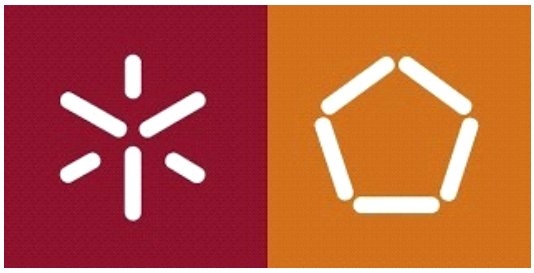

Universidade do Minho

Laboratórios de Informática III

Relatório grupo 72 – Fase 2

3 de fevereiro de 2022

José Costa (a100749)

Jorge Teixeira (a100838)

Nuno Aguiar (a100480)

Índice:

Introdução

Nesta Unidade Curricular, foi-nos proposto criar um programa, em linguagem de programação C, capaz de ler ficheiros de formato .CSV para, através de mecanismos eficientes (como modularidade e encapsulamento, estruturas dinâmicas de dados, etc.), responder a determinados problemas propostos. Estes problemas foram respondidos em dois modos de operação:

- ***batch:*** que no momento de execução é dado como parâmetro o path para os datasets e um path para um ficheiro com as inputs a serem executadas, sendo os resultados destas inputs escritos numa pasta com este nome.

- *interativo*: ao executar o programa sem parâmetros, será iniciada uma UI, criada através da biblioteca 'ncurses', e será pedida a inserção do path para o dateset e em seguida poderam ser executadas as diferentes queries.

Relativamente à primeira fase, decidimos converter o projeto para usar glib, visto que, a segunda fase requer estruturas de dados mais complexas e a glib torna mais fácil lidar com estas.

Estruturas de armazenamento dos Dados:

```
struct data {
   GArray *drivers;
   GHashTable *users;
   GList *ordDrivers;
   GList *ordUsers;

   GList *rides;
   GHashTable *ridesCity;
   GHashTable *topDrivCity;
};
typedef struct data *DATA;
```

Esta struct tem como objetivo armazenar todas as estruturas de dados necessárias para a execução das queries, estando por isso presente com parâmetro destas. Sendo que, que estas estruturas são criadas por uma função que os lê os ficheiros e carrega para a memória a informação necessária.

***GArray*drivers* ->** Esta estrutura guarda as informações relativas aos drivers, em que o id do driver indica o índice em que a sua informação será guardada. Utilizado na query 1.

***GHashTable *users* ->** Guarda toda a informação dos users, que são inseridos na hashtable, sendo que esta é inicializada com g_str_hash e com g_str_equal e como key é usado o username do user. Utilizada na query 1.

***GList *ordDrivers / GList *ordUsers* ->** Estas estruturas são listas ligadas, implementadas com auxílio da glib e que guarda todos os drivers e users, ordenados consoante as dependências das queries 2 e 3 respetivamente.

GList *rides -> Esta GList guarda todas as rides do programa ordenadas pela data, em que estas aconteceram. Utilizado nas queries 5, 8 e 9.

GHashTable *ridesCity -> Esta hashtable guarda as rides sendo que é usado como key cidades e a essas keys estão associdas GList com todas as rides relativas a essa cidade. Utilizado nas queries 4 e 6.

GHashTable *topDrivCity -> Esta hashtable recebe, como a anterior, como key cidades, mas desta vez o value são listas dos top Drivers da cidade correspondente.

Todas as funções, para interagir com estas estruturas, estão definidas no datastuc.c. Sendo estas funções para inserir elementos, para os procurar e por fim para libertar todo o espaço alocado para estas estruturas e sendo criadas e iterados no ficheiro loadfiles.c (ficheiro onde é efetuado o parser dos ficheiros).

Organização do projeto:

O nosso trabalho está dividido por vários módulos devidamente encapsulados e divididos consoante as respetivas struct necessárias. Sendo que nos ficheiros users.c, drivers.c, rides.c e datastruc.c têm os respetivos sets e gets para ser possível extrair a informação.

Assim, temos:

- **main.c** – main do nosso programa principal.

.

- **main_teste.c** – main do nosso programa que realiza os testes de desempenho.
- **loadfiles.c** – contém funções necessárias para carregar e processar os dadosnecessários dos ficheiros csv.
- **datastruc.c** – contém todas as funções necessárias para inicializar e aceder àestrutura de dados DATA.
- **drivers.c** – toda a informação necessária para tratar das informações doscondutores.
- **users.c** – toda a informação necessária para tratar das informações dosutilizadores.
- **rides.c** – informação necessária para atualizar os dados tanto a condutorescomoa utilizadores.
- **func.c** – contém funções auxiliares ao analisar dados.

- **bash.c** – interpreta as instruções e reencaminha a informação necessária para realizarcada query.
- **bashe.c** – interpreta as instruções e reencaminha a informação necessária pararealizar cada query, sendo também realizado os testes de desempenho.
- **querry1.c** – informação necessária para responder à 1ª Query.
- **querry2.c** – informação necessária para responder à 2ª Query.
- **querry3.c** – informação necessária para responder à 3ª Query.
- **querry4.c** – informação necessária para responder à 4ª Query.
- **querry5.c** – informação necessária para responder à 5ª Query.
- **querry6.c** – informação necessária para responder à 6ª Query.
- **querry7.c** – informação necessária para responder à 7ª Query.
- **querry8.c** – informação necessária para responder à 8ª Query.
- **querry9.c** – informação necessária para responder à 9ª Query.
- **interactive.c** – toda a informação necessária para realizar as queries de formainterativa.

Resolução das Queries:

Query 1 : "Listar o resumo de um perfil registado no serviço através do seu identificador, representado por <ID>."

```
GArray *drivers;
GHashTable *users;
```

A função utilizada na query 1 procura informações de utilizadores ou condutores a partir do **<ID>** dado. Este tanto pode ser o **username** ou o **ID** do user ou do driver.

A função começa por verificar se o **<ID>** começa com um número ou não. Caso comece com um número, a função usa o **GArray drivers** para procurar o driver com o ID especificado e verifica se é ou não ativa a conta do driver, caso seja imprime a informação pedida. Se o **<ID>** começar com uma letra, a função usa a **GHashTable users** para procurar o user com o username especificado e verifica se é ou não ativa a conta do driver, caso seja imprime a informação pedida.

Query 2 : "Listar os N condutores com maior avaliação média. Em caso de empate, o resultado deverá ser ordenado de forma a que os condutores com a viagem mais recente surjam primeiro. Caso haja novo empate, deverá ser usado o id do condutor para desempatar (por ordem crescente)."

GList *ordDrivers;

A query 2 começa por obter a lista ordenada de drivers com a função **getOrdDrivers**.

A função então itera sobre a lista usando o loop **while** e imprime as informações dos primeiros **n** drivers com status de conta ativo. As informações impressas são o resumo do perfil do driver.

Query 3 : "Listar os N utilizadores com maior distância viajada. Em caso de empate, o resultado deverá ser ordenado de forma a que os utilizadores com a viagem mais recente surjam primeiro. Caso haja novo empate, deverá ser usado o username do utilizador paradesempatar (por ordem crescente)."

GList *ordUsers;

A query 3 começa por obter a lista ordenada de users com a função **getOrdUsers**.

A função então itera sobre a lista usando o loop **while** e imprime as informações dos primeiros **n** users com status de conta ativo. As informações impressas são o resumo do perfil do user.

Query 4 : "Preço médio das viagens (sem considerar gorjetas) numa determinada cidade,representada por <city>."

GHashTable *ridesCity;

A query 4 usa a tabela mencionada em cima para calcular o custo médio de todas as rides numa cidade específica.

Usa a função **lookUpCityRides** para obter uma lista ligada de todas as viagens na cidade especificada. Se não encontrar a cidade, a função cria um ficheiro vazio.

A função, então, itera através da lista ligada, obtém o custo de cada viagem usando a função **getCostR**, adiciona à variável **r** e incrementa um contador **c** por cada ride.

Finalmente, o **custo médio** de todas as rides é calculado dividindo o custo total **r** pelonúmero de rides **c**.

Query 5 : "Preço médio das viagens (sem considerar gorjetas) num dado intervalo de tempo,sendo esseintervalo representado por <data A> e <data B>."

GList *rides;

A query 5 começa por usar a função **getRides** para obter a lista ligada completa de todas as rides. A função itera através da lista ligada, e salta as rides cujas datas são menores que o início do intervalo de datas **i** usando a função **getDateR**.

A função então itera através da lista ligada até que todas as rides cujas datas estão dentro do intervalo de datas [i, f] sejam percorridas. O custo de cada corrida é obtido usando a função **getCostR** e adicionado ao variável **r**. O contador **c** é incrementado para cada ride.

Finalmente, se o custo total **r** não é 0, o custo médio é calculado dividindo **r** por c.

Query 6 : " Distância média percorrida, numa determinada cidade, representada por <city>, num dado intervalo de tempo, sendo esse intervalo representado por <data A> e <data B>."

GHashTable *ridesCity;

- A Query 6 começa por usar a função **lookUpCityRides** para obter uma lista ligada de todas as rides realizadas na cidade específica. Se a cidade não exitir, a função retorna imediatamente.

A função então itera através da lista ligada, e salta as rides cujas datas são menores que o início do intervalo de datas **i** usando a função **getDateR**.

A função então itera através da lista ligada até que todas as rides cujas datas estão dentro do intervalo de datas [i, f] sejam percorridas. A distância

percorrida em cada ride é obtida usando a função **getDistanceR** e adicionada a variável **r**. O contador **c** é incrementadopor cada ride.

Finalmente, se a distância total **r** não é 0, a distância média é calculada dividindo **r** por **c**.

Query 7 : "Top N condutores numa determinada cidade, representada por <city> (no ficheiro rides.csv),ordenado pela avaliação média do condutor. Em caso de empate, o resultado deverá ser ordenado através do id do condutor, de forma decrescente."

GHashTable *topDrivCity;

A query 7 usa a tabela mencionada em cima para encontrar os top N drivers.

Usa a função **lookUpCityRides** para obter uma lista ligada de todos os drivers na cidade especificada. Se não encontrar a cidade, a função cria um ficheiro vazio.

A função, então, itera através da lista ligada e imprime os primeiros n elementos da lista.

Query 8 : "Listar todas as viagens nas quais o utilizador e o condutor são do género passado como parâmetro, representado por <gender> e têm per s com X ou mais anos, sendo X representado por <X>."

GList *rides;

A query 8 usa a GList *rides que representa a lista de todas as rides. Ela percorre a lista de corridas e verifica se o gênero do condutor e do user é igual ao gênero especificado na query **(g)** e se as idades dos perfis do driver e do user são maiores ou iguais à idade especificada na query **(age)**. Se as condições forem atendidas, a ride é adicionada à lista **gender**. A lista **gender** é então classificada de acordo com a ordenação especificada na função **depenGenderRides**. Por fim, a query imprime os dados das corridas na lista **gender**.

Query 9 : "Listar as viagens nas quais o passageiro deu gorjeta, no intervalo de tempo (data A, data B), sendo esse intervalo representado pelos parâmetros <data A> e <data B>, ordenadas por ordem de distância percorrida (em ordem decrescente). Em caso de empate, as viagens mais recentes deverão aparecer primeiro. Se persistirem empates, ordenar pelo id da viagem (em ordem decrescente)."

GList *rides;

A Query 9 começa por obter a lista rides com a **getRides(d)**. A lista é percorrida para encontrar rides que caem dentro do período especificado. Quando uma viagem é encontrada dentro do período, ela é adicionada à uma nova lista encadeadachamada **dateList**.

Depois de adicionar todas as corridas dentro do período à **dateList**, a lista é ordenada com a função **g_list_sort** usando a função de comparação **depenDistRides**. Em seguida, a função itera através da lista classificada e imprime a informação.

Testes de Desempenho:

Nesta componente do projeto foi-nos proposto que o trabalho realiza se os testes de desempenho, nomeadamente o tempo que cada query demora a executar, a confirmação do resultado dado pela query (verifica se o resultado obtido é suposto ou não) e por fim o tamanho total da memória utilizada pelo programa.

Para realizar esta tarefa criamos uma função main e uma batch novas com a utilidade de executar os testes, desta forma estes dois módulos (**main_testes.c** e **batche.c**) apenas são executados caso seja usado o comando **make programa-testes**. Ambos os módulos são bastante semelhantes aos do programa principal sendo que a principal diferença é que recebemos como input na **main_testes.c** uma pasta com as soluções que iram ser comparadas com o resultado que nós obtemos nas queries, neste módulo também calculamos a memória total utilizada através da **struct rusage** que pertence à biblioteca do **sys/resource.h**. Por outro lado, os testes de desempenho como o tempo utilizado e a comparação dos resultados são confirmados no módulo da **batche.c,** sendo que utilizamos a biblioteca **time.h** para calcular o tempo que cada query demora através da função **timer**. Para comparar os resultados obtidos, após criarmos o ficheiro na pasta de Resultados com o resultado que obtivemos na query enviamos para uma função chamada **comparaResultados** o mesmo e o ficheiro com que tem de ser comparado na pasta das correções e nesta função é verificado o resultado.

Fizemos uma tabela a comparar o tempo que cada query demora a fazer no computador de cada um, obtendo os seguintes resultados:

Queries	Luis	Nuno	Jorge
1	0.000011	0.000020	0.000009
2	0.000025	0.000039	0.00021
3	0.000029	0.000040	0.000025
4	0.023559	0.025246	0.023459
5	0.081455	0.074258	0.073258
6	0.016009	0.014765	0.014975
7	0.000044	0.000071	0.000059
8	0.132223	0.155410	0.130024
9	0.126966	0.124838	0.123517

Especificações dos computadores:

	Luís	Nuno	Jorge
Processador	intel i5-1135G7	Intel i7-9750H	Intel Core i5-10300H
RAM	16,00 GB	16,00 GB	8,00 GB

* O Nuno usa uma virtual box

Conclusão

Nesta fase do projeto tivemos especial dificuldade em implementar a validação de ficheiros, visto que inicialmente não estava a eliminar todas as linhas inválidas e além disso não conseguimos descobrir os problemas na ordenação da query 2 e 3. Tivemos também dificuldades a reparar os memory leaks e na criação de páginas no modo iterativo.

De resto, aprendemos muitos conceitos com a realização deste projeto.