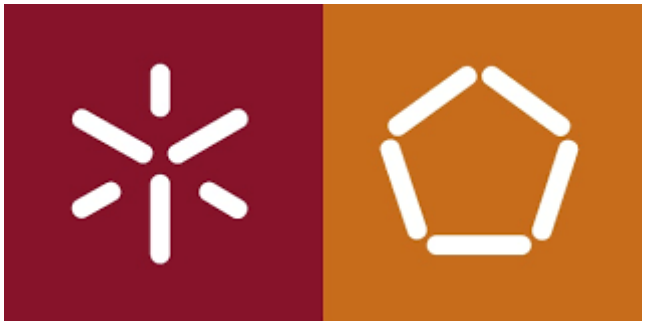

Engenharia Informática

Laboratórios de Informática III

Grupo 98

Ema Maria Monteiro Martins A97678

Henrique Nuno Marinho Malheiro A97455

Índice

Introdução 3
Modelo 3
MakeFile 3
Ficheiros 3
- Main 3
- loadData 3
- Users 3
- Drivers 4
- Rides 4
- Statistics 4
- LoadData2 5
- Q 5
  - Q1 5
  - Q2 5
  - ComparaDate 5
  - ComparaAM 5
  - ComparadriveID 5
  - Q4 5
  - Q5 5

Tabela de tempos de execução 6
Tabela de memóra utilizada no load 6
Observações finais 6

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.

Modelo

Depois de analisarmos a estrutura e finalidade das queries decidimos construir 4 HashTables (“UsersHash”, “DriversHash”, “RidesHash” e “StatisticsHash”), onde as 3 primeiras guardam a informação dos seus respetivos csv(s) e a última guarda os valores calculados que serão utilizados nas diversas queries.

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”.

Ficheiros

Main

Chama o loadData, passando-lhe como argumentos a diretoria dos cvs e o ficheiro com os comandos das queries, respetivamente.

loadData

Através da diretoria para a pasta dos cvs, cria o caminho para cada um deles, individualmente. Chama as diversas funções de parse que criam as respetivas hashs. De seguida chama o loadData2. Por ultimo dá free às hashs.

Users

Durante o parse do “users.csv”, realizado na função OpenFile1 , fomos copiando os vários tokens para o seu respetivo local na struct “users”, que por sua vez seriam guardados num array de struct “users” (“arrayStructs”).

Nesta função também criamos a “UsersHash” que leva como chaves os vários usernames e como valores o ponteiro da struct desse user. Ter em conta que só o fazemos para j>0 de maneira a não por o cabeçalho na hash.

No fim retorna-se um void* da “UsersHash”.

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

Drivers

Tal como no users, durante o parse do “drivers.csv”, realizado na função OpenFile2, fomos copiando os vários tokens para o seu respetivo local na struct “drivers”, que por sua vez seriam guardados num array de struct “drivers” (“arrayStructs2”).

Nesta função também criamos a “DriversHash” que leva como chaves os vários id(s) e como valores o ponteiro da struct desse driver. Ter em conta que só o fazemos para j>0 de maneira a não por o cabeçalho na hash.

No fim retorna-se um void* da “DriversHash”.

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

Rides

Analogamente aos módulos “Users” e “Drivers”, durante o parse do “rides.csv”, realizado na função OpenFile3, fomos copiando os vários tokens para o seu respetivo local na struct “rides”, que por sua vez seriam guardados num array de struct “rides” (“arrayRides”).

Nesta função também criamos a “RidesHash” que leva como chaves os vários id(s) e como valores o ponteiro da struct dessa ride. Ter em conta que só o fazemos para j>0 de maneira a não por o cabeçalho na hash.

No fim retorna-se um void* da “RidesHash”.

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 “BuildStatisticHash”, começamos por definir os 3 arrays de structs (“arrayStatisticsDrivers”,”arrayStatisticsUsers” e “arrayStatisticsCity”) que irão armazenar as diversas structs de “users”, “drivers” ou “city”. Os valores necessários para calcular as diversas estatísticas são obtidos através dos diversos “gets” da “RidesHash” e também da “DriversHash” (para saber qual a classe do carro do condutor e consequentemente a tarifa a ser cobrada na viagem).

Nesta função também criamos a “StatisticsHash” que leva 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.

No fim retorna-se um void* da “StatisticsHash”.

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

LoadData2

Esta função faz o parse de um documento que leva as queries e os argumentos necessários para a execução das mesmas. Dessa forma, faz-se o parse por linhas, uma vez que cada uma delas corresponde a uma chamada de uma querie. Dentro da linha, faz-se o parse pelo “ “ de modo a que o primeiro argumento dado seja o número da querie que vamos chamar do ficheiro “q” e o resto corresponde ao aos argumentos necessários para a realização da mesma.

Q

Q1

Verifica se o id/username passado existe nas HashTables e, caso existe, 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 num documento de texto.

Q2

Cria um array com as chaves das hash “drivers” de modo a criar um array de structs que contenha o id, o nome, a avaliação média e a data. Posteriormente aplicamos o qsort de modo a ordenar o array de structs primeiramente pela avaliação media. Em caso de empate ordena por data de viagem e em última instância ordena por id. Retorna os n primeiros elementos desse array. Escreve o resultado num ficheiro de texto.

ComparaDate

Verifica de uma data é mais recente, igual ou antiga que outra.

ComparaAM

Verifica se uma avaliação média é maior, igual ou menor que outra.~

ComparadriveID

Verifica se um id é maior, igual ou menor que outro.

Q4

Verifica se uma dada cidade se encontra na StatisticsHash. Em caso afirmativo escreve num ficheiro de texto o preço médio das viagens nessa cidade. Caso contrário, apenas cria um documento vazio.

Q5

Verificamos se as datas das diversas viagens estão entre as duas datas passados como argumento. Se essa condição se verificar, calculamos a tarifa da viagem (tendo em conta a car

class do driver), guardando as várias somas na variável valor. No fim, divide-se pelo número de viagens nesse intervalo, calculando então a média.

Tabela de tempos de execução

	Texec1 (s)	Texec2 (s)	Texec3 (s)	Texec4 (s)	Média (s)
LOAD	2.277593	2.278403	2.286326	2.268781	2.277776
Q1	0.000071	0.000105	0.000022	0.000105	0.000076
Q2	0.015689	0.016405	0.016555	0.016355	0.016251
Q4	0.000037	0.000037	0.000148	0.000030	0.000063
Q5	0.410535	0.412232	0.355857	0.411184	0.397452

Tabela de memóra utilizada no load

	1ªexce (G)	2ªexec (G)	3ªexec (G)	4ªexec (G)	Média (G)
LOAD	0.65	0.66	0.66	0.65	0.66

Observações finais

De um modo geral, consideramos que o tempo de execução das queries é bastante razoável. No entanto, percebemos que na segunda parte do trabalho temos de melhorar a gestão de memória, nomeadamente a quantidade de frees.