Laboratórios de informática III

Relatório Final

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 a implementação de um sistema que seja capaz de dar resposta a *queries* pedidas, sobre a leitura de um conjunto de dados fornecidos em formato de texto.

Na primeira fase foi-nos proposta a implementação de um *parser* em C, que validasse as linhas dos três ficheiros *.csv* (*users*, *drivers* e *rides*) que nos foram disponibilizados. Esta fase passaria também pela implementação de pelo menos três *queries*, das quais o grupo decidiu implementar a um, quatro, cinco e seis.

Nesta segunda e última fase, o objetivo era efetuar o modo de operação interativo, que inclui um menu de interação com o programa, e testes para avaliar o tempo de execução de cada *query*.

Nesta fase tivemos também de dimensionar a solução para ficheiros de entrada com uma ordem de grandeza superior e realizar as restantes *queries*, sendo que decidimos implementar todas as restantes.

O maior desafio com que nos deparamos na realizção do trabalho foi, sem dúvida, saber como criar e ordenar as listas ligadas para as *queries* de ordenação, de modo que o tempo de execução fosse o menor possível.

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

2 Solução Adotada

Inicialmente, decidimos que a solução seria 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*.

Posteriormente, nesta segunda fase, decidimos adicionar variáveis de estatística às *structs* dos *drivers* dos *users* e das *rides*, por exemplo, a variável da *struct* dos *drivers* 'total_auferido'. À medida que fazemos *parsing* dos ficheiros, atualizamos essas variáveis. Toda esta estratégia implementada permitiu um melhor desempenho a nível de tempo de execução das *queries*.

Nas *queries* em que era necessária fazer ordenação, construímos uma lista ligada a partir da *hashtable*, que posteriormente ordenamos.

3 Arquitetura do Sistema

A arquitetura do sistema é definida por cátalogos de *Users*, *Drivers* e *Rides*, cujos dados são retirados através da leitura dos ficheiros de texto Users.csv, Drivers.csv e Rides.csv, respetivamente e uma interface que permite a comunicação máquina-utilizador (Modo interativo).

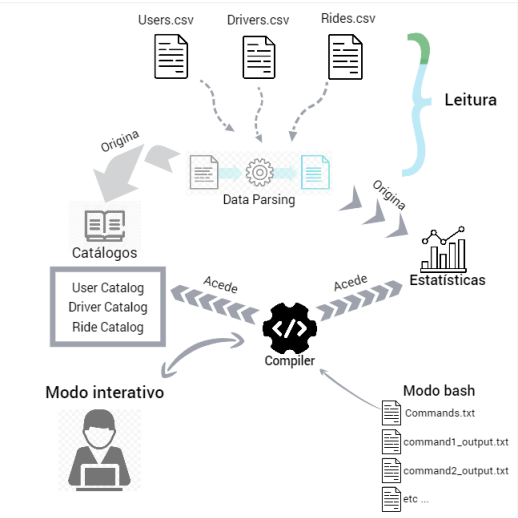

Figure 1: Arquitetura do Sistema

4 Estruturação do Projeto

4.1 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.

Na segunda fase, decidimos adicionar a esta *struct* e as variáveis *'total_gasto'*, *'score_u'*, *'n_viagensU'* e *'dist_t'*. Estas variáveis são atualizadas à medida que é feito o parsing dos ficheiros *.csv*. Fizemos esta adição com o objetivo de diminuir o tempo de execução das *queries*.

```
struct user {
  char* username;

  char* name;

  char gender;

  Date birth_date;

  Date account_creation;

  char* pay_method;

  char* account_status;
```

```
15
16    double total_gasto;
17
18    double score_u;
19
20    int n_viagensU;
21
22    int dist_t;
23 };
```

4.2 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.

Com o intuito de diminuir o tempo de execução das *queries*, decidimos, na última fase do projeto, implementar as seguintes variávies: *'total_auferido'*, *'score_d'*, *'n_viagensD'*, *'v_recente'*, *'avaliacao_cidade'* e *'viagens_cidade'*, estas variáveis são atualizadas á medida que é feito o parsing dos ficheiros *.csv*.

```
1  struct driver {
2
3    int id;
4
5    char* name;
6
7    Date birth_day;
8
9    char gender;
10
11   char* car_class;
12
13   char* license_plate;
14
15   char* city;
16
17   Date account_creation;
18
19   char* account_status;
20
21   double total_auferido;
22
23   double score_d;
24
25   int n_viagensD;
26
27   Date v_recente;
28
29   double avaliacao_cidade;
30
31   int viagens_cidade;
32 };
```

4.3 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.

A esta *struct*, decidimos implementar as variávies *account_creation_driver* e *account_creation_user*. Estas variáveis são atualizadas à medida que é feito o parsing dos ficheiros *.csv*. Adicionamos estas variáveis de modo a diminuir o tempo de execução das *queries*.

```
struct rides {

  int id;

  Date date;

  int driver;

  char* user;

  char* city;

  int distance;

  double score_user;

  double score_driver;

  double tip;

  char* comment;

  Date account_creation_driver;

  Date account_creation_user;

};
```

5 Modo Interativo

Quando o utilizador executa o programa em modo interativo (*./programa-principal ./entrada*), é lhe apresentado um Menu, onde será possível observar todas as *queries* implementadas.

Sempre que o utilizador quiser sair do programa, basta escrever '0' e dar **enter**. Se desejar executar uma *query*, tem de escrver o número da *query* e os argumentos da mesma.

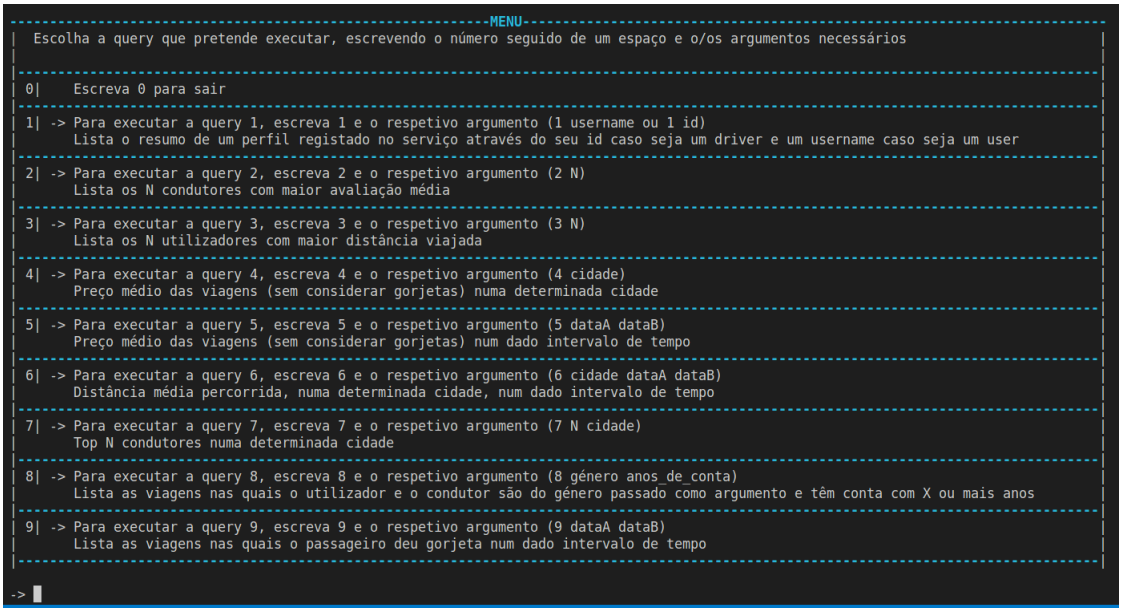

Figure 2: Menu interativo

6 Modo Batch

Quando o utilizador executa o programa em modo batch (*./programa-principal ./entrada entrada/input.txt*), o programa irá executar as *queries* escritas no ficheiro *input.txt* à medida que o lê linha a linha. Para cada querie vai ser gerado um ficheiro dentro da pasta *Resultados* com o seu output.

7 *Makefile*

```
# Name of the project
ALL: programa-principal programa-testes
PRINCIPAL: programa-principal
TESTE: programa-testes

# Compiler and linker
CC=gcc

# Flags for compiler
CC_FLAGS= -W        \
          -Wall     \
          -g
#Glib
GLIB= `pkg-config --cflags glib-2.0`

# Command used at clean target
RM = rm -rf

SRC_DIR = src
SRC_T = testes
BLD_DIR = build
INC_DIR = include
BLD_DIR = build

SRC = $(wildcard $(SRC_DIR)/*.c)
SRC_TESTS = $(wildcard $(SRC_T)/*.c)
OBJS= $(patsubst $(SRC_DIR)/%.c,$(BLD_DIR)/%.o,$(SRC))
DEPS = $(patsubst $(BLD_DIR)/%.o,$(BLD_DIR)/%.d,$(OBJS))
FILES = src/date.c src/drivers.c src/prints.c src/queries.c src/rides.c src/users.c

INCLDS = -I $(INC_DIR)

vpath %.c $(SRC_DIR)

programa-principal:
    @mkdir Resultados
    @mkdir Resultados-esperados
    gcc `pkg-config --cflags glib-2.0` $(SRC) $(CC_FLAGS) `pkg-config --libs glib-2.0` -o programa-principal

programa-testes:
    gcc `pkg-config --cflags glib-2.0` $(SRC_TESTS) $(FILES) $(CC_FLAGS) `pkg-config --libs glib-2.0` -o programa-testes

clean:
    @ $(RM) *.o programa-principal programa-testes *~ Resultados *~ Resultados-esperados
```

Figure 3: Makefile

7.1 Estrutura

A *makefile* permite-nos compilar o nosso programa, de modo a conseguirmos executá-lo.

Vamos falar um pouco sobre as nossas variávies, assim como o que cada uma contém.

- A variável **ALL** contém o *programa-principal* e o *programa-testes*;
- A variável **PRINCIPAL** contém o *programa-principal*;
- A variável **TESTE**, o *programa-testes*.
- A variável **CC** guarda o compilador que estamos a utilizar, o *gcc*.
- A variável **CC_FLAGS** guarda todas as *flags* que usamos no compilador.
- Na variável **GLIB** guardamos a biblioteca que usamos.

7.2 Comandos disponíveis

7.2.1 make

Comando que compila o programa criando assim os ficheiros executáveis (programa-principal e programa-testes) e as pastas *Resultados* e *Resultados-esperados* .

7.2.2 make clean

Sempre que executado, este comando remove todos os ficheiros criados durante a compilação do programa, os ficheiros executáveis (programa-principal e programa-testes), e ainda todos ficheiros desnecessários craidos por editores de texto e, por fim, apaga a diretoria *Resultados* e *Resultados-esperados*.

8 *Queries*

8.1 *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 do identificador poder 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;
- Nesta fase, de modo a alcançar a avaliação média do *user*, dividimos o resultado da *getScore_u* pelo resultado da *getn_viagensU*;
- De forma a conseguirmos obter o número de viagens, utilizamos a função *getn_viagensU*; 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 *getTotal_gasto*.

Na função *q1drivers*, inicialmente, através da função auxiliar *q1d*, percorremos a *hashtable* dos *drivers* através da função *g_hash_table_lookup* e guardamos numa *struct* Driver d os valores do *driver* que foi pedido como argumento.

De seguida, decidimos estruturar da seguinte maneira:

- Para obter o nome do *driver* pedido utilizamos a função *getDriverName*;
- Para obtermos o género do *driver* pedido utilizamos a função *getDriverGender*;
- Para conseguirmos obter a idade do *driver* pedido utilizamos a função *getBirthDay* para sabermos a data de nascimento do *driver* e uma função auxiliar *idade*, que dada a data de nascimento do *driver* calcula a sua idade;
- Nesta fase, de modo a alcançar a avaliação média do *driver*, dividimos o resultado da *getScore_d* pelo resultado da *getn_viagensD*;
- De forma a conseguirmos obter o número de viagens, utilizamos a função *getn_viagensD*; O valor do total auferido vai ser dado pela função *getTotal_auferido*.

8.2 *Query* 2

A segunda *query* é responsável por listar os N condutores com maior avaliação média.

Nesta *query* começamos por usar a função *'sort_ maior_ avaliacao_ media'*, onde aplicamos a função *'g_ hash_ table_ get_ values'* que retorna uma lista ligada com todos os valores armazenados da *hastable* dos *drivers* e, de seguida, usamos a função *'g_ list_ sort'*, que recebe, como parâmetro, a lista ligada dos *drivers* e uma função de comparação (*'compara_ avaliacao_ media'*), que é usada para determinar a ordem dos elementos. Após isto tudo, percorremos a lista ligada ordenada e vamos dando *print* para um ficheiro, dos N primeiros condutores.

8.3 *Query* 3

Nesta *query* pretende-se listar os N utilizadores com maior distância viajada.

Na terceira *query*, começamos por utilizar a função *'sort_ dist_ t'*, onde aplicamos a função *'g_ hash_ table_ get_ values'* que retorna uma lista ligada com todos os valores armazenados da *hastable* dos *users* e, de seguida, usamos a função *'g_ list_ sort'*, que recebe, como parâmetro, a lista ligada dos *users* e uma função de comparação (*'compare_ dist'*), que é usada para determinar a ordem dos elementos.

Após isto tudo, percorremos a lista ligada ordenada e vamos dando *print* para um ficheiro, dos N primeiros utilizadores.

8.4 *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.

8.5 *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.

8.6 *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.

8.7 *Query* 7

Esta *query* tem como objetivo devolver o top N de condutores numa determinada cidade, ordenado pela avaliação média do condutor.

Na presente *query* começamos por utilizar a função *'sort_dist_t'*, onde aplicamos a função *'g_hash_table_get_values'* que retorna uma lista ligada com todos os valores armazenados da *hastable* dos *drivers* e, de seguida, usamos a função *'g_list_sort'*, que recebe, como parâmetro, a lista ligada dos *drivers* e uma função de comparação (*'compara_avaliacao_media'*), que é usada para determinar a ordem dos elementos.

Após isto tudo, percorremos a lista ligada ordenada e vamos dando *print* para um ficheiro, dos N primeiros utilizadores.

8.8 *Query* 8

Na *query 8* era pedido para listar todas as viagens nas quais o utilizador e o condutor são do género passado como parâmetro, e têm perfis com X ou mais anos. O output desta *query* tem de estar ordenado de forma que os *drivers* com as contas mais antigas apareçam primeiro. Na implementação desta *query* utilizamos a função *'sort_conta_mais_antiga'*, onde chamamos a função *'list_rides_8'*, de forma a obtermos uma lista ligada com todas as *rides* onde o utilizador e o condutor têm o género passado como parâmetro, e perfis com X ou mais anos. De seguida aplicamos a função *'g_list_sort'* à lista ligada criada e usamos a função *'conta_mais_antiga'* como função de comparação.

Após isto tudo, percorremos a lista ligada ordenada e vamos dando *print* para um ficheiro dos dados dos *drivrs* e dos *utilizadores* relacionados com cada *ride* presente na lista ligada.

8.9 *Query* 9

Nesta *query*, o objetivo é listar as viagens nas quais o passageiro deu gorjeta, no intervalo de tempo (data A, data B). O output deve ser ordenado por ordem de distância percorrida (em ordem decrescente). Temos uma função *'sort_menor_dist'* que chama a função *'list_rides_9'* de modo a obter uma lista ligada com todas as *rides* nas quais o passageiro deu gorjeta, no intervalo de tempo passado como argumento. Depois de obtida a lista com a função *'g_list_sort'* ordenamos a lista ligada tendo como função de comparação o *'compara_distancia'*.

Após isto tudo, percorremos a lista ligada ordenada e vamos dando *print* para um ficheiro dos dados das *rides* e dos presente na lista ligada.

9 Testes de Performance

Queries	Tempo de Execução
Parsing	2.527722
Query 1 - *drivers*	0.000018
Query 1 - *users*	0.000080
Query 2	0.010658
Query 3	0.586100
Query 4	0.121206
Query 5	0.418517
Query 6	0.104340
Query 7	0.164783
Query 8	0.834333
Query 9	0.169969

Table 1: Tempos de execução das *queries* em segundos com os ficheiros *data regular errors*

10 Conclusão

Em suma, os resultados obtidos ao longo da realização do presente projeto, foram, de certo modo, bastante satisfatórios, visto que o mesmo foi realizado em tempo útil e conseguimos implementar na totalidade, todas as *queries*.

Com a realização deste trabalho, acreditamos que conseguimos ganhar mais conhecimentos sobre o encapsulamento e modularidade, bem como uma maior atenção na organização do código, pois torna mais simples a verificação e correção de erros, assim como a estruturação do trabalho. No futuro, tudo isto pode configurar uma grande vantagem uma vez que são pontos fundamentais que nos facilitam a posterior analise do código.

Para o grupo, a implementação de certas *queries* foi um pouco desafiante e estimulador, pois tivemos de implementar diversos métodos e estratégias de forma a implementarmos as *queries* da melhor forma, de maneira a obter o melhor tempo de execução. Apesar de todas as adversidades, acreditamos que atingimos os objetivos.

Assim, terminado este projeto, o grupo conseguiu perceber os diferentes aspetos no funcionamento de programação em larga escala, uma vez que passaram pelo nosso programa uma grande quantidade de dados, aumentando assim a complexidade do trabalho.