Aula 08 - Conteúdo

1) Algoritmos de Ordenação
2) Algoritmos de Busca
3) Implementações destes algoritmos
4) Exercícios

Introdução

Estudaremos, neste tópico, os seguintes métodos de classificação:

a) Classificação por troca
- Buble Sort (método bolha)
- Quick Sort

b) Classificação por seleção
- Selection Sort (método da seleção)

c) Classificação por inserção
- Insertion Sort (método da inserção)
- Shell Sort

d) Classificação por intercalação
- Merge Array
- Merge Sort

e) Classificação de Raízes
- Radix Sort

Para efeitos de explicação utilizaremos tanto nos métodos de busca quanto nos métodos de ordenação, nos casos que assim permitirem, um vetor **v** de inteiros de tamanho **n**, sendo que os índices do vetor **x** assumiram valores entre **0** e **n-1**.

Obs.: apesar dos algoritmos implementados utilizarem um vetor, podemos estender a lógica das classificações também para arquivos.

Classe utilizada na implementação dos métodos:

```
TVetInt = class
  protected
    v: array of integer;
    tamanho: integer;
    ordenado: boolean;
    procedure Particionar(lb,ub:integer; var j:integer);
    Procedure QuickSort(lb,ub:integer);
    Procedure ShellSort(incrmnts: array of integer);
  public
    Constructor InicVetInt(tam: integer);
    Destructor  DelVetInt;
    Procedure Imprime;
    Function  GetTamanho: integer;
    Procedure Buble;
    Procedure Quick;
    Procedure Selection;
    Procedure Insertion;
    Procedure Shell;
    Procedure Merge;
    Procedure Radix;
    Function BuscaSeq(val: integer):integer;
    Function BuscaBin(val: integer):integer;
    Function VetMaior:integer;
  end;
```

Algoritmos de Ordenação

Classificação por Troca

Buble Sort (Método Bolha – arquivo ProjEx801.dpr e Ex801.pas)

A idéia básica do Buble Sort é percorrer os dados do vetor (arquivo) sequencialmente várias vezes. Em cada passagem pelos dados devemos comparar cada elemento com o seu sucessor (x[k] com x[k+1]). Se os elementos não estiverem ordenados devemos trocá-los de posição.

Exemplo:

Iteração 0:	25	57	48	37	12	92	86	33
Iteração 1:	25	48	37	12	57	86	33	**92**
Iteração 2:	25	37	12	48	57	33	**86**	**92**
Iteração 3:	25	12	37	48	33	**57**	**86**	**92**
Iteração 4:	12	25	37	33	**48**	**57**	**86**	**92**
Iteração 5:	12	25	33	**37**	**48**	**57**	**86**	**92**
Iteração 6:	12	25	**33**	**37**	**48**	**57**	**86**	**92**
Iteração 7:	12	**25**	**33**	**37**	**48**	**57**	**86**	**92**

Note que na iteração 1 o maior elemento (**92**) está na posição correta, na iteração 2 o elemento **86** está na posição correta e na iteração **k** o elemento **x[n-k]** estará na posição correta. Podemos melhorar a rapidez deste algoritmos com a seguinte alteração:

Iteração 1:	(n-1) comparações
Iteração 2:	(n-2) comparações
Iteração 3:	(n-3) comparações
:	:
última Iteração:	1 comparação (x[0] com x[1])

Obs.: note que pode ocorrer dos dados estarem ordenados antes de se efetuar (n-1) passagens pelo vetor

Algoritmo Buble Sort

```
Procedure TVetInt.Buble;
var aux,j,passo,chave: integer;
begin
  if ( ordenado ) then
  begin
    ShowMessage('Vetor já está ordenado!');
    Exit;
  end;
  chave := 1;
  for passo:=0 to tamanho-1 do
  begin
    if ( chave = 1 ) then
    begin
      chave:=0;
      for j := 0 to (tamanho-passo-1) do
      begin
        if (v[j] > v[j+1]) then
        begin
          chave:=1;
          aux:=v[j];
          v[j]:=v[j+1];
          v[j+1]:=aux;
        end
```

```
      end
    end;
  end;
  ordenado := TRUE;
  Imprime;
end;
```

Quicksort (Método Rápido – arquivo ProjEx801.dpr e Ex801.pas)

Devemos escolher um elemento **b** qualquer do vetor **x** (por exemplo o 1º elemento - b=x[0]) e baseado neste elemento devemos particionar **x** as seguinte forma:

- colocamos **b** na sua posição correta **j**
- todos os elementos entre **0** e **j-1** devem ser menores do que **b** (x[j])
- todos os elementos entre **j+1** e **n** devem ser maiores do que **b** (x[j])

Após particionarmos o vetor devemos repetir o processo para os subvetores entre x[0] até x[j-1] e x[j+1] até x[n-1] e assim sucessivamente.

Exemplo:

25	57	48	37	12	92	86	33
12	**25**	48	37	57	92	86	33
12	**25**	48	37	57	92	86	33
12	**25**	33	37	**48**	92	86	57
12	**25**	**33**	37	**48**	92	86	57
12	**25**	**33**	**37**	**48**	92	86	57
12	**25**	**33**	**37**	**48**	**57**	86	92
12	**25**	**33**	**37**	**48**	**57**	**86**	92
12	**25**	**33**	**37**	**48**	**57**	**86**	**92**

Algoritmo Quicksort

```
procedure TVetInt.Particionar(lb,ub:integer; var j:integer);
var a,temp,down,up: integer;
begin
  a := v[lb];
  up := ub;
  down := lb;
  while ( down < up ) do
  begin
    while ( (v[down] <= a) and (down < ub) ) do down:=down+1;
    while ( v[up] > a ) do up := up - 1;
    if ( down < up ) then
    begin
      temp := v[down];
      v[down] := v[up];
      v[up] := temp;
    end;
  end;
  v[lb]:=v[up];
  v[up]:=a;
  j := up;
end;

Procedure TVetInt.QuickSort(lb,ub:integer);
var j: integer;
begin
  if ( lb < ub ) then
  begin
    Particionar(lb,ub,j);
```

```
    QuickSort(lb,j-1);
    QuickSort(j+1,ub);
  end;
end;

Procedure TVetInt.Quick;
begin
  if ( ordenado ) then ShowMessage('Vetor já está ordenado!')
  else
  begin
    QuickSort(0,tamanho-1);
    ordenado := True;
    Imprime;
  end;
end;
```

Classificação por Seleção

Selection Sort (Método da Seleção – arquivo ProjEx801.dpr e Ex801.pas)

Este método consiste em encontrar repetidamente o maior elemento do vetor x (ou subvetor) e colocá-lo na sua devida posição.

Maior dos (n-1) elementos:	trocar com o n-ésimo elemento
Maior dos (n-2) elementos:	trocar com o (n-1)-ésimo elemento
:	:
Maior dos 2 últimos elementos	torcar com o segundo elemento

Exemplo:

Iteração 0:	25	57	48	37	12	92	86	33
Iteração 1:	25	57	48	37	12	33	86	**92**
Iteração 2:	25	57	48	37	12	33	**86**	**92**
Iteração 3:	25	33	48	37	12	**57**	**86**	**92**
Iteração 4:	25	33	12	37	**48**	**57**	**86**	**92**
Iteração 5:	25	33	12	**37**	**48**	**57**	**86**	**92**
Iteração 6:	25	12	**33**	**37**	**48**	**57**	**86**	**92**
Iteração 7:	12	**25**	**33**	**37**	**48**	**57**	**86**	**92**

Algoritmo Selection Sort

```
Procedure TVetInt.Selection;
var i,indmaior,j,maior: integer;
begin
  if ( ordenado ) then
  begin
    ShowMessage('Vetor já está ordenado!');
    Exit;
  end;
  for i:= tamanho-1 downto 1 do
  begin
    maior := v[0];
    indmaior := 0;
    for j:=1 to i do
    begin
      if ( v[j] > maior ) then
      begin
      maior := v[j];
```

```
      indmaior := j;
      end;
    end;
    v[indmaior] := v[i];
    v[i] := maior;
  end;
  ordenado := TRUE;
  Imprime;
end;
```

Classificação por Inserção

Insertion Sort (Método da Inserção – arquivo ProjEx801.dpr e Ex801.pas)

Neste método devemos inserir os elementos nas suas posições corretas utilizando o arquivo (vetor) original de dados. Em cada iteração tomamos um elemento e o colocamos na sua posição correta deslocando os elemenstos que estão em posições incorretas.

Exemplo:

Iteração 0:	25	57	48	37	12	92	86	33
Iteração 1:	25	57						
Iteração 2:	25	48	57					
Iteração 3:	25	37	48	57				
Iteração 4:	12	25	37	48	57			
Iteração 5:	12	25	37	48	57	92		
Iteração 6:	12	25	37	48	57	86	92	
Iteração 7:	12	25	33	37	48	57	86	92

Algoritmo Insertion Sort

```
Procedure TVetInt.Insertion;
var i,k,y: integer;
begin
  if ( ordenado ) then
  begin
    ShowMessage('Vetor já está ordenado!');
    Exit;
  end;
  for k:=1 to tamanho-1 do
  begin
    y := v[k];
    i := k-1;
    while ( (i >= 0) and (y < v[i]) ) do
    begin
      v[i+1] := v[i];
      i := i - 1;
    end;
    v[i+1] := y;
  end;
  ordenado := TRUE;
  Imprime;
end;
```

Shell Sort (arquivo ProjEx801.dpr e Ex801.pas)

A classificação Shell ou **classificação de incremento decrescente** particiona o arquivo (vetor) original em **k** subarquivos (subvetores) e depois classifica cada um deles. Após a ordenação dos k subarquivos, será definido um novo valor de k (menor que o anterior) e o processo de particionamento e classificação de subarquivos continuará até o arquivo (vetor) original estar ordenado.

Exemplo:

Seja o arquivo (vetor) original abaixo:

```
x = (25 57 48 37 12 92 86 33)
```

Considere a seqüência de incrementos k = (5, 3, 1)

Iteração 1 (k = 5) – subarquivos (subvetores) abaixo

```
(25 92)  (x[0] x[5]) =>  (25 92)
(57 86)  (x[1] x[6]) =>  (57 86)
(48 33)  (x[2] x[7]) =>  (33 48)
(37)     (x[3])      =>  (37)
(12)     (x[4])      =>  (12)
x = (25 57 33 37 12 92 86 48)
```

Iteração 2 (k = 3) – subarquivos (subvetores) abaixo

```
(25 37 86)  (x[0] x[3] x[6]) =>  (25 37 86)
(57 12 48)  (x[1] x[4] x[7]) =>  (12 48 57)
(33 92)     (x[2] x[5])      =>  (33 92)
x = (25 12 33 37 48 92 86 57)
```

Iteração 2 (k = 1) – subarquivo (subvetor) abaixo

```
(25 12 33 37 48 92 86 57) (x[0] x[1] ... x[7])
x = (25 12 33 37 48 92 86 57)
```

Algoritmo Shell Sort

```
Procedure TVetInt.ShellSort(incrmnts: array of integer);
var numinc,incr,j,k,tam,y : integer;
begin
  numinc := High(incrmnts);
  for incr := 0 to numinc do
  begin
    tam := incrmnts[incr];
    for j := tam to tamanho - 1 do
    begin
      y := v[j];
      k := j - tam;
      while( (k >= 0) and (y < v[k]) ) do
      begin
        v[k+tam] := v[k];
        k := k - tam;
      end;
      v[k+tam] := y;
    end
  end
end;
```

```
Procedure TVetInt.Shell;
var incrmnts: array of integer;
begin
  if ( ordenado ) then
  begin
    ShowMessage('Vetor já está ordenado!');
    Exit;
  end;
  SetLength(incrmnts,3);
  incrmnts[0] := 5;
  incrmnts[1] := 3;
  incrmnts[2] := 1;
  ShellSort(incrmnts);
  ordenado := TRUE;
  Imprime;
end;
```

Classificação por Intercalação

A classificação por intercalação tem como idéia principal combinar dois ou mais vetores (arquivos) classificados num outro vetor (arquivo) também classificado.

Merge Array (arquivo ProjEx801.dpr e Ex801.pas)

O método abaixo toma dois vetores já ordenados **a** e **b** de tamanhos **na** e **nb**, respectivamente, e monta um vetor **c** de tamanho **nc** (na+nb), também ordenado, através da intercalação dos elementos de **a** e **b**.

Exemplo:

```
a = (5 7 10 15 20 25)    na = 6
b = (2 4 12 13 17 32 40) nb = 7
c = (2 4 5 7 10 12 13 15 17 20 25 32 40) nc = 13
```

Merge Sort (arquivo ProjEx801.dpr e Ex801.pas)

Neste método devemos dividir o vetor (arquivo) em **n** subvetores (subarquivos) de tamanho 1 e intercalar pares de vetores adjacentes. Neste ponto teremos aproximadamente **n/2** vetores ordenados de tamanho 2. Devemos agora intercalar os vetores adjacentes de tamanho 2 em vetores de tamanho 4. Teremos o vetor totalmente ordenado se repetirmos este processo até obtermos um único vetor de tamano de n.

Exemplo:

Vetor original:	25	57	48	37	12	92	86	33
Iteração 1:	25	57	37	48	12	92	33	86
Iteração 2:	25	37	48	57	12	33	86	92
Iteração 3:	12	25	33	37	48	57	86	92

Algoritmo Merge Sort

```
Procedure TVetInt.Merge;
var i,j,k,a1,a2,u1,u2,tam: integer;
    aux: array of integer;
begin
  if ( ordenado ) then
  begin
    ShowMessage('Vetor já está ordenado!');
    Exit;
  end;
  SetLength(aux,tamanho);
  tam := 1; //intercala subvetores de tamanho 1
  while (tam < tamanho ) do
  begin
    a1 := 0; k := 0;
    while ((a1 + tam) < tamanho ) do
    begin
      a2 := a1 + tam;
      u1 := a2 - 1;
      if ( (a2+tam-1) < tamanho ) then u2 := a2+tam-1
      else u2 := tamanho - 1;
      i := a1; j := a2;
      while ( (i <= u1) and (j <= u2) ) do
      begin
        if (v[i] <= v[j]) then begin aux[k] := v[i]; i:= i+1; end
        else begin aux[k] := v[j]; j := j+1; end;
        k := k+1;
      end;
      while ( i <= u1 ) do
      begin
        aux[k] := v[i];
        k := k+1;
        i := i+1;
      end;
      while ( j <= u2 ) do
      begin
        aux[k] := v[j];
        k := k+1;
        j := j+1;
      end;
      a1 := u2+1;
    end;
    //copia o restante de x em aux
    i := a1;
    while ( k < tamanho ) do
    begin
      aux[k] := v[i];
      k := k+1;
      i := i+1;
    end;
    //copia aux em x
    for i := 0 to tamanho-1 do v[i] := aux[i];
    tam := tam * 2;
  end;
  ordenado := TRUE;
  Imprime;
end;
```

Classificação de Raízes

Radix Sort (arquivo ProjEx801.dpr e Ex801.pas)

Este método de classificação baseia-se nas representações posicionais dos valores dos dígitos dos números a serem classificados.

Exemplo:

Vetor original x = (25 57 48 37 12 92 86 33)

Filas baseadas no dígito menos significativo

	Início		Final
Fila [0]			
Fila [1]			
Fila [2]	12	...	92
Fila [3]	37	...	33
Fila [4]			
Fila [5]	25		
Fila [6]	86		
Fila [7]	57		
Fila [8]	48		
Fila [9]			

Após a primeira passagem temos:

Vetor x = (12 92 33 25 86 57 37 48)

Filas baseadas no dígito mais significativo

	Início		Final
Fila [0]			
Fila [1]	12		
Fila [2]	25		
Fila [3]	33	...	37
Fila [4]	48		
Fila [5]	57		
Fila [6]			
Fila [7]			
Fila [8]	86		
Fila [9]	92		

Após a segunda passagem temos:

Vetor x = (12 25 33 37 48 57 86 92)

Algoritmo Radix Sort em notação algoritmica

```
Para (k=dígito menos signif; k <= dígito mais signif; k++) faça
{
  para (i = 0; i < n; i++) faça
  {
    y = x[i];
    j = k ésimo dígito de y;
    InsFila(fila[j],y);
  }
  k = 0;
  para (q = 0; q < 10; q++) faça
  {
    Enquanto ( não FilaVazia(fila[q]) )
```

```
    {
      y = DelFila(fila[q]);
      x[k++] = y;
    }
  }
```

Algoritmo Radix Sort em Pascal

```
//as funções utilizadas no código abaixo e não apresentadas
//estão implementadas nos arquivos ProjEx801.dpr e Ex801.pas
Procedure TVetInt.Radix;
var k,i,j,ndig,y: integer;
  Function pega_digito(num,posrel: integer): integer;
  begin
    if ( posrel = UNIDADE  ) then pega_digito := (num mod 10)
    else if ( posrel = DEZENA   ) then
      pega_digito := ((num div 10) mod 10)
    else if ( posrel = CENTENA  ) then
      pega_digito := ((num div 100) mod 10)
    else if ( posrel = MILHAR   ) then
      pega_digito := ((num div 1000) mod 10)
    else if ( posrel = DEZMILHAR) then
      pega_digito := ((num div 10000) mod 10)
  end;
  Function num_digitos(num: integer): integer;
  begin
    if ( pega_digito(num,DEZMILHAR) <> 0 )   then num_digitos := 5
    else if ( pega_digito(num,MILHAR) <> 0)  then num_digitos :=4
    else if ( pega_digito(num,CENTENA) <> 0) then num_digitos :=3
    else if ( pega_digito(num,DEZENA) <> 0)  then num_digitos :=2
    else num_digitos := 1;
  end;
begin //método radix
  if ( ordenado ) then
  begin
    ShowMessage('Vetor já está ordenado!');
    Exit;
  end;
  Fila := TFilaLVet.InicFila(10,tamanho);
  j := VetMaior;
  ndig := num_digitos(v[j]);
  for k:=0 to (ndig-1) do
  begin
    for i :=0 to (tamanho-1) do
    begin
      y := pega_digito(v[i],k+1);
      Fila.InsFila(y,v[i]);
    end;
    j := 0;
    for i:=0 to 9 do
    begin
      while( not Fila.FilaVazia(i) ) do
      begin
      Fila.DelFila(i,y);
      v[j] := y;
        j := j+1;
      end;
    end;
  end;
  ordenado := TRUE;
  Imprime;
end;
```

Algoritmos de Busca

Sequential Search (Busca Seqüencial –arquivo ProjEx801.dpr e Ex801.pas)

Devemos percorrer o vetor **x** sequencialmente até encontrar o elemento desejado ou o seu final.

Algoritmo Sequential Search

```
Function TVetInt.BuscaSeq(val: integer):integer;
var k: integer;
begin
  for k:=0 to (tamanho - 1) do
  begin
    if ( v[k] = val ) then
    begin
      BuscaSeq := k;
      exit;
    end;
  end;
  BuscaSeq := -1;
end;
```

Binary Search (Busca Binária – arquivo ProjEx801.dpr e Ex801.pas)

Este método exige que os dados estejam previamente ordenados e é um dos métodos mais eficientes de busca.

Algoritmo Binary Search

```
Function TVetInt.BuscaBin(val: integer):integer;
var low,hi,mid : integer;
begin
  if ( not ordenado ) then
  begin
    ShowMessage('Vetor NÃO ordenado!');
    BuscaBin := -1;
    Exit;
  end;
  low := 0;
  hi := tamanho-1;
  while ( low <= hi ) do
  begin
    mid := (low + hi) div 2;
    if ( v[mid] = val ) then
    begin
      BuscaBin := mid;
      Exit;
    end;
    if ( val < v[mid] ) then hi := mid - 1
    else low := mid + 1;
  end;
  BuscaBin := -1;
end;
```

Busca Utilizando árvores (arquivo ProjEx501.dpr à ProjEx701.dpr)

As Aulas 05, 06 e 07 abordaram várias estruturas e métodos de busca utilizando árvores.

Exercícios

8.01) Aplique os métodos de ordenação Buble, Quick, Selection, Insertion, Shell, Merge e Radix Sort descritos para os seguintes vetores de inteiros:

```
a) (30 10 20 15 25)
b) (20 15 07 32 18 26 40 05 21)
c) (12 60 10 05 19 31 02 33 51 47 17 25)
```

8.02) Reescreva o método Buble Sort realizando sucessivas passagens em direções opostas. O algoritmo deve percorrer o vetor no sentido de 0 à n-1 e colocar o maior elemento no final do vetor. Após este passo deve percorrer o vetor no sentido de n-1 à 0 e colocar o menor elemento no início do vetor. Repetir este processo até ordenar o vetor.

8.03) O método Selection Sort implementado nesta seção toma a cada iteração o maior elemento do vetor e o coloca em sua posição correta. Altere o algoritmo de forma que, em cada passagem pelo vetor, o método tome o menor e o maior elemento e os coloque nas suas posições corretas (arquivo Ex810.cpp).

8.04) Escreva o algoritmo de Busca Binária utilizando recursividade.