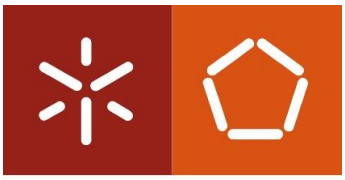

Universidade do Minho

Licenciatura em Engenharia Informática

Laboratórios de Informática III
Trabalho Prático - Fase 2

Grupo 13

Gonçalo Pereira (A96849) José Moreira (A95522)
Santiago Domingues (A96886)

Ano Letivo 2022/2023

Índice

Capítulo 1

Introdução

Apresentamos o relatório referente à Fase 2, e última, do projeto a desenvolver na Unidade Curricular de Laboratórios de Informática III. Como referido no relatório da fase 1, o projeto sugerido pelos docentes desafia ao desenvolvimento de uma aplicação, em C, que tenha a capacidade de guardar, manipular e realizar "queries"sob o conteúdo de ficheiros extensos ".csv". Com isto, é esperado um projeto fiável que, enfrentando grandes quantidades de dados, consiga responder às exigências sem que haja perda de dados e desempenho. Na fase 2, é pretendido que a nossa aplicação efetue as seguintes tarefas: a totalidade das queries descritas no enunciado do trabalho prático (9 queries), modo de operação interativo (incluindo o menu de interação com o programa e um módulo de paginação para apresentação de resultados longos), evolução dos aspetos relacionados com a modularidade, encapsulamento e qualidade do código observados na sessão de apresentação da fase 1 e análise e discussão sobre o desempenho da solução desenvolvida. É também esperado que o programa esteja preparado para receber ficheiros de entrada de dimensão superior e também algumas validações simples sobre o formato dos ficheiros de dados e, tudo isto, tendo em atenção o consumo de memória do programa, tentado, obviamente, que este seja o menor possível.

Capítulo 2

Visão geral da fase 1 e alterações

Tal como referido no relatório da primeira fase, a modularidade e o encapsulamento são características fulcrais neste projeto. O desenvolvimento do projeto com estes temas em mente permite que o código seja mais flexível, seguro e fiável, uma vez que há um controlo exemplar dos acessos a cada porção do mesmo. No capítulo seguinte, referente à arquitetura do projeto, iremos abordar cada um dos ficheiros e estruturas dos projetos, onde serão, por conseguinte, abordadas estes temas.

Iremos agora abordar as alterações feitas em relação à primeira fase. Estas alterações são provenientes de uma extensiva análise do grupo ao que havia sido feito até ao momento e também de sugestões dadas pelos docentes na defesa da fase 1.

Relativamente às estruturas de dados, inicialmente, o grupo desenvolveu três estruturas, os catálogos correspondentes a cada um dos tipos presentes nos ficheiros de entrada (*users*, *drivers* e *rides*). Uma vez que estas estruturas serviam de base à realização as queries que era suposto o programa efetuar na fase 1 (1/3 das queries), o grupo decidiu trabalhar com estas, sendo que já sabíamos que na fase 2 iríamos ter de fazer várias alterações a este respeito. O grupo optou então por criar duas estruturas adicionais, *stats* e *city_stats*. Estas estruturas serão explicadas posteriormente neste relatório.

Outro processo extremamente fundamental e que implicou alterações ao que havia sido feito na primeira fase, foi a procura da minimização do custo da aplicação em termos de memória. Todo este processo foi trabalhado extensivamente e foi com a ajuda de ferramentas como o **valgrind**, **gdb** e os testes automáticos que o grupo conseguiu "ultrapassar"este desafio, sendo a diferença notória. É de notar que este processo foi extensivo, pois não só na primeira fase estas melhorias tinham de ser feitas, mas também porque como esperado, ao longo desta segunda fase os ficheiros tiveram um acréscimo muito grande no que toca à dimensão. Seria impensável não realizar de forma rigorosa e, de certa forma, prioritária este trabalho de otimização. Numa fase posterior deste documento encontra-se todo um capítulo referente ao desempenho computacional.

Capítulo 3

Arquitetura do projeto (Estruturas e Ficheiros)

Em relação à arquitetura do projeto, no presente capítulo serão abordados de uma forma sucinta todos os módulos existentes na mesma. É importante referir que parte destes módulos/estruturas pertenciam já à primeira fase do projeto, que serão abordadas em primeiro lugar. Ao longo da explicação de cada estrutura, vão também ser explicadas as relações entre as mesmas. Novamente, com a descrição destes ficheiros, irá ser mencionado um dos principais objetivos da segunda fase, a validação. Está implícito que para cada ficheiro do tipo *.c* (à exceção dos ficheiros que contêm as *main functions*) abordado neste capítulo, existe um *header file*. Estes *header files* encontram-se na pasta *includes* do nosso projeto.

3.1 driver.c/user.c/ride.c

De forma a simplificar a explicação, decidimos agrupar estes três módulos na mesma secção. Cada um destes ficheiros representa a singularidade de cada tipo, ou seja, o ficheiro *driver.c* contém a informação de cada um dos milhares de condutores existentes no ficheiro csv de entrada. O mesmo acontece para os ficheiros *user.c* e *ride.c*. Para além destas estruturas (as quais são apresentadas nas figuras seguintes), estes ficheiros contêm as funções de criar/eliminar as mesmas, e os *getters* para cada um dos campos da informação relativa a cada um dos "objetos", que devolvem *strdup* do campo pretendido de forma a não haver violação de encapsulamento.

```
struct driver {
    char* driver_id;
    char* driver_name;
    char* birthday;
    char* gender;
    char* car_class;
    char* license_plate;
    char* driver_city;
    char* account_creation;
    char* account_status;
};
```

```
struct user {
    char* username;
    char* name;
    char* gender;
    char* birth_date;
    char* account_creation;
    char* pay_method;
    char* account_status;
};
```

```
struct ride {
    char* ride_id;
    char* date;
    char* driver_id;
    char* user;
    char* city;
    char* distance;
    char* score_user;
    char* score_driver;
    char* tip;
    char* comment;
};
```

Figura 3.1: **driver**, **user** e *ride*

3.2 cat_drivers.c/cat_users.c/cat_rides.c

Da mesma forma da secção anterior, todos estes ficheiros são semelhantes, de maneira que serão abordados todos "de uma só vez". De forma a agrupar as estruturas criadas anteriormente, desenvolveu-se as estruturas *drivers*, *users* e *rides* que são, tal como o nome indica, organizações coletivas das estruturas respetivas anteriormente referidas, ou seja, catálogos que, neste caso, são representadas por arrays dinâmicos. Cada um destes arrays possui um apontador para o tipo de estrutura que acolhe, ou seja, *DRIVER**, *USER** e *RIDE**, um valor inteiro *length* e um valor inteiro *size*. Também como na secção anterior, estes ficheiros contêm as funções que servem de *getters* mas, desta vez, contêm também a função relativa ao parsing dos ficheiros de entrada. Ou seja, por exemplo, em *users.c* existe a função *read_users* que trata de passar a infrmação do ficheiro *users.csv* para a respetiva estrutura. O mesmo se passa para *drivers.c* e *rides.c*.

```
struct cat_drivers{
    DRIVER* driver;
    int length;
    int size;
};
```

```
struct cat_users {
    USER* user;
    int length;
    int size;
};
```

```
struct cat_rides {
    RIDE* ride;
    int length;
    int size;
};
```

Figura 3.2: **drivers**, **users** e *rides*

3.3 dates.c

O ficheiro *dates.c* é um ficheiro que serve apenas para armazenar as funções que o grupo criou para realizar operações sobre datas. Estas funções são uma espécie de funções auxiliares, visto que o tratamento de datas é um pouco mais complexo do que o tratamento dos restantes campos (devido ao seu formato), e estas operações são as seguintes: transformar uma data com o formato dd/mm/aaaa numa data com o formato aaaammdd, verificar se uma data se encontra no intervalo de outras duas datas, e calcular a idade a partir de uma certa data.

3.4 util.c

Tal como o ficheiro explicado acima trata das datas, *util.c* é o ficheiro que contém funções auxiliares para "manipulação"de strings. As funções nele existentes servem para conectar duas strings, trocar as posições de dois elementos num array, remover o "
n" de uma string, colocar todos os elementos de uma string em letra maiúscula, converter do tipo int para o tipo string e, por último, tem também a função de "limpar"o terminal.

3.5 parsing.c

Encarregue de fazer todo o parsing dos ficheiros csv de entrada. Nele tem apenas a função de parsing, que cria as estruturas de acordo com o parsing feito a cada ficheiro, cria também,

adicionalmente duas estruturas: *stats* e *city_stats* (que serão abordadas a seguir) e "chama"a função para a resolução de queries utilziando estas estruturas, eliminando-as de seguida.

3.6 queries_aux.c

Este ficheiro é o que serve de base para a realização de todas as queries. Nele contém as funções que calculam os resultados para todas as queries. Então, todas as operações necessárias para a realização de uma dada query, quer seja operação de "tirar"informação das estruturas, calcular médias, etc, econtram-se todas neste ficheiro. Tem também a estrutura *result*, estrutura que vai guardar os resultados da query 1, apenas, para posteriormente os imprimir, e também a estrutura *all_rides*, usada nas queries 8 e 9, de forma a obter a informação necessária para a impressão do resultado.

3.7 queries.c

Aqui estão contidas as funções para a impressão dos resultados nos ficheiros de resultados. Tem também a função para o parsing do ficheiro de input que contém as queries a realizar, de forma a realizar as queries corretas, chamando para isso a função *choose_query*.

3.8 param_queries.c

A estrutura *param*, que é apresentada na figura seguinte, é a estrutura criada pelo grupo para guardar os resultados resultantes da exexução de cada query para posterior impressão, à exceção da query 1 em que, como referido acima, os resultados são guardados em *result*. As funções presentes neste ficheiro permitem criar/eliminar a estrutura, os *getters* para os diferentes componentes, inserção de elementos, ordenação dos dados e impressão. Relembra-se que a ordenação é muito importante pois em grande parte das queries é necessário ordenar os resultados.

```
struct param {
    char** ids;
    double* values;
    char** dates;
    char** names;

    int arrays_size;
    int arrays_length;
};
```

Figura 3.3: *param*

3.9 stats.c

Este ficheiro contém uma das novas estruturas principais do projeto faladas na alínea anterior, a estrutura *stats*. A criação desta estrutura foi um ponto fulcral no trabalho pois definiu-se de que maneira se pode atingir os dados pretendidos. Como é possível observar na figura seguinte, esta estrutura é muito diferente das restantes até aqui abordadas. É composta por cinco *hash tables* e recorre à biblioteca *glib*.

```
struct stats {
    GHashTable* drivers;
    GHashTable* users;

    GHashTable* users_indexs;
    GHashTable* drivers_indexs;
    GHashTable* rides_indexs;
};
```

Figura 3.4: *stats*

- *drivers* - A chave é o id do condutor e o valor é a lista dos ids das viagens associadas a ele;
- *users* - A chave é o id do utilizador e o valor é a lista dos ids das viagens associadas a ele;
- *users_indexs* - A chave é o id do utilizador e o valor é o seu índice no catálogo;
- *drivers_indexs* - A chave é o id do condutor e o valor é o seu índice no catálogo;
- *rides_indexs* - A chave é o id da viagem e o valor é o seu índice no catálogo;

3.10 city_stats.c

A outra estrutura adicionada foi a *city_stats*, estrutura essa que é composta por um array de estruturas do tipo *city*. Por sua vez, essa mesma estrutura é responsável por associar, a um *char** nome (da cidade), os índices, correspondentes ao catálogo das *rides*, de todas as viagens realizadas nessa mesma cidade.

```
struct city {
    char* name;

    int* rides;
    int rides_size;
    int rides_length;
};
```

Figura 3.5: *city_stats*

```
struct city_stats {
    CITY* cities;
    int cities_size;
    int cities_length;
};
```

Figura 3.6: *city_stats*

3.11 validation.c

Este é o ficheiro referente a um dos principais objetivos desta segunda fase, a validação dos ficheiros de entrada. A validação consiste em analisar os campos de cada linha nos ficheiros csv e, caso algum campo seja inválido, a linha deve ser descartada, não sendo necessário efetuar validação entre ficheiros uma vez que, por exemplo, se o utilizador/condutor não for válido, as viagens associadas a ele não existem, pelo que não é necessário validar essas viagens. O ficheiro *validation.c* tem então as funções de validação para cada um dos campos a considerar, enumerados no enunciado.

3.12 pages.c

Já no modo interativo, uma das tarefas mais desafiantes foi a paginação. De modo a realizar esta tarefa, o grupo criou duas estruturas: *page* e *pages*. A primeira representa uma página e a segunda um array de páginas.

```
struct page {
    char* fst;
    double value;
    char* snd;
    char* trd;
    char* fth;
    int flag;
};
```

Figura 3.7: *page*

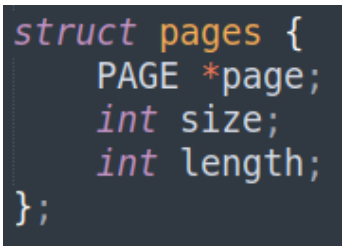

Figura 3.8: *pages*

Como sabemos, é suposto que cada query imprima um conjunto de campos relativos ao que é pedido, quer seja condutores, utilizadores ou viagens. De forma a conseguir passar esses campos para as páginas, o grupo introduziu na estrutura *page* as variáveis *fst*, *value*, *snd*, *trd* e *fth*. Tem também a variável *flag* para indicar o modo de atuação dependendo da query a ser realizada. Para além disto, o ficheiro apresenta as funções esperadas de manipulação de páginas: criar/eliminar páginas, passar os dados da estrutrura *param* para a estrutura das páginas, a impressão das páginas e uma função de controlador, que itera sob o conjunto de páginas e imprime as pretendidas, permitindo também a opção de voltar para trás e selecionar para a página pretendida.

3.13 ui.c

Este ficheiro "apenas"engloba as funções de tudo o que vai ser impresso no modo interativo e no modo de testes. Isto é, apresenta o menu, que contém as opções de selecionar qualquer uma das queries ou sair, e também as opções de interação a cada página, avançar ou retroceder. Por outro lado, este módulo também fornece as funções de visualização e escolha de opção para a realização de testes.

3.14 interface.c

O ficheiro *interface.c* trata de fazer a ligação entre o terminal e o resultado das queries no modo interativo. Por outras palavras, estando no modo interativo, se for feito o pedido de uma query, as funções que tratam de obter o resultado da query para imprimi-lo estão neste ficheiro. Deste modo, este ficheiro desenvolve o trabalho útil do modo interativo, "encaminhando"o mesmo, se necessário, para o módulo de paginação.

3.15 execution.c

Por último, este ficheiro contém as funções relativas à avaliação de execução das queries. Estas funções utilizam a biblioteca *time.h* de forma a calcular o tempo de execução de cada uma das mesmas. Este módulo também é responsável por avaliar a execução das queries, comparando os seus resultados a resultados bem definidos presentes em ficheiros já existentes, de modo a verificar se os mesmos são corretos.

3.16 main.c e tests_main.c

Ambos os ficheiros são responsáveis por armazenar as funções *main* presentes no projeto. Deste modo, o módulo *main.c* contém a função inicial de execução de todo o trabalho, enquanto que o ficheiro *tests_main.c* guarda a função inicial de execução dos testes.

Capítulo 4

Desempenho e Custo Computacional

No que toca ao desempenho computacional, irão ser apresentados a seguir os valores obtidos para os testes feitos pelo grupo. Para isto, foi usada a máquina ASUS TUF Gaming F15 com as seguintes especificações: processador Intel Core i9-11900H, placa gráfica NVIDIA GeForce RTX 3060, armazenamento 1 TB SSD e memória RAM 16GB DDR4. Todos os testes foram feitos com o cabo de alimentação ligado e foi utilizado o dataset regular. Note-se também que estes valores correspondem a uma média obtida de dez ensaios, dos quais se retiraram os valores mais alto e mais baixo. Repare-se que o ficheiro de input utilizado apenas possui uma query de cada tipo.

Foram obtidos os seguintes resultados:

- query 1 - 0.000040 segundos
- query 2 - 0.761572 segundos
- query 3 - 0.784181 segundos
- query 4 - 0.048199 segundos
- query 5 - 0.097632 segundos
- query 6 - 0.000005 segundos
- query 7 - 0.413177 segundos
- query 8 - 0.470171 segundos
- query 9 - 0.121410 segnudos

Através da utilização do **valgrind**, foi possível saber que a alocação de bytes para uma execução de cada tipo é de 511029508 bytes, valor esse que será ligeiramente alterado aquando a alteração do tipo dos argumentos das queries. É também importante salientar que a nossa aplicação não consegue responder da forma pretendida aos testes fornecidos para os datasets do tipo *large*, uma vez que a memória utilizada é superior a 4GB.

Capítulo 5

Manual de Utilização

Tal como pedido pelos docentes, a nossa aplicação apresenta dois modos de operação: *batch* e *interativo*, tendo também a implementação dos testes. Este capítulo servirá de manual de utilização (e também explicação) para cada um destes modos.

5.1 Batch

Para executar o programa usando o modo de operação *batch*, o utilizador apenas necessita de três passos. Inicialmente, encontrando-se já na diretoria correta, executa o comando **make**. Desta forma, o programa fica compilado e é criada a pasta para onde vão os ficheiros dos resultados. De seguida, basta introduzir o comando **./programa-principal dataset_path input_path**. Note-se que *entrada* corresponde à pasta onde se encontram os ficheiros de entrada csv, cujo nome tem de ser colocado dentro de aspas. O argumento seguinte é o caminho para o ficheiro de imput (também dentro de aspas) que tem as queries a serem realizadas. Quando executado este comando, deverão ser lidos os ficheiros dataset e realizada a totalidade das queries presentes no ficheiro de input.

Por último, o utilizador deve digitar o comando **make clean** de forma a apagar todos os registos resultantes das queries, desde ficheiros de resultados a pastas *obj*.

5.2 Interativo

Para executar o programa usando o modo de operação *interativo*, o utilizador começa por introduzir novamente o comando **make**. De seguida, executa o comando **./programa-principal**. Após isto, o programa irá pedir o caminho/nome da pasta que contém os ficheiros de entrada. Com este já colocado, aparece no terminal do utilizador o seguinte menu:

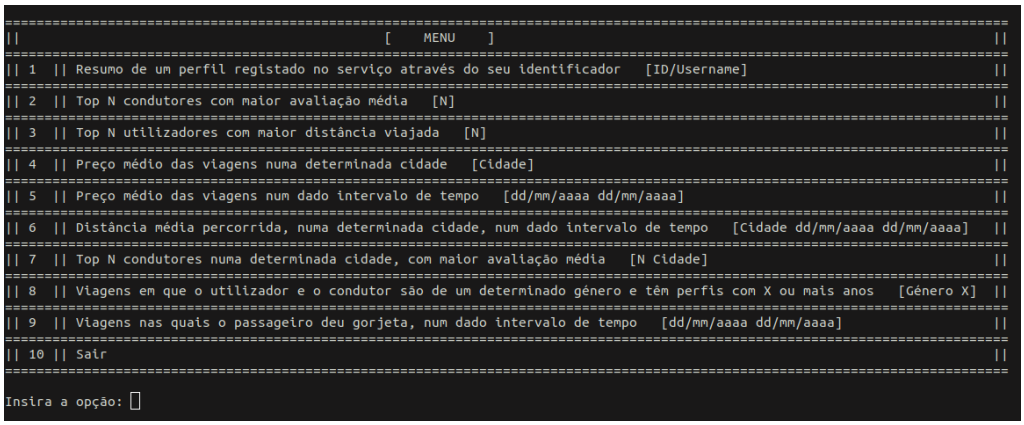

Figura 5.1: Menu do modo interativo

Como é possível observar na figura, o utilizador pode selecionar uma opção no intervalo de 1 a 10, sendo as primeiras nove relativas às queries, e a última para terminar o programa. Tal como no modo *batch*, uma vez finalizada a execução do programa, o utilizador deve executar o comando **make clean**.

5.3 Testes

Em relação à execução dos testes, após fazer o comando **make** tal como nos modos anteriores, o utilizador deve executar o comando **./programa-testes**. O terminal irá então imprimir as duas opções possíveis que o grupo decidiu colocar como pré-definidas. Essas opções são as da figura seguinte.

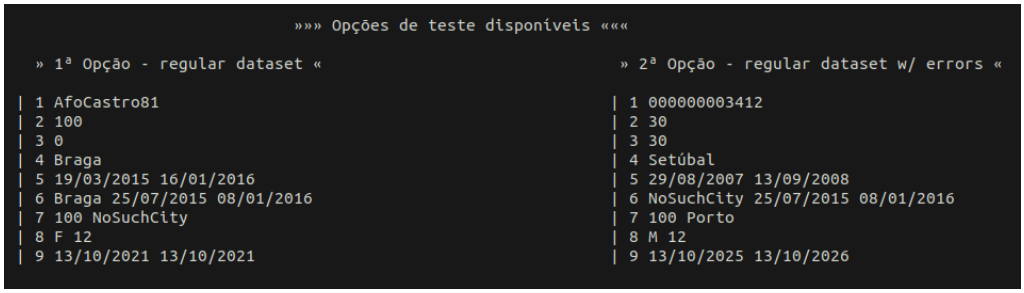

Figura 5.2: Menu de testes

Após selecionada uma das opções pelo utilizador, o programa irá apresentar a tabela de resultados. Novamente, no final da utilização, o utilizador deverá executar o comando **make clean**.

Capítulo 6

Conclusão

A realização deste projeto permitiu aos elementos do grupo adquirir novas competências quer ao nível da linguagem de programação C, quer ao nível de ferramentas e análise. Por um lado, foram adquiridos novos conhecimentos relativamente à linguagem de programação utilizada, conhecimentos esses provenientes do facto de que o trabalho roda em torno da modularidade e do encapsulamento, o que permitiu ao grupo a utilização de elementos de C com os quais o grupo nunca tinha tido contacto. Por outro lado, em relação às ferramentas, utilização do *valgrind* foi também uma experiência completamente nova, e foi o que nos permitiu corrigir as *leaks* de memória que a aplicação tinha, tendo, obviamente, impacto direto na memória utilizada. Em adição a tudo isto, o projeto foi importante para a análise de desempenho, como citado anteriormente neste relatório, o que permitiu ao grupo perceber quais as melhores estruturas a adotar para os objetivos pretendidos. Relativamente ao enunciado, a aplicação cumpre todos os objetivos propostos e realiza com sucesso os testes automáticos, apresentando, porém, a condição relativa aos datasets do tipo *large* explicada no capítulo do desempenho computacional.

Dá-se então como terminado o relatório do projeto desenvolvido no âmbito da UC de Laboratórios de Informática III, no presente ano letivo de 2022/2023.