UEA – Universidade do Estado do Amazonas

Métodos de ordenação de dados internos

Manaus – AM

2019

Felipe da Silva Braga
Ícaro Santos Pereira
Yasser Schuck Antonio Tuma

Métodos de Ordenação de dados internos
Bubble sort* e *Selection sort

Trabalho apresentado ao curso de Algoritmo e Estrutura de Dados II como requisito para obtenção de nota.

Orientador : Dr. Sergio Cleger Tamayo.

Manaus – AM
2019

Sumário

Introdução

Bubble sort é um algoritmo questionavelmente eficiente pra ordenar listas, por ser simples, é um dos primeiros a serem ensinados em cursos de computação, visto que ajuda a criar uma noção intuitiva do que é ordenação.

Selection sort é outro exemplo de algoritmo de ordenação que por sua vez funciona como uma reordenação de valores por seleção.

Ordenação é um conceito simples, colocar coisas em ordem, porém, é um conceito básico que se estende dos programas mais simples aos mais complexos. Velocidade é extremamente importante a se considerar quando a quantidade de dados começa a aumentar, visto que eficiência é primordial para qualquer software. Veremos a seguir como *Bubble sort* e *Selection sort* se encaixam em relação as suas complexidades, sua eficiência, sua relação com outros algoritmos entre outros fatores que denotam sua importância no aprendizado de áreas da computação.

2.1 Como e onde usar

- *Bubble Sort*

Para começarmos a definir como e onde usar cada algoritmo primeiro veremos uma breve definição sobre o mesmo:

Bubble Sort possui esse nome como uma referência ao fundo do oceano, a ideia do mesmo é que se visualizem os dados como bolhas, que a cada passagem se fazem flutuar as maiores para o topo da sequência de valores. Tal ordenação pode ser mais bem entendida se a matriz a ser ordenada for vista como uma coluna vertical cujos menores elementos estão no topo e os maiores na base. A matriz é varrida da base para cima e dois elementos adjacentes são intercambiados caso sejam encontrados fora de ordem um com relação ao outro.

Os usos desse algoritmo são limitadíssimos além do aprendizado do que é ordenação, visto que ele se torna vastamente inferior a outras formas de algoritmo de ordenação.

- *Selection Sort*

Assim como o Bubble Sort, o Selection Sort precisa percorrer toda a lista para cada elemento que ela possui, podendo inclusive fazê-lo mais de uma vez. Esse algoritmo pode ser especialmente utilizado em dados organizados sem a ajuda de uma outra estrutura de dados auxiliar, e pela sua simplicidade consegue ter vantagens de performance sobre algoritmos mais complicados nesses mesmos ambientes onde a memória auxiliar é limitada.

2.2 Tipo de ordenação

- Bubble Sort

Bubble sort é um algoritmo de ordenação completa, ou seja, retorna todas as suas variáveis em ordem, visto que todo o vetor é percorrido por completo variável por variável, não deixando que hajam frações não analisadas, partes eliminadas ou variáveis não acessadas.

- Selection Sort

Selection Sort também é um algoritmo de ordenação completa, visto que eventualmente todas as suas variáveis serão acessadas como a menor do conjunto, até a última, resultando em uma total análise do conjunto de dados.

2.3 Formas de ordenação

- Bubble Sort

Bubble sort ordena de forma que as variáveis fora de posição são trocadas pelas de posição anterior, assim analisadas um por um até que se perceba que nenhuma está fora de ordem, ou seja, funcionam por forma de *Exchange*, troca ou intercâmbio.

- Selection Sort

Selection Sort, como diz o nome funciona por forma de seleção, onde são escolhidos os menores valores de um conjunto de variáveis e os mesmos são arranjados no começo de cada vetor, de forma que a próxima varredura ignore o valor que acabou de ser realocado.

2.4 Complexidade

-Bubble Sort

O algoritmo realizará n-1 trocas para o primeiro passo, depois n-2 trocas para o segundo elemento e assim sucessivamente. Trocas = n-1+n-2+n-3+...+2+1 aproximadamente N^2 trocas.

$C(n) = O(n^2)$

-Selection Sort

O algoritmo leva em consideração 3 quesitos:

1. O tempo de execução para todas as chamadas do indiceMenor
2. O tempo de execução para todas as chamas de troca

3. O tempo de execução para a fazer a busca.

Os quesitos 2 e 3 são O(n) no pior dos casos, mas o 1 quesito tem uma interação de 1+2+...+n-1+n, é uma série aritmética e ela resulta em $(n^2)/2 + (n/2)$ o que resulta em $O(n^2)$ na notação do Big O. Como é uma análise completa do algoritmo, prevalece o maior tempo.

$C(n)=O(n^2)$

2.4.1 Complexidade de pior caso

- Bubble sort

A complexidade do Bubble sort de acordo com a escala de complexidade do tempo para pior caso é $O(n^2)$, ou seja, quadrática.

- Selection sort

A complexidade do Selection sort de acordo com a escala de complexidade do tempo para pior caso é $O(n^2)$.

2.4.2 Complexidade de caso médio

- Bubble sort

A complexidade do Bubble sort de acordo com a escala de complexidade do tempo para pior caso é $O(n^2)$.

- Selection sort

A complexidade do Selection sort de acordo com a escala de complexidade do tempo para caso médio é $O(n^2)$.

2.4.3 Complexidade de melhor caso

- Bubble sort

A complexidade do Bubble sort de acordo com a escala de complexidade do tempo para melhor caso é O(n), ou seja, linear.

- Selection sort

A complexidade do Selection sort de acordo com a escala de complexidade do tempo para melhor caso é $O(n^2)$.

2.5 Estabilidade

- Bubble Sort

O algoritmo é estável, visto que dois elementos similares na entrada ainda estarão em sequência na saída, visto que o mesmo trabalha de forma linear. Ao serem ordenados, os elementos seguem subindo, sem precisar ordenar elementos iguais pois eventualmente eles ficarão um ao lado do outro na ordem em que vieram.

- Selection Sort

O algoritmo não é estável, visto que não apenas reposiciona, mas substitui o menor pelo elemento antigo que estava onde o menor estava, gerando inconsistência no posicionamento de elementos similares.

2.6 Vantagens e desvantagens

- Bubble Sort

Vantagens:

- Fácil de implementar
- Fácil de entender
- É mais eficiente quando o vetor já está ordenado
- Não precisa de memória externa

Desvantagens:

- Extremamente lento, especialmente conforme o número de elementos aumenta.

- Selection Sort

Vantagens:

- Simples de implementar e entender
- Funciona muito bem em pequena quantidade de elementos aleatória

Desvantagens:

- Extremamente custoso, $O(n^2)$

2.7 Exemplos de aplicações reais

Bubble Sort e Selection Sort são bons para aplicações que não requerem velocidade ou que não trabalhem com muitos dados. Ambos não necessitam de um vetor auxiliar, assim ocupando menos memória. Como vimos no tópico 2.4 que fala sobre Complexidade, tanto no melhor quanto no pior dos casos os dois algoritmos se comportam com uma complexidade $O(n^2)$ o que é muito lento, comparado com os demais algoritmos de ordenação, por isso existem poucas ou quase nenhuma aplicações reais que utilizem o Bubble ou Selection Sort.

2.8 Código comentado

2.8.1 Algoritmo Bubble Sort

```
void bubbleSort( int vetor[], int tam){

    int trocou=1, novoTam=tam-1, guarda=tam, lacoInterno, aux;

    while(trocou == 1){
        trocou=0;  //flag
        lacoInterno=0;
        while(lacoInterno < novoTam){
        if( vetor[lacoInterno] > vetor[lacoInterno+1]){
            aux=vetor[lacoInterno];  //inicio troca
            vetor[lacoInterno]=vetor[lacoInterno+1];
```

```
                    vetor[lacoInterno+1]=aux; // fim da troca
                        trocou=1;
                    guarda=lacoInterno;  //último elemento trocado
        }
        lacoInterno=lacoInterno+1;
            }
        novoTam=guarda;
        }
}
```

Para exemplificar, vamos considerar que os elementos do vetor que queremos ordenar são valores inteiros. Assim, consideremos a ordenação do seguinte vetor.

[78, 85, 84, 21, 7, 92, 52, 6, 74, 16]

Seguimos os passos indicados:

78 85 84 21 7 92 52 6 74 16	78x85
78 85 84 21 7 92 52 6 74 16	85x84 troca
78 84 85 21 7 92 52 6 74 16	85x21 troca
78 84 21 85 7 92 52 6 74 16	85x7 troca
78 84 21 7 85 92 52 6 74 16	85x92
78 84 21 7 85 92 52 6 74 16	92x52 troca
78 84 21 7 85 52 92 6 74 16	92x6 troca
78 84 21 7 85 52 6 92 74 16	92x74 troca
78 84 21 7 85 52 6 74 92 16	92x16 troca
78 84 21 7 85 52 6 74 16 92	final da primeira passada

Neste ponto, o maior elemento, 92, já está na sua posição final. E como houve troca, o algoritmo vai reiniciar o processo, entretanto, iremos diminuir o tamanho da lista. Visto que o último elemento já está no seu devido lugar e não existe a necessidade de verificar a sua posição novamente.

78 84 21 7 85 52 6 74 16 92	78x84
78 84 21 7 85 52 6 74 16 92	84x21 troca
78 21 84 7 85 52 6 74 16 92	84x7 troca
78 21 7 84 85 52 6 74 16 92	84x85
78 21 7 84 85 52 6 74 16 92	85x52 troca
78 21 7 84 52 85 6 74 16 92	85x6 troca
78 21 7 84 52 6 85 74 16 92	85x74 troca
78 21 7 84 52 6 74 85 16 92	85x16 troca
78 21 7 84 52 6 74 16 85 92	final da segunda passada

Neste ponto, o segundo maior elemento, 85, já está na sua posição final.

78 21 7 84 52 6 74 16 85 92	78x21 troca
21 78 7 84 52 6 74 16 85 92	78x7 troca
21 7 78 84 52 6 74 16 85 92	78x84
21 78 7 84 52 6 74 16 85 92	84x52 troca
21 78 7 52 84 6 74 16 85 92	84x6 troca
21 78 7 52 6 84 74 16 85 92	84x74 troca
21 78 7 52 6 74 84 16 85 92	84x16 troca
21 78 7 52 6 74 16 84 85 92	final da terceira passada

Idem para 84.

21 78 7 52 6 74 16 84 85 92	21x78
21 78 7 52 6 74 16 84 85 92	78x7 troca
21 7 78 52 6 74 16 84 85 92	78x52 troca
21 7 52 78 6 74 16 84 85 92	78x6 troca
21 7 52 6 78 74 16 84 85 92	78x74 troca
21 7 52 6 74 78 16 84 85 92	78x16 troca
21 7 52 6 74 16 78 84 85 92	final da quarta passada

Idem para 78.

21 7 52 6 74 16 78 84 85 92	21x7 troca
7 21 52 6 74 16 78 84 85 92	21x56
7 21 52 6 74 16 78 84 85 92	52x6 troca
7 21 6 52 74 16 78 84 85 92	52x74
7 21 6 52 74 16 78 84 85 92	74x16 troca
7 21 6 52 16 74 78 84 85 92	final da quinta passada

Idem para 74.

7 21 6 52 16 74 78 84 85 92	7x21
7 21 6 52 16 74 78 84 85 92	21x6 troca
7 6 21 52 16 74 78 84 85 92	21x52
7 6 21 52 16 74 78 84 85 92	52x16 troca
7 6 21 16 52 74 78 84 85 92	final da sexta passada

Idem para 52.

7 6 21 16 52 74 78 84 85 92	7x6 troca
6 7 21 16 52 74 78 84 85 92	7x21
6 7 21 16 52 74 78 84 85 92	21x16 troca
6 7 16 21 52 74 78 84 85 92	final da sétima passada

Idem para 21.

6 7 16 21 52 74 78 84 85 92	6x7
6 7 16 21 52 74 78 84 85 92	7x16
6 7 16 21 52 74 78 84 85 92	final da oitava passada

Como não houve nenhuma troca, a variável troca vai permanecer com o valor 0. Portanto o programa sairá do Loop, e consequentemente do algoritmo *Bubble Sort*. O vetor se encontra totalmente ordenado:

[6, 7, 16, 21, 52, 74, 78, 84, 85, 92]

2.8.2 Algoritmo Select Sort

```
void selecSort( int vetor[ ], int tam){

    int lacoExterno, lacoInterno, indiceMenor, aux;

    for(lacoExterno=0; lacoExterno < tam-1; lacoExterno++){
        indiceMenor=lacoExterno;
        for(lacoInterno=lacoExterno+1;  lacoInterno<tam;  lacoInterno++){
//Pesquisar linearmente o índice do menor elemento
            if(vetor[lacoInter] < vetor[indiceMenor]){
                indiceMenor=lacoInterno; //atualiza o menor elemento
            }
        }//processo de troca
        aux=vetor[lacoExterno];
        vetor[lacoExterno]=vetor[indiceMenor];
        vetor[indiceMenor]=aux; //fim processo de troca
    }
}
```

A ideia do algoritmo é:

- Selecione o menor elemento do conjunto
- Troque com o primeiro elemento

A seguir, repita essas duas operações com os n-1 elementos restantes, depois com os n-2 elementos, até que reste apenas um elemento. Para exemplificar, vamos considerar que os elementos do vetor que queremos ordenar são valores inteiros. Assim, consideremos a ordenação do seguinte vetor.

[78, 85, 84, 21, 7, 92, 52, 6, 74, 16]

Seguimos os passos indicados, assuma "menor" como uma variável que guarda os valores do índice do menor elemento na lista:

	menor recebe 1, primeira passada
78 85 84 21 7 92 52 6 74 16	78x85
78 85 84 21 7 92 52 6 74 16	78x84
78 85 84 21 7 92 52 6 74 16	78x21 menor recebe 4
78 85 84 **21** 7 92 52 6 74 16	21x7 menor recebe 5
78 85 84 21 **7** 92 52 6 74 16	7x92
78 85 84 21 **7** 92 52 6 74 16	7x52
78 85 84 21 **7** 92 52 6 74 16	7x6 menor recebe 8
78 85 84 21 7 92 52 **6** 74 16	6x7
78 85 84 21 7 92 52 **6** 74 16	6x16
78 85 84 21 7 92 52 **6** 74 16	final da primeira passada, troca.
<u>6</u> 85 84 21 7 92 52 78 74 16	

Como observamos, só trocamos no final da passada. O Select sort lida com índices, como foi a primeira passada iremos trocar o valor inicial, pelo o valor que a variável menor aponta. Troca o elemento da posição 1 com o elemento da posição 8 (o último valor associado a variável menor). Neste ponto, o menor número, 6, já está na sua posição final.

	menor recebe 2, segunda passada
<u>6</u> **85** 84 21 7 92 52 78 74 16	85x84
<u>6</u> **85** 84 21 7 92 52 78 74 16	85x21 menor recebe 4

6 85 84 **21** 7 92 52 78 74 16	21x7 menor recebe 5
6 85 84 21 **7** 92 52 78 74 16	7x92
6 85 84 21 **7** 92 52 78 74 16	7x52
6 85 84 21 **7** 92 52 78 74 16	7x78
6 85 84 21 **7** 92 52 78 74 16	7x74
6 85 84 21 **7** 92 52 78 74 16	7x16
6 85 84 21 **7** 92 52 78 74 16	final da segunda passada, troca
6 7 84 21 85 92 52 78 74 16	

Neste ponto, o menor número, 7, já está na sua posição final.

	menor recebe 3, terceira passada
6 7 **84** 21 85 92 52 78 74 16	84x21 menor recebe 4
6 7 84 **21** 85 92 52 78 74 16	21x85
6 7 84 **21** 85 92 52 78 74 16	21x92
6 7 84 **21** 85 92 52 78 74 16	21x52
6 7 84 **21** 85 92 52 78 74 16	21x78
6 7 84 **21** 85 92 52 78 74 16	21x74
6 7 84 **21** 85 92 52 78 74 16	21x16 menor recebe 10
6 7 84 21 85 92 52 78 74 **16**	final da terceira passada, troca
6 7 16 21 85 92 52 78 74 84	

Idem para 16.

	menor recebe 4, quarta passada
6 7 16 **21** 85 92 52 78 74 84	21x85
6 7 16 **21** 85 92 52 78 74 84	21x92
6 7 16 **21** 85 92 52 78 74 84	21x52
6 7 16 **21** 85 92 52 78 74 84	21x78
6 7 16 **21** 85 92 52 78 74 84	21x74
6 7 16 **21** 85 92 52 78 74 84	21x84
6 7 16 **21** 85 92 52 78 74 84	final da quarta passada, troca

6 7 16 21 85 92 52 78 74 84

Idem para 21.

menor recebe 5, quinta passada

6 7 16 21 **85** 92 52 78 74 84	85x92
6 7 16 21 **85** 92 52 78 74 84	85x52 menor recebe 7
6 7 16 21 85 92 **52** 78 74 84	52x78
6 7 16 21 85 92 **52** 78 74 84	52x74
6 7 16 21 85 92 **52** 78 74 84	52x84
6 7 16 21 85 92 **52** 78 74 84	final da quinta passada, troca
6 7 16 21 52 92 85 78 74 84	

Idem para 52.

	menor recebe 6, sexta passada
6 7 16 21 52 **92** 85 78 74 84	92x85 menor recebe 7
6 7 16 21 52 92 **85** 78 74 84	85x78 menor recebe 8
6 7 16 21 52 92 85 **78** 74 84	78x74 menor recebe 9
6 7 16 21 52 92 85 78 **74** 84	74x84
6 7 16 21 52 92 85 78 **74** 84	final da sexta passada, troca
6 7 16 21 52 74 85 78 92 84	

Idem para 74.

	menor recebe 7, sétima passada
6 7 16 21 52 74 **85** 78 92 84	85x78 menor recebe 8
6 7 16 21 52 74 85 **78** 92 84	78x92
6 7 16 21 52 74 85 **78** 92 84	78x84
6 7 16 21 52 74 85 **78** 92 84	final da sétima passada, troca
6 7 16 21 52 74 78 85 92 84	

Idem para 78.

	menor recebe 8, oitava passada
6 7 16 21 52 74 78 **85** 92 84	85x92
6 7 16 21 52 74 78 **85** 92 84	85x84 menor recebe 10
6 7 16 21 52 74 78 85 92 **84**	final da oitava passada, troca
6 7 16 21 52 74 78 84 92 85	

Idem para 84.

	menor recebe 9, nona passada
6 7 16 21 52 74 78 84 **92** 85	92x85 menor recebe 10
6 7 16 21 52 74 78 84 92 **85**	final da nona passada, troca
6 7 16 21 52 74 78 84 85 92	

Idem para 85 e, consequentemente para o 92. O vetor se encontra totalmente ordenado:

[6, 7, 16, 21, 52, 74, 78, 84, 85, 92]

2.9 Avaliação com 100000 números

Foram testados 5 ordens de lista para fazer essa avaliação:

Ordem 1: lista ordenada em ordem decrescente;

Algoritmo	**Tempo**
BubbleSort	32,550s
SelectSort	20,596s

Ordem 2: lista ordenada em ordem crescente;

Algoritmo	**Tempo**
BubbleSort	0,210s
SelectSort	12,621s

Ordem 3: primeira metade da lista em ordem crescente e a segunda metade em ordem decrescente;

Algoritmo	**Tempo**
BubbleSort	14,267s
SelectSort	14,550s

Ordem 4: primeira metade da lista em ordem decrescente e a segunda metade em ordem crescente;

Algoritmo	**Tempo**
BubbleSort	14,310s
SelectSort	14,505s

Ordem 5: lista totalmente desordenada.

Algoritmo	**Tempo**
BubbleSort	39,785s

SelectSort	12,623s

Como podemos ver, os dois algoritmos são muito parelhos, mas o algoritmo Bubble Sort mostrou uma vantagem considerável na avaliação de Ordem 2. Já o Select Sort obteve grande êxito na Ordem 5, lista desordenada.

Referências Bibliográficas

SOUZA, Jairo. Método Bubble Sort**.** Minas Gerais, Brasil. Disponível em: < http://www.ufjf.br/jairo_souza/files/2009/12/2-Ordenação-BubbleSort.pdf>, Acesso em: 17 agosto. 2019, 00:13:30.

DROZDEK, Adam. Estrutura de dados e algoritmos em C++. 3. ed. rev. Pensilvania: EUA, 2005.

LAFORE, Robert. Estrutura de dados e algoritmos em Java. 2. ed. rev. Califórnia: EUA, 2003.

TENEMBAUM, LANGSAM, AUGENSTEIN. Estrutura de dados usando C. Nova Jersey: EUA, 1985.
Disponível em: <https://www.cin.ufpe.br/~garme/public/(ebook)Estruturas%20de%20Dados%20Usando%20C%20(Tenenbaum).pdf >, Acesso em: 17 agosto. 2019, 02:27:00.

Cid Carvalho de Souza. Complexidade de Algoritmos I. São Paulo, Brasil. Disponível em: < http://www.ic.unicamp.br/~zanoni/mo417/2011/aulas/handout/04-ordenacao.pdf>, Acesso em: 17 agosto. 2019, 03:59:03.