Laboratórios de Informática III

MIEI/LEI

2020/2021

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

Introdução e Objetivos

O projeto de C da disciplina de LI3 tem por objetivo fundamental ajudar à consolidação experimental dos conhecimentos teóricos e práticos adquiridos nas UCs anteriores. A estes conhecimentos acrescenta-se a introdução de novos conceitos fundamentais da área de Engenharia de Software e o desafio de processar grandes volumes de dados.

Os objetivos de LI3, e deste trabalho, não se restringem apenas a aumentar os conhecimentos dos alunos na linguagem C, o que seria até questionável, mas, e talvez fundamentalmente, apresentar aos alunos de forma pragmática, os desafios que se colocam a quem concebe e programa aplicações software (em qualquer linguagem), quando passamos a realizar a designada programação em larga escala, ou seja, aplicações com grandes volumes de dados e com mais elevada complexidade algorítmica e estrutural.

De facto, quando passamos para tais patamares de complexidade, torna-se imperioso conhecer e usar os melhores princípios da Engenharia de Software, de modo a que tais projetos de software, em geral realizados por equipas, possam ser concebidos de forma estruturalmente correta, de modo a que sejam mais facilmente modificáveis, e sejam, apesar da complexidade, otimizáveis a todos os níveis.

Para que tal seja possível teremos que introduzir novos princípios de programação, mais adequados à programação em grande escala, designadamente:

- Modularidade e encapsulamento de dados usando construções da linguagem;
- Criação de código reutilizável;
- Escolha otimizada das estruturas de dados e reutilização;
- Testes de performance e até profiling.

Este projeto, a desenvolver em trabalho de grupo (de no máximo 3 alunos), visa aplicação destas práticas de desenvolvimento de software usando a linguagem C, práticas que são extensíveis a outras linguagens e paradigmas.

O desenvolvimento deste projeto deve ser feito colaborativamente com o auxílio de *Git* e *GitHub*. Os docentes serão membros integrantes da equipa de desenvolvimento de cada trabalho e irão acompanhar semanalmente a evolução dos projetos. A entrega do trabalho será efetuada através desta plataforma e os elementos do grupo serão avaliados individualmente de acordo com a sua contribuição para o repositório. Serão fornecidas

algumas regras que os grupos deverão seguir para manter e estruturar o repositório, de forma a que o processo de avaliação e de execução dos trabalhos possa ser uniforme entre os grupos e possa ser efetuada de forma automática.

Requisitos

2.1 Ficheiros de dados

O projeto a desenvolver tem como fonte de dados três ficheiros *.csv* disponibilizados no BB da disciplina: *users.csv*, *businesses.csv*, e *reviews.csv*. Cada linha nestes ficheiros representa um registo com a informação de um utilizador, business, e review, respectivamente. Cada registo é uma sequência de campos separados por ';'. O conteúdo de cada ficheiro é o seguinte (campos do tipo string, salvo indicação em contrário):

users.csv:

- user_id ;
- name;
- friends (ids de amigos separados por ',').

business.csv:

- business_id;
- name;
- city;
- state;
- categories (separadas por ',').

reviews.csv:

- review_id;
- user_id;
- business_id;
- stars (float);
- useful (int);
- funny (int);
- cool (int);
- date (YYYY-MM-DD HH:MM:SS);
- text.

2.2 Arquitetura da aplicação

No sentido de se deixar desde já uma orientação de base quanto à arquitetura final da aplicação a desenvolver, pretende-se de facto que a aplicação possua uma arquitetura na qual, tal como apresentado na figura seguinte, se identifiquem de forma clara as fontes de dados, a leitura das mesmas e os módulos de dados a construir (acompanhados pela respectiva API):

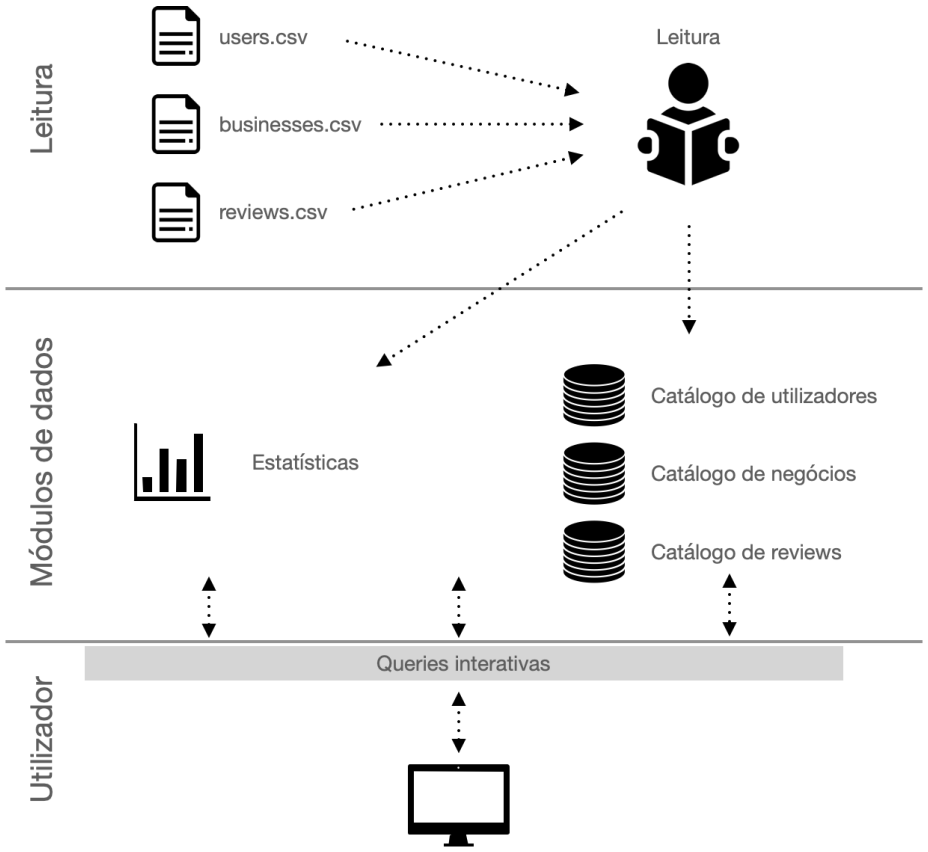
- Leitura: função ou parte do código no qual é realizada a leitura (e tratamento) dos dados dos 3 ficheiros;

- Catálogo de utilizadores: módulo de dados onde deverão ser guardados todos os utilizadores provenientes de registos válidos do ficheiro users.csv, organizados por identificador;

- Catálogo de Negócios: módulo de dados onde deverão ser guardados todos os negócios e respectiva informação contida no ficheiro business.csv;

- Catálogo de Reviews: módulo de dados onde armazena convenientemente as reviews recolhidas do ficheiro reviews.csv;

- Stats: módulo que efetua relações entre os 3 principais modelos, proporcionando acesso mais rápido a dados e resultados pedidos nas queries do programa.



Arquitetura de referência para a aplicação a desenvolver

Para a estruturação otimizada dos dados destes módulos de dados será crucial analisar as queries que a aplicação deverá implementar. É de realçar que não devem assumir que o programa irá trabalhar sempre sobre os mesmos ficheiros de dados. Concebendo os módulos de dados anteriormente referidos, obtemos a primeira arquitetura de base para o projeto, sendo desde já de salientar que esta arquitetura (ou algo equivalente) irá ser construída por fases, módulo a módulo, testando cada fase e cada módulo até se ter a garantia de que podemos juntar todas as unidades do projeto na aplicação final. A aplicação final terá ela própria uma estruturação particular (a apresentar e

a estudar), usando-se um padrão de software em que as três grandes componentes de uma aplicação serão codificadas de forma totalmente separada e independente:

- camada de dados e algoritmos, que designaremos por MODELO;
- camada visual de interação com o utilizador, cf. menus, listas de resultados, etc, que designaremos por APRESENTAÇÃO;
- camada responsável pela ordem de execução correta da aplicação em função dos desejos dos utilizadores, que designaremos por CONTROLO DE FLUXO ou apenas FLUXO ou CONTROLADOR;

2.3 Funcionalidades

O módulo que representa o sistema SGR (Sistema de Gestão de Recomendações) deverá obrigatoriamente suportar um conjunto de funcionalidades (queries), cuja assinatura é apresentada a seguir. O valor de retorno de cada query, à excepção da primeira, consiste num tipo de dados TABLE, cuja utilidade será melhor compreendida na secção seguinte. A API deste módulo que agrega todos os módulos de dados do sistema é a seguinte:

```
SGR init_sgr();

void free_sgr(SGR sgr);

/* query 1 */
SGR load_sgr(char *users, char *businesses, char *reviews);
/* query 2 */
TABLE businesses_started_by_letter(SGR sgr, char letter);
/* query 3 */
TABLE business_info(SGR sgr, char *business_id);
/* query 4 */
TABLE businesses_reviewed(SGR sgr, char *user_id);
/* query 5 */
TABLE businesses_with_stars_and_city(SGR sgr, float stars, char *city);
/* query 6 */
TABLE top_businesses_by_city(SGR sgr, int top);
/* query 7 */
TABLE international_users(SGR sgr);
/* query 8 */
TABLE top_businesses_with_category(SGR sgr, int top, char *category);
/* query 9 */
TABLE reviews_with_word(SGR sgr, int top, char *word);
```

Inicialmente, deverá definir um tipo de dados SGR, o qual deverá armazenar a informação pertinente do sistema, usando as devidas técnicas de modularidade e encapsulamento, i.e. deve declarar o tipo SGR de forma opaca no ficheiro `sgr.h` enquanto a sua implementação concreta é apenas declarada num ficheiro `sgr.c`. Para além das queries requeridas, a interface do sistema (`sgr.h`) deve também oferecer as funcionalidades para *inicializar* e *destruir* estruturas do tipo SGR. A descrição detalhada de cada uma das funcionalidades requeridas é a seguinte:

Query 1. Dado o caminho para os 3 ficheiros (Users, Businesses, e 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. Note-se que qualquer nova leitura destes ficheiros deverá de imediato reiniciar e refazer as estruturas de dados em memória como se de uma reinicialização se tratasse.

Query 2. Determinar a lista de nomes de negócios e o número total de negócios cujo nome inicia por uma dada letra. Note que a procura não deverá ser *case sensitive*.

Query 3. Dado um id de negócio, determinar a sua informação: nome, cidade, estado, stars, e número total reviews.

Query 4. 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.

Query 5. 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.

Query 6. 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.

Query 7. 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.

Query 8. 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.

Query 9. Dada uma palavra, determinar a lista de ids de reviews que a referem no campo *text*. Note que deverá ter em conta os vários possíveis símbolos de pontuação para delimitar cada palavra que aparece no texto. Nota: função `ispunct` da biblioteca `ctype.h`.

2.4 Interpretador de comandos

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. Mais concretamente, os resultados dessas operações devem poder ser visualizados, manipulados e atribuídos a variáveis, através de um conjunto mínimo de instruções básicas. De forma a tratar os resultados das queries de forma genérica, independentemente da query invocada, requer-se a criação do tipo de dados TABLE, para o qual todos os valores de retorno das queries devem ser mapeados (à excepção da query 1). Consequentemente, o único tipo de dados que o interpretador de comandos deve conhecer e manipular é o tipo TABLE.

De forma a suportar a interpretação de comandos, espera-se o desenvolvimento de um módulo INTERPRETADOR capaz de interpretar uma sequência de um ou mais comandos, separados por ";" e efetuar as operações correspondentes. Os comandos que o programa deve suportar são os seguintes (assumir x e y como variáveis do tipo Table):

- **Atribuição do valor de queries**: Guardar numa variável o valor do resultado da invocação de uma query.
  sintaxe: x = businesses_started_by_letter(sgr, “A”);

- ***Visualização* de variáveis:** Visualizar o valor de uma variável num formato visual estruturado, sob a forma de uma tabela.
  sintaxe: show(x);

- **Escrita para csv:** Enviar o conteúdo de uma variável para um ficheiro de formato csv.
  sintaxe: toCSV(x, delim, filepath);

- **Leitura de csv:** Atribuir a uma variável o conteúdo de um ficheiro csv.
  sintaxe: x = fromCSV(filepath, delim);

- **Filtragem de resultados:** Filtrar dados de uma tabela, dada uma coluna, um valor de comparação e um operador de comparação. Este operador será um tipo **Enum** definido como "enum OPERATOR{LT, EQ, GT}", correspondendo a maior, igual e menor, respectivamente. Poderão assumir que a variável *value* é do tipo *char*,* podendo o seu valor e a respetiva comparação a ser efetuada inferidos através do seu valor.
  sintaxe: y = filter(x, column_name, value, oper);

- **Projeção de colunas:** Obter um subconjunto de colunas de uma tabela.
  sintaxe: y = proj(x, cols);

- **Indexação**: Aceder a registos/valores contidos numa determinada posição da tabela.
  sintaxe: z = x[1][1];

- **Terminação:** Sair do programa.

sintaxe: quit;

Para além dos comandos apresentados, os grupos de trabalho são livres de implementar comandos adicionais, passíveis de receber valorização extra. Comandos sugeridos: count, join, max, min, avg.

Os comandos deverão assumir que existe já uma variável ***sgr*** inicializada e carregada no arranque do programa, podendo manipular essa variável na invocação das queries.

2.5 Paginação

Tal como referido anteriormente, este trabalho consiste em realizar programação em larga escala com grandes volumes de dados sem apoio de Bases de Dados. Desta forma, a execução de uma simples query poderá gerar uma lista com milhares de entradas, cuja visualização e consulta poderá ser difícil sem um suporte minimamente sofisticado. Deste modo, um dos requisitos deste projeto é conceção de mecanismos adequados de paginação dos resultados obtidos nas queries. Tal implicará a criação de um módulo adicional genérico, designado Paginação, que permite apresentar o tipo de dados TABLE de uma forma mais conveniente.

Este módulo deverá ser capaz de *apresentar* grandes volumes de dados (provenientes das queries) em páginas, oferecendo funcionalidades simples e intuitivas como avançar e recuar de página.

2.6 Testes de performance

Depois de desenvolver e codificar todo o seu projeto tendo por base os ficheiros inicialmente fornecidos, deverá realizar alguns testes de performance e apresentar os respetivos resultados. Pretende-se que os grupos efetuem comparações de performance entre diversos ficheiros de input com diferentes tamanhos e que serão posteriormente fornecidos. Todos os ficheiros serão fornecidos numa pasta disponibilizada via BB. Para a obtenção destes tempos pode ser usada a biblioteca *standard* de C time.h ou outra qualquer equivalente. Os tempos a registar devem ser homogéneos, isto é, ou registamos sempre os designados *elapsed times*.

2.7 Codificação final

A codificação final deste projeto deverá ser realizada usando a linguagem C e o compilador ***gcc***. O código fonte deverá compilar sem erros usando os switches “-Wall -std=c99”. Podem também ser utilizados switches de otimização.

Deverá existir uma Makefile genérica que permitirá gerar um executável “program" através do comando “make program". Este comando deverá compilar um ficheiro “main.c" que irá incluir o ficheiro “sgr.h". Respeitar esta nomenclatura é importante para garantir que o projeto é validado pelos scripts que irão executar nos repositórios Git para validar o trabalho após a entrega. Qualquer utilização de bibliotecas de estruturas de dados em C deverá ser sujeita a prévia validação por parte da equipa docente.

O código final será sujeito a uma análise para detectar similaridades entre o código dos diferentes grupos. Quando a percentagem de similaridade ultrapassar determinados níveis, os grupos serão chamados a uma clara justificação para tal facto.

2.8 Apresentação e relatório

O projeto será extraído automaticamente do repositório de cada grupo imediatamente após a data de entrega. Commits posteriores à data de entrega com alterações efetuadas no código do projeto não serão consideradas para fins de avaliação. A makefile deverá gerar o código executável, e este deverá executar corretamente. Projetos com erros de makefile, de compilação ou de execução serão de imediato rejeitados. Cada elemento do grupo será avaliado individualmente de acordo com a sua contribuição para o repositório do trabalho, tendo em conta o seu número de commits e respetivas alterações efetuadas no código do projeto.

Para além do código do projeto, cada grupo deverá elaborar um relatório (máx. 10 páginas) no deve ser descrita a implementação do projeto e todas as decisões de concepção do mesmo, designadamente:

- Descrição de cada módulo em termos de API (.h), respetivas estruturas de dados (apresentar um desenho destas) e justificação objetiva para a escolha das mesmas;
- Arquitetura final da aplicação e razões para tal modularidade;
- Complexidade das estruturas e otimizações realizadas;
- Otimizações e eventuais estratégias para otimizar queries;
- Resultados dos testes realizados.

O relatório será entregue aquando da apresentação do projeto e servirá de guião para a avaliação do mesmo. A avaliação do projeto implica a presença de todos os membros do grupo de trabalho, sob pena de reprovação imediata em caso de ausência injustificada. Para além da análise e avaliação da solução em termos estruturais e de eficácia de execução, e do relatório, alguma avaliação individual poderá ser realizada quando tal se justificar. Uma avaliação final inferior a 10 valores neste primeiro projeto implica a reprovação imediata à UC de LI3. Todas as questões adicionais deverão ser esclarecidas ao longo das aulas junto da equipa docente.

```
user@localhost> ./program
x = businesses_started_by_letter(sgr, 'T');
show(x);

_____________________________________________________
|      business_id      |           name            |
_____________________________________________________
| 0xsadjasdjsadjsajdsaj |     Tasco da Azeitona     |
_____________________________________________________
| 0xsodmsadmagasjjjjadd |      Tasco da Moela       |
_____________________________________________________

toCSV(x,';', "tascos.txt");
y = fromCSV("tascos.txt",';');
show(y);

_____________________________________________________
|      business_id      |           name            |
_____________________________________________________
| 0xsadjasdjsadjsajdsaj |     Tasco da Azeitona     |
_____________________________________________________
| 0xsodmsadmagasjjjjadd |      Tasco da Moela       |
_____________________________________________________

t = y[0][0];
show(t);

__________________________
|      business_id      |
__________________________
| 0xsadjasdjsadjsajdsaj |
__________________________
```

Exemplos de interação com o programa