Lógica de Programação

ORDENAÇÃO

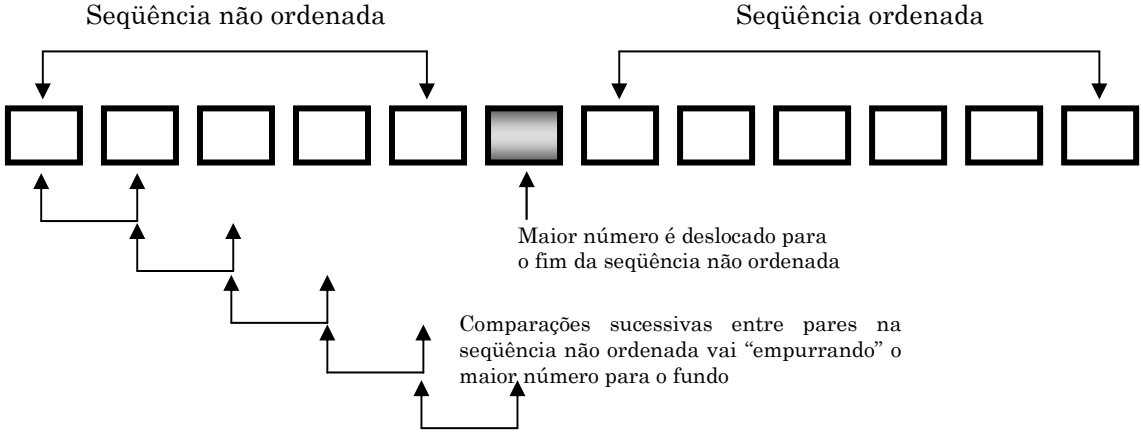
Temos como objetivo abordar algumas técnicas para construção de algoritmos para ordenação e busca de dados.

Normalmente, o usuário que vai inserir os dados não está ou não pode estar preocupado com a ordem de entrada dos dados no momento de sua inserção na relação, de modo que é comum encontrarmos elementos ordenados de maneira aleatória em nossos sistemas.

Muitas vezes necessitamos que esses dados apresentem uma ordem para que possamos realizar ações de como se determinado cliente pagou uma conta, se uma pessoa está em uma lista de convidados para uma festa e assim por diante. Devido a essas necessidades, foram desenvolvidos vários algoritmos de ordenação que consistem, basicamente, em realizar comparações sucessivas e trocar os elementos de posição. Veremos agora alguns métodos de ordenação básicos.

ORDENAÇÃO POR TROCAS – MÉTODO DA BOLHA

O método de **ordenação por trocas** é considerado o mais simples de todos. Consiste em comparar pares consecutivos de valores e permuta-los caso estejam fora de ordem. O algoritmo determina uma seqüência de comparações sistemáticas que varrem a seqüência de dados como um todo, fazendo com que o maior valor (ou menor, de acordo com a ordem desejada) acabe no final da seqüência e uma nova série de comparações sistemáticas se inicia.



| Figura 1 | O*rdenação pelo Método da Bolha*

Em cada passagem, um elemento é deslocado para sua posição final, isto é, um elemento é ordenado. Assim, uma seqüência com N elementos terá, após a primeira passagem, um elemento ordenado e $N-1$ elementos por ordenar. Na segunda passagem, a seqüência terá dois elementos ordenados e $N-2$ elementos por ordenar e assim sucessivamente.

A idéia desse tipo de ordenação é análoga à idéia de jogar pedras na água. Enquanto as pedras (elementos mais pesados) vão para o fundo, as bolhas de ar (elementos mais leves) vão para a superfície. Daí o nome do método ser conhecido como **Método da Bolha**.

EXEMPLO 1: ALGORITMO DE ORDENAÇÃO POR TROCAS – MÉTODO DA BOLHA

```
1.  Algoritmo Exemplo1
2.  Var
3.      numeros: vetor de inteiros
4.      aux, i, j: inteiro
5.  Inicio
6.      Para i de <inicio> até <fim - 1> Faça
7.          Para j de <inicio> até <fim – 1 - i> Faça
8.              Se (números[j] > números [j + 1])
9.                  aux = numeros [j]
10.                 numeros [j] = numeros [j + 1]
11.                 numeros [j + 1] = aux
12.             Fim-Se
13.         Fim-Para
14.     Fim-Para
15. Fim.
```

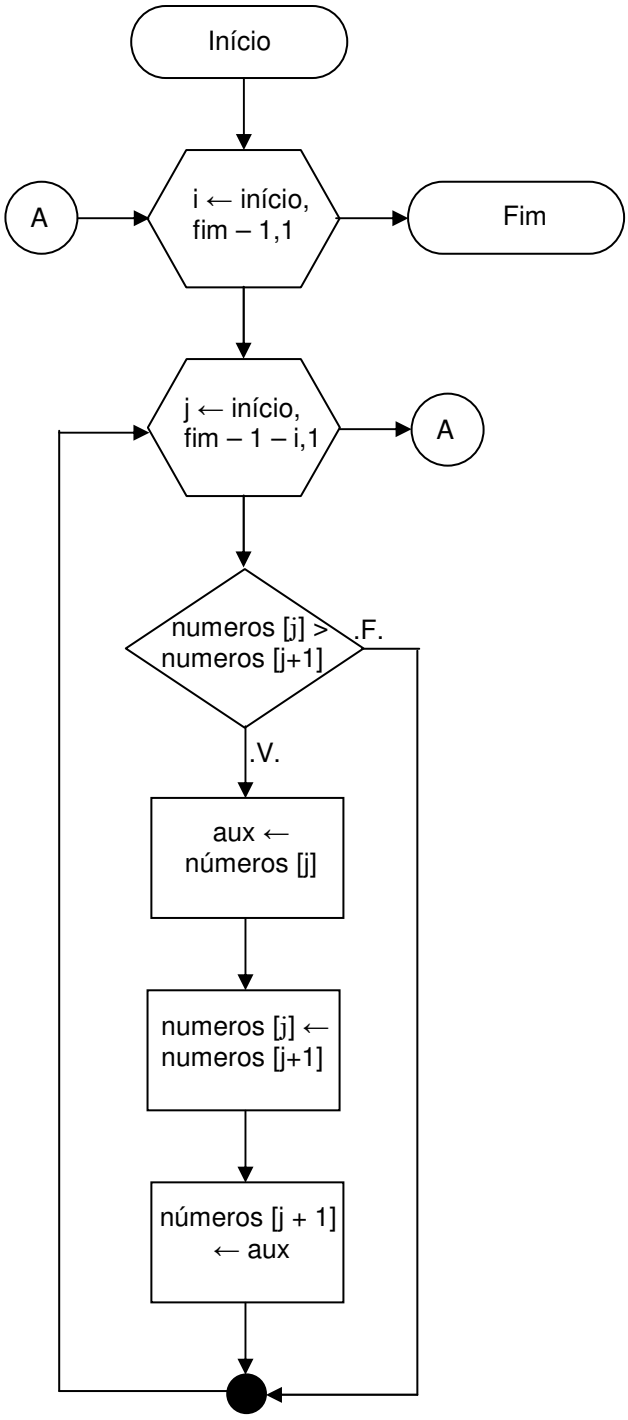
Na linha 6, a variável *i* terá o valor inicial definido pela expressão *<inicio>* e o valor final definido pela expressão *<fim – 1>*. O algoritmo faz uma série de passagens do inicio até o fim da seqüência que se deseja ordenar, também na linha 6. Como os dois últimos números serão ordenados simultaneamente (quando o penúltimo estiver ordenado, o último também estará), a estrutura de repetição irá do índice do primeiro elemento até o índice do penúltimo.

Em cada passagem, serão feitos um conjunto de comparações sucessivas (linhas8 a 12) e a troca dos valores que estiverem fora de ordem. Nessa caso, o número de comparações é determinado por uma estrutura de repetição (linha7) que vai desde o primeiro até o último elemento não ordenado, que é dado pelo índice do último elemento da seqüência a ser ordenada *(fim – 1)* menos a quantidade de elementos que já foram ordenados *(i)*.

Observe que a variável auxiliar utilizada para a permuta dos elementos deve ser do mesmo tipo que os elementos que estão sendo permutados.

Considerando que cada passagem ordena um elemento por meio de comparações sucessivas entre os elementos não ordenados, uma seqüência de *n* elementos terá *n – 1* passagens (a última passagem ordena dosi números ao mesmo tempo) e cada passagem terá *n – 1* comparações, em que *i* é o número da passagem.

FLUXOGRAMA:



Vamos considerar, para o exemplo 1, que o vetor contém os seguintes elementos: [9, 1, 3, 2, 7, 5, 4]. Então, substituindo as variáveis *<inicio> e <fim – 1>*, teríamos a seguinte representação no pseudocódigo (linhas 6 e 7) e no fluxograma:

```
Para i de 0 até 5 faça
        Para j de 0 até S – i faça
```

Na Tabela 1 são apresentados, passo a passo, os valores ordenados a cada execução do laço, considerando o vetor exemplo [9, 1, 3, 2, 7, 5, 4].

Essa tabela apresenta os valores de um vetor carregado inicialmente com os valores representados pela coluna 0. Cada coluna subseqüente representa os passos do algoritmo necessários para ordenar totalmente o vetor. As linhas nomeadas com *i* e *j* representam os valores que essas variáveis assumiriam no decorrer do processo.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
i	0	0	0	0	0	0	0	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5
j	0	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3	0	1	2	0	1	0
0	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	9	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	3	3	9	2	2	2	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	2	2	2	9	7	7	7	7	7	7	5	5	5	5	5	4	4	4	4	4	4	4
4	7	7	7	7	9	5	5	5	5	5	7	4	4	4	4	5	5	5	5	5	5	5
5	5	5	5	5	5	9	4	4	4	4	4	7	7	7	7	7	7	7	7	7	7	7
6	4	4	4	4	4	4	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

|Tabela 1| O*rdenação de um vetor*

É interessante observar que, nesse caso, o algoritmo já concluiu a ordenação no 15º passo, o que não ocorreria com outra ordem de entrada dos valores (coluna 0). Esse método de tabular os dados pode ser utilizado para outros exemplos, facilitando a compreensão e a verificação da eficácia do algoritmo. Não utilizaremos essa representação para todos os casos, cabendo ao leitor aplicá-la quando julgar necessário.