

Pesquisa e Ordenação

SCE121- Introdução a Programação

ICMC - USP - São Carlos
2009

Algoritmo de Procura

- Algoritmo de Procura
 - O problema de procurar, pesquisar alguma informação numa tabela ou num catálogo é muito comum
- Exemplo:
 - procurar o telefone de uma pessoa no catálogo
 - procurar o n^o da conta de um certo cliente
 - consultar o seu saldo em um terminal automático

Algoritmo de Procura

- A tarefa de “pesquisa”, “procura” ou “busca” é, como se pode imaginar, *uma das funções mais utilizadas*
- rotinas que a executem de uma forma *eficiente*
- *Eficiente*: uma rotina que faça a busca no menor tempo possível
- O TEMPO GASTO pesquisando dados em tabelas depende do TAMANHO da tabela.

Algoritmo de Procura

O tempo
gasto para se
descobrir um
telefone na
lista de São
Paulo

>

O tempo gasto
para se descobrir
um telefone na
lista de uma
cidade do interior
com 3000
habitantes

Algoritmo de Procura

- TEMPO GASTO pode variar muito dependendo do algoritmo utilizado
- Para os algoritmos de busca que se seguem vamos denotar por:
 - **Tab** - um vetor de n posições
 - **Dado** - elementos que devemos procurar em **Tab**
 - **Achou** - indica o sucesso ou falha na pesquisa
 - **Ind** - aponta para a posição do elemento encontrado

Parte que se repetirá...

(*)

```
Int Tab[100;      {tabela de pesquisa}  
Int Ind;          {retorna a posição do  
                  elemento}  
boolean Achou;   {sucesso ou falha na  
                  busca}  
Int Dado;        {valor a ser procurado}  
Int I;           {auxiliar}
```

Algoritmo 1 - Busca em Tabela

```
#includes
(*)
int main(){
    printf("Entre com os valores da  tabela");
    for (i=0;i<N;i++)
        scanf("%d", &Tab[I]);
    printf("Entre com o valor a ser procurado");
    scanf("%d",&Dado);
    Achou=false;
    for (i=0;i<N;i++)
        if Tab[I] == Dado{
            Achou=true;
            Ind = I;
        }
    if Achou
        printf("%d se encontra na posição:%d", Dado, Ind)
    else printf("%d não se encontra na tabela", Dado);
    getch();
}
```

N comparações

(~ percorrer 1 dicionário todo)

Algoritmo 2 - Busca em tabela

- Pára-se o processo de busca quando o dado for encontrado.

- 1º modo

$I \leftarrow 1$

Enquanto (Tab [I] \neq dado) **and** (I \leq n) **faça**

$I = I + 1$

- Este algoritmo não funciona pois

$Ind \leftarrow N + 1$

e

Tab [N+1] - referência inválida

2º Modo (Com o uso da variável BOOLEAN)

```
Program Alg2;
```

```
(*)
```

```
int Main(){
```

```
    Achou = false; Procura= true;
```

```
    Ind = 0;
```

```
    printf("Entre com os valores da tabela");
```

```
    for (i=0;i<N;i++)
```

```
        scanf("%d", &Tab[I]);
```

```
    printf("Entre com o valor do dado");
```

```
    scanf("%d",&Dado);
```

```
    while Procura{
```

```
        Ind++;
```

```
        if Ind > N Procura= false;
```

```
        else Procura = Tab[Ind] != Dado;
```

```
    }
```

```
    if (Ind <= N) Achou = true;
```

```
    if Achou printf("%d se encontra na posição %d", Dado, Ind);
```

```
    else printf(" %d não se encontra na tabela", Dado);
```

```
}
```

**Dado pode estar na 1ª posição
ou Dado pode estar na última
Na média: $N/2$ comparações**

Obs: 2 testes

- Procura = true
- Ind > N

Algoritmo 3: Busca com Sentinela

- Se soubermos que o dado se encontra na tabela na precisaríamos fazer o teste
Ind > N
- **INSERIR** o Dado no final da tabela

```

Program Alg3; {Bem + simples}
(*)
begin
    printf("Entre com os valores da  tabela");
    for (i=0;i<N;i++)
        scanf("%d", &Tab[i]);
    printf("Entre com o valor a ser procurado");
    scanf("%d",&Dado);
    Achou= false;
    Ind= 0;
    Tab[N]= Dado;
while (Tab[Ind] != Dado){
        Ind++;
    }
    Achou= Ind != N;
    if Achou
        printf("%d se encontra na posição %d",Dado, Ind);
    else printf(" %d não se encontra na tabela", Dado);
end.

```

Algoritmo 4 -Busca binária (+ eficiente)

Dicionário - *Tarol*

- Abre-se o dicionário ao meio → letra J
- Abandonamos a 1ª metade
- Tomamos a metade a partir de J → letra P
- Abandonamos a 1ª metade
- Tomamos a metade a partir de P → letra S (pág. 1318)
- Dividimos novamente, chegamos a palavra *Tomo* (pág. 1386)

∴ palavra está entre 1318 e 1386

Algoritmo 4 - Busca binária

- A cada passo dividimos a área de pesquisa à metade
- Caso o dicionário tenha 1500 palavras

$$1500/2 \rightarrow 750$$

$$24/2 \rightarrow 12$$

$$750/2 \rightarrow 375,5$$

$$12/2 \rightarrow 6$$

$$376/2 \rightarrow$$

$$188/2 \rightarrow$$

$$94/2 \rightarrow 47$$

$$47/2 \rightarrow 23,5$$

$$2/2 \rightarrow 1$$

11 pesquisas = $\log_2 1500$

$N \Rightarrow \log_2 N$

$32.000 \Rightarrow 15$ comparações

Program Alg4;

(*)

Int Inicio, Fim, Meio;

begin

printf("Entre com os valores da tabela");

for (i=0;i<N;i++)

scanf("%d", &Tab[I]);

printf("Entre com o valor a ser procurado");

scanf("%d",&Dado);

Achou= false;

Inicio=0; Fim= N-1; Meio = (1+N)/2;

while (Dado != Tab[Meio]) **and** (Inicio != Fim){

if Dado > Tab[Meio]

Inicio = Meio + 1;

else Fim = Meio;

Meio= (Inicio + Fim) / 2;

}

Achou = Dado == Tab[Meio];

if Achou

printf("%d se encontra na posição %d",Dado, Meio);

else printf(" %d não se encontra na tabela", Dado);

end.

Algoritmos de Ordenação

Algoritmos de Ordenação

- Da mesma forma que a BUSCA, a ORDENAÇÃO é uma das tarefas básicas em processamento de dados
- A *ordenação* de um vetor significa fazer com que os seus elementos estejam colocados de acordo com algum critério de ordenação

Algoritmos de Ordenação (continuação)

- Supor que os elementos de um vetor sejam **inteiros**
- Critério de ordenação: ordem **crescente** ou **decrescente**
- Assim, se o vetor tem N elementos:

$VET[I] \geq VET[J]$ se $I > J \rightarrow$ **crescente**

$VET[I] \leq VET[J]$ se $I > J \rightarrow$ **decrescente**

Método da Seleção

VET = [46 15 91 59 62 76 10 93]

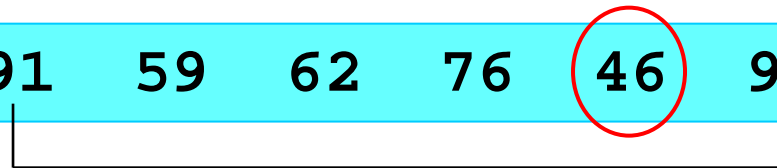
- Ordem crescente

- **1º passo:-** para saber quem fica na posição **1**,
determina-se o menor elemento do
vetor da posição **2** até a **8**

VET[**1**]

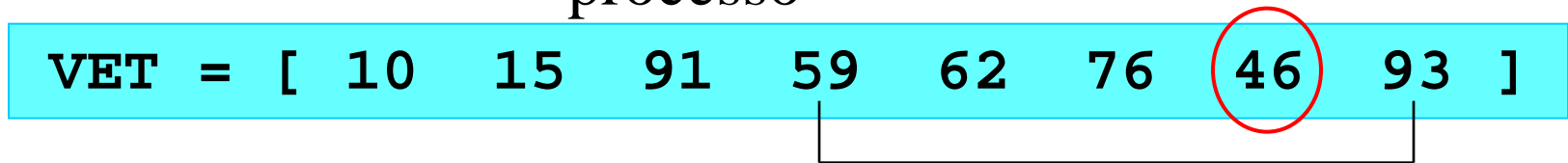
- Compara-se com o elemento
 - se for menor troca-se as posições

VET = [10 15 91 59 62 76 46 93]

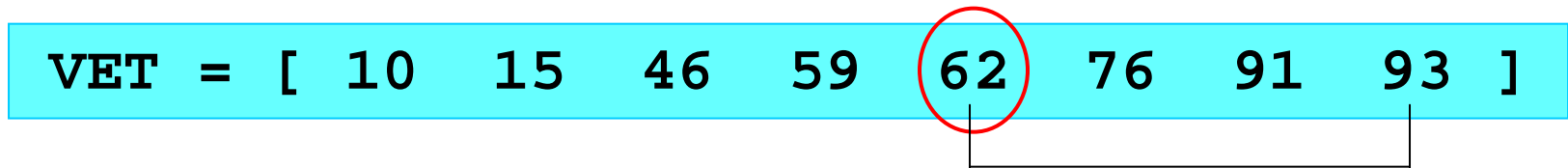


Método da Seleção (continuação)

- **2º passo:-** procura-se o menor elemento da posição **3** até a **8** e repetimos o processo



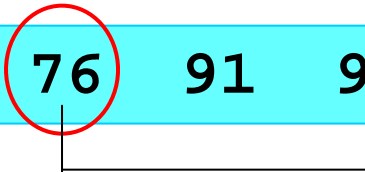
- **3º passo:**



Método da Seleção (continuação)

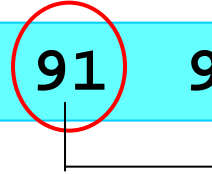
– 4º passo:

VET = [10 15 46 59 62 76 91 93]



– 5º passo:

VET = [10 15 46 59 62 76 91 93]



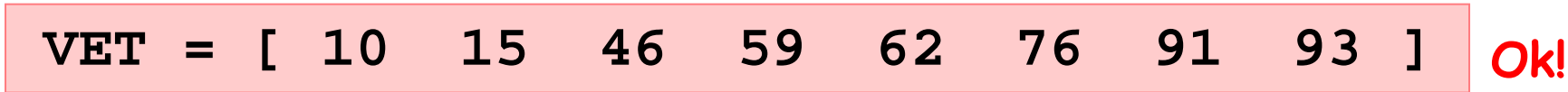
– 6º passo:

VET = [10 15 46 59 62 76 91 93]



– 7º passo:

VET = [10 15 46 59 62 76 91 93]



Algoritmo1 - Ordenação por Seleção

```
(**) program ordena1;  
    #includes  
    .....  
    int Tab[10] {tabela de pesquisa}  
    int Ind1, Ind2; {marcadores}  
    int Aux; {posição para a troca}  
    int IndMin; {posição do menor elemento}
```

Algoritmo1 - Ordenação por Seleção (continuação)

```
Int Main(){
(**)
printf("Entre com os valores da  tabela");
for (i=0;i<N;i++)
    scanf("%d", &Tab[I]);
for(Ind1=0;Ind1< N - 1;Ind1++){
    IndMin= Ind1;
    for(Ind2= IndMin + 1;Ind2 < N;Ind2++){
        if Tab[Ind2] < Tab[IndMin]
            IndMin= Ind2;
    if Tab[IndMin] < Tab[Ind1]{
        Aux = Tab[IndMin];
        Tab[IndMin]= Tab[Ind1];
        Tab[Ind1]= Aux
    }
}
printf("O vetor ordenado e\n");
for (Ind1=0;Ind1<N;Ind1++){
    printf("%d",Tab[Ind1]);
}
```

Esforço Computacional

1º passo: 7 comparações
2º passo: 6 comparações
3º passo: 5 comparações
4º passo: 4 comparações
5º passo: 3 comparações
6º passo: 2 comparações
7º passo: 1 comparação

N elementos : soma do N-1 primeiros inteiros

$$\therefore N_C = (N-1) \cdot N / 2$$

$$N_T = (N-1) \text{ \{no máximo\}}$$