Universidade do Minho
Escola de Engenharia

Relatório - Fase 1
Laboratórios de Informática III

Martim Redondo
A100664

Rodrigo Castro
A100694

Tiago Moreira
A100541

18 de novembro, 2023 - Grupo 95

1 Introdução

No contexto da unidade curricular de Laboratórios de informática III foi-nos pedido a realização de um projeto focado na Linguagem C e em Engenharia de Software, nomeadamente, modularidade e encapsulamento, estruturas dinâmicas de dados, validação funcional e medição de desempenho

2 Modelação

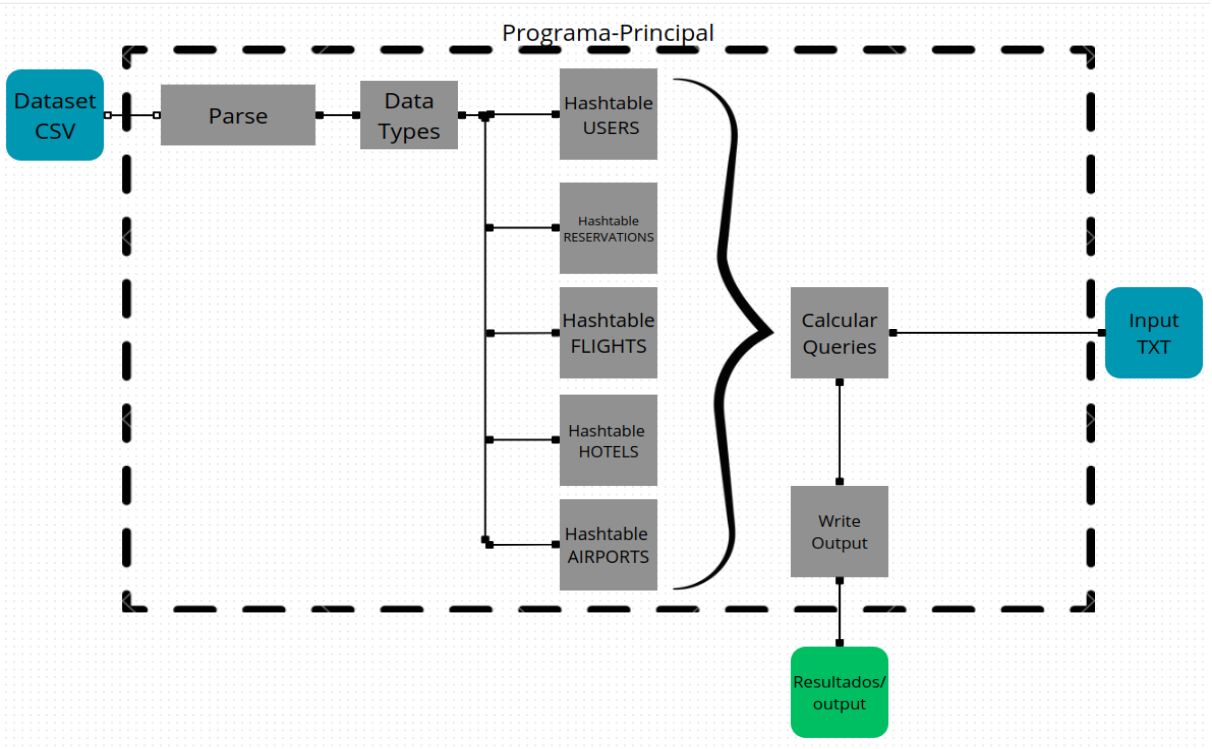
2.1 Arquitetura de Aplicação

A arquitetura do nosso projeto é constituída principalmente por 3 ficheiros principais, sendo eles:

- **Parse**, responsável tanto pela leitura do ficheiro como do tratamento de dados e a sua inserção de forma válida nas HashTables. Este ficheiro utiliza alguns outros ficheiros auxiliares (parseFicheiroFlights, parseFicheiroPassagers, parseFicheiroReservations, parseFicheiroUsers), estes ainda têm acesso aos ficheiros das HashTables específicas (exemplo: hotel).

- **CalcularQueries**, responsável por ler o input do utilizador e perceber o que o programa deve fazer com aquela informação.

- **WriteOutPut** é a função responsável por executar a query certa (tem acesso a todas as queries implementadas).



2.2 Organização dos Dados

2.2.1 Estrutura de Dados

Todas as nossas estruturas de dados principais são HashTables, todas estas foram guardadas em ficheiros parse específicos (exemplo: parseFicheiroFlights).

Ao todo foram criadas 5 HashTables, sendo elas, **HashTable users**, **HashTalbe voos**, **HashTable reservations**, **HashTable tablehotel** e **HashTable aeroportos**.

Foras as HashTables existem, também, as subestruturas que compõem as estruturas referidas em cima. Passo a citá-las:

-**Users**, guarda os seguintes dados de um user: id, nome, email, telemóvel, data de nascimento, sexo, número de passaporte, código do País de origem, data da criação da conta e status da conta (ativa/inativa).Guarda também uma lista de voos em que este já fez parte e também das reservas.

-**Flights**, guarda os seguintes dados de um voo: id, companhia aérea, modelo do avião, número de lugares, origem, destino, data marcada de saída, data marcada de chegada, data real de saída, data real de chegada, nome do piloto, nome do copiloto e número de passageiros, que é incrementado cada vez que há uma entrada válida no passengers.csv com este voo.

-**Reservation**, guarda os seguintes dados de uma reserva: id da reserva, id do user, id do hotel, nome do hotel, estrelas do hotel, imposto da cidade, data de início da estadia, data de fim da estadia, preço por noite, inclui pequeno-almoço, e avalição do user.

-**Hotel**, guarda todos os dados referentes aos hotéis: id do hotel, nome do hotel, preço por noite, imposto da cidade, número de avaliações, avaliação total, e uma lista de todas as suas reservas.

-**Aeroporto**, guarda todos os dados referentes aos aeroportos: nome do aeroporto, lista dos voos atrasados e lista de voos.

2.2.2 Motivo de usarmos HashTables

A decisão de usar HashTables para a organização da informação dos .csv foi, maioritariamente, devido ao facto de ser uma estrutura que tem capacidade de guardar grandes quantidades de dados (sem precisar de muito tempo para os aceder), o que nos deu grande vantagem ao longo da implementação da queries, visto que usamos os dados da HashTable de forma recorrente. Além disso é de fácil implementação e uso, sendo que só precisa da KEY para termos acesso a toda a informação.

3 Implementação

3.1 Nota

O input de todas as queries tem duas formas de ser impreso, de uma maneira formata, que será checada no WriteOutPut.c, caso o a[1] == 'F' e de uma maneira não formatada caso o a[1]!='F'.

3.2 Query1

A query1 pode ser chamada com uma das 3 seguintes HashTables, users, voos e reservations dependendo do id dado no input. Se função WriteOutPut verificar que o id é referente a um user, será executada a WriteOutPutQuery1user, se o id é referente a um voo, será executada a WriteOutPutQuery1voo e se o id é referente a uma reservation, será executada a WriteOutPutQuery1reservation.

Cada função fará a procura da informção nas respetivas HashTables e imprimirá o resultado.

3.3 Query3

Esta query recebe a HashTable de hoteis e a partir do id do hotel vai à HashTable e retira o hotel correto. Esta query foi resolvida de forma simples, pois a estrutura de dados está organizada de uma maneira que facilita todo o processo (cada hotel guarda a informação do número de avaliações e a avaliação total, logo é só necessário fazer uma divisão). Normalmente, seria preciso procurar pela lista de reservations onde o id do hotel é igual ao input e ir somando e contando quantas avalições são no total, para no final se fazer a média.

3.4 Query4

Esta query recebe a HashTable dos hoteis. Primeiramente, o programa vai buscar todas as informações do ID do hotel dado no input à HashTable, organiza-se essas informações tendo em conta os parâmetros do enunciado, primeiro a data de início da reserva (mais recente para a mais antiga), numa segunda instãncia, caso as datas sejam iguais, desempatar com o ID da reserva.

3.5 Query5

Esta query recebe sempre a HashTable de aeroportos. Primeiramente, o programa usa a informação que já está disponibilizada na struct do aeroporto e cria uma lista de voos, esta lista é depois ordenada por data de voo mais antigo para voo mais recente.

3.6 Query8

Esta query recebe, exclusivamente, a HashTable hotel. O programa obtem o hotel correto, através de um look up, e como a struct hotel guarda a lista de reservas, este só precisa de correr cada reserva e ir adicionando a receita de todas elas, o número de noites é calculado a partir das datas especificadas.

3.7 Query9

Esta query recebe a HashTable users e vai guardar numa Glist todos os users que tenham o prefixo dado no input, depois disso organiza a Glist por nome de user, e se preciso de desempatar por id.

4 Desempenho

Tendo em conta, unicamente, à primeira fase, o programa corre, nos testes da plataforma de LI3, em 0.8 segundos com margem de erro de 0.5 segundos.

5 Memory Leaks

Tendo em conta, unicamente, à primeira fase, o programa, de acordo com os testes da plataforma de LI3, contém, apenas, 0.01884 MB de memory leak, visto ser o espaço ocupado pela biblioteca GLIB-2.0, sendo impossível reduzir mais o número de MB de memory leak.

6 Conclusão

De modo geral a primeira fase correu bem, contudo temos a noção de que temos de alterar a estrutura de dados para a segunda fase, de modo que as queries restantes sejam rápidas e eficientes ao executar.

No nosso ponto de vista, o que deu mais trabalho e foi mais custoso foi conseguir alcançar 0 bytes de memory leak e ao mesmo tempo ter um rápido tempo de execução.