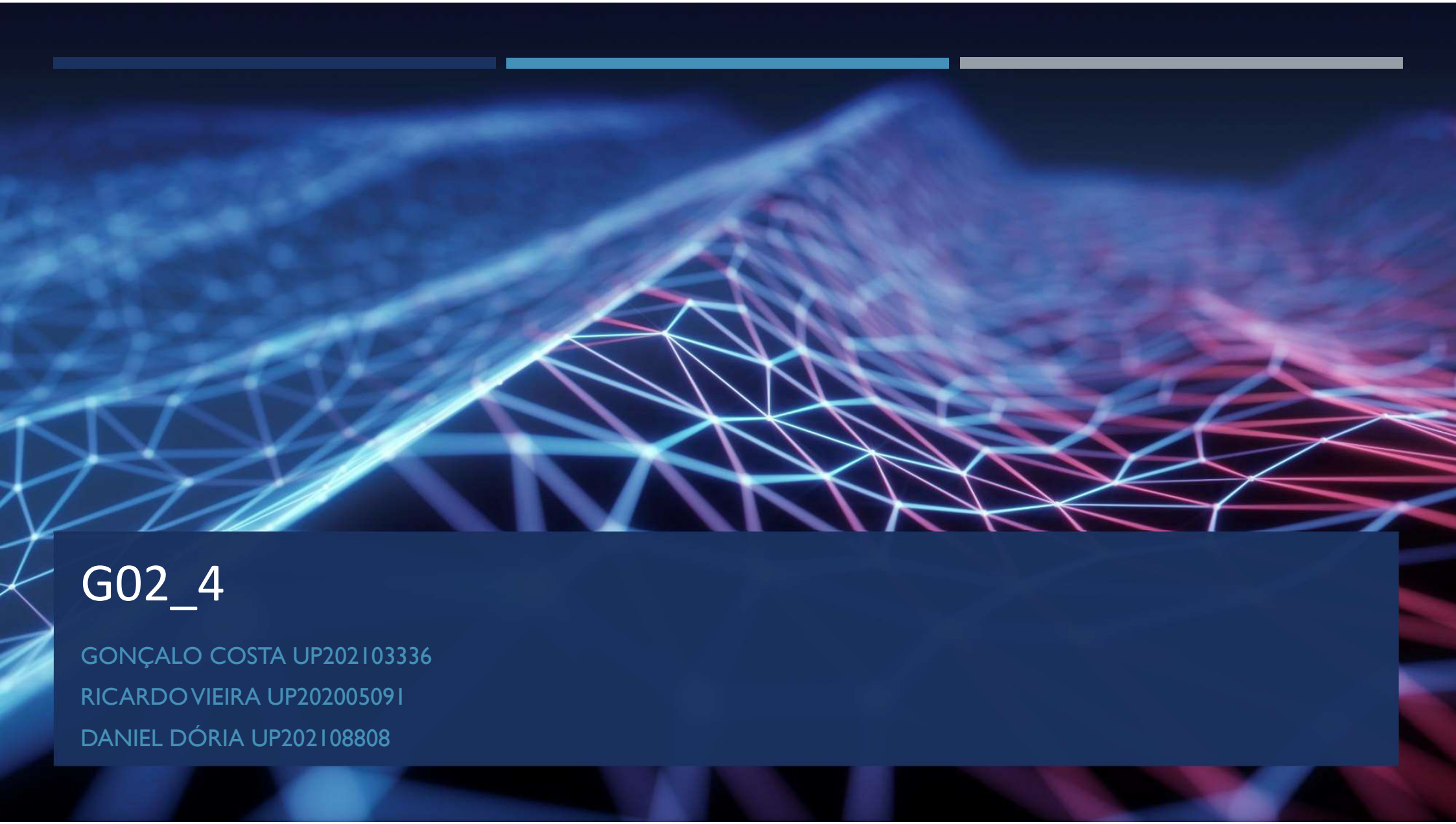


G02_4

GONÇALO COSTA UP202103336

RICARDO VIEIRA UP20200509I

DANIEL DÓRIA UP202108808

CLASSES

- LoadFiles;
- Graph;
- VertexEdge;
- MutablePriorityQueue;
- ToyGraph;
- App;
- Functions

LOADFILES

```
class LoadFiles {

public:

    void chooseExtra();

    void readExtra(string str);

    void loadExtra(string str);

    vector<ToyGraph> getExtraVector();

    Graph getExtraGraph();

    void chooseToy();

    void readToy(string path);

    void loadToy(string str);

    Graph getToyGraph();

    void chooseReal();

    void readReal(const string& path);

    void loadReal(const string& str);

    Graph getRealGraph();

    vector<Vertex*> getRealVector();

    void createAdj(const string& str);

    void readNodesReal (const string& path);

    void deleteAll();

    vector<vector<float>> createDistanceMatrix(const Graph& graph) const;


private:

    vector<ToyGraph> extra;

    vector<ToyGraph> toy;

    vector <Vertex*> real;
};
```

- Leitura de cada ficheiro com base na escolha do utilizador com a criação de vetores;
- Função createAdjs() após a leitura fazia a conexão entre os elementos do grafo

ESCOLHA E LEITURA - TOYGRAPHS

```
void LoadFiles::chooseToy(){
    cout << "\nWhich toy graph would you like to work with?" << std::endl;
    cout << "For stadiums graph, press [1]" << std::endl;
    cout << "For tourism graph, press [2]" << std::endl;
    cout << "For shipping graph, press [3]" << std::endl;
    int numVertices;
    cin >> numVertices;
    string result;


    switch(numVertices) {
        case 1:
            return readToy( path: "../Code/dataset/Project2Graphs/Toy-Graphs/stadiums.csv");

        case 2:
            return readToy( path: "../Code/dataset/Project2Graphs/Toy-Graphs/tourism.csv");

        case 3:
            return readToy( path: "../Code/dataset/Project2Graphs/Toy-Graphs/shipping.csv");

        default:
            cout << "Invalid input, please try again\n";
            return chooseToy();
    }

}
```

```
void LoadFiles::readToy(string path){
    string extraFilePath = path;
    fstream extraFile;
    extraFile.open( s: extraFilePath);

    if (extraFile.fail()) {
        cerr << "Unable to open " << extraFilePath << endl;
        return;
    }

    cout << "Reading File..." << endl;


    int jump = 0;
    while (extraFile.peek() != EOF) {
        string line;
        vector<string> strings;
        getline( &: extraFile,  &: line);
        if (jump==1) {
            loadToy( str: line);
        }
        jump=1;
    }

    cout << "File read successfully!" << endl;
    extraFile.close();
}
```

```
void LoadFiles::loadToy(string str){
    vector<string> result;
    stringstream ss(str);
    string item;
    while (getline( &: ss,  &: item,  delim: ',')) {
        result.push_back(item);
    }

    int pA = stoi( str: result[0]);
    int pB = stoi( str: result[1]);
    float pC = stof( str: result[2]);

    ToyGraph aux ( orig: pA,  dest: pB,  dist: pC);

    for (auto e : toy){
        if (aux.getOrig()==e.getOrig() && aux.getDest()==e.getDest() && aux.getDist()==e.getDist() ){
            return;
        }
    }

    toyGraph.addVertex( id: aux.getOrig());
    toyGraph.addVertex( id: aux.getDest());
    toy.push_back(aux);

    toyGraph.findVertex( id: pA)->addEdge( orig: toyGraph.findVertex( id: pA),  dest: toyGraph.findVertex( id: pB),  dist: pC);
    toyGraph.findVertex( id: pB)->addEdge( orig: toyGraph.findVertex( id: pB),  dest: toyGraph.findVertex( id: pA),  dist: pC);

}

Graph LoadFiles::getToyGraph() {
    return toyGraph;
}
```

ESCOLHA E LEITURA - EXTRAGRAPHS

```
void LoadFiles::chooseExtra() {

    cout << "\nHow many vertices do you want? Choose from: 25, 50, 75, 100, 200, 300, 400, 500, 600, 700, 800, 900" << std::endl;
    int numVertices;
    cin >> numVertices;
    string result;

    switch (numVertices) {
        case 25:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_25.csv");

        case 50:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_50.csv");

        case 75:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_75.csv");

        case 100:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_100.csv");

        case 200:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_200.csv");

        case 300:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_300.csv");

        case 400:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_400.csv");

        case 500:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_500.csv");

        case 600:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_600.csv");

        case 700:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_700.csv");

        case 800:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_800.csv");

        case 900:
            return readExtra( str: "../Code/dataset/Project2Graphs/Extra_Fully_Connected_Graphs/edges_900.csv");
        default:
            cout << "YOUR CHOICE IS INVALID\n";
            return chooseExtra();
    }
}
```

```
void LoadFiles::readExtra(string str) {
    string extraFilePath = str;
    fstream extraFile;
    extraFile.open( s: extraFilePath);

    if (extraFile.fail()) {
        cerr << "Unable to open " << extraFilePath << endl;
        return;
    }

    cout << "Reading File..." << endl;

    while (extraFile.peek() != EOF) {
        string line;
        vector<string> strings;
        getline( &: extraFile,  &: line);
        loadExtra( str: line);
    }

    cout << "File read successfully" << endl;
    extraFile.close();

}
```

```
void LoadFiles::loadExtra(string str) {

    vector<string> result;
    stringstream ss(str);
    string item;
    while (getline( &: ss,  &: item,  delim: ',')) {
        result.push_back(item);
    }

    int pA = stoi( str: result[0]);
    int pB = stoi( str: result[1]);
    float pC = stof( str: result[2]);

    ToyGraph aux ( orig: pA,  dest: pB,  dist: pC);

    for (auto e :ToyGraph : getExtraVector()){
        if (aux.getOrig()==e.getOrig() && aux.getDest()==e.getDest() && aux.getDist()==e.getDist()){
            return;
        }
    }

    extraGraph.addVertex( id: aux.getOrig());
    extraGraph.addVertex( id: aux.getDest());
    extra.push_back(aux);

    extraGraph.findVertex( id: pA)->addEdge( orig: extraGraph.findVertex( id: pA),  dest: extraGraph.findVertex( id: pB),  dist: pC);
    extraGraph.findVertex( id: pB)->addEdge( orig: extraGraph.findVertex( id: pB),  dest: extraGraph.findVertex( id: pA),  dist: pC);
}

vector<ToyGraph> LoadFiles::getExtraVector() {
    return extra;
}

Graph LoadFiles::getExtraGraph() {
    return extraGraph;
}
```

ESCOLHA E LEITURA - REAL WORLD GRAPHS

```
void LoadFiles::chooseReal() {

    cout << "\nWhich one of the real world graph would you like to choose?" << endl;
    cout << "For real 1 press [1]" << endl;
    cout << "For real 2 press [2]" << endl;
    cout << "For real 3 press [3]" << endl;

    int num;
    cin >> num;

    switch (num) {
        case 1:
            readReal( path: "../Code/dataset/Project2Graphs/Real-World-Graphs/graph1/nodes.csv");
            readNodesReal( path: "../Code/dataset/Project2Graphs/Real-World-Graphs/graph1/edges.csv");
            break;


        case 2:
            readReal( path: "../Code/dataset/Project2Graphs/Real-World-Graphs/graph2/nodes.csv");
            readNodesReal( path: "../Code/dataset/Project2Graphs/Real-World-Graphs/graph2/edges.csv");
            break;

        case 3:
            readReal( path: "../Code/dataset/Project2Graphs/Real-World-Graphs/graph3/nodes.csv");
            readNodesReal( path: "../Code/dataset/Project2Graphs/Real-World-Graphs/graph3/edges.csv");
            break;

        default:
            cout << "YOUR CHOICE IS INVALID\n";
            return chooseReal();
    }

}
```

```
void LoadFiles::readReal(const string& path) {

    string realFilePath = path;
    ifstream realFile;
    realFile.open( s: realFilePath);

    if (realFile.fail()) {
        cerr << "Unable to open " << realFilePath << endl;
        return;
    }

    cout << "Reading File..." << endl;

    string line;
    getline( &: realFile, &: line); // Skip the first line

    for (string line; getline( &: realFile, &: line);) {
        loadReal( str: line);
    }

    realFile.close();

}
```

```
void LoadFiles::loadReal(const string& str) {

    vector<string> result;
    stringstream ss(str);
    string item;
    while (getline( &: ss, &: item, delim: ',')) {
        result.push_back(item);
    }


    int pA = stoi( str: result[0]);
    double pB = stod( str: result[1]);
    double pC = stod( str: result[2]);

    Vertex aux( id: pA, longitude: pB, latitude: pC);

    realGraph.addVertex( vertex: const_cast<const Vertex*>(&aux));

}
```

```
void LoadFiles::readNodesReal (const string& path){
    string realFilePath = path;
    fstream realFile;
    realFile.open( s: realFilePath);

    if (realFile.fail()) {
        cerr << "Unable to open " << realFilePath << endl;
    }

    string line;
    getline( &: realFile, &: line); // Skip the first line

    for (string line; getline( &: realFile, &: line);) {
        createAdj( str: line);
    }

    cout << "File read successfully" << endl;
    realFile.close();
}

Graph LoadFiles::getRealGraph() {
    return realGraph;
}

vector<Vertex*> LoadFiles::getRealVector() {
    return real;
}
```

GRAPH

```
class Graph {
    int n;

public:
    Graph();

    bool addVertex(const int id);

    bool addVertex (const Vertex* vertex);

    bool addEdge(const int &source, const int &dest, double w);

    static Vertex *findVertex (const int &id);

    vector<Vertex *> getVertexSet() const;

    void deleteGraph();

    bool has_edge(int u, int v) const;

    pair<Graph, double> prim() const;

    pair<double, double> tspTriApprox(vector<unsigned> &path) const;

private:
    static vector<Vertex *> vertexSet;


};
```

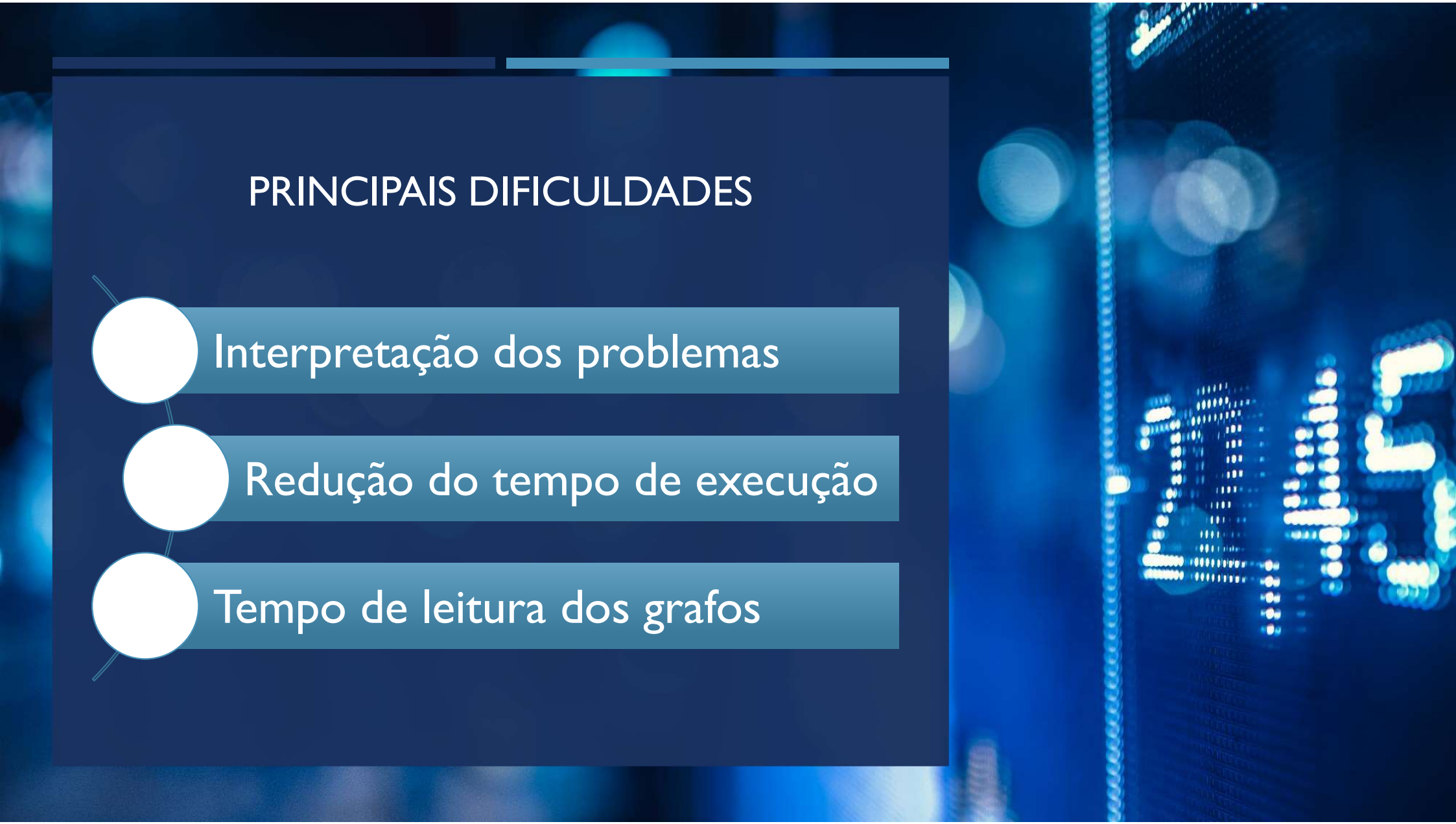
VERTEXEDGE

```
class Vertex{

public:
    Vertex(int id);
    Vertex(int id, double longitude, double latitude);
    int getID() const;
    vector<Edge*> getAdj() const ;
    bool isVisited() const ;
    void setVisited(bool _bool);
    float getDist() const ;
    void setDist (float dist);
    Edge * addEdge (Vertex *orig, Vertex *dest, float dist);
    Edge * getEdge(const Vertex *dest) const;
    bool removeEdge(Vertex * orig, Vertex *dest);

    void preorderDFS(vector<unsigned> &preorder);
    double distance(const Vertex *destination) const;
    bool operator<(Vertex & vertex) const;

    Edge *getPath() const;
    void setPath(Edge *path);


    double getLongitude() const;
    double getLatitude() const;

    friend class MutablePriorityQueue<Vertex>;

protected:
    int id;
    vector<Edge *> adj;
    float dist;
    bool visited = false;
    double longitude;
    double latitude;
    int queueIndex = 0;
    Edge *path = nullptr;

};
```

```
class Edge{
public:
    Edge(Vertex *orig, Vertex *dest, float dist);
    Vertex* getDest() const ;
    Vertex* getOrig() const ;
    float getDist() const ;
    double getWeight() const;

protected:
    Vertex *dest;
    Vertex *orig;
    float dist;
    double weight;
};
```

FUNCIONALIDADES IMPLEMENTADAS

- tsp;
  - Função principal para a TSP do exercício 2.1. Retorna o tempo que o backtracking demora, o min path e a sua distância.
- triangularApproximationH;
  - Função principal do exercício 2.2. Retorna o min cost da MST, a distância da tour e o tempo que a função demora.
- linKernighan;
  - Aplica o algoritmo heurístico de Lin-Kernighan para encontrar uma tour melhorada para a TSP (1/2 do exercício 2.3)
- PerformLinKernighanOptimization;
  - Faz um passo de otimização local usando regras de melhoramento de Lin-Kernighan numa dada tour com o algoritmo heurístico de Lin-Kernighan (2/2 do exercício 2.3)


PRINCIPAIS DIFICULDADES
Interpretação dos problemas
Redução do tempo de execução
Tempo de leitura dos grafos

OBRIGADO!

2° PROJETO – DESENHO DE ALGORITMOS DO LEIC

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO