Laboratórios de Informática III

Relatório do Projeto

GRUPO 49

Diogo Pinto
Jorge Rodrigues
Mariana Pinto

Conteúdo

Introdução

O projeto atribuído tem como objetivo dar a conhecer ao grupo os obstáculos de realizar uma aplicação de software em média-larga escala, isto é, com grandes volumes de dados, sendo que estes dados estão contidos em 3 ficheiros (rides, drivers e users) no formato *.csv* (Comma separated values). Para a realização do mesmo, usamos a livraria "glib" em C, que contém diversas estruturas de dados e API's das mesmas.

Estruturação

A estruturação do código consistiu em 3 API's base (rides, drivers e users) que tinham como propósito tratar e organizar os dados lidos a partir dos ficheiros dados. Assim, com o módulo de leitura (read.c) era possível traduzir os ficheiros CSV para estruturas de dados organizadas, de modo a facilitar o desenvolvimento e desempenho das funcionalidades pedidas. Também o módulo Catálogo (model.c) a partir dos módulos anteriores falados teria a capacidade de desenvolver as queries pedidas de maneira bastante simples, isto porque os dados tinham sido organizados e as funções disponibilizadas pelos módulos anteriores.

Por fim, o interpretador (interperter.c) é responsável pela interação entre o utilizador e o programa, através de um ficheiro *input* e os respetivos ficheiros de *Output* gerado na diretoria "Resultados".

Módulos

1. Utilizadores (users.c)

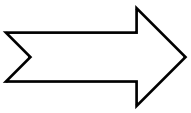
✓ Descrição

Os users, que estão dispostos na respetiva Collection, englobam diferentes parâmetros, como, por exemplo, o username, genéro, data de nascimento (…).

Relativamente à Users Collection, tem como parâmetro uma GHashTable (tabela de Hash da biblioteca *glib)* em que a key usada para mapeamento de dados é o username e o valor guardado é uma struct user (Users).

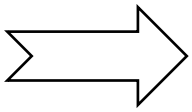
Cada user é armazenado de acordo com a seguinte struct, sendo os parâmetros *score*, *distance_total, total_gasto e num_rides,* obtidos através do ficheiro "rides.csv".

Cada user é armazenado baseando-se nesta struct.



```c
struct user {
    char *username;
    char *name;
    char *gender;
    char *birth_date;
    char *account_creation;
    char *pay_method;
    char *account_status;
    char *last_ride_date;
    double *score;
    double *distance_total;
    double *total_gasto;
    int*num_rides;
};
```

User collection → os users estão distrubuídos nesta collection. A GHashTable contém a key que é o username e o value é uma struct user.



```c
struct ucoll {
    GHashTable *table;
};
```

✓ **Procedimentos utilizados**

- Inits e Frees da Tabela e da Struct User.
- Utilização de Getters e Setters com a função de duplicação (strdup) de forma a garantir o encapsulamento.

Exemplos de funções que trabalham com o array das keys

```c
void sort_array_users (char **arrayKey,int N,Users_Collection ucoll,int num_vezes){

    for (int i = 0;i<N && num_vezes>=0;i++){

       Users u = look_up_users(ucoll,arrayKey[i]);
       if (strcmp("inactive",u_getAccount_status(u))==0) num_vezes++;
        int keep = i;
        for (int j = i;j<N;j++){
            Users temp = look_up_users(ucoll,arrayKey[j]);
            Users temp2 = look_up_users(ucoll,arrayKey[keep]);

            if (temp->last_ride_date==NULL && *(temp->num_rides)==0) continue;
            if (temp2->last_ride_date==NULL && *(temp2->num_rides)==0) continue;

            double diff = *(temp->distance_total) - *(temp2->distance_total);
            int date = more_recent2(temp->last_ride_date, temp2->last_ride_date);

            if (diff > 0)
                keep = j;
            else if (diff == 0 && date == 0)
                keep = j;
            else if (diff == 0 && date == 2 && strcmp(temp->username,temp2->username) < 0)
                keep = j;

        }
        char *swap = arrayKey[i];
        arrayKey[i] = arrayKey[keep];
        arrayKey[keep]=swap;


        num_vezes--;
    }
```

Exemplos de funções que manipulam a tabela

```c
int inserUser (Users_Collection ucoll, Users new){

  return g_hash_table_insert (ucoll->table,new->username,new);

}
```

Condutores (drivers.c)

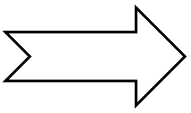
✓ Descrição

Este módulo agrupa todos os condutores e a respetiva informação contida no ficheiro. Os drivers, que estão dispostos na respetiva Collection, englobam diferentes parâmetros, como, por exemplo, o nome, genéro, data de nascimento, cidade (…).

Relativamente à Drivers Collection, tem como parâmetro uma GHashTable (tabela de Hash da biblioteca *glib)* em que a key usada para mapeamento de dados é o id e o valor guardado é uma struct driver (Drivers).

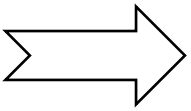
Cada driver é armazenado de acordo com a seguinte struct, onde os parâmetros *score*, *total_gasto* e *num_rides* são obtidos através do ficheiro “rides.csv”.

Cada driver é guardado baseando-se nesta struct.



```c
struct driver {
    char *id;
    char *name;
    char *birth_date;
    char *gender;
    char *car_class;
    char *license_plate;
    char *city;
    char *account_creation;
    char *account_status;
    char *last_ride_date;
    double *score;
    double *total_gasto;
    int*num_rides;
};
```

Driver collection → os drivers estão distrubuídos nesta collection. A GHashTable contém a key que é o id e o value é uma struct driver.

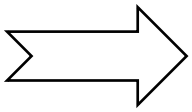


```c
struct dcoll {
    GHashTable *table;
};
```

✓ **Procedimentos utilizados**

- Utilização de Getters e Setters com a função de duplicação (strdup) de forma a garantir o encapsulamento.
- Inits e Frees da Tabela e da Struct Driver.

Exemplos de funções que manipulam a tabela



```c
int inserDrivers(Drivers_Collection dcoll, Drivers new){

    return g_hash_table_insert (dcoll->table,new->id,new);
```

2. Viagens (rides.c)

✓ Descrição

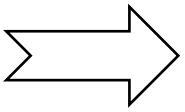
Este módulo agrupa todas as viagens e a respetiva informação contida no ficheiro. As rides, que estão dispostos na respetiva Collection, englobam diferentes parâmetros, como, por exemplo, a data, o condutor, a cidade (…).

Relativamente à Rides Collection, tem como parâmetro uma GHashTable (tabela de Hash da biblioteca *glib)* em que a key usada para mapeamento de dados é o id e o valor guardado é uma struct ride (Rides).

Cada ride é armazenada de acordo com a seguinte struct, onde todos os parâmetros são obtidos através do ficheiro "rides.csv".

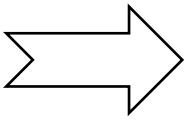
Neste módulo, durante o parsing das rides há certos parâmetros calculados e inseridos no utilizador, como por exemplo, last ride,coreusers, (?).

Cada ride é guardada baseando-se nesta struct.



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

Rides collection → as rides estão distrubuídas nesta collection. A GHashTable contém a key que é o id e o value é uma struct ride.

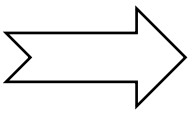


```
struct rcoll {
    GHashTable *table;
};
```

✓ Procedimentos utilizados

- Utilização de Getters e Setters com a função de duplicação (strdup) de forma a garantir o encapsulamento.
- Inits e Frees da Tabela e da Struct Driver.

Exemplos de funções que manipulam a tabela



```
int inserRide(Rides_Collection rcoll, Rides new){

   return g_hash_table_insert (rcoll->table,new->id,new);

}
```

3. Leitura (read.c)

✓ Descrição

Nesta secção, os ficheiros .csv (dataset) são lidos e convertidos nas estruturas de dados adequadas utilizadas no projeto, como por exemplo, os users e a respetiva tabela, rides e respetiva tabela e drivers e respetiva tabela.

❖ Sequencia de procedimentos:

- Carregar um ficheiro para a respetiva estrutura de dados
- Converter uma linha na respetiva estrutura de dados
- Verificação de linhas.
- Estatística.

Na função rides para além de ser feito o parsing e ser criada a tabela de rides também são atualizadas algumas componentes como, por exemplo, o "score" para o respetivo utilizador e driver.

4. Catálogo (model.c)

✓ Descrição

A estrutura de dados contém uma Collection correspondente a uma junção das coleções dos ficheiros anteriores, nomeadamente, Rides_collection, Users_collection, Drivers_collection. Para além disso, possui dois arrays com as keys correspondentes às estruturas Users_collection e Drivers_collection.

```
struct model {
    Rides_Collection rides;
    Drivers_Collection drivers;
    char** keyArray_d;
    char** keyArray_u;
    Users_Collection users;
};
```

Queries

Q1 →

Faz um resumo de um perfil registado no serviço através do ID. É uma query implementada de forma simples que retorna os dados x, dependendo do input, ou seja, caso seja um id de um driver ou o username de um utilizador.

Q2 →

É responsável por listar os N condutores com maior avaliação média. Em caso de empate, os condutores com a viagem mais recente surgem primeiro. Em caso de novo empate, o id do condutor, por ordem crescente, é o fator de desempate.

Nesta query, recorremos ao array de keys, que, juntamente com as funções de *sort* do ficheiro "drivers.c", permitiram colocar no início do array os N condutores com maior avaliação média (ignorando os condutores inativos).

```
void q2 (Model m,char*buffer,FILE *fptr) {
    shiftLeft(buffer,2);
    int N = atoi(buffer);
    int size;
    if (m->keyArray_d==NULL)
      m->keyArray_d=d_give_array(m->drivers,&size);
    sort_array_drivers(m->keyArray_d,size,m->drivers,N);

    for (int i = 0; i<N;i++){
        Drivers d = look_up_drivers(m->drivers,m->keyArray_d[i]);
        if (strcmp("inactive",d_getAccount_status(d))==0){
            N+=1;
            continue;
        }
        int num = d_getNumRides(d);
        double p;
        if (num == 0) { p = 0.0;}
        if (num != 0) { p = *(d_getScore(d))/(double) num;}
        fprintf(fptr,"%s;%s;%.3f\n",m->keyArray_d[i],d_getName(d),p);
    }
    free_key_array(m->keyArray_d,size);
}
```

Sort do array que contém as keys

Ciclo para dar print aos N primeiros utilizadores com maior avaliação média, ignorando os condutores inativos.

Q3 →

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.

```c
void q3 (Model m,char*buffer,FILE *fptr) {
    shiftLeft(buffer,2);
    int N = atoi(buffer);
    int size;
    if (m->keyArray_u==NULL)
      m->keyArray_u=u_give_array(m->users,&size);
    sort_array_users(m->keyArray_u,size,m->users,N);
    for (int i = 0; i<N;i++){
        Users u = look_up_users(m->users,m->keyArray_u[i]);
        if (strcmp("inactive",u_getAccount_status(u))==0){
            N+=1;
            continue;
        }
        double p = *u_getDistanceTotal(u) ;
        fprintf(fptr,"%s;%s;%.0f\n",m->keyArray_u[i],u_getName(u),p);
    }
    free_key_array(m->keyArray_u,size);
}
```

Sort do array que contêm as keys

Ciclo para dar print aos N primeiros utilizadores com maior distância viajada, ignorando os condutores inativos.

5. Interpretador (interpreter.c)

✓ Descrição

O interpretador é responsável pela interação entre o utilizador e o programa através de um ficheiro *input* e os respetivos ficheiros de *Output* gerado na diretoria “Resultados”.

Para este efeito, recorremos à função *interpretador* que recebe como argumentos o *path* para o *dataset* e o *path* para o input.

```c
void interpertador (char *dataset_input,char*inuput_file) {

char*l_users=g_build_path("/", dataset_input, "users.csv", NULL);
char*l_drivers=g_build_path("/", dataset_input, "drivers.csv", NULL);
char*l_rides=g_build_path("/", dataset_input, "rides.csv", NULL);


Model m =load_model(l_rides,l_drivers,l_users);
FILE *fp = fopen(inuput_file, "r");
size_t max_size = 128;
char *line = malloc(sizeof(char) * max_size);
    while (getline(&line, &max_size, fp) > 0) {

        start_querie(line,m);

    }
    fclose(fp);
    g_free(l_users);
    g_free(l_drivers);
    g_free(l_rides);
}
```

6. Main (main.c)

   ✓ Descrição

A main recebe os argumentos e invoca a função interpretador.

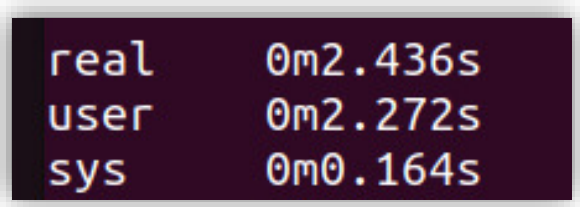
Aspetos a Melhorar

1. Memory Leaks

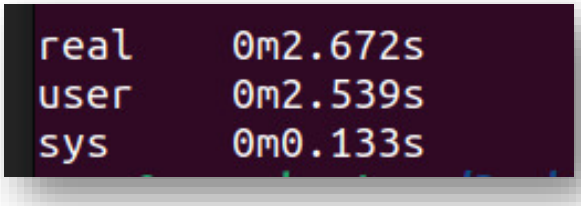
Apesar da maioria dos memory leaks não ser significativa, não conseguimos libertar as tabelas porque obtínhamos segmentation fault com a junção das funções criadas por nós com as funções de tratamento de estruturas da glib.

Custos Computacionais

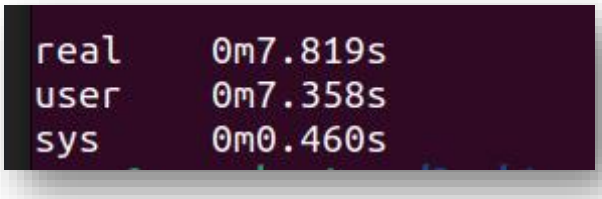
Execução do executável para o data-set fornecido gerado pela Makefile a correr numa máquina virtual Ubuntu 22.10 com 8Gb de RAM a correr num MacBook Air M1 2020.



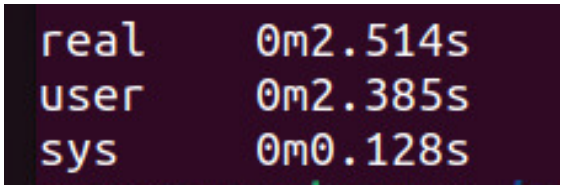
Q1 – Custos computacionais

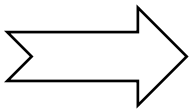


Q2 – Custos computacionais para o comando 2 50



Q3 – Custos computacionais para o comando 2 50





Parsing dos dados

Conclusão

Em suma, entendemos que a estruturação do projeto em 3 API's diferentes foi bem sucedida, na medida em que consideramos que a modularidade e o encapsulamento foram bem cumpridos. Neste sentido, apesar das dificuldades face à libertação de memória, julgamos que cumprimos com as instruções propostas e tencionamos aperfeiçoar na fase seguinte.