RELATÓRIO SOBRE ALGORITMOS DE ORDENAÇÃO DE DADOS

MICHELINO, Ananda Natalia[1]
PESSOA, Lucas Fernandes[2]

Resumo

O presente trabalho tem por objetivo traçar uma comparação entre diferentes algoritmos para ordenação de dados, exemplificando diferenças de desempenho em alguns testes realizados, vantagens, desvantagens, complexidade encontrada pelo grupo na resolução da lógica de programação etc.

Palavras-chave: Algoritmo; Estrutura de dados; Javascript; Ordenação de dados.

Tipos de Ordenação de Dados

Na presente análise foram testados 4 tipos de ordenação de dados: Bubblesort, Seleção Direta, Inserção Direta e Quicksort.

O método Bubblesort foi o primeiro a ser implementado e tem por pressuposto a realização de trocas de valores a cada iteração, para tanto é utilizada uma variável auxiliar que guarda o valor a ser trocado, inserindo na ordem estabelecida (seja crescente ou decrescente) quando a condição é satisfeita.

O segundo método a ser implementado foi a seleção direta. Tal método, diferente do anterior, não faz trocas a cada iteração, mas somente preserva a posição do valor dentro do vetor que mudará de posição somente ao final da iteração, uma única vez.

[1] Graduanda em Análise e Desenvolviento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: ananda.anm@gmail.com.
[2] Graduando em Análise e Desenvolviento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico:lucasdepessoa@gmail.com.

A inserção direta foi o terceiro método a ser implementado e muito se assemelhou à seleção direta, porém diferentemente deste, na inserção direta é calculada a cada iteração quantas posições o valor precisaria mover-se para ocupar seu lugar na ordenação no vetor.

Por fim, utilizamos o método quicksort, que não foi implementado pelos alunos, dada sua complexidade, mas pressupõe que dividir o vetor ao meio, localizando o pivô e a partir dele realizar comparações compartimentadas, reduzindo a quantidade de trocas e de iterações. A partir do pivô, são comparados, à direita, os valores que são maiores que ele e, à esquerda, os valores que são menores que ele.

Esta pressupõe a primeira fase, a segunda fase é a recursão.

No quicksort, o código é dividido, portanto, em duas fases: uma de partição do vetor e posterior ordenação e outra de recursão, onde são repassados os valores de array, posição direita e esquerda como parâmetros dentro da própria função.

A cada implementação dos métodos observamos que houve a presença de variações de tempo uma vez que os vetores foram gerados aleatoriamente, utilizando o método Math.Random, presente por padrão nas bibliotecas javascript.

Após fazer a geração aleatória dos vetores, fizemos uma cópia de cada vetor com uma determinada quantidade de elementos para que as comparações entre cada método fossem fidedignas e tivessem os mesmos parâmetros.

Por fim, foram repassados os vetores por referência, tendo por resultado a ordenação dos elementos.

Para realizar a contagem do tempo de execução utilizamos o método performance.now(), que retorna o *DOMHighResTimeStamp*, medido em milisegundos, com precisão de cinco milésimos de milissegundo (5 microsegundos). Portanto, fizemos a diferenciação entre o momento de início e o momento de finalização da execução.

No que tange ao tempo de execução, conforme pôde ser observado abaixo, e observamos em outras execuções que foram realizadas, não houve diferença significativa na ordenação de pequenos valores. Por vezes, o Bubblesort, mesmo com mais iterações, executou a ordenação com maior velocidade.

Observamos também em diversas execuções que conforme o número de elementos aumentava, o método de seleção direta tornava-se mais eficiente que o de inserção

direta, provavelmente por ser maior o número de recolocações que eram feitas, tornando a troca por posições mais eficiente.

A grande diferença entre os métodos de comparação se deu com grandes valores, a partir de vetores com 10.000 elementos. Quanto maior a quantidade de elementos, mais eficiente tornou-se o quicksort. Foi possível observar que com relação ao bubblesort, o quicksort foi mais de 140 vezes mais veloz.

Exemplificativamente, colacionamos abaixo alguns dos resultados encontrados:

MÉTODO	**QUANTIDADE DE ELEMENTOS**	**TEMPO DE EXECUÇÃO EM MS - TESTE 01**	**TEMPO DE EXECUÇÃO EM MS - TESTE 02**
Bubblesort	100	1.1300000005576294	0.295000001671724
Bubblesort	1.000	5.634999997710111	2.404999999213 0324
Bubblesort	10.000	134.66999999945983	132.67999999877065
Bubblesort	100.000	12519.199999998818	12390.565000001516
Seleção Direta	100	3.6000000000058208	0.260000000707805 16
Seleção Direta	1.000	8.030000000871 95	1.964999999443 5348
Seleção Direta	10.000	102.85500000100001	86.729999999050 05
Seleção Direta	100.000	7867.520000003424	7765.745000000577
Inserção Direta	100	3.169999999954598	0.264999998791608896
Inserção Direta	1.000	8.479999996779952	2.369999998307321
Inserção Direta	10.000	150.83999999842 49	126.35999999841 442
Inserção Direta	100.000	11988.385000000	11660.810000001

		562	22
Quicksort	100	0.6649999995715916	0.08499999967170879
Quicksort	1.000	4.0650000009918585	0.6549999985331167
Quicksort	10.000	9.790000000066357	6.355000001378357
Quicksort	100.000	87.68000000054599	69.25499999849126