Renair

Uma rede de voos

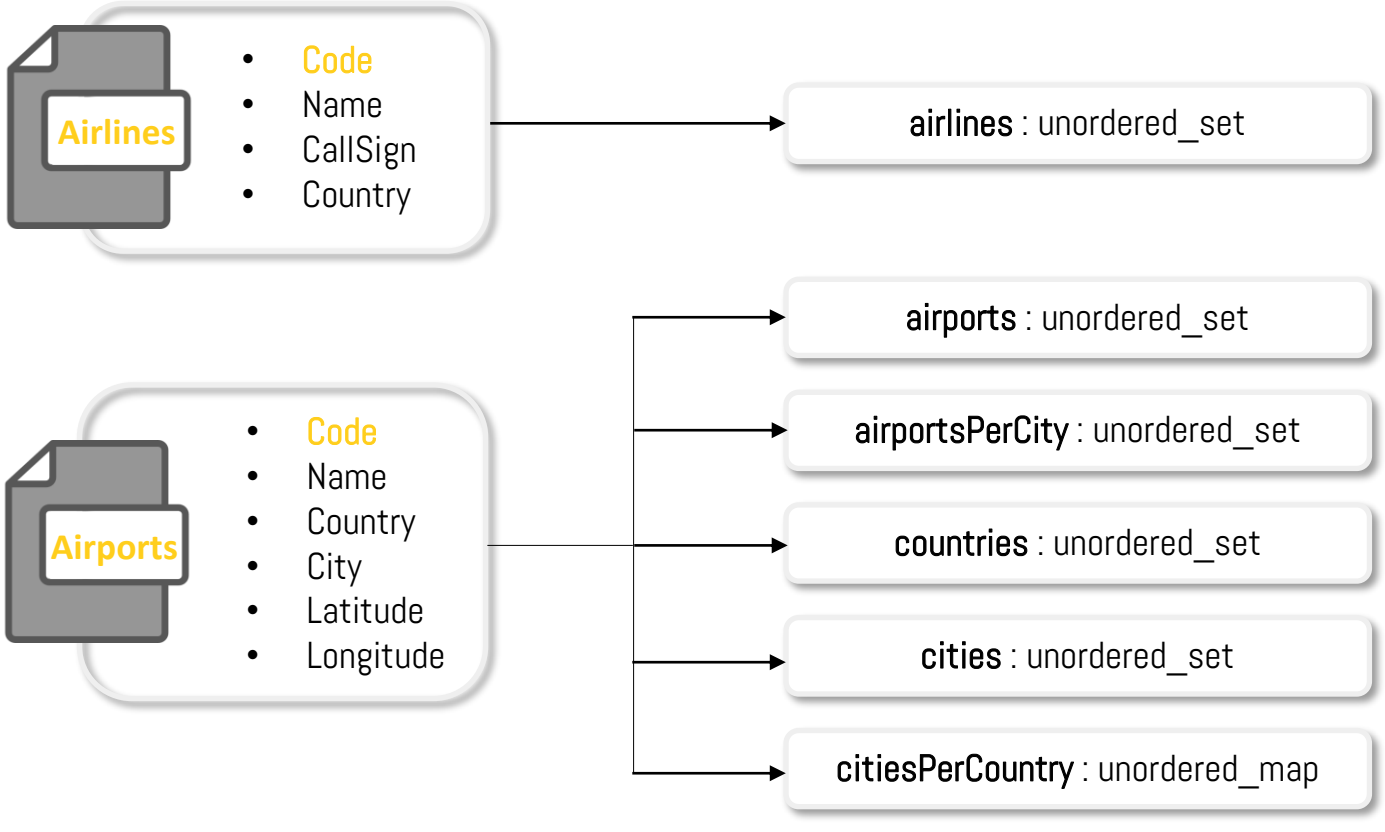
Trabalho prático 2

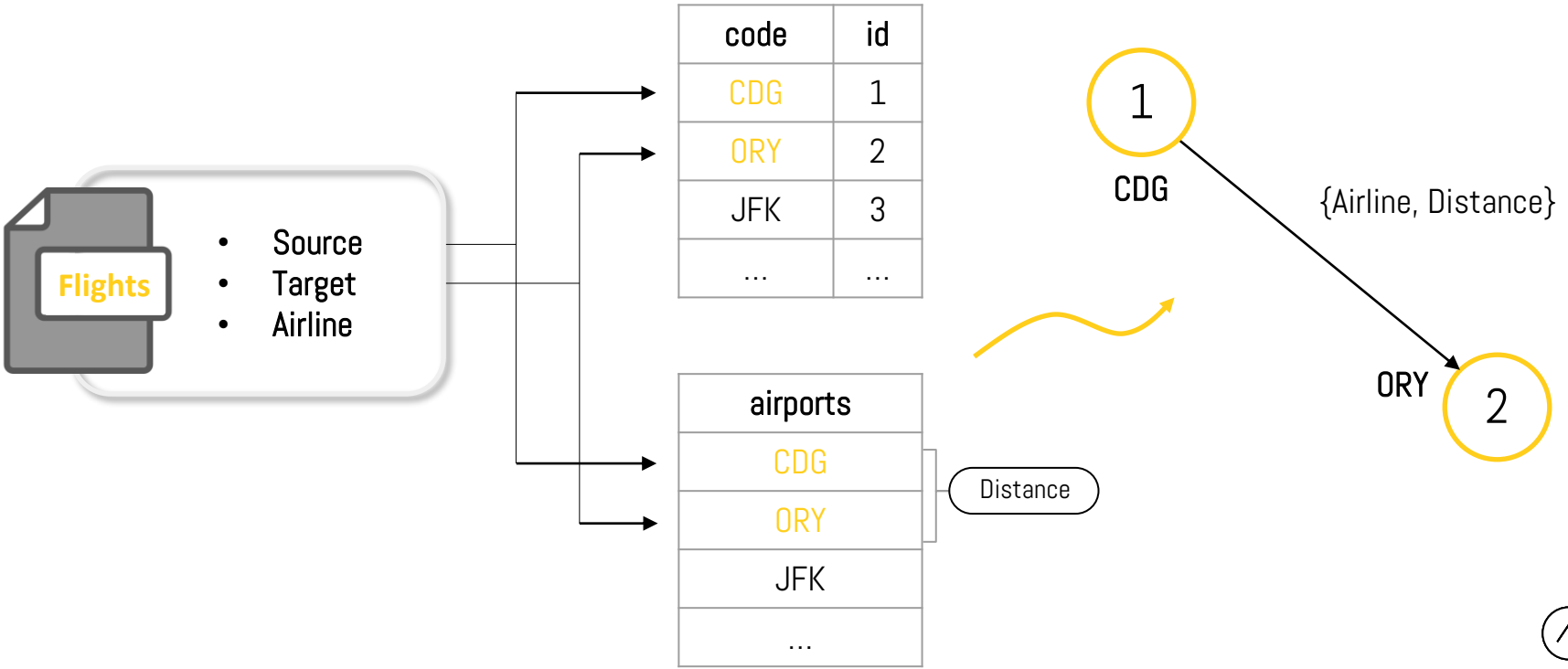
Unidade curricular: Algoritmos e Estruturas de Dados

2022/2023

Feito por: José Santos | Luís Du | Madalena Ye | G<69>


Airport
code : string
name : string
city : string
country : string
latitude : string
longitude : string
Airline
code : string
name : string
callSign : string
country : string
Supervisor
graph : Graph
airports : AirportH
airlines : AirlinesH
airportsPerCity : CityH
countries : unordered_set<string>
cities : unordered_set<string>
idAirports : unordered_map<string, int>
citiesPerCountry : unordered_map<string,list<string>>
nrAirportsPerCountry : map<string,int>
localAirports(latitude : double, longitude : double, radius : double) : vector<string>
countAirportsPerCountry() : void
processFlight (vector src, vector dest, unordered_set<Airline, AirlineHash, AirlineHash>) : list<pair<string,string>>
Graph
size : int
nodes : vector<Node>
addEdge(src, dest, airline, distance) : void
addAirport(src, airport) : void
distance(lat1, lon1, lat2, lon2) : double
nrFlights(src, dest, unordered_set : airlines) : int
listAirports(int nI, int max) : unordered_set
diameter() : double
articulationPoints(airlines) : list<int>
dijkstra(src, dest, airlines) : node
findPaths(paths, path, v) : void
flightsPerAirport() : vector<pair<int, string>>
Node
adj : list<Edge>
airport : Airport
visited : bool
distance : double
parents : vector<int>
num : int
low : int
art : bool
Edge
dest: int
Airline : airline
distance : double


Airlines
Code
Name
CallSign
Country
airlines : unordered_set
Airports
Code
Name
Country
City
Latitude
Longitude
airports : unordered_set
airportsPerCity : unordered_set
countries : unordered_set
cities : unordered_set
citiesPerCountry : unordered_map


Flights
Source
Target
Airline
code
id
CDG
1
ORY
2
JFK
3
...
...
airports
CDG
ORY
JFK
...
Distance
1
CDG
{Airline, Distance}
ORY
2

Criar o grafo

```
void Supervisor::createGraph(){
    ifstream inFile;
    string source, target, airline, line;
    inFile.open("../data/flights.csv");
    getline(inFile, line);
    while(getline(inFile, line)){
        istringstream is(line);
        getline(is,source,',');
        getline(is,target,',');
        getline(is,airline,',');
        auto d = Graph::distance(airports.find(Airport(source))->getLatitude(),airports.find(Airport(source))->getLongitude()
                ,airports.find(Airport(target))->getLatitude(),airports.find(Airport(target))->getLongitude());
        graph.addEdge(id_airports[source],id_airports[target],Airline(airline),d);
    }
}
```

```
struct Node {
    list <Edge> adj;
    Airport airport = Airport("");
    bool visited;
    double distance;
    vector<int> parents;
    int num=0;
    int low;
    bool art;
};
```

```
struct Edge {
    int dest{};
    Airline airline;
    double distance{};
};
```

Realizar Operação

Origem

Pretende partir de:

[1] Um aeroporto em específico
[2] Uma cidade
[3] Uma localização

src : vector<string>

Destino

Pretende chegar a:

[1] Um aeroporto em específico
[2] Uma cidade
[3] Uma localização

dest : vector<string>

Rede de voos

Pretende escolher as companhias aéreas a usar?

[1] Sim
[2] Não

airlines : AirlinesH

Realizar Operação

Nº mínimo de voos ($O(|V| + |E|)$)

```
int Graph::nrFlights(int src, int dest, unordered_set<Airline,Airline::AirlineHash,Airline::AirlineHash> airlines){
    for (int i = 1; i <= size; i++) {
        nodes[i].visited = false;
        nodes[i].distance = 0;
    }
    queue<int> q; q.push(src);
    nodes[src].visited = true;
    while(!q.empty()){
        int u = q.front(); q.pop();
        for (const Edge& e : nodes[u].adj){
            if (!airlines.empty() && airlines.find(e.airline) == airlines.end()) continue;
            int w = e.dest;
            if (!nodes[w].visited){
                q.push(w);
                nodes[w].visited = true;
                nodes[w].distance = nodes[u].distance + 1;
            }
        }
    }
    return nodes[dest].distance;
}
```

Realizar Operação

Melhores trajetos ($O(n \times m \times (|V|+|E|))$)

```
list<pair<string,string>> Supervisor::processFlight(int& bestFlight, const vector<string>& src, const vector<string>& dest,
                        const unordered_set<Airline, Airline::AirlineHash, Airline::AirlineHash>& airline) {
    bestFlight = INT_MAX;
    int nrFlights;
    list<pair<string,string>> res;
    for (const auto &s: src)
        for (const auto &d: dest) {
            if (s == d) continue;
            nrFlights = graph.nrFlights(id_airports[s], id_airports[d], airline);
            if (nrFlights != 0 && nrFlights < bestFlight) {
                bestFlight = nrFlights;
                res.clear();
                res.emplace_back(s,d);
            }
            else if(nrFlights == bestFlight)
                res.emplace_back(s,d);
        }
    return res;
}
```

Consultar Informação

Que tipo de informação deseja ver?

[1] Aeroporto específico
[2] Aeroportos
[3] Companhias Aéreas
[4] Países
[5] Pontos de articulação
[6] Diâmetro da rede

Ver Estatísticas

Que dados pretende analisar?

[1] Estatísticas de um aeroporto
[2] Nº de voos
[3] Nº de aeroportos
[4] Nº de companhias
[5] Nº de pontos de articulação

Interface

```
Bem-vindo!
(Pressione [0] sempre que quiser voltar atrás)

O que deseja fazer hoje?

[1] Realizar operação
[2] Consultar informação
[3] Ver estatísticas
[4] Sair

Opção: 4
```

```
Trajeto nº1: LGW --- ( TAP ) --- OPO

Trajeto nº2: STN --- ( RYR ) --- OPO

No total, existem 2 trajetos possíveis

O número mínimo de voos é 1
```

London, United Kingdom -> OPO
Usando a RYR e a TAP

```
Aeroportos distintos alcancáveis a partir de CIY:

KUN: Kaunas Intl
PSA: Pisa
CIA: Ciampino
DUB: Dublin
LIN: Linate
STN: Stansted
HHN: Frankfurt Hahn
CRL: Brussels South
```

01 Distância mínima percorrida

```
Indique o critério a usar:

[1] Número mínimo de voos
[2] Distância mínima percorrida
```

Algoritmo usado

```
==========================================================

Trajeto n°1: CDG --- ( EZY ) --- OPO

A distância mínima é 1231.14 km

==========================================================
```

```
Graph::Node Graph::dijkstra(int src, int dest, unordered_set<Airline, Airline::AirlineHash,
                            Airline::AirlineHash> airlines) {

    MinHeap<int, int> q(size, -1);
    for (int v=1; v<=size; v++) {
        nodes[v].distance = INF;
        q.insert(v, INF);
        nodes[v].visited = false;
        nodes[v].parents.clear();
    }
    nodes[src].distance = 0;
    nodes[src].parents.push_back(src);
    q.decreaseKey(src, 0);
    while (q.getSize()>0) {
        int u = q.removeMin();
        nodes[u].visited = true;
        for (const auto& e : nodes[u].adj) {

            if (!airlines.empty() && airlines.find(e.airline) == airlines.end()) continue;
            int v = e.dest;
            double w = e.distance;
            if (!nodes[v].visited && nodes[u].distance + w < nodes[v].distance) {
                nodes[v].distance = nodes[u].distance + w;
                auto aux =nodes[u].parents;
                if (find(aux.begin(),aux.end(),v) == aux.end()) aux.push_back(v);
                nodes[v].parents = aux;
                q.decreaseKey(v, nodes[v].distance);
            }
        }
    }
    return nodes[dest];
}
```

02 Top-k aeroportos/países

Pretende ver os aeroportos:

[1] Totais
[2] De um país
[3] Com mais voos
[4] Com mais companhias aéreas

Pretende ver os países:

[1] Com mais aeroportos
[2] Com menos aeroportos

Deseja consultar:

[1] Top 10
[2] Top 20
[3] Outro

Selecione um valor para o top: 5

1. CDG - 102 companhias aéreas
2. FRA - 97 companhias aéreas
3. BKK - 96 companhias aéreas
4. FCO - 91 companhias aéreas
5. LHR - 84 companhias aéreas

03 Diâmetro

```
double Graph::bfsDiameter(int v) {
    for (Node& node: nodes){node.visited = false; node.distance = -1.0;}
    queue<int> q;
    q.push(v);
    nodes[v].visited = true;
    nodes[v].distance = 0.0;
    double max = 0;
    while(!q.empty()){
        int u = q.front(); q.pop();
        for (const auto& e: nodes[u].adj){
            int w = e.dest;
            if (!nodes[w].visited){
                q.push(w);
                nodes[w].visited = true;
                nodes[w].distance = nodes[u].distance + 1;
                if (nodes[w].distance > max) max = nodes[w].distance;
            }
        }
    }
    return max;
}

double Graph::diameter() {
    nodes[1].visited = true;
    double max = bfsDiameter(1);
    for (int i = 1; i <= size; i++){
        if (!nodes[i].visited){
            nodes[i].visited = true;
            double diameter = bfsDiameter(i);
            if (diameter > max) max = diameter;
        }
    }
    return max;
}
```

04 Pontos de articulação

```
void Graph::dfsArt(int v, int index, list<int>& res,Airline::AirlineH airlines) {
    nodes[v].num = nodes[v].low = index;
    index = index+1;
    nodes[v].art = true;
    int count = 0;
    for (const auto& e : nodes[v].adj){
        auto w = e.dest;
        if(airlines.find(e.airline) != airlines.end() || airlines.empty()){
            if (nodes[w].num == 0){
                count++;
                dfsArt(w,index,res,airlines);
                nodes[v].low = min(nodes[v].low,nodes[w].low);
                if (nodes[w].low >= nodes[v].num && std::find(res.begin(),res.end(),v) == res.end()) {
                    if (index == 2 && count > 1) res.push_back(1);
                    else if (index != 2 && std::find(res.begin(),res.end(),v)== res.end()) res.push_back(v);
                }
            }
            else if (nodes[v].art) {
                nodes[v].low = min(nodes[v].low, nodes[w].num);
            }
        }
    }
}

list<int> Graph::articulationPoints(const Airline::AirlineH& airlines) {
    list<int> answer;

    for (int i = 1; i <= size; i++)
        nodes[i].visited = nodes[i].art = false;

    int index = 1;
    for (int i = 1; i <= size; i++)
        if (nodes[i].num == 0){
            dfsArt(i,index,answer,airlines);
        }
    return answer;
}
```

05 Validação de input

```
string Menu::validateCity(const string& country) {
    string city;
    cout << " Insira o nome da cidade (ex: Porto): ";

    getline(cin,city,'\n');

    while(cin.fail() || !supervisor->isValidCity(country,city)) {
        if (city == "0") return "0";
        if (!supervisor->isCity(city)) cout << " Esta cidade não existe\n";
        else cout << " Esta cidade não pertence a este país :/\n";
        cout << " Insira o nome da cidade (ex: Porto): ";
        cin.clear();
        getline(cin,city,'\n');
    }
    return city;
}
```

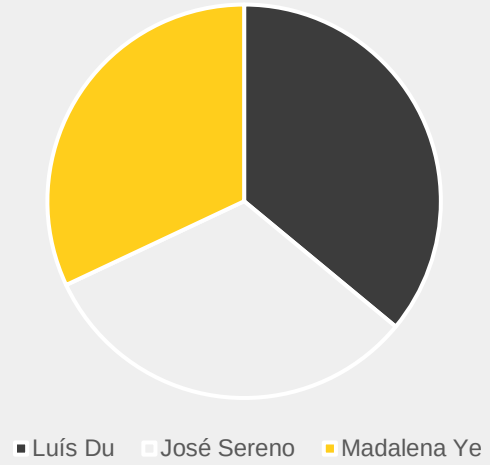
```
string validateCountry();
static string validateOption(const string& message);
string validateAirline();
static double validateLatitude();
static double validateLongitude();
static double validateRadius();
string validateCity(const string& country);
string validateAirport();
vector<string> validateLocal();
```

```
Insira o nome do país (ex: Portugal): Portugal
Insira o nome da cidade (ex: Porto): AED
Esta cidade não existe
Insira o nome da cidade (ex: Porto): Madrid
Esta cidade não pertence a este país :/
```

Dificuldades

- Implementação da menor distância possível
- Pontos de articulação
- Possibilidade de ter todos os caminhos de voos possíveis

Esforço



Extra: Grafo (auxílio nos testes)