Algoritmos e Estruturas de Dados

1ª Série

(Problema)

Junção ordenada de ficheiros, sem repetições

49470 Ana Carolina Pereira
50546 Rafael Nicolau

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2022/2023

16/4/2023

Índice

1. Introdução

Neste relatório vão ser apresentadas duas soluções para a resolução do problema apresentado. Para a implementação das diversas funções, foram utilizados algoritmos lecionados ao longo das aulas.

2. Junção ordenada de ficheiros, sem repetições

O problema proposto consistiu em desenvolver uma aplicação que permita juntar, de forma ordenada, os dados provenientes de vários ficheiros, produzindo um novo ficheiro de texto (output.txt) ordenado de modo crescente, sem repetições e com uma palavra por linha. Os ficheiros originais encontram-se ordenados de modo crescente e contêm uma palavra por linha.

2.1 Análise do problema

A resolução deste problema baseou-se em duas implementações. A primeira implementação, JoinFiles1.kt, consistiu em utilizar apenas a estrutura de dados *Array,* enquanto que a segunda, JoinFiles2.kt permite utilizar todas as estruturas de dados que considerássemos necessárias, exceto o uso de *Arrays*.

2.2 Estruturas de Dados

Tendo sido pedida a resolução do problema utilizando Arrays e outro tipo de ADT, optou-se por utilizar listas mutáveis na segunda versão.

Um array é uma estrutura de dados que consiste em armazenar vários elementos do mesmo tipo de dados, tem tamanho fixo (não podem aumentar nem diminuir o seu tamanho) e são mutáveis enquanto que uma lista mutável podem aumentar ou diminuir o seu tamanho.

```
val a = arrayOf(1, 2, 3)
a[0] = a[1] // está correto
val l = listOf(1, 2, 3)
l[0] = l[1] // está incorreto
val m = mutableListOf(1, 2, 3)
m[0] = m[1] // está correto

val a = arrayOf(1, 2, 3)
println(a.size) // vai ser sempre 3

val l = mutableListOf(1, 2, 3)
l.add(4)
println(l.size) // vai alterar e ficar 4
```

2.3 Algoritmos e análise da complexidade

Para a resolução deste problema foi necessário utilizar *Priority Queues* e recorrer a amontoados binários - neste caso, o *Heap Sort*. Um *heap* é uma estrutura de dados parcialmente ordenada, representada numa árvore binária completa (ou quase completa).

Um *heap* tem duas propriedades:

- Estrutural, em que a árvore está completa ou quase completa.
- De ordenação, em que o valor de cada um dos nós é maior ou igual que o valor dos nós dos seus filhos.

Os valores dos nós de um heap estão ordenados de cima para baixo, ou seja, a sequência de valores de qualquer caminho desde a raiz a uma folha é decrescente.

Existem dois tipos de *heaps*, minHeapify e maxHeapify. Em ambos os casos o valor dos nós satisfaz a propriedade de ordenação.

No maxHeapify, para todo o nó diferente da raiz H [Parent (i)] >= H [i] enquanto que no minHeapify, para todo o nó diferente da raiz H [Parent (i)] <= H [i]

Para a resolução deste problema, foi necessário recorrer ao uso do minHeapify.

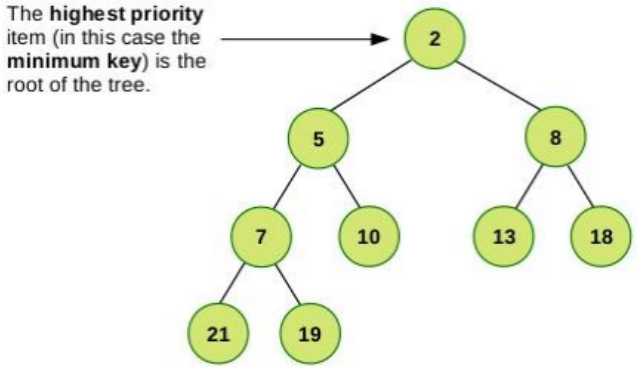

Figura 1: exemplificação do uso de um minHeapify

3. Avaliação Experimental

Nesta secção é possível observar os testes experimentais realizados. É ainda importante referir que os valores atribuídos para a realização da tabela e do gráfico foram obtidos numa máquina com processador AMD Ryzen 5 4500U with Radeon Graphics 2.38 GHz, RAM 16,0 GB e sistema operativo de 64 bits, processador baseado em x64.

Nº Ficheiros	JoinFiles1	JoinFiles2
3	394,9672	452,4716
6	571,9278	627,5374
9	567,6104	595,9177
12	898,7067	977,2857
15	612,8982	594,7714
18	707,5508	677,8908
21	840,9809	847,1529
24	925,6448	1210,6908
27	957,0541	1103,9774
30	1255,154	1184,3316

Tabela 1: Resultados do tempo de execução para os ficheiros

A Figura 2, ilustra em termos comparativos através de um gráfico, os tempos de execução dos dois algoritmos realizados para os dados ficheiros.

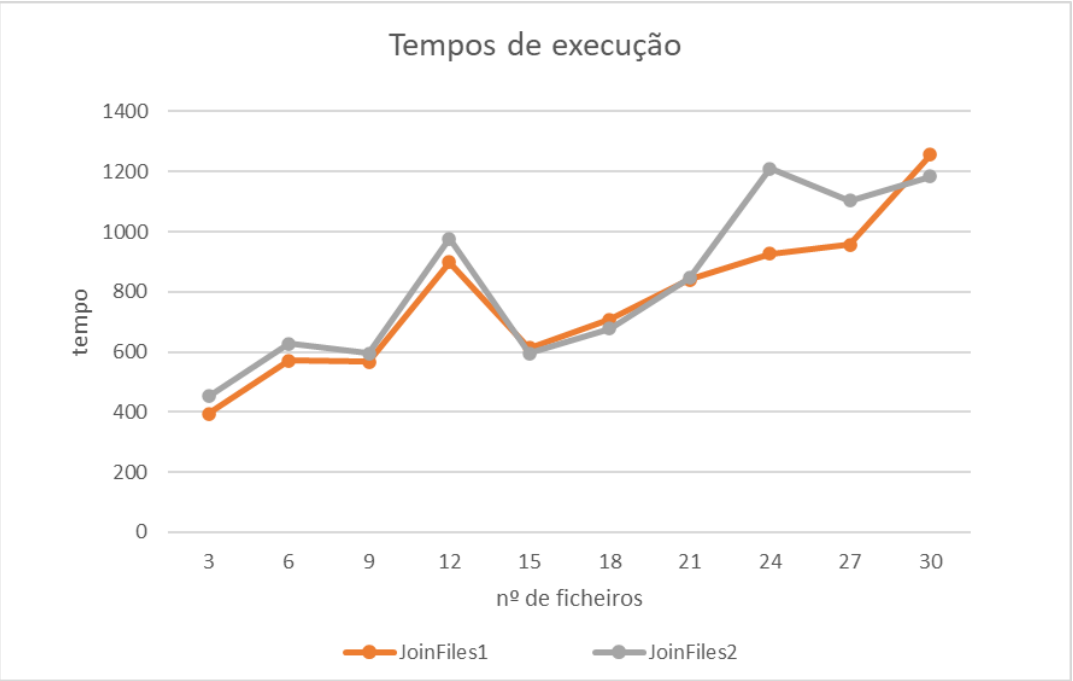

Figura 2: Comparação dos tempos de execução dos algoritmos de ordenação.

Os resultados das avaliações experimentais mostram que os algoritmos têm complexidade temporal O(m log2 n) - como era pedido no guião do trabalho.

4. Conclusões

Com esta série de exercícios, conseguimos aprofundar o conhecimento sobre algoritmo e perceber a sua importância no desenvolvimento de programas, de modo a que sejam mais rápidos e ocupem menos memória.

Referências

[1] “Disciplina: Algoritmos e Estruturas de Dados - 2223SV,” Moodle 2022/23. [Online]. Available: https://2223.moodle.isel.pt. [Accessed: 16-03-2023].

[2] Introduction to Algorithms, 3º Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. MIT Press.