

**Algoritmos e Estruturas de Dados   
1ª Série**

**Juncão ordenada de ficheiros sem repetições**

|  |  |
| --- | --- |
| N. 50471 | Brian Melhorado |
| N. 50543 | Arthur Oliveira |

Licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2022/2023

16/4/2023

**Índice**

[1. Introdução 2](#_Toc129882168)

[2. Título Problema 3](#_Toc129882169)

[2.1 Análise do problema 3](#_Toc129882170)

[2.2 Estruturas de Dados 3](#_Toc129882171)

[2.3 Algoritmos e análise da complexidade 3](#_Toc129882172)

[3. Avaliação Experimental 5](#_Toc129882173)

[4. Conclusões 7](#_Toc129882174)

[Referências 8](#_Toc129882175)

# Introdução

Pretende-se desenvolver uma aplicação que permita juntar de forma ordenada os dados provenientes de vários ficheiros, produzindo um novo ficheiro de texto ordenado de modo crescente e sem repetições. Os ficheiros originais encontram-se ordenados de modo crescente e contem uma palavra por linha.

O objetivo da aplicação a desenvolver é a produção de um novo ficheiro de texto, ordenado de modo crescente, que contenha as palavras dos ficheiros pertencentes ao conjunto F, mas sem repetições. O ficheiro produzido deverá também conter uma palavra por linha.

O relatório divide-se em três secções:

• Abordagem do problema

• Avaliação experimental

• Conclusão

# Juncão ordenada de ficheiros, sem repetições

Esta secção tem como intuito abordar o problema a resolver. Vamos dividi-la em três partes:

• **Analise do problema**, onde será apresentada uma descrição do problema, assim como também as operações necessárias a implementar e os métodos utilizados para o resolver

• **Estrutura de Dados**, onde serão ilustradas as principais estruturas de dados (ADT) utilizadas na solução do problema, com alguns exemplos da sua utilização.

## •Algoritmos e análise da complexidade, que vai constar da descrição dos principais algoritmos utilizados na implementação das operações usadas para o desenvolvimento da solução, assim como também a analise da sua complexidade

## Análise do problema

O problema é descrito por:

• Um conjunto F = {f1, ..., fn} de n ficheiros de texto, em que n > 0;

• Cada linha de um ficheiro fn tem uma palavra;

• Os ficheiros fn estão ordenados lexicograficamente, de modo crescente;

• O número total de palavras é m.

O objetivo da aplicação a desenvolver é a produção de um novo ficheiro de texto, ordenado de modo crescente, que contenha as palavras dos ficheiros pertencentes ao conjunto F, mas sem repetições. O ficheiro produzido deverá também conter uma palavra por linha.

Aqui deverá ser apresentada uma descrição do problema, assim como as operações que são necessárias implementar, e abordagem utilizada para o resolver.

## Estruturas de Dados

O nosso Array apresenta o primeiro elemento de cada ficheiro de input e vamos retirando o menor elemento do mesmo e apos isso substituímos esse elemento pelo próximo elemento do mesmo ficheiro

Aqui deverão ser ilustradas as principais estruturas de dados (ADT) utilizadas na solução do problema, com alguns exemplos da sua utilização. As figuras devem ser referenciadas no texto

como por exemplo ao explicar o que representa a Figura 1.

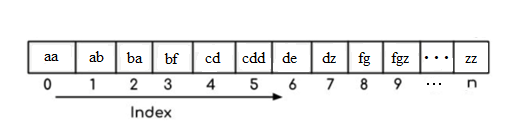


Figura 1: Exemplo da estrutura de dados *array.*

## Algoritmos e análise da complexidade

Para a implementação do algoritmo foi utilizado uma Priority Queue aonde é utilizado um minHeap para a ordenação dos elementos a serem armazenados no outputFile. Para ordenação do minHeap utilizamos o minHeapify que ordena o Array de maneira crescente, o offer para adicionar um novo elemento no Array de forma que o mesmo se mantenha ordenado, o peak que permite visualizar o menor elemento do Heap e o poll que remove o menor elemento do Heap e o substitui pelo elemento a seguir. Utilizando a Priority Queue o espaço necessário para armazenamento vai ser n (aonde n é o número de ficheiros lidos pelo programa) e a complexidade temporal vai ser m(log(n)) aonde m é o número total de palavras dos ficheiros. Ao iniciar o minHeap são feitos n offers para preencher o Array e a cada offer é chamada a função minHeapify para manter a ordenação da estrutura. Depois de lido os n ficheiros é chamada a função poll e logo em seguida a função offer (para retirar o menor elemento e colocá-lo no outputFile caso não seja repetido e ler a próxima palavra do ficheiro), esses processos se repetem vezes até o algoritmo ler todas as palavras dos ficheiros.

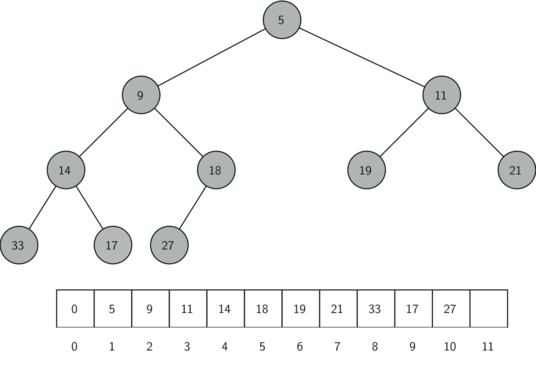


Figura 2: Exemplo de atualização de uma estrutura de dados (ordenação de modo crescente dos dados utilizando o algoritmo Priority Queue).

O minHeapify é responsável pelo maior custo gerado na Priority Queue ficando com o custo total de logaritmo de base 2 de (n).

# Avaliação Experimental

Para fazer a avaliação experimental utilizamos os três ficheiros (f1.txt, f2.txt e f3.txt) disponibilizados pela professora, assim como também outros 4 ficheiros para fazer testes adicionais (f4.txt, f5.txt, f6.txt e f7.txt), o f4 é contido por apenas 8 palavras ordenadas lexicograficamente e de maneira crescente, mas os restantes são replicas dos primeiros 3 ficheiros e foram implementados desta forma para comprovar como é que a aplicação reagia nestes casos.

Características da máquina onde foram realizados os testes:

Processador: AMD Ryzen 7 3700X 8-Core Processor 3.59 GHz

RAM : 16,0 GB (15,9 GB usable)

Tipo de sistema Sistema operativo de 64 bits, processador baseado em x64

Tabela 1: Resultados do tempo de execução de algoritmos de ordenação considerando várias amostras.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Número de ficheiros | | | | | |
|  | 2 | 3 | 4 | 5 | 6 | 7 |
| Priority Queue | 134.171ms | 187.939ms | 196.451ms | 253.826ms | 319.698ms | 367.530ms |
| AED  Priority Queue | 151.772ms | 198.638ms | 239.432ms | 384.272ms | 400.390ms | 421.270ms |

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Figura 3: ilustra em termos comparativos através de um gráfico, os tempos de execução das duas execuções.

Através das avaliações realizadas pode-se perceber um crescimento linear logarítmico no tempo de execução da função, sendo que quanto maior a soma do número de elementos nos ficheiros a serem analisados maior o tempo de execução obtido nas análises.

# Conclusões

Podemos concluir que os resultados obtidos nas experiências realizadas são satisfatórios, assim como também, a abordagem do problema e os métodos implementados na resolução do mesmo. Em relação as limitações, observamos algumas na disponibilidade de memoria o que nos forçou a utilizar métodos mais eficientes para abordar este problema.

A utilização da priority queue facilitou-nos uma organização eficiente dos dados desejados e para alem disso, a implementação deste algoritmo possibilitou-nos uma redução considerável da memoria utilizada para este processo.

# 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.

[3] <https://www.canva.com/es_mx/graficas/graficas-de-lineas/>

[4]