Métodos de Sorting

37762 - Ricardo Mochila
40102 - Inês Verissimo

January 2019

1 Introdução

Este relatório tem como objetivo mostrar noções sobre vários métodos de sorting com base no trabalho realizado.
Foram escolhidos os algoritmos de ordenação Heap e Quick, consoante o input do utilizador.

2 public static Comparable[ ] leArray()

Neste método pretende-se receber o input do utilizador. Foi utilizado um Scanner para saber a quantidade de elementos que se pretende organizar, e outro que recebe os próprios elementos.
Para tratar o input, utilizou-se um StringTokenizer, cada token foi adicionado a um array de Comparables.

3 public static void printArray(Comparable[ ] x

Este método destina-se unicamente à impressão do array que foi gerado acima.

4 HeapSort

O HeapSort inicialmente organiza o array de forma a ter o maior elemento no início e, de seguida, envia-o para o fim do array trabalhando com os restantes elementos recursivamente até o mesmo estar organizado.

4.1 private static void build_heap(Comparable[ ] A)

Neste metodo é criada a Heap inicial. É usado um ciclo que corre pela metade do tamanho do array que se pretende organizar. Dentro deste é chamado outro método que exerce o controlo sobre o mesmo.

4.2 public static void heapify(Comparable[ ] array, int i, int size)

Nesta fase do algoritmo, é criado um inteiro 'fim' apartir da variavel recebida pela função 'i', que através de sucessivas comparações altera o array para que os elementos de maior valor fiquem no início do array.

4.3 public static void method1(Comparable [] A)

Este método destina-se à construção do algoritmo de sorting, chamando o build_heap(). Seguidamente, é criado um ciclo que vai trocar o primeiro elemento do array pelo do fim ficando, assim, o último dado na posição correta, posteriormente chama-se o metodo heapify trabalhando apenas com a parte desordenada, este processo realizado repetidamente organizará o array.

5 QuickSort

Este algoritmo passa pela escolha de um pivot, comparando os elementos à sua esquerda e à sua direita, trocando-os sempre que um valor da esquerda for maior que o valor da direita do pivot.

5.1 private static int particao(Comparable[] a, int esq, int dir)

Neste método calcula-se o pivot pela divisão da soma dos elementos com que se está a trabalhar, corremos o array do início para o fim comparando sempre com o pivot, quando o elemento for menor pára-se o ciclo. Percorre-se novamente do fim para o início enquanto os elementos forem maiores que o pivot, caso contrário o ciclo termina, segue-se uma comparação entre o número de passos que se realizaram entre a esquerda e a direita, trocando os elementos sempre que a condição se realizar.

5.2 private static void qsort(Comparable[] a, int esq, int dir)

Nesta fase, cria-se uma condição entre dois inteiros, caso se verifique, dá-se a uma variável o valor retornado pelo método partição(), e corre-se recursivamente o qsort() com o valor da variável, quando esta recursividade falhar, chama-se novamente o metodo qsort atribuindo a um dos elementos o tamanho da partição mais um, finalizando assim a ordenação do array.

5.3 public static void method2(Comparable[] A)

Este método destina-se apenas a fazer correr o algoritmo QuickSort.

6 Conclusão

Neste trabalho foi possível compreender os algoritmos utilizados, percebendo como se aplicam em casos reais. O código sobre os algoritmos teve como fonte os diapositivos disponibilizados no moodle da disciplina, sendo este percebido na integra e só implementado o que realmente se compreendeu. Os algoritmos funcionam apenas com números inteiros devido ao uso do StringTokenizer, porém de fácil adaptação caso seja pretendida a organização de elementos de outro tipo.