# Flight Management System

Ano Letivo 2023/2024

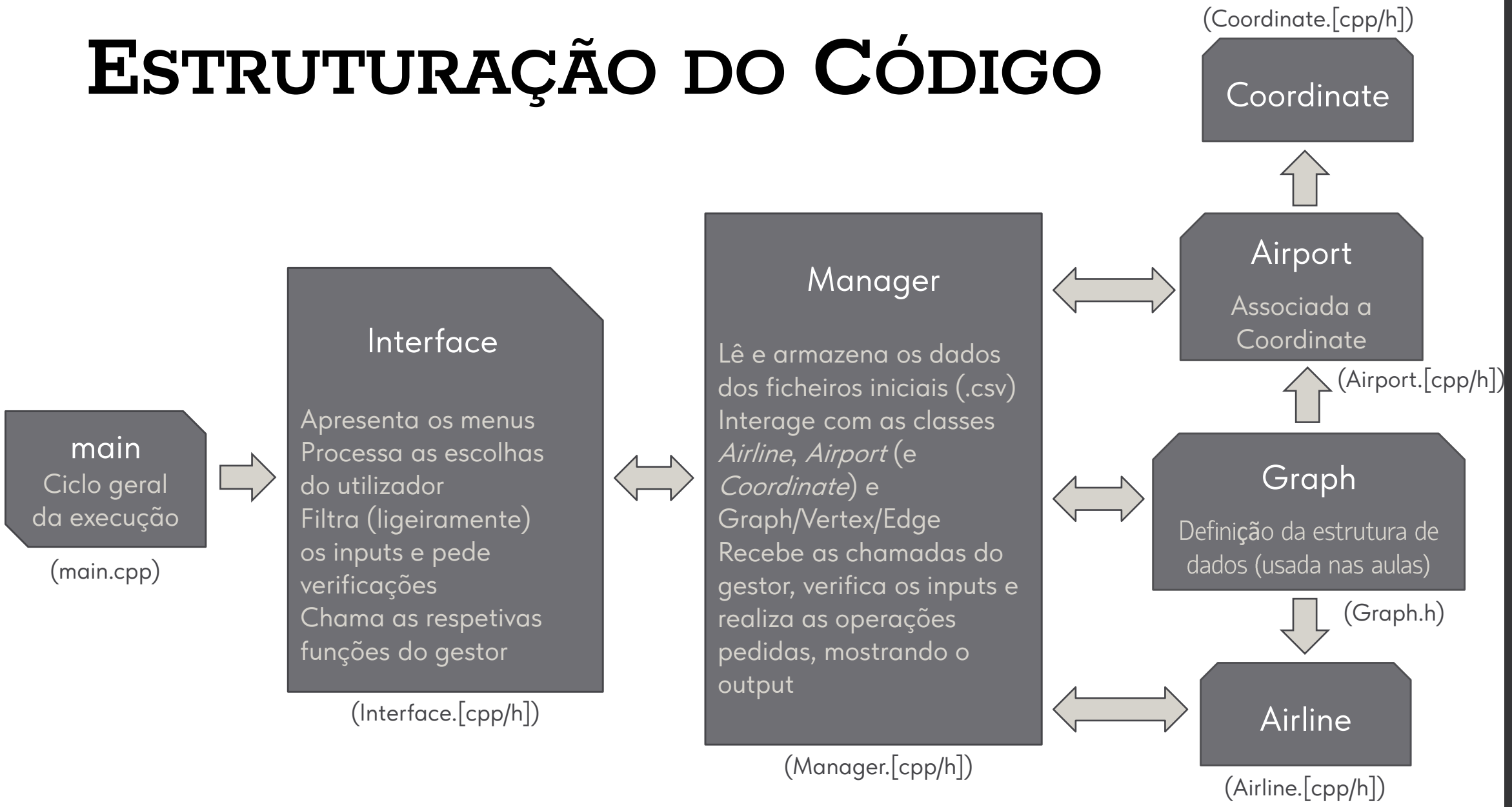
Algoritmos e Estruturas de Dados

(L.EIC011)

Turma 01, Grupo 15

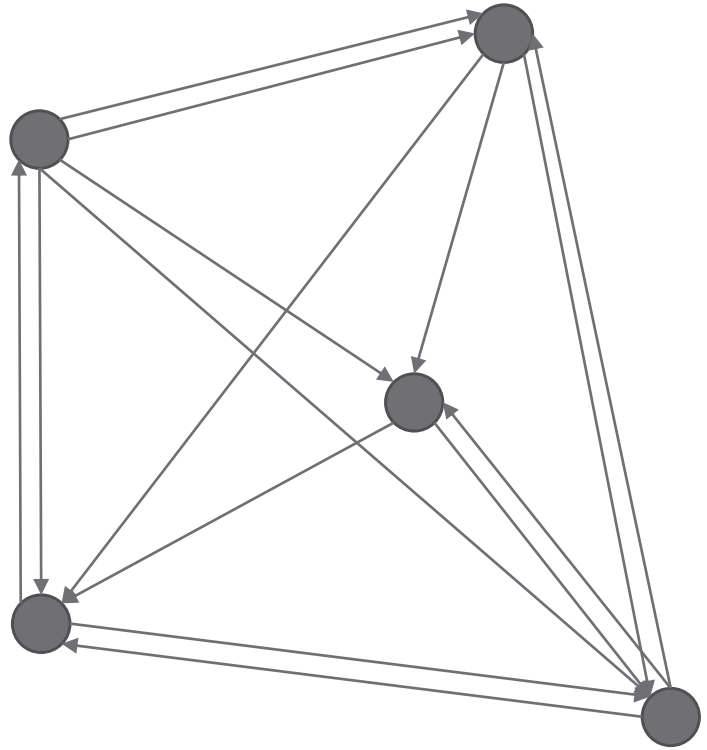
Rafael Teixeira de Magalhães
Ricardo de Freitas Oliveira
Rodrigo Albergaria Coelho e Silva

# Estruturação do Código



# Descrição do Grafo

- Baseado no ficheiro usado nas aulas
- Classes de Graph, Edge e Vertex mantém o seu aspeto geral, apenas com ligeiras alterações
  - Alterações na função Graph:
    - Algoritmos para calcular o diâmetro e os pontos de articulação do grafo (assim como funções auxiliares para os mesmo algoritmos)
  - Alterações na função Edge:
    - Variáveis weight (*double*) e info (*string*) que permitem guardar a distância associada ao voo e informação da companhia aérea que o realiza
  - Alterações na função Vertex:
    - Variáveis lowest, visitIndex e auxiliar (*int*) para usar nos algoritmos de diâmetro e pontos de articulação do grafo
    - Conjunto parents (*unordered_set<string>*) para guardar os valores dos vértices anteriores no BFS usado para obter o melhor voo



O grafo usado é direcionado e permite a existência de várias arestas entre os mesmo vértices, desde que o *info* associado seja diferente.

No Manager, é guardado um grafo em que cada vértice é um *Airport* e cada aresta guarda a distância entre os dois aeroportos e o código da *Airline*.

# Leitura do Dataset

- A leitura e armazenamento do dataset são realizados pelo Manager
- Para facilitar as pesquisas que são colocadas sobre o dataset, o Manager guarda o grafo e alguns sets e mapas auxiliares para facilitar a pesquisa e validação dos dados
- A leitura dos dados, de qualquer um dos ficheiro pode ser dividida em duas partes:
  - Leitura de cada um dos campos para variáveis auxiliares (bastante parecida para qualquer um dos ficheiros csv)
  - Carregamento dos dados para as estruturas de dados adequadas (mais específica de cada um dos ficheiros e apresentada em maior detalhe nas imagens seguintes)

```
class Manager {
private:
    Graph<Airport> flightNet;
    unordered_set<Airline, AirlineHash, AirlineHash> airlines;
    unordered_set<Airport, AirportHash, AirportHash> airports;
    unordered_set<std::string> cities;
    unordered_set<std::string> countries;

    unordered_map<string, string> airportNameToCode;
    unordered_map<string, Vertex<Airport> *> airportCodeToVertex;
    unordered_map<string, int> countryToAirportCount;
    unordered_map<string, int> countryToAirlineCount;
    unordered_map<pair<string, string>, int> countryCityToAirportCount;
```

```
bool Manager::extractAirlines(std::string fname) {
    ifstream input(fname);
    std::string line;

    if (!getline(input, line)) return false;

    std::string code, name, callsign, country;

    getline(input, line);
    do {
        istringstream lineInput(line);
        getline(lineInput, code, ',');
        getline(lineInput, name, ',');
        getline(lineInput, callsign, ',');
        getline(lineInput, country, '\r');


        Airline airline = Airline(code, name, callsign, country);
        airlines.insert(airline);
        countries.insert(country);
        countryToAirlineCount[country]++;
    } while (getline(input, line));
    return true;
}
```

# Leitura do Dataset

```
bool Manager::extractAirports(std::string fname) {
    ifstream input(fname);
    std::string line;

    if (!getline(input, line)) return false;

    std::string code, name, city, country, lats, lgts;
    double lat, lgt;

    getline(input, line);
    do {
        istringstream lineInput(line);
        getline(lineInput, code, ',');
        getline(lineInput, name, ',');
        getline(lineInput, city, ',');
        getline(lineInput, country, ',');
        getline(lineInput, lats, ',');
        getline(lineInput, lgts, '\r');
        lat = stod(lats);
        lgt = stod(lgts);

        Airport airport = Airport(code, name, city, country, Coordinate(lat, lgt));
        airports.insert(airport);
        Vertex<Airport> *vx = flightNet.addVertex(airport);
        airportCodeToVertex[code] = vx;
        airportNameToCode[name] = code;
        countryToAirportCount[country]++;
        countryCityToAirportCount[make_pair(country, city)] += 1;
        cities.insert(city);
        countries.insert(country);
    } while (getline(input, line));
    return true;
}
```

```
bool Manager::extractFlights(std::string fname) {
    ifstream input(fname);
    std::string line;

    if (!getline(input, line)) return false;

    std::string code1, code2, airlineCode;

    getline(input, line);
    do {
        istringstream lineInput(line);
        getline(lineInput, code1, ',');
        getline(lineInput, code2, ',');
        getline(lineInput, airlineCode, '\r');

        Vertex<Airport> *a1 = airportCodeToVertex[code1];
        Vertex<Airport> *a2 = airportCodeToVertex[code2];
        double distance = a1->getInfo().getCoordinate().distance(a2->getInfo().getCoordinate()

        flightNet.addEdge(a1, a2, distance, airlineCode);
    } while (getline(input, line));
    return true;
}
```

- Uma vez que os voos dependem das Airlines e dos Airports, têm de ser carregados no final das restantes

# Funcionalidades Implementadas

- Leitura dos ficheiros iniciais
- Apresentação de estatísticas/informações sobre o dataset/grafo
  - Informações sobre os aeroportos, as companhias aéreas, os voos e as cidades/países
  - Informações sobre o grafo (pontos de articulação, diâmetro, maiores trajetos)
- Pesquisa dos melhores trajetos entre aeroportos de origem e destino
  - Aeroportos selecionados com base em:
    - Código ou nome dos aeroportos
    - Conjunto País/Cidade
    - Coordenadas
- Filtros para a pesquisa dos melhores trajetos
  - Restrições de aeroportos (ou até cidades/países)
  - Restrições de companhias aéreas
  - Preferência de companhias aéreas
- Interface Intuitiva

# Funcionalidades Implementadas

## Informações sobre os aeroportos

- Listagem e contagem total dos aeroportos [O(n)] / [O(1)]
- Listagem e contagem dos aeroportos de um determinado país/cidade [O(n)]
- Listagem ordenada dos n aeroportos com maior número de voos [O(n*log(n)]
- Listagem ordenada dos n aeroportos com maior número de companhias aéreas (partidas) [O(n*m*log(n)]
- Informação sobre uma companhia aérea em específico [O(1)]
- Listagem e contagem das companhias aéreas e voos que partem de um determinado aeroporto [O(n)]
- Aeroportos, Cidades e Países diretamente alcançáveis a partir de um determinado aeroporto [O(|V| + |E|)]
- Aeroportos, Cidades e Países alcançáveis com n voos a partir de um determinado aeroporto [O(|V| + |E|)]

## Informações sobre as companhias aéreas

- Listagem e contagem total das companhias aéreas [O(n)] / [O(1)]
- Listagem e contagem das companhias aéreas que têm partidas de um determinado aeroporto [O(n)]
- Listagem e contagem das companhias aéreas de um determinado país (país de origem da companhia) [O(n)]
- Informação sobre uma companhia aérea em específico [O(1)]

## Informações sobre os voos

- Listagem e contagem total dos voos [O(|V| + |E|)]
- Listagem e contagem dos voos de uma determinada companhia aérea [O(|V| + |E|)]
- Listagem e contagem dos voos de partida de um determinado país/cidade [O(|V| + |E|)]
- Listagem e contagem dos voos de chegada de um determinado país/cidade [O(|V| + |E|)]

```
void listAllAirlines();
void numberAirlines();
void listAirlinesAirport(string airport
void listAirlinesCountry(string country
void airlineInfo(string airline);

void listAllAirports();
void numberAirports();
void listAirportsCountryCity(string cou
void listAirportsMostAirlines(int n);
void listAirportsMostFlights(int n);
void airportInfo(string airport);
void listAirportFlights(string airport)
void reachableAirports(string airport,
void reachableCities(string airport, in
void reachableCountries(string airport,

void listAllFlights();
void numberFlights();
void listFlightsAirline(string airline)
void numberFlightsAirline(string airlin
void listArrivalsCountryCity(string cou
void numberArrivalsCountryCity(string c
void listDeparturesCountryCity(string c
void numberDeparturesCountryCity(string

void listCountriesMostAirlines(int n);
void listCountriesMostAirports(int n);
void listCitiesMostAirports(int n);

void articulationPoints();
void diameter();
void longestPaths();
```

# Funcionalidades Implementadas

## Informações sobre Cidades/Países

- Listagem e contagem das companhias aéreas de um determinado país [O(n)]
- Listagem e contagem dos aeroportos de um determinado país/cidade [O(n)]
- Listagem ordenada dos n países com maior número companhias aéreas [O(n*log(n))]
- Listagem ordenada dos n países com maior número aeroportos [O(n*log(n))]
- Listagem ordenada dos n países/cidades com maior número aeroportos [O(n*m*log(n*m))]

## Pontos de Articulação [O(|V|+|E|)]

- Contagem dos aeroportos essenciais para a capacidade de circulação da rede (pontos de articulação)
- Listagem dos aeroportos e do seu local

## Diâmetro do Grafo [O(|V|*(|V|+|E|))]

- Maior rota (mínima) entre dois aeroporto na rede (maior número de paragens entre dois aeroportos)

## Maiores Trajetos (pares origem-destino)

- Listagem dos pares de aeroportos origem-destino cujo nº de paragens mínimo é o diâmetro do grafo [O(|V|*(|V|+|E|))]

```
void listAllAirlines();
void numberAirlines();
void listAirlinesAirport(string airport);
void listAirlinesCountry(string country);
void airlineInfo(string airline);

void listAllAirports();
void numberAirports();
void listAirportsCountryCity(string country, s
void listAirportsMostAirlines(int n);
void listAirportsMostFlights(int n);
void airportInfo(string airport);
void listAirportFlights(string airport);
void reachableAirports(string airport, int n=1
void reachableCities(string airport, int n=1);
void reachableCountries(string airport, int n=

void listAllFlights();
void numberFlights();
void listFlightsAirline(string airline);
void numberFlightsAirline(string airline);
void listArrivalsCountryCity(string country, s
void numberArrivalsCountryCity(string country,
void listDeparturesCountryCity(string country,
void numberDeparturesCountryCity(string countr

void listCountriesMostAirlines(int n);
void listCountriesMostAirports(int n);
void listCitiesMostAirports(int n);

void articulationPoints();
void diameter();
void longestPaths();
```

# Funcionalidades Implementadas

## Pesquisa do Melhor Trajeto (nº mínimo paragens)

$O(n * m * (|V| + |E|)$

- Leitura de um conjunto de valores de aeroportos de origem e destino como combinação de:
  - Código ou nome dos aeroportos
  - Par País/Cidade
  - Coordenadas
- Pesquisa dos trajetos com menor número de paragens desde as várias origens ao vários destinos
- Reconstrução dos caminhos voo-a-voo
- Apresentação dos trajetos com o menor número de paragens de todos (entre todas as possíveis origens e destinos)
- Apresentação de todas as companhias aéreas possíveis entre todos os voos considerados num trajeto (todos as possíveis arestas entre os dois vértices de um caminho mínimo)

## Filtros para a Pesquisa (restrições e preferências)

- Leitura de um conjunto de filtros para:
  - Companhias aéreas preferenciais (procura trajetos apenas dessas companhias)
  - Companhias aéreas restritas (procura trajetos que não usem essas companhias)
  - Aeroportos/Cidades/Países restritos (procura trajetos que não usem esses aeroportos)
- Apresentação da pesquisa do melhor trajeto recorrendo ao conjunto de filtros selecionados

# Interface com o Utilizador

- A interface geral do programa é feita à base de menus sucessivos apresentados na tela

**Para uma melhor visualização, recomenda-se correr o programa no terminal ou ativar a emulação de terminal da consola do CLion (para permitir "limpar o ecrã" depois de apresentar os dados)**

- A interface com o utilizador pode ser dividida em:
  - Funções de Input (presentes na Interface)
  - Funções de Output (presentes no Manager)
- A parte de input começa com a seleção do menu a ser apresentado, baseada na opção escolhida de entre as apresentadas no ecrã (a seleção e as opções são feitas de acordo com um número, sendo o [0] o de andar para trás ou sair do programa)
- O input engloba também as funções que pedem parâmetros ao utilizador que depois são passados ao Manager para realizar operações sobre o dataset



# Interface com o Utilizador

- A parte de output está associada ao manager e à interface e é padronizada e modularizada de acordo com o tipo de dados que pretendemos apresentar
- Na parte dos menus, existem várias funções para apresentar as opções de menu seguinte, o cabeçalho e rodapé, a leitura de dados atual...
- O output está relacionado com os resultados de pesquisas, realizadas pelo Manager, que é feito a partir de funções para uma maior uniformidade na apresentação
- Na maioria dos casos, estes são apresentados no formato de tabela com a contagem ou valor associado no final, ou até apenas o valor
- Para listagens ordenadas, são apresentados os resultados por ordem com o seu número associado e o valor que os permite ordenar
- Para outro tipo de listagens é apresentado o título da lista e os vários valores de forma sequencial
- Para as melhores viagens, são apresentados da esquerda para a direita os caminhos que se podem percorrer

```
void printCount(int number, string text);
void printDouble(double number, string text);
void printListHeader(string text);
void printListValue(string text);
void printOrderedValue(int pos, int count, string text);
void printOrderedValueAirport(int pos, int count, string text1, string text2);
void printOrderedValueCity(int pos, int count, string text1, string text2);
void printAirlineHeader();
void printAirline(const Airline &airline);
void printAirlineFooter();
void printAirportHeader();
void printAirport(const Airport &airport);
void printAirportFooter();
void printDepartureHeader();
void printDeparture(string source, const Airport &dest, string airline, double distance);
void printDepartureFooter();
void printArrivalHeader();
void printArrival(const Airport &source, string dest, string airline, double distance);
void printArrivalFooter();
void printFlightHeader();
void printFlight(const Airport &source, const Airport &dest, string airline, double distance);
void printFlightFooter();
int printPaths(vector<string>* airlinePreferences, vector<string> *airlineRestrictions);
void printFlightAirlines(const string &currentAirport, const string &nextAirport, vector<string>* airlinePreferences, vector<string> *airlineRestrictions);
```

| SRC | DEST | DESTINATION NAME | AIRLINE | DISTANCE |
|---|---|---|---|---|
| BLA | CCS | Simon Bolivar Intl | RLN | 257.707 |
| BLA | CCS | Simon Bolivar Intl | OCA | 257.707 |
| BLA | CCS | Simon Bolivar Intl | VCV | 257.707 |
| BLA | PZO | General Manuel Carlos Piar | OCA | 292.766 |
| BLA | VLN | Arturo Michelena Intl | OCA | 354.604 |

```
Number of Flights: 5
Longest Flight Distance [km]: 354.604
```

```
> N Airports with Most Flights selected
Number: 5

Airports with most Flights:
1)  ATL  (Hartsfield Jackson Atlanta Intl)   909
2)  ORD  (Chicago Ohare Intl)                556
3)  PEK  (Capital Intl)                      526
4)  LHR  (Heathrow)                          525
5)  CDG  (Charles De Gaulle)                 518
```

```
> Process Operation selected
Path   1:   LHR [ AAL AFR BAW ] CDG
Path   2:   LGW [ EZY ] CDG
Path   3:   LTN [ EZY ] CDG
Path   4:   LHR [ BER BAW IBE ] ORY
Path   5:   LCY [ AFR BCY ] ORY

Minimum path distance (Flights): 1
Total number of different paths: 5
```

# Documentação do Código

No que toca à documentação, a mesma foi gerada usando o Doxygen, tal como era pedido na descrição do projeto. Desta faz parte:

- Descrição simples das classes usadas no projeto e da sua função
- Atributos e funções membro de todas as classes da função
- Descrição detalhada das funções e algoritmos mais importantes da classe Manager e Graph, assim como a sua complexidade temporal associada

```
/**
 * @brief Prints the best flight option, considering a number of preferences.
 * Complexity: O(n * m * (|V| + |E|), n being the size of sources, m the size of destinations,
 * @param sources - vector of names and/or codes of the airports of departure.
 * @param destinations - vector of names and/or codes of the airports of arrival.
 * @param airlinePreferences - Vector of codes of preferred airlines.
 * @param airlineRestrictions - Vector of codes of airlines to avoid.
 * @param airportRestrictions - Vector of codes of airports to avoid.
 */
void Manager::bestFlightOption(vector<string> *sources, vector<string> *destinations, vector<st
```

◆bestFlightOption()

void Manager::bestFlightOption ( vector< string > * sources,
vector< string > * destinations,
vector< string > * airlinePreferences,
vector< string > * airlineRestrictions,
vector< string > * airportRestrictions
)

Prints the best flight option, considering a number of preferences. Complexity: O(n * m * (|V| + |E|), n being the size of sources, m the size of destinations, |V|

**Parameters**

| | |
|---|---|
| sources | - vector of names and/or codes of the airports of departure. |
| destinations | - vector of names and/or codes of the airports of arrival. |
| airlinePreferences | - Vector of codes of preferred airlines. |
| airlineRestrictions | - Vector of codes of airlines to avoid. |
| airportRestrictions | - Vector of codes of airports to avoid. |

# Destaque de Funcionalidades

Já no trabalho anterior, um dos aspetos que consideramos mais importantes foi a organização dos menus e a forma como os dados são apresentados ao utilizador, uma vez que o objetivo é criar um sistema que além de conseguir realizar algumas operações ou pesquisas, seja intuitivo e fácil de usar do ponto de vista de alguém sem grande experiência.

Neste trabalho, decidimos focar ainda mais em melhorar a forma como é feito o input e o output de informação e achamos que o resultado é bastante proveitoso. Os resultados são apresentados com uma boa formatação e isso, aliado à rapidez da maioria das operações permite que o uso do sistema de gestão de voos que criamos seja bastante fluida, mesmo com um dataset tão grande como o que nos foi fornecido.

# Conclusão

Em conclusão, podemos dizer que a realização deste projeto foi muito útil pois permitiu-nos consolidar os conhecimentos todos das aulas de Algoritmos e Estruturas de Dados, assim como trabalhar com as estruturas de dados e algoritmos já lecionados de uma maneira mais prática e aplicada à vida real.

Por ser um projeto de maior dimensão, a dificuldade é claramente um pouco maior do que os exercícios semanais, mas mesmo assim consideramo-nos bastante satisfeitos com o resultado final que apresentamos, embora tenhamos enfrentado algumas dificuldades (nomeadamente, na pesquisa do melhor voo recorrendo a filtros e várias opções em conjunto).

No final, a nossa solução tem todas as funcionalidades pedidas e ainda mais algumas implementadas e utiliza as estruturas de dados pedidas mantendo complexidades temporais e tempos de acesso razoáveis. A divisão de tarefas e o esforço de cada elemento do grupo foi adequada.

Turma 01, Grupo 15

Rafael Teixeira de Magalhães
Ricardo de Freitas Oliveira
Rodrigo Albergaria Coelho e Silva