LP – Linguagem de Programação I

Ordenação

Introdução

Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.

Classificação dos métodos de ordenação (Interna e Externa):

– **Ordenação interna**: arquivo a ser ordenado cabe todo na memória principal.
– **Ordenação externa**: arquivo a ser ordenado não cabe na memória principal.

Diferenças entre os métodos:

– Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado.
– Em um método de ordenação externa, os registros são acessados seqüencialmente ou em grandes blocos.

Ordenação Interna

- Medidas de complexidade levam em conta:
  - O número de comparação entre as chaves
  - O número de trocas entre os itens

- São classificados em dois tipos:
  - **Métodos Simples**: mais recomendados para conjuntos pequenos de dados. Usam mais comparações, mas produzem códigos menores e mais simples;
  - **Métodos Eficientes ou Sofisticados**: adequados para conjuntos maiores de dados. Usam menos comparações, porém produzem códigos mais complexos e com muitos detalhes.

Exemplos:

- Ordenação por Seleção (Selection Sort)
- Ordenação por Inserção (Insertion Sort)
- Ordenação por Seleção e Troca (Bubble Sort)

} Métodos Simples

- Ordenação por Inserção através de incrementos decrescentes (ShellSort)
- Ordenação por Particionamento (QuickSort)
- Ordenação de Árvores (HeapSort)

} Métodos Eficientes

1. Classificação por Troca

Na classificação por troca, a classificação de um conjunto de registros é feita através de **comparações** entre os elementos e **trocas** sucessivas desses elementos entre posições no arquivo.

1.1 Ordenação bolha ou bubble sort

O algoritmo de “ordenação bolha”, ou “bubble sort”, recebeu este nome pela imagem pitoresca usada para descrevê-lo: os elementos maiores são mais leves, e sobem como bolhas até suas posições

corretas. A idéia fundamental é fazer uma série de comparações entre os elementos do vetor. Quando dois elementos estão fora de ordem, há uma inversão e esses dois elementos são trocados de posição, ficando em ordem correta. Assim, o primeiro elemento é comparado com o segundo. Se uma inversão for encontrada, a troca é feita. Em seguida, independente se houve ou não troca após a primeira comparação, o segundo elemento é comparado com o terceiro, e, caso uma inversão seja encontrada, a troca é feita. O processo continua até que o penúltimo elemento seja comparado com o último. Com este processo, garante-se que o elemento de maior valor do vetor será levado para a última posição. A ordenação continua, posicionando o segundo maior elemento, o terceiro, etc., até que todo o vetor esteja ordenado.
Para exemplificar, vamos considerar que os elementos do vetor que queremos ordenar são valores inteiros. Assim, consideremos a ordenação do seguinte vetor:

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

Seguimos os passos indicados:

```
25 48 37 12 57 86 33 92   25x48
25 48 37 12 57 86 33 92   48x37 troca
25 37 48 12 57 86 33 92   48x12 troca
25 37 12 48 57 86 33 92   48x57
25 37 12 48 57 86 33 92   57x86
25 37 12 48 57 86 33 92   86x33 troca
25 37 12 48 57 33 86 92   86x92
25 37 12 48 57 33 86 92   final da primeira passada
```

Neste ponto, o maior elemento, 92, já está na sua posição final.

```
25 37 12 48 57 33 86 92   25x37
25 37 12 48 57 33 86 92   37x12 troca
25 12 37 48 57 33 86 92   37x48
25 12 37 48 57 33 86 92   48x57
25 12 37 48 57 33 86 92   57x33 troca
25 12 37 48 33 57 86 92   57x86
25 12 37 48 33 57 86 92   final da segunda passada
```

Neste ponto, o segundo maior elemento, 86, já está na sua posição final.

```
25 12 37 48 33 57 86 92   25x12 troca
12 25 37 48 33 57 86 92   25x37
12 25 37 48 33 57 86 92   37x48
12 25 37 48 33 57 86 92   48x33 troca
12 25 37 33 48 57 86 92   48x57
12 25 37 33 48 57 86 92   final da terceira passada
```

Idem para 57.

```
12 25 37 33 48 57 86 92   12x25
12 25 37 33 48 57 86 92   25x37
12 25 37 33 48 57 86 92   37x33 troca
12 25 33 37 48 57 86 92   37x48
12 25 33 37 48 57 86 92   final da quarta passada
```

Idem para 48.

```
12 25 33 37 48 57 86 92   12x25
12 25 33 37 48 57 86 92   25x33
12 25 33 37 48 57 86 92   33x37
12 25 33 37 48 57 86 92   final da quinta passada
```

Idem para 37.

```
12 25 33 37 48 57 86 92   12x25
12 25 33 37 48 57 86 92   25x33
12 25 33 37 48 57 86 92   final da sexta passada
```

Idem para 33.

```
12 25 33 37 48 57 86 92   12x25
12 25 33 37 48 57 86 92   final da sétima passada
```

Idem para 25 e, consequentemente, 12.

```
12 25 33 37 48 57 86 92 final da ordenação
```

A parte sabidamente já ordenada do vetor está sublinhada. Na realidade, após a troca de `37x33`, o vetor se encontra totalmente ordenado, mas esse fato não é levado em consideração por esta versão do algoritmo.

Uma função que implementa esse algoritmo é apresentada a seguir. A função recebe como parâmetros o número de elementos e o ponteiro do primeiro elemento do vetor que se deseja ordenar. Vamos considerar o ordenação de um vetor de valores inteiros.

```
/* Ordenação bolha */
void bolha (int n, int* v)
{
       int i,j;
       int temp;
       for (i=n-1; i>=1; i--)
          for (j=0; j<i; j++)
              if (v[j]>v[j+1]) { /* troca */
                     temp = v[j];
                     v[j] = v[j+1];
                     v[j+1] = temp;
              }
}

/* Testa algoritmo de ordenação bolha */
#include <stdio.h>
int main (void)
{
       int i;
       int v[8] = {25,48,37,12,57,86,33,92};
       bolha(8,v);
       printf("Vetor ordenado: ");
       for (i=0; i<8; i++)
              printf("%d ",v[i]);
       printf("\n”);
       return 0;
}
```

Para evitar que o processo continue mesmo depois de o vetor estar ordenado, podemos interromper o processo quando houver uma passagem inteira sem trocas, usando uma variante do algoritmo apresentado acima:

```
/* Ordenação bolha (2a. versão) */
void bolha2 (int n, int* v)
{
       int i, j;
       int temp;
       for (i=n-1; i>0; i--)
       {
              int troca = 0;
              for (j=0; j<i; j++)
                     if (v[j]>v[j+1])
                     { /* troca */
                            temp = v[j];
                            v[j] = v[j+1];
                            v[j+1] = temp;
                            troca = 1;
                     }
              if (troca == 0) /* nao houve troca */
                     return;
       }
```

```
}
```

A variável troca guarda o valor 0 (falso) quando uma passada do vetor (no for interno) se faz sem nenhuma troca.
O esforço computacional despendido pela ordenação de um vetor por este procedimento é fácil de se determinar, pelo número de comparações, que serve também para estimar o número máximo de trocas que podem ser realizadas. Na primeira passada, fazemos n-1 comparações; na segunda, n-2; na terceira n-3; e assim por diante. Logo, o tempo total gasto pelo algoritmo é proporcional a (n-1) + (n-2) + ... + 2 + 1. A soma desses termos é proporcional ao quadrado de n. Dizemos que o algoritmo é de ordem quadrática e representamos isso escrevendo O(n2).

Implementação recursiva

Analisando a forma com que a ordenação bolha funciona, verificamos que o algoritmo procura resolver o problema da ordenação por partes. Inicialmente, o algoritmo coloca em sua posição (no final do vetor) o maior elemento, e o problema restante é semelhante ao inicial, só que com um vetor com menos elementos, formado pelos elementos v[0],…,v[n-2].
Baseado nessa observação, é fácil implementar um algoritmo de ordenação bolha recursivamente. Embora não seja a forma mais adequada de implementarmos esse algoritmo, o estudo dessa recursão nos ajudará a entender a idéia por trás do próximo algoritmo de ordenação que veremos mais adiante.
O algoritmo recursivo de ordenação bolha posiciona o elemento de maior valor e chama, recursivamente, o algoritmo para ordenar o vetor restante, com n-1 elementos.

```
/* Ordenação bolha recursiva */
void bolha_rec (int n, int* v)
{
	int j;
	int troca = 0;
	int temp;
	for (j=0; j<n-1; j++)
		if (v[j]>v[j+1])
		{ /* troca */
			temp = v[j];
			v[j] = v[j+1];
			v[j+1] = temp;
			troca = 1;
		}
		if (troca != 0) /* houve troca */
	bolha_rec(n-1,v);
}
```

Observações

- **Método extremamente lento: só faz comparações entre posições adjacentes**
- **É o método mais ineficiente entre os simples**
  - **Melhor caso: vetor já ordenado**
  - **Pior caso: vetor de entrada em ordem reversa**
- **Cada passo aproveita muito pouco do que foi "descoberto" em relação à ordem das chaves no passo anterior (exibe informações redundantes)**

2. Classificação por Inserção

Na classificação por inserção, a classificação de um conjunto de registros é feita **inserindo** registros num subarquivo classificado anteriormente, ou seja, a inserção de um elemento é feita na posição correta dentro de um subarquivo classificado.

2.2 Inserção Direta

Dentre os métodos de inserção, o método da Inserção Direta é o mais simples, porém, é o mais rápido entre os outros métodos considerados básicos – BubbleSort e Seleção Direta. Ele deve ser utilizado para pequenos conjuntos de dados devido a sua baixa eficiência.

A principal característica deste método consiste em ordenar o arquivo utilizando um sub-arquivo ordenado localizado em seu início, e a cada novo passo, acrescentamos a este sub-arquivo mais um elemento na sua posição correta, até chegar ao último elemento do arquivo gerando um arquivo final ordenado. Exemplo:

Iniciando com um arquivo qualquer desordenado que segue:

370	250	210	330	190	230

O passo seguinte é inserir o segundo elemento ao nosso sub-arquivo ordenado. No nosso exemplo o número 250, que é copiado para uma variável auxiliar. Caminharemos pelo sub-arquivo a partir do último elemento para o primeiro. Assim poderemos encontrar a posição correta da nossa variável auxiliar dentro do sub-arquivo.

Notamos no nosso exemplo que a variável auxiliar, 250, é menor que o último elemento do nosso sub-arquivo ordenado (o nosso sub-arquivo só possui por enquanto um elemento, o número 370). O número 370 é então copiado uma posição para a direita.

370	370	210	330	190	230

Nossa variável auxiliar com o número 250, é colocada em sua posição correta no sub-arquivo ordenado.

250	370	210	330	190	230

Vale notar que nosso sub-arquivo já possui dois elementos e está ordenado. Todo esse processo é repetido para o elemento seguinte, o terceiro elemento. O próximo elemento, 210, é copiado na nossa variável auxiliar e será inserido no sub-arquivo ordenado. Então faremos a comparação da variável auxiliar com os elementos de nosso sub-arquivo, sempre a partir do último elemento para o primeiro.

Na primeira comparação notamos que a variável auxiliar é menor que o último elemento de nosso subarquivo.

Assim, copiamos este elemento para a direita e continuamos com nossas comparações.

250	370	370	330	190	230

Novamente, a nossa variável auxiliar é menor que o elemento do sub-arquivo que estamos comparando. Por isso ele deve ser copiado para a direita, abrindo espaço para que a variável auxiliar seja colocada em sua posição correta.

250	250	370	330	190	230

A variável auxiliar com o número 210 é colocada em sua posição correta.

210	250	370	330	190	230

Veja que nosso sub-arquivo ordenado possui 3 elementos. Continuaremos o processo de ordenação copiando mais uma vez o elemento imediatamente superior, 330, ao nosso sub-arquivo na variável auxiliar. Vamos, então, comparar essa variável auxiliar com os elementos do nosso sub-arquivo a partir do último elemento.

Observe que nossa variável auxiliar é menor que o elemento que está sendo comparado no nosso subarquivo.

Então ele deve ser copiado para a direita para que continuemos com nossas comparações.

210	250	370	370	190	230

Nessa comparação notamos que a variável auxiliar é maior que o elemento do sub-arquivo que estamos comparando (250<330). Portanto, encontramos a posição correta para a inserção de nossa variável auxiliar. Vamos inseri-la na sua posição correta, isto quer dizer que vamos copiá-la para o elemento imediatamente superior ao elemento que estava sendo comparado.

210	250	330	370	190	230

Note que nosso sub-arquivo possui quatro elementos ordenados. Iremos, então, repetir o processo de ordenação, copiando o próximo elemento do arquivo para uma variável auxiliar. Repetiremos as comparações de trás para frente até encontrarmos a posição correta para inserirmos o nosso elemento. Sempre que necessário, copiaremos na sua posição imediatamente à direita os elementos que forem maiores que a variável auxiliar até que se encontre um elemento menor ou até que se chegue ao início do arquivo que indicará a posição procurada. O passo-a-passo é mostrado a seguir.

210	250	330	370	190	230

Aux = 190

210	250	330	370	370	230

210	250	330	330	190	230

210	250	250	330	370	230

210	250	250	330	370	230

190	210	250	330	370	230

Mais uma vez chegamos a um sub-arquivo ordenado. Os passos seguintes mostram a última etapa de ordenação do nosso arquivo.

190	210	250	330	370	230

Aux = 230

190	210	250	330	370	370

190	210	250	330	330	370

190	210	250	250	330	370

190	210	230	250	330	370

Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

- **Vantagens:**
  - **O número mínimo de comparações e movimentos ocorre quando os itens já estão originalmente ordenados**
  - **O número máximo ocorre quando os itens estão originalmente em ordem reversa, o que indica um comportamento natural para o algoritmo**

3.Classificação por Seleção

Na classificação por seleção, a classificação de um conjunto de registros é feita através de sucessivas **seleções do menor** elemento de um arquivo ou sub-arquivo, durante o processo o menor valor encontrado é colocado na sua posição correta final no arquivo e o processo é repetido para o sub-arquivo que contém os elementos que ainda não foram selecionados.

3.1 Seleção Direta

- **Um dos algoritmos mais simples**
- **Mais recomendado para conjuntos pequenos**
- **Procedimento:**
  - **Selecione o menor item do conjunto e troque-o com o item que está na posição i**
  - **Repita essas operações com os demais itens até que reste apenas um elemento**

O metódo de Seleção Direta possui melhor desempenho que o método Bubblesort, porém, só deve ser utilizado em pequenos arquivos. Assim como o Bubblesort, esse método também é bastante usado durante o estudo de algoritmos e desenvolvimento do raciocínio lógico.

A Seleção Direta se baseia na fixação da primeira posição e na busca, por todo o arquivo, do menor elemento quando então é feita a troca dos valores. No final, teremos o menor valor (ou o maior, conforme a comparação) na primeira posição do arquivo.

Este primeiro passo nos garante que o menor elemento fique na primeira posição. Continuamos, assim, a buscar os demais elementos, comparando-os com a segunda posição do arquivo (já desconsiderando a primeira posição, que foi anteriormente ordenada em relação ao arquivo como um todo).

Dado o exemplo abaixo, vamos seguir a idéia do método Seleção Direta.

10	20	12	5	8	15

Vamos fixar a primeira posição e, utilizando uma variável auxiliar, procurar o menor elemento do arquivo e trocá-lo pelo elemento que ocupava a primeira posição.

5	20	12	10	8	15

Agora, fixaremos a segunda posição e repetiremos o processo acima.

5	8	12	10	20	15

Passaremos para a terceira posição e repetiremos o processo. Acompanhe os passos para conclusão do algoritmo.

5	8	10	12	20	15

5	8	10	12	20	15

5	8	10	12	15	20

Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

Bibliografia

Ziviane. Projeto de Algoritmos – Cap.4 Ordenação.

Rangel, José Lucas, Celes, Waldemar. Apostila de Estrutura de Dados. PUC-Rio – Curso de Engenharia. 2002.