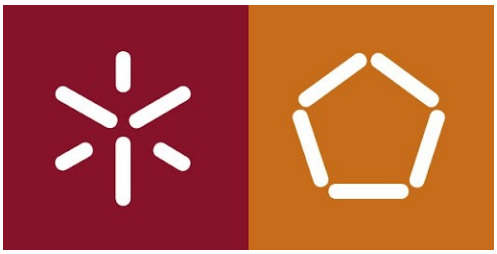

Universidade do Minho

Licenciatura em Engenharia Informática

LI3 - Sistema de organização e escrita de dados

Relatório da 2ª fase do Projeto

Grupo 21

Gonçalo Cruz (A104346) Gonçalo Cunha (A104003)
Luis Freitas (A104000)

Ano Letivo 2023/2024

Índice

Capítulo 1

Introdução e dificuldades iniciais

A segunda fase deste projeto exigiu de nós um cuidado muito mais elevado em relação à escolha das estruturas de dados a utilizar e ao método escolhido para abordar cada *query*. Na primeira fase as nossas estruturas eram eficientes, tendo em conta a dimensão do *dataset* usado, mas vieram a revelar-se extremamente ineficientes para o *data_large*, pelo que tivemos de refazer grande parte do projeto. Acabámos por conseguir obter um tempo de execução e um consumo de memória bastante positivos (a análise a estes resultados será feita à frente).

Para este trabalho, decidimos usar tabelas de *hash* (criadas por nós) para o armazenamento das informações necessárias, uma vez que as achámos estruturas muito eficientes e simples de trabalhar. Inicialmente sentimos algumas dificuldades relativamente a este tipo de dados, mas quando conseguimos resolver uma *query*, tornou-se razoavelmente fácil adaptar às restantes. O principal problema que surgiu com esta estrutura foram as colisões (na tabela dos utilizadores), já que nos obrigou a ter outros cuidados no manuseamento dos dados. Uma outra dificuldade com que nos deparámos foi a implementação dos testes funcionais e de desempenho, visto que era algo que nunca tínhamos usado, mas conseguimos também ultrapassar este obstáculo.

Capítulo 2

Estruturas de dados e módulos de utilidade

2.1 – Aux_functions

Este módulo contém funções auxiliares usadas por módulos como o das *queries* ou o dos catálogos. Nele estão presentes funções de comparação usadas pela *qsort,* funções que retornam apenas o ano, o mês e o ano, ou o dia, o mês e o ano de uma data fornecida como argumento, funções que calculam valores (como o número de noites ou o preço de uma reserva), entre outras.

2.2 – Aux_validation

De forma semelhante ao módulo anterior, este contém funções que auxiliam na validação dos dados. Alguns exemplos são a *verify_email*, a *check_date* e a *check_price_per_night*, que são usadas em vários módulos, além do Validation.

2.3 – Catalogs

Neste módulo constam todas as estruturas usadas no projeto. Tem também funções responsáveis por armazenar e processar as informações das quatro entidades (os *getters* e os *setters*), funções *create* (que alocam memória e inicializam alguns parâmetros) e outras funções necessárias.

2.4 – Free

Como o nome indica, neste módulo temos as funções de libertação de toda a memória alocada ao longo do programa, tanto de estruturas completas como dos vários campos.

2.5 – Hash

Aqui temos funções de *hash* que, recebendo uma chave (*key*), retornam o valor correspondente, ou seja, o índice da *hash table* para onde deve ir essa chave.

2.6 – Interactive_mode_screens

Este é o módulo responsável por criar as telas auxiliares do modo interativo, exibidas no ecrã após o início do programa. Contém funções responsáveis por escrever o texto no ecrã, ler os *inputs* do utilizador e chamar funções de outros módulos.

2.7 – Interpreter

No módulo *Interpreter* temos três funções que leem e interpretam as *queries* requeridas (quer pelo ficheiro *input.txt* no modo *batch*, quer pelo utilizador no modo interativo). Também é este o módulo responsável pela contagem do tempo de execução de cada *query*.

2.8 – Menu_options

De forma semelhante ao Interactive_mode_screens, este módulo é responsável por apresentar os ecrãs do menu inicial do modo interativo, bem como a tela de ajuda e a de créditos.

2.9 – Menu

Aqui temos apenas a função *create_menu* que, utilizando a biblioteca *ncurses*, cria o ecrã principal do modo interativo, escrevendo o nome da aplicação e as opções e chamando a função correspondente ao que o utilizador pretender fazer.

2.10 – Output_errors

Este módulo é responsável por realizar a escrita dos parâmetros inválidos do *dataset* na respetiva saída (terminal ou ficheiro de erros).

2.11 – Output_queries

Aqui temos as funções que escrevem nos ficheiros de *output* ou no terminal os resultados das queries e ainda a função *create_output*, que é chamada pelas mesmas e que cria esses ficheiros.

2.12 – Pagination

Tal como o nome indica, este módulo trata da paginação do *output* das queries quando, no modo interativo, o utilizador escolhe que este seja mostrado no terminal. Isto é feito pela função *show_output*.

2.13 – Parser

Neste módulo, a única função que existe é a *open_files*, que abre os ficheiros do *dataset* (de passageiros, voos, reservas e utilizadores) e faz o *parsing* dos mesmos (extrai os dados necessários e guarda na matriz parâmetros). Realiza também algumas operações que auxiliam as *queries*.

2.14 – Performance

O nome deste módulo é, mais uma vez, intuitivo, já que contém as funções necessárias à medição do tempo de execução da aplicação e de cada *query* e do consumo de memória.

2.15 – Queries

As funções das *queries* estão todas presentes neste módulo, executando o que é pedido por cada uma delas, com recurso às diversas funções auxiliares e chamando as funções de *output* correspondentes.

2.16 – Tests

Este módulo tem duas funções: a *compare_files*, que verifica, linha a linha, se o ficheiro de *output* gerado pelo programa é igual ao da pasta dos *outputs* corretos, e a *find_equal_files*, que abre os dois arquivos necessários, chama a primeira e, conforme o resultado por esta retornado, imprime o necessário no ficheiro *output_tests.csv*.

2.17 – Validation

Finalmente, o módulo Validation contém as quatro funções de validação dos dados, que usam as funções presentes no Aux_validation.

Capítulo 3

Arquitetura da aplicação

Graças à modularidade do código, há uma forte interligação entre as diversas componentes. Como se pode ver na figura 1, há nove módulos principais, que recorrem a outros para funcionarem. Como exemplo, o Menu_options trabalha com o Interactive_mode_screens, chamando a função *start_query*, e o Catalogs recorre ao Output_errors (nomeadamente às funções *complete*) para imprimir os parâmetros inválidos mal os lê, não tendo assim de os guardar.

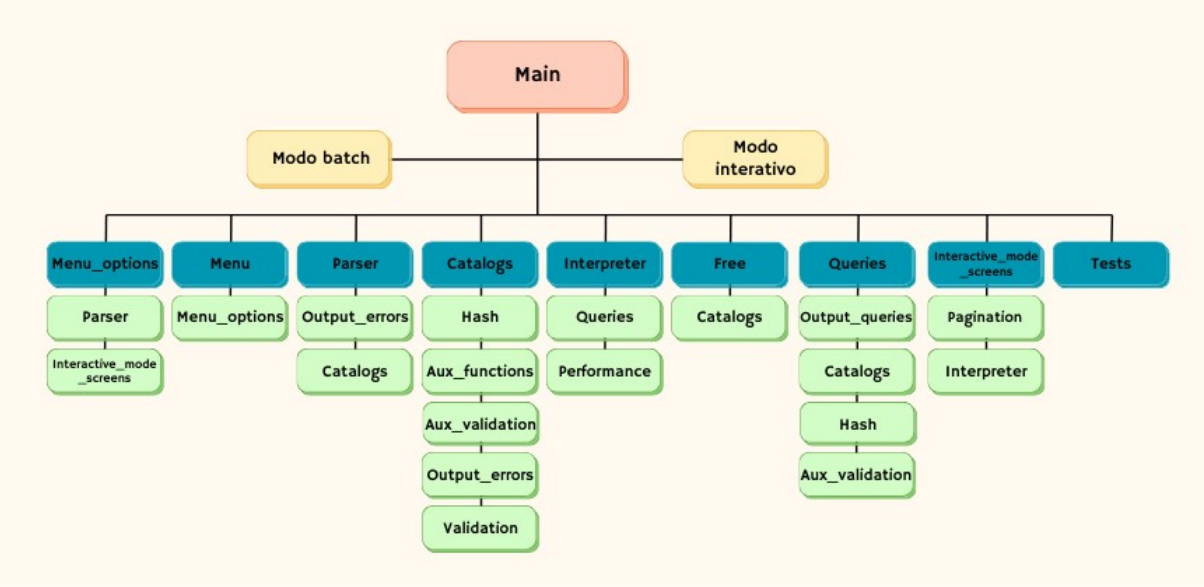

Figura 1 – Arquitetura da aplicação

Capítulo 4

Testes funcionais e de desempenho

Em relação ao desempenho da nossa aplicação, acreditamos ser bastante positivo, tanto em termos do tempo de execução como do uso de memória. Tomámos a decisão de usar memória estática para as *hash tables*, visto que verificámos que, assim, o programa executava de forma mais rápida. Já para outras estruturas, como a matriz parâmetros ou listas auxiliares, aplicámos conhecimentos de memória dinâmica. Ao longo do desenvolvimento deste trabalho, tivemos sempre a preocupação de manter um tempo e um consumo de memória baixos, pelo que houve a necessidade de alterar estratégias para certas *queries*. Apresentam-se a seguir os resultados obtidos em três testes (com o *dataset large*) em cada uma das máquinas dos membros do grupo:

```
Query 1 time: 0.006260 seconds
Query 2 time: 0.001637 seconds
Query 3 time: 0.191683 seconds
Query 4 time: 0.541630 seconds
Query 5 time: 0.785093 seconds
Query 6 time: 0.001788 seconds
Query 7 time: 0.000582 seconds
Query 8 time: 0.404536 seconds
Query 9 time: 0.895986 seconds
Query 10 time: 0.000371 seconds

Application time: 57.453701 seconds
Memory usage: 2785292 KB
```

Figura 2.1 – Desempenho no primeiro teste na máquina 1

```
Query 1 time: 0.006379 seconds
Query 2 time: 0.001650 seconds
Query 3 time: 0.190875 seconds
Query 4 time: 0.538323 seconds
Query 5 time: 0.793454 seconds
Query 6 time: 0.001819 seconds
Query 7 time: 0.000588 seconds
Query 8 time: 0.402913 seconds
Query 9 time: 0.893726 seconds
Query 10 time: 0.000373 seconds

Application time: 56.979473 seconds
Memory usage: 2785176 KB
```

Figura 2.2 – Desempenho no segundo teste na máquina 1

```
Query 1 time: 0.006288 seconds
Query 2 time: 0.001623 seconds
Query 3 time: 0.190539 seconds
Query 4 time: 0.539259 seconds
Query 5 time: 0.788154 seconds
Query 6 time: 0.001795 seconds
Query 7 time: 0.000595 seconds
Query 8 time: 0.402929 seconds
Query 9 time: 0.894863 seconds
Query 10 time: 0.000377 seconds

Application time: 57.184296 seconds
Memory usage: 2785252 KB
```

Figura 2.3 – Desempenho no terceiro teste na máquina 1

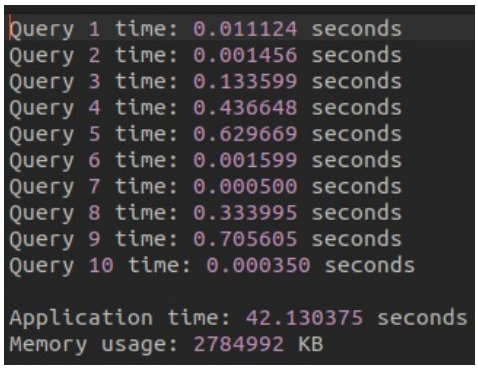

Figura 3.1 – Desempenho no primeiro teste na máquina 2

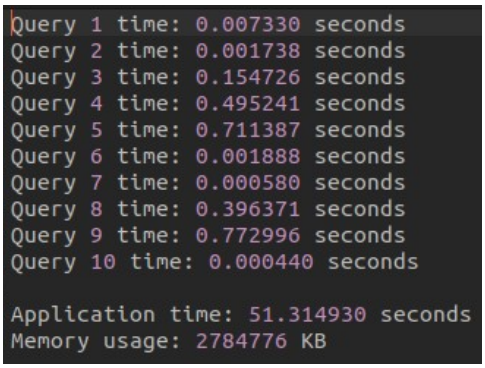

Figura 3.2 – Desempenho no segundo teste na máquina 2

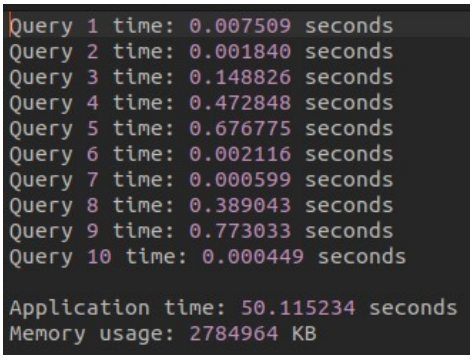

Figura 3.3 – Desempenho no terceiro teste na máquina 2

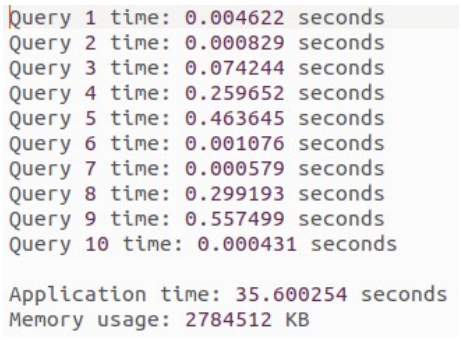

Figura 4.1 – Desempenho no primeiro teste na máquina 3

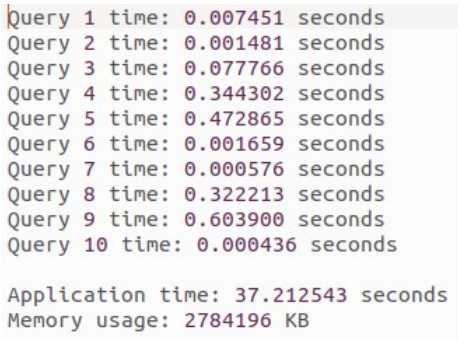

Figura 4.2 – Desempenho no segundo teste na máquina 3

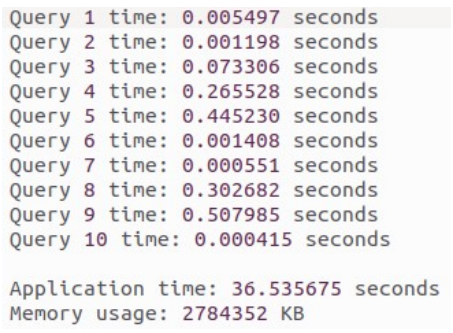

Figura 4.3 – Desempenho no terceiro teste na máquina 3

Tempo da Aplicação (s)	Máquina 1	Máquina 2	Máquina 3
Teste 1	57,45	42,13	35,6
Teste 2	56,98	51,31	37,21
Teste 3	57,18	50,12	36,54
Média	**57,2**	**47,85**	**36,45**

Figura 5 – Valores do tempo de execução da aplicação (em segundos) para 3 testes nas diferentes máquinas

A partir destes resultados, podem-se tirar algumas conclusões:

- O consumo de memória é aceitável e praticamente sempre o mesmo, independentemente da máquina, e o tempo de execução, apesar de variar um pouco, também anda sempre à volta dos mesmos valores, comparando uma mesma máquina.

- Em relação ao tempo de execução da aplicação, foi possível chegar a um valor bastante baixo, graças às estratégias de otimização implementadas e à utilização de algoritmos eficientes, nomeadamente o facto de termos passado a inserir os elementos no início das listas ligadas, uma vez que a inserção no final demorava uma quantidade significativa de tempo, e o uso do algoritmo de ordenação *qsort*, que se traduziu numa redução considerável do tempo de execução, comparando com um outro método de ordenação usado inicialmente por nós.

- Todas as *queries* têm um tempo de execução baixo (menor do que 1 segundo), sendo que a mais lenta é a *query* 9, já que tem de copiar a lista dos utilizadores para uma lista auxiliar, ordenar essa lista, guardar os nomes e identificadores pretendidos e ordenar novamente uma outra lista, com os nomes cujo o prefixo é o pedido pela *query*.

- As *queries* mais rápidas são a 7 e a 10, visto que grande parte do trabalho das mesmas é realizado aquando do *parsing* dos ficheiros (não sendo, assim, contabilizado no tempo na *query* em si).

Capítulo 5

Conclusão

A título de balanço geral, consideramos que foi um projeto bem conseguido, tendo em conta os objetivos de funcionamento da aplicação e os desafios associados (como modularidade, encapsulamento, adequação de algoritmos e estruturas, entre outros). Como já foi referido neste relatório, cumprimos ainda as metas de eficiência, tendo adquirido conhecimentos em relação a este aspeto. Conseguimos, ainda, que o nosso programa execute sem *memory leaks*.

Para o futuro, levamos reforçada a importância do planeamento de um projeto e também diversos conhecimentos adquiridos com este trabalho, não só relativamente aos conceitos explorados mas também a nível de trabalho em grupo.