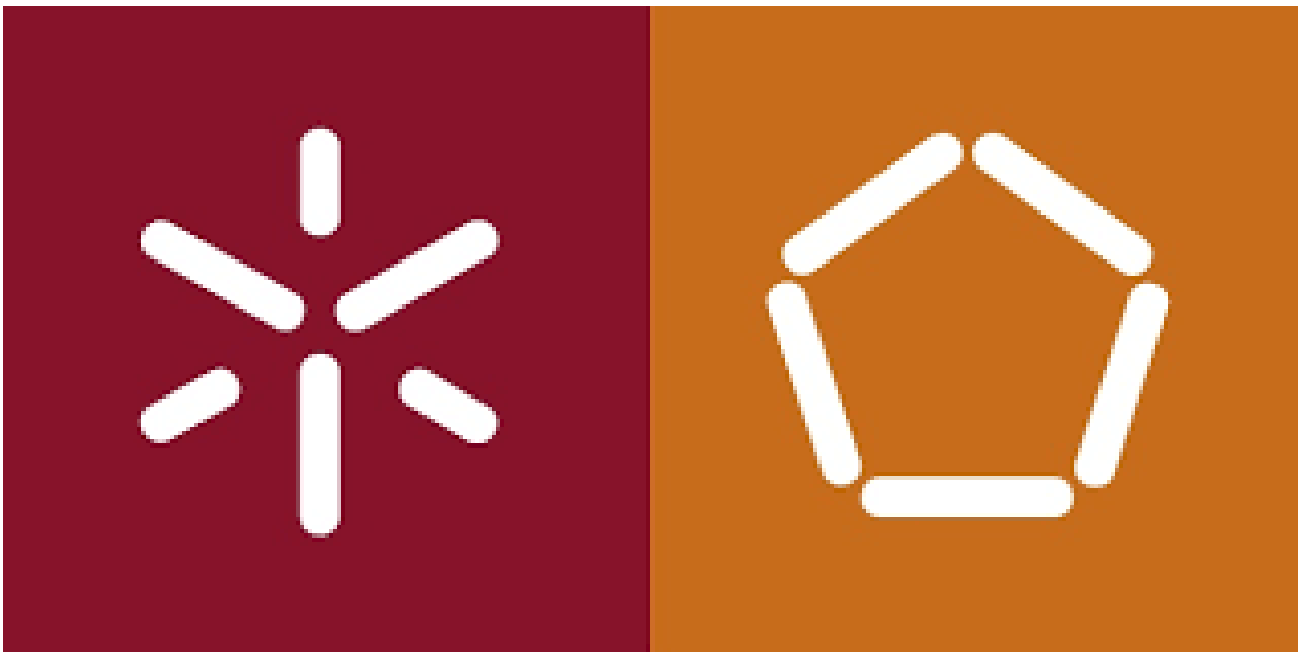

Universidade do Minho

Departamento de Informática

Laboratórios de Informática III

Grupo 4

Gonçalo Costa	José Oliveira	Pedro Lopes
a100824	a100547	a100759

Contents

Chapter 1

1.Introdução

O presente relatório tem como objetivo apresentar o projeto realizado no âmbito da unidade curricular de Laboratórios de Informática III, ao longo do primeiro semestre, do segundo ano, da Licenciatura em Engenharia Informática da Universidade do Minho. Este consiste na criação de um programa capaz de ler , armazenar e gerir toda a informação válida contida em ficheiros csv(s) e devolver os dados pretendidos pelo utilizador da forma mais eficiente e rápida possível.

1.1. Descrição do Problema

Os 3 ficheiros csv(s) deverão ser processados: users.csv, drivers.csv e rides.csv, cada linha nesses ficheiros deverá conter um registo com a informação de um utilizador, condutor e viagem, respetivamente. Um dos objetivos deste trabalho é armazenar toda a informação contida nesses ficheiros nas estruturas de dados mais adequadas para cada fim e responder aos pedidos do utilizador sobre as mesmas.

1.2 Análise da Solução

Nesta primeira fase, averiguamos quais as estruturas de dados que melhor se adequam à resolução dos problemas que enfrentávamos. A nossa escolha de organização teve sempre em vista a rapidez das queries, bem como a menor utilização de memória possível. Durante este processo, tivemos também em atenção o encapsulamento e a modularidade do programa, bem como as boas práticas da linguagem.

Chapter 2

2. Código do Projeto

A finalidade deste capítulo é expor detalhadamente, e com a devida fundamentação, a nossa abordagem para o armazenamento, gestão de dados e resolução das Queries do utilizador. Esta componente constitui a maior parte da lógica do nosso programa.

2.1. Parser

Para a conversão dos ficheiros csv(s), criamos um parser genérico que converte os 3 ficheiros csv em strings através do uso das funções fgets() e strsep(), pois tivemos em conta os campos vazios nos comentários dos utilizadores. Feita a conversão de cada linha dos ficheiros num array de tokens, estes vão ser transformados numa struct de cada tipo e os valores dos tokens vão ser atribuídos aos campos das structs através do uso das funções de processamento, e uma função para organizar cada catálogo numa hashtable e num array. O parser também irá utilizar funções adicionais para realizar as estatísticas em tempo de parse, e para garantir o encapsulamento nós utilizamos pointers para funções que podem processar esses dados.

2.2. Tratamento dos dados

2.2.1. Transformação em Structs

O parser.c irá ler os ficheiros e utilizando a parse_func correspondente de cada módulo, que é passada ao parser pela main e irá converter o array de string tokens em utilizadores, condutores e viagens, nos campos definidos nas structs abaixo.

```c
struct user{
    char *username;
    char *name;
    char gender;
    unsigned short birth_date;
    short account_creation;
    enum method pay_method;
    char account_status;
    short trips;
    double total_spent;
    unsigned short total_dist;
    short aval;
    short aval_m;
    unsigned short last_trip_date;
};
```

```c
struct ride {
    int id;
    short date;
    int driver;
    char *user;
    char *city;
    short distance;
    short score_user;
    short score_driver;
    float tip;
    char *comment;
};
```

```c
struct driver
{
    int id;
    char *name;
    unsigned short birth_date;
    char gender;
    char *car_class;
    char license_plate[sizeof("00-00-AA")];
    char *city;
    unsigned short account_creation;
    char account_status;
    int total_dist;
    unsigned short trips;
    unsigned short aval;
    double total_spent;
    double aval_m;
    unsigned short last_trip_date;
};
```

2.2.2. Criação das DataBases

Estando as structs criadas com os valores dos tokens nos respectivos campos, vamos agora inserir as structs criadas numa Database, através do uso de uma hashtable para o armazenamento destas structs, seguindo a seguinte definição da Database:

```
typedef struct data_base_users{
    User** users_array;
    GHashTable* users_hashtable;
    int len;
    int order;
} DB_users;
```

```
struct data_base_drivers
{
    Driver **drivers_array;
    GHashTable *drivers_hashtable;
    int order;
    int len;
};
```

```
typedef struct data_base_rides{
    void** rides_array;
    GHashTable* rides_hashtable;
} DB_rides;
```

2.2.3. Estatísticas

Para a resposta das queries, nós necessitamos de calcular algumas estatísticas dos users e drivers, como por exemplo, o número total de viagens, a distância total de viagem, a avaliação média e o total gasto. Para isto, para não termos de ler todos os dados uma segunda vez e, assim, termos muito mais tempo de execução desnecessário, decidimos calcular estas estatísticas ao longo do parsing dos dados, usando as funções set_user_stats() e set_driver_stats().

2.3. Dates

Inicialmente tivemos a ideia de usar strings para guardar os valores pretendidos, no entanto chegamos à conclusão que ocuparia demasiada memória e não seria eficiente visto que teríamos de comparar strings sempre que quiséssemos comparar datas, por isso criamos um módulo próprio onde utilizamos shorts para tal efeito, pois assumimos também como data "base" o dia 1 de Janeiro 1950 que se encontra a 26579 dias da data presente (valor menor do que o máximo que um short aguenta). Criamos também uma struct para facilitar a utilização e o cálculo entre as datas .

2.4. Queries

Para a resolução das queries, depois da criação das Databases e armazenamento dos dados provenientes do parser, criamos funções para responder a cada query, dependendo da Database a que acedemos. Para a query 1, verificamos se o primeiro caracter do ID é uma letra ou um número e, dependendo disso, chamamos a answer_query1_user e answer_query1_driver, respetivamente, que vão retirar a struct da hashtable com a key igual ao ID de input da query.

Para as queries 2 e 3, resolvemos de maneira semelhante, ordenando os arrays utilizados e armazenados nas Databases de drivers e users, respetivamente, retirando do array N structs, dependendo do N utilizado como argumento em ambas estas queries. Depois disto, resolvemos os empates que pode haver entre certos campos das structs retiradas e organizamos o resultado, dependendo do que é pedido em cada query.

2.5. Main

Aqui, para a execução do código, primeiro verificamos se o número de argumentos da main é menor que 2 e, caso seja, dá um erro, pois nesta fase o número de argumentos tem de ser pelo menos 2 para respeitar o modo batch. Depois disto, procedemos à abertura, envio dos 3 ficheiros csv para o parser e armazenamento dos dados provenientes do parser em arrays de void *. Uma dificuldade que tivemos aqui foi em como usar o primeiro argumento da main (argv[1]) para a abertura dos ficheiros csv, mas depois concatenamos o primeiro argumento da main com a localização dos ficheiros csv e já resolvemos isso. Depois disto feito, abrimos e damos parse do ficheiro de input também e armazenamos o resultado da mesma forma que com os csv(s) e, finalmente, com estes dados, respondemos as queries através da função answer_queries(), que vai usar os arrays provenientes na abertura e conversão dos ficheiros para responder as queries. Finalmente, depois de respondidas as queries, libertamos toda a memória alocada na execução do projeto .

Chapter 3

Conclusão

Mediante os exercícios propostos enfrentamos alguns desafios mas concluímos que atingimos aquilo que nos foi pedido. Reconhecemos a possibilidade de melhorar aquilo que já foi feito, tanto na eficiência como na rapidez e na gestão de memória. O encapsulamento, de fato, foi um grande desafio, mas consideramos que chegamos a um bom resultado em mantê-lo nas funções, mesmo que na struct users e struct drivers não esteja perfeito. Teremos ainda, para a segunda fase, de analisar essencialmente como vamos lidar com o aumento do número de drivers, rides e users, mas estamos satisfeitos com aquilo que obtivemos até agora.