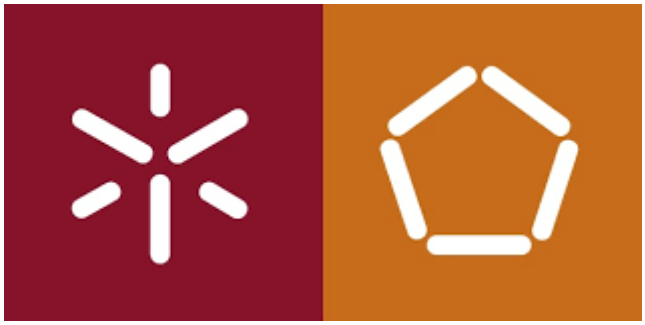

Engenharia Informática

Laboratórios de Informática III

Grupo 98

Ema Maria Monteiro Martins A97678

Henrique Nuno Marinho Malheiro A97455

Índice

Introdução

No âmbito da UC de Laboratórios de Informática III, foi-nos proposto a realização de um programa capaz de ler, armazenar e manipular 3 csv(s) (“users.csv”,”drivers.csv” e “rides.csv”).

Primeiro tivemos que fazer a leitura das várias linhas dos csv(s), armazenando-as de maneira a um fácil acesso sempre que necessário.

Tivemos sempre em conta a melhor maneira de manipular os dados para que o tempo de execução das queries fosse o menor possível, tentando maximizar o encapsulamento do programa, numa tentativa de realizar o maior trabalho possível nos builds.

Adotamos diferentes estratégias de parse e de armazenamento de dados, com vista também a reduzir a memória utilizada.

Modelo

Depois de analisarmos a estrutura e finalidade das queries decidimos construir 5 Coleções, “UsersCollection”, “DriversCollection”, “RidesCollection”, “StatisticsCollection” e a “Q7Collection”, onde as 3 primeiras guardam a informação dos seus respetivos csv(s) (em adição as estruturas que serão utilizadas nas queries que somente dependem desse csv), a “StatisticsCollection” guarda os valores calculados em diversas estruturas que serão utilizados nas queries e a Q7Collection somente é utilizada na chamada da “Q7”.

MakeFile

Cria o executável “programa-principal”, através dos “.o” dos diversos módulos, incluindo as diversas flags (como por exemplo as da glib). Cria ainda a pasta resultados onde serão armazenados os outputs das queries e implementa o make clean que limpa essa mesma pasta, o executável e os “.o”.

Modo Batch

Batch

Este modo funciona através da chamada do executável criado no MakeFile com mais dois argumentos (diretoria da pasta que contem os csv(s) e a diretoria para o ficheiro com os inputs para as diversas queries).

Modo Interativo

Interativo

Este modo funciona através da chamada do executável criado no MakeFile sem mais argumentos, sendo eles passados pelo utilizador durante a execução do programa.

Módulos

Main

Módulo que contem a função main do programa, onde se estabelece se este está a correr em modo Batch ou Interativo, dependendo do número de argumentos passados ao correr o executável.

Parse

Neste modulo, temos a função de parse que recebendo uma linha e um delimitador, faz o parse dessa linha em diversos tokens sempre que esse respetivo delimitador aparece na linha.

Valid

Neste módulo temos as funções que permitem verificar se uma determinada linha é ou não válida, testando os seus diversos parâmetros com as funções “date_valid”, “toLower_string”, “is_decimal_string” e “is_int_string”, que verifica se uma data é valida, poem uma string em minúsculas, se um numero é decimal ou inteiro, respetivamente.

Users

A função ”buildUsers”, recebe a diretoria para o “users.csv” e percorre o mesmo linha a linha. Durante a chamada das diversas linhas, realiza o parse de cada uma deles recorrendo à função de parse implementada no parse.c.

De maneira a guardar os diversos valores resultantes do array de tokens da função de parse, aloca espaço para a sua coleção que irá guardar um array de User(s) structs, onde cada User tem a informação de cada linha.

Liberta-se o array de tokens a cada iteração do ciclo e guarda-se o número de linhas válidas na coleção (poderá ou não diferir do número de linhas do csv dependendo se o mesmo tem ou não linhas inválidas que são testadas linha a linha).

O módulo tem também outras duas funções, a “create_UsersHash” que percorre o array de structs e cria uma GHashTable, do módulo <glib.h>, onde as chaves são os usernames e os valores ponteiros para o respetivo User e a função “sortUsersArray” que aloca um array de Q3 structs, que irá guardar os User(s) que cumpram os requisitos da query “Q3”, guardando-o já respetivamente ordenado pelos critérios estabelecidos nessa mesma query (usando as funções auxiliares do helpers.c).

Neste módulo, também temos os diversos “gets” que pesquisam através do username na “UsersHash” o parâmetro que procuram.

Drivers

A função ”buildDrivers”, recebe a diretoria para o “drivers.csv” e percorre o mesmo linha a linha. Durante a chamada das diversas linhas, realiza o parse de cada uma deles recorrendo à função de parse implementada no parse.c.

De maneira a guardar os diversos valores resultantes do array de tokens da função de parse, aloca espaço para a sua coleção que irá guardar um array de Driver(s) structs, onde cada Driver tem a informação de cada linha.

Liberta-se o array de tokens a cada iteração do ciclo e guarda-se o número de linhas válidas na coleção (poderá ou não diferir do número de linhas do csv dependendo se o mesmo tem ou não linhas inválidas que são testadas linha a linha).

O módulo tem também outras duas funções, a “create_DriversHash” que percorre o array de structs e cria uma GHashTable, do módulo <glib.h>, onde as chaves são os ids e os valores ponteiros para o respetivo Driver e a função “sortDriversArray” que aloca um array de Q2 structs, que irá guardar os Driver(s) que cumpram os requisitos da query “Q2”, guardando-o já respetivamente ordenado pelos critérios estabelecidos nessa mesma query (usando as funções auxiliares do helpers.c).

Temos também a função “create_Q7_collection”, que guarda numa coleção uma Hash de Hash(s) onde a primeira tem como chaves a cidade onde ocorrem as viagens e como valores ponteiros para “drivers_HashQ7”, onde as mesmas têm como chaves os id(s) dos drivers e como valores ponteiros para as structs “drivers_Q7” que guardam os dados necessários na Q7.

Neste módulo, também temos os diversos “gets” que pesquisam através do id na “DriversHash” o parâmetro que procuram.

Rides

A função “buildRides”, recebe a diretoria para o “rides.csv” e percorre o mesmo linha a linha. Durante a chamada das diversas linhas, realiza o parse de cada uma deles recorrendo à função de parse implementada no parse.c.

De maneira a guardar os diversos valores resultantes do array de tokens da função de parse, aloca espaço para a sua coleção que irá guardar um array de Ride(s) structs, onde cada Ride tem a informação de cada linha.

Liberta-se o array de tokens a cada iteração do ciclo e guarda-se o número de linhas válidas na coleção (poderá ou não diferir do número de linhas do csv dependendo se o mesmo tem ou não linhas inválidas que são testadas linha a linha).

O módulo tem também outra função, a “create_RidesHash” que percorre o array de structs e cria uma GHashTable, do módulo <glib.h>, onde as chaves são os ids e os valores ponteiros para o respetivo Driver.

Neste módulo, também temos os diversos “gets” que pesquisam através do id na “RidesHash” o parâmetro que procuram.

Statistics

Neste módulo, decidimos criar 3 estruturas “statisticsDrivers”,”statisticsUsers” e “statisticsCity”.

Na função “create_StatisticsCollection”, alocamos espaço para a “StatisticsCollection” que irá guardar os diversos arrays para as diversas queries (já respetivamente ordenados conforme os requisitos de cada query) e a StatisticHash.

A StatisticsHash que tem como chaves os vários “usernames”, “driversID” e “city” e como valores o ponteiro da struct desse User, Driver ou City. Começamos por verificar se a chave já está, ou não, na hash. Se não estiver, inicializamos as suas estatísticas em estado “neutro” para depois serem calculados os seus valores quando essa chave for encontrada na hash.

Neste módulo, também temos os diversos “gets” que pesquisam através das chaves (username, driverID ou city) na StatisticHash o parâmetro que procuram.

Helpers

Aqui, temos todas as funções auxiliares aos restantes módulos, desde a “idade” e “data_number” que calcula a idade (tendo em conta a data atual definida como 9/10/2022) e passa uma data para um inteiro representativo dessa data, respetivamente.

Temos também, uma função que cria um documento ,“cria_documento”, e as diversas funções auxiliares do qsort e a number_lines que define um limite de linhas a ser imprimido no modo interativo.

Query_selector

Neste módulo temos duas funções, a “query_selector_batch” que lê o ficheiro de inputs linha a linha, e faz o seu parse com a função do módulo Parse, chamando a respetiva query mediante o argumento passado, e a “query_selector_interativo” que interagindo com o utilizador vai guardando os argumentos e realizando as respetivas queries.

Queries

Em todas as funções das queries, faz-se uma diferenciação entre a sua utilização em modo batch, n diferente de -1, e para o modo interativo, onde os resultados são imprimidos num ficheiro “.txt” ou no terminal, respetivamente.

Caso não haja algum resultado, as funções imprimem uma linha vazia.

Q1

Começa por verificar se o id/username passado existem nas HashTables e, caso existam, identifica se é um user ou um driver. Escreve os diversos nomes, géneros, idades, scores, número de viagens e total auferido/gasto associados ao id/username.

Q2

Percorre o array “sorted_Drivers_array” até à posição do top pretendida, imprimindo os top <N> drivers com melhor média de avaliações.

Q3

Percorre o array “sorted_Users_array” até à posição do top pretendida, imprimindo os top <N> users com a maior distância viajada.

Q4

Verifica se uma dada cidade se encontra na StatisticsHash. Em caso afirmativo, escreve o preço médio das viagens nessa cidade.

Q5

Percorre o array “sorted_Q5_6_array” enquanto as datas forem menores do que a segunda passada como argumento, somando ao valor total o seu custo, caso a data seja também maior do que a primeira (estando assim compreendido entre as duas).

Q6

Percorre o array “sorted_Q5_6_array” enquanto as datas forem menores do que a segunda passada como argumento, somando ao valor total o seu custo, caso a data seja também maior do que a primeira (estando assim compreendido entre as duas) e a viajem ocorra na cidade passada como argumento.

Q7

Faz o lookup na “citysHash_Q7” da cidade passada como argumento, percorrendo a hash de drivers resultante e guardando os diversos drivers num array que irá ser posteriormente ordenado com o qsort.

Depois percorre esse array nas primeiras <N> posições, imprimindo os respetivos drivers.

Q8

Percorre o array “sorted_Q8_array”, que conterá só as viagens onde condutor e utilizador são do sexo feminino ou masculino, dependendo do sexo passado como argumento.

Se a menor das idades entre o condutor e utilizador for maior do que a passado como argumento, os seus id(s) e nomes serão imprimidos.

Q9

Percorre o array “sorted_Q9_array”, e se a data da viagem estiver compreendida entre as duas datas passada como argumento imprime o id da viagem, a sua data, distância percorrida, cidade e gorjeta.

Free

Contém as funções que libertam toda a memória alocada dinamicamente ao longo do programa, libertando todas

Considerações

De maneira a que a execução das queries fosse a mais rápida possível, adotamos estruturas particulares para cada query, como array(s) previamente ordenados, passando unicamente os parâmetros necessários para a ordenação dos mesmos, e HashTables.

Inicialmente, fazíamos o parse de cada csv de maneira isolada, não sendo essa a melhor estratégia a adotar. Decidimos depois realizar um parse genérico que copiava os diversos tokens dos csv(s) para uma "matriz", que apesar de ser rápida ocupava muita memória (cerca de 11 Gb para o total do programa com o data-large). Logo, optamos por fazer o parse linha a linha copiando os tokens da linha diretamente para a struct, libertando o array de tokens retornado pela função de parse.

Contudo, só essa mudança não permitia que o trabalho cumprisse os requisitos de memória ocupada, logo decidimos guardar os dados que fossem possíveis como inteiros ou dobles, ocupando menos memória do que o seu valor em string e reorganiza-mos os parâmetros das structs por ordem crescente de memória.

Liberta-mos toda a memória alocada dinamicamente, de maneira a prevenir memory leaks, não sendo possível libertar algumas leaks provenientes da utilização de funções da biblioteca <glib.h>.

Quanto à modularidade e encapsulamento do código, optamos por utilizar "gets" nas diversas HashTables, que copiavam um determinado parâmetro, sendo necessário libertar o mesmo quando não fosse mais necessário.

Tabelas

As medições apresentadas foram obtidas através da execução do programa numa máquina de modelo HP HP ENVY x360 Convertible, com 16,0 GiB de RAM, um processador AMD® Ryzen 7 4700u with radeon graphics × 8, no sistema operativo Ubuntu 22.04.1 LTS de 64-bit.

Tabelas de Tempos de execução

Tempo Parse/Build (s)	Data-regular	Data-regular-errors	Data-Large	Data-Large-errors
1ªexec	8.643881	8.343409	83.562613	84.503309
2ªexec	8.681796	8.269391	83.753854	84.579750
3ªexec	8.632836	8.289104	84.252030	84.068876
4ªexec	8.663193	8.277806	84.557430	83.717042
5ªexec	8.707203	8.228002	84.985015	83.784436
6ªexec	8.634091	8.189161	85.892374	83.102947
7ªexec	8.656226	8.260203	83.709713	83.530944
8ªexec	8.660954	8.214028	84.396919	82.868619
9ªexec	8.766824	8.321876	83.691723	83.101212
10ªexec	8.725937	8.255963	84.396919	83.999026
Média	8.677294	8.266894	84.319849	83.728278

Tempo Queries(s)	Q1	Q2	Q3	Q4
1ªexec	0.000045	0.001622	0.003676	0.000009
2ªexec	0.000047	0.001501	0.002778	0.000009
3ªexec	0.000016	0.000344	0.000009	0.000048
4ªexec	0.000018	0.000036	0.003817	0.000023
5ªexec	0.000010	0.000925	0.002798	0.000027
6ªexec	0.000017	0.001552	0.003817	0.000024
7ªexec	0.000106	0.000925	0.003651	0.000021
8ªexec	0.000017	0.000023	0.002822	0.000025
9ªexec	0.000011	0.000269	0.001474	0.000022
10ªexec	0.000026	0.001552	0.002933	0.000040
Média	0.000031	0.000875	0.002778	0.000025

Tempo Queries(s)	Q5	Q6	Q7	Q8	Q9
1ªexec	0.000012	0.115081	0.000011	0.044754	0.259165
2ªexec	0.088243	0.000484	0.206420	0.044916	0.255872
3ªexec	0.022326	0.000027	0.206823	0.061918	0.257680
4ªexec	0.105177	0.000010	0.207248	0.062508	0.273589
5ªexec	0.102235	0.068778	0.207876	0.045396	0.260535
6ªexec	0.078621	0.396388	0.207121	0.067701	0.203035
7ªexec	0.003455	0.356522	0.207273	0.044754	0.250086
8ªexec	0.115081	0.047405	0.208796	0.062508	0.253624
9ªexec	0.103934	0.001290	0.000010	0.054882	0.257762
10ªexec	0.029741	0.004093	0.210209	0.053026	0.257775
Média	0.064883	0.099008	0.188857	0.054236	0.252913

Tabela de memória utilizada no programa

Memória(GB)	Data-regular	Data-regular-errors	Data-Large	Data-Large-errors
Total	0.4008	0.3643	3.9574	3.8069

Conclusão

Este projeto permitiu-nos melhorar o nosso entendimento sobre a manipulação de diversas estruturas de dados, otimização de tempo de execução e gestão de memória, bem como deu-nos uma nova perspetiva sobre as boas práticas de modularidade e encapsulamento de programas.

Permitiu-nos também ter um primeiro contacto com diversas ferramentas como o “GDB”, “Valgrind”, “time”.

Tendo em consideração o conjunto de trabalho realizado na UC, consideramos que a mesma nos foi muito útil na evolução da nossa capacidade de trabalho em C.