

Trabalho 2 AED

Gonçalo Martins | up202108707 Leonor Filipe | up202204354 Luís Alves | up202108727

FEUP | 2022-2023 2LEIC15 + 2LEIC06

C GestaoA oities : tabHoities p airlines : tabHairlines flightNetwork : Graph maxCityLength: int maxCountryLength: int maxAirportNameLength: int maxAirlinenameLength: int maxAirlineCountryLength : int o tabHairlines : unordered set<Airline, AirlineHash, AirlineHash> o tabHcities : unordered_set<CityCountry, CityHash, CityHash> e GestaoA() e readAirlines(): void e readAirports(): void e readFlights(): void auxCenterDraw(int, bool) : pair<int, int> drawMenu(): void e drawListagemMenu() : void C Graph drawNumberMenu(): void drawYMenu(): void on:int drawDiameterMenu(): void □ airports : vector<Airport> drawFlyMenu(): void airportsInfo: map<string, int> e drawAirlineMenu() : void e drawAPMenu(): void · Graph() drawCity(const CityCountry &, bool) const : void · Graph(int) drawCities(const vector<CityCountry> &) const : void . addFlight(int, int, const string &) : void C Menu drawAirport(const string &, bool) : void . getAirports() const : vector<Airport> getAirportName(const string &) const : string drawAirports(const vector<string> &): void • getAirportCountry(const string &) const : string e teclaErro() : void drawAirline(const Airline &, bool) const : void · voltar(): void drawAirlines(const vector<Airline> &) const : void getAirportCode(const string &) const : string · drawFlight(const string &, const string &, bool) const : void · aeroportoNaoExiste(): void . getAirportName(const string &) const : string e drawFlights(const string &) : void aeroportoNaoEncontrado(): void getAirportCountry(const string &) const : string e drawNumberOfAirlines(const string &) : void e countryNotFound(): void getAirportFlights(const string &) const : list<pair<int, string>> · airportNotFound(): void drawNumberOfTargets(const string &): void e getAirportInfo(const string &) const : int main(): int · aeroportosiguais(): void drawNumberOfCountries(const string &): void e setNode(const string &, const string &, const string &, const string &, long double, long double) : void semCaminhoPossivel(): void isAirlineValid(const string &, const vector<string> &): bool dtsAirlines(int, int, vector<infoFlight> &, const vector<string> &): void drawYAirports(const string &, int): void drawYCities(const string &, int) : void e companhiaNaoExiste(): void bfsAirlines(int, const vector<string> &): void e caminhoCompanhiaPossivel() : void · drawYCountries(const string &, int) : void · abriuAplicacao(): void bfsDiameter(int) : int auxAirlineExist(const string &, const vector<string> &) : bool a airlineHelper(const vector-string> &): vector-string> outle, long double, const vector-string> &): void estWayPossible(int, int, vector-int)[ight > & vector-intoFlight> &): bool · fechouAplicacao(): void . bfsDiameterCountry(int, string) : int drawDiameter(): void drawDiameterCountry(const string &): void drawAirportsByCityCountry(const vector<string> &) const : void auxSPTH(const vector<string> &, const string &, const vector<string> &): pair<int, int> drawAirportsByAirport(const string &) const : void auxSPTW(const string &, const vector<string> &, const vector<string> &): pair<int, int> e dfsArticulationPoints(int, stack<int> &, list<int> &, int, int &) : void auxShortestPath(const string &, const string &, const vector<string> &, const vector<string> &, int, const vector<string> &): void e shortestPath(const string &, const string &, const vector<string> &) : void drawArticulationPoints(): void e getAirportsByCityCountry(const CityCountry &) : vector<string> · drawListArticulationPoints(): void yAirports(const string &, int) : set<int> haversine(double, double, double, double) : double yCities(const string &, int) : set<CityCountry> yCountries(const string &, int) : set<string> e ordenar() : bool · findCountry(const string &) : bool · setMaxCityLength(int) : void setMaxCountryLength(int) : void setMaxAirportNameLength(int) : void setMaxAirlineNameLength(int) : void setMaxAirlineCountryLength(int) : void e getAirlinesFromAirport(const string &) const : set<string> getAirports() const : vector<Airport> e getCities() const : tabHolties e getAirlines() const : tabHairlines · getFlightNetwork() const : Graph e getMaxCityLength() const : void getMaxCountryLength() const : void e getMaxAirportNameLength() const : void ■ e getMaxAirlineNameLength() const : void e getMaxAirlineCountryLength() const : void (S) Airport C Airline g flights : list<Flight> C CityCountry a code : string g code : string a name : string g name : string a city : string (S) infoFlight a city : string g callsign : string a country_:string (S) AirlineHash (S) CityHash (S) Flight a country : srting g country_: string · CityCountry() g latitude : long double g src : string n dest : int Airline(const string &, const string &, const string &, const string &) operator()(const Airline &) const : size t CityCountry(const string &, const string &) a dest : string a longitude : long double airline : string poperator()(const Airline &, const Airline &) const : bool getCode() const : string e getCity() const : string airline : string a visited : bool e getName() const : string getCountry() const : string a dist : int . getCountry() const : string operator==(const CityCountry &) const : bool a low : int e operator==(const Airline &) const : bool e operator<(const CityCountry &) const : bool g num : int operator<(const Airline &) const : bool g in_stack : bool g is articulation : bool

Leitura do dataset a partir dos ficheiros dados

- De forma a armazenar as informações fornecidas, foi implementado um método por cada ficheiro na classe GestaoA, que cria uma variável ifstream para abertura e leitura das respetivas linhas:
 - Método readAirlines: leitura do ficheiro airlines.csv e armazenamento das companhias aéreas num unordered_set
 - Método readAirports: leitura do ficheiro airports.csv, armazenamento das cidades e respetivos países num unordered_set e criação de um nó no grafo por cada aeroporto
 - Método readFlights: leitura do ficheiro flights.csv e criação de uma aresta no grafo por cada voo, que conecta o aeroporto de destino ao aeroporto de origem desse mesmo voo

Grafo utilizado para representação do dataset

- De forma a representar o dataset da melhor forma possível, foi utilizado um grafo dirigido e não pesado
- Os nós representam os aeroportos (cuja posição identifica internamente cada aeroporto da classe Graph)
- As arestas correspondem aos voos realizados entre cada par de aeroportos

```
struct infoFlight {
   string src;
   string dest;
   string airline;
struct Flight {
   string airline; // airline
struct Airport {
   string code;
   string name;
   string city;
   string country;
Graph();
explicit Graph(int nodes); // Constructor: n.º nodes
```

Funcionalidades implementadas e algoritmos

1. Menor número de voos dados dois locais – opção "Viajar entre dois locais"

"Local" e "Rede" de voos:

- Diretamente de um aeroporto para outro
 - Usar qualquer companhia
 - Usar apenas uma companhia
 - Usar um conjunto de companhias
- Entre duas cidades
 - Usar qualquer companhia
 - Usar apenas uma companhia
 - Usar um conjunto de companhias
- Entre duas localizações (coordenadas)
 - Usar qualquer companhia
 - Usar apenas uma companhia
 - Usar um conjunto de companhias

- Algoritmo: adaptação de pesquisa em largura (BFS); utilização da fórmula de Haversine
- Implementação:
 - Algoritmo BFS: utilizado para determinar a distância (nº de nós) a que cada aeroporto se encontra do ponto de partida
 - Fórmula de Haversine: implementada no cálculo da distância entre dois pontos, através das suas latitudes e longitudes
- ♦ Complexidade temporal: O(|V|+|E|) no primeira opção e O(n) nas restantes

Funcionalidades implementadas e algoritmos

2. Informações sobre um aeroporto

- Listagens completas
 - Listar cidades
 - Listar aeroportos
 - Listar companhias aéreas
 - Listar voos
 - Listar aeroportos numa cidade
 - Listar companhias aéreas num aeroporto
- N a partir de um aeroporto
 - Nº companhias aéreas num aeroporto
 - Nº destinos diferentes a partir de um aeroporto
 - Nº países diferentes a partir de um aeroporto
- Destinos com máximo de Y voos
 - Aeroportos atingíveis num máximo de Y voos
 - Cidades atingíveis num máximo de Y voos
 - Países atingíveis num máximo de Y voos

- Algoritmo: adaptação de pesquisa em largura (BFS) na opção N a partir de um aeroporto
- Implementação:
 - Algoritmo BFS: utilizado para determinar a distância (nº de nós) a que cada aeroporto se encontra do ponto de partida
- Complexidade temporal: O(n²) nas listagens e nos destinos com máximo de Y voos, e O(n) em N a partir de um aeroporto

Interface com o utilizador

- Este projeto dispõe de um menu interativo, intuitivo e amigável, com múltiplas opções encadeadas dentro de cada funcionalidade diferente
- Todos os inputs do utilizador são validados, apresentando mensagens de erro claras e sucintas quando estes estão incorretos

```
GESTAO DE AEROPORTOS
  [1] - Indicar dois aeroportos
  [2] - Indicar duas cidades
  [3] - Indicar duas localizacoes (coordenadas)
  [V] - Voltar
Escolha a opcao e pressione ENTER:
Insira o nome do país de partida:portugal
Insira o nome da cidade de partida:paris
Insira o nome do pais de chegada: france
Insira o nome da cidade de chegada: paris
  Nao existe nenhum aeroporto no pais e cidade indicados
Insira o nome do pais de partida:
```

Destaque de funcionalidades

- Menor número de voos dados dois locais
- Destinos com máximo de Y voos

Dificuldades encontradas

- Complexidade elevada do projeto
- Dificuldade em otimizar a complexidade dos algoritmos
- Dificuldade em resolver alguns bugs

Esforço de cada elemento do grupo

Gonçalo: 33.3%

Leonor: 33.3%

♦ Luís: 33.3%

Tarefas de valorização

Diâmetros

- Cálculo do diâmetro (grafo geral)
- Cálculo do diâmetro para um país específico

Pontos de articulação

- Nº pontos de articulação
- Listagem dos pontos de articulação

- Algoritmo: adaptação de pesquisa em profundidade (DFS) e pesquisa em largura (BFS)
- Implementação:
 - Algoritmo BFS: utilizado para determinar a distância (nº de nós) a que cada aeroporto se encontra do ponto de partida e, assim, possibilitar encontrar a maior distância (diâmetro)
 - Algoritmo DFS: utilizado para procurar os pontos de articulação do grafo
- \diamond Complexidade temporal: O(|V|*(|V|+|E|))

E: número total de arestas do grafo N: tamanho/número total de nós do grafo