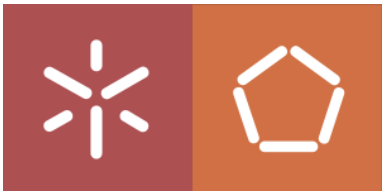
Relatório da 1ªFase de Laboratórios de Informática III

GRUPO 056

Novembro 2022



Universidade do Minho

Gonçalo Marinho

Luís Caetano

Maya Gomes

A90969, Gonçalo Duarte Lopes Marinho Gonçalves

A100893, Luís de Castro Rodrigues Caetano

A100822, Maya Gomes

Conteúdos

1 Introdução

Face à Unidade Curricular de Laboratórios de Informática III, do segundo ano da Licenciatura em Engenharia Informática, foi, por meio de um enunciado disponibilizado na Blackboard, proposto pelos docentes o desenvolvimento de um projeto na linguagem C. Este projeto consiste na capacidade de processar os ficheiros em formato csv disponibilizados pelos docentes. Perante estes ficheiros, através de queries, desenvolvemos estratégias de modo a organizar estas informações consoante o pedido em concreto em cada problema. Este projeto estará divido em duas fases, sendo este relatório relativo à primeira. Esta fase do trabalho tem por objetivo a implementação e estratégia de componentes correspondentes ao *Parsing* dos dados de cada entrada, ao modo de operações *batch*, três das nove queries descritas no enunciado do trabalho e ao catálogo de dados para os três ficheiros de entrada (*users.csv, drivers.csv e rides.csv*). Iremos também ter em consideração a modularidade e o encapsulamento das funções. Ao longo deste relatório vamos também desenvolver as estratégias que utilizamos para resolver os problemas que fomos encontrando e o que utilizámos ao nosso dispor (biblioteca GLib).

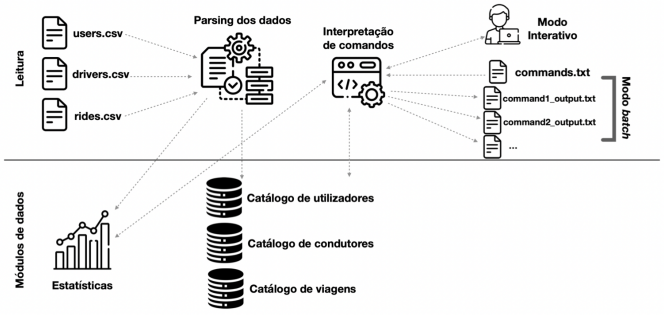


Figure 4: Arquitetura de referência para a aplicação a desenvolver.

2 Leitura e tratamento de dados

De modo a conseguir responder às queries propostas de forma eficiente, primeiro organizámos o grande volume de informação presente nos *Users*, nos *Drivers* e nas *Rides*. Desta forma, criámos catálogos diferentes para cada tipo de dados com a ajuda da biblioteca GLib. Inicialmente começamos por fazer um parsing que requeria ter os ficheiros todos abertos, guardando a informação à medida que era obtida. Isto não se verificou ser muito eficiente. Face a isto, dividimos o parsing dos dados em quatro funções diferentes, criando assim 3 catálogos diferentes.

2.1 Parsing dos dados

2.1.1 parsing-users.h

```
typedef struct users{
    char* username;
    char* name;
    char* gender;
    char* birth_date;
    char* account_creation;
    char* account_status;
}User;
```

Figure 5: Struct users

Esta estrutura guarda a informação de um **User**, sendo esta composta por *username, name, gender, birth date, account creation e account status*. O método de pagamento não é incluído, pois não é utilizado nas queries.

2.1.2 parsing-drivers.h

```
typedef struct drivers{
    char* id;
    char* name;
    char* birth_date;
    char* gender;
    char* car_class;
    char* license_plate;
    char* city;
    char* account_creation;
    char* account_status;
}Driver;
```

Figure 6: Struct drivers

Esta estrutura guarda a informação de um **Driver**, sendo esta composta por *id, name, birth date, gender, car class, license plate, city, account creation e account status*.

2.1.3 parsing-rides.h

```
typedef struct rides{
    char* id;
    char* date;
    char* driver;
    char* user;
    char* city;
    char* distance;
    char* score_user;
    char* score_driver;
    char* tip;
}Ride;
```

Figure 7: Struct rides

Esta estrutura guarda a informação de uma **Viagem**, sendo esta composta por *id, date, driver, user, city, distance, score user, score driver, tip, comment*. Os comentários não entram na estrutura, pois esse tipo de dados não são necessários para as diversas queries.

2.2 Interpretação dos dados

2.2.1 Instrucoes.h

```
typedef struct instruction{
    int query;
    char* id;
}Inst;
```

Figure 8: Struct instruction

Esta estrutura guarda a informação de uma **Instrução** relativamente à query que deve executar, sendo esta composta pelo número da *query e id*.

3 Desenvolvimento do projeto - Queries

3.1 Query 1

Esta query tem por objetivo listar o resumo de um perfil registado no serviço através do seu identificador, representado por *ID*. Para ambas as estruturas (*Users* e *Drivers*), optamos pela criação de HashTables visto que estas possuem a vantagem de associar valores a keys. Ou seja, a partir de uma key, neste caso o *ID* para os *drivers* e o *username* para os *users* é possível desenvolver uma pesquisa rápida afim de obter todas as informações sobre a pessoa desejada no que toca à informação presente. No que toca à idade, decidimos converter tanto a data de nascimento como a data que utilizamos como base em dias e a partir daí dividir por 365 enquanto o resto fosse 0, deste modo conseguimos calcular a idade. Já para calcular a informação que se encontra no ficheiro rides.csv, utilizamos a Queue como método de armazenamento de informação. O nosso programa vai percorrer esta Queue de forma a guardar o *score* atribuído ou ao *user* ou ao *driver*, assim como o total gasto/auferido, que vai sempre incrementando. Também possui um contador que vai incrementando em 1 valor sempre que há uma correspondência, de forma a contar o número de viagens. Posto isto, no fim, obtemos todos os valores necessários, menos a avaliação média, o que basta dividir o total de avaliações pelo número de viagens e obter-se-á o resultado.

3.2 Query 2

```
typedef struct topdriv{
    double av_med;
    int n_viagens;
    char* data;
    char* id;
}TopDriv;
```

Figure 9: Struct topdriv da query 2

Esta query tem por objetivo listar os N condutores com maior avaliação média. Para isso, recebemos um inteiro N que corresponde ao número de condutores que serão listados. Primeiramente, enquanto percorremos o ficheiro rides.csv, simultaneamente com o parsing das rides para a respetiva Queue, guardamos na nossa estrutura HashTable a soma das avaliações que são atribuídas ao condutor pelos clientes e vamos sempre somando o valor no parâmetro *av_med*, já no parâmetro *n_viagens* vamos incrementando o

valor em 1 valor sempre que há uma correspondência. Ao atribuir o ID como key da HashTable, sempre que aparece este ID basta atualizar os valores lá presentes. Finalmente, criámos uma função que altera toda a HashTable dividindo a soma total pelo número de viagens total fazendo com que o resultado seja a avaliação media. Para além disso, criámos um array de pointers para a struct inserida no valor da HashTable de forma a ordenar todos os valores da HashTable e uma função que compara todos os valores, primeiro pela avaliação média, depois pela data da última viagem feita, e finalmente pela ordem alfabética do ID do condutor. Finalmente, depois do array estar ordenado, imprime-se as suas N últimas posições que a query pedir.

3.3 Query 3

```
typedef struct topuser{
    int distancia;
    char* data;
    char* id;
}TopUser;
```

Figure 10: Struct topuser da query 3

Esta query tem por objetivo listar os N utilizadores com maior distância viajada. Para isso, recebemos um inteiro N correspondente ao número de utilizadores que serão listados. Primeiramente, decidimos criar uma struct onde armazenamos a soma das distâncias sempre que o utilizador aparece, assim como data, onde vamos atualizando conforme a mais recente, e mantemos sempre o *username* do utilizador, para depois poder ir buscar o seu nome à HashTable dos *users*, quando necessário. Para além disso, criamos uma função que compara as distâncias, depois a data da última viagem e, finalmente a ordem alfabética do *username* do utilizador, isto em caso de empate em alguma delas, ordenando de forma idêntica ao método utilizado na Query 2.

3.4 Query 4

Esta query tem por objetivo obter o preço médio das viagens numa determinada cidade que representamos por *city*. Enquanto se percorre as rides, é dada o nome de uma cidade como key de forma a armazenar os valores pretendidos. Estas correspondem à nossa struct que possui o número de viagens e a soma dos preços. Desta forma, quando é processada a query, ao

```
typedef struct pmedcidade{
    double preco;
    int n_viagens;
}PMed;
```

Figure 11: Struct pmedcidade da query 4

fornecer o nome da cidade, que será key da HashTable, rapidamente obtemos a informação que pretendemos, bastando dividir o preço acumulado pelo total de viagens. Assim, obtemos o preço médio das viagens.

4 Testes de Performance

4.1 Tempos de execução

As nossas funções calculam as queries todas sempre que nos é pedido uma delas, pelo que, os tempos de execução são semelhantes em todas, derivado ao facto de o input dado ser constituído por todas as queries.

Table 1: Tempos de Execução

Sistema Operativo	Processador	Armaze -namento	RAM	Query	Argumentos	Tempo de Exec (s)
Linux Manjaro 5.3.19	i7-6500U	SSD	8GB	1	1 SaCruz110	4.264
Linux Manjaro 5.3.19	i7-6500U	SSD	8GB	2	2 10	4.287
Linux Manjaro 5.3.19	i7-6500U	SSD	8GB	3	3 10	4.648
Linux Manjaro 5.3.19	i7-6500U	SSD	8GB	4	4 Braga	4.253
Linux Mint 21	i7 4thGen	SSD	8GB	1	1 SaCruz110	3.636
Linux Mint 21	i7 4thGen	SSD	8GB	2	2 10	3.591
Linux Mint 21	i7 4thGen	SSD	8GB	3	3 10	4.003
Linux Mint 21	i7 4thGen	SSD	8GB	4	4 Braga	3.576
MacOS Monterey	i5 Quad Core	SSD	8GB	1	1 SaCruz110	5.836
MacOS Monterey	i5 Quad Core	SSD	8GB	2	2 10	5.930
MacOS Monterey	i5 Quad Core	SSD	8GB	3	3 10	6.148
MacOS Monterey	i5 Quad Core	SSD	8GB	4	4 Braga	6.032

5 Apreciação crítica e limitações

O nosso código acabou por apresentar demasiadas *memory leaks*, e identificá-los e corrigi-los acabou por se tornar num desafio que enfrentámos ao longo do trabalho, não conseguindo livrar-nos na totalidade dos mesmos. Outro problema que enfrentamos foi o facto de não conseguirmos consolidar bem o compromisso entre segurança e rapidez. Nesta fase do trabalho, estamos a calcular as queries todas sempre que queremos uma delas, pensamos, no entanto, que para a segunda fase poderíamos arranjar uma forma de calcular somente a que queremos e não todas, assim como projetar uma estratégia de forma a torná-las mais eficientes e rápidas.

6 Conclusão

Notamos assim que o grande desafio deste projeto era a programação à larga escala, com base numa grande quantidade de dados disponíveis. Ou seja, exigiu-nos a obrigação de trabalhar com estruturas de dados e algoritmos que se desenvolvessem tanto em complexidade assim como em eficiência, de modo a que pudéssemos concretizar o nosso objetivo. Isto obrigou-nos a explorar e a conhecer uma nova API (GLib) e consequentemente ter em consideração que métodos é que são mais eficientes para cada caso. Consideramos que, em paralelo com esta ideia, o projeto tinha também como um dos objetivos consolidar,num contexto mais prático, o conhecimento obtido em outras unidades curriculares, nomeadamente Programação Imperativa (do 1ºano) e Algoritmos e Complexidade (do 2ºano).