Laboratórios de Informática III

Relatório do projeto

2020/2021

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

Índice

1. Introdução

2. LoadFiles

3. SGR

4. Queries

5. TABLE

6. Interpretador

7.Conclusão

Introdução

O projeto de C da disciplina de LI3 tem por objetivo fundamental ajudar à consolidação experimental dos conhecimentos teóricos e práticos adquiridos em UCs anteriores. O desafio deste projeto é processar grandes volumes de dados com o maior nível de otimização possível, no presente relatório explicamos as escolhas feitas e as estratégias usadas para atingir este objetivo.

Módulo: LoadFiles

Descrição

Este módulo tem como objetivo carregar para memória os ficheiros de input necessários para a realização do trabalho prático.

Arquitetura

Cada ficheiro é carregado individualmente, portanto, uma mudança na estrutura de um determinado ficheiro, ou até a introdução de um novo ficheiro com informação adicional, não implica uma restruturação total do módulo.

Ainda que frágil, fica garantida alguma modularidade. Isto porque, como seria de esperar, este módulo depende de uma estrutura bem definida dos ficheiros de input (.csv).

Estrutura

A estrutura & assinatura das funções que carregam ficheiros são apresentadas abaixo.

Função responsável por carregar o ficheiro *business.csv*.

```
27  /**
28   * @brief                Load business file (.csv).
29   *
30   * @param bus_file       Input File pointer.
31   * @param business_table HashTable where
32   *                          key   = bus_id
33   *                          value = struct_of_business_info
34   * @param business_names GTree where
35   *                          key   = first_letter_of_bus_name
36   *                          value = list_of_business_ids
37   * @param cities         GTree where
38   *                          key   = city
39   *                          value = list_of_business_ids
40   * @param categories     GTree where
41   *                          key   = category
42   *                          value = list_of_business_ids
43   * @return int           0 -> OK
44   *                       1 -> file_not_loaded.
45   */
46  int load_business_file( FILE       *bus_file,
47                          GHashTable *business_table,
48                          GTree      *business_names,
49                          GTree      *cities,
50                          GTree      *categories);
```

Função responsável por carregar o ficheiro users.*csv*.

```
53  /**
54   * @brief             Load users file (.csv).
55   *
56   * @param usr_file    Input File pointer.
57   * @param users_table HashTable where
58   *                      key   = usr_id
59   *                      value = struct_of_usr_info
60   * @return int        0 -> OK
61   *                    1 -> file_not_loaded.
62   */
63  int load_users_file(FILE *usr_file, GHashTable *users_table);
```

Função responsável por carregar o ficheiro reviews.*csv*.

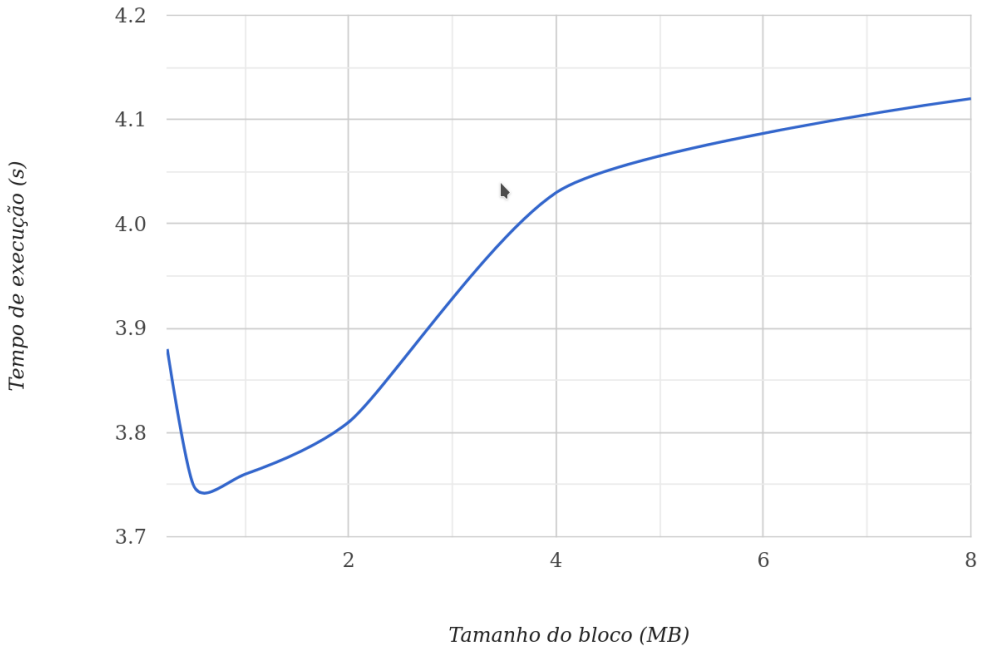
```
65  /**
66   * @brief              Load users file (.csv).
67   *
68   * @param rev_file     Input File pointer.
69   * @param users_table  HashTable where
70   *                        key   = usr_id
71   *                        value = struct_of_usr_info
72   * @param business_table HashTable where
73   *                        key   = bus_id
74   *                        value = struct_of_business_info
75   * @return int         0 -> OK
76   *                     1 -> file_not_loaded.
77   */
78  int load_reviews_file(FILE* rev_file, GHashTable* users_table, GHashTable* business_table);
```

Otimizações

Leitura por blocos: Os ficheiros de input (.csv) estão a ser *lidos por blocos*. A escolha para o tamanho do bloco derivou de um processo de testes apresentado abaixo.[1]

Atenção: Os tempos indicados incluem também o processo de libertar a memória alocada pelas estruturas.[2]

Tempo de execução do módulo LoadFiles em função do tamanho dos blocos



O tamanho de cada bloco, utilizado neste projeto, é de 0.5MB (ou 512kB).

[1] Todos os testes presentes neste relatório utilizam os *switches "-O2 -Wall –std=c99 $(pkg-config --cflags --libs glib-2.0)".*

[2] Os tempos de execução são obtidos utilizando o comando *"perf stat -d -r 10 ./prog_test"*

Módulo SGR

Descrição

Este módulo tem as funções relativas à alocação de memória para as estruturas do SGR, assim como as funções que libertam esta memória.

Arquitetura

A estrutura SGR encontra-se definida na imagem abaixo.

```
3   struct sgr_struct{
4       char *reviews_filepath;     // Indicates reviews file (.csv) path (used in query 9).
5       GHashTable* businesses;     // key=bus_id,   value=name,city,state,stars,#reviews
6       GTree* bus_names;           // key=letter,   value=list_of_bus_id
7       GTree* categories;          // key=category, value=list_of_bus_id
8       GTree* cities;              // key=city,     value=list_of_bus_id
9       GHashTable* users;          // key=usr_id,   value=usr_name,list_of_reviews_id
10      gboolean dictionary_loaded; // Is the dictionary loaded into memory?
11      GHashTable *dictionary;     // Dictionary structure => key=word, value=list_of_reviews_id
12  };
```

A função que carrega a estrutura SGR (query 1 *aka* load_sgr()) é contida por

1. load_business_file()
2. load_users_file()
3. load_reviews_file()
4. sortTree(cities): ordena a GTree de cidades de acordo com o rating dos negócios.

Estrutura e Otimizações

A estrutura do SGR foi escolhida tendo em conta as necessidades de cada query.

No que diz respeito aos **dados extraídos do ficheiro *business.csv*** foram criadas **4 estruturas**.

1. **GHashTable** *businesses: Esta estrutura garante **inserção/procura** em ***O(1)***. Esta estrutura **resolve a query 3** (business_info) em ***O(1)***.
2. **GTree** *bus_names: Esta estrutura (árvore binária balanceada – altura logarítmica) armazena todos os **id's dos negócios organizados** pela **primeira letra do nome**. Deste modo, a **query 2** (business_started_by_letter) é resolvida em **O(log N^3)**. Devido ao limitado número de letras, a altura da árvore tem um máximo de 12 ($N \leq 12$).
3. **GTree** *categories: Esta árvore binária balanceada (com altura logarítmica) **armazena** as diferentes **categorias** que existem **assim como a GList de *business id's* associada**. Deste modo a **query top_business_with_category** (**8**), com a ajuda da *hash table* (1.), é resolvida em ***O(log N)*** sendo que apenas é necessária uma procura nesta árvore para encontrar uma lista ordenada (por ratings) de business id's.
4. **GTree** *cities: Esta árvore binária balanceada tem o mesmo formato que a anterior, mas **armazena cidades**. Deste modo, a query **business_with_stars_and_city** (5) é

[3] Representa a altura da árvore.

resolvida da mesma forma que a query 8 (apresentada anteriormente). Esta estrutura também resolve a **query top_business_by_city** (6), sendo que se trata da execução da query 5 para cada cidade, ou seja, a execução da query 5 *foreach node* da árvore de cidades. Como será de esperar temos: ***O(C * log N)***.[4]

No que diz respeito ao ficheiro ***reviews.csv***, mantemos uma estrutura ***dictionary***.

1. **GHashTable** *dictionary: Armazena o texto das reviews (palavra a palavra) efetuadas pelos utilizadores. As **inserções/procuras** são feitas em ***O(1)***.

No caso da outra *hash table*

1. **GHashTable** *users: Guardar a lista de businesses que receberam uma review de um determinado user, sendo assim a key da hashtable é o id do utilizador e o conteúdo é uma Glist com todos os ids dos businesses que eles fizerem review.

[4] 'C' representa o número de cidades / nodos da árvore.

Módulo Queries

Descrição

Este módulo é constituído pelas funções que definem as queries do projeto.

Arquitetura

As ideias gerais para a resolução de cada query encontram-se abaixo descritas.

1. *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.*

Solução: Leitura de ficheiros tal como descrita no módulo **LoadFiles**.

2. *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.*

   Solução: **Procura na *GTree bus_names***. A procura pela primeira letra devolve uma GList de id's de negócios. Podemos procurar cada um destes id's na *hash table* de businesses para obter o nome de cada negócio.

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

   Solução: **Procura na *hash table*** onde **key=bus_id**, **value=business_info**.

4. *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.*

   Solução: Usando a hashtable dos utilizadores que criamos no load sgr, facilmente obtemos a lista de ids dos negócios ao quais o utilizador fez as reviews. Depois usando a lista de ids e utilizando a hash table dos negócios, obtemos facilmente uma lista com os nomes dos negócios, tendo as duas listas basta apenas inseri-las na table.

5. *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.*

   Solução: **Procura na GTree de cidades**. Depois de obter a lista de business id's

presentes na cidade, **converter cada id no par (id, nome)** pedido pela query. Esta conversão é feita utilizando a *hash table* businesses.

6. *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.*

   Solução: Fazer uma **travessia pela *GTree de cidades***. **Para cada nodo** devemos **converter listas de id's** de negócios **em (id, nome, stars)** com ajuda a *hash table* businesses (fazendo uma procura por id).

7. *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.*

   Solução: Iterar sobre a hastable dos utilizadores e verificar se ele é um utilizador internacional percorrendo a lista de negocios a qual este utilizador fez review , presente na hash table dos utilizadores, e usando a hash table dos negócios verificar se esta lista contém mais que um estado diferente. Caso tenha podemos acrescentar este utilizador a lista, caso contrário não.

8. *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.*

   Solução*:* A metedologia é semelhante à apresentada até agora. Fazemos uma **procura na GTree** de categorias **para obter a GList de business id's** que constituem a categoria. Depois **convertemos**, com a ajuda da informação presente na *hash table*, cada id, na informação necessária para resolver a query: **id, nome e stars**.

9. *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.*

   Solução: **Cada palavra é processada e armazenada numa *hash table*.** Esta estrutura tem a forma **key=palavra**, **value=GList de id's de reviews onde a palavra surge**. Não são processadas palavras repetidas dentro da mesma review.

Otimizações

Neste módulo, o desempenho de cada query depende, *a priori*, das estruturas escolhidas para constituir o SGR.

Testes

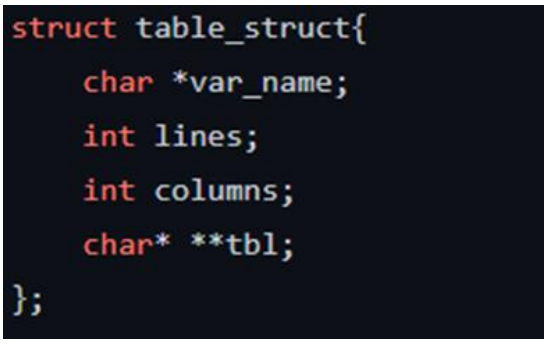
A versão atual do projeto está a executar o **load_sgr**, e respetivo **free_sgr**, **em conjunto com todas as queries** (2-8, **exceto reviews_with_word**) em *aproximadamente **4,93s***.

A **query 9** (**reviews_with_word**) **implica carregar para memória** a ***GHashTable dictionary*** que, em média, demora cerca de **67,97s**. **Após carregar o dicionário**, a query 9 é executada em ***O(1)***, visto que apenas é necessário fazer uma procura na hash table.

Módulo TABLE

Arquitetura

A estrutura **table** encontra-se definida na imagem abaixo.



Estrutura e Otimizações

1- Char ***var_name** - string para o nome da variável que lhe vamos atribuir para ser fácil identificar e voltar a acessar a mesma para fazer show, filter ou outras funções que for necessário.
2- int **lines/columns** - Colocamos a informação de quantas linhas e colunas a matriz tbl tem para termos um maior controlo.
3- Char ***** **tbl** - uma matriz de strings que contem toda a informação de uma respetiva query, optamos por colocar na linha 0 os headers para não haver necessidade de criar mais um array na estrutura.

Módulo Interpretador

Descrição

Este é o modulo principal e responsável pelo fluxo do programa, ou seja, é o que analisa o formato do comando efetuado por o utilizador e envia para as respetivas funções. Para além dos comandos exigidos pelo guião, tomamos a liberdade de acrescentar mais dois para melhor a experiência do usuário. Os novos comandos são “clear” que simplesmente limpa o terminal e “help” que imprime no ecrã todos os possíveis comandos, as queries com os seus formatos e também, caso exista, o nome das **TABLE** existentes no programa.

Arquitetura

A parte mais importante do interpretador é o comando que associa uma variável a uma **query**, após este comando ser executado, é enviado a parte da **query** para uma função que por sua vez separa o nome da **query** e as suas variáveis, analisa se o nome da **query** existe nas **queries** possíveis, se não existir, é imprimido uma mensagem a notificar o usuário que a sua TABLE não foi criada, caso a função reconheça a **query** pedida, passa para a validação de variáveis e após aprovação e executado a respetiva **query** que por sua vez é adicionada a uma **GLIST** que o interpretador tem acesso onde guarda todas as TABLES executadas no programa.

Otimizações

Optamos por colocar as variáveis numa GLIST de forma a otimizar o acesso as **TABLE**, também optamos por fazer **prepend** em vez de **append** para otimizar a colocação de **TABLE**.

Conclusão

Refletindo sobre o trabalho feito, achamos que atingimos o nosso objetivo quanto se trata a qualidade do trabalho. Agora terminado o projeto conseguimos perceber melhor as dificuldades e respetivas soluções em relação ao processamento de grandes volumes de dados e saímos daqui com um melhor conhecimento de como otimizar e organizar estes processos.

Made by:

Jorge Manuel Costa Lima – a93183

João André Monteiro Martins – a91669

Ruben Samuel Alves Santos – a93257