Laboratórios de informática III

Relatório da Fase 1

Afonso Laureano Barros Amorim A97569
Bianca Araújo do Vale A95835
Matilde Maria Ferreira de Sousa Fernandes A95319


A97569


A95835


A95139

Grupo 18
2022/2023
Universidade do Minho

Índice

1 Introdução

No âmbito da Unidade Curricular de Laboratórios de Informática III do segundo ano da licenciatura em Engenharia Informática, foi-nos proposta, nesta primeira fase, a implementação de um parser em C que validasse as linhas dos três ficheiros *.csv* (users,drivers e rides) que nos foram disponibilizados.

Foi-nos também sugerida a implementação de pelo menos três *queries*. Dessa forma, optamos pela *query* um, quatro, cinco e seis.

Para a realização desta fase, aplicamos os conhecimentos essenciais da linguagem C e Engenharia de *Software*, principalmente, modularidade e encapsulamento.

2 Solução Adotada

Após ler o enunciado do presente projeto, disponibilizado pelos docentes da UC, decidimos que a solução iria ser criar catálogos para os *users*, *drivers* e *rides* e percorrê-los de forma iterativa, utilizando funções da biblioteca *glib*, de modo a conseguirmos aceder facilmente a todas as informações necessárias que estão nos ficheiros *.csv*, que precisamos para a realização das *queries*.

3 Estruturação do Projeto

3.1 Users

Criamos a *struct users* com os seguintes tipos para poder armazenar a informação contida numa linha do ficheiro *users.csv*. Utilizamos depois um catálogo *users* construido com base na *struct users* com o objetivo de conseguirmos encontrar e manipular, de modo eficiente, informações sobre um certo *user*, através do seu *username*, que é a *key* da *hashtable*. Deste modo, decidimos armazenar a informação dos *users* através do uso de *hashtables*, pois achamos que é a maneira mais eficiente de implementar o objetivo descrito.

```
struct user {
  char* username;

  char* name;

  char gender;

  Date birth_date;

  Date account_creation;

  char* pay_method;

  char* account_status;
};
```

3.2 Drivers

Criamos a *struct driver* com os seguintes tipos para poder armazenar a informação contida numa linha do ficheiro *drivers.csv*. Recorremos ao uso de um catálogo *drivers* construído com base na *struct driver* com o objetivo de encontrarmos e manipularmos, eficientemente, informações sobre um certo *driver*, através do seu *id* que é a *key* da *hashtable*. Deste modo, decidimos armazenar a informação dos *drivers* através do uso

de *hashtables*, pois, tal como nos *users*, achamos que é a maneira mais eficiente de implementar o objetivo descrito.

```
struct driver {

  int id;

  char* name;

  Date birth_day;

  char gender;

  char* car_class;

  char* license_plate;

  char* city;

  Date account_creation;

  char* account_status;

};
```

3.3 Rides

Criamos a *struct rides* com os seguintes tipos para poder armazenar a informação contida numa linha do ficheiro *rides.csv*. Decidimos usar um catálogo *rides* contruido com base na *struct rides* com o objetivo de conseguirmos encontrar e manipular, de modo eficiente, informações sobre uma certa *ride*, através do seu *id*, que é a *key* da *hashtable*. Deste modo, decidimos armazenar a informação das *rides* através do uso de *hashtables*, pois achamos que é a maneira mais eficiente de implementar o objetivo descrito.

```
struct rides {

  int id;

  Date date;

  int driver;

  char* user;

  char* city;

  int distance;

  double score_user;

  double score_driver;

  double tip;

  char* comment;

};
```

4 Queries

Para a resolução das *queries* decidimos utilizar uma estratégia semelhante em todas, que passa por recorrer a uma hastable à qual iremos aceder para retirar valores necessários para calcular pedidos em cada query

4.1 Query 1

Esta primeira *query* é responsável por listar o resumo de um perfil registado no serviço através do seu identificador, representado por <ID>. Um aspeto importante a ter em conta é o facto de que o identificador pode corresponder a um utilizador ou a um condutor.

Visto que um utilizador corresponde a um *char* username* no ficheiro *users.csv* e um condutor corresponde a um *int id* no ficheiro *drivers.csv*, decidimos criar duas funções: a *q1users* e a *q1driver*, respetivamente.

Na função *q1users*, inicialmente, através da função auxiliar *q1u*, percorremos a *hashtable* dos *users* através da função *g_ hash_ table_ lookup* e guardamos numa *struct* User u os valores do *user* que foi pedido como argumento.

De seguida, decidimos estruturar da seguinte maneira:

- Para obter o nome do *user* pedido utilizamos a função *getUserXName*;
- Para obtermos o género do *user* pedido utilizamos a função *getUserGender*;
- Para conseguirmos obter a idade do *user* pedido utilizamos a função *getBirthDate* para sabermos a data de nascimento do *user* e uma função auxiliar *idade*, que dada a data de nascimento do *user* calcula a sua idade;
- Calculamos a avaliação média do *user* pedido através da função *score_ user_ medio*, que vai iterar todas as *rides* da *hashtable* das *rides* e para cada *ride*, se o *username* associado à mesma for igual ao *username* do *user* pedido que obtemos através da função *getUsername*, incrementamos o valor da variável *avaliação_ m.*No final a variável vai conter valor dado pela soma do scoreuser de todas as *rides* que tenham o *user* pedido associado. Por fim, o valor da avaliação média é dada pela divisão dessa variável pelo número de *rides* que tenham o *username* pedido associado.
- Para obter o número de viagens do *user* pedido utilizamos a função *numero_ viagens_ users* que vai iterar todas as *rides* da *hashtable* das *rides*. E para cada *ride*, se o *username* associado à mesma for igual ao *username* do *user* pedido que obtemos através da função *getUsername*, incrementamos 1 ao número de viagens que é dado pela variável *viagens* ;
- De modo a alcançarmos o total gasto, precisamos de saber a classe do carro e o número de kilometros que percorreu. A classe do carro obtemos através da função *car_ class* que percorre a *hashtable* dos *drivers* através da função *g_ hash_ table_ lookup* e devolve uma *string* com a classe do carro associada ao *id* do *driver*, passado como argumento. O valor do total gasto vai ser dado pela função *total_ gasto* que vai iterar todas as *rides* da *hashtable* das *rides* e para cada *ride*, se o *username* associado à mesma for igual ao *username* do *user* pedido que obtemos através da função *getUsername*, vamos incrementar o total gasto que é dado pela varável *d* consoante a classe do carro os número de kilometros e a tip associados à *ride* em questão.

4.2 Query 4

Esta *query* é responsável por obter o preço médio das viagens numa determinada cidade, que é representada por <city>.

Para obter o preço médio por cidade é igualmente necessário saber a classe do carro e o número de kilometros. A obtenção da classe do carro é feita como na *query* 1 e o valor do preço médio vai ser dado pela função *preco_ medio* que vai iterar todas as *rides* da *hashtable* das *rides* e para cada *ride*, se a *city* associada à mesma for igual à *cidade* passada como argumento, vamos incrementar o valor da variável *p* consoante a classe do carro e o número de kilometros que estão associados à *ride* em questão. No final, o valor contido na variável *p* vai ser dividido pelo número de *rides* que tenham a *city* pedida associada.

4.3 Query 5

A resolução desta *query* consiste em apresentar o preço médio das viagens num dado intervalo de tempo, sendo que esse intervalo é representado por <data A> e <data B>.

Para obtermos o preço médio entre datas é necessário saber a classe do carro e se uma certa data está entre outras duas. A obtenção da classe do carro é feita como na *query* 1 e para saber se uma data está entre outras duas fizemos a função *entre_ datas*. O preço médio entre datas é calculado através da função *preço_ médio_ entre_ datas* que vai iterar todas as *rides* da *hashtable* das *rides* , e para cada *ride* vai verificar se a data da *ride* se encontra entre as datas dadas como argumento, com a função *entre_ datas*. Caso esteja, vamos incrementar o valor da variável *p* consoante a classe do carro e o número de kilometros que estão associados à *ride* em questão. No final, o preço médio entre datas vai ser dado pela divisão de *p* pelo número de *rides* que têm a data entre as duas datas dadas como argumento.

4.4 Query 6

A *query* 6 é responsável por devolver a distância média percorrida, numa determinada cidade, que se representa por <city>, num dado intervalo de tempo, que é representado por <data A> e <data B>.

Para calcular a distância média percorrida, numa determinada cidade entre duas datas, vamos usar a função *distancia_ medio* que vai iterar todas as *rides* da *hashtable* das *rides* e para cada *ride*, se a *city* associada à mesma for igual à *cidade* passada como argumento, incrementamos o valor da variável *dist*. No final, a variável vai conter o valor dado pela soma de todas as *distance* de todas as *rides* que tenham a *city* pedida associada. Por fim, o valor da distância média percorrida, numa determinada cidade entre duas datas é dado pela divisão dessa variável pelo número de *rides* que tenham a *city* pedida associada.

5 Conclusão

Em modo de conclusão, acreditamos que com esta fase do projeto, conseguimos implementar os objetivos descritos no enunciado.

Com a realização deste trabalho, acreditamos que conseguimos ganhar mais conhecimentos sobre o encapsulamento e modularidade, bem como uma maior atenção com a organização do código, pois torna mais simples a verificação e correção de erros, bem como a estruturação do trabalho, sendo vantajoso para analisar o código mais tarde.

Para o grupo, a implementação de certas *queries* foi um pouco desafiante, pois a solução obtida é diferente da esperada.