Laboratórios de Informática III – 2022/2023
Fase 2

Grupo 24

João Pedro Baptista (a100705), João Rodrigues (a100896), Mateus Martins (a100645)

1. Encapsulamento

Para a concretização da continuação do trabalho (fase 2) tivemos que encontrar uma melhor estratégia para o encapsulamento, pois admitimos que o da primeira fase era muito escasso.

Para tal, decidimos implementar uma série de funções que foram inseridas nos módulos dos catálogos de cada CSV, tais funções que permitem obter as características de cada User, Driver ou Ride sem comprometer a privacidade das estruturas.

Começamos por remover as estruturas dos ficheiros ".h", deixando lá apenas os "typedef", transferindo as mesmas para os ficheiros ".c" correspondentes. Posteriormente, tivemos de criar funções tais como "int get_n_drivers()", "bool is_driver(char *id)" para que não fosse necessário os módulos restantes acederem ao catálogo em si.

Finalmente, apenas foi necessário trocar os momentos em que as queries acediam aos catálogos diretamente, pelas funções agora criadas, e remover os catálogos dos argumentos das mesmas.

2. Verificação de Dados

Em relação à verificação de dados encontramos alguns contratempos, pois no enunciado referia que os score_user e score_driver poderiam ser decimais e iguais a zero, porém posteriormente foi-nos dito que teriam de ser, tais como a distância, inteiros e maiores que 0.

Implementamos as funções "isDvalid", "isUvalid" e "isRvalid" para verificar se um dado Driver, User ou Ride seriam válidas para se adicionar ao catálogo correspondente. Finalmente, apenas precisamos de as inserir nas funções "inserir_drivers", "inserir_users" e "inserir_rides" para que apenas fosse inserido nos respetivos catálogos os elementos válidos.

3. Memória (Datasets Maiores)

Para resolver alguns problemas de memória que tínhamos, tais como "memory exceeded", decidimos implementar arrays realocáveis (usando realloc). As arrays dos catálogos começavam com capacidade para 2 elementos, sendo que quando fosse necessário, o realloc interviria para alocar memória para mais elementos, duplicando a capacidade.

Outra estratégia que utilizamos para diminuir a memória usada do programa foi diminuir a o tamanho de cada elemento, como por exemplo: diminuímos o tamanho do ID de 15 para 11 characters, fazendo o mesmo para as datas, entre outros.

4. Modo Interativo

Para a realização do modo Interativo, adicionamos uma condição na função main que verifica quando é que o programa é executado sem argumentos (argc == 1). Quando o programa é executado nesse modo, já não cria a pasta “Resultados”.

Inicialmente, o programa irá “pedir” para o utilizador digitar o caminho para os ficheiros CSV, para o mesmo conseguir chamar as funções que os inserem nos respetivos catálogos.

Decidimos também adicionar um novo argumento às queries (mode) para as mesmas “saberem” se estão no modo Batch (0) ou no modo Interativo (1).

Criamos um módulo adicional referente ao modo Interativo que contém o funcionamento idêntico ao input do modo Batch, porém passa-se como argumento “mode” 1, ao invés de 0 do modo Batch para que apenas imprima no terminal as respostas às queries quando o programa se encontra no modo Interativo.

Para terminar, quando o utilizador pretende sair do programa, apenas precisa de digitar “exit”.

Em relação à paginação dos resultados longos, não conseguimos implementar essa feature, pois demos mais importância a outros aspetos.

5. Queries

As queries realizadas nesta fase (Q7, Q8 e Q9) são muito parecidas com as queries anteriores.

Em particular, a Q7 é idêntica à Q2, necessitando apenas de adicionar uma condição para colocar condutores apenas da cidade pedida.

Em relação às Q8 e Q9, as estratégias utilizadas foram similares às restantes, apenas necessitando de uma pequena adaptação para a respetiva Query.

Retardamos a ordenação das queries Q2, Q3, Q7, Q8 e Q9 para a 2ª Fase pois já estávamos com falta de tempo. O pensamento para a mesma foi similar para todas as queries, usamos o quicksort da biblioteca C, aplicando uma função auxiliar para cada exercício que apenas ordenava dois elementos, sendo que a função incluída já ordenava a array independentemente do tipo de dados.

6. Testes Funcionais e de Desempenho

Implementamos um módulo “tests.c” que faz os mesmos testes do site li3.di.uminho.pt. A partir de cada Dataset, verifica cada input do mesmo e imprime no ecrã “Test 12 passed”, sendo 12 a linha de input. Mostra também o tempo de execução de cada query, quer o teste tenha passado ou não.

Para tal, fizemos uma função “compare” que, utilizando um loop while com 2 fgets verifica se cada linha dos dois ficheiros são iguais, até chegar ao fim dos mesmos.