

Sistema de Gestão de voos

Algoritmos e Estruturas de Dados


Miguel Lopes Guerrinha - up202205038

Rui Pedro Silva Cruz - up202208011

Yves François Joseph Bonneau - up202205062

Grupo G36

Classes usadas

- Airline (_code, _name, _callSign, _country); Files: Airline.hpp, Airline.cpp
 - Airport (_code, _name, _city, _country, _latitude, _longitude); Files: Airport.hpp, Airport.cpp
 - Flight (_airportSource, _airportTarget, _airline); Files: Flight.hpp, Flight.cpp
 - FlightManager (_flightSystem, _flights, _airportMap, _airlineMap); Files: FlightManager.hpp, FlightManager.cpp
 - Graph (_vertexSet, _stack_, _listscs_); Files: Graph.hpp, Graph.cpp
 - Edge (_dest, _weight); Files: Graph.hpp, Graph.cpp
 - Vertex (info, _adj, _visited, _processing, _indegree, _num, _low, _stops, _longestTrip); Files: Graph.hpp, Graph.cpp
 - Haversine (struct Coordinates - latitude, longitude); Files: Haversine.hpp, Haversine.cpp
 - Interface (); Files: Interface.hpp, Interface.cpp
- 

Leitura do dataset

Feita a partir de 3 funções com os mesmos métodos:

- Transformar o ficheiro em stream
- Repartir a stream em linhas
- Repartir cada linha em tokens, colocando-os em várias variáveis
- Guardar a informação de maneira correta no devido container

As funções utilizadas são:

- `load_airlines` → armazena os objetos do tipo `Airline`

```
13 void FlightManager::load_airlines(const string &file) {
14     ifstream arquivo( s: file);
15     string linha;
16     getline(& arquivo, & linha);
17
18     while (getline(& arquivo, & linha)) {
19         stringstream linhaStream( str: linha);
20         string code, name, callsign, country;
21         if (getline(& linhaStream, & code, delim: ','))
22             && getline(& linhaStream, & name, delim: ',')
23             && getline(& linhaStream, & callsign, delim: ',')
24             && getline(& linhaStream, & country, delim: ',') {
25             _airlineMap.emplace(code, std::make_shared<Airline>(code, name, callsign, country));
26         }
27     }
28 }
```

Leitura do dataset

- load_airports → armazena objetos do tipo Airport
- load_flights → armazena objetos do tipo Flight

```
30 void FlightManager::load_airports(const std::string &file) {
31     ifstream arquivo(&file);
32     string linha;
33     getline(&arquivo, &linha);
34
35     while (getline(&arquivo, &linha)) {
36         stringstream linhaStream(str(linha));
37         string code, name, city, country, latitude, longitude;
38         if (getline(&linhaStream, &code, delim: ','))
39             && getline(&linhaStream, &name, delim: ',')
40             && getline(&linhaStream, &city, delim: ',')
41             && getline(&linhaStream, &country, delim: ',')
42             && getline(&linhaStream, &latitude, delim: ',')
43             && getline(&linhaStream, &longitude, delim: ',') {
44             // Atribuir a um unordered_map por exemplo Airport(code, name, city, country, stod(latitude), stod(longitude))
45             Airport airport(code, name, city, country, latitude: stod(str(latitude)), longitude: stod(str(longitude)));
46             _airportMap.emplace(code, airport);
47             _flightSystem.addVertex(lin: airport);
48         }
49     }
50 }
```

```
52 void FlightManager::load_flights(const std::string &file) {
53     ifstream arquivo(&file);
54     string linha;
55     getline(&arquivo, &linha);
56
57     while (getline(&arquivo, &linha)) {
58         stringstream linhaStream(str(linha));
59         string source, target, airline;
60         if (getline(&linhaStream, &source, delim: ','))
61             && getline(&linhaStream, &target, delim: ',')
62             && getline(&linhaStream, &airline, delim: ',') {
63             _flights.emplace(airportSource: source, airportTarget: target, airline);
64             _flightSystem.addEdge( source: _airportMap.at(k: source), dest: _airportMap.at(k: target), wt: _airlineMap.at(k: airline));
65         }
66     }
67 }
68 }
```

Grafo usado

Graph:

- `unordered_map<string, Vertex*> _vertexSet` → map de todos os vértices do grafo. Existe uma associação do código do aeroporto ao respectivo aeroporto permitindo encontrar um aeroporto apenas com o seu código
- inteiro `_index` → índice de cada vértice para identificação dos mesmos
- `stack<Vertex> _stack` → pilha para armazenar vértices durante o algoritmo DFS
- `list<list<Airport>> _list_sccs` → Lista de listas de aeroportos para armazenar componentes fortemente conectadas

Edge:

- `Vertex* _dest` → Vértice (aeroporto) de destino da aresta
- `std::shared_ptr<Airline> _weight` → Companhia aérea associada a cada aresta

```
237 → class Graph {  
238     /// Hash table com o código do aeroporto e um  
239     unordered_map<string, Vertex *> _vertexSet;  
240  
241     /// Índice para identificação de vértices  
242     int _index;  
243  
244     /// Pilha para armazenar vértices durante o  
245     stack<Vertex> _stack;  
246  
247     /// Lista de listas de aeroportos para armaze  
248     list<list<Airport>> _list_sccs;  
249 }
```

```
198 → class Edge {  
199     /// Vértice (aeroporto) de destino  
200     Vertex *_dest;  
201  
202     /// Companhia aérea associada à aresta  
203     std::shared_ptr<Airline> _weight;
```

Grafo usado

Vertex:

- Airport info → Aeroporto associado ao vértice
- vector<Edge> _adj → Lista de vértices (aeroportos) que saem desse vértice
- booleano _visited{} → Booleano que indica se o vértice já foi visitado numa pesquisa
- bool _processing{} → Booleano que indica se o vértice está a ser processado numa pesquisa
- int _indegree{} → Número de arestas (voos) que entram neste vértice
- int _num{} → Número identificador único atribuído a este vértice
- int _low{} → Valor utilizado em algoritmos de busca em profundidade para determinar a menor conectividade
- int _stops{} → Número de paragens necessárias para chegar a este vértice a partir de um ponto de origem
- int _longestTrip = 0 → A maior distância registrada em voos a partir deste aeroporto

```
32 class Vertex {
33 private:
34     /// Aeroporto associado
35     Airport info;
36
37     /// Lista de vértices
38     vector<Edge> _adj;
39
40     /// Booleano que indica se o vértice já foi visitado numa pesquisa
41     bool _visited{};
42
43     /// Booleano que indica se o vértice está a ser processado numa pesquisa
44     bool _processing{};
45
46     /// Número de arestas
47     int _indegree{};
48
49     /// Número identificador único atribuído a este vértice
50     int _num{};
51
52     /// Valor utilizado em algoritmos de busca em profundidade para determinar a menor conectividade
53     int _low{};
54
55     /// Número de paragens
56     int _stops{};
57
58     /// A maior distância registrada em voos a partir deste aeroporto
59     int _longestTrip = 0;
```

Melhor rota para viajar

Os métodos criados permitem ao utilizador uma enorme flexibilidade na escolha da origem e do destino da sua viagem

É permitido ao utilizador escolher o local de origem/destino:

- Código do aeroporto
- Nome do aeroporto
- Cidade
- Localização por coordenadas

```
394 int Interface::displayBestFlightsMenu2() {  
395     system( "command: \"clear\"");  
396     cout << "----- Melhor viagem entre dois aeroportos -----" << endl;  
397     cout << "1 - Código do aeroporto de destino" << endl;  
398     cout << "2 - Nome do aeroporto de destino" << endl;  
399     cout << "3 - Cidade do aeroporto de destino" << endl;  
400     cout << "4 - Localização do aeroporto de destino" << endl;  
401     cout << "0 - Voltar atrás" << endl;  
402     cout << "Selecione uma opção: ";  
403     int choice;  
404     cin >> choice;  
405     return choice;  
406 }
```

O utilizador também pode filtrar a sua viagem por companhia aérea:

- Número de companhias aéreas em que deseja viajar e quais.

Melhor rota para viajar

Para implementar a flexibilidade do utilizador escolher a origem/destino:

- Transformar um nome de aeroporto (bestTripAirportName ($O(|V|)$), cidade (bestTripCityName ($O(|V|)$) ou coordenadas (bestTripCoordinates($O(|V|)$) num vetor de códigos de aeroportos

```
434 ↗ string FlightManager::bestTripAirportName(const string &src, const Graph &graph = _flightSystem) {
435     Vertex *source = nullptr;
436     for (const auto &vertex : constpair<...> & : graph.getVertexSet()) {
437         if (vertex.second->getInfo().getName() == src) {
438             source = vertex.second;
439         }
440     }
441     if (source == nullptr) {
442         std::cout << "Aeroportos de origem ou destino não encontrados." << std::endl;
443         return "";
444     }
445     return source->getInfo().getCode();
446 }
447
448 ↗ vector<string> FlightManager::bestTripCityName(const std::string &srcCity, const Graph &graph = _flightSystem) {
449     vector<string> sourceAirports;
450     for (const auto &vertex : constpair<...> & : graph.getVertexSet()) {
451         if (vertex.second->getInfo().getCity() == srcCity) {
452             sourceAirports.push_back(vertex.second->getInfo().getCode());
453         }
454     }
455     if (sourceAirports.empty()) {
456         std::cout << "Aeroportos de origem ou destino não encontrados." << std::endl;
457         return {};
458     }
459     return sourceAirports;
460 }
```


Melhor rota para viajar

Construir um grafo de forma a filtrar quais as companhias aéreas desejadas:

- Construir um vetor de companhias aéreas desejadas
- Criar um grafo que apenas utiliza as companhias aéreas presentes no vetor (filterGraphByAirline ($O(|V| + |V| * |E|)$))

```
408 Graph Interface::filterFlightsByAirline() {
409     vector<string> airlines;
410     string airline;
411     char choice;
412
413     std::cout << "Deseja filtrar voos por companhia aérea? (Y/N): ";
414     std::cin >> choice;
415
416     while (toupper(c: choice) == 'Y') {
417         std::cout << "Qual o código da Airline? ";
418         std::cin >> airline;
419         airlines.push_back(airline);
420
421         std::cout << "Quer filtrar por mais alguma companhia aérea? (Y/N): ";
422         std::cin >> choice;
423     }
424
425     Graph filteredGraph = FlightManager::getFlightSystem();
426     if (!airlines.empty()) {
427         filteredGraph = FlightManager::filterGraphByAirline(airlines);
428     }
429     return filteredGraph;
430 }
```

Melhor rota para viajar

Percorrer o grafo retornado da função que filtra por companhias aéreas através de uma BFS, partindo do(s) aeroporto(s) de partida e terminando no(s) aeroporto(s) de destino

Por fim construímos o percurso mais favorável ao utilizador utilizando uma DFS

Toda esta função tem complexidade: `bestTripAirportCode` ($O(|V| + 2|E| + N^2)$)



Interface

- Decidimos organizar a interface principalmente em duas categorias: **informações gerais** (onde separamos os voos dos demais dados, para uma melhor organização) e as **melhores opções de viagem** disponíveis para o utilizador, considerando um ponto de origem e destino. Além disso, oferecemos ao utilizador a flexibilidade de escolher a maneira como deseja inserir os dados.

```
----- Sistema de gestão de voos -----  
1 - Informações gerais sobre aeroportos, companhias aéreas e cidades.  
2 - Informações gerais sobre voos.  
3 - Melhor viagem entre dois aeroportos.  
0 - Sair  
Selecione uma opção:
```

```
----- Informações gerais -----  
1 - Número de aeroportos e número de voos disponíveis.  
2 - Número de voos de saída e airlines de um aeroporto.  
3 - Número de voos total numa cidade.  
4 - Número de voos de uma airline.  
0 - Voltar atrás  
Selecione uma opção:
```

Informações gerais sobre voos ←

→ Informações gerais sobre aeroportos,
companhias aéreas e cidades

```
----- Informações gerais de voos -----  
1 - Número de países para o qual se pode voar partindo de uma dada cidade.  
2 - Número de países para o qual se pode voar partindo de um dado aeroporto.  
3 - Número de destinos disponíveis a partir de um dado aeroporto.  
4 - Número de aeroportos, cidades e países atingíveis dentro de um certo número de paragens.  
5 - Maior viagem possível.  
6 - O enésimo aeroporto com o maior número de voos.  
7 - Aeroportos essenciais.  
0 - Voltar atrás  
Selecione uma opção: |
```

```
----- Melhor viagem entre dois aeroportos -----  
1 - Código do aeroporto de partida  
2 - Nome do aeroporto de partida  
3 - Cidade do aeroporto de partida  
4 - Localização do aeroporto de partida  
0 - Voltar atrás  
Selecione uma opção: |
```

→ Melhores viagens



Funcionalidade de destaque

- Apesar de nosso projeto aderir estritamente às funcionalidades esperadas, temos orgulho da flexibilidade oferecida na entrada de dados. Por exemplo, os utilizadores podem escolher um código de aeroporto como ponto de partida e o nome do aeroporto como ponto de chegada, entre outras combinações. Essa versatilidade se estende a todas as opções disponíveis.
- É fundamental destacar que, se o mesmo decidir mudar de ideia e visitar ou modificar uma opção anterior, pode fazê-lo sem comprometer a integridade dos dados. Essa capacidade de retrocesso oferece uma experiência mais dinâmica e interativa aos utilizadores.



Principais dificuldades

- Em relação aos inputs das funções, enfrentamos desafios ao lidar com casos em que o nome de uma cidade ou aeroporto continham espaços, resultando na não identificação adequada pelo programa.
- Além disso, no contexto do "**Best Flight**", o algoritmo apresentou maior complexidade em comparação com as demais funcionalidades. O principal obstáculo encontrava-se na gestão e acesso eficiente ao caminho do voo, especialmente quando tentávamos armazenar ou recuperar essa informação.



Observações

Cada membro do grupo desempenhou um papel fundamental na concretização deste projeto, com as responsabilidades sendo distribuídas de maneira justa entre todos os elementos.