



Flight Management System

Algoritmos e Estruturas de Dados, 2023/24

Bruno Ferreira – up202207863

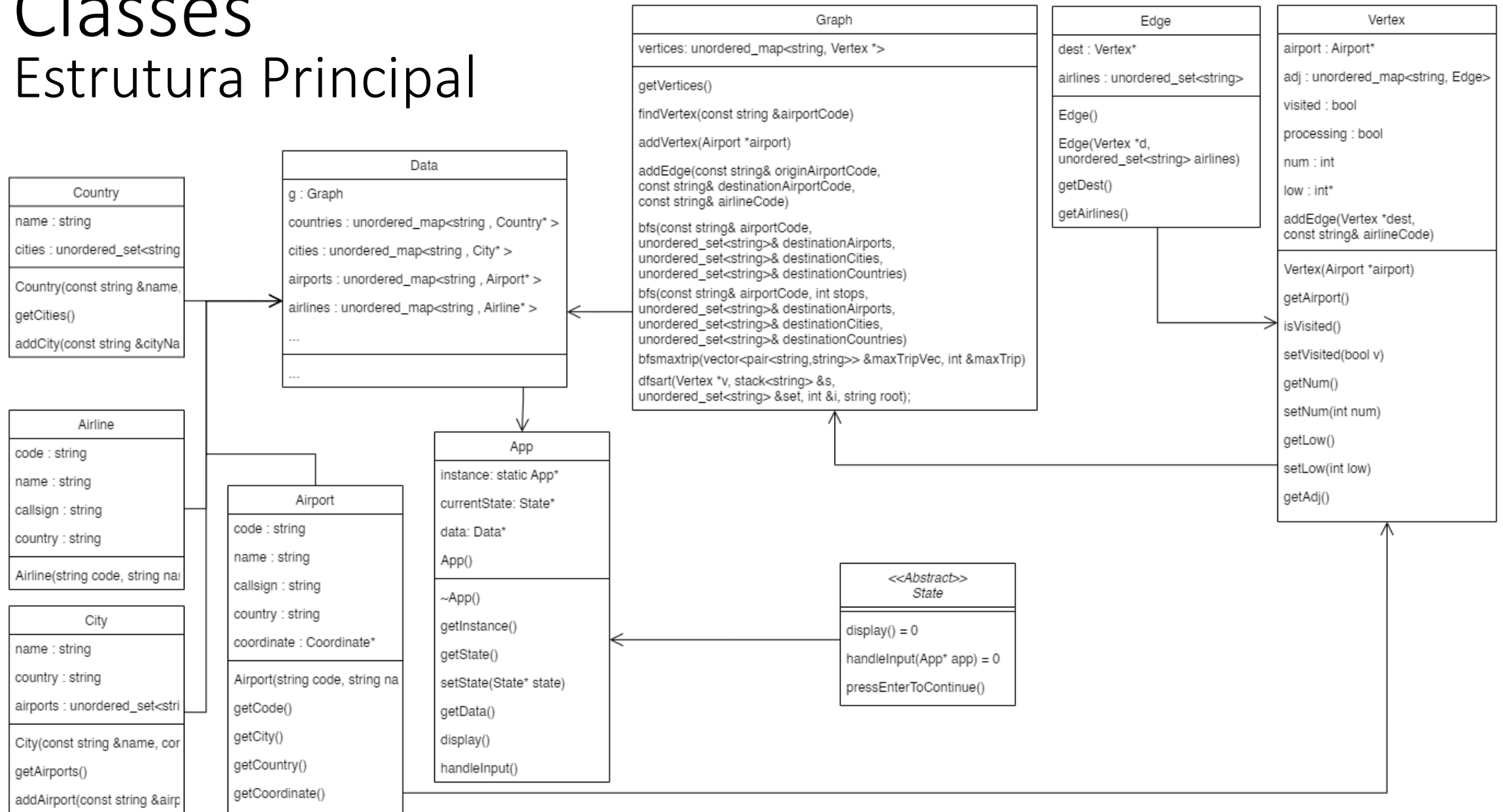
Eduardo Ferreira – up202206628

Xavier Martins - up202206632



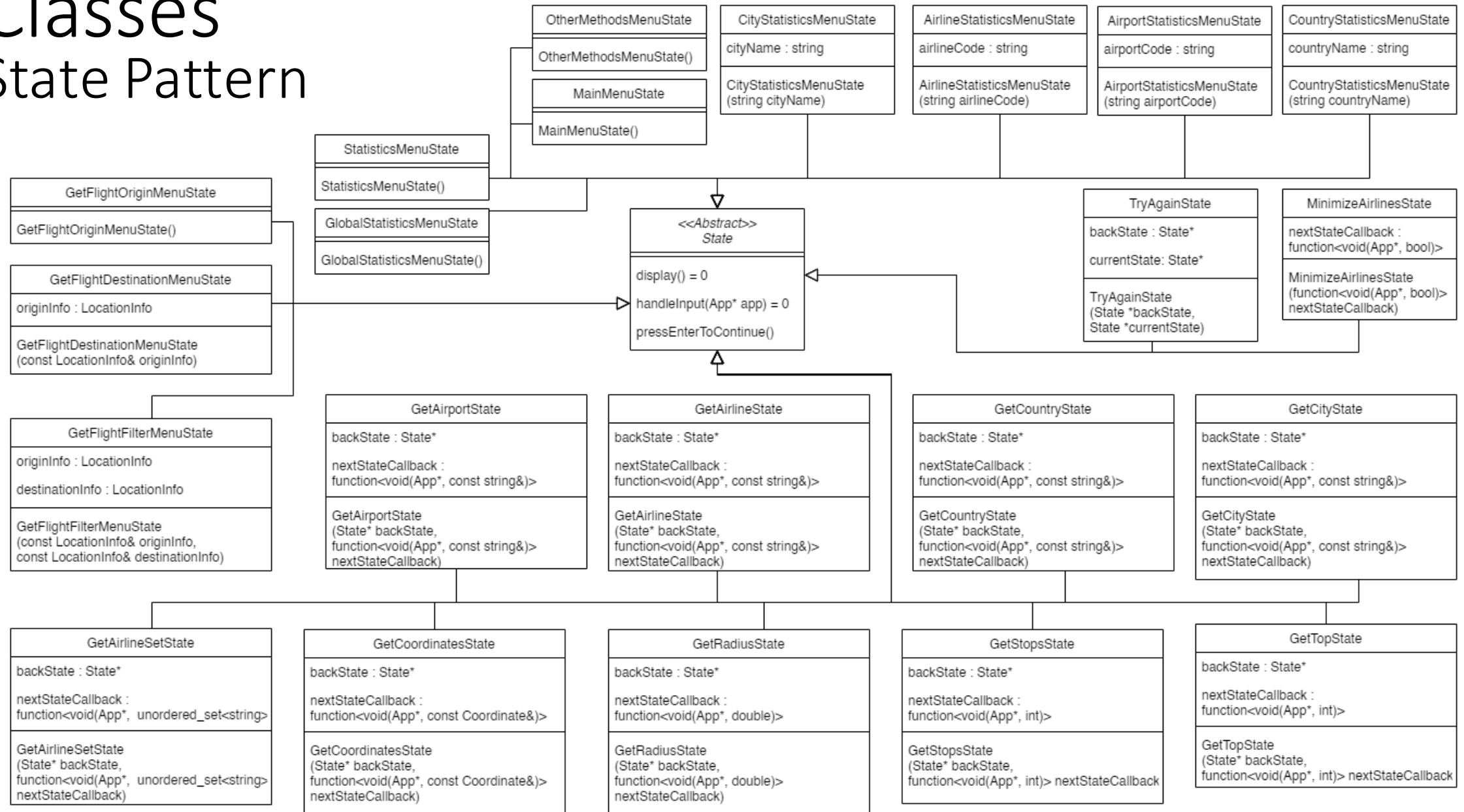
Classes

Estrutura Principal



Classes

State Pattern



Classe Data

Data	
airlines : unordered_map<string , Airline* > airports : unordered_map<string , Airport* > cities : unordered_map<string , City* > countries : unordered_map<string , Country* > g : Graph processFlights(const string& destiny, const unordered_map<string, pair<list<string>, int>>& airportTrack) convertLocation(const LocationInfo& location) minimalAirlines(const list<list<string>>& flights)	
Data() readFileAirlines() readFileAirports() readFileFlights() countryExists(const string& basicString) cityExists(const string& basicString) airportExists(const string& basicString) airlineExists(const string& basicString) numberOfAirports() numberOfCities() numberOfCountries() numberOfAirlines() numberOfFlights() numberOfAirportsCountry(const string& basicString) numberOfCitiesCountry(const string& basicString) airportsInCountry(const string& countryName) numberOfAirlinesCountry(const string& countryName) numberOfFlightsCountry(const string& countryName) numberOfDestinationsCountry(const string& countryName) numberOfDestinationsXStopsCountry(const string& countryName, int stops) numberOfAirportsCity(const string& cityName)	numberOfCountriesCity(const string& cityName) numberOfAirlinesCity(const string& cityName) numberOfFlightsCity(const string& cityName) numberOfDestinationsCity(const string& cityName) numberOfDestinationsXStopsCity(const string& cityName, int stops) airportsInCity(const string& cityName) numberOfFlightsAirline(const string& airlineCode) numberOfDestinationsAirline(const string& airlineCode) numberOfFlightsAirport(const string& basicString) numberOfAirlinesAirport(const string& airportCode) numberOfCountriesAirport(const string& airportCode) numberOfDestinationsAirport(const string& airportCode) numberOfDestinationsXStopsAirport(const string& airportCode, int stops) airportNearCoordinate(Coordinate coordinate) airportsInLocation(Coordinate coordinate, double radius) getFlights(const LocationInfo& originLocation, const LocationInfo& destinyLocation, unordered_set<string> airlineSet, bool unwantedAirlines, bool minimizeAirlines) maximumTrip() topKAirports(int k) essentialAirports() sortTopAirports(const pair<string, int>& a, const pair<string, int>& b)

Leitura do DataSet

A leitura do dataset é feito por intermédio de 3 funções, cada uma responsável pela leitura de um dos ficheiros:

readFileAirlines(), responsável pela leitura do ficheiro onde constam as airlines. O ficheiro é lido linha a linha e são criados objetos do tipo Airline, uma classe que contém as várias informações acerca da airline. Estes objetos são armazenados num mapa não ordenado, que tem como chave o código da respetiva airline.

readFileAirports(), responsável pela leitura do ficheiro que contém os aeroportos. Neste método são criados objetos do tipo Airport, estes são armazenados num mapa não ordenado, para permitir acesso rápido, e num grafo, sendo os vértices do mesmo. Ao longo da leitura deste ficheiro também são criados objetos do tipo City e Country, a cidade e o país ao qual o aeroporto pertence, que são guardados em mapas não ordenados para que o acesso às cidades e países existentes no sistema seja facilitado.

readFileFlights(), responsável pela leitura do ficheiro que contém os voos. Estes são representados no grafo como sendo as arestas, fazendo a ligação entre os aeroportos. As diferentes airlines que fazem um mesmo trajeto são armazenadas dentro de um set não ordenado que é guardado dentro da classe Edge, de forma a diminuir o número de arestas entre dois mesmos aeroportos.

Leitura do DataSet

```
void Data::readFileAirlines() {
    ifstream file(s: "../dataset/airlines.csv");

    if(!file.is_open()) {
        cout << "Error opening the file" << endl;
    } else {
        string line;
        getline(& file, & line);

        while(getline(& file, & line)) {
            string code, name, callsign, country;
            stringstream ss(str: line);
            getline(& ss, & code, delim: ',');
            getline(& ss, & name, delim: ',');
            getline(& ss, & callsign, delim: ',');
            getline(& ss, & country, delim: ',');

            Airline* airline = new Airline(code, name, callsign, country);
            this->airlines.insert(x: { & code, & airline});
        }
    }
}
```

```
void Data::readFileFlights() {
    ifstream file(s: "../dataset/flights.csv");

    if(!file.is_open()) {
        cout << "Error opening the file" << endl;
    } else {
        string line;
        getline(& file, & line);
        while(getline(& file, & line)){
            replace( first: line.begin(), last: line.end(), old_value: ',', new_value: ' ');

            stringstream ss(str: line);

            string origin, destination, airlineCode;
            ss >> origin >> destination >> airlineCode;
            g.addEdge( originAirportCode: origin, destinationAirportCode: destination, airlineCode);
        }
    }
}
```

```
void Data::readFileAirports() {
    ifstream file(s: "../dataset/airports.csv");

    if(!file.is_open()) {
        cout << "Error opening the file" << endl;
    } else {
        string line;
        getline(& file, & line);
        while(getline(& file, & line)) {
            string code, name, cityName, countryName, latitude, longitude;
            stringstream ss(str: line);
            getline(& ss, & code, delim: ',');
            getline(& ss, & name, delim: ',');
            getline(& ss, & cityName, delim: ',');
            getline(& ss, & countryName, delim: ',');
            getline(& ss, & latitude, delim: ',');
            getline(& ss, & longitude, delim: ',');

            Coordinate* coordinate = new Coordinate( latitude: stod( str: latitude), longitude: stod( str: longitude ));
            Airport* airport = new Airport(code, name, city: cityName+"."+countryName, country: countryName, coordinate);

            if(g.addVertex(airport)) {

                this->airports.insert(x: { & code, & airport});

                auto citiesIt :iterator<...> = cities.find(x: cityName); //????
                if(citiesIt == cities.end()) {
                    City* city = new City( name: cityName+"."+countryName, countryName, airportCode: code);
                    cities.insert(x: {x: cityName+"."+countryName, & city});
                } else citiesIt->second->addAirport( airportCode: code);

                auto countriesIt :iterator<...> = countries.find(x: countryName);
                if(countriesIt == countries.end()) {
                    Country* country = new Country( name: countryName, cityName: cityName+"."+countryName);
                    countries.insert(x: { & countryName, & country});
                } else countriesIt->second->addCity( cityName: cityName+"."+countryName);
            }
        }
    }
}
```

```
unordered_map<string , Airline* > airlines;
unordered_map<string , Airport* > airports;
unordered_map<string , City* > cities;
unordered_map<string , Country* > countries;
```


Grafo

Vértices (Aeroportos):

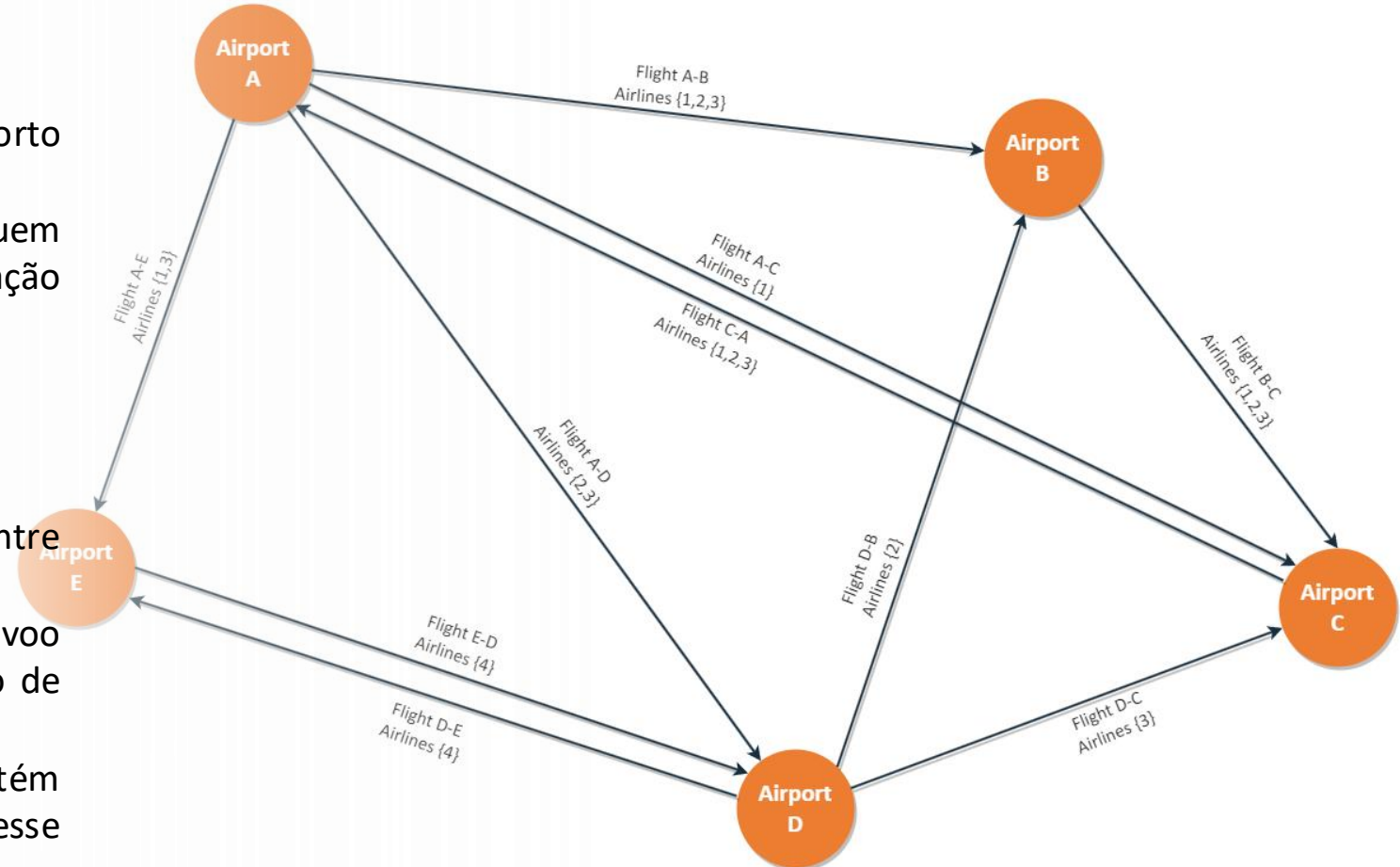
- Cada vértice do grafo simboliza um aeroporto individual.
- As informações associadas a cada vértice incluem dados como o nome do aeroporto, localização geográfica, capacidade, entre outros.

Exemplo: Vértice A que representa o Aeroporto A.

Arestas (Voos):

- Cada aresta do grafo simboliza um voo direto entre dois aeroportos.
- A aresta é direcionada, indicando a direção do voo de um aeroporto de origem para um aeroporto de destino.
- Um conjunto (set) associado a cada aresta contém os códigos das companhias aéreas que operam esse voo específico.

Exemplo: Uma aresta de A para B com o conjunto de códigos {1, 2, 3}, indica que as companhias aéreas 1, 2, 3 oferecem voos diretos do Aeroporto A para o Aeroporto B.



Grafo

```
class Vertex {
private:
    Airport* airport;           // contents
    unordered_map<string, Edge> adj; // list of outgoing edges
    bool visited;               // auxiliary field
    bool processing;            // auxiliary field
    int num;                     // auxiliary field
    int low;
private:
    void addEdge(Vertex *dest, const string& airlineCode);
public:
    Vertex(Airport *airport);
    Airport *getAirport() const;
    void setAirport(Airport *airport);
    bool isVisited() const;
    void setVisited(bool v);
    bool isProcessing() const;
    void setProcessing(bool p);
    int getNum() const;
    void setNum(int num);
    int getLow() const;
    void setLow(int low);
    const unordered_map<string, Edge> &getAdj() const;
    void setAdj(const unordered_map<string, Edge> &adj);
    friend class Graph;
};
```

o da leitura do dataset a
partir dos ficheiros dado

```
class Edge {
private:
    Vertex *dest;               // destination vertex
    unordered_set<string> airlines; // edge weight
public:
    Edge() = default;
    Edge(Vertex *d, unordered_set<string> airlines);
    Vertex *getDest() const;
    void setDest(Vertex *dest);
    unordered_set<string> getAirlines() const;
    void setAirlines(unordered_set<string> airlines);

    friend class Graph;
    friend class Vertex;
};
```

```
class Graph {
private:
    unordered_map<string, Vertex *> vertices; // vertex set
public:
    const unordered_map<string, Vertex *> &getVertices() const;
    Vertex *findVertex(const string &airportCode) const;
    bool addVertex(Airport *airport);
    bool addEdge(const string& originAirportCode, const string& destinationAirportCode, const string& airlineCode);
    void bfs(const string& airportCode, unordered_set<string>& destinationAirports, unordered_set<string>& destinationCities, unordered_set<string>& destinationCountries);
    void bfs(const string& airportCode, int stops, unordered_set<string>& destinationAirports, unordered_set<string>& destinationCities, unordered_set<string>& destinationCountries);
    void bfsmaxtrip(vector<pair<string, string>> &maxTripVec, int &maxTrip);
    void dfsart(Vertex *v, stack<string> &s, unordered_set<string> &set, int &i, string root);
};
```




Funcionalidades implementadas(Estatísticas)

Uma das funcionalidades do nosso sistema é a possibilidade de dar várias opções estatísticas ao utilizador.

O cálculo destas estatísticas é feito por métodos que se encontram dentro da classe Data. No cálculo da maioria delas faz-se recurso a mapas não ordenados desta classe, onde estão representados os aeroportos, airlines, cidades e países que o dataset contém, como tal as funcionalidades mais simples são executadas com tempo constante. Algumas outras recorrem ao grafo, de forma a ter acesso às conexões entre aeroportos, levando a algoritmos de maior complexidade.

Estatísticas mais complexas como o número de destinos, ou o número de destinos com x paragens recorrem a bfs representadas na classe Graph.

Funcionalidades implementadas(Estatísticas)

```
===== STATISTICS =====
```

1. Global
2. Country
3. City
4. Airline
5. Airport

q. Main Menu

```
===== GLOBAL STATISTICS =====
```

Number of:

1. Airports
2. Cities
3. Countries
4. Airlines
5. Flights

b. Statistics Menu

q. Main Menu

```
===== COUNTRY STATISTICS =====
```

Portugal

Number of:

1. Airports
2. Cities
3. Airlines
4. Flights
5. Destinations
6. Reachable Destinations with max X stops

b. Statistics Menu

q. Main Menu

```
===== AIRPORT STATISTICS =====
```

OPO

Number of:

1. Countries airport flies to
2. Airlines out of the airport
3. Flights
4. Destinations
5. Reachable Destinations with max X stops

b. Statistics Menu

q. Main Menu

```
===== CITY STATISTICS =====
```

Montevideo,Uruguay

Number of:

1. Airports
2. Cities
3. Airlines
4. Flights
5. Destinations
6. Reachable Destinations with max X stops

b. Statistics Menu

q. Main Menu

```
===== AIRLINE STATISTICS =====
```

TAP

Number of:

1. Flights
2. Destinations

b. Statistics Menu

q. Main Menu

Funcionalidades implementadas(Melhor voo)

Esta funcionalidade essencial é processada, maioritariamente, pelo método `GetFlights()` da classe `Data`, capaz de calcular os melhores voos entre duas localidades considerando as restrições das companhias aéreas a utilizar. Depende de alguns outros métodos auxiliares como o `minimalAirline()`, para identificar quais voos necessitam do menor número de companhias aéreas diferentes, ou o `processFlights()`, responsável por criar as rotas reais com base no relacionamento entre aeroportos.

A lógica geral do algoritmo `GetFlights()` é calcular, a partir de cada aeroporto de origem possível, qual é o primeiro aeroporto de destino alcançável, usando uma abordagem de bfs. No processo são consideradas a distância atual do aeroporto de origem e um histórico de relações entre aeroportos, para que seja possível escolher os voos com menor número de escalas e, com a ajuda do método `processFlights()`, calcular todas as diferentes rotas que têm essa distância mínima.

Para escolher os voos que necessitam do menor número de companhias aéreas diferentes a abordagem foi passar por cada conexão de cada voo, e armazenar um histórico da utilização das companhias aéreas, de forma a calcular qual é o número mínimo de mudanças de companhia aérea necessárias para esse voo específico.

Funcionalidades implementadas(Melhor voo)

```
===== TYPE OF ORIGIN =====
```

1. Airport
2. City
3. Country
4. Coordinates
5. Coordinates & Radius

q. Main Menu

```
-----
```

```
===== FLIGHT FILTERS =====
```

OPO

Ø

LGA

1. Set of airlines
2. Set of unwanted airlines
3. No filter

b. Go Back

q. Main Menu

```
-----
```

Insert airport code: OPO

```
===== TYPE OF DESTINY =====
```

OPO

Ø

1. Airport
2. City
3. Country
4. Coordinates
5. Coordinates & Radius

b. Go Back

q. Main Menu

```
-----
```

```
Show only flights with the minimum number of different airlines? [s/n]: s
```

```
>> Top flights found: 1
```

Minimal number of airlines: 1

OPO - YYZ - LGA

```
-----
```

Funcionalidades implementadas(Outros)

Maximum Trip - Representa o diâmetro do grafo, este valor é calculado através de uma bfs especial, representada na classe Graph que retorna o valor da maximum trip e os aeroportos de origem e de destino que estão a essa distância.

Top K airport with the greatest air traffic capacity - Apresenta um top dos aeroportos com maior tráfego, sendo o tamanho do top escolhido pelo utilizador. O algoritmo utilizado passa por calcular o número de voos em cada aeroporto e guardar num vetor, posteriormente ordenado.

Essential Airports – Representa a quantidade de aeroportos essenciais(articulation points) para que não haja aeroportos sem comunicação com algum outro. Nesta implementação é usado um algoritmo aproximado ao dado em aula para calcular os articulation points.

```
===== OTHER METHODS =====  
1. Maximum trip  
2. Top K airport with the greatest air traffic capacity  
3. Essential airports  
  
q. Main Menu  
-----
```


Interface do Utilizador

Para a interface do utilizador procuramos que fosse o mais simples e intuitiva possível, como tal recorremos ao uso de cores e sugestões de input para facilitar a leitura e a compreensão do que é pretendido.

```
===== MAIN MENU =====
```

- 1. Statistics
- 2. Get Flight
- 3. Other Methods

q. Exit

```
-----  
Enter your choice:
```

```
===== COUNTRY STATISTICS =====
```

Portugal

Number of:

- 1. Airports
- 2. Cities
- 3. Airlines
- 4. Flights
- 5. Destinations

- 6. Reachable Destinations with max X stops

b. Statistics Menu

q. Main Menu

Destaque de funcionalidades

A funcionalidade que julgamos ser a melhor no programa é a que nos indica o melhor voo tendo em conta a preferência do utilizador, visto que tem grande variedade quer na escolha da origem quer do destino, além de permitir fazer alguma filtragem nas airlines a usar.

===== TYPE OF ORIGIN =====

1. Airport
2. City
3. Country
4. Coordinates
5. Coordinates & Radius

q. Main Menu

===== FLIGHT FILTERS =====

OPO

Ø

LGA

1. Set of airlines
2. Set of unwanted airlines
3. No filter

b. Go Back

q. Main Menu

Insert airport code: OPO

===== TYPE OF DESTINY =====


OPO

Ø

1. Airport
2. City
3. Country
4. Coordinates
5. Coordinates & Radius

b. Go Back

q. Main Menu



Principais dificuldades/Esforço dos elementos

A maior dificuldade certamente foi fazer o sistema de escolha do melhor voo, visto que as hipóteses de input são muito variadas e tem de se prestar bastante atenção aos diversos casos.

Além disso, inicialmente, houve também alguma dificuldade no planeamento da melhor forma de implementar e estruturar todo o sistema de voos.

O trabalho foi bem distribuído entre os vários elementos do grupo, tendo todos contribuindo de igual forma para o mesmo.