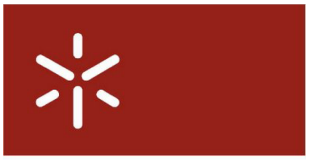
Universidade do Minho

Departamento de Informática



Universidade do Minho

Laboratórios de Informática III

Sistema de Gestão e Consulta de Voos e Reservas

Grupo 62 - Fase 2

Henrique Pereira

Mariana Morais

Tomás Valente

Henrique Morais Pereira A100831
Mariana Filipa Morais Gonçalves A100662
João Tomás Gonçalves de Sousa Carneiro Valente A100540

Janeiro de 2023

Conteúdo

1 Introdução

Este projeto foi apresentado no presente ano letivo no âmbito da unidade curricular de Laboratórios de Informática III do curso de Licenciatura em Engenharia Informática, visando consolidar os conteúdos teóricos e práticos, enriquecendo os conhecimentos adquiridos nas UCs de Programação Imperativa e de Algoritmos e Complexidade.

Numa fase inicial do projeto tivemos a oportunidade de desenvolver um projeto em larga escala para criar uma aplicação que simule um sistema de Gestão e Consulta de voos e reservas, tentando manter sempre a proteção de dados através do encapsulamento dos mesmos e a utilização eficiente de memória. Nesta fase o nosso foco foi criar as estruturas onde iríamos armazenar os nossos dados de forma a manipular posteriormente os mesmos desejados dependendo das diferentes necessidades que iriam surgir.

Nesta segunda fase, após a discussão com os docentes, foi nos encaminhado para também trabalhar sobre a modularidade (separação do código coerentemente), um código reutilizável e uma aplicação de estruturas adequadas para a manipulação dos dados. Assim o fizemos tal como já tínhamos em mente, assim como a implementação de encapsulamento, mas agora de forma mais clara. Posto isto, foram adicionadas mais algumas funcionalidades.

Contudo, é importante a nosso ver referir que não foi possível reestruturar o nosso trabalho com estruturas mais eficientes e capazes de lidar com *datasets* de maior escala, devido à falta de tempo proveniente de uma má gestão inicial de tempo.

2 Esquema Do Projeto

Sendo uma aplicação de um projeto em larga escala, a aplicação aceita a entrada de quatro ficheiros de texto: um ficheiro com uma listagem de utilizadores, um ficheiro com uma listagem de voos, um ficheiro com uma listagem das reservas e ainda um ficheiro com a listagem dos passageiros; Implementou-se uma estrutura semelhante para o tratamento de dados de todos os ficheiros, uma vez que a catalogação se divide em quatro catálogos distintos.

No entanto, de modo mais específico, com o intuito de otimizar o trabalho prático, decidimos implementar *HashTables* de *structs*.

Esta decisão foi tomada pelo grupo com base nas seguintes características:

- A capacidade de armazenar um alto número de elementos
- Rápida procura
- Facilidade na remoção de elementos
- Eficiência na inserção de elementos

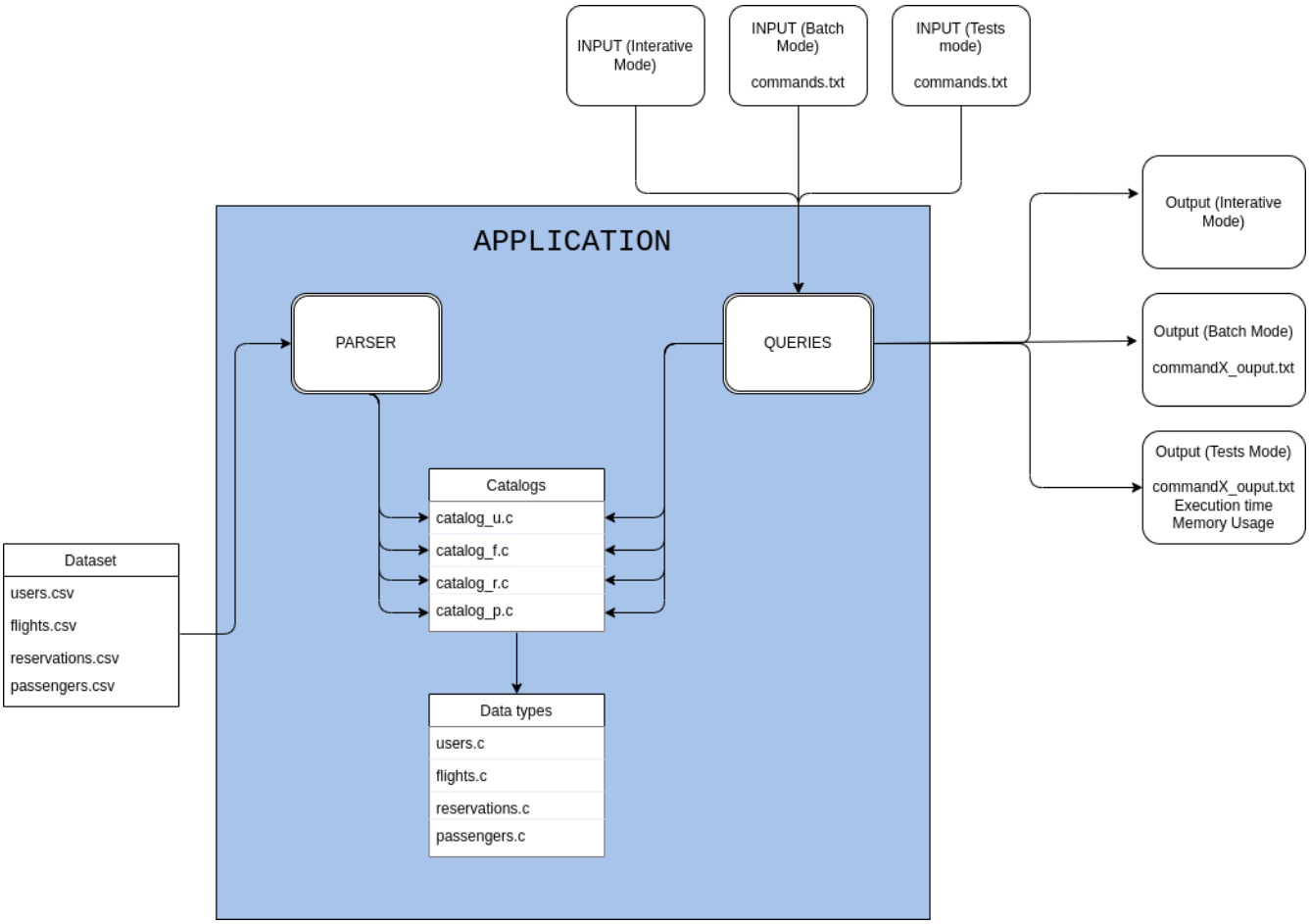
Devemos também referir o uso da Glist uma vez que mantém a ordem dos elementos, algo que nos pareceu bastante necessário e útil nas *queries* apresentadas posteriormente.

2.1 Estrutura de dados

Os dados para a elaboração do trabalho prático estão divididos em quatro ficheiros. Sendo eles:

- **flights.csv**, com voos onde cada linha com a seguinte informação: identificador do voo; companhia aérea; modelo do avião; número de lugares totais disponíveis; aeroporto de origem; aeroporto de destino; data e hora estimada de partida; data e hora estimada de chegada; data e hora real de partida; data e hora real de chegada; nome do piloto; nome do copiloto e observações sobre o voo.
- **passengers.csv**, onde apenas temos a informação do identificador do voo e do identificador do utilizador de cada passageiro.
- **reservations.csv**, sobre as reservas temos a seguinte informação: identificador da reserva; identificador do utilizador; identificador do hotel; nome do hotel; número de estrelas do hotel; percentagem do imposto da cidade (sobre o valor total); morada do hotel; data de início; data de fim; preço por noite; se a reserva inclui pequeno-almoço; detalhes sobre o quarto; classificação atribuída pelo utilizador e comentário sobre a reserva.
- **users.csv**, cada utilizador tem a si associado os seguintes dados: identificador do utilizador; nome; email; número de telemóvel; data de nascimento; sexo; número do passaporte; código do país de residência; morada; data de criação da conta; método de pagamento e estado da conta.

Após analisar os dados que tínhamos para tratar e o que nos era pedido, pensamos no modo de implementação. Posto isto, temos aqui o esquema ilustrativo da nossa aplicação que é o motor e a referência para o desenvolvimento do projeto pretendido.



Esquema ilustrativo da nossa aplicação

3 Estrutura Do Projeto

3.1 Catálogo dos Voos

De modo a tratar os dados que desejamos, criarmos um catálogo para os Voos. É neste módulo que os dados do ficheiro flights.csv são guardados. Os dados são guardados em *HashTables*, onde é usado o ID do Voo como chave da mesma. Tendo isto em conta, é realizado o *parse* do .csv onde abrimos o arquivo, cada linha do arquivo é processada e se válida é inserida na *HashTable* usando o ID do voo como chave. Caso contrário segue para o ficheiro .csv de erros correspondentes.

3.2 Catálogo dos Passageiros

Neste módulo que os dados do ficheiro passengers.csv são guardados. Os dados são guardados em *HashTables*, onde é usado o ID do passageiro como chave da mesma. Tendo isto em conta, é realizado o parse do .csv onde abrimos o arquivo, cada linha do arquivo é processada e se válida é inserida na *HashTable* usando o ID do utilizador e do voo como chave. Caso contrário segue para o ficheiro .csv de erros correspondentes.

3.3 Catálogo das Reservas

Neste módulo que os dados do ficheiro reservations.csv são guardados. Os dados são guardados em *HashTables*, onde é usado o ID da Reserva como chave da mesma. Tendo isto em conta, é realizado o parse do .csv onde abrimos o arquivo, cada linha do arquivo é processada e se válida é inserida na *HashTable* usando o ID da reserva como chave. Caso contrário segue para o ficheiro .csv de erros correspondentes.

3.4 Catálogo dos Utilizadores

Neste módulo que os dados do ficheiro users.csv são guardados. Os dados são guardados em *HashTables*, onde é usado o ID do Utilizador como chave da mesma. Tendo isto em conta, é realizado o parse do .csv onde abrimos o arquivo, cada linha do arquivo é processada e se válida é inserida na *HashTable* usando o ID do utilizador como chave. Caso contrário segue para o ficheiro .csv de erros correspondentes.

4 Módulos

Um projeto desta envergadura exige diversos módulos de modo a estruturar e proteger os dados. Uma vez que um dos objetivos é a aplicação de encapsulamento e modulação, o projeto está implementado em vários destes. Seguimos com uma breve elucidação.

- ***catalog_f.h*** Módulo que guarda os dados associados aos voos.
- ***catalog_p.h*** Módulo que guarda os dados associados aos passageiros.
- ***catalog_r.h*** Módulo que guarda os dados associados às viagens.
- ***catalog_u.h*** Módulo que guarda os dados associados aos utilizadores.
- ***date.h*** Módulo para o manuseamento das datas e horas. É definida uma estrutura de modo a conseguir tratarmos dos dados desejados. E após isto são feitas manipulações de modo a comparar e calcular diferenças entre datas. É também aqui que validamos uma data ou não.
- ***executor.h*** Módulo que optámos por utilizar que para além do modo *batch* vai também tratar do modo iterativo dando início à realização das *queries*.
- ***flights.h*** Neste módulo, iremos tratar dos voos. É criada uma estrutura de modo a tratarmos os dados da forma mais conveniente que encontramos. Nesta estrutura são guardados os dados dos mesmos para poderem ser usados posteriormente conforme as necessidades que encontramos.
- ***passengers.h*** Este módulo, semelhante ao anterior, é também elaborado com o pensamento e necessidade de tratar dos dados dos passageiros.

- ***reservations.h*** A partir das reservas, também é necessário aceder e manipular os seus dados de forma a responder ao pedido. Temos também uma estrutura com todos os dados associados às reservas. Neste módulo já temos algum tratamento de dados, tais como a comparação das datas de duas reservas e a ordenação de reservas com base na data do seu início.
- ***users.h*** Como nos anteriores, será criada a estrutura que permite ter todos os dados associados, mas, neste caso, aos utilizadores. É também validada alguma informação pertinente.
- ***queries.h*** Neste módulo, já entramos no desenvolvimento do que fizemos anteriormente. Iremos manipular os nossos catálogos de acordo aos requisitos pretendidos em cada consulta.
- ***validacao.h*** Este breve módulo garante que certos valores do sistema não sejam zero conforme o que o programa necessita.
- ***menus.h*** Será o módulo que apenas nos permite utilizar o nosso menu inicial do modo interativo.
- ***page.h*** Módulo responsável pela paginação no modo interativo.
- ***output.h*** Módulo que trata dos *outputs* que vão sendo gerados.
- ***parser.h*** Módulo responsável pelo *parser* dos csv's dados.
- ***tests_funcs.h*** Módulo responsável por calcular o tempo e a memória do programa. Compara também as nossas linhas de *output* com os *outputs* dados pelos docentes. É também aqui que calculamos o tempo de execução das *queries*.
- ***view.h*** Módulo responsável pelos menus que nos são apresentados no terminal.

5 Queries

5.1 QUERY 1

A *query* 1 pretende listar, a partir do seu identificador, o resumo de um utilizador, de um voo ou de uma reserva.

Primeiramente, a função tenta encontrar um utilizador com o ID fornecido, se não encontrar o utilizador, tenta encontrar um voo com o mesmo ID, se também não encontrar o voo, tenta encontrar uma reserva com o mesmo ID.

Se um utilizador for encontrado, verifica se a conta do utilizador está ativa, se a conta estiver inativa, escreve uma *string* vazia no arquivo. Caso contrário, obtém informações adicionais, como o número de voos, o número de reservas e o total gasto, e converte essas informações numa *string*.

5.2 QUERY 2

A *query* 2 pretende listar os voos ou reservas de um utilizador, dependendo se o segundo argumento for "flights"ou "reservations"respetivamente. Caso este seja uma string vazia, apresenta tanto os voos quanto as reservas.

Inicialmente é realizado uma verificação do *input* relativo ao campo *type*. De seguida, é verificado se o utilizador existe e se a conta associada ao mesmo está ativa.
No caso do *type* ser "reservations"ou string vazia, itera a *HashTable* das reservas, e para cada reserva associada ao utilizador com o ID fornecido, insere numa *HashTable* (infoHash), temporariamente, o ID da reserva e a data de início associada à mesma.

Se o *type* for "flights"ou string vazia, itera a *HashTable* dos voos, e para cada voo associada ao utilizador com o ID fornecido, insere numa *HashTable* (novamente a infoHash) o ID do voo e a data prevista de partida. Também é adicionado a uma lista o ID do voo em questão.

Por último, os resultados obtidos são ordenados por data, da mais recente para a mais antiga.

5.3 QUERY 3

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

Para tal iremos iterar a *HashTable* das reservas e calculamos a soma dos *ratings* de todas as reservas que possuam o ID do Hotel em análise, bem como a quantidade dessas mesmas reservas.

Por último iremos proceder ao cálculo da classificação média dividindo o valor da soma dos *ratings* pelo número total de reservas previamente calculados.

5.4 QUERY 4

A *query* 4 pretende listar as reservas de um hotel, ordenadas por data de início, com um critério de desempate baseado no identificador da reserva.

Começamos por iterar a *HashTable* das reservas, e para cada reserva associada ao ID do Hotel em análise, guardamos numa *HasTable* (auxHash) o ID da reserva associado à *struct* da reserva em questão.

Após isto iremos ordenar as reservas da data mais recente para a mais antiga, destacando o facto que em caso de empate, é utilizado o identificador da reserva.

Aquando da preparação do *output* será calculado o preço total de cada reserva.

5.5 QUERY 5

A *query* 5 consiste em listar os voos com origem num dado aeroporto, entre duas datas, ordenados por data estimada de partida, da mais recente para a mais antiga. Em caso de empate, o identificador do voo deverá ser usado como critério de desempate (de forma crescente).

Começamos por iterar a *HashTable* dos voos, e para cada voo associado ao aeroporto de origem em análise se a data estimada estiver entre as datas fornecidas é adicionado à *HashTable* auxHash o id do voo associado à *struct* do voo em questão. Posteriormente, os voos serão ordenados pela sua data estimada de partida como previmos anteriormente utilizando o método de desempate quando necessário.

5.6 QUERY 6

A *query* 6 visa listar o top N aeroportos com mais passageiros, para um dado ano, sendo apenas considerados os voos com a data estimada de partida do ano fornecido. O critério de desempate será o nome do aeroporto.

Para tal é feita uma iteração sobre os voos, verificando se o ano de partida iguala o ano fornecido e armazena a contagem de passageiros desse voo no aeroporto de partida e de chegada numa *HashTable* auxiliar (auxHash) que possui a contagem total de passageiros para cada aeroporto.

Posteriormente os aeroportos são ordenados pela sua respetiva contagem de passageiros utilizando o critério de desempate caso necessário. Desta ordenação são selecionados os N menores elementos.

5.7 QUERY 7

A *query* 7 pretende listar o top N aeroportos com a maior mediana de atrasos, em segundos. Em caso de empate, o nome do aeroporto deverá ser usado como critério de desempate.

Começamos por iterar a *HashTable* dos voos e calcular o atraso de partida do voo, armazenando esse valor na lista presente na *HashTable*(auxHAsh) que possui o ID de aeroportos de origem associados a uma lista de atrasos do mesmo.

Após obter a informação de todos os voos calcula a mediana de cada aeroporto, ordenando a lista armazenada na HashTble(auxHAsh) e obtendo a mediana. Este novo valor é inserido numa nova *HashTable* (topHash) onde a *Key* é o ID do aeroporto.

De forma semelhante à *query* 6, os aeroportos serão ordenados conforme os critérios definidos. Desta forma, será possível retirar o TopN desejado.

5.8 QUERY 8

Esta *query* pretende apresentar a receita total de um hotel entre duas datas, a partir do seu identificador.
Começamos por iterar a *HashTable* das reservas e caso a reserva seja relativa ao ID do Hotel em análise efetua-se o cálculo do número de noites presentes no período em análise. No caso desse valor ser positivo, ao valor da receita será adicionado o número de noites multiplicado pelo preço por noite do respetivo hotel.

5.9 QUERY 9

Esta *query* pretende listar todos os utilizadores cujos nomes começam com o prefixo passado por argumento, ordenados por nome (de forma crescente). Em caso de empate deverá ser usado o seu identificador como critério.
Começamos por iterar a *HashTable* dos utilizadores e caso o utilizador tenha a sua conta ativa começamos por ir buscar o prefixo do nome de cada utilizador com o tamanho igual ao do prefixo em análise.
Caso os dois prefixos coincidam armazenamos o ID do utilizador associado ao nome do mesmo na *HashTable* (auxHAsh).
As informações presentes na *HashTable* auxHash são ordenadas segundo os critérios definidos pelo enunciado. Tal como sugerido pela equipa docente, de modo a obter a ordem correta dos resultados, foi utilizada o 'strcoll' para comparar tanto os nomes como os ID's dos utilizadores. Foi também aplicado o critério de desempate quando necessário.

5.10 QUERY 10

A *query* 10 procura apresentar várias métricas gerais da aplicação, tais como o número de novos utilizadores registados, o número de voos, o número de passageiros, o número de passageiros únicos e o número de reservas.

De modo a conseguir responder a todas as métricas pedidas, foi necessário criar uma *struct* que possui informações sobre dados estatísticos, fazendo parte dos valores armazenados pela *HashTable* agregatorHash.

Primeiramente é verificado qual o tipo de pesquisa a realizar, dependendo dos argumentos fornecidos para execução da *query*. Começamos por iterar a *HashTable* dos utilizadores para obter o número de utilizadores registados no ano/mês/dia em análise. De seguida iteramos a *HashTable* dos voos para obter os voos com a data prevista de partida no ano/mês/dia em análise. Posteriormente, iteramos a *HashTable* dos passageiros, e para cada um deles vamos buscar a data de partida do voo correspondente, para obter os passageiros do ano/mês/dia em análise. Ao longo da iteração é mantida uma lista de passageiros já analisados, com o seu respetivo ID, de modo a ser possível obter também o número de passageiros únicos. Por fim, iteramos a *HashTable* das reservas para obter o número de reservas no ano/mês/dia a analisar.

Agora ordenamos toda a informação obtida consoante o indicador do ano/mês/dia armazenado na *HashTable* agregatorHash.

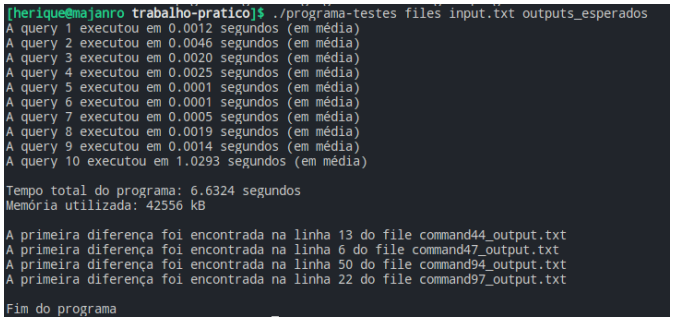
5.11 Output Queries

De modo a utilizar o módulo de *output*, cada *query* quando possui informação em vez de imprimir para o terminal ou escrever para um ficheiro, armazena numa *string* o conteúdo de cada uma das entradas, colocando essa *string* num *array* de *strings*. No fim da execução, cada *query* retorna o *array* de *strings*, e as funções fora das *queries* têm de utilizar essa informação gerada passando-a às respetivas funções do módulo *output* que imprimem a informação para o terminal ou inserem a mesma num ficheiro.

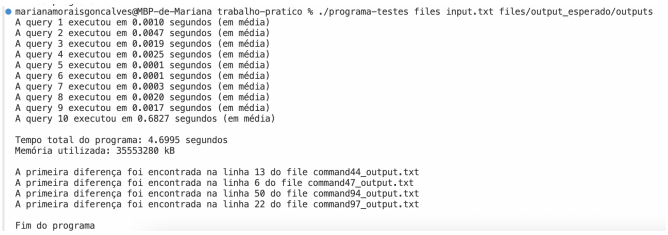
5.12 Testes

Foi feita uma análise do desempenho do programa através da criação dos métodos sugeridos no enunciado que calculam o tempo de CPU, assim como a memória utilizada pelo programa. Para além disto foi criado também um método para o cálculo do tempo de execução das *queries* implementadas, sendo que este determina o momento em que a *query* é iniciada e o momento em que a sua execução termina, devolvendo assim o tempo decorrido. Caso uma lista de comandos não necessite de alguma das *queries* implementadas, o algoritmo indica que a *query* em questão não foi executada. Foi implementado ainda um método que compara as linhas do ficheiro de *output* com as dos *datasets* fornecidos pela equipa docente, devolvendo a primeira linha divergente que existir e o ficheiro em que esta se encontra.

Em baixo segue o teste feito na máquina 1 com base no *input* e outputs esperados da fase 1.



Testes funcionais: Máquina 1



Testes funcionais: Máquina 2

Das figuras anteriores podemos concluir que a máquina utilizada tem influência no tempo de execução do programa, este que é significativo devido à execução da *query* 10.

6 Conclusão

De modo a concluir esta segunda fase do projeto, o qual foi bastante complexo e desafiador, no entanto, também serviu como um bom mecanismo de evolução e consolidação da teoria abordada por outras unidades curriculares. Tem sido também uma boa motivação para a busca de novos conceitos que até então não tinham sido despertados totalmente ou da melhor maneira.

Houve uma grande dificuldade na validação, o que gerou alguns conflitos estruturais. A legibilidade e a indentação do código foi algo que acabou por não ser a melhor, causando também algumas complicações na comunicação e trabalho de grupo. Além disso, a implementação de *queries* foi algo que também exigiu muito de nós e que mesmo assim não correspondeu totalmente aos objetivos previstos, pois infelizmente não reestruturamos as nossas estruturas de armazenamento para que fosse possível escalar, sendo possível responder de forma positiva ao *datasets* maiores.

Contudo, de forma geral, fomos conseguindo realizar o trabalho conscientes que não satisfaz na totalidade todas as condições requeridas, mas apresenta um balanço positivo.