UAlg – ISE – DEE
Algoritmia e Estruturas de Dados
Relatório do trabalho sobre ordenação

Marcos Ramos
Universidade do Algarve
Faro
a63059@ualg.pt

Resumo
Este relatório consiste na explicação do processo de desenvolvimento de um código em Java para ordenação de ficheiros com números inteiros desorganizados, utilizando dos algoritmos Shell Sort, Bubble Sort e Quick Sort.

1. INTRODUÇÃO

O trabalho proposto tinha como premissa o desenvolvimento de um código que criasse um ficheiro em formato binário, contendo uma certa quantidade de números inteiros de forma desorganizada. Neste caso foram criados seis ficheiros, cada um com respetivamente, dez, cem, mil, dez mil, cem mil e um milhão de números. O objetivo principal era ler o conteúdo desses ficheiros e organizar os números de forma crescente, utilizando os algoritmos estudados durante as aulas de Algoritmia e Estrutura de Dados, nomeadamente *Shell Sort*, *Quick Sort* e *Bubble Sort.*

Por fim era necessário realizar uma comparação entre os tempos que cada algoritmo precisava para terminar a ordenação dos números e dessa maneira, comparar a eficiência e desempenho de cada um conforme o tamanho do *array*.

2. ALGORITMOS DE ORDENAÇÂO ESTUDADOS

Bubble Sort - é o algoritmo de ordenação mais simples que funciona trocando repetidamente, os elementos adjaxcentes caso estes estejam na ordem errada. Por exemplo: [3, **6**, 5, 1] → [3, 5, 1, **6**] seria o resultado da primeira iteração e assim sucessivamente seguiria ordenando comparando os elementos adjacentes.

Shell Sort – baseado na inserção e a ordenação funciona dividindo o conjunto em subconjuntos. Diferente do Bubble Sort, a comparação é feita por elementos que estão distantes. Essa distância é inicialmente calculada pelo tamanho do *array* divido por dois, e após todas as comparações serem feitas e inseridas é novamente divido o valor dessa "distância" por dois e assim sucessivamente até todos array estar de fato ordenado.

Quick Sort - é um algoritmo eficiente de ordenação por divisão e conquista. O funcionamento baseia-se em uma rotina fundamental cujo nome é particionamento. Particionar significa escolher um número qualquer presente no array, chamado de pivot, e colocá-lo em uma posição tal que todos os elementos à esquerda são menores ou iguais e todos os elementos à direita são maiores. Assim é feito recursivamente até o *array* estar ordenado.

3. IMPLEMENTAÇÃO DO TRABALHO

O trabalho conta com sete classes. Três classes referenciando os algoritmos, duas classes demos (uma para criar os ficheiros e outra para ler e organizar os ficheiros), e mais duas classes que implementam o funcionamento da criação e leitura dos ficheiros.

Após a criação dos ficheiros, estes são salvos numa pasta *messyFiles*, e após sua organização são salvos numa pasta *orderFiles.*

Vale ressaltar aqui que para o correto funcionamento do software é necessário alterar a variável global *DIRECTORY* nas classes *ReadBinFile* e *CreateBinFile*, de maneira que o caminho onde será salvo esteja correto. Além disso é necessário a criação prévia das pastas acima referidas.

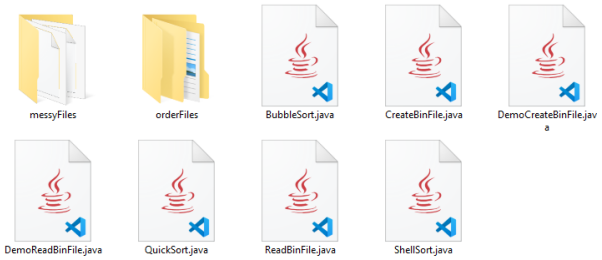

Figura 1 Classes e Pastas criadas para o projeto

4. MODO FUNCIONAMENTO

O código foi desenvolvido de maneira ser corrido por duas classes *demos*, onde a primeira *DemoCreateBinFile* é a responsável por criar de fato os ficheiros binários com os números aleatórios, se utilizando do método estático da classe *CreateBinFile* chamado *createRandomFile*. Este método escreve dentro do ficheiro números aleatórios dentro do ficheiro a ser criado, iterando dentro de um ciclo *for*, com n iterações onde n é o tamanho/quantidade de números que o ficheiro vai ter; n também é definido e passado como argumento de entrada.

Em seguida é necessário executar a classe *DemoReadBinFile*, onde é mostrado um pequeno menu ao utilizador onde este tem a possibilidade de escolher qual dos ficheiros gostaria de organizar. Conforme a escolha, três objetos são criados com base na classe *ReadBinFile;* estes objetos contém um *array* como variável de instância onde estão todos os números do ficheiro desorganizados.

Essa variável é passada como argumento para os métodos respetivos de todos os três algoritmos estudados. Cada método retorna o *array* organizado. Por fim é utlizado, o outro método estático de *CreateBinFile*, chamado *createOrderFile* para criar um novo ficheiro binário com os dados organizados. São criados três ficheiros, cada um referenciando o algoritmo utilizado na ordenação.

O ponto chave do código está no cálculo do tempo que cada algoritmo demora para realizar a ordenação. Para isso é utilizado em cada algoritmo o método *System.nanoTime*. Cria-se uma variável que guarda o tempo inicial e outra o final. Realiza-se a diferença entre estas e temos o tempo que o método que representa o algoritmo leva para terminar sua função.

4.1 Ficheiros de entrada

Após a execução da classe *DemoCreateBinFile*, são criados 6 ficheiros binários cada um contendo respetivamente dez, cem, mil, dez mil, cem mil e um milhão de números. Estes ficheiros são usados como input para leitura na classe *DemoReadBinFile.*

4.2 Condições especiais

É necessário alterar o diretório especificado, para um caminho que faça sentido na máquina do utilizador.

A alteração é feita nas classes *ReadBinFile* e *CreateBinFile.*

5. CONCLUSÕES E TRABALHO FUTURO

Após vários testes realizados foi possível concluir que de todos os algoritmos o mais eficiente é o *QuickSort* por ser realizado de forma recursiva.

Na figura abaixo é possível ver os tempos em milissegundos que cada algoritmo precisou para organizar o respetivo ficheiro.

Ficheiros/Números	QuickSort	ShellSort	BubbleSort
10	0,0058	0,0038	0,0044
100	0,0201	0,0223	0,1732
1000	0,2685	0,3961	4,0587
10000	1,0445	2,4379	107,9894
100000	10,1912	13,4586	17068,6243
1000000	108,5692	136,0303	17695436,99

Figura 2 - Comparação dos tempos em Milissegundos

6. REFERÊNCIAS

QuickSort, GeeksforGeeks, Janeiro 2022.

<https://www.geeksforgeeks.org/quick-sort/>

ShellSort, GeeksforGeeks, Agosto 2021.

< https://www.geeksforgeeks.org/shellsort/>

BubbleSort, GeeksforGeeks, Março 2022.

<https://www.geeksforgeeks.org/bubble-sort/>

[Lam12] Lam, R.. Ficheiros para ordenação, Novembro 2012.

<http://w3.ualg.pt/~rlam/FicheiroOrdenacao.html>