Laboratórios de Informática III

2023-2024

1ª Fase

Elaborado por:

Adélio José Ferreira Fernandes – a78778

Leonardo Gomes Alves – a104093

Salvador Duarte Magalhães Barreto – a104520

Índice

Introdução

No âmbito da disciplina de Laboratórios de Informática III do 2º ano da Licenciatura de Engenharia Informática, no presente ano letivo 2023-2024, fomos desafiados a expandir os nossos conhecimentos de programação, nomeadamente da linguagem C.

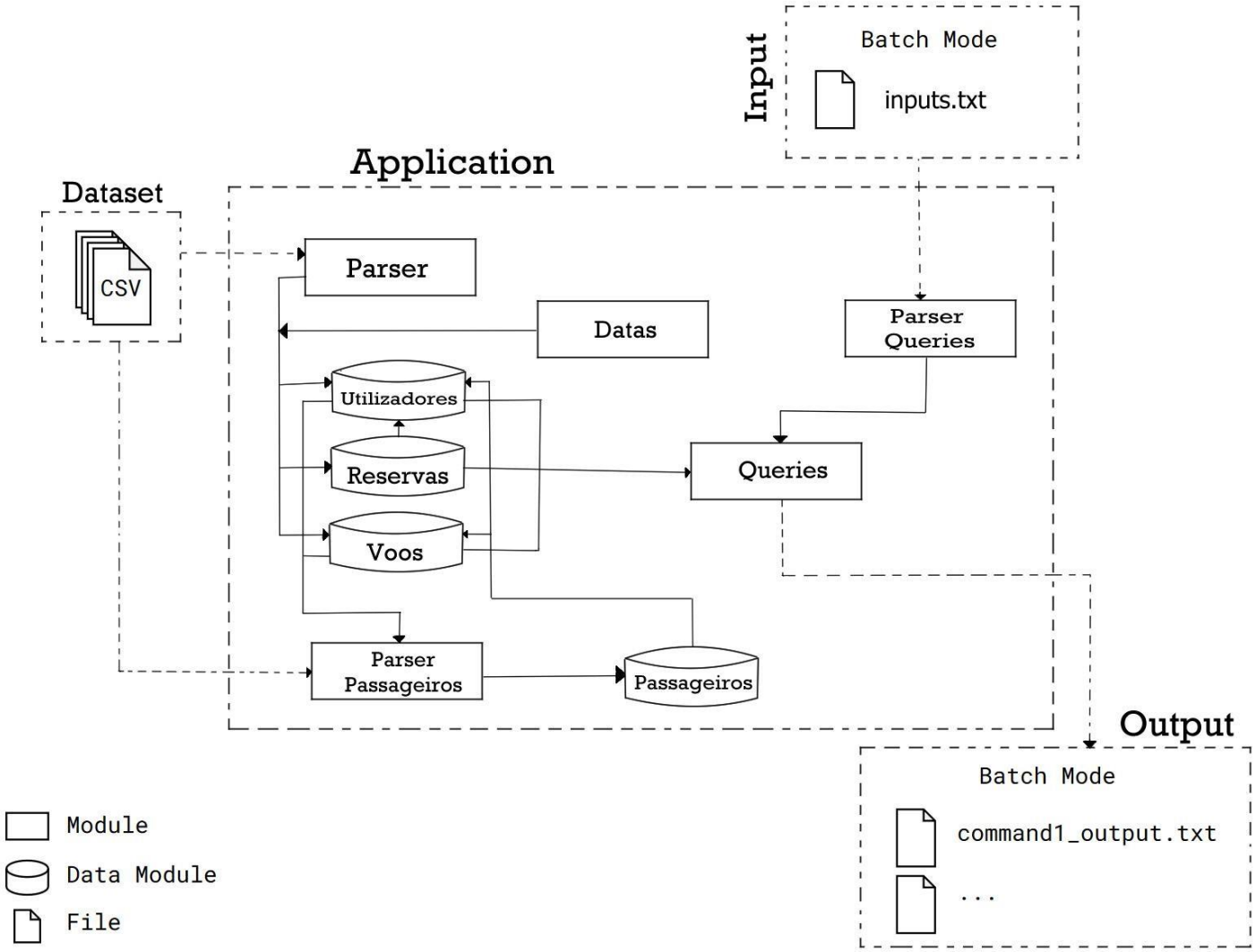
Temos como objetivo principal, aprender a manipular datasets, utilizando estruturas de dados; modularidade e encapsulamento; gestão de memória etc....

Para tal, foram nos fornecidos 4 datasets, com informações relativas a voos, passageiros, reservas e utilizadores. Cada dataset está entreligado com pelo menos outro dataset.

Exemplo:
O dataset dos voos tem informações relativas a um voo como por exemplo, a data de partida ou o número do voo. Por sua vez o dataset dos passageiros, tem informação acerca dos passageiros que viajam em um determinado voo.

Nas próximas páginas, iremos abordar os métodos de resoluções que utilizámos para os desafios propostos.

1. Estruturas de dados



1.1. Representação gráfica

Os ficheiros CSV's dos utilizadores, reservas e voos são enviados para o Parser que, com ajuda de um módulo de Datas (que contem funções que tratam datas e horas) guarda nos respetivos módulos. Já o ficheiro CSV dos passageiros, é enviado para o Parser de Passageiros, junto com os módulos de datas dos utilizadores e voos, para a sua informação poder ser tratada e posteriormente atualizar os módulos dos voos e dos utilizadores. Os três módulos são enviados para o módulo das Queries.

Paralelamente, o ficheiro de input é enviado para o Parser Queries, onde a informação de cada linha é tratada e enviada para o módulo das Querys.

Finalmente, a partir do módulo das Queries, é gerado o output de cada querie.

1.2 Estruturas de dados

No âmbito de fazermos o parsing relativo aos datasets, tivemos que decidir quais seriam as estruturas utilizadas para guardar todos os dados. Surgiu então a ideia de utilizarmos hashtables pois, não só o tempo de procura era nulo, como era simples aplicar aos nossos dados. No entanto, surgiu a dúvida de como seria possível caso quiséssemos acessar a todos os elementos de um dado tipo (e.g. utilizadores). Face a esta dúvida, decidimos aplicar um array com todos os elementos de um dado tipo.

Para podermos guardar os elementos de um dado dataset, utilizámos várias structs para cada tipo de dados. Desta forma, tanto poderíamos associar a struct ao array, como à hashtable.

Temos estruturas para guardar voos, reservas, utilizadores, hotéis e aeroportos; é de realçar que, a partir da leitura de um dos datasets, podemos inserir informações em outra estrutura de outro dataset, como é o exemplo das reservas, que são inseridas também informações nos utilizadores.

1.3 Datas

Para o tratamento das datas, escolhemos armazená-las em formato de string. Consequentemente, utilizamos a função **dataToArray**, que transforma a string num array, contendo a posição 0 o ano, a 1 o mês, etc.

Para fazer cálculos com as datas, escolhemos usar anos. Consideramos a data atual do sistema como 2023/10/01, ou seja, 2023.836 anos.

2. Parser

Uma das funções mais importantes, que é a base do nosso programa, é a função de parser, que tem como objetivo ler dos ficheiros dos datasets e, associar cada linha, caso seja válida, a uma dada estrutura de dados.

Para tal, a nossa função de parser, começa por ler linha a linha dos ficheiros e, a cada uma, ele retira os "pontos e vírgulas", utilizando a função **strsep**, e guarda cada uma das palavras num array auxiliar.

Com o array auxiliar criado, é chamada a função **verifica**, que é diferente para cada estrutura de dados e, como o próprio nome diz, verifica se uma dada linha pode ser válida, verificando se não existe nenhuns tipos de erros possíveis que nos são mencionados no guião do trabalho. Caso uma linha não for válida por falhar qualquer condição, será impressa no ficheiro de erros; caso contrário, é inserida na sua struct respetiva e inserida num array.

Por fim, a fase anterior repete-se até à última linha do dataset e, finda esta fase, é chamada a função **organiza**, que recebe como argumento a o array que foi criado até então com as linhas válidas e, associa então cada linha a uma dada hashtable. Por exemplo, uma linha do dataset dos *utilizadores*, é guardada numa hashtable em que a sua chave será o id do utilizador. É criada então a struct de Save_x (onde x pode ser users, reservations ou voos), que contém a hashtable de cada estrutura e o seu array de structs com todos os elementos válidos.

3. Queries

O que se segue são as nossas respetivas resoluções para cada query que desenvolvemos na primeira fase do trabalho.

- *Query* 1

A função da **query 1** recebe um id (que tanto pode ser de uma reserva, de um voo, ou de um utilizador); as estruturas de dados dos utilizadores, dos voos e das reservas; um inteiro **i** com o objetivo de gerar o nome do arquivo de saída e um inteiro **type**, que decide a forma como o output será impresso.

A função tem como objetivo procurar o objeto associado ao id e imprimir as suas informações relativas no ficheiro de saída. Em primeiro lugar, a função verifica qual é o tipo de id que se trata (utilizadores, voos e reservas) e, após esta verificação, utiliza a função **g_hash_table_lookup** para ir buscar as suas informações à correspondente hashtable e, imprime no ficheiro de output. É de realçar que, caso sejam pedidas informações acerca de um utilizador, apenas são retornados valores caso o estado da sua conta seja ativo.

- *Query 2*

A função da **query 2** recebe como argumentos, um id de um utilizador e o tipo de dados que quer imprimir, voos, reservas ou ambos e tem como objetivo também, organizar os dados por data.

Para tal, a função principal começa por criar verificar se tal id é válido e existe na hashtable dos utilizadores. De seguida, antes de realizar qualquer operação, tal como acontece na query 1, a função verifica se o usuário está ativo e, caso isto for verdadeiro, prossegue para os próximos passos.

Depois, verifica qual o tipo de dados que o input quer acerca do utilizador (reservas, voos ou ambos), caso quiser ambos, o input será apenas o número da query e o id. Posto isto, conforme o input, verifica se as datas estão organizadas e, caso não estejam, são chamadas funções auxiliares para as organizar (**organizaVoosUser**, **organizaReservasUser** e **organiza_VoosEReservasUse**r) e, por fim, chama as funções de impressão.

- *Query* 3

A **query 3**, tem o objetivo de apresentar a classificação média de um hotel, a partir do seu identificador.

Para tal, a função começa por validar o identificador do hotel e, posteriormente, é chamada a função **calculaMediaHotel**, que calcula a média do hotel que nos é fornecido no input. Por fim, como todas as outras funções sobre queries, imprime no seu respetivo ficheiro.

- *Query 4*

A query 4 tem o objetivo de listar as reservas de um hotel, ordenadas por data de início (da mais recente para a mais antiga), caso ambas as datas forem iguais, aparece em primeiro lugar a reserva que tiver o menor identificador.

Começa então por verificar se o hotel dado no input é válido e, caso seja, é chamada a função **sortBookingHoteis**. Esta função recebe os dados de todos os hotéis e reservas e, procura o tal hotel na sua hashtable. De seguida, com o intuito de organizar as reservas por data, verifica reserva a reserva qual é a sua data de início, utilizado a estrutura das reservas. Desta forma, comparando as datas, organiza-as. Por fim, é passado novamente este algoritmo, mas que desta vez verifica as datas e, caso duas sejam iguais, verifica qual é a que tem menor id e coloca-a em primeiro lugar.

Por fim, é chamada a função de impressão da query.

- *Query 7*

A query 7 tem como objetivo listar o top N aeroportos com a maior mediana de atrasos (caso dois aeroportos tenham a mesma mediana, o nome do aeroporto deverá ser usado como critério de desempate).

Para responder a esta query criamos uma nova struct: um array de aeroportos, em que cada aeroporto contém: o seu nome, o número total de voos, um array com os delays de cada voo que partiu desse aeroporto e a própria mediana de delays.

Através da função **criaSaveAeroportos**, que percorre a estrutura de dados dos voos e vai verificando se um voo, vindo de uma dada origem existe, se sim, atualiza o conteúdo na nova estrutura, se não, acrescenta-o à estrutura.

No final de todos os voos serem percorridos, utilizamos a função **calculaMedianaDelays**, que primeiro, usando a função qsort, ordena os delays de cada voo por ordem crescente e de seguida calcula a mediana desse conjunto.

De seguida, usamos a função **ordenarPorMedianaDecrescente**, que usa a função qsort para ordenar por ordem decrescente o array de aeroportos de acordo com as suas medianas de delays ou por ordem alfabética caso se aplique.

Na função da query 7 que recebe esta base de dados de aeroportos, imprime os aeroportos, conforme o N que o input pedir.

- *Query 9*

A query 9 tem como objetivo listar todos os utilizadores cujo nome começa pelo prefixo dado no input.

Em primeiro lugar, tiramos possíveis caracteres (aspas) que não façam parte de um nome, do prefixo. De seguida, copiámos o array dos utilizadores para um array auxiliar, com o objetivo de organizar os nomes. Assim, é chamada função qsort, em que os argumentos são o array auxiliar, o tamanho desse array, o tamanho da struct, e a função **compararUsers**, que tem o objetivo de organizar os nomes por ordem alfabética.

Por fim, passámos pelo array dos utilizadores e verificámos se o nome do utilizador começa com o prefixo e se tal tem a conta ativa. Verificadas estas condições, damos print no ficheiro conforme o nosso input.

4. Main

Na função Main, primeiro verificamos se o número de argumentos é diferente de 3, caso seja, dá erro, pois precisamos de 3 argumentos para executar o modo batch.

Depois, procedemos à abertura e envio dos 4 ficheiros CSV para o respetivo parser, para que possam ser tratados e armazenados.

Se seguida procedemos à organização das novas estruturas de dados usadas para certas queries.

Posteriormente, abrimos o ficheiro de input e enviamos para o parser de queries que vai ser responsável também por enviar cada linha tratada para a função que escolhe qual a query que vai ser executada.

No final, damos free as estruturas que usamos.

5. Conclusão

Para finalizar o nosso trabalho da 1ª fase, gostaríamos de mencionar que achámos que fizemos um bom trabalho para aquilo que nos foi proposto, no entanto, pretendemos para a segunda fase, melhorar algumas das nossas funções para que, sejam mais eficientes e mais rápidas e também explorar mais a modularidade e melhor o encapsulamento das nossas funções.

Gostaríamos de realçar ainda que, esta primeira fase enriqueceu os nossos conhecimentos acerca de estruturas de dados e manipulação de ficheiros, visto que, nunca nos tínhamos aprofundado tanto neste tema, como nesta U.C. de Laboratórios de Informática III.