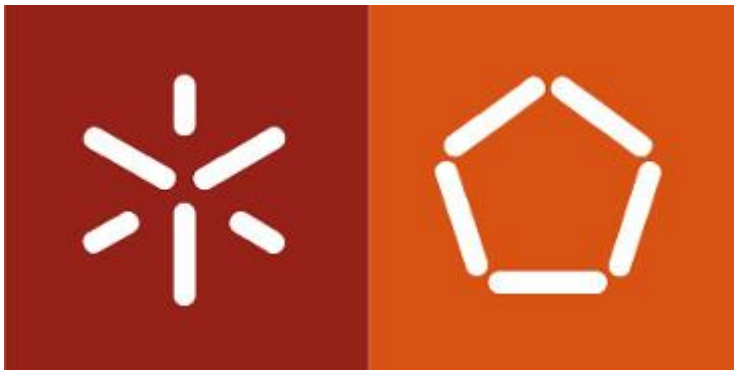

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática 3

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

Grupo 75

Arthur Cardoso Franca (a80011)

Duarte Nuno Pereira Moreira (a93321)

Lucas da Silva Carvalho (a93176)

Ano 2020/21

Índice

Introdução e desafios

Este é um projeto focado, maioritariamente, na capacidade de gestão e consulta de dados, realizado na linguagem de programação C. Sendo este um projeto de larga escala, ou seja, que trabalha com grandes volumes de dados, foi utilizado como forma de auxílio para a escrita de estruturas de dados, a biblioteca GLib da GNOME. As principais estruturas de dados utilizadas são Linked Lists e Hash Tables.

Numa fase inicial, uns dos primeiros obstáculos que encontramos foi a forma de como armazenar um grande volume de dados de maneira eficiente para poder executar as querys propostas no menor tempo possível. Para além disso, a correta gestão de memória atrasou o nosso projeto, uma vez que, na implementação de algumas querys era necessária a obtenção de uma grande quantidade de informação, ou seja, o programa acabava por "morrer" em tempo de execução, contudo conseguimos corrigir a maior parte destes problemas sendo possível efetuar todas as querys com a exceção da query 7, pois de todas é a que usa a maior quantidade de dados.

Módulos e Estruturas de Dados

a. User

Para armazenar um User, utilizamos uma estrutura de dados que contém duas strings, uma para o seu id, outra para o seu nome e uma lista ligada que irá guardar os id's dos seus amigos. Neste módulo são implementadas funções de inicialização e obtenção de informação.

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

b. Business

Para armazenar um Business, utilizamos uma estrutura de dados semelhante á dos Users, que para além de strings com o id e do nome desse negócio, também contém o nome da cidade, do estado e uma lista ligada com categorias. Também neste, são implementadas funções de inicialização e obtenção de informação.

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

c. Review

O mesmo método foi usado aqui, mas com informação relativa a uma review (review id, user id, business id, stars, useful, funny, cool, date e text). Mais uma vez, são implementadas funções de inicialização e obtenção de informação.

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

d. Catalogos: Users/Businesses/ Reviews

Nestes módulos são lidos e armazenados todos os users/businesses/reviews dos ficheiros em três catálogos distintos, para os quais utilizamos uma hash table em que o id de cada user/business/review é a respetiva chave. Aqui são implementadas funções de leitura, inicialização, validação e obtenção de informação.

e. Stats

Este módulo é responsável por efetuar relações entre os três principais modelos, proporcionando acesso mais rápido a dados e resultados pedidos nas queries do programa.

f. SGR

O módulo SGR (Sistema de gestão de recomendações) tem a função de suportar a implementação das queries propostas. Para tal, foi definido o tipo de dados SGR, que será uma junção dos três catálogos anteriores.

```
struct sgr{
    UserC catalogoUsers;
    BusinessC catalogoBusinesses;
    ReviewC catalogoReviews;
};
```

g. Table

Este módulo contém informação relativa ao tipo de dados conhecido pelo módulo SGR. Para a implementação do mesmo decidimos criar outro tipo de dados denominado coluna. Assim, uma coluna terá informação sobre o nome da mesma, a informação de cada campo e o número de linhas. Por sua vez, o tipo table será um array de colunas.

```
struct table{
    COLUNA * colunas;
    int colunasOcupadas;
};
```

```
struct coluna{
    char * topo;
    char ** campos;
    int numLinhas;
};
```

h. Interface

O módulo interface, tal como o nome indica, contém funções que tratam da parte visual do programa, como mostrar ao utilizador os resultados das queries, o menu inicial do programa, entre outros.

i. Controlador

Este será o módulo principal e tem como objetivo controlar o fluxo do programa, ou seja, é este que receberá as ordens do utilizador e as passará aos restantes módulos.

Arquitetura

Este projeto foi desenvolvido segundo a estrutura Model View Controller (MVC), e por isso, encontra-se dividido em três principais camadas, independentes umas das outras:

- Camada de dados e algoritmos, designada MODEL;
- Camada visual de interação com o utilizador, como por exemplo, menus, listas de resultados, etc, designado VIEW;
- Camada responsável pela ordem de execução correta da aplicação em função dos desejos dos utilizadores, designada CONTROLLER;

De forma resumida, o controlador recebe os pedidos do utilizador e envia-os ao modelo (executar querys, visualizar resultados, sair, etc), que por sua vez, executa este pedido e o encaminha para a camada view, que irá mostrar os resultados ao utilizador. Assim, estas camadas distintas funcionam em conjunto para formar o programa.

Complexidade das Estruturas

Como dito anteriormente, foi utilizado o tipo de dados hash table para armazenar a grande quantidade de volume de dados passados ao programa. Esta estrutura foi usada nos módulos catálogos e foi escolhida pelo facto de que tornaria o acesso a estes dados muito mais eficiente. Para a implementação destas tabelas, utilizamos como key (chave) o id do respetivo value (valor), que poderá ser um user, business ou review.

Para além disso, foi implementado o tipo de dados TABLE, como uma estrutura que armazena informação sobre os resultados obtidos nas queries, o que permitiu facilitar a sua apresentação. Este é constituído por um array de colunas e um inteiro, que indica o número o total das mesmas. Por sua vez, este tipo de dados COLUNA, apresenta, uma matriz de strings que guardará informação de cada campo da coluna, um inteiro que indicará o número de linhas da coluna, e uma string com o nome do tópico correspondente à mesma (ex: business_id, name, etc).

Também foi necessário implementar uma estrutura de dados para conseguirmos efetuar certas querys (6 e 8) de maneira eficiente.

No caso das querys 6 e 8, criamos uma struct chamada businessData que contem o id de um negócio, o seu nome, a sua média de estrelas (com base na pontuação das reviews) e o seu número de reviews, para assim criar uma tabela de hash. Deste modo conseguimos fazer com que o tempo de execução desta query fosse extremamente reduzido em comparação com o código inicial, que para alem de possuir funções auxiliares que por si só demoravam imenso tempo, também exigiam o processamento de uma grande quantidade de informação.

Testes de performance

Após a realização de vários testes obtivemos os seguintes resultados:

- Query 1

Valgrind	Tempo de execução
Sem	9s
Com	8min45s

- Query 2

Input	Tempo de execução
a	79.851ms
y	60.596ms
w	101.150ms

- Query 3

Input	Tempo de execução
6iYb2HFDywm3zjuRg0shjw	0.030ms
NlecnbisT1Qz5-Fl9q31ng	0.028ms
ngmLL5Y5OT-bYHKU0kKrYA	0.027ms

- Query 4

Input	Tempo de execução
RtGqdDBvvBCjcu5dUqwfzA	0.495026s
ZZw6jT9Sq81f0eVvIBu_pA	0.489019s
zZ-LTkL_SQgFjwF2MDW3kg	0.527532s

- Query 5

Input	Tempo de execução
5, Portland	0.684469s
4, Austin	1.032332s
2, Orlando	1.181436s

- Query 6

Input	Tempo de execução
10	2.011809s
100	2.019858s
1000	2.085279s

- Query 7

Input	Tempo de execução
10, Dentists	1.822568s
100, Dentists	1.821518s
1000, Dentists	1.847948s

- Query 8

Input	Tempo de execução
inGrace	1.680684s
and	1.918687s
the	1.800394s

Após a verificação destes resultados podemos afirmar que não existem grandes variações de tempo quando se altera o input de uma query, o que nos leva a crer que a procura de informação está a ser feita de forma eficiente. A query mais demorada aparenta ser a 6, demorando cerca de 2 segundos tanto como para inputs pequenos como grandes.

Recorrendo à ferramenta Valgrind verificamos que o nosso programa ainda apresenta falhas no que respeita a memory leaks. Estas acontecem aquando da execução das queries 6 e 8, uma vez que não foram terminadas de forma adequada. Ainda usando o valgrind é de notar que a query 1, que diz respeito à leitura dos ficheiros, apresenta uma diferença esmagadora no tempo de execução com o uso desta ferramenta. No entanto, após a sua execução verificamos que não existem memory leaks.

Conclusão

Com base no que foi dito anteriormente, e tendo em conta que não conseguimos acabar todos os requisitos do projeto (paginação, parte do interpretador de comandos e a query 7), sabemos que o nosso projeto podia estar mais completo, mas mesmo assim podemos afirmar que os requisitos que alcançamos estão, de certa forma bem conseguidos.

Apesar de algumas serem difíceis de compreender à primeira vista, utilizamos estruturas eficientes, sendo bastante simples e rápida a localização de um elemento em “listas” com milhares ou até milhões de elementos.

Quanto às querys, algumas têm um código extenso e difícil de compreender, contudo, estão todas a ser executadas em pouco tempo, com uma boa gestão de memória.