Relatório - Laboratórios de Informática III - Grupo 20

Estrutura modular do projeto

A leitura e o tratamento dos dados dos 3 ficheiros (users.csv, business.csv e reviews.csv) foi tratada no módulo designado por *interface*. Este é constituído por 3 funções, cada uma responsável pela leitura de um dos ficheiros, validando as estruturas *businesses*, *review* e *users*. A divisão dos três catálogos referidos foi feita por 6 módulos.

Primeiramente, temos 2 módulos destinados ao catálogo de negócios: *business* e *businesses*. O primeiro contém as funções relativas aos *getters*, *setters* e libertação de memória da estrutura *BUSINESS*, permitindo manipular cada negócio. O segundo contém funções que manipulam o catálogo de negócios em si, podendo adicionar negócios novos, procurar por algum negócio em específico, entre outras. Toda esta gestão é feita através de uma estrutura que contém uma hashtable:

keys: **business_idA**	**business_idB**	**business_idC**
char * business_idA char * business_nameA char * cityA char * stateA char * categoriesA	char * business_idB char * business_nameB char * cityB char * stateB char * categoriesB	char * business_idC char * business_nameC char * cityC char * stateC char * categoriesC

Por outro lado, responsáveis pelo catálogo de reviews temos outros 2 módulos: *review* e *reviews*. Analogamente, o primeiro diz respeito aos *getters*, *setters* e libertação de memória da estrutura *REVIEW*, bem como a validação de datas de reviews e o segundo diz respeito à manipulação do catálogo em si, uma vez que nele estão contidas funções que validam reviews, adicionam reviews ao catálogo, entre outras.

keys: **review_idA**	**review_idB**	**review_idC**
char * review_idA char * user_id char * business_id float stars int useful int funny int cool char * date char * text	char * review_idB char * user_id char * business_id float stars int useful int funny int cool char * date char * text	char * review_idC char * user_id char * business_id float stars int useful int funny int cool char * date char * text

No que diz respeito ao catálogo de utilizadores, temos 2 módulos: *user* e *users*. O primeiro responsabiliza-se pelos *getters* e *setters*, libertação de memória da estrutura *USER* e a validação de um utilizador. O segundo manipula o catálogo de utilizadores em si, ao validar utilizadores, adicionar utilizadores ao catálogo, entre outros.

keys: **user_idA**	**user_idB**	**user_idC**
char * user_idA char * nameA	char * user_idB char * nameB	char * user_idC char * nameC

Dentro dos três catálogos previamente referidos temos ainda auxiliares para as queries, que foram criadas com o objetivo de se poder ter acesso direto à hashtable a percorrer, por exemplo a dos negócios ou das reviews. Gerar um clone de uma hashtable daquelas dimensões, para usar no módulo sgr, pareceu um uso excessivo de memória. Além disso, também prejudicaria a sua velocidade.

Para além disso, foi elaborado um módulo *stats* que contém duas estruturas auxiliares para as queries, funções para as inicializar, libertar memória, getter, setter, entre outros. Enquanto a estrutura *BUS_STATS* associa um id de negócio ao número de reviews que recebeu e ao número total de estrelas que lhe foram atribuídas, a estrutura *USER_VISITS* associa o id do utilizador a um número mínimo de cidades que visitou e a uma cidade específica que tenha visitado.

char * business_id; int n_reviews; float total_stars;		char * user_id; int n_cities; char * city_visited
BUS_STATS		**USER_VISITS**

O módulo *sgr* representa o Sistema de Gestão de Recomendações e implementa 9 funcionalidades (*queries*), com suporte de uma estrutura *SGR* que contém as 3 estruturas do catálogo.

Vale a pena notar que, para a *query* 1, aproveitamos as funções de loading do módulo *interface*, de forma a carregar a estrutura *SGR*.

Na *query* 2, uma das funções auxiliares que foi utilizada foi definida no módulo *businesses* e é responsável por recolher os negócios iniciados pela letra pretendida, carregando-os para um array que é depois atribuído a uma estrutura TABLE (que será posteriormente explicada com mais detalhe) de forma a posteriormente ser possível de imprimir o seu conteúdo.

Por outro lado, na *query* 3, depois de ser selecionado o negócio correspondente, recorre-se a uma auxiliar capaz de determinar o número de estrelas e de efetuar a contagem de *reviews.* Assim, é possível no final serem adicionados à TABLE o nome, a cidade, o estado, a média de estrelas e o número de *reviews*.

Quanto à *query* 4, recorre-se a uma função auxiliar do módulo *reviews*, responsável por determinar os negócios avaliados pelo utilizador em questão, depois de inicializar a *table*. Aproveitamos também para guardar a palavra mais longa de forma a facilitar a visualização da TABLE posteriormente.

Já na *query* 5, foi utilizada mais do que uma função auxiliar, todas de diferentes módulos. Primeiro recolhem-se os negócios que existem na cidade pretendida para uma hashtable e atualiza-se a estrutura com o número de reviews e o número total de estrelas de cada um. De seguida, percorre-se essa hashtable para adicionar à TABLE a informação dos negócios que cumprem o requisito de média de estrelas.

Relativamente à *query* 6, primeiro recolhe-se para uma hashtable todos os negócios que possuem *reviews*, associando-os ao seu número total de reviews e de estrelas, através de uma das estruturas auxiliares do módulo *STATS*. De seguida, percorre-se a hashtable previamente referida para organizar todos estes negócios numa nova hashtable que permite associar a uma cidade uma array de negócios situados nessa mesma cidade. Por fim, percorre-se esta última para organizar cada array de negócios e selecionar, no máximo, o número de negócios pretendidos.

Para a *query* 7, aplicamos a função auxiliar do módulo *reviews* para obter a lista de ids de utilizadores e o número total de utilizadores. Depois de inicializar a *table*, guarda-se a lista de ids de utilizadores internacionais e adiciona-se à *table* o número de internacionais.

Quanto à *query* 8, recolhem-se para uma hashtable os negócios da categoria pretendida através de uma auxiliar localizada no módulo *businesses* e atualiza-se com a informação relativa ao número de estrelas e de reviews. Cada estrutura inserida na hashtable, é copiada para um array que é depois ordenado. Por fim, seleciona-se o número pretendido de negócios.

Em relação à *query* 9, aplicamos uma função auxiliar responsável por obter os reviews com uma dada palavra, do módulo *reviews*. Esta auxiliar tem o cuidado de verificar a pontuação e vários casos de exceção. Depois de inicializar a *table*, guardam-se as avaliações que começam pela letra desejada.

De forma a controlar o que é imprimido na consola, temos o módulo *view*. Neste módulo temos a estrutura *TABLE* que é, como foi visto, o retorno de quase todas as *queries*. Nele estão definidas funções que manipulam a estrutura, como por exemplo adicionar um novo elemento, ou até determinar o tamanho da tabela. E, para além das funções que imprimem as *queries* e dos *setters* e *getters,* aplicamos funções para imprimir o menu, mensagens de sucesso ou erro e cabeçalho do programa, bem como funções para formatar a estética da consola apresentada ao utilizador.

Este projeto foi ainda acompanhado de um módulo de macros que permite a reutilização de determinados valores ao longo do programa.

Por fim, a gestão da receção input do utilizador durante a execução do programa foi realizada no módulo *controller*. Este permite carregar o conteúdo das queries para as variáveis pretendidas pelo utilizador, imprimir este conteúdo, efetuar indexação e converter para .csv. Para a gestão das variáveis, foi utilizada uma hashtable com uma struct variable auxiliar. Por questões de tempo, não foi possível implementar um comando readCSV, proj e filter.

Como foi visto, houve uma preferência pelo uso de hashtables e arrays ao longo do projeto, por ter sido considerado que seriam estruturas mais eficientes para manipular a quantidade de dados. Para além disso, era pretendido algo que possibilitasse uma procura o mais rápida possível, e nesse aspeto teve-se em conta a menor complexidade temporal das hashtables para este fim.

Arquitetura final da aplicação

MODELO	APRESENTAÇÃO	FLUXO
• interface • business • businesses • review • reviews • user • users • stats • sgr • macro	• view	• controller

Complexidades das estruturas e otimizações efetuadas

Relativamente a complexidade das estruturas, tanto nos GPtrArrays como nas GHashTables, a procura e a inserção correspondem a tempo constante. No entanto, a função do GPTrArray correspondeu a tempo quadrático.

Quanto a otimizações efetuadas, como foi dito previamente, foi escolhido criar auxiliares nos módulos que continham as *hashtables* principais de modo a não ser obrigatório clonar estruturas de dados de tais dimensões, o que se trataria de um gasto enorme de memória e prejudicaria a velocidade do programa. Para além disso, foi relevante aproveitar código útil a várias queries em auxiliares. Por fim, as estruturas auxiliares criadas no módulo stats foram também fulcrais no que toca à organização da informação necessária para obter os resultados pretendidos.

Testes realizados

Ao longo do projeto foi usado o valgrind para manter as memory leaks do programa o mais baixas possíveis. Desta forma, este foi terminado com apenas a memória residual não libertada pela glib que não foi possível eliminar manualmente. Comprova-se com a imagem abaixo, correspondente à execução do programa com chamada de algumas queries. No entanto, o projeto foi testado para todos os comandos, funções e queries do programa, não se tendo encontrado qualquer problema.

```
==7631== HEAP SUMMARY:
==7631==     in use at exit: 18,604 bytes in 6 blocks
==7631==   total heap usage: 31,926,415 allocs, 31,926,409 frees, 1,522,665,022 bytes allocated
==7631==
==7631== LEAK SUMMARY:
==7631==    definitely lost: 0 bytes in 0 blocks
==7631==    indirectly lost: 0 bytes in 0 blocks
==7631==      possibly lost: 0 bytes in 0 blocks
==7631==    still reachable: 18,604 bytes in 6 blocks
==7631==         suppressed: 0 bytes in 0 blocks
==7631== Reachable blocks (those to which a pointer was found) are not shown.
==7631== To see them, rerun with: --leak-check=full --show-leak-kinds=all
```

Relativamente a tempos de execução, a tabela seguinte representa os resultados obtidos. Foi usado o método seguinte para medir os tempos:

- clock_t time = clock() , no ínicio da função
- time = clock() - time , no fim da função
- float result = (float) ((((double) time) / CLOCKS_PER_SEC) * 1000)

Os testes seguintes foram realizados num computador com estas características e com otimização O2:

- Memória 15,5 GiB
- Processador Intel® Core™ i7-10750H CPU @ 2.60GHz × 12
- Capacidade de Disco 512,1 GB
- Placa Gráfica NVIDIA GeForce GTX 1650

Função	Tempos (ms)
Leitura dos ficheiros	2595.594971
Query 2 (testada com a letra ‘A)	39.571999
Query 3	167.759003
Query 4	186.770996
Query 5 (testada com 3.5 e Austin)	355.351013
Query 6 (testada para top 5)	247.761993
Query 7	680.213013
Query 8 (testada para top 20 e food)	311.427002
Query 9 (testada com a palavra “and”)	366.838013