Estrutura de Ordenação

Quando desenvolvemos um algoritmo ou programa estruturado, muitas vezes precisamos ordenar as informações de diferentes tipos de dados que podemos utilizar no programa. Para isso, podemos utilizar a estrutura de dados chamada Ordenação para ordenar essas informações no programa.

A estrutura de dados de ordenação é bastante utilizada quando tem como prioridade realizar comparações sucessivas e trocar os elementos de posição. Por exemplo, às vezes, precisamos que as informações estejam em determinada ordem para que possamos realizar ações como verificar se determinado CPF está na lista dos nomes com apresentarem cheque sem fundo ou se determinada pessoa está numa lista de convidados de uma festa de casamento.

Há diferentes estruturas de ordenação utilizados na computação, por exemplo: ordenação por inserção, ordenação por seleção, ordenação por bolhas, ordenação de shell, ordenação de heap, ordenação rápida – quicksort, ordenação por fusão – mergesort, ordenação de raiz, etc.

A ordenação é uma estrutura de dados muito utilizada na ciência da computação para a programação. Neste capítulo, veremos como e quando trabalhar com ordenação, bem como, inserir, remover, consultar e manipular dados na ordenação.

Definição de ordenação

Em diversas aplicações, os elementos precisam ser armazenados em determinada ordem. Ordenar significa rearranjar um conjunto de elementos numa disposição crescente ou decrescente e o principal objetivo da ordenação é facilitar a busca de determinado elemento do conjunto ordenado.

Para exemplificar a importância da ordenação, podemos pensar em como seria difícil consultar uma lista telefônica de uma cidade, se as informações contidas nesta lista não estivessem em ordem alfabética pelo sobrenome das pessoas. Outro exemplo seria a de procurar uma palavra num dicionário de idiomas, se as palavras nesse dicionário não estivessem ordenadas.

Alguns algoritmos podem desenvolver a ordenação dos seus elementos para garantir um melhor desempenho computacional e isso pode ser feito de duas formas: ordenar os elementos na estrutura de dados no momento em que os elementos são inseridos ou ordenar os elementos, a partir de um algoritmo, numa estrutura de dados com elementos já inseridos. Pensando na segunda forma, desenvolveremos, neste capítulo, algoritmo que ordenam elementos já inseridos numa estrutura de dados, mas que garantam a eficiência da ordenação em tempo reduzido.

Algoritmos de Ordenação

Nesta seção, estudaremos alguns algoritmos de ordenação conhecidos e qual a idéia ou técnica utilizada para realizar a ordenação.

Ordenação bolha ou bubble sort

O algoritmo ordenação bolha ou *bubble sort* é assim conhecido pois utiliza a idéia de que os elementos maiores são mais leves e sobem como bolhas até suas posições corretas. Ele é um dos algoritmos mais conhecidos na computação, pois é fácil de ser entendido e utilizado. No entanto, pode ser considerado um dos algoritmos menos eficientes para ordenação.

A técnica consiste em realizar comparações entre os elementos dois-a-dois e quando um elemento fora de ordem é detectado, ocorre a troca desses elementos. Assim, o primeiro elemento é comparado com o segundo, se os elementos estiverem fora de ordem, a troca desses elementos é realizada. Em seguida, o segundo elemento é comparado com o terceiro elemento, se os elementos estiverem fora de ordem, a troca é realizada e assim o processo continua até que todos os elementos tenham sido comparados. Esse processo garante que o último elemento é o maior dentre todos os elementos.

O processo continua, considerando agora todos os elementos menos o último, que já está em sua posição correta. Depois que o penúltimo elemento já está na posição correta, o algoritmo continua até que o primeiro elemento esteja na posição correta, ou seja, seja o menor elemento dentre todos os elementos.

Para exemplificar, vamos considerar os valores inteiros relacionados a seguir:

26	47	38	11	95

Seguindo o algoritmo explicado acima, a troca dos elementos segue as iterações seguintes:

26	47	38	11	95	compara 26 e 47 – não efetua a troca
26	47	38	11	95	compara 47 e 38 – realiza a troca
26	38	47	11	95	compara 47 e 11 – realiza a troca
26	38	11	47	95	compara 47 e 95 – não realiza a troca
26	38	11	47	**95**	final do primeiro processo

Ao final dessa primeira etapa, o maior elemento, **95**, já está em sua posição correta. Vamos continuar o processo:

26	38	11	47	**95**	compara 26 e 38 – não efetua a troca
26	38	11	47	**95**	compara 38 e 11 – efetua a troca
26	11	38	47	**95**	compara 38 e 47 – não efetua a troca
26	11	38	**47**	**95**	final do segundo processo

Ao final dessa segunda etapa, o segundo maior elemento, **47**, já está em sua posição correta. Vamos continuar o processo:

26	11	38	47	**95**	compara 26 e 11 – efetua a troca
11	26	38	47	**95**	compara 26 e 38 – não efetua a troca
11	26	**38**	**47**	**95**	final do terceiro processo

Ao final dessa terceira etapa, o terceiro maior elemento, **38**, já está em sua posição correta. Vamos continuar o processo:

11	26	**38**	**47**	**95**	compara 11 e 26 – não efetua a troca
11	**26**	**38**	**47**	**95**	final do último processo

Ao final dessa última etapa, o primeiro e o segundo elementos, **11** e **26**, respectivamente, já estão em suas posições corretas. Assim, ao final do processo inteiro, os números inteiros estão todos ordenados.

Ordenação por troca de partição ou quicksort

O algoritmo ordenação por troca de partição ou *quicksort* é assim conhecido pois utiliza a idéia de ordenar os elementos por partições.

A técnica consiste em dividir uma lista de elementos em sublistas, que são conhecidas como partições. Essas partições são novamente subdivididas em outras partições até que a subdivisão não seja mais necessária, ou seja, quanto a partição tem somente um elemento. A intenção é dividir o problema em subproblemas que possam ser resolvidos com mais facilidade e rapidamente.

Para determinar as partições, um elemento chamado **pivô** é escolhido dentro da lista, de forma que, os elementos da primeira partição são menores ou iguais ao elemento **pivô** e todos os elementos da segunda partição são maiores ou iguais ao elemento **pivô**.

Para exemplificar, vamos considerar os valores inteiros relacionados a seguir:

25 57 48 37 12 92 86 33

agora, vamos colocar o número 25 em sua posição correta. Assim, teremos o seguinte resultado:

12 (**25**) 57 48 37 92 86 33 pivô 25

Isso significa que todos os elementos antes do pivô 25 são menores ou iguais a 25, no nosso exemplo, temos a lista (12), e todos os elementos depois do pivô 25 são maiores ou iguais a 25, no nosso exemplo, temos a lista (57 48 37 92 86 33). Como o pivô 25 está em sua posição correta, agora temos a nossa lista dividida em duas sublistas.

Como a primeira sublista tem um único elemento, o 12, ela já está ordenada, ou seja, já está em sua posição correta, então não há nada para fazer com ela. Por isso, devemos ordenar somente a segunda sublista e, para isso, seguiremos o mesmo processo. Vejamos como está a lista:

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

agora, vamos colocar o número 57 em sua posição correta. Assim, teremos o seguinte resultado:

12 **25** (48 37 33) **57** (92 86) pivô 57

seguindo, passo a passo, o mesmo processo, temos as seguintes seqüências:

12 **25** (37 33) **48** **57** (92 86) pivô 48
12 **25** (33) **37** **48** **57** (92 86) pivô 37
12 **25** **33** **37** **48** **57** (92 86) pivô 33
12 **25** **33** **37** **48** **57** (86) **92** pivô 92
12 **25** **33** **37** **48** **57** **86** **92** pivô 86

Depois que o processo do *quicksort* foi aplicado em todas as sublistas, temos então a lista ordenada.

Ordenação por intercalação ou mergesort

O algoritmo ordenação por intercalação ou *mergesort* é assim conhecido pois utiliza a idéia de ordenar os elementos a partir da união de elementos já ordenados.

A técnica consiste em unir metades ordenadas de duas sublistas de elementos, para gerar a lista ordenada dos elementos. No entanto, essas sublistas de elementos precisam ser ordenadas primeiro, ou seja, elas foram geradas de sublistas de elementos ordenados dessas sublistas. Essa técnica de dividir listas em duas sublistas pára quando a sublista tem menos de dois elementos.

Em uma lista de elementos, a primeira sublista tem a primeira metade da lista de elementos e a segunda sublista tem a segunda metade da lista de elementos.

Para exemplificar, vamos considerar os valores inteiros relacionados a seguir:

25 57 48 37 12 92 86 33

Agora vamos dividir a lista de elementos, na metade, em duas sublistas:

(25 57 48 37) (12 92 86 33)

Vamos repetir a divisão das sublistas, de forma que cada sublista seja dividida, na metade, em outras duas sublistas, assim, teremos:

(25 57) (48 37) (12 92) (86 33)

repetindo o processo, teremos:

(25) (57) (48) (37) (12) (92) (86) (33)

Agora, vamos intercalar as listas duas a duas, ordenando os elementos parciais das sublistas, resultando nas seguintes sublistas:

(25 57) (37 48) (12 92) (33 86)

Vamos repetir o processo de intercalar as listas duas a duas, ordenando os elementos parciais das sublistas, resultando nas seguintes sublistas:

(25 37 48 57) (12 33 86 92)

E, intercalando as duas últimas sublistas, teremos;

(12 25 33 37 48 57 86 92)

Depois de intercalas as últimas duas sublistas, temos a lista com os elementos ordenados.