Laboratorios de Informática III

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

Source: https://www.yelp.com/dataset

Vasco Oliveira, a93208

Miguel Fernandes, a94269

Diogo Matos, a93234

09.04.2021

MIEI/LEI

Universidade do Minho

Introdução

Este trabalho tem como objetivo analisar e organizar grandes quantidades de dados de uma forma eficiente para ajudar no seu processamento.

Para tal foram usados os vários conhecimentos adquiridos de diferentes cadeiras bem como o nosso conhecimento sobre a linguagem de programação C.

Os dados utilizados referem-se à plataforma Yelp, para mais informações sobre estes visite este link.

Processo

APIs presentes e sua estruturação

Neste projeto tentamos sempre manter a modularidade e o encapsulamento, para tal organizamos os métodos da seguinte forma[1]:

- AuxStructs - Este módulo contém as funções que nos permitem comparar, organizar e transformar determinados dados para nos ajudar na realização das queries.
- Business - Este módulo contém funções relativas ao tipo de dados business e maneiras de retirar informações deste.
- Review - Este módulo contém funções relativas ao tipo de dado review e maneiras de retirar informações deste.
- User - Este módulo contém funções relativas ao tipo de dados user e maneiras de retirar informações deste.
- Prog - Este módulo contém a função main que irá correr o programa.
- Sgr - Este módulo contém as queries implementadas.
- Interpretador - Este módulo comunica com o utilizador chamando as queries demandadas por este.
- Table - Este módulo contém a estrutura de dados TABLE, bem como funções para apresentá-la e manipulá-la.

[1] Para mais informações relativamente ao funcionamento dos diferentes métodos ver a documentação fornecida nos includes(.h).

Estrutura dos dados mais significativos

User-

```
struct user

{

    char *user_id;   /** Apontador para um ID de um utilizador. */

    char *user_name; /** Apontador para o nome de um utilizador. */

    char *friends;   /** Hashtable de amigos de um determinado utilizador. */

};
```

AllUsers-

```
struct allUsers

{

    GHashTable *users_full; /** GHashTable de todos os utilizadores. */

};
```

Business-

```
struct business

{

    char *business_id;      /** Id de um dado business. */

    char *name;             /** Nome de uma dado business. */

    char *city;             /** Cidade de um dado business. */

    char *state;            /** Estado de um dado business*/

    GHashTable *categories; /**Categorias a que um business pertence */

};
```

AllBusiness-

```
struct allBusiness

{

    GHashTable *business_full; /** GHashTable de todos os businesses*/

};
```

Review-

```
struct review

{
```

```
    char *review_id;   /** Apontador do tipo char para o id da review. */
    char *user_id;     /** Apontador do tipo char para o id do utilizador. */
    char *business_id; /** Apontador do tipo char para o id do business. */
    double stars;      /** Número de estrelas de um dado business. */
    int useful;        /** Classificação de quão "useful" é um business. */
    int funny;         /** Classificação de quão "funny" é um business. */
    int cool;          /** Classificação de quão "cool" é um business. */
    char *date;        /** Apontador do tipo char para a data em que foi publicada a review. */
    GTree *text;       /** Apontador para uma GTree do texto da review. */
};
```

AllReviews-

```
struct allReviews
{
    GHashTable *reviews_full; /** GhashTable de todos os reviews */
};
```

Table-

```
struct table
{
    int num_lines, num_columns;
    char **column_names;
    GSList *list;
};
```

Line-

```
typedef struct line
{
    int num_columns;
    char **values;
} * LINE;
```

Descrição do funcionamento das diferentes queries.[2]

Query 1:

Esta query destina-se apenas à inicialização e leitura dos dados SGR.

Query 2:

A query 2 percorre sempre todos os elementos de business, na qual, para cada um compara a primeira letra do nome do negócio com a letra pretendida. É usada uma função de comparação que é case sensitive.

Query 3:

Nesta query são percorridas todas as reviews, em cada uma é comparado o business_id da mesma com o procurado. Caso exista compatibilidade entre os ids, a média é atualizada.

Query 4:

Para a resolução desta query, espelhamos o que foi feito na anterior mudando o fator de comparação. Para evitar o facto de retornar business_id iguais, implementamos uma hashtable organizada por este mesmo fator.

Query 5:

A query 5, uma vez mais, percorre todas as reviews. Para auxiliar no cálculo das estrelas de cada business_id é utilizada a estrutura de dados info_business. Esta estrutura vai ser o value de uma hashtable organizado pelos ids dos negócios. Em cada iteração esta estrutura de dados é incrementada em caso de repetição do id negócio ou então é criada. No fim esta Hashtable é toda percorrida em busca dos negócios com a média desejada.

Query 6:

A resolução deste problema passa por usar a query 5 numa escala maior. Cada nome de uma cidade vai ocupar uma posição numa HashTable em que o valor é outra Hashtable igual à referida na query anterior. No fim as Hashtable de cada cidade são percorridas e os value são guardados numa lista ordenada pelo número médio de stars.

Para esta query pensamos em utilizar uma lista logo desde o início. Porém uma procura e uma inserção numa hashtable é muito mais rápida que numa lista e o facto de continuamente aparecem business_ids repetidos, levou-nos a escolher a HashTable como forma de guardar os dados.

[2] Para mais informações mais informações sobre o funcionamento das queries ver sgr.c.

Query 7:

Uma vez mais usamos uma Hashtable para auxiliar na resolução do problema. À medida que as reviews são percorridas é guardada nesta Hashtable organizada pelo o User_id uma estrutura de dados com o nome de infoUser. Nesta estrutura será guardado o primeiro estado que apareceu na iteração dos user e um inteiro que define se este user já visitou mais que um estado.

Query 8:

Na query 8 optamos pela mesma estratégia utilizada na 6, ou seja, guardar todos os dados numa HashTable organizada pelo business_id e no value guardando o info_business. No fim guardamos os dados numa lista ordenada e guardamos na table os n primeiros.

Query 9:

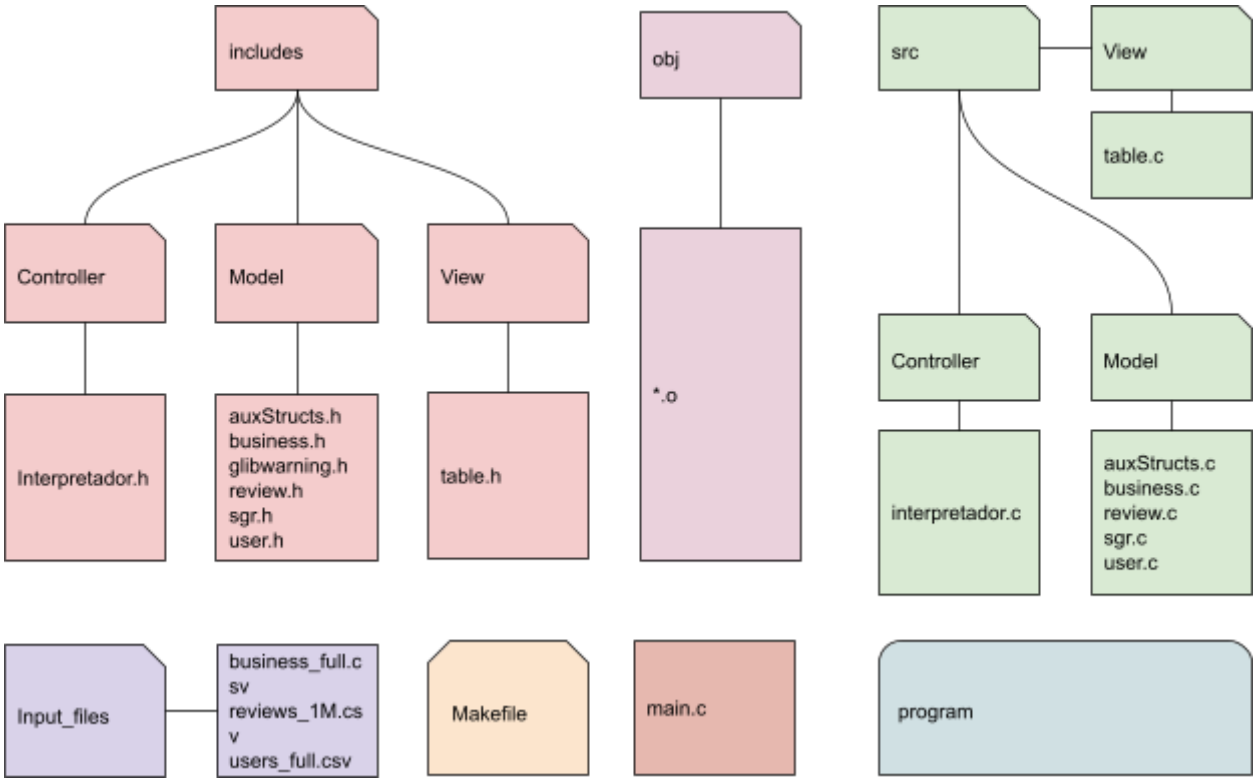
O facto de termos guardado os textos numa tree facilita muito mais a procura que em vez de procurar num array com N palavras irá procurar numa árvore, em que o tempo máximo é logarítmico.

Notas finais sobre as Queries:

A razão para as inúmeras utilizações das HashTables, prende-se no facto de que na maioria das queries é necessário realizar muitas procuras. Uma vez que numa HashTable bem implementada, o tempo médio de procura é de O(1), achamos que esta seria a melhor opção.

Arquitetura final da aplicação e razões para tal modularidade

Aqui encontra-se um diagrama da estrutura utilizada:



A estrutura utilizada permite isolar os ficheiros .c, .h, .o e a MakeFile de modo a poder efetuar alterações sem prejudicar as diferentes arquiteturas, bem como permite manter a organização estrutural intacta.

Testes de Performance das diferentes queries

Query 1-

Tentativa	Tempo de execução(em segundos)
1	33.0
2	32.8
3	32.8
média	32.9

Query 2(com letra s)-

Tentativa	Tempo de execução(em segundos)
1	0.39
2	0.38
3	0.36
média	0.38

Query 3(com business_id buF9druCkbuXLX526sGELQ)-

Tentativa	Tempo de execução(em segundos)
1	1.2
2	1.2
3	1.2
média	1.2

Query 4(com user_id RtGqdDBvvBCjcu5dUqwfzA)-

Tentativa	Tempo de execução(em segundos)
1	0.99
2	0.97
3	1.00
média	0.99

Query 5(4 estrelas e Atlanta)-

Tentativa	Tempo de execução(em segundos)
1	1.2
2	1.4
3	1.2
média	1.3

Query 6(top 10)-

Tentativa	Tempo de execução(em segundos)
1	1.8
2	1.8
3	1.9
média	1.8

Query 7-

Tentativa	Tempo de execução(em segundos)
1	5.0
2	4.9
3	5.0
média	5.0

Query 8(top 50 and Food category)-

Tentativa	Tempo de execução(em segundos)
1	2.5
2	2.5
3	2.4
média	2.5

Query 9(palavra summer)-

Tentativa	Tempo de execução(em segundos)
1	18.2
2	17.7
3	17.7
média	17.9

Resultados[3]

Query 2(com letra s)-

11277

Query 3(com business_id buF9druCkbuXLX526sGELQ)-

| Prides Osteria | Beverly | MA | 3.65 | 84 |

Query 4(com user_id RtGqdDBvvBCjcu5dUqwfzA)-

670 Businesses

Query 5(4 estrelas e Atlanta)-

7773 Businesses

Query 7-

19318

Query 9(palavra summer)-

9870

Conclusão

Devido ao restrito tempo, houve várias otimizações que não foram implementadas, como por exemplo, na query 9, pensamos em implementar uma Gtree para o texto em que cada value correspondia a posição em que a palavra ficaria na frase inicial, assim podendo recuperar o estado inicial do texto.

No entanto, após várias tentativas e tratamentos de erros, devido a impossibilidade de libertar memória(usando os frees disponibilizados na glib), que nos impediram de implementar corretamente esta estratégia, desistimos desta ideia.

No interpretador, também optamos por manter as funcionalidades básicas deste, sem

[3] Devido à extensão dos resultados da query 6 e 8 e a e ao número limitado de páginas, achamos melhor não a incluir nos resultados, e caso seja necessário será apresentada na execução do programa.

implementar o max e o min entre outros, problema que se resolveria com uma melhor organização estrutural do trabalho.

Numa nota final, achamos que em geral, o trabalho tinha um nível de dificuldade adequado, no entanto, para além das dificuldades em terminar o trabalho no prazo indicado, encontramos várias dificuldades a nível da organização do trabalho, principalmente na distribuição das tarefas, ficando um indivíduo(Miguel Fernandes) com a carga mais pesada.

Especificações de computador de teste

Memória: 15,5 GiB

Processador: Intel Core i7-8750H CPU @ 2.20GHz * 12

Placa gráfica: Intel UHD Graphics 630 (CFL GT2)

GNOME: 3.28.2

Sistema Operativo: 64-bits (UBUNTU 18.04.5 LTS)

Disco: 86,2 GB