

# AED PROJETO 2

FLIGHT MANAGEMENT

Afonso Montenegro Gonçalves Ribeiro da Cruz (up202006020)

David Tavares Simões (up202210329)

Simão José Costa Neri (up202206370)

Class: 2LEIC10

AED: LEIC011

Group: G102

# DIAGRAMA DE CLASSES

O nosso diagrama é composto por um conjunto de 5 classes, as quais desempenham papéis distintos e interagem de maneira coordenada para dar forma à estrutura do projeto

**Classe Menu -> Interface dos menus**

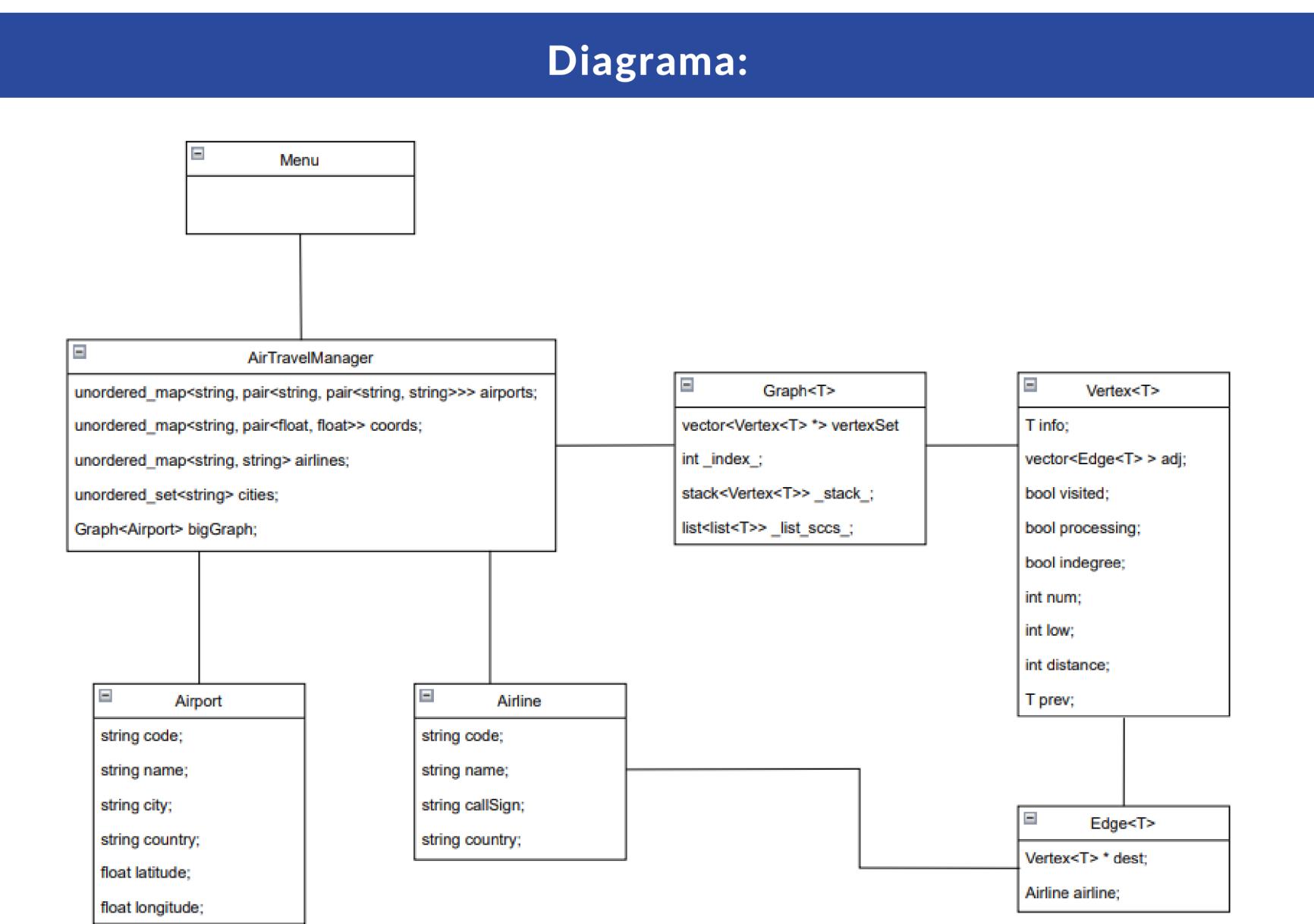
**Classe AirTravelManager -> Criação do nosso grafo e todos os métodos utilizados para desenvolver o projeto**

**Classe Airport -> Armazena os dados de todos os aeroportos**

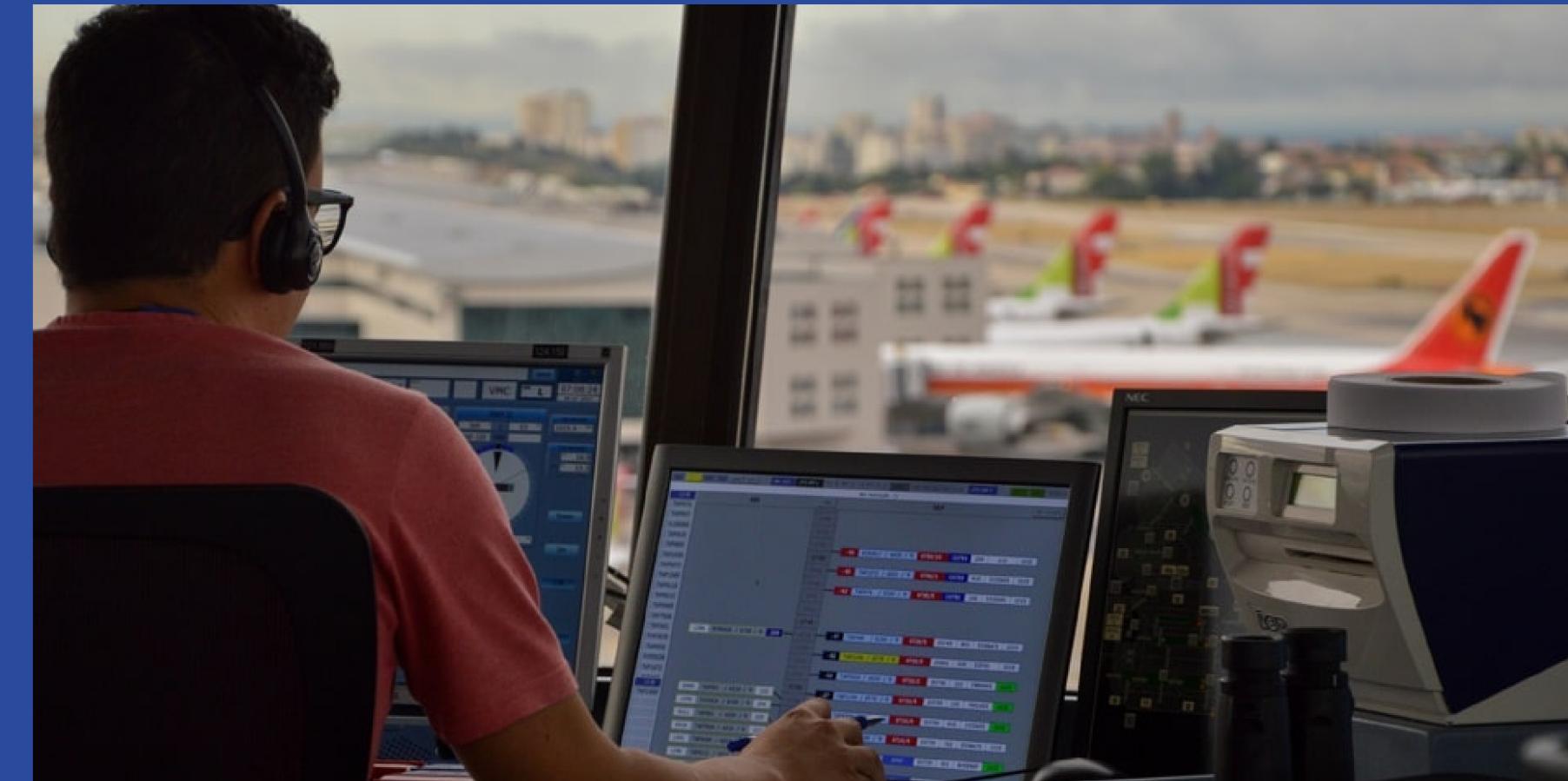
**Classe Airline -> Armazena os dados de todas as companhias aéreas**

**Classe Graph -> Grafo utilizado nas aulas com certas modificações**

**Diagrama:**



# LEITURA DO DATASET



1. A leitura dos ficheiros .csv é feita na classe “AirTravelManager.cpp”.
2. Nesta classe foram implementadas 3 funções: `readAirports()`, `readAirlines()` e `readFlights()`, que lê e analisa os ficheiros `airports.csv`, `airlines.csv` e `flights.csv` respetivamente
3. Todas as funções armazenarão a informação lida em `unordered map` distintos:
  - `readAirports()` armazena os aeroportos, coordenadas e cidades dos mesmos nas estruturas “`airports`”, “`coords`” e “`cities`”. Irá também adicionar os aeroportos como vértices no nosso Graph.
  - `readAirlines()` guarda todas as airlines na estrutura “`airlines`”.
  - `readFlights()` atribuirá nomes às airlines existentes na estrutura “`airlines`” e a informação lida será adicionada como edges do nosso Graph
4. Estas 3 funções são chamadas na classe “`Main.cpp`” para que os ficheiros sejam lidos assim que o programa seja executado.

# BIG GRAPH

- No nosso projeto apenas um grafo é utilizado, o BigGraph, sendo este um objeto da classe Graph.
- Esta classe é semelhante à fornecida nas aulas, sendo dividida em:
  - Vertex;
  - Edge;
  - Graph.
- Apenas 3 mudanças foram feitas no grafo original:
  - 1º- O peso de uma edge é a airline que realiza o voo entre os vértices source e destination dessa edge;
  - 2º- Foi adicionado um campo auxiliar “distance” para armazenar a quantidade de paragens entre 2 aeroportos, de modo a facilitar o cálculo do percurso mais rápido;
  - 3º- Também foi adicionado outro campo auxiliar denominado de “prev”, criado para armazenar o vértice de onde o atual vértice é proveniente.

# ESTRUTURA

```
/// Edge of the graph -> Represents flights(wighted : airlines)
template <class T> class Edge;
/// Graph -> Represents the all Flight System
template <class T> class Graph;
/// Vertex of the graph -> Represents airports
template <class T> class Vertex;
```

```
template <class T>
class Edge {
    Vertex<T> * dest;
    /// Wight of the edge -> Airline responsible for the flight
    Airline airline;
```

```
template <class T>
class Vertex {
    T info;           // contents
    vector<Edge<T> > adj; // list of outgoing edges
    bool visited;     // auxiliary field
    bool processing; // auxiliary field
    int indegree;    // auxiliary field
    int num;          // auxiliary field
    int low;          // auxiliary field
    int distance;    // auxiliary field
    T prev;
```

# FUNCIONALIDADES IMPLEMENTADAS (ESTATÍSTICAS)

**void globalStats() const**

*Imprime o número total de aeroportos, companhias aéreas e voos da Base de Dados*

*Complexidade:  $O(V)$  ->  $V$  = número de aeroportos*

**void airportInfo(const string& airport) const**

*Imprime o número de voos partindo de um aeroporto e o número de companhias aéreas que realizam esses voos (também pode imprimir uma lista de todos os destinos e companhias aéreas)*

*Complexidade:  $O(V + E)$  ->  $V$  = número de aeroportos,  $E$  = número de voos*

**void airlineFlights(const string& airline) const**

*Imprime o número de voos que uma companhia aérea faz e o número de aeroportos de onde essas companhias aéreas voam (também pode imprimir uma lista de todos os voos de origem-destino)*

*Complexidade:  $O(V * (E + A))$  ->  $V$  = número de aeroportos,  $E$  = número de voos,  $A$  = número de airlines*

**void cityFlights(const string& city)**

*Imprime a quantidade de aeroportos da cidade, o total de voos saindo da cidade e a quantidade de companhias aéreas responsáveis por esses voos (pode imprimir uma lista de voos de cada aeroporto da cidade)*

*Complexidade:  $O(V * E)$  ->  $V$  = número de aeroportos,  $E$  = número de voos*

# CONTINUAÇÃO...

**void airportCountries(const string& airport) const**

*Imprime o número de países para os quais podemos viajar a partir do aeroporto escolhido (pode imprimir uma lista de todos os países)*

*Complexidade:  $O(V * E)$  -> V = número de aeroportos, E = número de voos*

**void citiesCountries(const string& city) const**

*Imprime o número de países para os quais podemos viajar a partir da cidade escolhida (pode imprimir uma lista de todos os países)*

*Complexidade:  $O(V * E)$  -> V = número de aeroportos, E = número de voos*

**void airportDestinations(const string& airport) const**

*Imprime o número de países, cidades e aeroportos para os quais você pode voar diretamente do aeroporto escolhido (pode imprimir uma lista com todos os países, cidades e aeroportos)*

*Complexidade:  $O(V * E * \log n)$  -> V = número de aeroportos, E = número de voos*

**void reachable\_destinations(string airport, int stops) const**

*Imprime o número de países, cidades e aeroportos para onde podemos viajar a partir do aeroporto escolhido no número de paradas fornecidas (pode imprimir uma lista com todos os aeroportos, cidades e países)*

*Complexidade:  $O(V * E)$  -> V = número de aeroportos, E = número de voos*

**void maximum\_trip() const**

*Imprime a viagem máxima do Sistema de Viagem e os pares Origem-Destino que estão tão distantes um do outro*

*Complexidade:  $O(V * (V + E))$  -> V = número de aeroportos, E = número de voos*

# FUNCIONALIDADES IMPLEMENTADAS (TRAVEL MANAGER): OPTIONS

**vector<string> cityToAirport(string& city)**

*Encontra e retorna todos os aeroportos existentes numa dada cidade*

*Complexidade: O(n)*

**vector<string> countryToAirport(string& country);**

*Encontra e retorna todos os aeroportos existentes num dado país*

*Complexidade: O(n)*

**vector<string> geoToAirport(string& lat, string& longi);**

*Encontra e retorna o aeroporto mais perto de uma dada coordenada*

*Complexidade: O(n)*

**void findFlights(vector<string> &source, vector<string> &destination,  
vector<string> &air, vector<string> &picky, vector<string> &cit,  
vector<string> &countries)**

*Encontra o melhor caminho entre todas as combinações de aeroportos de origem-destino (se for um país ou cidade) e os imprime*

*Complexidade: O( $n^3$ )*

**vector<string> bestPath(string &source, string &destination)**

*Encontra o melhor caminho entre um aeroporto de origem e um aeroporto de destino*

*Complexidade: O(V + E) -> V = número de aeroportos, E = número de voos*

# FUNCIONALIDADES IMPLEMENTADAS (TRAVEL MANAGER): FILTERS

```
vector<string> bestPathFiltered(string &source, string&  
destination, vector<string> &airlines);
```

Encontra o melhor caminho entre um aeroporto de origem e um aeroporto de destino usando apenas as companhias aéreas desejadas

Complexidade:  $O(V + E)$  ->  $V$  = número de aeroportos,  $E$  = número de voos

```
vector<string> bestPathPickyAirports(string &source, string&  
destination, vector<string> &airports)
```

Encontra o melhor caminho entre um aeroporto de origem e um aeroporto de destino sem parar nesses aeroportos

Complexidade:  $O(V + E)$  ->  $V$  = número de aeroportos,  $E$  = número de voos

```
vector<string> bestPathPickyCities(string &source, string&  
destination, vector<string> &cit)
```

Encontra o melhor caminho entre um aeroporto de origem e um aeroporto de destino sem parar nas referidas cidades

Complexidade:  $O(V + E)$  ->  $V$  = número de aeroportos,  $E$  = número de voos

```
vector<string> bestPathPickyCountries(string &source, string&  
destination, vector<string> &countries)
```

Encontra o melhor caminho entre um aeroporto de origem e um aeroporto de destino sem parar nos referidos países

Complexidade:  $O(V * E)$  ->  $V$  = número de aeroportos,  $E$  = número de voos

```
void findFlightsMin(vector<string> &source, vector<string>  
&destination);
```

Encontra o melhor caminho entre um aeroporto de origem e um aeroporto de destino utilizando o menor número possível de companhias

Complexidade:  $O(V + E)$  ->  $V$  = número de aeroportos,  $E$  = número de voos

# INTERFACE COM O UTILIZADOR

A interface com o utilizador é na sua maioria feita através de menus, tendo sido implementados 6 menus diferentes de modo a que a experiência do utilizador seja simples e direta.

Menu Principal

```
|-----|-----|-----|
|      Find Flights(F) |      Statistics(S) | |
|---|---|---|
| Options:           | Filters:          | Options:        |
|-----|-----|-----|
| Source:            | ->Choose Your Airlines | 1-> Global Information |
| -Airport           |                         | 2-> Airport Flights |
| -City              | ->Minimize Airlines   | 3-> City Flights    |
| -Country            |                         | 4-> Airline Flights  |
| -Coordinates        | ->Avoid Airports     | 5-> Countries By Airport |
|                         |                         | 6-> Countries By City |
| Destination:       | ->Avoid Cities       | 7-> Airport Direct Destinations |
| -Airport           |                         | 8-> Airport Reachable Destinations |
| -City              | ->Avoid Countries    | 9-> Maximum Trip |
| -Country            |                         | 10-> Top Airports In Flights |
| -Coordinates        |                         | 11-> Bottom Airports In Flights |
|                         |                         | 12-> Essential Airports |
|                         |                         | 13-> Strongly Connected Components |
|-----|-----|-----|
|                         |                         | Q->Leave |
|-----|-----|-----|
Do you want to:
--> Find a Flight (F)
--> See Statistics (S)
--> Leave (Q)
```

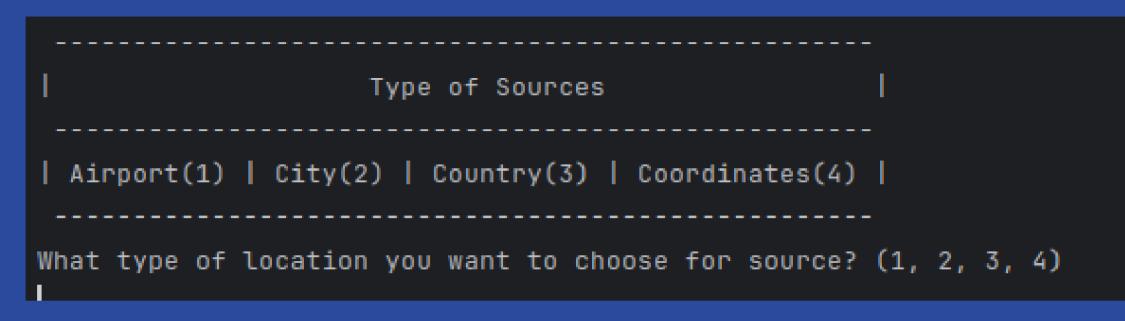
Menu das Estatísticas:

```
Choose One Option:
-----
1-> Global Information;
2-> Airport Flight;
3-> City Flights;
4-> Airline Flights;
5-> Countries By Airport;
6-> Countries By City;
7-> Airport Direct Destinations;
8-> Airport Reachable Destinations;
9-> Maximum Trip;
10-> Top Airports In Flights;
11-> Bottom Airports In Flights;
12-> Essential Airports;
13-> Strongly Connected Components.
```

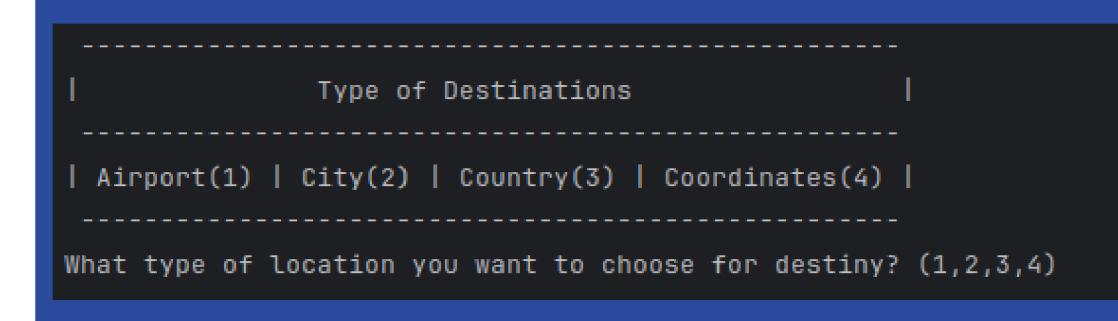
# INTERFACE COM O UTILIZADOR

Ao escolher a opção “Find a Flight (F)”, a fim de encontrar os melhores voos, passamos a aceder a três menus:

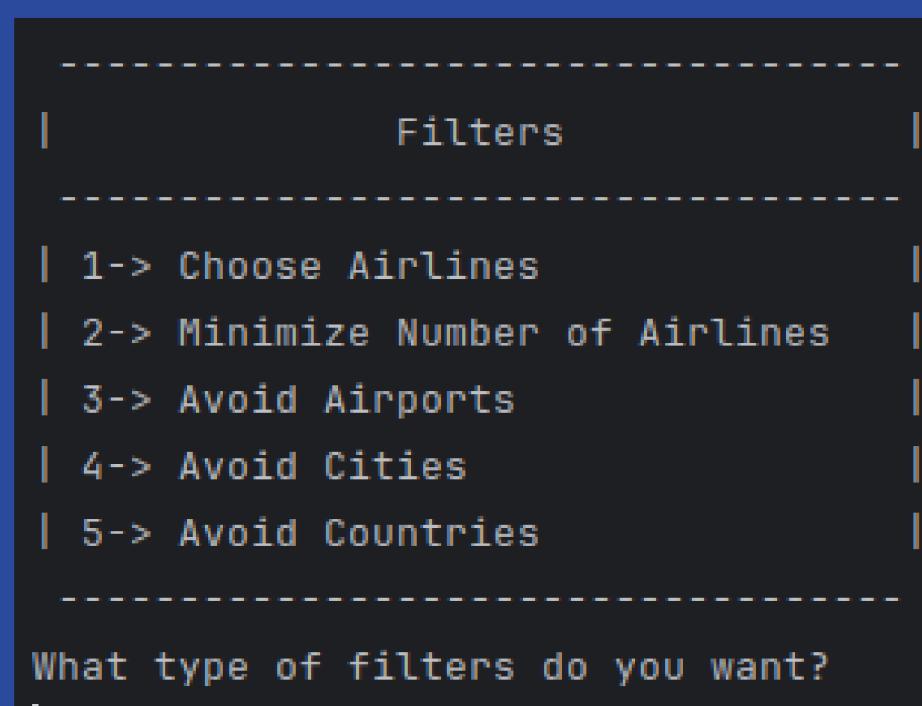
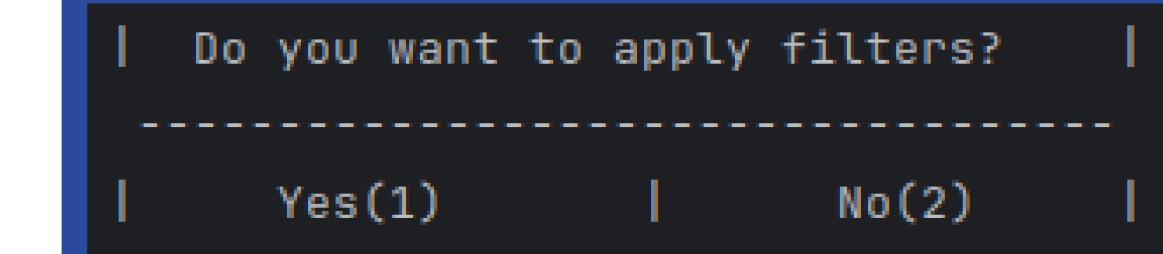
Escolha de Source:



Escolha de Destination:



Possibilidade de Aplicar Filtros:



Por fim, ao escolher “Yes (1)” no menu da possibilidade de aplicar filtros, podemos aceder ao menu de escolhas de filtros:

# DESTAQUE DE FUNCIONALIDADES

- Desenvolvimento de um projeto pautado na criação de classes criteriosamente divididas e estruturadas de maneira sólida e coesa.
- Incorporação de funcionalidades adicionais que elevam o nível de desenvolvimento do projeto, conferindo-lhe assim uma completude notável.
- Desenvolvimento e execução de um Menu meticulosamente bem organizado, proporcionando uma leitura fluida e acessível para os usuários.



# DIFICULDADES ENCONTRADAS

- No decorrer do projeto, cada integrante do grupo desempenhou um papel ativo e significativo, contribuindo de maneira efetiva para a sua conclusão.
- Conforme avançávamos por cada fase do projeto, deparamo-nos com desafios em constante ascensão, tendo sido a aprendizagem, bem como a manipulação de articulation points e strongly connected components a nossa tarefa mais complexa e exigente.



**FIM**

