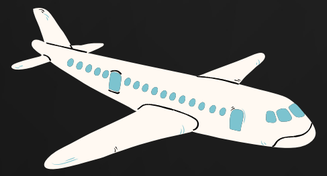
TRANSPORTES AÉREOS



Algoritmos e Estruturas de Dados

Elementos do Grupo:

Ana Sofia Baptista - up202207334
Eduardo Santos - up202207521
Pedro Pedro - up202206961

Classes

- Airline - code, name, callSign, country
- Airport - code, name, city, country, coordinate
- City - name, country, unordered set de aeroportos
- Coordinate - latitude, longitude
- Country - name, set de cities
- Graph - (class Vertex - aeroporto, lista de Edges, boolenano visited, booleano processing, parent), (class Edge - destino, airline)
- DataManip - unordered map de aeroportos, unordered map de airlines, unordered map de cities, unordered map de countries
- Menu - com um Data, onde são criados vários menus para utilização do user

Descrição da leitura do dataset

A partir dos ficheiros dados, a leitura dos mesmos foi feita com apenas 3 funções, funcionando todas do mesmo modo:

1. filestream com o ficheiro;
2. são criados objetos istringstream para cada linha lida, o que facilita a leitura dos diferentes campos separados por vírgulas;
3. é criado dinamicamente um objeto com os parâmetros lidos e é guardada essa informação no respetivo container.

```
void DataManip::readAirlines() {

    ifstream in( s: "../dataset/airlines.csv");
    string line, code, name, callSign, countryName;
    getline( &: in, &: line);

    if (in.is_open()) {

        while(getline( &: in, &: line)){

            istringstream iss( str: line);

            getline( &: iss, &: code, delim: ',');
            getline( &: iss, &: name, delim: ',');
            getline( &: iss, &: callSign, delim: ',');
            getline( &: iss, &: countryName, delim: ',');

            Airline *airline = new Airline(code, name, callSign, country: countryName);
            airlines_.insert( x: { &: code, &: airline});

        }

    } else cout << "Could not open the file\n";

}
```

Descrição da leitura do dataset

```cpp
void DataManip::readAirports() {

    ifstream in( s: "../dataset/airports.csv");
    string line, code, name, city, country;
    double latitude, longitude;
    getline( &: in,  &: line);

    if (in.is_open()) {

        while(getline( &: in,  &: line)){

            istringstream iss( str: line);

            getline( &: iss,  &: code,  delim: ',');
            getline( &: iss,  &: name,  delim: ',');
            getline( &: iss,  &: city,  delim: ',');
            getline( &: iss,  &: country,  delim: ',');
            iss >> latitude;
            iss.ignore();
            iss >> longitude;

            auto findCities :iterator<...> = cities_.find( x: city + "," + country);
            if(findCities!=cities_.end()){
                findCities->second->addAirport( airportCode: code);
            }
            else{
                City *city_ = new City( &: city,  &: country);

                city_->addAirport( airportCode: code);
                cities_.insert( x: { x: city + "," + country, &: city_});
            }

            Airport *airport = new Airport( &: code,  &: name,  &: city,  &: country, latitude, lo
            airports_.insert( x: { &: code,  &: airport});
            countries_.insert( x: { &: country,  y: new Country( &: country)});
```

```cpp
void DataManip::readFlights() {

    ifstream in( s: "../dataset/flights.csv");
    string line, source, target, airline;
    getline( &: in,  &: line);

    for (auto it :iterator<...> = airports_.begin(); it != airports_.end(); it++){
        graph_.addVertex( in: it->second);
        graph_.findVertex( airportCode: it->second->getCode())->setIndegree(0);
        graph_.findVertex( airportCode: it->second->getCode())->setOutdegree(0);
    }

    if (in.is_open()) {

        while(getline( &: in,  &: line)){

            istringstream iss( str: line);

            getline( &: iss,  &: source,  delim: ',');
            getline( &: iss,  &: target,  delim: ',');
            getline( &: iss,  &: airline,  delim: ',');

            Vertex* sourceVertex=graph_.findVertex( airportCode: source);
            Vertex* targetVertex=graph_.findVertex( airportCode: target);

            sourceVertex->setOutdegree(sourceVertex->getOutdegree()+1);
            targetVertex->setIndegree(targetVertex->getIndegree()+1);

            graph_.addEdge( sourcCode: source,  destCode: target, airline);
        }
```

Grafo

Vertex:

- Pointer para objeto airport
- Vetor de Edges
- String parent
- Booleana visited: para ver se o vértice já foi visitado
- Booleana processing
- Inteiro indegree
- Inteiro outdegree
- Inteiro num: usado nos articulation points
- Inteiro distance: para ver o nº de edges até um certo vértice
- Inteiro low: usado nos articulation points
- Void addEdge

```
class Vertex {
    Airport* airport;                // contents
    vector<Edge> adj;
    string parent;          // list of outgoing edges
    bool visited;           // auxiliary field
    bool processing;        // auxiliary field
    int indegree;           // auxiliary field
    int outdegree;          // auxiliary field
    int num;                // auxiliary field
    int distance;            // auxiliary field
    int low;                // auxiliary field

    void addEdge(Vertex *dest, string airline);
    //bool removeEdgeTo(Vertex *d);
public:
    Vertex(Airport* in);
    Airport* getAirport() const;
    void setAirport(Airport* in);
    bool isVisited() const;
    void setVisited(bool v);
    bool isProcessing() const;
    void setProcessing(bool p);
    const vector<Edge> &getAdj() const;
    //void setAdj(const vector<Edge> &adj);

    int getIndegree() const;

    void setIndegree(int indegree);

    int getOutdegree() const;
```

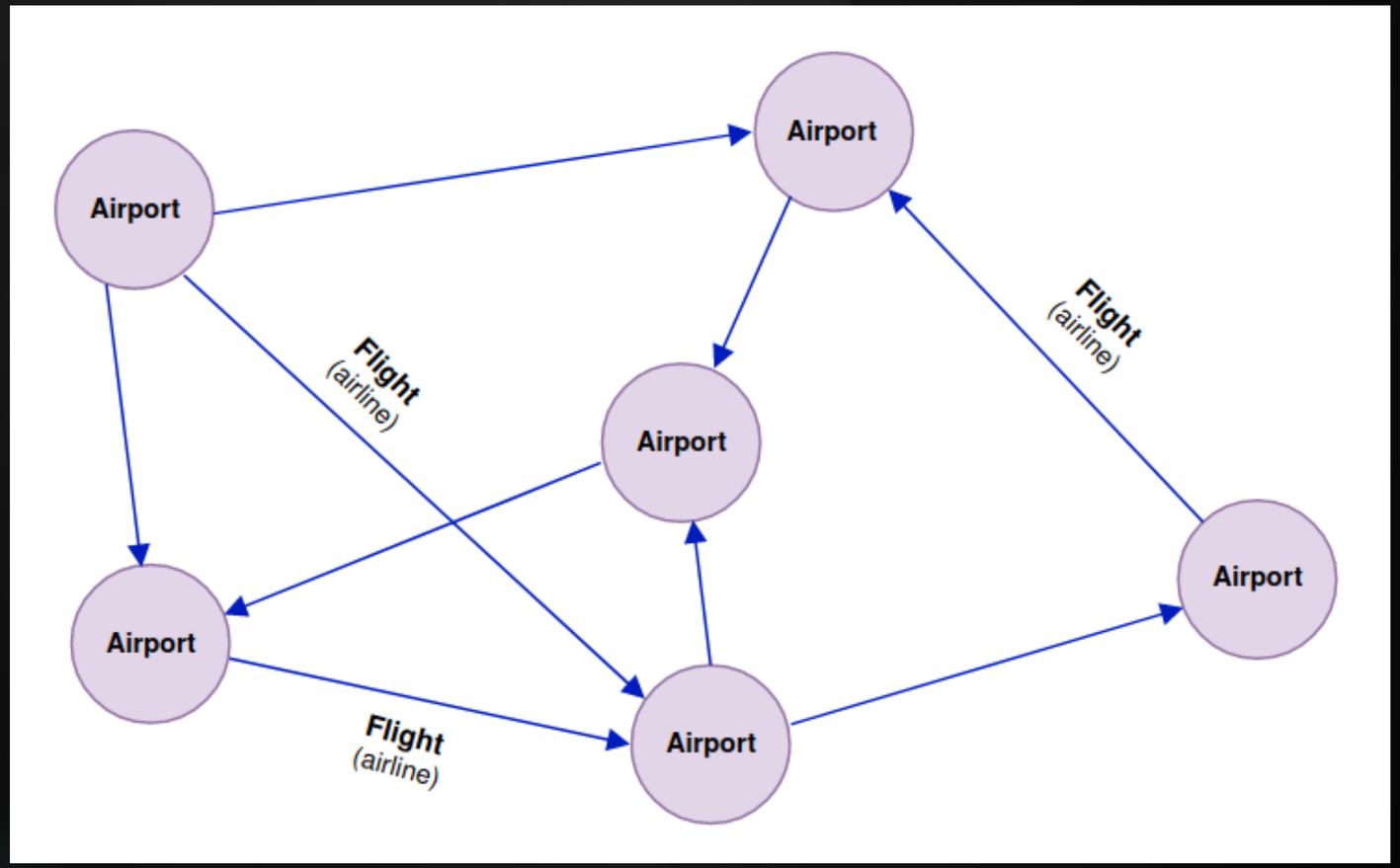
Grafo

Edge:

- Vértice de destino que representa um aeroporto
- Nome da companhia aérea

```
class Edge {
    Vertex * dest;      // destination vertex
    string airline ;         // edge weight
public:
    Edge(Vertex *d, string airline);
    Vertex *getDest() const;
    //void setDest(Vertex *dest);
    string getAirline() const;
    //void setWeight(double weight);
    friend class Graph ;
    friend class Vertex ;
};
```

Grafo


Airport
Airport
Airport
Airport
Airport
Airport
Flight
(airline)
Flight
(airline)
Flight
(airline)

Viajar de um lugar para outro...

Métodos implementados:

- Informação sobre o percurso mais eficaz para realizar uma deterninada viagem.
- Partindo de:
  - um Aeroporto
  - uma Cidade
  - uma Coordenada
  - uma Coordenada + Distância Máxima desejada
- Podendo ser:
  - a partir de qualquer companhia aérea
  - usar filtros para as companhias aéreas

```
void DataManip::getFlights(string origin, string dest, int oType, int dType, vector<string>& filters , int oRadius = 0, int dRadius = 0){
    vector<string> originAirp = {};
    vector<string> destAirp = {};
    vector<vector<string>> paths = {};
    switch (oType) {
        case 1: originAirp.push_back(origin); break;
        case 2: originAirp = getAirportsInCity( CityPlusCountry: origin); break;
        case 3: originAirp.push_back(getClosestAirport( coordinate: origin)); break;
        case 4: originAirp = getAirportsNearLocation( coordinate: origin, radius: oRadius); break;

    }
    switch (dType) {
        case 1: destAirp.push_back(dest); break;
        case 2: destAirp= getAirportsInCity( CityPlusCountry: dest); break;
        case 3: destAirp.push_back(getClosestAirport( coordinate: dest)); break;
        case 4: destAirp = getAirportsNearLocation( coordinate: dest, radius: dRadius); break;

    }

    for(auto o :string : originAirp){
        for( auto d :string : destAirp){
            vector<vector<string>> a = graph_.getPath( origin: o, dest: d, &: filters);
            if(!a.empty()){
                paths=a;
            }
        }
    }
    if(paths.empty()){
        cout<< "No flights available from "<< origin << " to " << dest << "..." << endl;
    }
```

Estatísticas

Globais:

- Total de Aeroportos - O(1)
- Total de Airlines - O(1)
- Total de Voos - O(V + E)

Aeroportos:

- Total de Voos - O(1)
- Total de Airlines - O(E)
- Total de Direct. Cities - O(1)
- Total de Direct. Countries - O(V + E)
- Total de Direct. Airports - O(1)
- Total de Reach. Cities - O(V + E)
- Total de Reach. Countries - O(V + E)
- Total de Reach. Airports - O(V + E)

Airlines:

- Total de Voos - O(V + E)

Cidades:

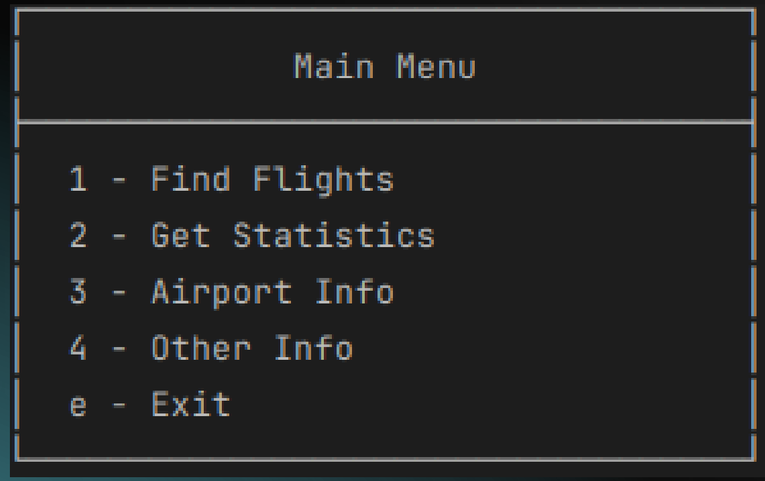
- Total de Voos - O(V + E)
- Total de Direct. Cities - O(V + E)
- Total de Direct. Countries - O(V + E)
- Total de Direct. Airports - O(V + E)

Interface

Menu Principal:

A partir do nosso menu principal, é possível consultar:

- Voos;
- Estatísticas;
- Informações sobre os Aeroportos;
- Outro tipo de informações.



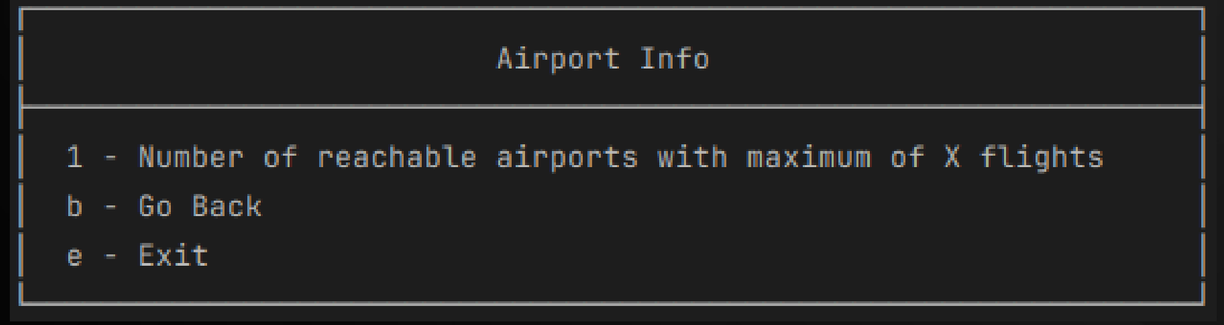
Interface

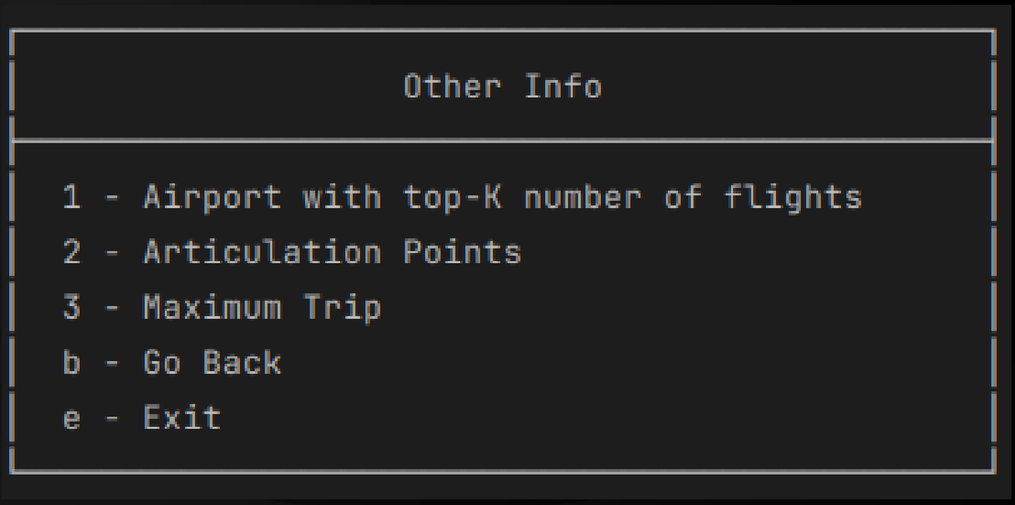
Informações sobre Aeroportos:

- Saber o nº de aeroportos que se consegue alcançar com um máximo de X voos.

Outras informações:

- Saber o aeroporto com o top-K nº de voos;
- Saber os aeroportos essenciais;
- Saber a viagem mais longa.





Destaque de Funcionalidades

- Através da realização deste trabalho, deixou-nos bastante orgulhosos todos os algoritmos que fomos capazes de criar e conseguimos ficar muito mais à vontade a trabalhar com grafos.

- A partir dos algoritmos criados, juntamente com toda a interface desenvolvida, somos capazes de garantir uma boa experiência ao utilizador, onde permitimos a pesquisa dos voos mais rápidos entre variados destinos, fornecer estatísticas à cerca dos aeroportos, das companhias aéreas, das cidades, entre muitas outras coisas.

- Mesmo até sobre a interface, achamos todos que desta vez fomos capazes de criar algo mais intuitivo e completo, o que não aconteceu no primeiro projeto pois ainda não estávamos bem familiarizados com todo o conceito e, por isso, deixou-nos bastante orgulhosos.

Dificuldades

- Ao longo deste projeto, fomos por várias vezes desafiados. Um dos maiores desafios foi o facto de ter de o fazer durante as férias, juntamente com o tentar aproveitar o Natal e a Passagem de Ano.
- Algumas complicações pelo facto de haverem cidades com o mesmo nome, no entanto, ficaram resolvidos.

Esforço de cada elemento do grupo:

- O trabalho foi dividido entre os 3 elementos do grupo de modo a contribuir para um rápido desenvolvimento do mesmo.

FIM!

Obrigada por assistirem a esta apresentação.