Universidade do Minho
Departamento de Informática

Fase 1

Laboratórios de Informática III

Grupo 7

Flávio Alexandre Silva (A97352)
José Pedro Vasconcelos (A100763)

24 de novembro de 2023

Índice

1 Introdução **2**

2 Desenvolvimento **3**
2.1 Arquitetura de aplicação 3
2.2 Validação de dados 3
2.3 *Parsing* e inserção nas estruturas de dados 4
2.4 Modo *Batch* 4
2.5 Queries 4
2.5.1 Query 1 4
2.5.2 Query 2 5
2.5.3 Query 3 5
2.5.4 Query 4 5
2.5.5 Query 5 6
2.5.6 Query 7 6
2.5.7 Query 8 6
2.5.8 Query 9 6

3 Dificuldades sentidas **8**

4 Conclusão **9**

1 Introdução

Este projeto está a ser desenvolvido no âmbito da Unidade Curricular de Laboratórios de Informática III, no presente ano letivo de 2023/2024, e tem como objetivos a consolidação de conhecimentos essenciais da linguagem C e de Engenharia de Software, nomeadamente, modularidade e encapsulamento, estruturas dinâmicas de dados, validação funcional, e medição de desempenho, assim como a consolidação do uso de ferramentas essenciais ao desenvolvimento de projetos em C, nomeadamente, compilação, linkagem, definição de objetivos de projeto com base nas suas dependências, depuração de erros, avaliação de desempenho e consumo de recursos, e gestão de repositórios colaborativos.

Nesta primeira fase do projeto, que terá duas fases, foi-nos proposto que implementássemos o *parsing* dos dados, o modo *batch*, 60% das *queries* presentes no enunciado, assim como a validação do *dataset* e que o nosso programa corra sem *memory leaks*.

No final, pretende-se que o nosso programa responda, com sucesso, a todos estes requesitos, com alta performance.

2 Desenvolvimento

2.1 Arquitetura de aplicação

Com o intuito de se ter uma visão geral do funcionamento da nossa aplicação, foi criada a seguinte arquitetura:

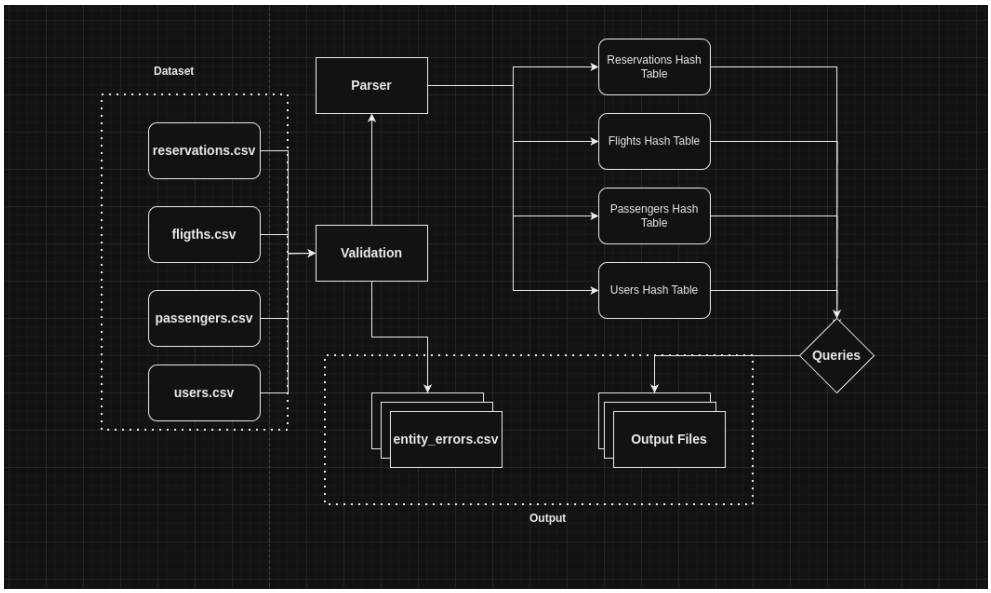

Figura 1: Diagrama da arquitetura da nossa aplicação.

Cada módulo presente nesta arquitetura, assim como o funcionamento do *modo batch* e das *queries* desenvolvidas nesta primeira fase do projeto, serão explorados com mais detalhe nos próximos capítulos deste relatório.

2.2 Validação de dados

De acordo com o enunciado que nos foi fornecido, nesta fase do projeto, teríamos que ter o cuidado de verificar se cada campo dos ficheiros *csv* são válidos. Assim, criamos um programa que faz esta validação, para o *user, flight, passenger, reservation*.

Desenvolvemos um módulo responsável pela validação de dados, denominado *validation.c*. Este programa integra diversas funções genéricas de validação, abrangendo *strings, ints, doubles e dates*, que são tipos específicos de dados essenciais para a filtragem. As funções *invalid_flights*, *invalid_users*, *invalid_passengers* e*invalid_reservations* percorrem cada linha do respetivo ficheiro *.csv*, dividindo-a em tokens separados por ";"usando `strtok`. Ao validar cada campo da linha com a função respectiva ao seu formato, a variável *checker* é inicializada com o valor 1 mas pode se tornar 0 se um termo for inválido ou se a linha não tiver o número correto de campos, resultando na escrita da linha no arquivo *file.errors.csv* correspondente.

Para verificar se determinados usuários ou voos inválidos estão presentes nas reservas ou nos passageiros e precisam ser validados, utilizaremos duas listas: `invalid_uids` e `invalid_fids`, que armazenarão os valores dos IDs dos usuários e dos voos inválidos, respectivamente. Essas listas serão usadas para verificar a presença desses IDs nas listas de passageiros (`passengers`) e reservas (`reservations`).

Se a sua presença for confirmada, atualiza-mos o valor do verificador (`checker`) para 0.

2.3 Parsing e inserção nas estruturas de dados

Depois de chegarmos à conclusão que uma dada linha é valida, verificamos se já existe algum registo, na respetiva *Hashtable*, com o *id* especificado. Se já existir, eliminamos o novo registo, se não, inserimos na *Hashtable*. Como já foi referido na explicação dada acima, foram usadas *hash tables* já implementadas, na *glib* tanto para os utilizadores, como para os voos, reservas e passageiros. No caso específico dos passageiros, criamos um *id* para cada um , que é a concatenação do *flight_id* com "_"e o *user_id*.

Nós escolhemos utilizar estas estruturas de dados da *glib* pois, para além de já estarem definidas, têm funções de procura que nos vão ajudar a encontrar as informações que necessitamos, para cada *query*, em tempo de execução dentro do estipulado.

2.4 Modo Batch

Para utilizar o modo batch, temos que passar dois argumentos ao programa: o caminho, a partir da diretoria em que nos encontramos, **grupo-07/trabalho-pratico**, para o *dataset* e para o ficheiro de entrada, *input.txt*. Como nos foi proposto, neste modo, o programa deve criar um ficheiro, na pasta Resultados, por cada linha do ficheiro de entrada. Por exemplo, o ficheiro *command1_output.txt* seria o resultado da execução *query* presente na primeira linha do ficheiro *input.txt*.

Caso uma *query* no modo *batch* contenha a flag 'F', o seu resultado deverá ser apresentado com o formato *field: value*, para além da indicação do número do registo no seu início (através de — *n* —, onde *n* é o número do registo). Foram criadas funções para cada *query* implementada, para alterar a forma como as informações são escritas no ficheiro de *output*, e.g , para a *query 1* existe a função *query1* e a *query1_format*.

2.5 Queries

2.5.1 Query 1

Esta primeira *query* consiste em *"listar o resumo de um utilizador, voo, ou reserva, consoante o identificador recebido por argumento."*, onde os identificadores são o *id* para os *users*, o *id* para os *flights* e o *id* para as *reservations*, tendo em atenção que *não deverão ser retornadas informações para utilizadores com account_status = "inactive".*

O facto destes identificadores serem a chave de cada uma das respetivas *hash tables*, ajudou bastante para encontrar o perfil que estavamos à procura, em tempo útil. Apenas para os últimos campos do *output*, tivemos um pouco mais de trabalho, tendo que iterar pelas variadas *hash tables* para calcular dados como *number_of_reservations* para os *users*, ou *total_price* para as *reservations*, ou até mesmo os *passengers* de um dado *flight*.

2.5.2 Query 2

A segunda *query* consiste em *"listar os voos ou reservas de um utilizador, se o segundo argumento for flights ou reservations, respetivamente, ordenados por data (da mais recente para a mais antiga). Caso não seja fornecido um segundo argumento, apresentar voos e reservas, juntamente com o tipo (flight ou reservation)."*, tendo em conta que, tal como na *query 1, utilizadores com account_status = "inactive" deverão ser ignorados e que para os voos, date = schedule_departure_date (contudo, deverá ser descartada a componente das horas/minutos/segundos no output), enquanto que para as reservas, date = begin_date.* Para além disto, são fornecidos critérios de desempate, os quais devemos ter em conta, a saber, *Em caso de empate, ordenar pelo identificador (de forma crescente).*

Para resolução desta *query*, utilizamos estruturas de dados auxiliares da *glib*, a lista ligada (*GList*). Se nos for pedido, a título de exemplo, as reservas de um *user*, percorremos a *hash table* das *reservations* , e verificamos quais foram feitas pelo nosso *user*. À medida que encontramos essas reservas vamos adicionando-as à nossa lista ligada, que posteriormente será ordenada conforme os resquisitos da *query* em questão. Após isso, só nos resta iterar pela lista e escrever no ficheiro de *output*.

2.5.3 Query 3

A terceira *query* consiste em *"apresentar a classificação média de um hotel, a partir do seu identificador."*

Na nossa opinião, esta é uma das *queries* mais simples presentes no enunciado, pelo que, foi uma das primeiras a serem resolvidas pelo nosso grupo. A estratégia que utilizamos foi percorrer a *hash table* das *reservations*, guardando a soma de todas as *ratings* do *id* do hotel que nos foi passado como argumento da *query* e a contagem do número de *reservations* desse mesmo hotel. Para finalizar, só temos que fazer a divisão da soma dos *ratings* pelo número de registos desse hotel e escrever no ficheiro de *output* com o formato "%.3f" (três casas decimais).

2.5.4 Query 4

A quarta *query* consiste em *"listar as reservas de um hotel, ordenadas por data de início (da mais recente para a mais antiga)."* Em caso de empate, *deve ser usado o identificador da reserva como critério de desempate (de forma crescente).*

À semelhança da *query 2*, utilizamos uma estrutura de dados auxiliar, a *GList*, na qual armazenamos as reservas do hotel em questão, dentro das datas estipuladas. Posteriormente, ordenamos esses registos de forma crescente e escrevemos essa lista no ficheiro de *output*.

2.5.5 Query 5

A quinta *query* consiste em *"listar os voos com origem num dado aeroporto, entre duas datas, ordenados por data de partida estimada (da mais antiga para a mais recente). Um voo está entre begin_date e end_date caso a sua respetiva data estimada de partida esteja entre begin_date e end_date (ambos inclusivos)"*. Em caso de empate, "o identificador do voo deverá ser usado como critério de desempate (de forma crescente)".

Mais uma vez, usamos uma lista ligada como estrutura auxiliar, na qual começamos por armazenar os valores que estavam na *hash table* dos *flights*. Ordenamo-la por ordem crescente e posteriormente, invertemos a ordem. Para finalizar, percorremos a lista dos *flights* e sempre que nos surgiu um *flight* que reunia os requesitos da *query*, escrevemo-lo no ficheiro de *output*.

2.5.6 Query 7

A sétima *query* consiste em *"listar o top N aeroportos com a maior mediana de atrasos. Atrasos num aeroporto são calculados a partir da diferença entre a data estimada e a data real de partida, para voos com origem nesse aeroporto. O valor do atraso deverá ser apresentado em segundos."*. Em caso de empate, *"o nome do aeroporto deverá ser usado como critério de desempate (de forma crescente)"*.

Começamos por colocar todas as letras dos nomes dos aeroportos em maiúsculas. Depois, preparamos a lista de delays para cada aeroporto, para posteriormente, ordenamos a lista por ordem crescente. Calcula-se então a mediana, e ordenamos a lista final de forma decrescente com base nas medianas, e finalizamos imprimindo o resultado.

2.5.7 Query 8

A oitava *query* consiste em *"apresentar a receita total de um hotel entre duas datas (inclusive), a partir do seu identificador. As receitas de um hotel devem considerar apenas o preço por noite (price_per_night) de todas as reservas com noites entre as duas datas"*.

Semelhantemente à *queries* até então implementadas, começamos por armazenar todas as *reservations* de um dado *hotel_id* numa lista ligada. Após isso, percorremos a lista, e, para cada uma das reservas, se estiver entre as datas passadas como argumento da query, calculamos a receita dessa reserva e somamos a uma variável que guarda a receita total. No final, escrevemos no ficheiro de *output* a receita total.

2.5.8 Query 9

A nona *query* consiste em *"listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome (de forma crescente)."*Caso dois utilizadores tenham o mesmo nome, deverá ser usado o seu identificador como critério de desempate (de forma crescente).

Na presente consulta, a dificuldade surgiu na interpretação do *input*, que poderia ser do tipo "João"ou algo como "João M". Para lidar com ambos os casos, utilizamos a função

strtok para isolar a primeira ocorrência de , removemos a primeira instância de "\n"e as aspas, obtendo assim o prefixo completo a ser utilizado. Após obtermos o prefixo, verificamos, para cada usuário, se o seu *"account_status"* não é *"inactive"* e, em seguida, verificamos a sua existência no campo *name* do *user* utilizando a função *strncmp*. O resultado é então impresso no respetivo ficheiro de *output*.

3 Dificuldades sentidas

Consideramos que devíamos ter tido mais em atenção a questão do encapsulamento, apesar de, nesta primeira fase, o principal elemento de avaliação ser a validação do *dataset* e a implemetação do modo *batch*, temos noção que esta questão é de grande relevância, não só para o nosso presente, como estudantes de Engenharia Informática, mas especialmente para o nosso futuro, no mercado de trabalho. Devemos trabalhar a nossa aplicação nesse sentido, na próxima fase. Consideramos, ainda, que podiamos ter trabalhado no projeto de forma mais regular, com o objetivo de termos uma semana de entrega do projeto mais descansada, no entanto, o facto de sermos apenas dois elementos a trabalhar, dificultou-nos um bocado a vida nesse aspeto, sendo que os grupos formados foram de três elementos.

4 Conclusão

Resumindo, apesar das dificuldades sentidas, consideramos que conseguimos atingir os principais objetivos desta primeira fase do trabalho. Estamos com grande expectativa para a segunda fase, onde iremos melhorar o código até então produzido, implementar as queries que faltam, assim como os testes e o modo interativo, tendo mais em atenção o encapsulamento e o desempenho da nossa aplicação.