz

Faz a remoção do nó quando a subárvore direita está vazia

<pre> else begin q := inter (p); p^.info := q^.info; retira (p^.dir, q^.info); end; end; end; end;</pre>	<div style="border: 1px solid black; padding: 5px;"> <p>Faz a remoção do nó quando as duas subárvores não são vazias. Encontra o menor dos maiores, reposiciona este valor na árvore e chama recursivamente a rotina enviando a subárvore direita e o valor reposicionado</p> </div>
--	--

Programa 8 - Procedimento que retira um nó da árvore

Perceba na rotina retira que o nó desalocado é sempre um nó folha.


Quando o nó a ser removido possui as duas subárvores não vazias, realiza-se a chamada da rotina inter, que irá retornar o menor valor da subárvore a direita do nó a ser removido que deverá substituir a posição do valor a ser retirado. De posse do endereço desse nó, substitui-se a informação da posição a ser removida pela informação do nó retornado da rotina inter e chama-se recursivamente a rotina retira, para que o nó retornado da rotina inter seja removido.

```

procedure pre_ordem (a : arvore);
begin
    if a <> nil then
        begin
            write (a^.info);
            pre_ordem (a^.esq);
            pre_ordem (a^.dir);
        end
    end;
procedure em_ordem (a : arvore);
begin
    if a <> nil then
        begin
            em_ordem (a^.esq);
            write (a^.info);
            em_ordem (a^.dir);
        end
    end;
procedure pos_ordem (a : arvore);
begin
    if a <> nil then
        begin
            pos_ordem (a^.esq);
            pos_ordem (a^.dir);
            write (a^.info);
        end
    end;
end;
```

Programa 9 - Rotinas de percurso em árvore binária

Perceba que as rotinas de percurso na árvore binária são todas recursivas e elas se diferem apenas pela posição do comando de escrita da informação do nó.

	<p>Acesse o ambiente virtual de aprendizagem no item atividades e faça os exercícios propostos sobre implementação da árvore.</p>
---	---

3. Métodos de Classificação

Classificação ou ordenação de dados é uma das tarefas mais freqüentes e importantes em processamento de dados. A importância da classificação de dados pode ser avaliada se considerarmos o que seria o problema de localização do nome de um assinante em uma lista telefônica, na qual os nomes não estivessem em ordem alfabética (Tenenbaum, 2005).

Classificar ou ordenar um conjunto de dados consiste em receber como entrada uma seqüência de n valores $\langle a_1, a_2, \dots, a_n \rangle$ e produzir como resultado uma permutação (reordenamento) dos valores de entrada $\langle a_1', a_2', \dots, a_n' \rangle$ de tal forma que $a_1' \leq a_2' \leq \dots \leq a_n'$.

Exemplo: para a seqüência de entrada for $\langle 21, 32, 45, 12, 8, 57 \rangle$ o resultado de um algoritmo de classificação deve ser: $\langle 8, 12, 21, 32, 45, 57 \rangle$.

O processo de classificação de um conjunto de dados inteiramente contido na memória principal é chamado de classificação interna, ao passo que a classificação de um conjunto de dados não inteiramente armazenado na memória primária é chamada de classificação externa.

Existem 3 métodos gerais para classificação interna de dados:

- classificação por troca
- classificação por seleção
- classificação por inserção

Para entender estes métodos, suponha o problema de ordenação das cartas do baralho.

- **Classificação por troca:** Espalhe as cartas numa mesa voltadas para cima e então troque as cartas de ordem até que todo o baralho esteja ordenado.
- **Classificação por Seleção:** Espalhe as cartas na mesa, selecione a carta de menor valor, retire-a do baralho e segure-a na sua mão. Este processo continua até que todas as cartas estejam na sua mão.
- **Classificação por Inserção:** Segure todas as cartas na sua mão. Ponha uma carta por vez na mesa, sempre inserindo na posição correta. O maço estará ordenado quando não restarem mais cartas em sua mão.

3.1. Classificação por Troca

Estes métodos caracterizam-se por efetuarem a classificação por comparação sucessiva de pares de elementos, trocando-os de posição caso estejam fora da ordem desejada.

```
const MAX = 10;  
type tipovetor = array[1..MAX] of integer;
```

Programa 10 - Estrutura base para implementação dos algoritmos de ordenação

Para a implementação das rotinas de ordenação utilizaremos uma constante e um vetor como estrutura para aplicação dos algoritmos.

```
procedure BubbleSort (var vetor : tipovetor);  
var troca: boolean;  
    i,salva: integer;  
begin  
    repeat  
        troca := false;  
        for i := 1 to MAX-1 do  
            begin  
                if (vetor[i] > vetor[i+1]) then  
                    begin  
                        salva := vetor[i];  
                        vetor[i] := vetor[i+1];  
                        vetor[i+1] := salva;
```

```

                                troca := true;
                                end;
                                until not troca;
                                end;
end;

```

Programa 11 - Rotina de classificação por troca

Para o vetor <50,73,54,89,22,46,91,11,29,2> a sequência de passos que será executado por este método para ordenação do vetor é:

50	73	54	89	22	46	91	11	29	2
50	54	73	22	46	89	11	29	2	91
50	54	22	46	73	11	29	2	89	91
50	22	46	54	11	29	2	73	89	91
22	46	50	11	29	2	54	73	89	91
22	46	11	29	2	50	54	73	89	91
22	11	29	2	46	50	54	73	89	91
11	22	2	29	46	50	54	73	89	91
11	2	22	29	46	50	54	73	89	91
2	11	22	29	46	50	54	73	89	91
2	11	22	29	46	50	54	73	89	91

Tabela 1 - Acompanhamento da classificação por troca

Obs: O nome Bubble (bolha) deve-se ao fato que durante o processo de ordenação é como se os valores menores fossem as bolhas flutuando para ocupar as primeiras posições no vetor.

3.2. Classificação por Seleção

Nestes métodos, a classificação é efetivada por seleção sucessiva do menor valor dentro do vetor. A cada passo o elemento de menor valor é colocado em sua posição definitiva no vetor classificado e o processo é repetido para o segmento que contém os elementos ainda não selecionados.

```

procedure Selecao (var vetor: tipovetor);
var i, j, menor, salva : integer;
begin
  for i := 1 to MAX-1 do
    begin
      menor := i;
      for j := i+1 to MAX do
        if (vetor[j] < vetor[menor])
          then menor := j;
      salva := vetor[i];
      vetor[i] := vetor[menor];
      vetor[menor] := salva;
    end;
  end;
end;

```

Programa 12 - Rotina de classificação por seleção

Para o vetor <52,75,36,20,17,46,60,78,51,80> a sequência de passos que será executado por este método para ordenação do vetor é:

52	75	36	20	17	46	60	78	51	80
17	75	36	20	52	46	60	78	51	80
17	20	36	75	52	46	60	78	51	80
17	20	36	75	52	46	60	78	51	80
17	20	36	46	52	75	60	78	51	80
17	20	36	46	51	75	60	78	52	80
17	20	36	46	51	52	60	78	75	80
17	20	36	46	51	52	60	78	75	80
17	20	36	46	51	52	60	75	78	80

Tabela 2 - Acompanhamento da classificação por seleção

3.3. Classificação por Inserção

A característica comum de todos os métodos de classificação por inserção é que eles efetivam a ordenação do vetor pela inserção de cada um dos elementos em sua posição dentro de um subvetor classificado.

```

procedure Insercao (var vetor: tipovetor);
var i, j, x : integer;
begin
  i := 2;
  while i <= MAX do
  begin
    x := vetor[i];
    vetor[0] := vetor[i];
    j := i-1;
    while x < vetor[j] do
    begin
      vetor[j+1] := vetor[j];
      j := j-1;
    end;
    vetor[j+1] := x;
    i := i+1;
    mostra(vetor);
  end;
end;

```

Programa 13 - Rotina de classificação por inserção

Para o vetor <33,69,12,80,36,46,94,71,30,74> a sequência de passos que será executado por este método para ordenação do vetor é:

33	69	12	80	36	46	94	71	30	74
33	69	12	80	36	46	94	71	30	74
12	33	69	80	36	46	94	71	30	74
12	33	69	80	36	46	94	71	30	74
12	33	36	69	80	46	94	71	30	74
12	33	36	46	69	80	94	71	30	74
12	33	36	46	69	80	94	71	30	74
12	33	36	46	69	71	80	94	30	74
12	30	33	36	46	69	71	80	94	74
12	30	33	36	46	69	71	74	80	94

Tabela 3 - Acompanhamento da classificação por inserção



Acesse o ambiente virtual de aprendizagem no item atividades e faça os exercícios propostos sobre classificação.

4. Bibliografia

TENENBAUM, AARON M., LANGSAM, YEDIDYAH, AUGENSTEIN, MOSHE J. – ***Estrutura de Dados usando C***, Makron Books, São Paulo, 2005.

PEREIRA, SÍLVIO DO LAGO - ***Estrutura de Dados Fundamentais: conceitos e aplicações***, Érica, São Paulo, 1996.

VELOSO, PAULO, SANTOS, CLÉSIO DOS, AZEREDO, PAULO e FURTADO, ANTÔNIO - ***Estrutura de Dados***, Campus, Rio de Janeiro, 1986.

SZWARCFTER, JAYME LUIZ - ***Estrutura de Dados e seus algoritmos***, LTC, Rio de Janeiro, 1994.

VILLAS, MARCOS VIANA - ***Estrutura de Dados: conceitos e técnicas de implementação***, Campus, Rio de Janeiro, 1993.