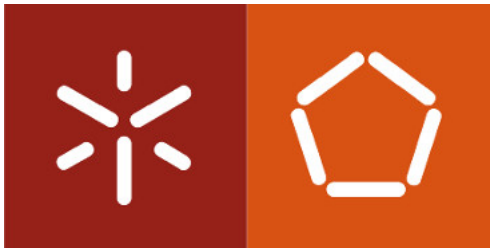

Universidade do Minho
Licenciatura em Engenharia Informática

Laboratórios de Informática III

Grupo 30

Filipa Rebelo - A90234 Pedro Pacheco - A61042
Pedro Pinto - A87983

Novembro 2022

Conteúdo

1 Introdução **3**

2 Estruturas de dados **4**
2.1 User 4
2.2 Ride 4
2.3 Driver 5
2.4 catalogoDrivers, catalogoRides e catalogoUsers 6
2.5 funcAux 6
2.6 data 6
2.7 queries 6
2.8 stats 7
2.9 Interpretador 7
2.9.1 modo interativo 7
2.9.2 modo batch 7

3 Queries **8**
3.1 Query 1: Listar o resumo de um perfil registado no serviço através do seu identificador 8
3.2 Query 4: Preço médio das viagens (sem considerar gorjetas) numa determinada cidade 8
3.3 Query 5: Preço médio das viagens (sem considerar gorjetas) num dado intervalo de tempo 8
3.4 Query 6: Distância média percorrida, numa determinada cidade, num dado intervalo de tempo 9

4 Conclusão **9**

Lista de Figuras

1 Arquitetura da aplicação desenvolvida 3

1 Introdução

Este relatório diz respeito à primeira fase de um projeto que nos foi proposto no âmbito da unidade curricular de Laboratórios de Informática III e consiste na implementação de um programa que seja capaz de processar grandes volumes de dados de forma eficiente, utilizando os princípios de modularidade e encapsulamento.
Este programa serve efectivamente para guardar em memória informação relativa a Utilizadores, Condutores e Viagens numa aplicação ao estilo 'Uber'.
Para a realização deste projeto foram fornecidos três ficheiros com um considerável volume de dados - *rides.csv*, *drivers.csv* e *users.csv*, que contêm a informação sobre a qual o programa irá funcionar, o que nos levou a uma preocupação na escolha das estruturas de dados a utilizar e da sua eficiência na resposta a cada uma das *queries* propostas.
Nesta fase inicial foi nos desafiado que implementásse-mos um *parsing* dos três ficheiros de *input* e escolhêssemos estruturas de dados que suportariam essa informação em memória. Após algum debate, a decisão caiu no uso de *hashtables* porque esta estrutura é uma maneira rápida de procurar, criar e remover informação. Mais especificamente, as *hashtables* são tipicamente mais eficientes a encontrar informação pretendida do que árvores de procura, que têm uma complexidade linear, ou árvores binárias de procura, que têm uma complexidade logarítmica. Portanto, independentemente do tamanho do *input*, as *hashtables* têm tipicamente um tempo constante nas três mecânicas principais - procura, criação e remoção de dados.
Sabendo que o modo de interpretação de comandos nesta fase iria ser feito através da leitura de um ficheiro de texto contendo as *queries* pretendidas, foram também desenvolvidas formas de interpretar cada uma e executar a respetiva *query* com os argumentos indicados.

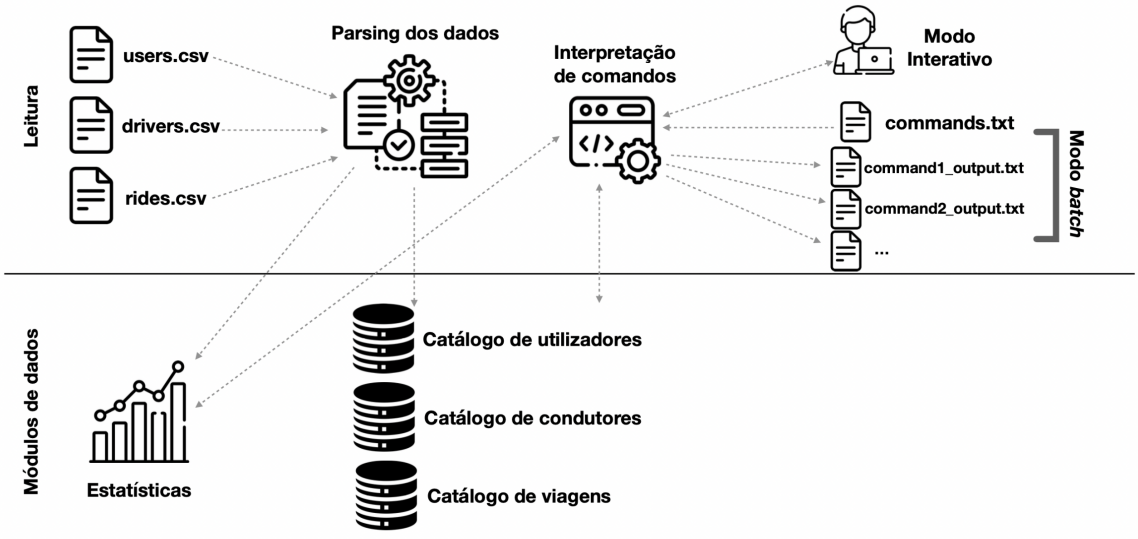

Figura 1. Arquitetura da aplicação desenvolvida

2 Estruturas de dados

2.1 User

```
struct user{
    char* username;
    char* nome;
    char* genero;
    struct tm data_nascimento;
    struct tm data_conta;
    char* metodo_pagamento;
    char* status;
    double score_medio;
    double score_total;
    int num_viagens;
    double total_gasto;
    struct tm data_ultima_viagem;
    double kms_percorridos;};
```

O módulo 'User' é onde definimos a estrutura que guarda a informação referente a um Utilizador mencionada no enunciado do Projeto.
Para cada utilizador, para além dos campos contidos no ficheiro *users.csv* acrescentamos os campos de 'score_medio', 'score_total', 'num_viagens', 'data_ultima_viagem' e 'kms_percorridos' que dizem respeito à avaliação média, avaliação total, número de viagens, data da última viagem e quilómetros percorridos de cada utilizador, de modo a facilitar a resolução das *queries* propostas no enunciado. É neste módulo que estão implementadas as funções para criação, validação e impressão de um utilizador. O módulo 'User' contém também as funções relativas aos *gets*, atualização de vários dos campos da estrutura e libertação de memória.

2.2 Ride

```
struct ride{
    char* id_ride;
    struct tm data_viagem;
    char* id_driver;
    char* username;
    char* city;
    double distance;
    double score_user;
    double score_driver;
    double tip;
    char* comment;
    char* car_class;
};
```

A estrutura 'Ride' guarda a informação pertinente a uma Viagem. Para cada viagem, além dos campos contidos no ficheiro *rides.csv*, adicionamos o campo 'car_class' que diz respeito à classe do veículo envolvido na viagem, para que se possa aceder a esta informação sem ter que percorrer a base de dados de Condutores e assim facilitar o algoritmo de resolução da *query 4*.
Este módulo possui funções relativas a *gets*, atualização de campos e libertação de memória da estrutura 'ride'. Contém ainda funções para criação, validação, impressão de uma viagem e para cálculo do custo da mesma.

2.3 Driver

```
    struct driver{
        char *id;
        char* nome;
        char* genero;
        struct tm data_nascimento;
        struct tm data_conta;
        char* classe_carro;
        char* matricula;
        char* cidade;
        char* status;
        double score_total;
        int num_viagens;
        double score_medio;
        double total_auferido;
        struct tm data_ultima_viagem;
};
```

Finalmente, o terceiro módulo pilar da aplicação é aquele que guarda a informação relativa ao Condutor. Cada condutor tem, para além dos campos contidos no ficheiro *drivers.csv* e à semelhança dos outros módulos elementares, outros campos que facilitam a resolução das *queries*. Tal como os módulos supracitados, neste também são definidas várias funções para tratamento da informação e e gestão de memória relacionada com o condutor.

2.4 catalogoDrivers, catalogoRides e catalogoUsers

```
    struct catDrivers{
        GHashTable* catDrivers;
};

    struct catRides{
        GHashTable* CatRides;
};
    struct catUsers{
        GHashTable* CatUsers;
};
```

Estes três módulos são o agrupamento das estruturas principais acima mencionadas em *hashtables* e funcionam como uma base de dados em memória para a informação contida nos três ficheiros de *input*.
Estes módulos contém funções de procura, inserção e inicialização feitas com o auxílio de funções disponibilizadas na biblioteca 'GLib'. O módulo 'CatRides' possui também uma função auxiliar para guardar todas os *values* da *hashtable* ('rides', neste caso) numa 'GList' que nos foi útil para as *queries* 4, 5 e 6.

2.5 funcAux

Módulo com funções auxiliares de carácter mais geral, que fazem o tratamento de dados relativos a datas e idade e que são utilizadas para mais que uma *query* e por mais do que um módulo. Contém a função *isNumber* que verifica se uma String tem apenas números, a função *converteStringData* que converte uma string para uma data numa struct tm, validando-a e verificando se está bem formatada. Além disso, a função *comparaDatas* que recebe duas datas e verifica qual é a mais antiga e outras funções que nos auxiliaram nas queries e no parsing.

2.6 data

O objetivo deste módulo é a leitura e tratamento dos dados dos três ficheiros de *input* (*rides.csv*, *drivers.csv* e *users.csv*) disponiblizados pela equipa docente. Aqui estão definidas três funções, uma por cada ficheiro, responsáveis pela leitura e validação dos dados.

2.7 queries

À data deste relatório, este módulo contém quatro funções relativas a resolução das *queries* 1, 4, 5 e 6 propostas no enunciado, e que irá futuramente conter os algoritmos das restantes.

2.8 stats

Este módulo tem como objetivo obter estatísticas para Utilizadores, Condutores e Viagens. A determinação das diversas estatísticas calculadas foi uma escolha do grupo, e portanto foram escolhidas algumas componentes que achamos mais pertinentes. No que diz respeito aos Utilizadores, é calculado o utilizador com mais quilómetros percorridos, mais dinheiro gasto e mais viagens.
Para as Viagens, é mostrada a viagem mais longa, a mais cara e o número de viagens para cada classe de carro.
Já nos Condutores, e em espelho com os utilizadores, calculamos o condutor com mais viagens e com mais dinheiro auferido.

2.9 Interpretador

O módulo interpretador está dividido em duas partes distintas - uma que contém as funções para o 'modo interarivo' e outra para o 'modo batch'.

2.9.1 modo interativo

Neste modo, o utilizador é apresentado com uma série de mensagens visuais através do *standard_output*, e de forma sucinta, tem que inserir a localização dos ficheiros de *input* e posteriormente escolher qual a informação que pretende que lhe seja mostrada.
De notar que é apenas através deste modo que é possível obter a informação relativa às estatísticas.

2.9.2 modo batch

Neste modo, o programa é executado com dois argumentos, o primero é o caminho para a pasta onde estão os ficheiros de entrada. Já o segundo corresponde ao caminho para um ficheiro de texto que contém uma lista de comandos (*queries*) a serem executados. O resultado da execução de cada comando é escrito num ficheiro individual criado pelo programa e que fica localizado na pasta 'resultados' da raiz da pasta 'trabalho-pratico'.
Este modo é executado à custa de duas funções - 'interpretadorBatch' e 'runQueries'. A primeira recebe os caminhos mencionados no parágrafo anterior assim como os apontadores iniciados para as *hashtables* de cada tipo. Esta procede ao preenchimento do restante dos caminhos para cada ficheiro de *input* e passa estas novas variáveis, assim como as restantes, à função 'runQueries'. Por sua vez é nesta função que é feito a maioria do trabalho, começando por ler os ficheiros de entrada e preencher as *hashtables*. É depois aberto o ficheiro de comandos e, para cada entrada, criado um ficheiro de *output* na pasta dedicada a estes, à medida que é lida cada linha e se vai chamando a função necessária à resolução da *query* pedida. O resultado é por fim escrito no ficheiro de saída.

3 Queries

Para esta fase do projeto foram desenvolvidas as *queries* 1, 4, 5 e 6 que iremos explicar com detalhe em seguida.

3.1 Query 1: Listar o resumo de um perfil registado no serviço através do seu identificador

```
char* query1(char* chave, CatDrivers drivers, CatUsers users);
```

Esta *query* recebe como argumento um identificador (*username* no caso de um Utilizador ou ID no caso de um Condutor), assim como apontadores para as *hashtable*s de Utilizadores e Condutores onde o identificador será procurado.
O algoritmo de resolução passa por detectar se o identificador é composto apenas por números (condutor) ou por caracteres alfanuméricos (utilizador) e para cada caso procurar pelo mesmo na respectiva *hashtable*. Caso o encontre e o estado da conta seja "*active*"então é calculada a idade e impressa numa *string* de *output* a informação pedida. No caso de o identificador ser inválido, a *string* de *output* será vazia.

3.2 Query 4: Preço médio das viagens (sem considerar gorjetas) numa determinada cidade

```
char* query4(char *city, CatRides r);
```

Para solucionar esta *query* começamos por guardar numa lista auxiliar todas as 'rides' e percorrer esta lista até chegar ao fim, indo para cada uma obter a sua cidade e comparar com a cidade passada como argumento da função. Se estas forem iguais adiciona-se a um contador 'valorviagem' o custo dessa viagem, calculado através de uma função auxiliar que se encontra no módulo 'rides', e incrementa-se a outro contador o número total de viagens. Finalmente para obter o preço médio das viagens, é necessário apenas dividir o 'valorviagem' pelo número total de viagens.

3.3 Query 5: Preço médio das viagens (sem considerar gorjetas) num dado intervalo de tempo

```
char* query5(char* data_inicial, char* data_final, CatRides rides);
```

A solução desta *query* começa por transformar as *strings* 'data_inicial' e 'data_final' passadas como argumentos da função *query* 5 em variáveis do tipo 'struct tm', através da função auxiliar 'converte_string_data' contida no módulo 'funcAux'. De seguida verificamos, através de uma outra função auxiliar 'comparaDatas', se as datas inicial e final são a mesma ou se a data inicial antecede a data final e, caso isto se verifique, são guardadas numa lista auxiliar todas as 'rides' da *hashtable* 'rides'. Esta lista é então percorrida e obtém-se para cada 'ride' a data da viagem. Por último, esta data é comparada com aquelas recebidas como argumentos e, se estiver compreendida

entre a 'data_inicial' e a 'data_final' são incrementados dois contadores - um com o número de viagens e outro com o total do custo de todas essas viagens.
Faltando apenas obter o preço médio das viagens, calculado dividindo um contador pelo outro.

3.4 Query 6: Distância média percorrida, numa determinada cidade, num dado intervalo de tempo

```
char* query6(char* data_inicial, char* data_final,char *city,
CatRides r);
```

Para resolver esta *query* começamos por guardar numa lista auxiliar todas as 'rides' da *hashtable* 'r' e tal como na *query* anterior, verificar através de uma função auxiliar, se as datas inicial e final são a mesma ou se a data inicial precede a data final. Caso isto se verifique percorre-se a lista auxiliar de 'rides' e para cada uma obtem-se a sua data e a cidade em que ocorreu. Posteriormente comparam-se estes valores com aqueles passados como parâmetros da função e, se a data da viagem estiver entre estas datas e a cidade for a aquela pedida, adiciona-se à variável 'distancia' o valor da distancia daquela 'ride' e incrementa-se a variável 'numero_viagens'. No final para obter a distância média percorrida divide-se o valor da 'distancia' pelo 'numero_viagens'.

4 Conclusão

A realização deste projeto permitiu-nos aprender e colocar em prática conceitos como modularidade e encapsulamento, bem como o uso das funções da biblioteca 'Glib' para tratamento de dados. Além disso foi possível aprender a manipular grandes volumes de dados de forma eficiente. Estamos, de um modo geral, satisfeitos com o resultado final e esperamos na próxima fase concluir com sucesso o projeto que nos foi proposto pela equipa docente.