Transportes Aéreos

Algoritmos e Estruturas de Dados, 2022/23

Elementos do grupo:

Carolina Viana - up202108802
Guilherme Monteiro - up202108668
Sofia Sá - up202108676

As classes do nosso projeto e os seus atributos

O nosso projeto envolveu as seguintes classes:

- Airport (código, nome, cidade e coordenada)
- City (nome, país, vetor de aeroportos e uma struct 'Country' com nome e um set de cidades)
- Airline (código, nome, callsign e country)
- Coordinate (latitude e longitude)
- Graph (struct Node - aeroporto, lista de Edges, um booleano visited, distancia e um pai -, struct Edge - destino e airline -, um "n" que corresponde ao número de nós do grafo, um booleano "hasDir", um unordered_map de cŏdigos de aeroportos e respetivos nós e um vetor "paths" relativamente aos diferentes caminhos para chegar a um dado destino)
- Data (unordered maps de aeroportos, cidades, países, airlines,, o nosso grafo com os voos e um vetor de pares de aeroporto e respetiva coordenadas)

- ApMethods (com uma Data e um Graph, a partir das quais efetuamos certas contagens relativas a um aeroporto e aos respetivos voos disponíveis)
- Statistics (onde são implementadas certos métodos da classe Data que consideramos serem estatística)
- Interface (com uma Data e um objeto de ApMethods, onde são criados vários menus para uma boa utilização dos nossos métodos por parte do user)

A maioria dos atributos das nossas classes são strings, uma vez que se tratam dos códigos dos mesmos e foi algo que ajudou na construção dos algoritmos e das estruturas de dados.

Descrição da leitura do dataset a partir dos ficheiros dados

Feita a partir de 3 funções.
Todas funcionam nos mesmos princípios:

1. filestream com o ficheiro
2. reparti-la em linhas
3. repartir cada linha em tokens, colocando-os num vetor
4. atribuir cada elemento do vetor à respetiva variável
5. guardar a informação da maneira correta no devido container.

As funções são:

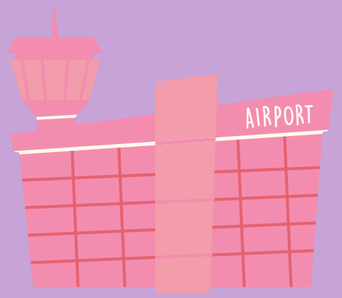
- readFile_airlines -> cria os objetos do tipo Airline

```
29  void Data :: readFile_airlines() {
30
31      //variables
32      string code, name, callSign, countryName;
33      vector <string> v;
34      //open file
35      ifstream input( s: AIRLINES);
36      if (input.is_open()) {
37
38          string line;
39          getline( &: input,  &: line); //skips first line
40          while (getline( &: input,  &: line)) {
41              stringstream iss( str: line);
42
43              //divide line in tokens;
44              string token;
45              while (getline( &: iss,  &: token,  delim: ',')) { v.push_back(token); }
46
47              //assign tokens to the correct variables
48              code = v[0];
49              name = v[1];
50              callSign = v[2];
51              countryName = v[3];
52              v.clear(); //O(n)
53              Airline *airline = new Airline(code, name,  callsign: callSign,  country: countryName);
54              airlines_.insert( x: { &: code, &: airline});
55
56          }
57      } else cout << "Could not open the file\n";
58  }
```

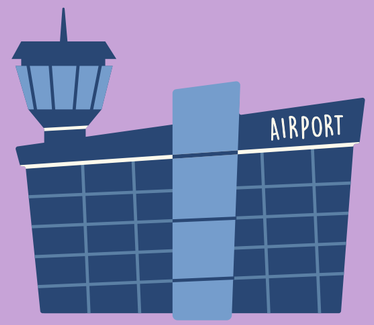
Descrição da leitura do dataset a partir dos ficheiros dados

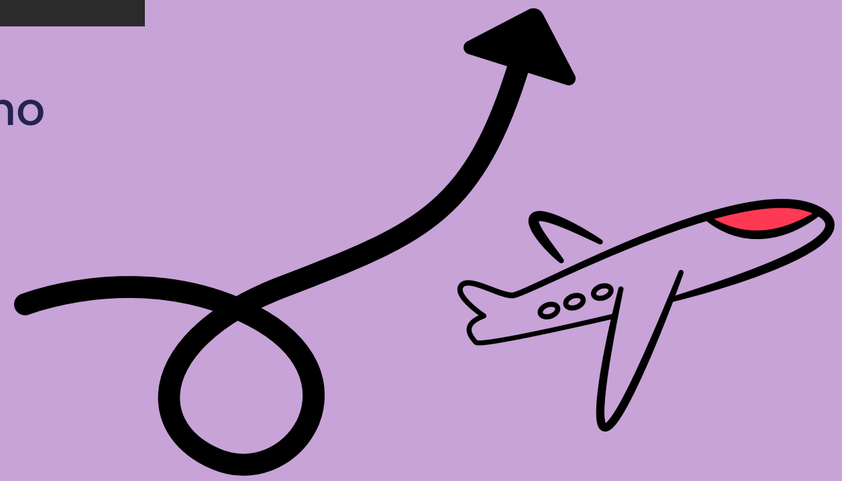
- readFile_airports -> cria os objetos dos tipo Airport, City, Country

```
    City *c = new City( name: city,  country: countryName);
    Airport *airport =new Airport(code, name, city,  lati: latitude,  longi: longitude);
    string key = city + ',' + countryName;
    cities_.insert( x: { &: key,  &: c});
    countries_.insert( x: { &: countryName, y: new Country{ .name_: countryName,  .cities_: {}}});
    cities_[key]->addAirport( ap: code);
    countries_[countryName]->cities_.insert( x: key);
    airports_.insert( x: { &: code,  &: airport});
    airportCoord_.push_back({ &: code,  y: Coordinate( lati: latitude,  longi: longitude)});
}
```



- readFile_flights -> cria o grafo que contém os aeroportos, como nós, e os voos como arestas





O nosso Grafo

Node: - Pointer para objeto Airport
- Lista de Edges
- booleana visited
- inteiro distance
- string parent
- inteiro num
- inteiro low
- booleana stacked

```
} */
class Graph {
    /**...*/
    struct Edge {
        string dest;
        string airline;
    };

    /**...*/
    struct Node {
        Airport* airport;
        list<Edge> adj; // The list of outgoing edges (to adjacent nodes)
        bool visited; // As the node been visited on a search?
        int distance;
        string parent;
        int num;
        int low;
        bool stacked;
    };

    int n;
    bool hasDir; // false: undirect; true: directed
    unordered_map<string, Node> nodes;
    vector<vector<string>> paths = {}; //caminhos para chegar ao destino

public:
```

Atributos auxiliares utilizados em funções:

visited: se o nó já foi visitado naquela travessia
distance: nº de edges até um certo nó
parent: nó que ligou ao nó atual
num e low: utilizados na computação de articulation points
stacked: utilizado na computação de articulation points

O nosso Grafo

Edge: - nome do Aeroporto de destino
- nome da companhia aérea desse voo

unordered_map nodes:

set de todos os nodes do grafo.
Fazemos a associação entre o código do aeroporto e o Nó que contém esse aeroporto.
Isto permite encontrarmos o Nó de um determinado aeroporto utilizando apenas a string do seu código.

hasDir:

true - é um grafo dirigido

paths:

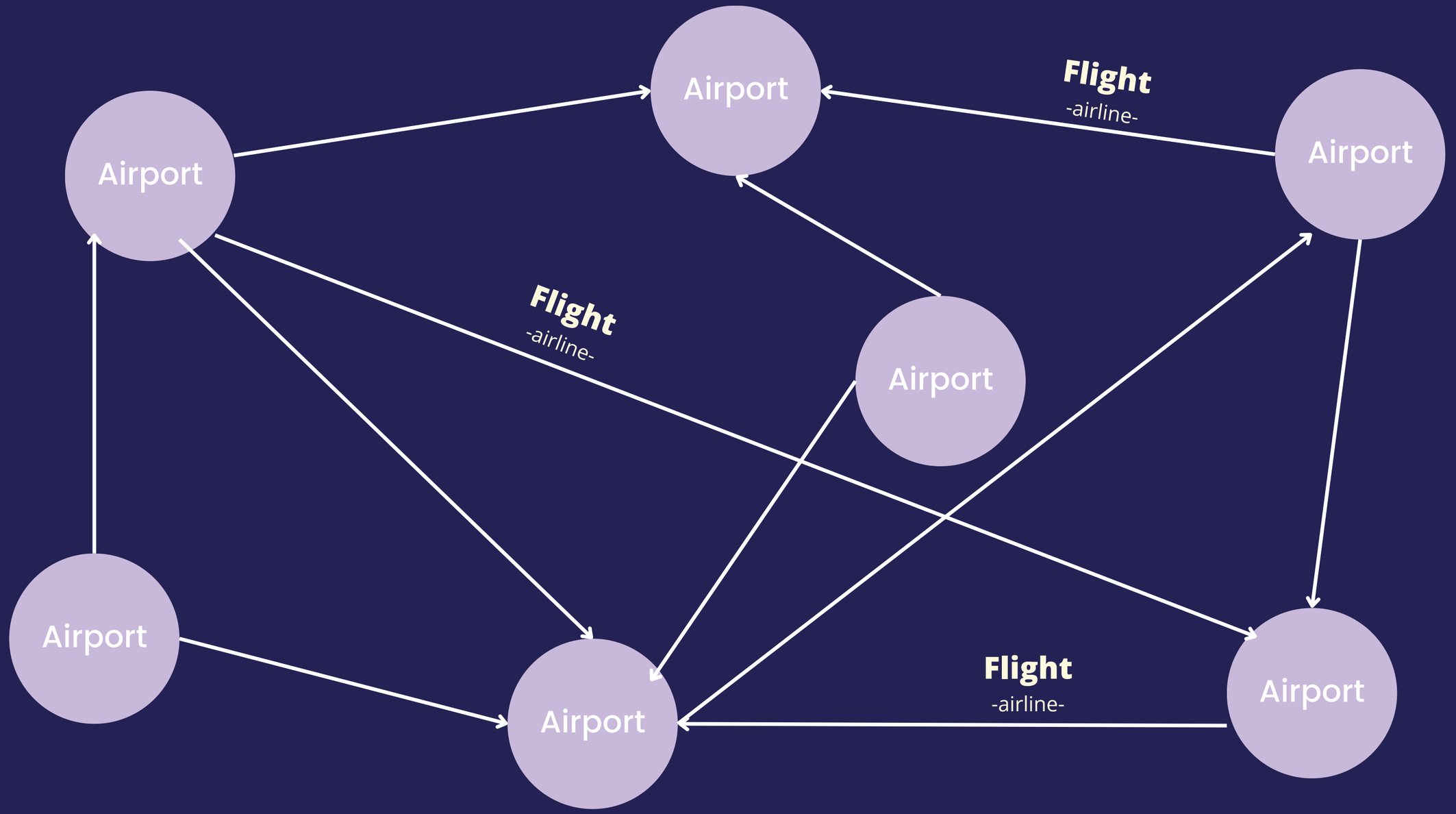
utilizado em funções auxiliares,

```
} */
class Graph {
    /**...*/
    struct Edge {
        string dest;
        string airline;
    };

    /**...*/
    struct Node {
        Airport* airport;
        list<Edge> adj; // The list of outgoing edges (to adjacent nodes)
        bool visited; // As the node been visited on a search?
        int distance;
        string parent;
        int num;
        int low;
        bool stacked;
    };

    int n;
    bool hasDir; // false: undirect; true: directed
    unordered_map<string, Node> nodes;
    vector<vector<string>> paths = {}; //caminhos para chegar ao destino

public:
```


Airport
Airport
Flight
-airline-
Airport
Flight
-airline-
Airport
Airport
Airport
Flight
-airline-
Airport

Melhor maneira de voar de um local para outro

Com várias interpretações de Origem/Destino

Os métodos que criámos respondem às seguintes questões:

- Indicar o percurso que corresponde à melhor maneira de voar de um locar para outro
- Local:
  - Aeroporto
  - Cidade
  - País
  - Coordenada
  - Coordenada + distância máx.
- Rede de voos:
  - qualquer companhia aérea
  - filtrar por companhia aérea (uma ou mais)

Para o conseguir procuramos:

1. Transformar uma cidade, país e coordenada num ou mais aeroportos (city2Airport (O(1)) , country2Airport (O(n^2)), coord2Airport (O(n)), coord2AirportWithDistance (O(n))); não necessário se já partirmos de um aeroporto
2. Construir o vetor de filtros (companhias aéreas permitidas)
3. Percorrer o grafo através de uma BFS, partindo do aeroporto original e tendo em conta os filtros (bfs, bfsWithFilters) (O(V+E))
4. Construir o percurso a seguir (makePath (O(V+E)))

Este algoritmo tem a complexidade O(n^2(V+E))

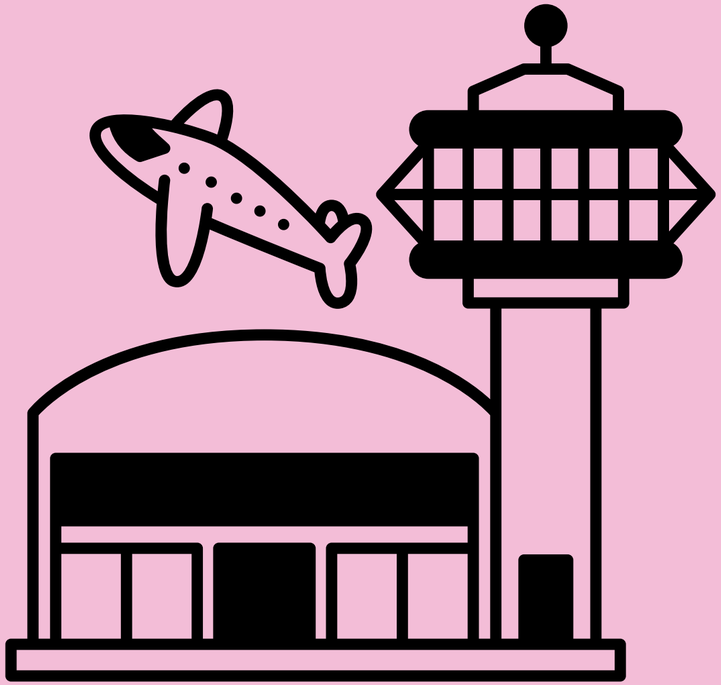
Error Handling

Interface que filtra possíveis erros do utilizador

Inicialmente, após desenvolvermos a Interface que iríamos utilizar, verificamos que tínhamos um problema: o programa crashava se o utilizador desse um input incorreto.

Corrigimos o problema de forma simples e eloquente, fazendo com que a interface verificasse sempre que o input estava correto (verificava se existia, por exemplo, o aeroporto dado no nosso map de aeroportos). Se o input não existisse, o programa avisa o utilizador e deixa o tentar de novo.

Como utilizamos principalmente unordered maps para guardar os nossos dados, esta verificação é suportada em tempo constante - O(1).



Airport Methods (ApMethods)

Classe criada para o desenvolvimento de métodos relacionados apenas com um dado aeroporto.

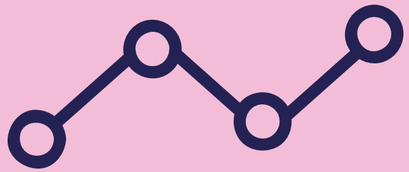
Os métodos que criámos respondem às seguintes questões:

- Nr. de voos e companhias aéreas que partem a partir de um dado aeroporto (nFlightsAirport, nAirlines com complexidade O(1) e O(n))
- Quantas cidades e países são possíveis de alcançar a partir desse aeroporto (nCities, nCountries - O(n))
- Quantos aeroportos, cidades ou países são atingíveis usando um máximo de Y voos? (nAirportsWithMaxFlights, nCitiesWithMaxFlights, nCountriesWithMaxFlights - O(|V|+|E|))

Para estas funções, iteramos sobretudo pelos:

- Edges do nosso grafo, que continham a informação sobre os destinos;
- Nodes, com as informações da origem, pois estes contêm uma lista de edges com os destinos e companhias aéreas. Na classe Graph, também foi implementado um "apMethodsHelper" que fornece a esta classe um set de códigos de aeroportos e ajuda nas contagens de certos métodos.

Estatísticas



Grupo de métodos relacionados com as estatísticas da rede

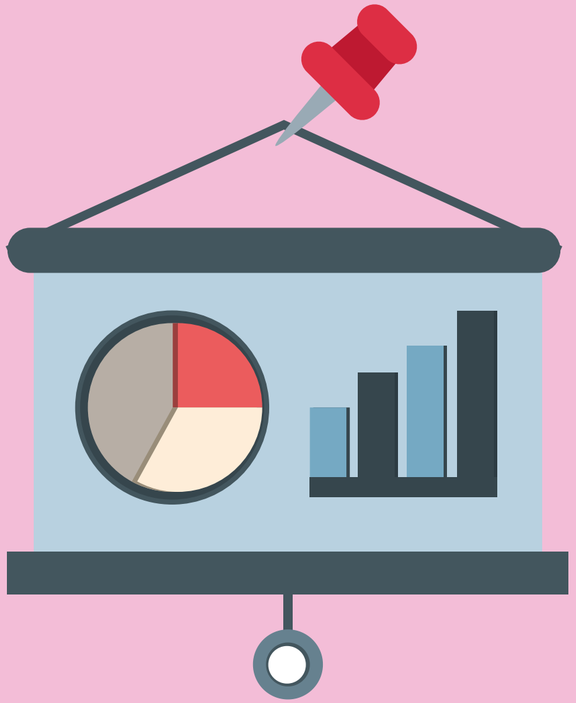
A nível global:
- Nr. de aeroportos - O(1)
- Nr. de cidades - O(1)
- Nr. de países - O(1)
- Nr. de airlines - O(1)
- Nr. de voos - O(1)

A nível de país:
- Nr. de cidades - O(1)
- Nr. de aeroportos - O(n^2)
- Nr. de airlines - O(n^2log(n))
- Nr. de voos - O(n^2)
- Nr. de destinos - O(n^2log(n))

A nível de cidade:
- Nr. de aeroportos - O(1)
- Nr. de airlines - O(n^2log(n))
- Nr. de voos - O(n)
- Nr. de destinos - O(n^2log(n))

A nível de aeroporto:
- Nr. de airlines - O(n log(n))
- Nr. de voos - O(1)
- Nr. de destinos - O(n log(n))

A nível de airline:
- Nr. de voos - O(n^2)
- Nr. de destinos - O(n^2)



outras estatísticas:
- top k aeroportos, em função do nr. de voos - O(n log(n))
- diâmetro - O(n^2(|V|+|E|))
- articulation points - O(|V| * (|V|+|E|))

Interface

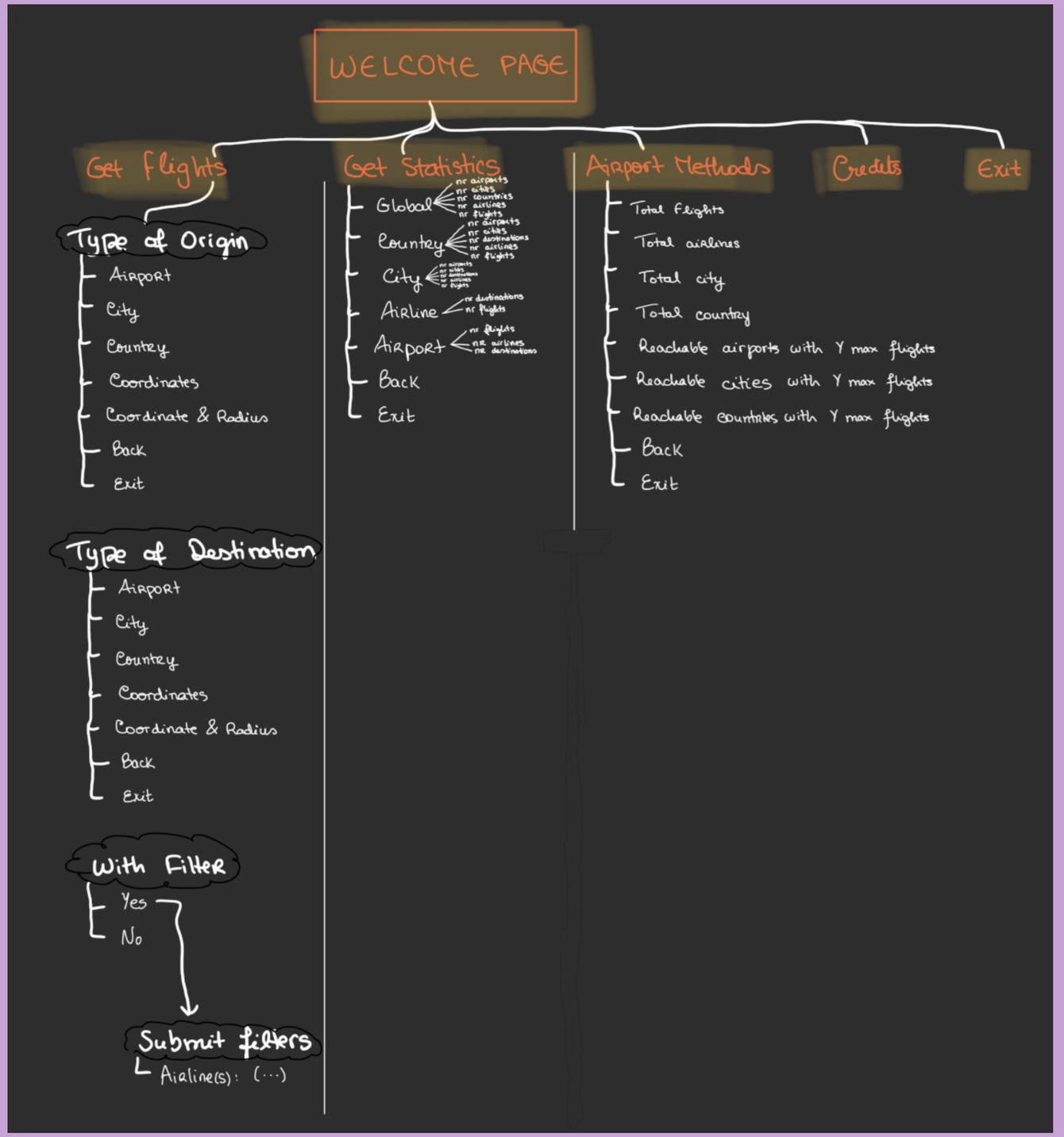
O nosso objetivo era desenvolver e construir uma interface simples e segura.

Esta apenas permite ao utilizador colocar os inputs corretos, de modo a que o programa os consiga interpretar e fornecer as respostas necessárias.

Com esta interface, queremos promover a melhor experiência possível. Para isso, a implementação foi bastante semelhante ao projeto anterior (mantendo também a consistência do nosso trabalho enquanto "developers").

Possui 9 menus com diversas opções, uma página de saída e a página de créditos. No próximo slide é apresentado um esquema de menus.

Esquema da interface do nosso projeto:



Destaque de funcionalidades

O nosso programa apresenta inúmeras funcionalidades que permitem ao utilizador ter uma experiência bastante completa, objetiva e esclarecedora.

Para nós, esse é um dos destaques do nosso trabalho, pois com a utilização do nosso grafo (colocando os dados dos voos e criando métodos sobre o grafo), conseguimos fornecer estatísticas, contagens e opções com filtragem de companhias aéreas.

Estamos bastante orgulhosos com a implementação dos nossos algoritmos e achámos que aprendemos bastante ao trabalhar com a nova estrutura de dados: o grafo.

Para além disso, preocupamo-nos em implementar algumas features "extra", que nos permitiram ir mais além do estritamente necessário. Referimo-nos ao cálculo do diâmetro do grafo, articulation points (algoritmo que gerou alguns bugs e deu particular trabalho), topK aeroportos com mais voos e muitas mais estatísticas do que as mencionadas no enunciado.

Assim, neste projeto tivemos especial cuidado em não ficar apenas pelo obrigatório, mas sim implementar mais algoritmos mais desafiantes. Estamos bastantes orgulhosos disso.

As dificuldades encontradas:

- Gestão de tempo: Fazer um projeto durante as férias - conseguir aproveitar o Natal e o Ano Novo, estudar para os exames que vamos ter e desenvolver o projeto integralmente

- Na eventualidade de haver vários percursos com o mesmo número de voos, apresentar as várias opções, ou considerar um meio de desempate - tentamos, no entanto não conseguimos implementar esta funcionalidade extra.

- Diametro do grafo - função com complexidade temporal muito alta, não conseguimos tornar mais eficiente.

- Articulation Points - bugs demorosos de resolver
-
- Implementação das funções na interface. Resolução de bugs recorrentes e aprendizagem com os mesmos. Alguns deles: erros de compilação de redefinição de funções (problema de "includes");

- Nome de cidades repetidos e, no caso dos EUA, sem forma de as distinguir - Gerou bastantes bugs difíceis de detetar, todavia conseguimos resolvê-los

Esforço de cada elemento do grupo:

- Trabalho bem repartido entre 3 colegas "amigos" e atribuição de tarefas consoante o conforto e fluidez de trabalho de cada elemento em cada uma delas.

Fim!

Obrigada pela atenção :)

07/01/2023