Sistema de Gestão de Voos

António Santos (up202205469)
Vanessa Queirós (up202207919)
Vasco Costa (up202109923)

Classes Utilizadas

Airline

- Identifica uma companhia aérea.
- Representa dados de airlines.csv.

Airport

- Identifica um aeroporto.
- Representa dados de airports.csv.

Graph

- Classe das aulas práticas ligeiramente modificada.
- Representa dados de flights.csv.

FlightManager

- Contém todos os dados necessários para fazer todas as operações.
- Classe principal.

Application

- Gere as várias opções do menu e chama as funções respetivas do FlightManager.
- Verifica e processa os inputs do utilizador.

Leitura de Dados

- Leitura feita através dos métodos:
  - `parseData()` -> `processAirlines(), processAirports(), processFlights()`

Tipo	Nome	Dados extraídos a partir de
Graph<Airport*>	airportNetwork	airlines.csv, airports.csv, flights.csv
unordered_map	airlineMap	airlines.csv
unordered_map	airportMap	airports.csv
... (*)		

* Foram utilizadas mais estruturas para aumentar o desempenho de certos métodos

Leitura de Dados

Estruturas de Dados

- unordered_map (*)

Método	Complexidade
at()	O(1)
size()	O(1)

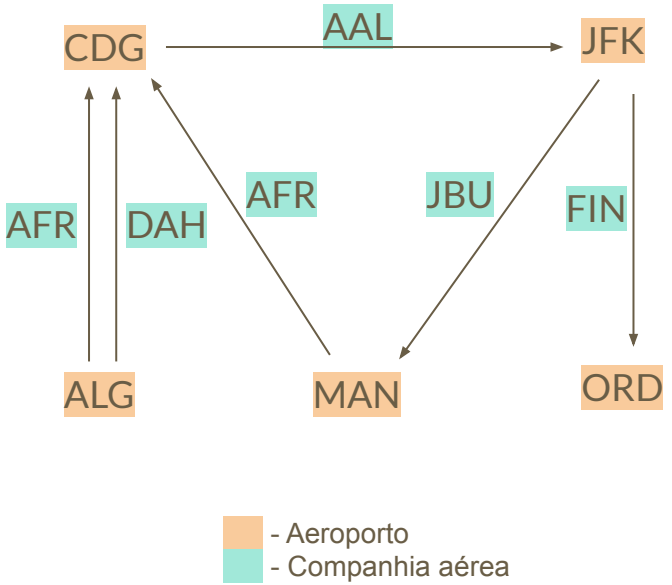
- Graph<T>

Método	Complexidade
findVertex()	O(V)
addVertex()	O(1)
calculateDistance()	O(1)

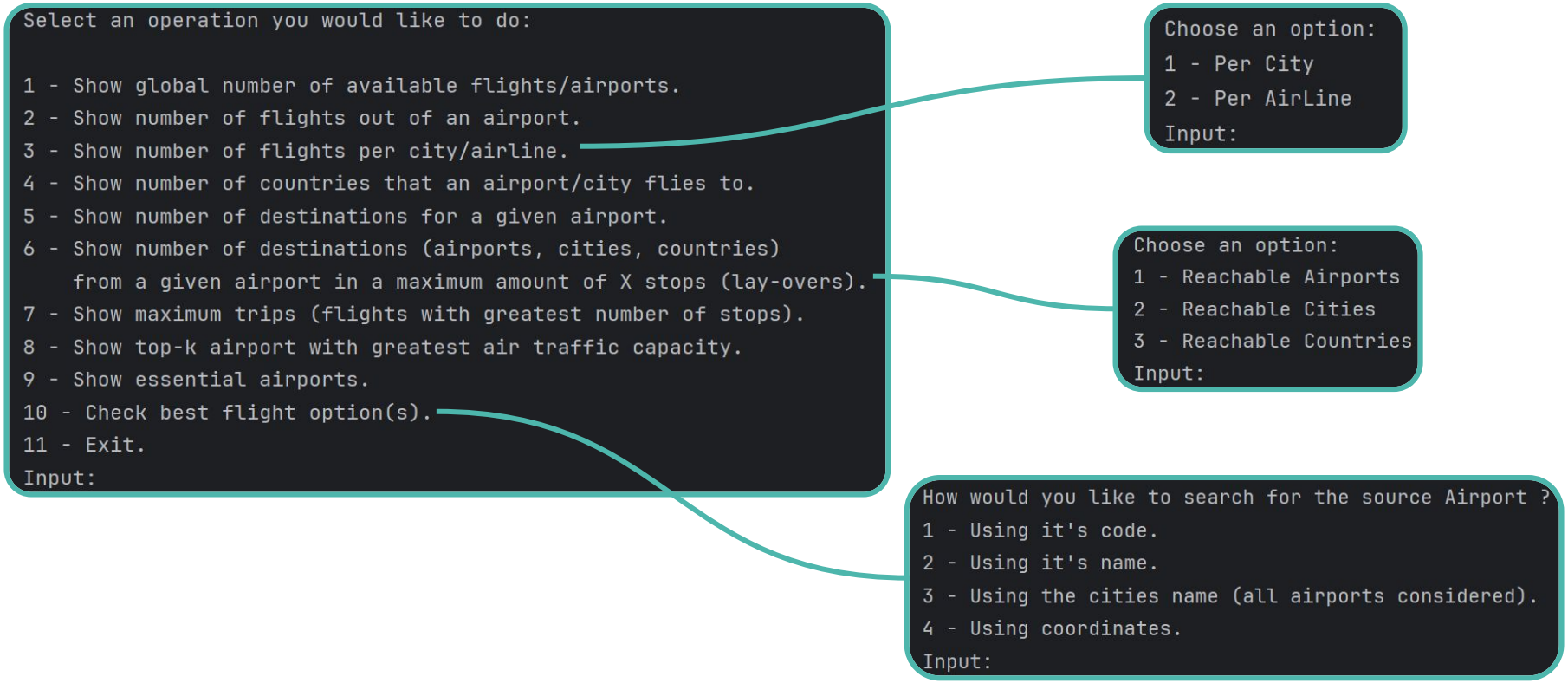
* Utilizado para termos complexidade constante na pesquisa de aeroportos e cidades

Grafo Utilizado

- ➔ Instanciado como Graph<Airport*>
- ➔ Atributos adicionados a Graph.h:
  - ◆ `Airline`: segundo weight para uma edge.
- ➔ Métodos adicionados a Graph.h:
  - ◆ `dfs_art()` para determinar o número de pontos de articulação no grafo.
  - ◆ `calculateDistance()` para calcular distância entre dois pontos a partir de coordenadas.



Menu Interativo



Funcionalidades Implementadas

Estatísticas da Rede Aérea

Alguns exemplos de estatísticas que o programa consegue apresentar:

- Número global de aeroportos e voos disponíveis
- Número de países para os quais um aeroporto/cidade voa.
- Viagem máxima
- Aeroportos essenciais

Funcionalidades Implementadas

Número global de aeroportos e voos disponíveis

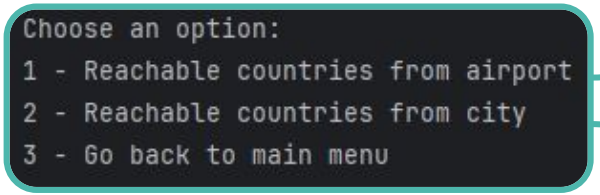
Implementamos um BFS para descobrir o número global de voos e aeroportos, garantindo assim que visitamos todos os aeroportos e consequemente todos os voos.

```
for (auto v :Vertex<Airport *> * : airportNetwork.getVertexSet()) {
    if (!v->isVisited()) {
        queue<Vertex<Airport*> *> q;
        q.push( x: v);
        v->setVisited( v: true);
        while (!q.empty()) {
            auto vertex :Vertex<Airport *> * = q.front();
            numAirports++;
            numFlights += vertex->getAdj().size();

            q.pop();
            for (auto & e :const Edge<Airport *> & : vertex->getAdj()) {
                auto w :Vertex<Airport *> * = e.getDest();
                if ( ! w->isVisited() ) {
                    q.push( x: w);
                    w->setVisited( v: true);
                }
            }
        }
    }
}
```

Funcionalidades Implementadas

Número de países para os quais um aeroporto/cidade voa



`printNumCountriesAirport()`: utiliza a estrutura `airportMap`, que mapeia os aeroportos aos seus códigos.

`printNumCountriesCity()`: utiliza a estrutura `airportsCityMap`, que mapeia os aeroportos a uma determinada cidade.

Funcionalidades Implementadas

Apresentar a melhor opção de voo

```
How would you like to search for the source Airport ?
1 - Using it's code.
2 - Using it's name.
3 - Using the cities name (all airports considered).
4 - Using coordinates.
Input:
```

```
How would you like to search for the target Airport ?
1 - Using it's code.
2 - Using it's name.
3 - Using the cities name (all airports considered).
4 - Using coordinates.
Input:
```

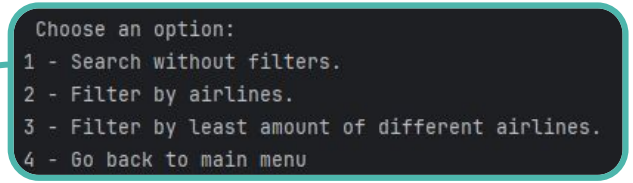
Combinações disponíveis

```
pair<vector<Vertex<Airport*>*>, vector<Vertex<Airport*>*>> converted = flightManager.getConvertedVertexesFromUser(input1, input2, input3, input4, input5, input6, radius);
```

Funcionalidades Implementadas

Apresentar a melhor opção de voo

`printFlightOptionAirlineFiltered(..., bool ignoreFilter)`: utiliza BFS modificado permitindo não só, a visita de aeroportos nivel a nivel, como também guardar todos os caminhos de `source -> target`. Seguidamente, filtra esses caminhos de forma a atingir as especificações requisitadas pelo utilizador.



Funcionalidades Implementadas

Apresentar a melhor opção de voo

`printFlightOptionMinimalAirlines(...)`: utiliza BFS modificado capaz de guardar os diferentes caminhos percorridos e as airlines usadas em cada voo. Posteriormente, filtra os caminhos obtidos de forma a retornar o caminho que usa menos airlines e atinge o `target`.

```
Choose an option:
1 - Search without filters.
2 - Filter by airlines.
3 - Filter by least amount of different airlines.
4 - Go back to main menu
```

```
struct bfsPath {
    std::vector<Vertex<Airport*>*> path;
    std::unordered_set<Airline*> airlines;
    int numAirlines = 0;
};
```

Funcionalidades a Destacar

Show Top-k airport:

- Passamos de uma complexidade de O(V*(V+E)) para O(V)
- Utilizamos um map para associar o número de tráfego a cada aeroporto
- Pode haver vários aeroportos em 5º lugar, por exemplo

```
for (auto vertex : Vertex<Airport *> * : airportNetwork.getVertexSet()){
    unsigned long numFlights = 0;
    numFlights += vertex->getAdj().size();
    numFlights += vertex->getIndegree();
    airportTraffic[numFlights].push_back(vertex->getInfo());
}
```

parte que calcula o número de tráfego

Dificuldades Encontradas

- Diminuir complexidades
- Encontrar as estruturas de dados certas para a pesquisa de aeroportos e cidade constante
- Ambiguidades na descrição do projeto
- Fazer cópia do grafo

Esforço de cada elemento:

António Santos: 100%

Vanessa Queirós: 100%

Vasco Costa: 100%