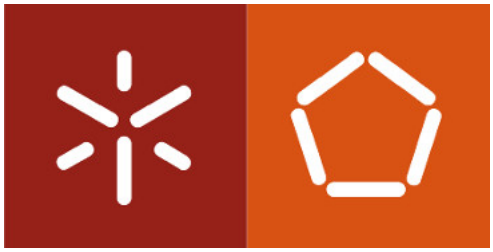

Universidade do Minho
Mestrado Integrado em Engenharia Informática

LI3 - Sistema de Gestão e consulta de Recomendações de negócios na plataforma Yelp

Grupo 31

André Silva (A87958) Armando Silva (A87949) João Nunes (A87972)

Maio 2021



Contents

1 Introdução

Este projeto, proposto na unidade curricular de Laboratórios de Informática III, consiste na elaboração de um Sistema de Gestão e consulta de Recomendações de negócios.

Para a elaboração deste projeto usamos uma arquitetura baseada em MVC (Model View Controller) e como é um projeto de programação em larga escala foi necessária a utilização de estruturas de dados eficientes que permitissem guardar e consultar grandes quantidades de informação. Para isso decidimos usar maioritariamente HashTables e GLists, disponibilizadas na biblioteca GLib da GNOME, tendo sempre em conta o encapsulamento, com vista a impedir possíveis alterações efetuadas por agentes externos.

2 Módulo e Estrutura de Dados

2.1 User

Para armazenar cada User usamos uma estrutura que guarda em Strings o seu ID, nome e, opcionalmente com a leitura dos seus amigos, uma HashTable que armazena tanto como key e value cada ID de cada amigo.

2.2 Review

Para armazenar cada Review usamos uma estrutura que guarda em Strings o seu ID, o ID do user, do business, date e text e guardamos em float ou inteiros, stars, useful, funny e cool.

2.3 Business

Para armazenar cada Business usamos uma estrutura que guarda em Strings o seu ID, nome, cidade, estado e um Array de Strings com as categorias.

2.4 Catálogo de Users

O Catálogo de User tem uma HashTable que armazena cada user, usando como key o seu ID e como value a estrutura User. Optamos pela HashTable, pois verificámos que não é necessário nenhum tipo de ordenação dos Users. Está definido da seguinte forma:

```
struct users {
        GHashTable *htUsers;
};
```

2.5 Catálogo de Reviews

O Catálogo de Reviews contém uma HashTable que armazena cada review, usando como key o seu ID e como value a estrutura Review.
Optamos pela HashTable, pois também verificámos que não é necessário nenhum tipo de ordenação das Reviews. Está definido da seguinte forma:

```
struct reviews {
        GHashTable *revs;
};
```

2.6 Catálogo de Businesses

O Catálogo de Businesses usa uma HashTable que armazena cada Business, usamos como key o seu ID e como value a estrutura Business.
Optamos pela HashTable, pois também verificámos que não é necessário nenhum tipo de ordenação das Businesses. Está definido da seguinte forma:

```
struct businesses {
        GHashTable *bizs;
};
```

2.7 SGR

Este módulo é constituído pelos três catálogos definidos acima. É este o módulo chamado pelo controlador para, por exemplo, responder às queries. Está definido da seguinte forma:

```
struct sgr {
        Businesses b;
        Reviews r;
        Users u;
};
```

2.8 Paginação

Este módulo tem como objetivo fornecer uma maneira fácil e intuitiva de mostrar e navegar pelos dados que o utilizador pediu. Possui quatro funcionalidades, avançar e recuar de páginas, mudar o número de entradas que o utilizador vê e voltar para o programa. De salientar que, antes de mostrar qualquer coisa, pede ao utilizador quantas entradas quer ver por página. Este módulo é chamado pela 'main' e invoca as funções do módulo da ui.

2.9 UI

Construímos este módulo para tratar da parte visual do programa. Fizemos, de forma simples, a paginação e os menus do programa.

3 Estrutura do Projeto

3.1 Encapsulamento

Restringimos o acesso aos nossos dados com alguns métodos de encapsulamento, como por exemplo as estruturas estarem definidas apenas no ficheiro .c o que não permite qualquer acesso direto às variáveis dos módulos.

3.2 Modularidade

Temos todos os dados em diferentes módulos, permitindo assim uma fácil expansão ou análise do código. Temos também a compilação separada dos respetivos módulos.

3.3 Model View Controller

Seguimos esta estrutura do seguinte modo:

- Model: composta pelo SGR, este que possui todos os módulos necessários para a sua realização, sendo estes, os catálogos de users, reviews e businesses.
- View: composta apenas pelo módulo UI.
- Controller: composta apenas pela Main, este que gere o tratamento dos pedidos e funcionamento do programa, iterage com os outros dois módulos.

4 Testes de Desempenho

Usando a biblioteca time.h fizemos várias medições do tempo que a execução do programa leva. O resultado de uma dessas medições encontra-se na tabela da figura seguinte.

	Businesses	Users	Reviews
A	50K	500K	100K
B	100K	1M	500K
C	160K	2M	1M
	A	B	C
Query 1A*	13.97s	26.31s	47.83s
Query 1B**	0.95s	2.82s	5.44s
Query 2	0.018s	0.043s	0.075s
Query 3	0.021s	0.15s	0.31s
Query 4	0.022s	0.15s	0.28s
Query 5	4.30s	75.82s	303.72s
Query 6	ind	ind	ind
Query 7	ind	ind	ind
Query 8	33.06s	532.48s	ind
Query 9	237.45s	ind	ind

Figure 1: Tempos de execução de cada query

(*), com leitura dos amigos (**), sem leitura dos amigos

Pela análise da tabela, podemos concluir:

- A leitura (query1) está bastante eficiente para qualquer tamanho dos ficheiros, têm um tempo baixo de leitura e armazenamento na respetiva estrutura.

- Da query2 à query4 como apenas existe interação entre duas hashtables (query3 e 4, businesses e reviews) estão, para nós, rápidas e eficientes tendo em conta que utilizamos a iteração de hashtables.

- Ficamos motivados a continuar a utilizar o mesmo método e infelizmente não deu os mesmos resultados. Depois de uma pesquisa mais exaustiva sobre as hashtables, descobrimos que o tempo de iteração e procura é O(n*n). Sem dúvida que o tempo tão demorado destas querys é devido a isso, e que apesar de não termos tido tempo suficiente para fazer todas as alterações, temos noção que a utilização de uma estrutura auxiliar com arrays teria sido muito mais eficiente.

Depois destes testes, realizámos também um teste de memória com a ferramenta *Valgrind*. Chegamos à conclusão que apenas existem *memory leaks* na leitura dos users, devido à falta de um *destroyer* da value do user, lapso que não observamos a tempo da entrega.

5 Conclusão

Em suma e tendo em conta os objetivos e desafios que este projeto continha, achamos que o mesmo se encontra num nível mediano. No nosso ponto de vista, a ineficiência que algumas querys têm, diminui bastante a avaliação e possível valorização do projeto. Tal problema poderia ter sido resolvido se tivéssemos optado por outros algoritmos na resolução das querys, como observamos mais detalhadamente na secção acima.

Se tivéssemos a hipótese de voltar a fazer o projeto do zero, todos nós concordámos que iriamos pensar e dedicar mais tempo à análise de todas as estruturas disponíveis, e só depois escolher as que melhor se adaptam ao projeto. Como também analisar melhor todas as maneiras possíveis de realizarmos as querys em questão.

No que toca aos restantes objetivos do projeto sentimos que conseguimos cumprir com os paradigmas do encapsulamento e da modularidade, e organizar o programa na estrutura MVC.

Concluindo, o grupo sente que este trabalho ajudou a consolidar as bases na linguagem C, especialmente devido ao uso do *Valgrind*, ferramenta que nunca tinhamos utilizado, e que é extremamente útil para percebermos erros de memória que de outra forma não seria possível.

6 Grafo de Dependências

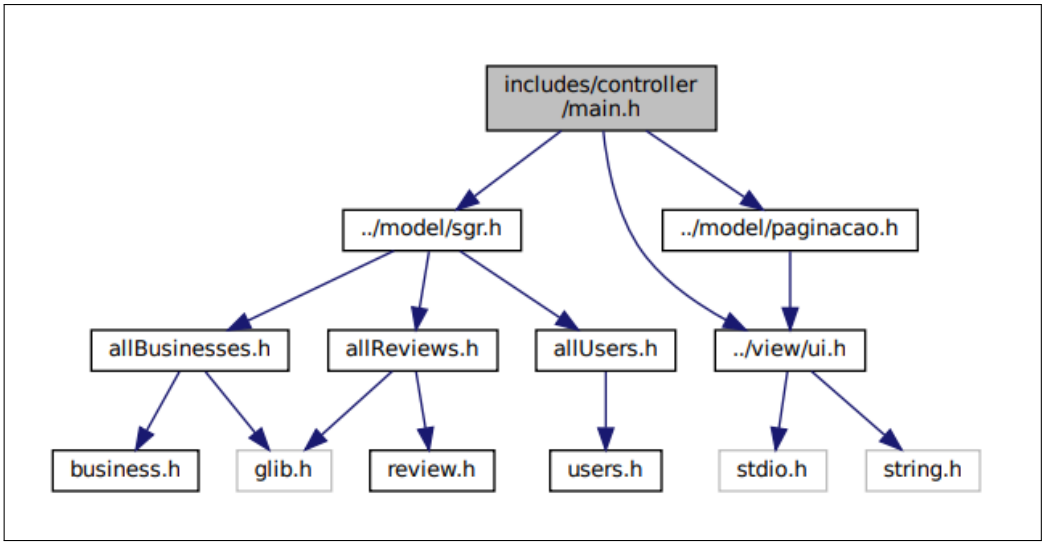

Figure 2: Grafo de dependências