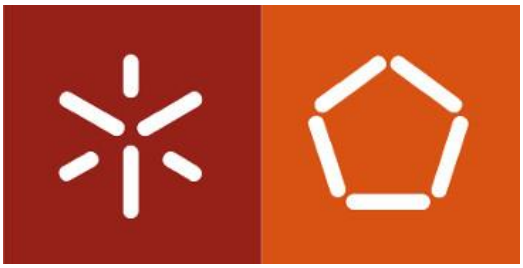

Departamento de Informática
Universidade do Minho

Sistema de gestão e consulta de recomendações de negócios na plataforma Yelp

2020/2021

Grupo nº 9

Rita Gomes, a87960

Leonardo Freitas, a93281

Joaquim Roque, a93310

Índice

1. Introdução

O presente relatório tem como objetivo descrever o processo de construção de um sistema de gestão de consulta de recomendações de negócios na plataforma 'Yelp'. Para proceder à resolução deste sistema é necessário proceder à construção de 3 modelos independentes (Users, Reviews e Businesses), que serão responsáveis por ler linha a linha cada user, cada review e cada negócio. Com a ideia de juntar todas as informações que são lidas linha a linha surgiram outros 3 módulos independentes (AllUsers, AllReviews, AllBusinesses).

Para evitar que possa haver alterações indevidas por agentes externos será fundamental garantir o encapsulamento dos dados armazenados no sistema, garantido assim a segurança do mesmo.

Ao longo deste relatório iremos descrever as abordagens adotadas pelo grupo para a resolução do problema proposto e, tal como é pedido, sustentaremos o mesmo com alguns testes de performance.

2. Formalização do problema

De uma forma geral, pretende-se exportar, dos 3 ficheiros dados para análise (users.csv, businesses.csv e reviews.csv), as seguintes queries:

- Dado o caminho para os 3 ficheiros (Users, Businesses, Reviews), ler o seu conteúdo e carregar as estruturas de dados correspondentes. Durante a interação com o utilizador (no menu da aplicação), este poderá ter a opção de introduzir os paths manualmente ou, opcionalmente, assumir um caminho por omissão.
- Determinar a lista de nomes de negócios e o número total de negócios cujo nome inicia por uma dada letra.
- Dado um id de negócio, determinar a sua informação: nome, cidade, estado, stars, e número total reviews.
- Dado um id de utilizador, determinar a lista de negócios aos quais fez review. A informação associada a cada negócio deve ser o id e o nome.
- Dado um número n de stars e uma cidade, determinar a lista de negócios com n ou mais stars na dada cidade. A informação associada a cada negócio deve ser o seu id e nome.
- Dado um número inteiro n, determinar a lista dos top n negócios (tendo em conta o número médio de stars) em cada cidade. A informação associada a cada negócio deve ser o seu id, nome, e número de estrelas.
- Determinar a lista de ids de utilizadores e o número total de utilizadores que tenham visitado mais de um estado, i.e., que tenham feito reviews em negócios de diferentes estados.
- Dado um número inteiro n e uma categoria, determinar a lista dos top n negócios (tendo em conta o número médio de stars) que pertencem a uma determinada categoria. A informação associada a cada negócio deve ser o seu id, nome, e número de estrelas.
- Dada uma palavra, determinar a lista de ids de reviews que a referem no campo text.

Para além da possibilidade de armazenar dados e efetuar queries sobre os registos contidos nos ficheiros de input, o programa a desenvolver deve também possibilitar a execução de comandos simples que permitam manipular os resultados das queries anteriormente descritas.

3. Conceção da Resolução

Nesta secção serão apresentadas as soluções usadas assim como a estruturação adotada pelo grupo para a resolução do problema.

3.1. Estruturas de Dados

Para guardar toda a informação necessária lida a partir dos ficheiros foram criadas HashTables, uma para cada estrutura de dados definida. Desta forma, temos os módulos: AllReviews, AllBusinesses, AllUsers, que são relativos a todas as reviews, todos os businesses e todos os users, respetivamente.

O grupo optou pelo uso de HashTable pelo simples facto de que o tempo de procura e inserção numa HashTable é (O(1)) e desde logo achamos que seria um bom argumento tendo em conta que um dos objetivos deste trabalho é a otimização. Salientar, ainda, que optamos por usar funções da biblioteca *GLib*, como recomendado pelos docentes.

3.2. Módulos e API

3.2.1. AllBusinesses

Este módulo contém uma estrutura com uma HashTable, no formato GHashTable*, ou seja, HashTables implementadas pela Glib, que irá conter toda a informação relativa aos negócios. Para além disto, a estrutura ainda inclui um char*, que corresponde a informação desnecessária para o funcionamento do módulo dai ter um nome tão vulgar como lixo.

```
struct allbusinesses{
   GHashTable* all;
   char *lixo;
};
```

Figura 1 - Parâmetros da Struct AllBusinesses

```
typedef struct allbusinesses *ALLBUS;

ALLBUS initALLBUS();
void insertBUS(ALLBUS, BUSINESS);
void destroyALLBUS(ALLBUS);
GHashTable* getALLBUS(ALLBUS);

char* get_lixo(ALLBUS);
void set_lixo(ALLBUS, char*);

int busFile(const char *, ALLBUS);
```

Figura 2 - Struct AllBusinesses e as funções principais do módulo

3.2.2. AllReviews

Esta estrutura, ao contrário da mencionada acima, apenas contem uma HashTable para guardar o necessário relativamente a todas as reviews feitas.

```
struct allreviews{
    GHashTable* all;
};
```

Figura 3 - Parâmetros da struct AllReviews

```
#define BUFFER_SIZE 1000000

typedef struct allreviews *ALLREV;

ALLREV initALLREV();
void insertREV(ALLREV, REVIEW);
void destroyALLREV(ALLREV);
GHashTable* getALLREV(ALLREV);

int revFile(const char *, ALLREV);
```

Figura 4 - Struct AllReviews e as funções principais do módulo

3.2.3. AllUsers

```
struct allusers{
    GHashTable* all;
};
```

Figura 5 - Parâmetros da struct AllUsers

```
#define BUFFER_SIZE 1000000

typedef struct allusers *ALLUSR;

ALLUSR initALLUSR();
void insertUSER(ALLUSR, USER);
void destroyALLUSR(ALLUSR);
GHashTable* getALLUSR(ALLUSR);

int userFile(const char *, ALLUSR);

#endif
```

Figura 6 -Struct Allusers e as funções principais do módulo

Esta estrutura segue exatamente o mesmo modelo da 3.2.2.

3.2.4. Business

Estrutura que contém todos os campos obrigatórios para o registo de um negócio.

```
struct business{
    char* business_id;
    char* name;
    char* city;
    char* state;
    char* categories;
};
```

Figura 7 – Parâmetros da struct Business

```
typedef struct business *BUSINESS;

BUSINESS initBUS();

char* get_busID(BUSINESS);
char* get_name_bus(BUSINESS);
char* get_city(BUSINESS);
char* get_state(BUSINESS);
char* get_categories(BUSINESS);

void set_busID(BUSINESS, const char*);
void set_name_bus(BUSINESS, const char*);
void set_city(BUSINESS, const char*);
void set_state(BUSINESS, const char*);
void set_categories(BUSINESS, const char*);

void destroyBUS(BUSINESS);

char* get_busID_and_name(const BUSINESS);
char* get_bus_info(const BUSINESS, const float, const int);
char* get_bus_info_param(const char*, const char*, const float);
char* format_city(const char*);
```

Figura 8 - Struct Business e funções principais desse módulo

3.2.5. Review

```
struct review{
    char* review_id;
    char* user_id;
    char* business_id;
    float stars;
    int useful;
    int funny;
    int cool;
    char* date;
    char* text;
};
```

Figura 9 - Parâmetros da struct review

```
typedef struct review *REVIEW;

REVIEW initREV();

char* get_reviewID(REVIEW);
char* get_userID_rev(REVIEW);
char* get_busID_rev(REVIEW);
float get_stars(REVIEW);
int get_useful(REVIEW);
int get_funny(REVIEW);
int get_cool(REVIEW);
char* get_date(REVIEW);
char* get_text(REVIEW);

void set_reviewID(REVIEW, const char*);
void set_userID_rev(REVIEW, const char*);
void set_busID_rev(REVIEW, const char*);
void set_stars(REVIEW, const float);
void set_useful(REVIEW, const int);
void set_funny(REVIEW, const int);
void set_cool(REVIEW, const int);
void set_date(REVIEW, const char*);
void set_text(REVIEW, const char*);

void destroyREV(REVIEW);

char* get_reviewID_and_text(const REVIEW, const char*);
```

Figura 10 - Struct Review e funções principais desse módulo

Estrutura que contém todos os campos obrigatórios para o registo de uma review.

3.2.6. User

```c
struct user{
    char* user_id;
    char* name;
    char* friends;
};
```

Figura 11 - Parâmetros da Struct User

```c
typedef struct user *USER;

/**Funções para lidar com a struct*/
USER initUSER();

/**Funções do tipo "get" para a struct.*/
char* get_userID(USER);
char* get_name(USER);
char* get_friends(USER);

/**Setters para a struct.*/
void set_userID(USER, const char*);
void set_name(USER, const char*);
void set_friends(USER, const char*);

/**Liberta a memória da struct.*/
void destroyUSER(USER);
char* formatUSER(const char*);
```

Figura 12 - Struct User e funções principais dos módulos

Estrutura que contém todos os campos obrigatórios para o registo de um user.

3.2.7. Main

```c
int main (){
    startController();
    return 0;
}
```

Figura 13 - Função para iniciar o programa

Este módulo apenas serve para correr o programa iniciando o modulo controller.

3.2.8. Controller

```c
int startController();
```

Figura 14 - Função principal para a execução de todo o projeto

Este módulo servirá apenas para assegurar a ligação entre o utilizador e o nosso sistema, isto é, todos os dados que o utilizador terá que inserir serão lidos aqui.

3.2.9. SGR

```
struct sgr{
    ALLUSR user;
    ALLBUS bus;
    ALLREV review;
};
```

Figura 15 - Parâmetros da struct sgr

Esta estrutura representará todo o nosso sistema e, por isso, irá englobar todas as HashTables referidas nos pontos acima, de forma a garantir o relacionamento entre os três principais modelos (Users, Reviews e Bussinesses). É neste módulo que iremos colocar o desenvolvimento das queries.

3.2.10. Table

```
struct table{
    char header[100];
    GPtrArray* lines;

    int intdata;
    char chardata;
    char* strdata;
    float floatdata;
};
```

Figura 17 - Parâmetros da struct table

Este módulo apresenta a estrutura que é responsável pela criação/destruição e da inserção de elementos nas tabelas que serão geradas por cada query.

3.2.11. View

```c
/**menus e outras tools**/

void menu();
void acedeMenu();
void printLinha(int);
void clear();
void printMensagem(char* string);
void promptError();

/** Navegador*/
void listDivider (GPtrArray* l, guint N, char* message, char* header, int q);

/**funções para apresentar o resultado das queries*/
void show(TABLE, int);
void showQuery2(TABLE p);
void showQuery3(TABLE info);
void showQuery4(TABLE list);
void showQuery5(TABLE avali);
void showQuery6(TABLE top);
void showQuery7(TABLE inter);
void showQuery8(TABLE avaliac);
void showQuery9(TABLE search);
```

Figura 18 - Funções apresentadas na view para exibir o menu o paginador e apresentar as querys.

Módulo que contém o menu de escolha da query a ser executada e o código relativo à paginação.

3.2.12. STARS

```c
typedef struct bus_info *INFO, *STARS;
```

Figura 19 - Estrutura Auxiliar

```c
struct bus_info{
    float stars;
    int num_of_revs;
    char *id;
    char *nome;
};
```

Figura 20 - Parâmetros da estrutura auxiliaar Stars

Esta Struct auxiliar localiza-se no módulo SGR onde o seu único propósito é auxiliar na execução das querys que necessitam de alguma espécie de ordenação, em que são comparadas e ordenadas as reviews, baseadas pelas estrelas atribuídas a cada negócio.

3.3. Desenvolvimento das Queries

Nesta secção serão mencionados os modos como foram executadas as queries pedidas no enunciado e qual o algoritmo desenvolvido para atingir o resultado esperado.

- **Query 1**

A query 1 consiste no carregamento da estrutura SGR, dado o path para cada um dos 3 ficheiros necessários. A query serve como ponto de ligação entre os módulos allUsers, allBusinesses e allReviews, invocando a respetiva função para a realização da leitura do ficheiro adequado, carregando, assim, cada uma das subestruturas. No caso do processo de leitura e carregamento de alguma destas falhar, a função interrompe a criação da estrutura, destruindo a mesma, se existente, e retornando ***NULL***, de forma a indicar o erro. Esta query é implicitamente executada no início de cada programa, de modo a carregar a estrutura SGR usando paths por omissão, tal como referido no enunciado.

- **Query 2**

A query 2, e maioria das que se seguem, partilham uma estrutura semelhante: inicialização da estrutura TABLE, carregar o cabeçalho adequado e, se necessário, dados adicionais para a mesma (neste caso, a letra pelo qual o nome dos businesses deverá começar para serem selecionadas) e iterar pela(s) subestrutura(s) de SGR adequada(s), fazendo uso de funções auxiliares aplicadas a cada par chave/valor das hash tables por meio de um *g_hash_table_foreach*. Se houver correspondência entre a letra submetida pelo utilizador e a primeira letra do nome do business, é então formatada a *string* com a informação necessária, de acordo com o enunciado, fazendo uso de uma função no módulo business. Esta é posteriormente adicionado ao array da TABLE.

De salientar que as linhas da TABLE são ordenadas lexicalmente no final de cada query, se assim se justificar.

- **Query 3**

Para esta query, calcula-se o número médio de stars das reviews cujo id de negócio corresponde ao id dado. Apesar de ser usado um iterador em vez de um *g_hash_table_foreach,* a abordagem ao problema é semelhante: iterar pela hash table das reviews e no caso de haver correspondência, atualizar a as variáveis auxiliares, ou seja, acrescentar o número de estrelas e incrementar o número de reviews. Percorrida toda a hash

table, é então calculado o número médio de estrelas e formatada a string com a informação pertinente, para depois guardar na TABLE.

- **Query 4**

Na query 4, é iterada a hash table das reviews, usando um iterador, e se houver correspondência com o user_id especificado, adiciona-se à TABLE a string devidamente formatada e com a informação do negócio adequada.

- **Query 5**

Para esta query, é novamente calculado o número médio de estrelas por meio de funções auxiliares e de uma hash table temporária, cuja chave é um id de negócio e o valor é uma estrutura STARS, ao qual se irá incrementar o número de reviews e somar as estrelas no caso de uma colisão. A hash table referida é então filtrada pelo número de estrelas, usando uma outra função auxiliar e guardando, de forma temporária, os valores na TABLE. Finalmente, filtra-se o array da TABLE, de acordo com a cidade dada e formatam-se as strings adequadamente.

- **Query 6**

A query 6 é eventualmente aquela que apresenta maior complexidade, pois requer que se relacione informação dos 3 módulos. Para isso, são usadas várias funções auxiliares e duas hash tables auxiliares: uma irá associar a cada id de negócio uma estrutura STARS (semelhante à query 5), e a outra irá associar a cada cidade uma lista de ids de negócio. De referir que a estrutura STARS inclui dois campos para o nome e para o id do negócio, que são usados na altura de adicionar strings à TABLE. Assim, a mesma estrutura irá conter toda a informação necessária. Carregadas as duas hash tables, cada array de cada cidade é consumido, procurando pelo id correspondente na hash das estrelas. Os arrays são então ordenados pelo número de estrelas e são selecionados os N melhores negócios.

- **Query 7**

Aqui, recorre-se novamente a uma hash table auxiliar. A cada id de utilizador associa-se uma lista dos ids de negócio aos quais este fez review. Depois, cada array de ids de negócio é consumido, substituindo o id pelo estado do negócio. Selecionam-se aqueles utilizadores cujo array associado contenha estados diferentes, através de uma procura linear.

- **Query 8**

A query volta a fazer uso de uma hash auxiliar, associando a cada negócio o número de estrelas, sendo depois filtrados aqueles que não pertencem à categoria. Para pertencer à categoria não basta ser uma substring das categorias associadas àquele negócio; verifica-se se os caracteres que precedem e sucedem o apontador da correspondência não são alfanuméricos. Cada um dos arrays da hash table é concatenado num só, sendo este ordenado e adicionam-se os N melhores negócios à TABLE.

- **Query 9**

Aqui, simplesmente itera-se pela hash das reviews e verifica-se se a palavra dada é uma substring do texto da review. De notar que, num processo semelhante ao da query 8, é feita a verificação de se os caracteres que precedem e sucedem o apontador não são alfanuméricos, de modo a garantir que a palavra existe mesmo e não apenas uma substring de uma outra palavra.

4. Testes e benchmarks

Para testar a execução do funcionamento do nosso projeto executamos diversos testes ao código para ver quais as funções que pesam mais na execução do programa.

Desta forma, para a medição dos tempos de execução utilizamos a biblioteca de C, *time.h,* e efetuamos alguns *benchmarks*, concluindo que os tempos obtidos são aceitáveis.

	1º teste	2º teste	3º teste
Query 1	3.48	3.47	3,5
Query 2	0.092	0.051	0.054
Query 3	0.258	0. 207	0.218
Query 4	0.208	0.205	0.207
Query 5	0.401	0.434	0.449
Query 6	0.724	0.925	0.936
Query 7	1.679	1.902	1.919
Query 8	0.562	0.555	0.587
Query 9	0.847	0.931	0.927

5. Conclusão

Com a realização deste trabalho foi possível consolidar e colocar em prática os conhecimentos obtidos ao longo não só da unidade curricular de Laboratórios de Informática III como também de Sistemas Operativos e Programação Imperativa. Para além disso, adquirimos um maior domínio da linguagem C, destreza no uso da mesma e também uma maior espontaneidade no processo de tomar decisões lógicas para programar aplicações software.

Tal como sabemos, o objetivo da programação em larga escala implica uma gestão cuidada dos grandes volumes de dados de forma a garantir o melhor desempenho possível.

No processo de desenvolvimento do projeto notamos algumas dificuldades nomeadamente desenvolvimento do interpretador, deste modo foi desenvolvido um método alternativo, através de um menu *user-friendly*, para que seja possível executar e ver o output de cada query. Para além disso, numa fase inicial demonstramos alguma indecisão na abordagem ao problema, nomeadamente na escolha da estrutura de dados para armazenar os dados. Realçar também uma certa inconsistência no que toca à execução das queries (utilização tanto de iteradores, como de *foreachs*).

Concluímos ter feito um bom trabalho face àquilo que nos foi proposto e consideramos que os conhecimentos adquiridos serão úteis em projetos futuros. No final de contas, o sistema está preparado para suportar todas as queries pedidas.