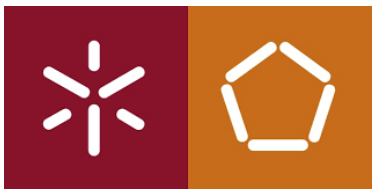

Universidade do Minho

Laboratórios de Informática III

Relatório grupo 72 – Fase 1

22 de Novembro de 2022

Luís Fraga (a100749)

Jorge Teixeira (a100838)

Nuno Aguiar (a100480)

Índice

Objetivo:

Nesta Unidade Curricular, foi-nos proposto criar um programa, em linguagem de programação C, capaz de ler ficheiros de formato XML para, através de mecanismos eficientes (como modularidade e encapsulamento, estruturas dinâmicas de dados, medição de desempenho, etc.), responder a determinados problemas propostos. Estes problemas seriam respondidos em dois modos de operação (*batch* e *interativo)*.

Método Prático Utilizado:

Após análise do projeto proposto decidimos, em grupo, como responder a este problema, mais concretamente à primeira fase do mesmo. Detetamos a necessidade de executar 3 “fases” para poder trabalhar com este tipo de dados.

Assim sendo, numa primeira instância organizamos dados de **Utilizadores** (users.csv), **Condutores** (drivers.csv) e **Viagens** (rides.csv) em estruturas adequadas (abordadas posteriormente).

Numa segunda abordagem, criamos forma de responder aos problemas (Queries) , no modo **Batch**, criando funções que nos permitissem ler e escrever em ficheiros.

Por fim, focamos em criar funções que respondessem em concreto a cada **Query**.

Organização dos Dados

Em relação aos **Condutores**, conseguimos organizar os seus dados num array, ao qual chamamos de "**Id**", em que o próprio **ID** define a posição em que a informação se encontra armazenada. Isto é, para o Condutor com **ID** '0…0' a sua informação encontra-se na primeira posição do array.

Já quanto aos **Utilizadores**, decidimos organizar as suas informações de forma semelhante aos Condutores. Contudo, para organizar os mesmos num array "**Iu**" utilizamos, com base em funções do género "hash table", um conjunto de funções que utilizam os **usernames** para criar um número inteiro que serve de índice e permite organizar os dados, ordenadamente no array.

Por fim, para as **Viagens,** ainda não implementamos nenhuma funcionalidade que guarde, em memória, o seu conteúdo visto ainda não ser necessário para responder às queries que selecionamos para a primeira fase. Assim, apenas analisamos o documento e atualizamos os dados referentes a cada **utilizador**/**condutor**.

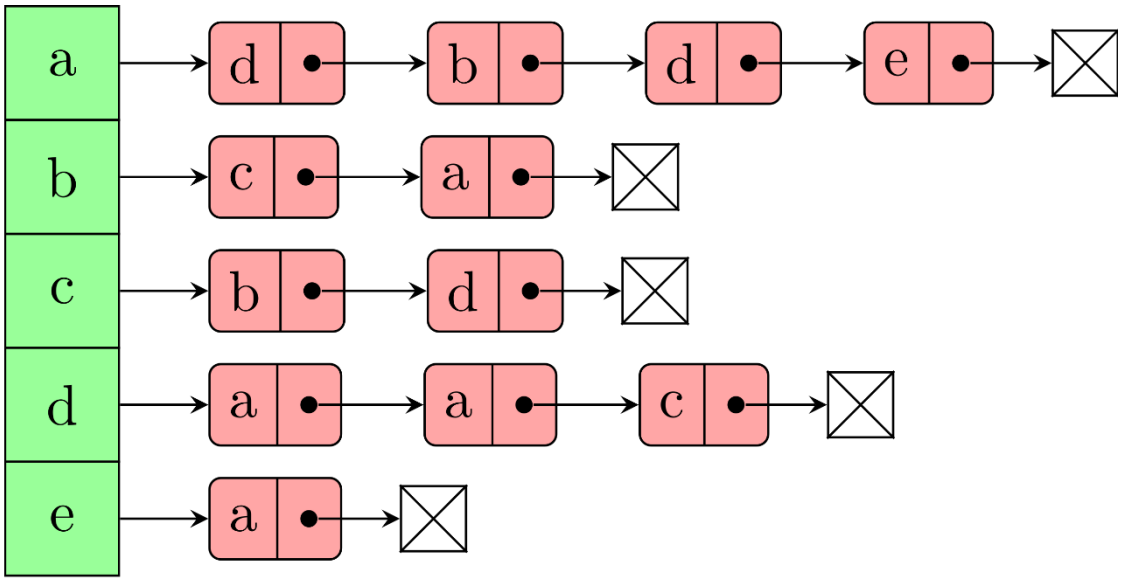

Imagem.1-Exemplo visual do método utilizado.

Hash Table

Na organização dos dados, utilizamos várias funções baseadas em “hashing”.

Com estas funções pretendemos:

-**Condutores**: Dado o facto de poder não ser do nosso conhecimento o número de condutores existentes, escrevemos uma função ***lookUpTableD*** que, para o caso de o número do ID do Condutor superar o tamanho definido do array **ld**, continua a ser possível armazenar o ID e os dados do mesmo.
Esta função, que recebe o **ID** e o próprio array (**ld)** como argumentos, utiliza o resto da divisão entre o **ID** e o **TABLE_SIZE_D** (tamanho atual do array) como **INDEX** e “anexa” ao endereço **INDEX** correspondente , em forma de lista ligada, a informação de um condutor. Por exemplo, se o limite fosse **100** e quiséssemos ler **150 condutores**, a partir do condutor com **ID 101**, o programa anexava à **posição 1** do array **ld**, em forma de lista ligada, a informação do condutor **101** (pois 101%100 = 1).

-**Utilizadores**: De forma semelhante ao método que usamos para ordenar os **condutores**, também utilizamos **funções de “hashing”** à ordenação dos **utilizadores**. Contudo, existe uma diferença para os tipos de dados. Visto que os condutores possuem **ID’s** sendo facilmente associados a posições de arrays, o mesmo já não acontece com os utilizadores. Assim sendo, utilizamos as funções ***hash*** para através do **username** de cada utilizador, gerar um número inteiro e desta forma conseguir agrupar ordenadamente as informações num array **lu**. Utilizando este método, verificamos a presença de sobreposições devido ao código que gerava números iguais através de usernames diferentes. Com o intuito de reduzir esses sobreposições utilizamos a mesma técnica anterior de **INDEX,** adicionando em listas ligadas a posições com usernames já existentes.

Em relação ao ficheiro das viagens, tal como dito anteriormente, não criamos funcionalidades que guardem em concreto as informações contidas neste documento. Até ao momento apenas aplicamos a funcionalidade de, através da função **loadTableR**, este ficheiro “atualizar” as informações relativas tanto aos **utilizadores** como aos **condutores**.

Modularização e Encapsulamento

O nosso trabalho está dividido por vários módulos divididos ponderadamente de acordo com as necessidades que consideramos existirem.

Assim, temos:

- **main.c** – main do nosso programa.
- **drivers.c** – toda a informação necessária para tratar das informações dos condutores.
- **users.c** – toda a informação necessária para tratar das informações dos utilizadores.
- **rides.c** – informação necessária para atualizar os dados tanto a condutores como a utilizadores.
- **func.c** – contém funções auxiliares ao analisar dados.
- **querry1.c** – informação necessária para responder à 1ª Query.
- **querry2.c** – informação necessária para responder à 2ª Query.
- **querry3.c** – informação necessária para responder à 3ª Query.

O objetivo de dividir/encapsular o projeto em diferentes módulos consiste no facto de cada um se tornar "independente" e assim se complementarem através da main.

Resolução das Queries

Query 1- Na nossa resolução a esta query usamos a função, por nós criada, **lookUpTableU** e **lookUpTableD** para ler diretamente a informação armazenada no array dos utilizadores e condutores, respetivamente. Dado que foi de resposta direta bastou-nos por fim, devolver as informações correspondentes ao **ID/Username** pedido.

Query 2- Nesta query, já não nos é possível apresentar uma resposta direta, pois temos de comparar as médias de todos os condutores, uns com os outros. Assim decidimos **criar uma estrutura auxiliar**, uma lista ligada de nome **linkedListD**. Esta lista guarda um apontador para um determinado "perfil" de um condutor. Assim, com as condições lógicas necessárias pusemos esta lista a guardar os **N** condutores com **maior média** imprimindo-os posteriormente e certificando-nos que libertávamos o espaço alocado temporariamente. Esta organização tem em conta que, **em caso de empate**, se considera a **data da última viagem** realizada (de forma descendente) e em caso de um segundo empate deverá ter em conta o **ID** dos condutores em questão.

Query 3- De modo semelhante à query anterior, também não nos é possível obter, de modo direto, a resposta a este problema. Consequentemente, aplicamos a técnica de criar uma **segunda estrutura auxiliar**, de nome **linkedListU**. Assim, guardamos nesta mesma estrutura as informações dos **N** utilizadores com maior distância média de viagem percorrida de forma decrescente, imprimindo-as posteriormente, novamente certificando-nos que libertamos o espaço alocado temporariamente. Esta organização considera para o caso de empate, a data da viagem da última viagem realizada (de forma descendente) e em caso de segundo empate, utilizando o **strcmp**, compara os **usernames**. O **strcmp** é uma função que dadas **duas strings**, por exemplo **a** e **b**, o valor que retorna **determina** qual das duas strings é colocada em **primeiro**. Se, neste caso **strcmp(a, b) < 0**, na lista, **a** apareceria **primeiro que b**.

Na resolução destas queries apenas tivemos em consideração condutores e utilizadores com conta ativa.