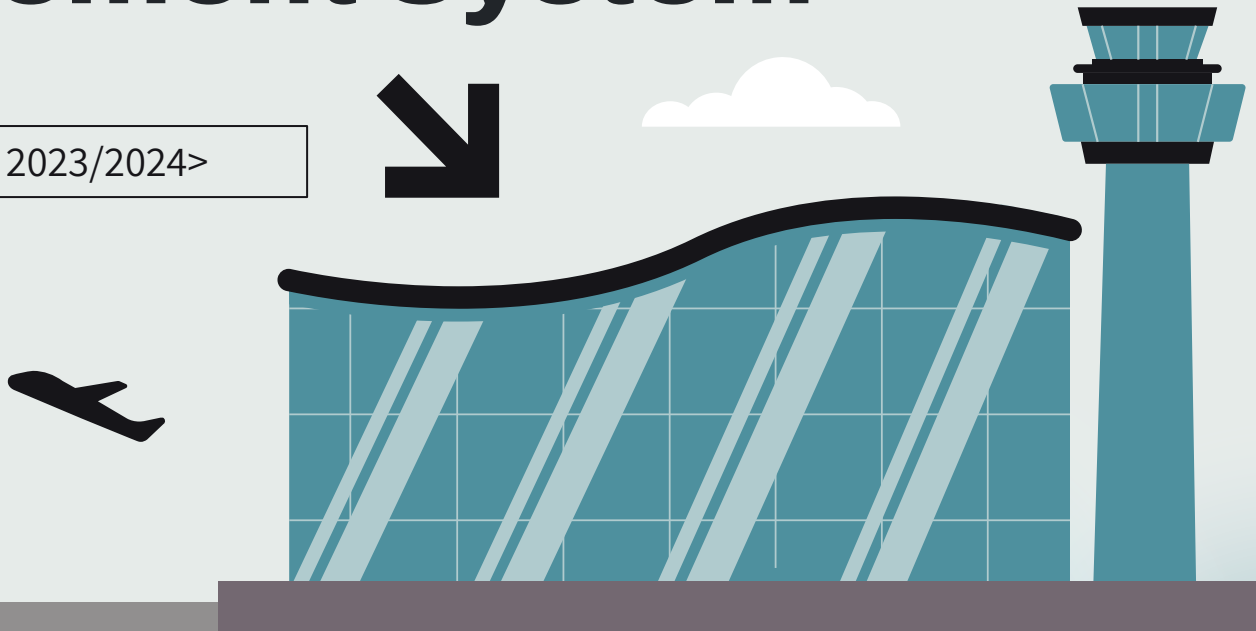
Air Travel Flight Management System

<Projeto 2 de AED 2023/2024>



Elementos do Grupo

Diogo Ferreira

up202205295

Gabriel Carvalho

up202208939

Guilherme Silva

up202205298

Classes

Airline

Airline.cpp
Airline.hpp

Airport

Airport.cpp
Airport.hpp

Graph

Graph.cpp
Graph.hpp
flights.csv

Menu

Menu.cpp
Menu.hpp

AllAirline

AllAirline.cpp
AllAirline.hpp
airlines.csv

AllAirport

AllAirport.cpp
AllAirport.hpp
airports.csv

Leitura do dataset

Para este projeto foram-nos fornecidos 3 ficheiros, com as informações necessárias para a implementação das funcionalidades, airports.csv, airlines.csv e flights.csv, sendo o seu conteúdo referente aos aeroportos, às companhias aéreas e aos voos, respetivamente. De modo a aceder ao conteúdo de cada ficheiro, permitindo-nos armazená-lo, procedemos à sua leitura com o recurso às seguintes funções:

1. Para o ficheiro airports.csv, utilizamos o construtor AllAirports(const string& file), presente na classe AllAirports, para aceder às informações referentes aos aeroportos, nomeadamente o seu código, o seu nome, bem como a sua localização geográfica (cidade, país e coordenadas);

2. Para o ficheiro airlines.csv, utilizamos o construtor AllAirlines(const string& file), presente na classe AllAirlines, para aceder às informações referentes às companhias aéreas, nomeadamente o seu código, o seu nome, o seu *callsign* e o seu país de origem;

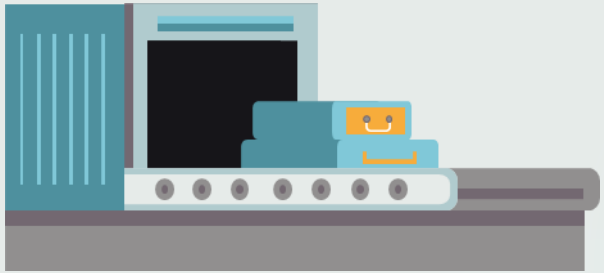
3. Para o ficheiro flights.csv, utilizamos o construtor Graph(const string& file), presente na classe Graph, para aceder às informações referentes aos voos, nomeadamente os aeroportos de partida e de destino, bem como a companhia aérea responsável pelo mesmo;

Descrição do Grafo

Para uma melhor manipulação dos dados fornecidos, recorremos a um grafo, com estrutura similar à utilizada nas aulas práticas em que:

- Os vértices do grafo representam os aeroportos;
- As arestas representam os voos;
- E o grafo em si representa a *Flight Network;*

De seguida, encontram-se excertos da implementação de todas as classes que formam o grafo.



Implementação do grafo em Graph.hpp

```cpp
class Vertex { // Each Vertex represents an Airport
private:
    string airportCode;
    vector<Edge> adj;
    bool visited;
    bool processing;
    int num;
    int low;

    void addEdge(Vertex *dest, const string &airline_code);

    bool removeEdgeTo(Vertex *dest);

public:
    Vertex(const string &airport_code);
// ...
};
class Edge { // Each Edge represents a flight
private:
    Vertex *dest;
    string airlineCode;

public:
    Edge(Vertex *dest, string airline_code);
// ...
};
```

```cpp
class Graph { // Graph represents the flight network
private:
    vector<Vertex *> airportSet;

    // Auxiliary functions for implementation of functionalities
    void dfsVisit(Vertex *v, vector<string> &res) const;

    vector<string> bfsWithStops(const string &source, int numLayovers) const;

    map<string, int> airportTrafficCapacity() const;

    void dfsForArtPoints(Vertex *v, unordered_set<string> &artPoints, int &i, const string &ogVertContent) const;

    vector<vector<string>> interpretInputs(const vector<string> &inputs, const AllAirports &allAirports) const;
    vector<string> interpretCoords(const string &input, const AllAirports &allAirports) const;
    double haversineFormula(double lat1, double lon1, double lat2, double lon2) const;
    string interpretAirportName(const string &input, const AllAirports &allAirports) const;
    vector<string> interpretCityName(const string &input, const AllAirports &allAirports) const;

    bool inputsValid(const vector<string> &normSource, const vector<string> &normDest) const;

    set<string> findAirlinesToFilter(const string &airlines) const;

    vector<vector<string>> findAllShortestPaths(const vector<string> &normSource, const vector<string> &normDest,
                                                int airlineFilter, const set<string> &airlinesToFilter) const;
    vector<vector<string>> bfsForAllShortestPaths(const string &source, const string &dest, int airlineFilter,
                                                  set<string> airlinesToFilter) const;
    void dfsBackwards(const string &vert, unordered_map<string, set<string>> &parentsOfVert,
                      vector<vector<string>> &res, list <string> &path) const;

    bool existPaths(const vector<vector<string>> &paths) const;

    vector<pair<TYPE_PATH, TYPE_MinAIRLINES>> findLeastAirlinesPerPath(const vector<vector<string>> &paths) const;

    vector<vector<string>> airlineFinder(const vector<string> &path) const;
    vector<vector<string>> getAllCombinations(const vector<vector<string>> &airlines) const;
    void generateCombinations(const vector<vector<string>> &airlines, vector<string> &current,
                              vector<vector<string>> &result, int depth) const;
    vector<vector<string>> getLeastAirlines(const vector<vector<string>> &combinations) const;

    void filter3Printer(int minNumAirlines, const vector<pair<TYPE_PATH, TYPE_MinAIRLINES>> &res,
                        const AllAirports &allAirports) const;
    void bestOptionPrinter(const vector<vector<string>> &paths, const AllAirports &allAirports) const;

public:
    Graph(const string& file);
// ...
};
```

Utilização de Hash Tables

Para atingir uma pesquisa em tempo constante - O(1) - para companhias aéreas, aeroportos e cidades, utilizamos a classe *unordered_set* da STL, tal como se verifica nos ficheiros AllAirlines.hpp e AllAirports.hpp. A implementação da estruturas que armazenam todas as companhias aéreas e todos os aeroportos, respetivamente, encontram-se abaixo:

```cpp
struct airlineHash {
    size_t operator()(const Airline& a) const {
        const string& code = a.getCode();
        size_t hashVal = 5381;

        for (char c : code) {
            hashVal = 33*hashVal + static_cast<size_t>(c);
        }

        return hashVal;
    }
};

struct airlineEquality {
    bool operator()(const Airline& a1, const Airline& a2) const {
        return a1.getCode() == a2.getCode();
    }
};


class AllAirlines {
private:
    unordered_set<Airline, airlineHash, airlineEquality> airlines;
public:
// ...
```

```cpp
struct airportHash {
    size_t operator()(const Airport& a) const {
        const string& code = a.getCode();
        size_t hashVal = 5381;

        for (char c : code) {
            hashVal = 33*hashVal + static_cast<size_t>(c);
        }

        return hashVal;
    }
};

struct airportEquality {
    bool operator()(const Airport& a1, const Airport& a2) const {
        return a1.getCode() == a2.getCode();
    }
};


class AllAirports {
private:
    unordered_set<Airport, airportHash, airportEquality> airports;
public:
// ...
```

Funcionalidades Implementadas

- Número total de aeroportos;
  - Algoritmo associado:
    - AllAirports::numAirports(); //Complexidade: O(1);

```
Insert the number corresponding to the option you wish to select:
1. Total number of airports
```

- Número total de voos;
  - Algoritmo associado:
    - Graph::numTotalFlights(); // Complexidade: O(|V| + |E|), em que |V| consiste no número total de vértices do grafo e |E| representa o número total de arestas;

```
Insert the number corresponding to the option you wish to select:
2. Total number of available flights
```

- Número total de voos e de companhias aéreas a partir de um aeroporto;
  - Algoritmos associados:
    - Vertex::getAdj().size(); // Complexidade: O(1);
    - Vertex::numDiffAirlines(); // Complexidade: O(n) , em que n corresponde ao número de voos a partir de um aeroporto;

```
Insert the number corresponding to the option you wish to select:
3. Number of flights out of a specific airport
```

Funcionalidades Implementadas

- Número total de voos a partir de uma cidade;

```
Insert the number corresponding to the option you wish to select:
4. Number of flights per city
```

  - Algoritmos associados:
    - Graph::numFlightsPerCity(); // Complexidade: O(n * |V|), em que n corresponde ao número de aeroportos da cidade;
    - AllAirports::airportsPerCity(); // Complexidade: O(n), em que corresponde ao número de aeroportos da base de dados;

- Número total de voos efetuados por uma companhia aérea;

```
Insert the number corresponding to the option you wish to select:
5. Number of flights per airline
```

  - Algoritmo associado:
    - Graph::numFlightsPerAirline(); // Complexidade: O(|V| + |E|);

Funcionalidades Implementadas

- Número total de países diferentes, alcançáveis diretamente, a partir de um aeroporto específico;

```
Insert the number corresponding to the option you wish to select:
6. Number of different countries you can fly directly to from a specific airport
```

  - Algoritmo associado:
    - Graph::numCountriesPerAirportDirect(); // Complexidade: O(|V| + |E|)

- Número total de países diferentes, alcançáveis diretamente, a partir de uma cidade específica;

```
Insert the number corresponding to the option you wish to select:
7. Number of different countries you can fly directly to from a specific city
```

  - Algoritmo associado:
    - Graph::numCountriesPerCityDirect(); // Complexidade: O(|V| + |E|);

Funcionalidades Implementadas

```
Insert the number corresponding to the option you wish to select:
8. Number of direct and non-direct destinations from a specific airport
```

- Número total de destinos diferentes, alcançáveis a partir de um aeroporto específico;
    - Algoritmos associados (o fator dominante de todas as complexidades seguintes advém da função dfsVisit():
        - Graph::numAirportsPerAirport(); // Complexidade: O(|V| + |E|);
        - Graph::numCitiesPerAirport(); // Complexidade: O(|V| + |E|)
        - Graph::numCountriesPerAirport(); // Complexidade: O(|V| + |E|)
        - Graph::dfs(const string & source); // Complexidade: O(|V| + |E|)
        - Graph::dfsVisit(); // Complexidade: O(|V| + |E|)
    - O fator dominante de todas as complexidades seguintes advém da função bfsWithStops():
        - Graph::numAirportsX(); // Complexidade: O(|V| + v2 + e2)
        - Graph::numCitiesX(); // Complexidade: O(|V| + v2 + e2);
        - Graph::numCountriesX(); // Complexidade: O(|V| + v2 + e2);
        - Graph::bfsWithStops(); // Complexidade: O(|V| + v2 + e2), em que v2 corresponde a todos os vértices do grafo cuja profundidade <= numLayovers e e2 consiste em todas as arestas do grafo cuja profundidade <= numLayovers;

Funcionalidades Implementadas

- Voos com o maior número de escalas;

```
Insert the number corresponding to the option you wish to select:
9. Flights with the greatest number of stops in between
```

  - Algoritmo associado:
    - Graph::diameter(); //Time Complexity O(|V| * (|V| + |E|));

- Top X aeroportos com o maior número de voos;

```
Insert the number corresponding to the option you wish to select:
10. Top X airports with the greatest air traffic capacity
```

  - Algoritmos associados:
    - Graph::topXAirports(); //Time Complexity O(|V| + |E| + |V| log |V| + k), em que k corresponde ao número de aeroportos requisitados pelo utilizador;
    - Graph::airportTrafficCapacity(); //Time Complexity O(|V| + |E|);

- Aeroportos essenciais (pontos de articulação do grafo);

```
Insert the number corresponding to the option you wish to select:
11. List of essential airports
```

  - Algoritmo associado:
    - Graph::articulationPoints ();//Time Complexity O(|V| + |E|);
    - Graph::dfsForArtPoints ();//Time Complexity O(|V| + |E|);

Funcionalidades Implementadas

- Melhores opções de voo, i.e., voos com menor número de escalas, com opções de busca com filtros;
    - Algoritmo associado:
        - Graph::findBestOption();
        - Graph::interpretInputs();
        - Graph::inputsValid();
        - Graph::findAirlinesToFilter();
        - Graph::findAllShortestPaths();
        - Graph::existPaths();
        - Graph::findLeastAirlinesPerPath();
        - Graph::filter3Printer();
        - Graph::bestOptionPrinter();

```
Insert the number corresponding to the option you wish to select:
12. Best flight options
```

```
================================================================================
Best Flight Option(s) Filter Selector Menu
--------------------------------------------------------------------------------
Insert the number corresponding to the filter you wish to use for the search of
the best flight option:
1. Whitelist airlines ( use only the inputted airline(s) )
2. Blacklist airlines ( don't use the inputted airline(s) )
3. Minimize the amount of different airlines used
4. No filters
5. Back to Main Menu
================================================================================
Option:
```

NOTA:
Todas estas funções recorrem a diversas funções auxiliares, sendo a sua complexidade dependente delas. No entanto, esta encontra-se descrita em detalhe na documentação Doxygen.

Descrição da Interface com o Utilizador

A interface com o utilizador encontra-se implementada no ficheiros Menu.hpp e Menu.cpp.

A implementação utiliza o *State Design Pattern*, em que a classe Current (que corresponde ao "contexto" do *design pattern)*, armazena uma referência (neste caso, um *pointer,* na *member variable* currentMenu) da interface Menu (que corresponde ao "estado interface" do *design pattern*). Esta, por sua vez, declara os métodos a ser implementados por cada subclasse Menu (estas subclasses correspondem aos "estados em concreto" do *design pattern*). Através do conceito de herança, cada subclasse Menu implementa os métodos declarados pela interface, de modo a interagir com o utilizador e obter a informação necessária para continuar a execução do programa. Com a noção de polimorfismo, a classe Current, no seu método run(), chama os métodos print() e handleInput() da interface Menu, sendo, em *runtime*, chamadas as implementações desses mesmos métodos da subclasse Menu que se encontra em execução no momento.
A classe Current também contém as estruturas que armazenam os dados dos ficheiros .csv (que se encontram nas *member variables* allAirlines, allAirports e network).

Perante as opções escolhidas pelo utilizador, cada subclasse Menu:

- Imprime o seu conteúdo no terminal - função print() de cada subclasse;
- Realiza as operações necessárias sobre as estruturas de dados na classe Current (através de uma "referência de volta" para a classe Current - neste caso, um *pointer* - que a função handleInput() de cada subclasse recebe nos seus argumentos);
- Indica qual o Menu seguinte a ser executado (subclasse seguinte a que a *member variable* currentMenu da classe Current se irá referir), através da variável nextMenu - função handleInput() de cada subclasse;

A constante impressão dos menus no terminal e manuseamento de *inputs* do utilizador, corresponde a um *while loop* - na função run() da classe Current - até que o estado final seja atingido (opção "Exit" seja escolhida pelo utilizador).

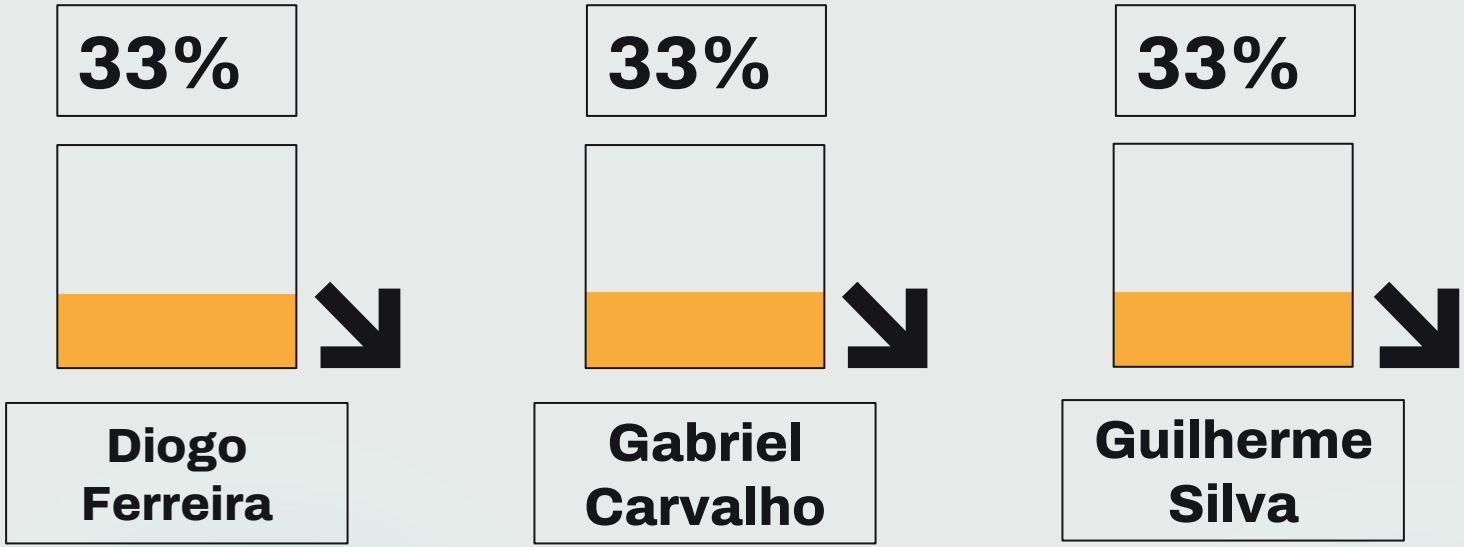
Destaque de Funcionalidades

↘			
Interface com o Utilizador	10/10	10/10	10/10
Melhores opções de voo	9/10	7.5/10	8/10
Destinos não diretos	8/10	9.5/10	9/10

Principais Dificuldades

- Melhores opções de voo
- Aeroportos essenciais

Esforço de cada elemento



Obrigado!

Há alguma questão?
Podem contactar-nos através de:

up202205295@up.pt
up202205298@up.pt
up202208939@up.pt