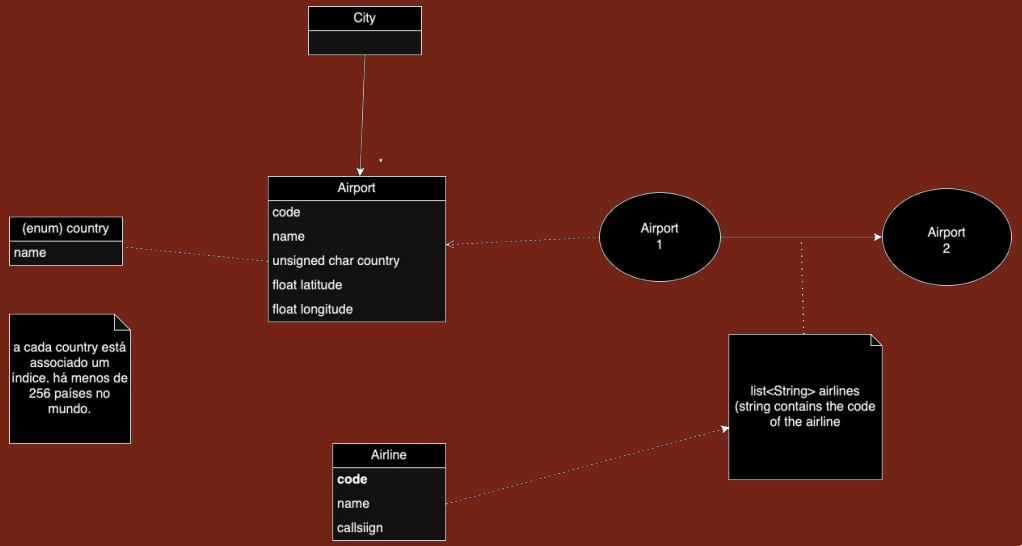
Sistema de Gestão de Voos

Filipe Correia
Gonçalo Nunes
Grupo G32

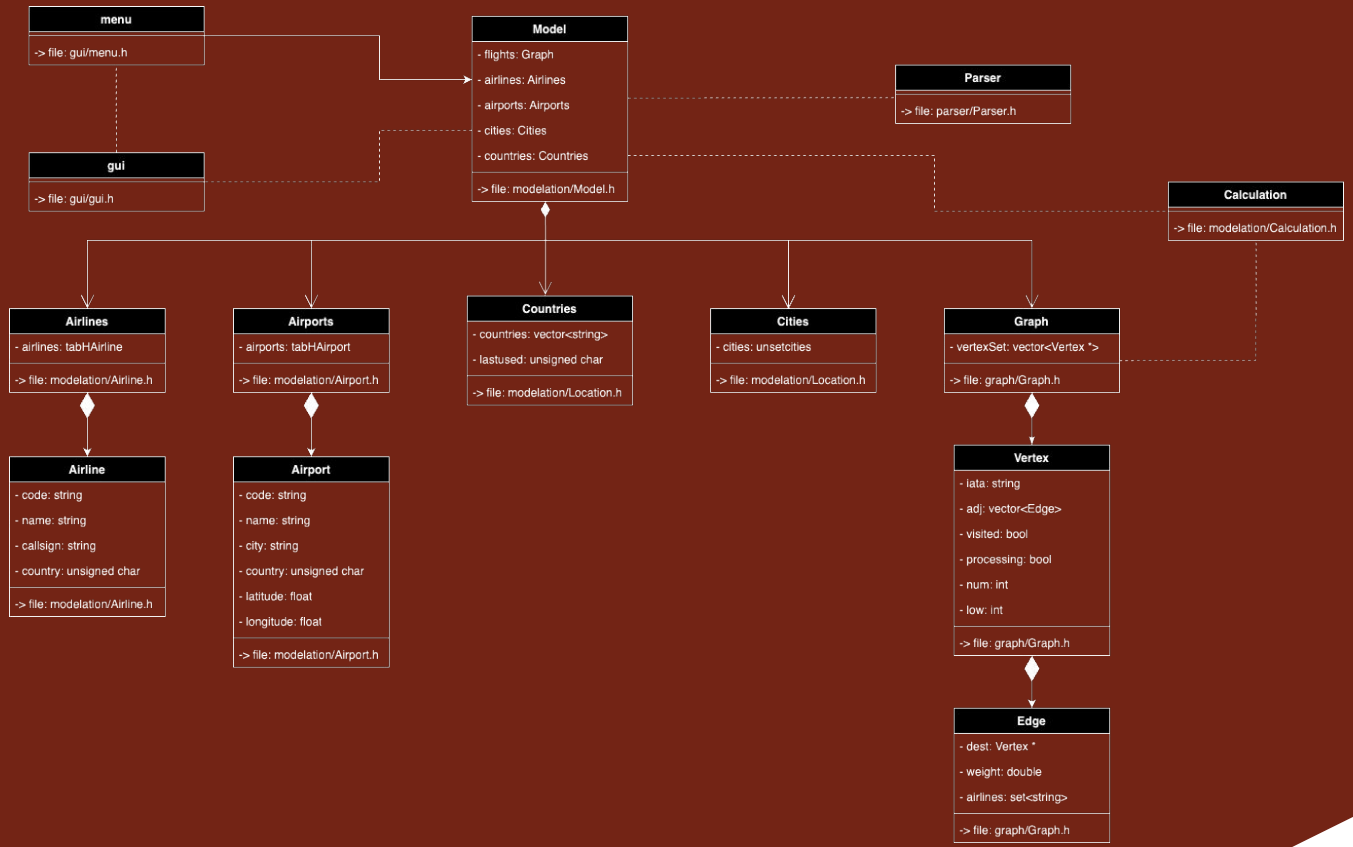
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Inicialmente...



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Diagrama de Classes Final



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Parsing dos dados

- Informação relativa a airports e airlines guardada em Hash Maps.
- Informação relativa aos flights guardada num Graph.
- Durante a leitura dos flights, apenas adicionamos uma nova edge se ainda não existir nenhum voo com o mesmo par (source, destination), se existir apenas adicionamos uma nova airline ao edge respetivo.
- À medida que eram encontradas diferentes países e cidades, durante a leitura de dados sobre airlines e airports, estes eram adicionados a um objeto de classes específicas (Countries e Cities) para termos registo de todas as localizações deste sistema.

Estruturas de Dados Usadas

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Airline, Airlines, Cities, Countries

```
class Airline {
private:
   std::string code;
   std::string name;
   std::string callsign;
   unsigned char country;
}

class Cities {
   unordered_set<pair<string,
unsigned char>> cities;
}
```

```
class Airlines {
private:
   unordered_set<Airline> airlines;
}

class Countries {
private:
    std::vector<std::string> countries;
         unsigned char lastused = 0;
}
```


U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

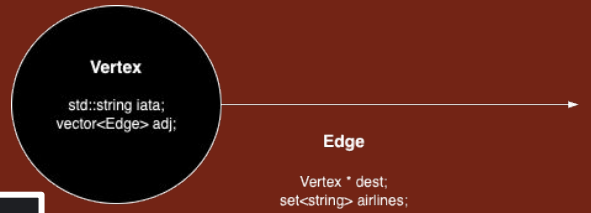
Airport, Airports, Calculation

```
class Airport {
private:
        std::string code;
        std::string name;
        std::string city;
        unsigned char country;
        float latitude;
        float longitude;
}
```

```
class Airports {
private:
        std::unordered_set<Airport>
airports;
}
```

```
class Calculation{
public:
        static double toRadians(double degree);
        static double calculateDistance(double
lat1, double lon1, double lat2, double lon2);
```

Graph



- Praticamente mesma estrutura do grafo das aulas
- Não é template
- Cada edge tem um set<string> representando todas as airlines que fazem aquele trajeto

```
class Vertex {
    std::string iata;               // conte
    vector<Edge> adj;  // list of outgoing ed
    bool visited;          // auxiliary field
    bool processing;       // auxiliary field
    int num;
    int low;

    void addEdge(Vertex *dest, double w);
    bool removeEdgeTo(Vertex *d);
public:
    Vertex(string in);
    string getIATA() const;
    void setIATA(const std::string& in);
    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 vector<Edge> &getAdj() const;
    vector<Edge> &getAdj();
    void setAdj(const vector<Edge> &adj);
    void addAdj(const Edge& edge);
    friend class Graph;
};
```

```
class Edge {
    Vertex * dest;  // destination vertex
    double weight;  // edge weight
    set<string> airlines; // airlines that

public:
    Edge(Vertex *d, double w);
    Vertex *getDest() const;
    void setDest(Vertex *dest);
    double getWeight() const;
    void setWeight(double weight);
    set<string> getAirlines() const;
    void addAirline(std::string airline);
    friend class Graph;
    friend class Vertex;
};
```

Graph

- Praticamente mesma estrutura do grafo das aulas
- Não é template
- Alguns métodos adicionados para auxiliar em alguns problemas específicos, por exemplo:
  - getDiameter
  - articulationPoints
  - bfsmaxXstops

```
class Graph {
    vector<Vertex *> vertexSet;    // vertex set
    void dfsVisit(Vertex *v,  vector<string> & res) const;
    bool dfsIsDAG(Vertex *v) const;
public:
    Vertex *findVertex(const string &in) const;
    int getNumVertex() const;
    bool addVertex(const string &in);
    bool removeVertex(const string &in);
    bool addEdge(const string &sourc, const string &dest, double w);
    bool removeEdge(const string &sourc, const string &dest);
    vector<Vertex * > getVertexSet() const;
    vector<string> dfs();
    int getDiameter(Vertex * vertex, vector<string>& lastLevelVertices);
    void dfs_articulationPoints(Vertex *v, stack<Vertex*> &s, unordered_se
    unordered_set<std::string> articulationPoints();
    int countEdges() const;

    unordered_set<std::string> bfsmaxXstops(string airport, int n);

    void setDefaults();
};
```

Model

```
class Model {
        Graph flights;
        Airlines airlines;
        Airports airports;
        Countries countries;
        Cities cities;
}
```

Funcionamento

Desde listagem de aeroportos até à escolha do melhor trajeto entre dois locais.

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Funcionalidades

- Estatísticas Gerais
  - Número de aeroportos
  - Número de voos
  - Número de cidades
  - Número de países
  - Número de companhias aéreas
- Estatísticas de um aeroporto
  - Número de voos
  - Número de companhias aéreas
  - Número de cidades destinos
  - Número de países destino
- Aeroportos alcançáveis em X paragens
  - Listagem e Número total

- Carregamento dos dados do ficheiro
- Desambiguamento de cidades com mesmo nome
- Determinação dos aeroportos com mais movimento aéreo
- Determinação dos aeroportos essenciais
- Determinação da maior viagem
- Escolher aeroporto(s) com base em:
  - Cidade
  - Código IATA
  - Coordenadas Geográficas
- Procurar melhor trajeto entre dois locais
  - Possível inserir filtros
  - Minimizar número de diferentes airlines

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Complexidade Ciclomática Temporal

Estatísticas gerais: Todas $O(1)$ exceto número de voos que é $O(V)$

Informações sobre aeroportos e airlines: $O(1)$

Estatísticas de aeroportos: $O(V)$

unordered_set<string> Graph::bfsmaxXstops(std::string airport, int n) : $O(V + E)$

vector<std::string> Model::highestAirTrafficCapacity(int k): $O(V + E)$

unordered_set<std::string> Model::essentialAirports(): $O(V + E)$

int Model::maximumTrip(list<std::pair<std::string, std::string>>& res): $O(V * (V + E))$

Todos os algoritmos para escolher a melhor viagem possível: $O(V + E)$ mas tem complexidade espacial: $O(V)$

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Interface

```
Insert -1 to quit!
Here are your options.
0 : Repeat Instructions
1 : Global Statistics
2 : Flights Of An Airport (inc. number of flights, number of airlines, number of destinations, )
3 : Flights By City
4 : Flights By Airline
5 : Reachable airports with max of X stops
6 : Maximum trip
7 : The greatest air traffic capacity airport
8 : Essential Airports
9 : Best Flight
10 : Countries Listing
11 : Airlines Listing
12 : Airports Listing

What's your option:
```


U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Exemplo de Utilização

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Em Destaque - Best Flight

Apesar da base deste algoritmo ser um algoritmo de pesquisa, é a funcionalidade menos parecida com aquilo que fizemos nas aulas, devido a ter de escolher o caminho mais curto e apenas selecionando os aeroportos que são úteis à viagem em vez de serem todos os visitados até chegar ao aeroporto que pretendemos chegar.

Resolvemos este problema priorizando o tempo ao espaço. Fizemos uma pesquisa por bfs, mas em vez de apenas guardarmos os Vertex's na fila, guardamos pares, em que o primeiro elemento era o apontador para 1 Vertex e o segundo elemento é o caminho mais curto para chegar a esse Vertex. Desta forma, apesar da complexidade espacial não ser excelente, a temporal não é mais do que a complexidade temporal de 1 pesquisa normal por bfs, ou seja, O(V + E).

Desta forma, todas as bestFlights que o user possa pedir são encontradas de forma quase instantânea e facilitou a implementação das funções relacionadas com filtros.

Principais Dificuldades

Dificuldade em testar as várias funções.

Tarefas de Valorização

De forma a permitir escalabilidade do sistema, criámos uma classe que lida com todo o output do sistema, permitindo que num futuro, o output seja apresentado de uma forma diferente.

Sempre que possível seguimos os princípios SOLID.