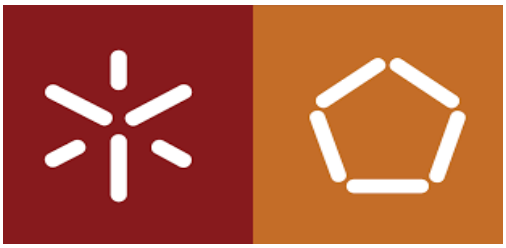
Relatório

Trabalho Prático - Fase 2

Grupo 44

Hugo Arantes Dias (a100758)
Ricardo Miguel Queirós de Jesus (a100066)
Rui Pedro Fernandes Madeira Pinto (a100659)



Departamento de Informática

Licenciatura em Engenharia Informática

2º ano | 1º Semestre

Laboratórios de Informática III

Índice

1 Introdução

O objetivo desta aplicação consiste em fazer o parsing de 3 ficheiros de entrada (.csv), e com os dados obtidos responder a um conjunto de queries com os argumentos indicados. Também temos como objetivo desenvolver um modo interativo apelativo, de modo a ser possível analisar os resultados das queries.

O presente relatório visa descrever e explicar em detalhe o trabalho desenvolvido pelo nosso grupo durante o projeto no âmbito da unidade curricular de Laboratórios de Informática III, lecionada no curso de Licenciatura em Engenharia Informática na Universidade do Minho, no 1º semestre do ano letivo de 2022/2023.

Este relatório apresenta-se como a continuação do relatório da fase 1, já entregue pelo grupo.

Os principais objetivos do grupo na segunda fase deste projeto consistem em melhorias em termos de memória, tempo de execução e encapsulamento e modularidade. Nesta fase tínhamos também como objetivo resolver todas as queries proposta, desenvolver uma interface interativa funcional e visualmente apelativa ao utilizador, permitindo uma fácil navegação pelos resultados obtidos.

2 Organização dos Ficheiros

Na 2ª fase do projeto, decidimos melhorar a organização da nossa estrutura dos ficheiros de forma a que a mesma se apresente mais organizada.

Neste projeto, dividimos o código em vários ficheiros consoante os argumentos e funcionalidades das funções.

Na pasta **src/** encontram-se os seguintes ficheiros:

- **main.c** - main do programa, responsável por chamas funções de inicialização de estruturas, de limpeza das mesma e também de funções responsáveis pela leitura de comandos.

- **mainTestes.c** – contém as mesmas funções que main.c mas aqui damos uma variável que leva o programa a executar o processo de testes em vez do modo batch.
- **loadGeral.c** - contém função que chama as funções responsáveis pelo parsing dos dados e pela construção dos topUsersDist e topDriversAval.
- **rides.c** - contém a estrutura, funções de inicialização, *get's*, *set's* e de limpeza relativas às *rides*.
- **users.c** - contém a estrutura, funções de inicialização, *get's*, *set's* e de limpeza relativas aos *users*.
- **drivers.c** - contém a estrutura, funções de inicialização, *get's*, *set's* e de limpeza relativas aos d*rivers*.
- **structs.c** - contém estrutura e função de inicialização da estrutura geral do projeto. Estão também presentes algumas funções auxiliares, funções de validação de campos, e função limpeza da estrutura geral.
- **avaliacao_driver.c** - contém a estrutura para efetuar o topDriversAval.
- **city.c** - contém funções relativas à struct city.
- **date.c** - contém estrutura e funções relacionadas à da estrutura *Day*.
- **dist_user.c** – contém a estrutura para efetuar o topUsersDist.
- **hash.c** - contém algumas funções da Glib relativas às HashTables.
- **parserDrivers.c** - contém função responsável pelo *parsing* dos *drivers*.
- **parserUsers.c** - contém a função responsável pelo *parsing* dos *users*.
- **parserRides.c** - contém a função responsável pelo *parsing* das *rides*.
- **interactive_mode.c** - contém a função *make_menu()*, principal função responsável pela parte interativa do projeto e outras funções auxiliares usadas pela função *make_menu()*.
- **queries.c** - contém funções relacionadas ao modo batch e funções relacionadas ao processo dos testes e validação dos mesmos.
- **query1.c** - contém funções utilizadas para resolver a *query* 1.
- **querie2.c** - contém funções utilizadas para resolver a *query* 2.
- **querie3.c** - contém funções utilizadas para resolver a *query* 3.
- **querie4.c** - contém funções utilizadas para resolver a *query* 4.
- **query5.c** - contém funções utilizadas para resolver a *query* 5.
- **query6.c** - contém funções utilizadas para resolver a *query* 6.
- **query7.c** - contém funções utilizadas para resolver a *query* 6.
- **query8.c** - contém funções utilizadas para resolver a *query* 6.
- **query9.c** - contém funções utilizadas para resolver a *query* 6.

3 Estruturas de Dados

3.1. - Estrutura de dados principal

```
struct geral
{
    GHashTable *hashDriv;
    GHashTable *hashUser;
    GList *rides;
    GHashTable *hashCity;
    GArray *top_Drivers_Aval;
    GArray *top_User_Dist;
    GPtrArray *day;
};
```

Estrutura principal do projeto, que serve para armazenar as structs de armazenamento de informação necessárias para responder às queries.

3.2. - Estrutura de dados auxiliares

```
struct aval_driver
{
    char* id;
    char *name;
    double avaliacao_med;
    int last_ride_days;
};
```

A estrutura **aval_driver** é uma estrutura auxiliar com o objetivo de facilitar o armazenamento dos dados da query 2, em que temos, para um driver, o seu id, nome, avaliação média e quando efetuou a sua última viagem(em dias).

```
typedef struct dist_user
{
    char *username;
    char *name;
    int distance;
    int last_ride_days;
} Dist_User;
```

A estrutura de dados dist_user é uma estrutura auxiliar que serve para facilitar o armazenamento dos dados da query 3, em que temos, para um user, o seu username, nome, distância percorrida e quando e quando efetuou a sua última viagem(em dias).

```
struct city
{
    char *cityname;
    double preco;
    int viagens;
    GList *drivers;
};
```

Esta estrutura de dados contém o nome da cidade, preço total de todas as viagens da cidade em questão e o número de viagens efetuadas na mesma. Estes dois últimos dados são armazenados para que seja possível calcular o preço médio de cada viagem na cidade de forma mais fácil nas queries em que tal informação é pedida. Também possui uma Glist de city_driver organizados pela sua avalição média em rides nessa cidade.

Esta estrutura de dados é usada para resolver as queries 5 e 6, que utilizam intervalos entre duas datas. A estrutura day é inicializada com todos os dias entre duas datas, e com uma lista de dados das viagens realizadas naquele dia. Por exemplo, o primeiro elemento desta estrutura é o dia 01/01/2000, o segundo é dia 02/01/2000 e por assim adiante. Cada dia contém também o número total de viagens realizadas nesse dia como também a soma de todo o preço das viagens nesse dia. E cada viagem armazenada nessa estrutura day, guarda informações sobre o id, nome da cidade, distância e gorjeta da viagem. Esta estrutura é preenchida durante o parsing dos dados.

```
typedef struct day {
    int day;
    int month;
    int year;
    int num_viagens;
    double total_preco;
    GPtrArray * listaViagens;
} day;

typedef struct viagensDatas {
    int id_viagem;
    char *city;
    double distance_Ride;
    double tip;
} viagensDatas;
```

```
struct city_driver
{
    char* driver_name;
    char* idDriver;
    short avaliacao;
    int viagens;
};
```

A struct city_driver serve para auxiliar a query 7, possui o nome do driver, id, avaliação e número de viagens.

3.3. - Estrutura de dados dos users, drivers e rides

Em baixo, encontram-se as estruturas de dados que guardam os dados fornecidos CSV relativamente aos users, drivers e rides.

```c
struct user
{
    char *username;
    char *name;
    char gender;
    char *birth_date;
    char *account_creation;
    char account_status;
    double totalgasto;
    int distance;
    int viagens;
    int avaliacao;
    int last_ride;
};
```

```c
struct driver
{
    char *id;
    char *name;
    char *birth_date;
    char *account_creation;
    double totalgasto;
    int viagens;
    int avaliacao;
    int last_ride;
    char gender;
    char carclass;
    char account_status;
};
```

```c
struct ride
{
    char *data;
    char *idDriver;
    char *username;
    char *city;
    char *distance;
    char *score_user;
    char *score_driver;
    char *tip;
    int id;
};
```

4 Explicação das Queries

Neste trabalho prático, estão presentes 9 queries às quais temos como objetivo responder.

Query 1:

"Listar o resumo de um perfil registado no serviço através do seu identificador…"

Através das duas hashTables de Drivers e Users dependendo do comando em questão, retiramos as informações necessárias através de funções auxiliares para responder à query em questão.

Query 2:

"Listar os N condutores com maior avaliação média."

De forma a resolver a query 2, optamos por pré-calcular uma lista em que se encontram por ordem todos os Drivers de acordo com a sua avaliação. Assim, a execução da query ficará bastante mais rápida, já que o único trabalho da query é pegar nos N drivers pedidos pelo comando em questão.

Query 3:

"Listar os N utilizadores com maior distância viajada."

À semelhança da query 2, para resolver esta query, pré-calculamos uma lista em que se encontram por ordem todos os Users de acordo com a distância percorrida e todos os respetivos casos de desempate, desta forma já que a lista já está calculada apenas precisamos de pegar nos N users e apresenta-los.

Query 4:

"Preço médio das viagens (sem considerar gorjetas) numa determinada cidade…"

Para responder à query 4, utilizamos uma função que conta as viagens todas efetuadas na cidade dada e depois passa esse valor à variável 'viagens' da struct City e outra função que calcula a soma de todos os preços das viagens efetuadas na cidade abordada (tendo em conta os parâmetros que vão inferir na mesma, como a carclass e a distância da viagem). Esta última vai incrementando a variável 'preco' na struct City, de forma a que esta fique com o preço total das viagens no fim. Temos ainda uma terceira função que dada uma cidade dá o preço médio por viagem na mesma. Finalmente, temos uma função que dá o output da query 4 para o mesmo ser escrito no ficheiro de texto, ainda nesta função efetuamos a criação do ficheiro **commandX_output.txt.**

Query 5:

"Preço médio das viagens (sem considerar gorjetas) num dado intervalo de tempo…"

Para resolver a query 5, foram utilizadas 3 funções: conta_viagens_datas, conta_preco_total e precoMed_2datas. A função conta_viagens_datas, percorre a estrutura Day toda e conta o número de viagens entre duas datas passadas como argumentos. A função conta_preco_total, faz o mesmo que a anterior, mas retorna a soma dos preços entre duas datas. A função precoMed_2Datas calcula a divisão da função conta_preco_total e conta_viagem_2datas tendo em conta duas datas passadas como argumento de modo a obter o preço médio das viagens durante duas datas.

Query 6:

"Distância média percorrida, numa determinada cidade, representada por <city>, num dado intervalo de tempo…"

Para resolver esta query foi usada uma lógica muito idêntica à usada na query 5. São usadas também 3 funções: a função conta_viagens_2datas_city, conta_distanciaTotal_2datas_city e distMedia_2datas_city.

A função conta_viagens_2datas_city percorre a estrutura Day e soma o número de viagens entre duas datas numa determinada cidade. A função conta_distancia_Total_2datas_city também percorre a estrutura Day e soma a distância total entre duas datas numa determinada cidade. A função distMedia_2datas_city, devolve a divisão entre o resultado obtido na funções conta_distanciaTotal_2datas_city e conta_viagens_2datas_city de modo a obter a distância média percorrida num intervalo de tempo, numa determinada cidade.

Query 7:

"Listar os N condutores nas rides de uma cidade com a melhor avaliação."

Para resolver esta query a nossa estratégia era, no decorrer do parser Rides ir guardando num city_driver qual driver efetuou a ride, em que cidade a efetuou e que avaliação foi dada ao driver. Posteriormente, percorriamos a HashTable das City e organizamos a Glist de cada city, de maneira a ter o top Driver dessa cidade.

Depois a query só precisava de pegar na Glist da cidade desejada e mostar os N elementos.

Query 8:

"Listar todas as viagens em que o condutor e o utilizador são do género passado, ordenado de forma crescente em relação à data de criação da conta."

Nesta query percorresse a Glist Rides, elemento a elemento, e juntasse as Rides que respeitam os resquesitos desejados a uma Glist auxiliar. Depois basta organizar.

Query 9:

"Listar as viagens nas quais o passageiro deu grojeta ordenadas por distancia percorrida."

Para esta query nós previamente (após o parser das Rides) ordenamos a Glist Rides por datas. Na query basta percorrer a Glist até chegar à primeira ride que, à qual, a data corresponde à data mínima dada pela query, depois passamos a guardar os elementos da Glist Rides para uma Glist auxiliar, até a data da Ride ser superior à data máxima.

5 Otimizações

Em relação a Otimizações feitas. Fizemos algumas, começando por modificar o modo de funcionamento das queries 2 e 3. Anteriormente nós criávamos a lista com os N Users/Drivers, dentro destas queries, mas o problema disto era que para cada comando em que pedia uma destas queries, o programa iria criar uma lista de raiz para isto. Agora nós optamos por pré calcular ambas as listas antes de executar qualquer query, para que o trabalho da query em si seja bastante mais simplificado. Também melhoramos bastante a nível de encapsulamento que será explicado com mais detalhe num ponto mais à frente, este era um ponto em que na 1ª fase tivemos dificuldade em perceber o que realmente era, mas que nesta fase tivemos uma clara melhora neste ponto. Modificamos também a forma como guardávamos a informação do ficheiro rides.csv de uma HashTable para uma LinkedList, pois chegamos à conclusão que não tinhamos nenhuma vantagem significante em guardar em HashTable. Mudamos também alguns tipos das structs nomeadamente por exemplo, os ids das rides que eram anteriormente guardados em strings, passam a ser guardados inteiros, poupando assim alguma memória. Alteramos também bastantes secções onde usávamos strcat's sucessivos para sprintf's.

7 Encapsulamento

Em relação ao encapsulamento, a estratégia que adotamos, foi criar várias funções, denominadas por getters e setters. Sendo que os setter são responsáveis por dar à variável a que estamos a aceder um valor, valor este dependente dos argumentos da função em questão. Já os getters são responsáveis por dar o output do valor anteriormente guardado. Assim no resto do código as informações nas estruturas, apenas são acessíveis através destes getters, impedindo assim que um módulo tenha acesso a informações de outro módulo diretamente, promovento a abstração da informação.

8 Testes de Desempenho

Aqui optamos por analisar os tempos da 1º aparição em que obtemos o resultado correto de cada query, no input usado para o data-regular.

Queries/Elementos	Texec Rui Pinto	Texec Ricardo	Texec Hugo
Query 1	**0.000016**	**0.000015**	**0.000011**
Query 2	**0.0059**	**0.008855**	**0.009502**
Query 3	**0.000038**	**0.000078**	**0.000044**
Query 4	**0.000010**	**0.000024**	**0.000063**
Query 5	**0.000035**	**0.000035**	**0.000111**
Query 6	**0.107**	**0.167263**	**0.308728**
Query 7	**NULL**	**NULL**	**NULL**
Query 8	**0.786**	**1.123835**	**1.815204**
Query 9	**0.52**	**0.823169**	**1.055591**

Especificações

Rui Pinto : i7 / 16GB RAM

Hugo Arantes: Ryzen 73700X 32GB RAM

Ricardo Jesus : AMD RYZEN 7 / 8GB RAM

9 Conclusão

Concluindo pensamos que este projeto nos deu mesmo muitas bases para o nosso futuro relacionado ao parsing de informações de ficheiros, e principalmente em relação à importância da modularização e encapsulamento, e a forma como estes dois parâmetros facilitam todo o processo de programação. Aprendemos também bastante acerca de várias estruturas de dados e quais as vantagens quer de uma HashTable como de uma LinkedList.

Em relação a dificuldades no projeto, tivemos bastante dificuldade em tratar de todas as leaks e na poupança de memória do programa. Isto aconteceu muito por causa de muito do projeto já ter sido feito, e só depois conforme o projeto se ia estruturando é que percebíamos o quão mal estava construído na sua base, desde o parsing de informações aos tipos das structs, coisas que ainda tentamos melhorar mas que quando alterávamos levavam a erros em todo o projeto, tornando a modificação destes parâmetros muito mais difícil.

Se tivéssemos a oportunidade de fazer este projeto novamente teríamos feito muitas coisas diferentes, começando pela modificação do parser que se encontra demasiado complexo para algo que não deveria esta complexidade, o que prejudica também a inserção dos dados nas estruturas de dados, e impedindo que tudo isto ocorra da forma mais rápida e com menos custo a nível de memória possível. Porém pensamos que se isto acontece é sinal que a aprendizagem foi enorme de todos os elementos do grupo.