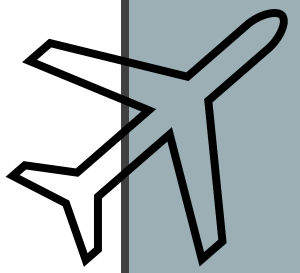
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

PROJECT 2
AIR TRAVEL FLIGHT MANAGEMENT SYSTEM



2LEIC08, grupo G86

Estudantes participantes:

Ana Catarina Patrício | up202107383

Bernardo Brandão | up202207939

Pedro Marinho | up202205693

CLASSES

- Classe Edge
  - Inclui as informações sobre os voos (arestas) [Graph.h]
- Classe Graph
  - Inclui as informações sobre os aeroportos (vértices), voos (arestas) e companhias aéreas (peso) [Graph.h]
- Classe Vertex
  - Inclui as informações sobre os aeroportos extraídas do dataset [Graph.h]

- Classe Menu
  - Inclui as seguintes funções:
    - Menu (menu principal do utilizador); [menu.h/menu.cpp]
    - displayPreferences (preferências do utilizador); [menu.h/menu.cpp]
    - Guide (rever as instruções do programa);
    - [menu.h/menu.cpp]

LEITURA DO DATASET

```
34  Graph<string> extractFromDatabase(int &numberOfAirports, int &numberOfAirlines, int &numberOfFlights) {
35      string line;
36      Graph<string> mainGraph;
37      ifstream file2( s: "../csv/airports.csv");
38      getline( &: file2, &: line); // this is to skip the first line of the file.
39      int count = 0;
40      while (getline( &: file2, &: line, dlm: '\n')) {
41          stringstream ss( s: line);
42          string airportCode, info;
43          getline( &: ss, &: airportCode, dlm: ',');
44          mainGraph.addVertex( in: airportCode);
45          mainGraph.addVertex( in: "P:" + line);
46          mainGraph.addEdge( sourc: airportCode, dest: "P:" + line, w: "AIRPORT");
47          count++;
48      }
49      numberOfAirports = count;
50      ifstream file3( s: "../csv/flights.csv");
51      getline( &: file3, &: line, dlm: '\n'); // this is to skip the first line of the file.
52      count = 0;
53      while (getline( &: file3, &: line, dlm: '\n')) {
54          stringstream ss( s: line);
55          string source, target, airline;
56          getline( &: ss, &: source, dlm: ',');
57          getline( &: ss, &: target, dlm: ',');
58          getline( &: ss, &: airline, dlm: ',');
59          mainGraph.addEdge( sourc: source, dest: target, w: airline);
60          count++;
61      }
```

LEITURA DO DATASET

```
63      ifstream file( s: "../csv/airlines.csv");
64      getline( &: file,  &: line); // this is to skip the first line of the file.
65      count = 0;
66      while (getline( &: file,  &: line,  dlm: '\n')) {
67          stringstream ss( s: line);
68          string airlineCode, info;
69          getline( &: ss,  &: airlineCode,  dlm: ',');
70          mainGraph.addVertex( in: "C:" + airlineCode);
71          mainGraph.addVertex( in: "L:" + line);
72          mainGraph.addEdge( sourc: "C:" + airlineCode,  dest: "L:" + line,  w: "AIRLINE");
73          count++;
74      }
75      numberOfAirlines = count;
76      return mainGraph;
77  }
78
79
80  #endif //PROJETO2AED_EXTRACTFROMDATABASE_H
81
```

DESCRIÇÃO DO GRAFOS USADO PARA REPRESENTAR O DATASET

- O main graph é o grafo que contém toda a informação recolhida do dataset:
  - Os vértices são os aeroportos;
  - Os edges são os voos;
  - O peso de cada edge e a airline;

PREFERÊNCIAS DO UTILIZADOR

```cpp
struct UserPreferences {
    std::vector<std::string> preferredAirlines;
    std::vector<std::string> avoidedAirlines;
    std::vector<std::string> avoidedCountries;
    std::vector<std::string> preferredCountries;
    std::vector<std::string> avoidedAirports;
    std::vector<std::string> preferredAirports;
    std::vector<std::string> avoidedCities;
    std::vector<std::string> preferredCities;
};
```

De acordo com as preferências do utilizador o grafo fica alterado, ou seja, se houver airlines/airports/cities/countries "avoided" pelo utilizador esses dados são removidos do grafo.
Ex: se preferir a RYR, ryanair, no grafo só vão aparecer as ligações em que a companhia aeria (correspondente ao peso do edge) corresponde a RYR

```cpp
/**
 * @brief Filter a graph based on user preferences.
 *
 * This function takes an original graph and applies user preferences to filter out vertices and edges.
 * It removes airports, edges with avoided airlines, airports from avoided countries and cities,
 * edges with non-preferred airlines, and airports not from preferred countries or cities.
 *
 * @tparam T The type of data stored in the graph vertices.
 * @param originalGraph The original input graph.
 * @param userPreferences The user preferences specifying filtering criteria.
 * @return A new graph containing only the vertices and edges that satisfy the user preferences.
 *
 * Example Usage:
 * @code
 *    Graph<string> originalGraph;
 *    // Populate originalGraph...
 *    UserPreferences preferences;
 *    // Set user preferences...
 *    Graph<string> filteredGraph = filterGraph(originalGraph, preferences);
 *    // Use filteredGraph for further processing...
 * @endcode
 */
template <class T>
Graph<T> filterGraph(const Graph<T> &originalGraph, const UserPreferences &userPreferences) {
    Graph<T> filteredGraph = originalGraph;

    // Remove airports based on avoidedAirports preference
    for (const auto &airport : const string & : userPreferences.avoidedAirports) {
        filteredGraph.removeVertex( in: airport);
    }
```

Definição do filtered grapth – as funções estão aqui implementadas

PESQUISA DE INFORMAÇÃO NOS GRAFOS

DFS SEARCH

DFS é um algoritmo de pesquisa por um grafo que se baseia em explorar o máximo possível num ramo antes de retroceder. Começando no nó inicial, este algoritmo visita um vizinho e continua a avançar até atingir o final de um ramo antes de retroceder e explorar outros ramos.

```
382     /****************** DFS ********************/
383
384     /**
385      * @brief Performs a depth-first search (DFS) traversal in the graph.
386      *
387      * Returns a vector containing the contents of the vertices in DFS order.
388      * Follows the DFS algorithm as described in theoretical classes.
389      *
390      * @return Vector with the contents of the vertices by DFS order.
391      */
392     template <class T>
393     vector<T> Graph<T>::dfs() const {
394         vector<T> res;
395         for (auto v : vertexSet)
396             v->visited = false;
397         for (auto v : vertexSet)
398             if (!v->visited)
399                 dfsVisit(v, res);
400         return res;
401     }
```

```
403     /**
404      * @brief Auxiliary function for DFS that visits a vertex and its adjacent vertices recursively.
405      *
406      * Updates a parameter with the list of visited node contents during DFS traversal.
407      *
408      * @param v Pointer to the current vertex.
409      * @param res Vector to store the contents of visited vertices in DFS order.
410      */
411     template <class T>
412     void Graph<T>::dfsVisit(Vertex<T> *v, vector<T> &res) const {
413         v->visited = true;
414         res.push_back(v->info);
415         for (auto &e : v->adj) {
416             auto w = e.dest;
417             if (!w->visited)
418                 dfsVisit(w, res);
419         }
420     }
```

PESQUISA DE INFORMAÇÃO NOS GRAFOS

BFS SEARCH

BFS é um algoritmo de pesquisa por um grafo que explora todos os vizinhos de um nó antes de passar para os vizinhos dos vizinhos. Começa no nó inicial e visita todos os nós no mesmo nível antes de passar para o próximo nível. Utilizando uma abordagem de fila, o algoritmo garante que os nós de níveis mais altos são visitados antes dos nós de níveis mais baixos. É ideal para problemas que envolvem a busca pelo caminho mais curto.

```
447  /****************** BFS ********************/
448
449  /**
450   * @brief Performs a breadth-first search (BFS) in the graph starting from the source node.
451   *
452   * Returns a vector containing the contents of the vertices in BFS order starting from
453   * the specified source node. If the source node is not found, an empty vector is returned.
454   *
455   * @param source The content of the source node for BFS.
456   * @return Vector with the contents of the vertices by BFS order from the source node.
457   */
458  template <class T>
459  vector<T> Graph<T>::bfs(const T &source) const {
460      vector<T> res;
461      auto s = findVertex(source);
462      if (s == NULL)
463          return res;
464      queue<Vertex<T> *> q;
465      for (auto v : vertexSet)
466          v->visited = false;
467      q.push(s);
468      s->visited = true;
469      while (!q.empty()) {
470          auto v = q.front();
471          q.pop();
472          res.push_back(v->info);
473          for (auto &e : v->adj) {
474              auto w = e.dest;
475              if (!w->visited) {
476                  q.push(w);
477                  w->visited = true;
478              }
479          }
480      }
481      return res;
```

ALGORITMOS PARA RECOLHER INFORMAÇÃO DO DATASET

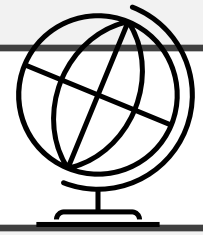
Estrutura em csv, portanto, a informação está separada por vírgulas sem espaços.

```
20  string getCityName(const string& info) {
21      string cityName;
22      stringstream ss( s: info);
23      // Extract city name (assuming it is the fourth field separated by commas)
24      getline( &: ss,  &: cityName,  dlm: ':');
25      getline( &: ss,  &: cityName,  dlm: ',');
26      getline( &: ss,  &: cityName,  dlm: ',');
27      getline( &: ss,  &: cityName,  dlm: ',');
28      return cityName;
29  }
30
```

```
42  string getCountryName(const string& info) {
43      string countryName;
44      stringstream ss( s: info);
45      // Extract country name (assuming it is the fifth field separated by commas)
46      getline( &: ss,  &: countryName,  dlm: ':');
47      getline( &: ss,  &: countryName,  dlm: ',');
48      getline( &: ss,  &: countryName,  dlm: ',');
49      getline( &: ss,  &: countryName,  dlm: ',');
50      getline( &: ss,  &: countryName,  dlm: ',');
51      return countryName;
52  }
53
```

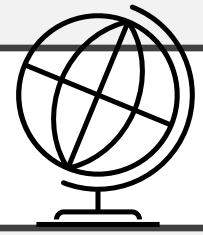
```
66  string getAirportName(const string& info) {
67      string airportName;
68      stringstream ss( s: info);
69      // Extract airport name (assuming it is the third field separated by commas)
70      getline( &: ss,  &: airportName,  dlm: ':');
71      getline( &: ss,  &: airportName,  dlm: ',');
72      getline( &: ss,  &: airportName,  dlm: ',');
73      return airportName;
74  }
75
```

DIJKSTRA'S ALGORITHM



```cpp
template <class T>
vector<vector<T>> findShortestPath(const Graph<T> &graph, const T &startAirport, const T &endAirport) {
    // Map to store distances from startAirport to each vertex
    unordered_map<T, int> distance;

    // Priority queue to store vertices and their distances
    priority_queue<pair<int, T>, vector<pair<int, T>>, greater<pair<int, T>>> pq;

    // Initialize distances
    for (const auto &vertex : graph.getVertexSet()) {
        if(!vertex->getAdj().empty()) {
            if(vertex->getAdj()[0].getWeight() == "AIRPORT") {
                distance[vertex->getInfo()] = INT_MAX;
            }
        }
    }

    // Set distance for the startAirport to 0
    distance[startAirport] = 0;

    // Insert startAirport into the priority queue
    pq.push({0, startAirport});
```

DIJKSTRA'S ALGORITHM



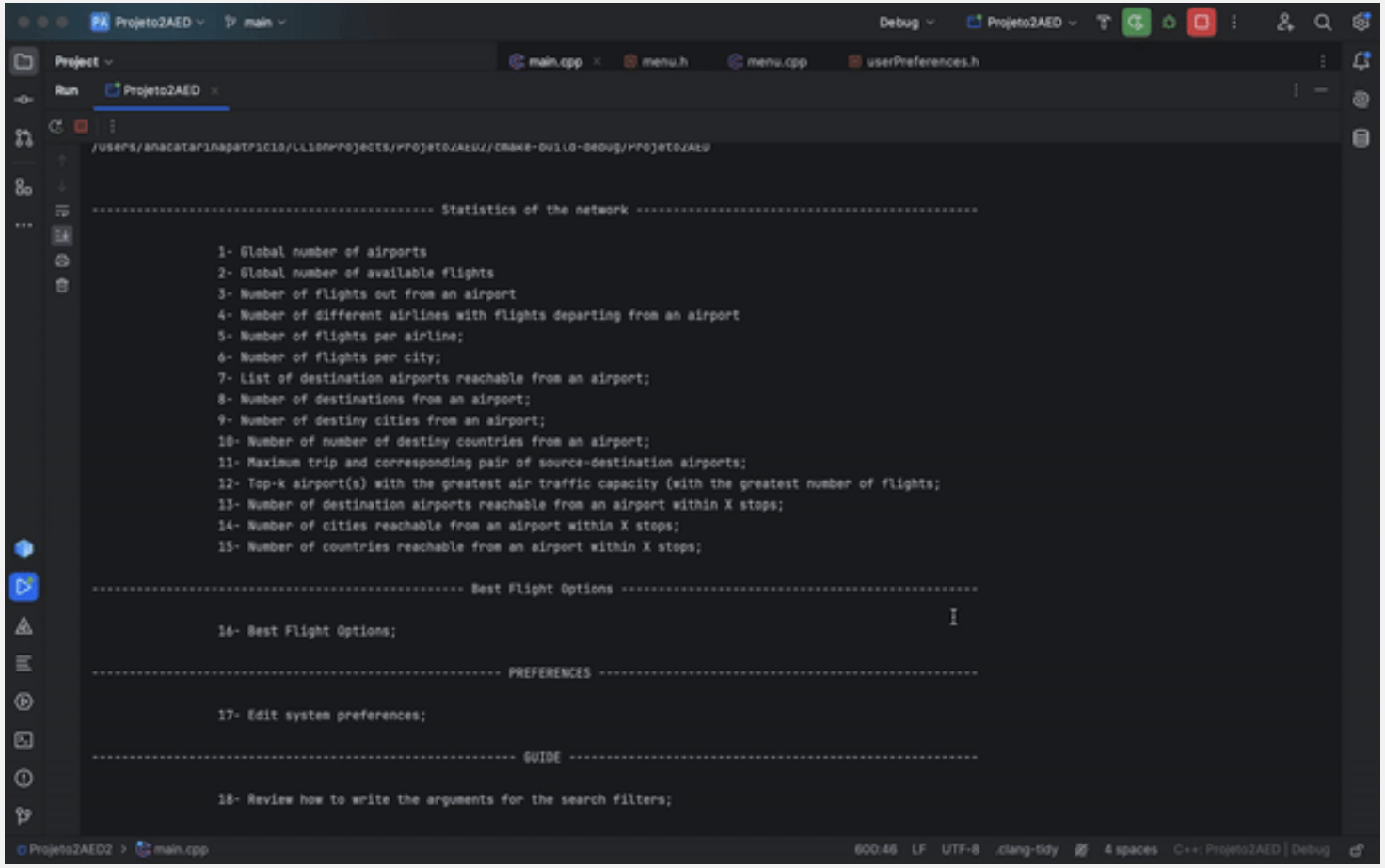
```
57      // Dijkstra's algorithm
58      while (!pq.empty()) {
59          // Extract the vertex with the smallest distance
60          T currentAirport = pq.top().second;
61          pq.pop();
62
63          // Get neighboring vertices of the currentAirport
64          if(!graph.findVertex( in: currentAirport)->getAdj().empty()) {
65              auto neighbors = graph.findVertex( in: currentAirport)->getAdj();
66              if(neighbors[0].getWeight() == "AIRPORT") {
67                  neighbors.erase(neighbors.begin());
68              }
69              // Update distances to neighboring vertices
70              for (const auto &edge : neighbors) {
71                  T neighborAirport = edge.getDest()->getInfo();
72                  int newDistance = distance[currentAirport] + 1; // Assuming equal weight for all edges
73
74                  // If a shorter path is found, update the distance
75                  if (newDistance < distance[neighborAirport] || distance[neighborAirport] == INT_MAX) {
76                      distance[neighborAirport] = newDistance;
77                      pq.push({newDistance, neighborAirport});
78                  }
79              }
80          }
81      }
```

```
83      // Reconstruct all paths from endAirport to startAirport
84      vector<vector<T>> allPaths;
85      vector<T> currentPath;
86      backtrackPaths(graph,distance, endAirport, startAirport, currentPath, allPaths);
87
88      return allPaths;
89  }
```

FUNCIONALIDADES IMPLEMENTADAS

Esta imagem é uma captura de ecrã do menu do utilizador que é outputed.

DESCRIÇÃO DO INTERFACE COM O UTILIZADOR



```cpp
/**
 * @brief Finds and returns a vector of vectors containing reachable airports within a specified maximum number of stops.
 *
 * This function performs a breadth-first search to find airports reachable from the provided airport code
 * within the given maximum number of stops. The result is a vector of vectors, where each vector represents
 * the airports reachable within a specific number of stops.
 *
 * @param graph The graph representing the airport connections.
 * @param airportCode The code of the starting airport.
 * @param maxStops The maximum number of stops allowed in the search.
 * @return A vector of vectors containing reachable airports at each level of stops.
 *
 * @note The input graph should represent airport connections using vertices and edges.
 */
vector<vector<string>> vectorOfReachableAirports(Graph<std::string> &graph, const std::string& airportCode, int maxStops){
    vector<vector<string>> airports;
    vector<string> airportsToVisit;
    vector<string> airportsVisited;
    airportsToVisit.push_back(airportCode);
    airports.push_back(airportsToVisit);
    for(int i = 0; i < maxStops; ++i){
        airportsToVisit.clear();
        for(const auto& airport : string const & airports[i]){
            auto vertex : Vertex<string> * = graph.findVertex( in: airport);
            for(int j = 1; j < vertex->getAdj().size(); ++j){
                if(find( first: airportsVisited.begin(), last: airportsVisited.end(), value_: vertex->getAdj()[j].getDest()->getInfo()) == airportsVisited.end()){
                    if(find( first: airportsToVisit.begin(), last: airportsToVisit.end(), value_: vertex->getAdj()[j].getDest()->getInfo()) == airportsToVisit.end()
                        airportsToVisit.push_back(vertex->getAdj()[j].getDest()->getInfo());
                        airportsVisited.push_back(vertex->getAdj()[j].getDest()->getInfo());
                    }
                }
            }
        }
        airports.push_back(airportsToVisit);
    }
    return airports;
```

DEFINIÇÃO DO VETOR DOS AEROPORTOS ACESSÍVEIS

UTILIZAÇÃO DO VETOR - NÚMERO DE AEROPORTOS ACESSÍVEIS

```cpp
/**
 * @brief Calculates the total number of reachable airports within a specified maximum number of stops.
 *
 * This function uses the vectorOfReachableAirports function to find airports reachable from the provided airport code
 * within the given maximum number of stops. The total count of reachable airports is then calculated and returned.
 *
 * @param graph The graph representing the airport connections.
 * @param airportCode The code of the starting airport.
 * @param maxStops The maximum number of stops allowed in the search.
 * @return The total number of reachable airports within the specified number of stops.
 *
 * @note The input graph should represent airport connections using vertices and edges.
 *
 * Example Usage:
 * @code
 *    Graph<std::string> airportGraph;  // Assume a properly initialized graph
 *    std::string startAirportCode = "JFK";
 *    int maxStopsAllowed = 3;
 *    int totalReachableAirports = numberOfReachableAirports(airportGraph, startAirportCode, maxStopsAllowed);
 *    // totalReachableAirports now contains the count of reachable airports within the specified stops
 * @endcode
 */
int numberOfReachableAirports(Graph<std::string> &graph, const std::string& airportCode, int maxStops) {
    auto airports : vector<vector<string>>  = vectorOfReachableAirports( &: graph, airportCode, maxStops);
    int count = 0;
    for(const auto& airport : vector<string> const & : airports){
        count += (int) airport.size();
    }
    return count-1;
}
```

TRABALHO DE EQUIPA

- Por um lado sentimo-nos desde o princípio mais preparados para desenvolver este projeto do que em relação ao projeto 1 pois já tínhamos a experiência de trabalhar com um dataset em csv, criar o user menu, etc.
- No entanto, o projeto foi igualmente desafiador pois tanto o data set como as funcionalidades do sistema que nos foi proposto implementar são bem mais complexos. Para além disso a organização da informação é diferente pois usamos uma nova estrutura de dados: os grafos.
- Posto isto, concluímos que este projeto fora muito importante para consolidar o que estivemos a estudar ao longo do semestre.
- Relativamente à contribuição de cada um para o projeto, sentimos que todos demos o nosso melhor e de igual forma.