

UNIVERSIDADE DO MINHO

Laboratórios de Informática III

Projeto C

Grupo 46

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

19 de maio de 2021



Patrícia Pereira, A89578



Meriem Khammasi, A85829



Alexandra Candeias, A89521

Conteúdo

1	Introdução	1
2	Modularidade e Encapsulamento	1
3	Módulos e Estruturas de dados	1
3.1	AVLTree	2
3.2	Catálogos	2
3.3	<i>User, Business e Review</i>	3
4	Estrutura do Projeto e Grafo de Dependências	3
5	Funcionalidades	4
5.1	Query 1	4
5.2	Query 2	4
5.3	Query 3	5
5.4	Query 4	5
5.5	Query 5	5
5.6	Query 6	6
5.7	Query 7	6
5.8	Query 8	6
5.9	Query 9	6
5.10	TABLE	7
6	Interpretador de comandos	7
6.1	Interação com o Utilizador	8
7	Testes de Performance	9
7.1	Tempos de Execução	9
8	Makefile	10
9	Conclusão	10

Lista de Figuras

1	Estrutura e Desenho da AVL e NodeAvl	2
2	Desenho da estrutura Catalog	2
3	Estruturas de User-Business-Review e respectivos catálogos	3
4	Grafo de dependências	4
5	Mudança Query 4	5
6	Correção do Interpreter	7
7	Menu Inicial e Menu da Query 2	8
8	Tempos de Execução	9
9	Tempos de Execução	9

1 Introdução

No âmbito da unidade curricular de Laboratórios de Informática III, foi-nos proposta a criação e desenvolvimento de um sistema de resposta (a *queries*) sobre um Sistema de gestão e consulta de recomendações de negócios na plataforma Yelp.

Para a primeira fase deste projeto, o desafio era implementar este sistema na linguagem C. Apesar da intenção de tornar rápida a execução das funcionalidades do programa a ser desenvolvido, dois dos focos principais deste projeto foram a Modularidade e o Encapsulamento das estruturas de dados por nós utilizadas.

Ao longo do desenvolvimento do projeto, consideramos o nosso maior desafio implementar as estruturas onde seria organizada a informação, de forma a o seu acesso ser rápido e eficiente.

Além disso, também consideramos um desafio a libertação de memória do programa.

2 Modularidade e Encapsulamento

A construção e o desenvolvimento de programas de dimensões já consideráveis, qualquer que seja a linguagem, envolvem a utilização de técnicas particulares que possam garantir que os projectos de software, apesar das suas dimensões, são controláveis e geríveis, quer no seu desenvolvimento quer no seu teste e manutenção.

Ao escrevermos o nosso programa, tivemos a preocupação não só de dividir o código fonte em módulos auto-suficientes, reutilizáveis e fáceis de manter e de corrigir ou aumentar, mas também a utilização de ficheiros (.h) que não incluem instruções, mas apenas protótipos e assinaturas das funções.

Assim, com o uso da modularidade e abstração de dados, conseguimos obter um código seguro, de maneira a proteger não só do utilizador como de alguém que pretenda aceder e alterar dados.

3 Módulos e Estruturas de dados

A abordagem feita às estruturas de dados é uma das fases mais importantes na realização do projeto.

Para responder às *queries* descritas no enunciado, foi necessária a utilização de estruturas de dados com o fim de organizar a toda informação contida nos ficheiros CSV fornecidos.

Assim, começamos por analisar como eram constituídos os utilizadores, os negócios e os reviews bem como o que nos era pedido em cada query de forma a percebermos quais eram as informações às quais iríamos recorrer constantemente.

O nosso projeto tem por base três módulos principais, sendo eles o dos utilizadores, o dos negócios e o das reviews. Assim, para cada um deste tipo de dados criamos uma estrutura própria.

3.1 AVLTree

Decidimos recorrer à utilização de AVLTree- árvores binárias balanceadas - para organizar a informação, uma vez que as consideramos de fácil utilização na inserção e rápida na procura de elementos.

Este módulo contém as funções de implementação de uma AVLTree balanceada. Cada estrutura AVLTree é constituída por um nodeAVL, e o tamanho da mesma. A estrutura nodeAVL pelo peso (*height*) do um apontador para a chave (*key*), apontadores para as estruturas de um nodeAVL da direita e outro para o nodeAVL da esquerda e, um void que permite associar ao nodeAVL um apontador para uma qualquer outra estrutura, garantindo que este módulo seja a base da implementação dos restante módulos.

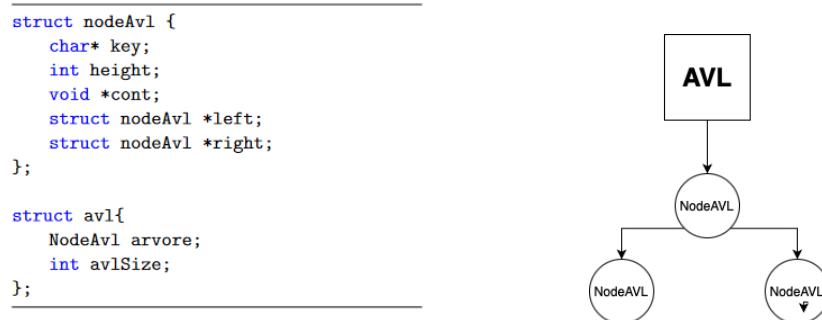


Figura 1: Estrutura e Desenho da AVL e NodeAvl

3.2 Catálogos

Foi-nos pedido para criar um catálogo de utilizadores, de negócios e de reviews.

Deste modo, um dos nossos principais desafios foi criar uma estrutura para catálogos que fosse a mais eficiente possível e que, ao mesmo tempo, nos ajudasse a tratar as informações dadas.

Chegamos à conclusão que uma boa opção seria utilizar um array de AVL's com 38 posições, organizando cada uma das estruturas pelo caracter inicial da key do nodoAVL, correspondentes aos Ids de users, business e reviews, sendo assim as primeiras 26 posições para as letras de A a Z, e as seguintes 10 para os números de 0 a 9, uma para o caracter '-' e, por fim, a última posição para qualquer outro caracter que não se enquadra nestes.

Esta escolha permite-nos obter níveis de eficiência muito elevados em termos de funções tais como inserções, remoções, alterações de conteúdo que iriam ser recorrentemente necessárias ao longo do desenvolvimento deste projeto.



Figura 2: Desenho da estrutura Catalog

3.3 *User, Business e Review*

Os módulos *User*, *Business* e *Review* contém estruturas que guardam todas as informações relativas a implementação destes, tal como se encontravam nos ficheiros CSV fornecidos.

Além das funções de inserção, remoção e alterações de conteúdo feitas com as funções definidas no *Catalog*, definimos getters dos parâmetros das estruturas que retornam sempre que possível uma cópia dos dados de modo a permitir o encapsulamento destes mesmos e assim facilitar a implementação das *queries*.

```
struct user{
    char* userID;
    char* username;
    char** friends;
    int tam;
};

struct catUsers{
    Catalog catalog;
};

struct business{
    char* businessID;
    char* bname;
    char* city;
    char* state;
    char** categorys;
    int tam;

    char** reviews;
    int qtdReviews;
};

struct CatBusinesses{
    Catalog catalog;
};

struct review{
    int a;
    char* reviewID;
    char* userID;
    char* businessID;
    float stars;
    int useful;
    int funny;
    int cool;
    char* date;
    char* text;
};

struct catReviews{
    Catalog catalog;
};
```

Figura 3: Estruturas de User-Business-Review e respetivos catálogos

4 Estrutura do Projeto e Grafo de Dependências

O Modelo do programa é gerido pelo módulo *sgr.c*. Este módulo contém a estrutura SGR e chama os módulos *user*, *business*, *review* e *querieStr* para fazer qualquer alteração necessária aos dados do programa (carregar a estrutura a partir de ficheiros, libertar memória, devolver dados, etc.).

Os dados são os catalogos de users, de reviews e de business, a quantidade total de users, businesses e reviews lida, bem como o caminho para cada um dos ficheiros que lemos e, por fim, um inteiro sendo este uma flag que indica se a estrutura foi inicializada corretamente.

A Vista do programa é controlada pelo módulo *menu.c*. Este módulo contém todas as funções que devolvem resultados visuais ao utilizador (resultados de queries, menus, etc.).

O Controlador do programa é gerido pelo módulo *-interpreter.c*. Este interage com o módulo Menu para devolver resultados visuais ao utilizador e com o módulo SGR para ter acesso às *queries* pretendidas. Além disso, o Interpreter também chama funções do módulo *querieStr* que recebem os dados para a execução das *queries*.

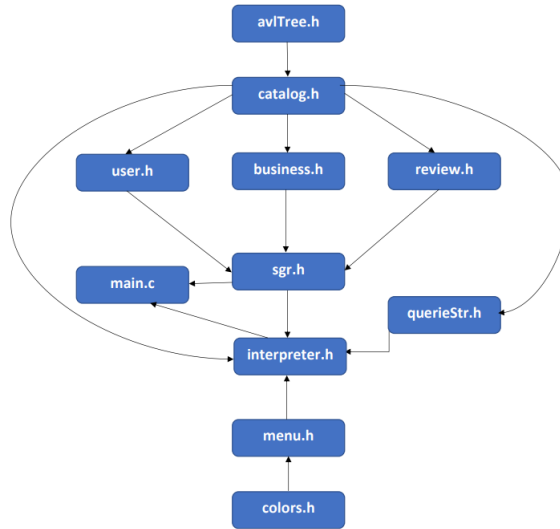


Figura 4: Grafo de dependências

5 Funcionalidades

Nesta secção iremos explicar a nossa abordagem perante as queries. O nosso pensamento na realização destas foi utilizar os catálogos e percorre-los de forma a obter as informações que pretendemos, inserindo-as em elementos da estrutura TABLE.

5.1 Query 1

Esta query é reativa ao carregamento dos ficheiros CSV. Como referido anteriormente usamos a estrutura de dados AVLTree para inserir os elemento, permitindo-nos ordenar a informação. Decidimos como melhor abordagem ordenar pelos IDs de User, Business ou Review.

Estes são inseridos nas distintas estruturas: Catálogo de Users, Catálogo de Businesses e Catálogo de Reviews, juntamente com o resto da informação presente nos ficheiros de cada. Estes são armazenados no SGR juntamente com o número total de Users, Businesses e Reviews.

No fim alteramos o valor da flag de validação, que é inicialmente 0, que nos permite saber se é possível utilizar as restantes queries.

5.2 Query 2

Nesta query é nos pedidos que retomenmos uma lista de todos os negócios cujo nome uma determinada letra e o seu número total.

Utilizando o catálogo de Businesses, inserimos o nome dos negócios percorridos pela letra a um array de Strings da estrutura TABLE.

5.3 Query 3

Nesta query pretendemos recolher a informação sobre um dado negócio.

Utilizamos novamente o Catálogo de Businesses, inserimos as informações nome, cidade, estado e quantidade de reviews na estrutura TABLE do negócio pretendido.

5.4 Query 4

Com esta query pretendemos saber o nome e os ids de negócio cujo um certo utilizador tenha realizado uma review.

Inicialmente acedemos ao Catálogo de reviews, e inserimos o businessId presente na estrutura do associada ao id do review dado a um array de Strings da estrutura TABLE. Para saber o nome dos negócios com esses Id's, percorremos a lista de Id's e a cada Id, percorrendo o catálogo de Businesses e inserimos o nome do negócio a outra lista contida na mesma estrutura TABLE. No print desta query apercebemo-nos que estamos a dar o print da mesma lista duas vezes. Sendo isto revolido mudando um get da lista como mostra na figura.

```
char** dID = getIdList(q);  
//char** dName = getIdList(q);  
char** dName = getList(q);
```

Figura 5: Mudança Query 4

5.5 Query 5

Atendendo ao pedido da query, que é listar todos os negócios de uma cidade que tenham número de estrelas maior ou superior ao pretendido, o nosso pensamento foi, primeiramente, aceder ao catalogo de Businesses e inserir todos os Id's cuja cidade fosse a pretendida, para isto usamos um processo muito semelhante ao explicado a cima. Tendo a lista de Business de uma cidade, acedemos ao catálogo de reviews e percorremos essa Lista. Com o auxilio de uma estrutura Pair contamos o número total de reviews feitas nesse negócio e o numero total de estrelas dados. Caso a media das estrelas do negócio fosse superior ou igual às estrelas dadas pelo utilizador adicionavamos o Id do utilizador à lista de uma nova TABLE. Para Saber os nome dos negócios usamos a mesma função usada na segunda parte da query 4.

Esta resolução não se encontra no último commit realizado no gitHub, foi entregue pela Alexandra Candeias no dia 5 de maio pelas 22horas e 40 minutos, no entanto não conseguimos encontrar esse commit.

5.6 Query 6

Esta query não foi enviada (encontra-se em comentário), uma vez que não estava a compilar, admitimos a não ter abordado esta query da maneira mais eficaz.

A query pedia um top de negócios de cada cidade. A abordagem pensada por nós foi criar uma lista com todas as cidades através do catálogo de businesses, com a função auxiliar `getAllCities` que se encontra no `business.c`. Com o catálogo de reviews percorriamos essa lista e utilizávamos a query 5 para, onde começávamos por procurar os negócios com 5 estrelas, acrescentando-o à lista da estrutura `TABLE`, e usávamos um contador para saber se já tínhamos a quantidade de negócios pretendida, se não, diminuíamos o número de estrelas em 0.1.

5.7 Query 7

Nesta query é nos pedido que listemos todos os utilizadores que tenham feito reviews em mais de um estado.

Na resolução desta query, acedendo ao catálogo de users, criamos uma lista com todos os users. Percorrendo essa lista, e usando como auxiliar a Query 4, criamos uma lista com os Id's de negócios que esses users, fizeram uma review. Com esta lista, verificamos se os Estados dos negócios eram de estados diferentes, usando o processo parecido ao já explicado.

Esta função não funciona, devido à falta de duas linhas de código quando acedemos ao catálogo. Esta query não está resolvida da maneira mais eficiente pelo que o programa é *killled*

5.8 Query 8

Esta query também não foi enviada. O pretendido era fazer um top de negócios para uma categoria. O nosso pensamento para esta query foi inicialmente ir buscar todos os Ids de negócios de uma categoria, com a função auxiliar `'getBusinessFromCategory'` do módulo `business.c`.

Com esta lista iríamos usar um processo semelhante ao da Query 5, com a função em comentário `'getBusinessAverageStars'` do módulo `review.c`.

5.9 Query 9

A Query 9 pretende saber quais as reviews que contêm uma palavra. Para isto, percorremos o catálogo de reviews, e comparamos o texto de cada `nodaAVL` com a palavra dada pelo utilizador, no caso da palavra estar presente no texto adicionamos o id da review à lista da estrutura `TABLE`. Esta query, no entanto, não está completamente correta, adicionando à lista elementos diferentes de id's de reviews (print de uma variável `x` que aparece no meio da lista).

5.10 TABLE

Esta Estrutura é usada para valor de retorno das queries. Esta é composta por um `char*` variável correspondente à variável dada pelo utilizador no interpretador, um `int` query que nos permite saber qual query é a ser invocada. e um `char*` path que seria o caminho de um ficheiro para as funções `toCSV` e `fromCSV`, que não estão implementadas.

De seguida, temos duas listas e os respetivos tamanhos para listar os ids ou nomes de users, negócios ou reviews, conforme pedido pela query.

Contém também um `bname`, `bcity`, `bstate` e um `int` `breviews`, para solução da query3.

```
struct table{
    char* variavel;
    int query;
    char* path;

    char** list;
    int numTotal;

    char** idList;
    int numTotalID;

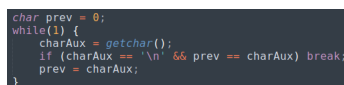
    char* bName;
    char* bCity;
    char* bState;
    int bReviews;
};
```

6 Interpretador de comandos

Os nosso interpretador não está como desejávamos uma vez que foi desenvolvido muito próximo da entrega. Isto deveu-se a um erro da nossa parte, uma vez que não interpretamos o enunciado da melhor maneira, pensando que o interpretador seria mais simples, tendo inicialmente criado um interpretador onde o utilizador apenas escolhia a funcionalidade pretendida e depois era pedido o input (ex. letra pretendida).

Este erro fez com que o nosso foco nos dias anteriores à entrega fosse quase exclusivamente dirigido à mudança do interpretador, fazendo com que nos desleixássemos noutros assuntos, tal como a realização e testes das queries, criação dos diferentes comandos, performances e tempos de execução a tempo da entrega, a documentação, e pequenos detalhes no código...

Nesta parte do projeto falhamos ao guardar as diferentes variáveis, uma vez que quando é mostrado o resultado de uma query, o programa falha, dando `segmentation fault` ao pressionar a tecla `j`, devendo-se isto a um `else` e vários `while`'s fechados no sítio errado na transição entre o `if` da funcionalidade `show` e o `else` que vem a seguir. Foi acrescentando também um `while` no fim da função que lê dois enters como apresentado na figura a baixo.



```
char prev = 0;
while(1) {
    charAux = getchar();
    if (charAux == '\n' && prev == charAux) break;
    prev = charAux;
}
```

Figura 6: Correção do Interpreter

6.1 Interação com o Utilizador

Quando o utilizador corre o programa, é lhe apresentado o menu com os comandos possíveis de utilizar.

Para conseguir executar os comandos é preciso efetuar a leitura dos ficheiros através do comando 1 em primeiro lugar, depois, atribuir o valor da query pretendida e por fim, para visualizar o valor da variável utilizar o comando *show* que retorna o resultado visual da query.

Para sair do programa basta escrever *quit*;

O toCSV e o fromCSV não estão funcionais. Ao escrever 'x=fromCSV(sgr,text.txt);' ou algo do género apenas guardamos o caminho text.txt na estrutura TABLE, apesar de não ser exatamente o pedido no enunciado nesta operação.

```

      YELP
-----

**** Bem vindo ao YELP ****
** Sistema de gestão e consulta de recomendações de negócios **

Escolha a operação que pretende realizar

Ler ficheiros csv
Clique <1><enter> para carregar os ficheiros

Listar e consultar numero de negócios por letra
Sintaxe a usar: x=businesses_started_by_letter(sgr,'letra pretendida');

Consultar negócio
Sintaxe a usar:
x=business_info(sgr,'Id do negócio pretendido');

Listar negócios por avaliados por um utilizador
Sintaxe a usar:
x=businesses_reviewed(sgr,'Id do utilizador pretendido');

Listar negócios por stars e cidade
Sintaxe a usar:
x=business_by_star_and_city(sgr,nºEstrelas,'nome da cidade pretendida');

Top negócios por cidade
Sintaxe a usar:
x=top_businesses_by_city(sgr,nº de negócios para listar);

Listar e consultar número de utilizadores por cidade visitada
Sintaxe a usar:
x=users_multiple_states(sgr);

Top negócios por categoria
Sintaxe a usar:
x=top_businesses_by_category(sgr,'categoria pretendida');

Lista de reviews por palavra-chave
Sintaxe a usar:
x=reviews_with_word(sgr,'palavra pretendida');

Sair
Sintaxe a usar: quit;

>

                                RESULTADOS DA QUERIE 2
* Lovely Nails & Skin Care
* Lord's Tree Service
* LensCrafters
* Lucky Nails
* Llana's Peruvian Creole
* Lord Hobo
* Le Bleu
* Lice Clinics of America - Portland
* Life Time Athletic Burlington
* Le Do Vietnamese Restaurant
* Louisville Recreation & Senior Center
* Lush Handmade Cosmetics
* Levi's Store
* Little Mexico Cantina
* Los V Tires & Auto Repair
* Lime Fresh Mexican Grill
* Local Slice II
* Lemon Grass Fusion Bistro
* Lavista Animal Hospital
* LaRog Brothers Jewelers
* Lucky's Market
* Lalo Salon
* La Molisana Ristorante
* LabCorp
* Lion's Share Digital
* Lanvin French Bakery
* Lafayette Eye Associates
* Little Caesar's Pizza
* LittleMoon Cafe and Tea
* Luscious Salon
* LaserCare Cosmetic Centers - Brookline
* Lifestyle Nail Bar
* Lake Oswego Pet Sitting
* Lighthouse Inn Restaurant and Bar
* Lovecup
* Lilo's Plates
* Lianna Vines Massage, Health and Wellness
* Little Szechuan
* Little Caesar's Pizza
* Luna on Park

1/170(40)
TOTAL DE NEGÓCIOS COMEÇADOS POR L : 6764
1 <- previous || 2 -> next
3 <- (-) negócios || 4 -> (+) negócios
```

Figura 7: Menu Inicial e Menu da Query 2

Se o utilizador executar uma opção que devolve uma lista de negócios ou utilizadores, depara-se-á com o menu de páginas, onde pode utilizar os comandos para navegar nestas, conforme a imagem da query 2 apresentada em cima.

7 Testes de Performance

7.1 Tempos de Execução

Posteriormente ao desenvolvimento e codificação de todo o projeto, foi-nos proposto realizar testes de performance que consistem na obtenção do tempo de carregamento dos ficheiros e dos tempos de execução das queries 1,2,3,4 e 9 para o qual usamos a biblioteca standard de C `time.h` conforme as imagens seguintes que demonstram a maneira que foi aplicada para obter os resultados eficientemente no terminal.

```
clock_t t1;
clock_t t2;
clock_t t3;
clock_t t4;
clock_t t5;
clock_t t6;
clock_t t7;
clock_t t8;
clock_t t9;
double time_taken;

time_taken = ((double)t2)/CLOCKS_PER_SEC;
printf("Query 2 : Load : %f seconds\n", time_taken);
t2 = clock();
q = businesses_started_by_letter(sgr, letra);
t2 = clock() - t2;
```

Figura 8: Tempos de Execução

O resultado do tempo de carregamento dos 3 ficheiros consecutivos foi de 23,68 segundos. Já em relação às restantes queries estão demonstrados os resultados no gráfico abaixo

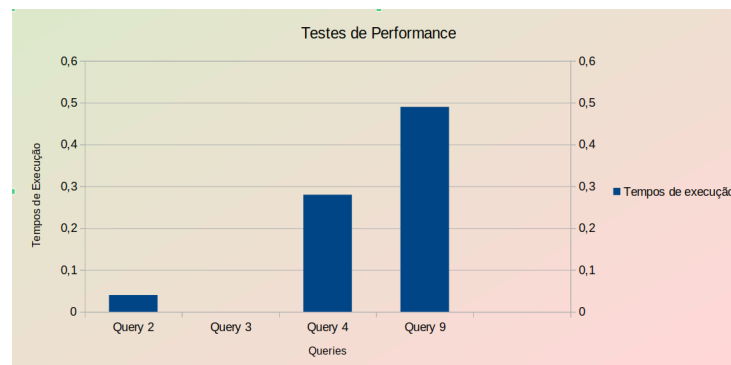


Figura 9: Tempos de Execução

O tempo de execução da query 3 é de 0.000034 segundos, sendo este muito reduzido uma vez que o programa não precisa de percorrer o catálogo de negócios inteiro.

8 Makefile

```
### Makefile ###

CC = gcc                                //Compilador a utilizar

CFLAGS = -Wall -O3 -ansi -D_GNU_SOURCE -g -std=c99          //Flags de compilação

INCLUDE = -I include                                     //Flag para incluir header files
SRC := src                                                //Diretória dos source files
OBJ := obj                                                //Diretória dos object files

SOURCES := $(wildcard $(SRC)/*.c)                        //Source files
NAME = program                                           //Nome do executável

OBJECTS := $(patsubst $(SRC)/%.c, $(OBJ)/%.o, $(SOURCES)) //Object files

program: $(OBJECTS)
        $(CC) $(CFLAGS) $(INCLUDE) -o $(NAME) $(OBJECTS) //Compila o programa

$(OBJ)/%.o: $(SRC)/%.c
        $(CC) $(CFLAGS) $(INCLUDE) -c $< -o $@          //Gera os object files

$(shell mkdir -p $(OBJ))                                //Cria uma diretoria para os objetos

clean:
        rm -r $(OBJ)
        rm -f program
        rm -f gesval                                     //Remove o executável e os object files
```

9 Conclusão

Para finalizar, cumprimos vários requisitos, tal como o funcionamento das *queries*, no que consideramos ter abordado de uma forma eficiente e bem estruturada, apesar de nem todas estarem funcionais.

Achamos ter conseguido fazer um bom trabalho na modularidade e encapsulamento, e também criar código reutilizável, apesar de considerar-mos poder feito ainda melhor neste sentido.

Onde nos deparamos com maiores dificuldades foi no interpretador, devido ao referido na secção dirigida a este.

Apesar de todos os obstáculos, fomos capazes de realizar um programa funcional respeitando os principais requisitos pedidos.