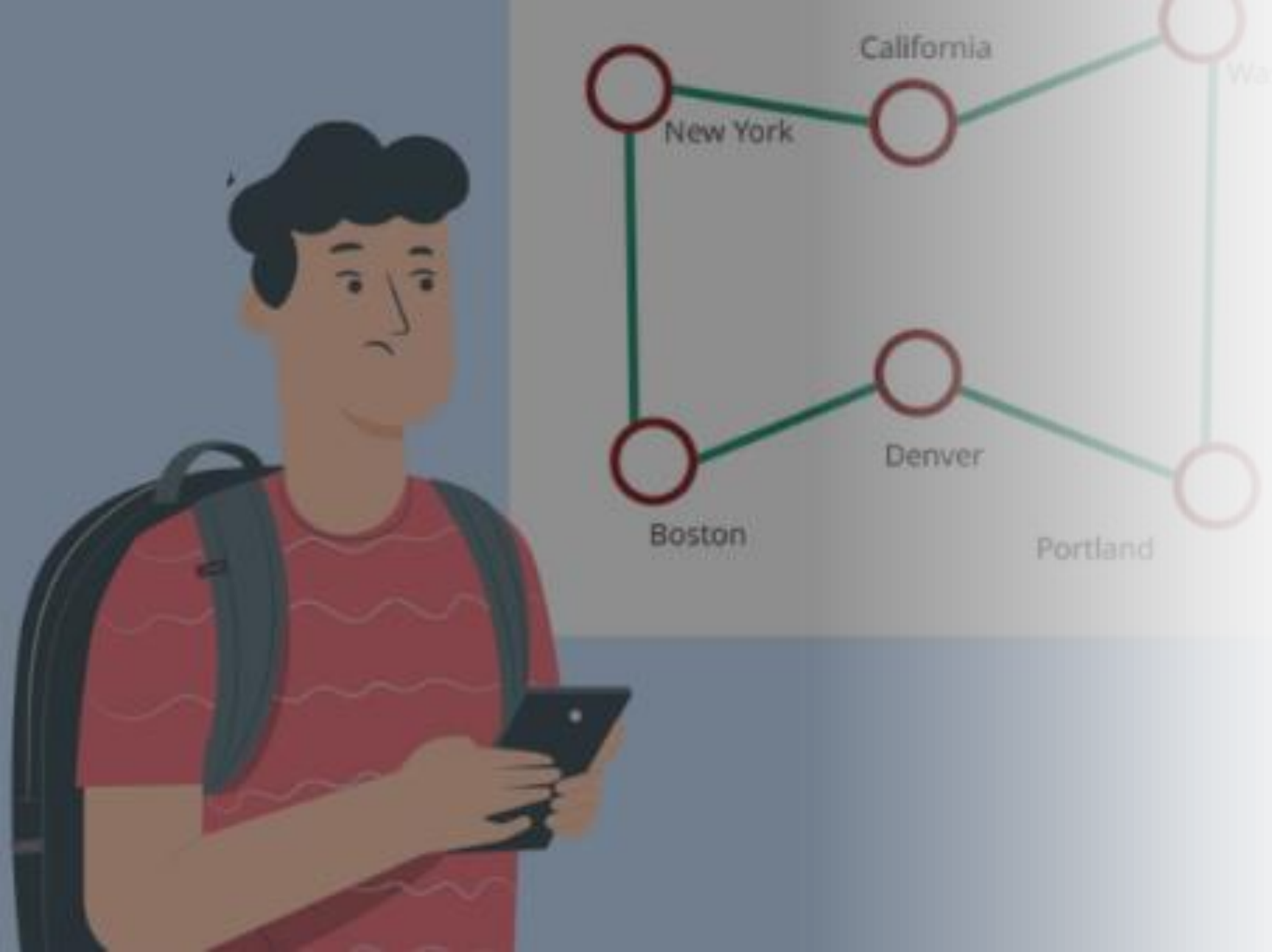
ANALYSIS OF TRAVELLING SALESPERSON PROBLEM (TSP)

DA Projeto 2

2023/2024

Turma 4

Elementos do Grupo:
1. Ângelo Oliveira(202207798)
2. Bruno Fortes(202209730)
3. José Costa(202207871)



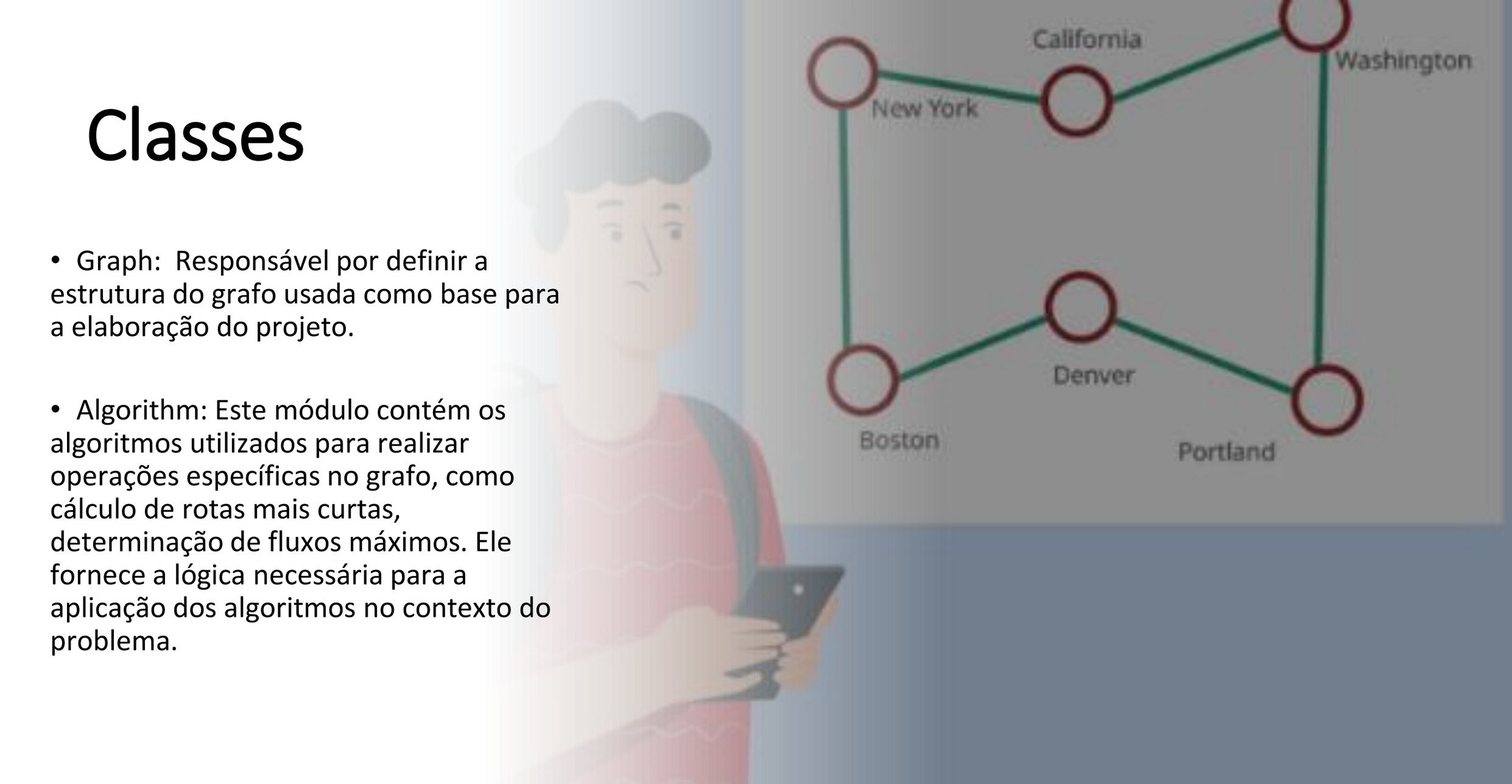
Objetivo

- O objetivo do trabalho é explorar o TRAVELLING SALESPERSON PROBLEM (TSP) através da implementação de uma abordagem exaustiva básica para encontrar soluções aproximadas.

Classes

- Graph: Responsável por definir a estrutura do grafo usada como base para a elaboração do projeto.

- Algorithm: Este módulo contém os algoritmos utilizados para realizar operações específicas no grafo, como cálculo de rotas mais curtas, determinação de fluxos máximos. Ele fornece a lógica necessária para a aplicação dos algoritmos no contexto do problema.



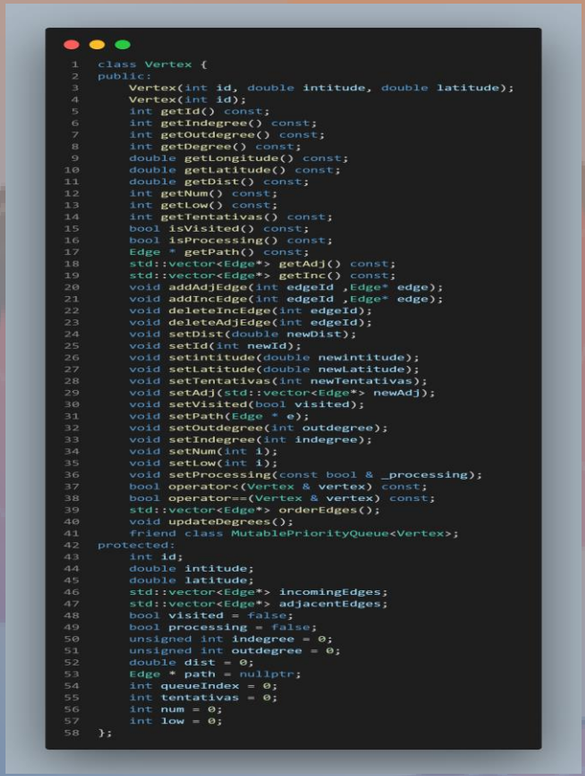
Classes

- LoadingFunctions: Ponto de partida da nossa aplicação, trata do processamento dos dados fornecidos nos ficheiros .csv lendo-os e organizando-os nas estruturas por nós escolhidas

- Ui: Interface servil do utilizador para a navegação do programa.

O Grafo

A classe Vertex é uma implementação de um vértice de grafo, projetada para suportar várias operações comuns em algoritmos de grafos, incluindo travessias, busca de caminhos mínimos e manipulação de estruturas de dados associadas a grafos. Ela encapsula tanto os dados quanto os métodos necessários para interagir com um grafo de forma eficiente e eficaz.



```cpp
class Vertex {
public:
    Vertex(int id, double intitude, double latitude);
    Vertex(int id);
    int getId() const;
    int getIndegree() const;
    int getOutdegree() const;
    int getDegree() const;
    double getLongitude() const;
    double getLatitude() const;
    double getDist() const;
    int getNum() const;
    int getLow() const;
    int getTentativas() const;
    bool isVisited() const;
    bool isProcessing() const;
    Edge * getPath() const;
    std::vector<Edge*> getAdj() const;
    std::vector<Edge*> getInc() const;
    void addAdjEdge(int edgeId ,Edge* edge);
    void addIncEdge(int edgeId ,Edge* edge);
    void deleteIncEdge(int edgeId);
    void deleteAdjEdge(int edgeId);
    void setDist(double newDist);
    void setId(int newId);
    void setintitude(double newintitude);
    void setLatitude(double newLatitude);
    void setTentativas(int newTentativas);
    void setAdj(std::vector<Edge*> newAdj);
    void setVisited(bool visited);
    void setPath(Edge * e);
    void setOutdegree(int outdegree);
    void setIndegree(int indegree);
    void setNum(int i);
    void setLow(int i);
    void setProcessing(const bool & _processing);
    bool operator<(Vertex & vertex) const;
    bool operator==(Vertex & vertex) const;
    std::vector<Edge*> orderEdges();
    void updateDegrees();
    friend class MutablePriorityQueue<Vertex>;
protected:
    int id;
    double intitude;
    double latitude;
    std::vector<Edge*> incomingEdges;
    std::vector<Edge*> adjacentEdges;
    bool visited = false;
    bool processing = false;
    unsigned int indegree = 0;
    unsigned int outdegree = 0;
    double dist = 0;
    Edge * path = nullptr;
    int queueIndex = 0;
    int tentativas = 0;
    int num = 0;
    int low = 0;
};
```

```cpp
class Edge {
public:
    Edge(Vertex* source, Vertex* destination, int edgeId , double weight);
    [[nodiscard]] Vertex* getSource() const;
    [[nodiscard]] Vertex* getDestination() const;
    [[nodiscard]] double getWeight() const;
    [[nodiscard]] double getPheromones() const;
    [[nodiscard]] int getId() const;
    bool isSelected() const;

    void setSource(Vertex* newSource);
    void setDestination(Vertex* newDestination);
    void setWeight(double newWeight);
    void setId(int newId);
    void setSelected(bool selected);
    void setPheromones(double pheromones);
private:
    Vertex* source;
    Vertex* destination;
    double weight;
    int id;
    bool selected;
    double pheromones;
};
```

A classe Edge é essencial para representar e manipular as conexões entre vértices em algoritmos de grafos, permitindo operações como cálculo de caminhos mínimos, construção de árvores geradoras, e aplicação de algoritmos.

O Grafo

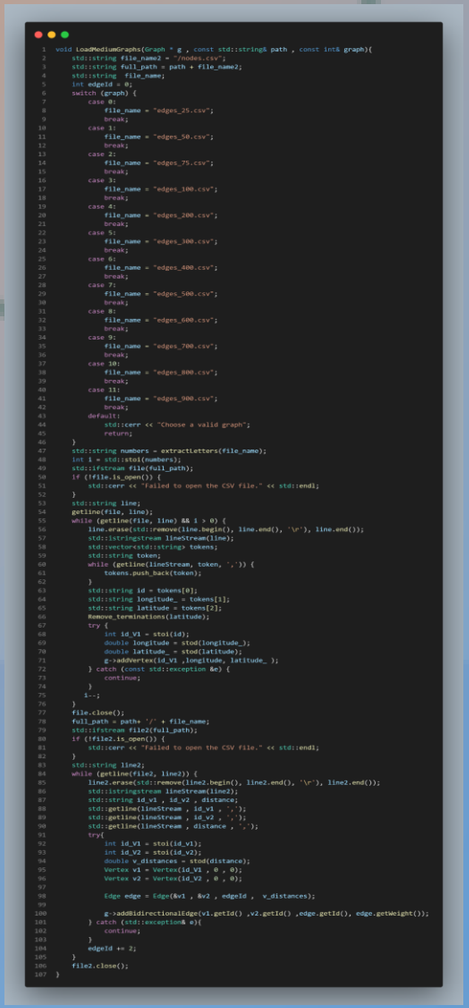
A classe Graph é uma implementação abrangente de um grafo, permitindo operações completas de criação, modificação e consulta. Os métodos fornecidos permitem adicionar e remover vértices e arestas, encontrar elementos específicos, calcular distâncias geográficas, imprimir informações detalhadas e garantir que o grafo esteja totalmente conectado.

```cpp
class Graph {
public:
    ~Graph();
    Edge* findEdge(int sourceId, int destId) const;
    Vertex* findVertex(int id) const;
    bool addVertex(int id , double longitude , double latitude);
    bool removeVertex(int id);
    bool addEdge(int sourceId, int destId,int edgeId, double weight) ;
    bool removeEdge(int sourceId, int destId) const;
    bool addBidirectionalEdge(int sourceId, int destId,int edgeId, double weight) ;
    int getNumVertex() const;
    Clock getClock();
    std::unordered_map<int, Vertex*> getVertexSet() const;
    std::unordered_map<int, Edge*> getEdgeSet() const;

    double Harverstein(double longitude1, double latitude1, double longitude2, double latitude2) const;
    void clear();
    void printNodesContente() const;
    void printGraphInfo() const;
    void populate_in_and_out_degree();
    bool makeFullyConnected() ;

    void orderVertexAdj(Vertex * v);

private:

    Clock clock;

    std::unordered_map<int, Edge*> edgeSet;
    std::unordered_map<int, Vertex*> vertexSet;
};

```

Descrição da leitura do dataset a partir dos ficheiros dados

A leitura e processamento dos datasets são cruciais para o funcionamento do sistema de gerência de abastecimento de água. Esta operação é realizada por meio de várias classes e métodos específicos, garantindo que os dados sejam carregados de forma correta e eficientemente no sistema.

Função responsável por carregar os dados das cidades a partir de um arquivo CSV e criar objetos DeliverySite para representar essas cidades.



```cpp
void LoadMediumGraphs(Graph * g , const std::string& path , const int& graph){
    std::string file_name2 = "/nodes.csv";
    std::string full_path = path + file_name2;
    std::string  file_name;
    int edgeId = 0;
    switch (graph) {
        case 0:
            file_name = "edges_25.csv";
            break;
        case 1:
            file_name = "edges_50.csv";
            break;
        case 2:
            file_name = "edges_75.csv";
            break;
        case 3:
            file_name = "edges_100.csv";
            break;
        case 4:
            file_name = "edges_200.csv";
            break;
        case 5:
            file_name = "edges_300.csv";
            break;
        case 6:
            file_name = "edges_400.csv";
            break;
        case 7:
            file_name = "edges_500.csv";
            break;
        case 8:
            file_name = "edges_600.csv";
            break;
        case 9:
            file_name = "edges_700.csv";
            break;
        case 10:
            file_name = "edges_800.csv";
            break;
        case 11:
            file_name = "edges_900.csv";
            break;
        default:
            std::cerr << "Choose a valid graph";
            return;
    }
    std::string numbers = extractLetters(file_name);
    int i = std::stoi(numbers);
    std::ifstream file(full_path);
    if (!file.is_open()) {
        std::cerr << "Failed to open the CSV file." << std::endl;
    }
    std::string line;
    getline(file, line);
    while (getline(file, line) && i > 0) {
        line.erase(std::remove(line.begin(), line.end(), '\r'), line.end());
        std::istringstream lineStream(line);
        std::vector<std::string> tokens;
        std::string token;
        while (getline(lineStream, token, ',')) {
            tokens.push_back(token);
        }
        std::string id = tokens[0];
        std::string longitude_ = tokens[1];
        std::string latitude = tokens[2];
        Remove_terminations(latitude);
        try {
            int id_V1 = stoi(id);
            double longitude = stod(longitude_);
            double latitude_ = stod(latitude);
            g->addVertex(id_V1 ,longitude, latitude_ );
        } catch (const std::exception &e) {
            continue;
        }
        i--;
    }
    file.close();
    full_path = path+ '/' + file_name;
    std::ifstream file2(full_path);
    if (!file2.is_open()) {
        std::cerr << "Failed to open the CSV file." << std::endl;
    }
    std::string line2;
    while (getline(file2, line2)) {
        line2.erase(std::remove(line2.begin(), line2.end(), '\r'), line2.end());
        std::istringstream lineStream(line2);
        std::string id_v1 , id_v2 , distance;
        std::getline(lineStream , id_v1 , ',');
        std::getline(lineStream , id_v2 , ',');
        std::getline(lineStream , distance , ',');
        try{
            int id_V1 = stoi(id_v1);
            int id_V2 = stoi(id_v2);
            double v_distances = stod(distance);
            Vertex v1 = Vertex(id_V1 , 0 , 0);
            Vertex v2 = Vertex(id_V2 , 0 , 0);

            Edge edge = Edge(&v1 , &v2 , edgeId ,  v_distances);

            g->addBidirectionalEdge(v1.getId() ,v2.getId() ,edge.getId(), edge.getWeight());
        } catch (std::exception& e){
            continue;
        }
        edgeId += 2;
    }
    file2.close();
}
```

Descrição da leitura do dataset a partir dos ficheiros dados

Esta função é responsável por carregar os dados dos reservatórios de água a partir de um arquivo CSV e criar objetos DeliverySite para representar esses reservatórios.

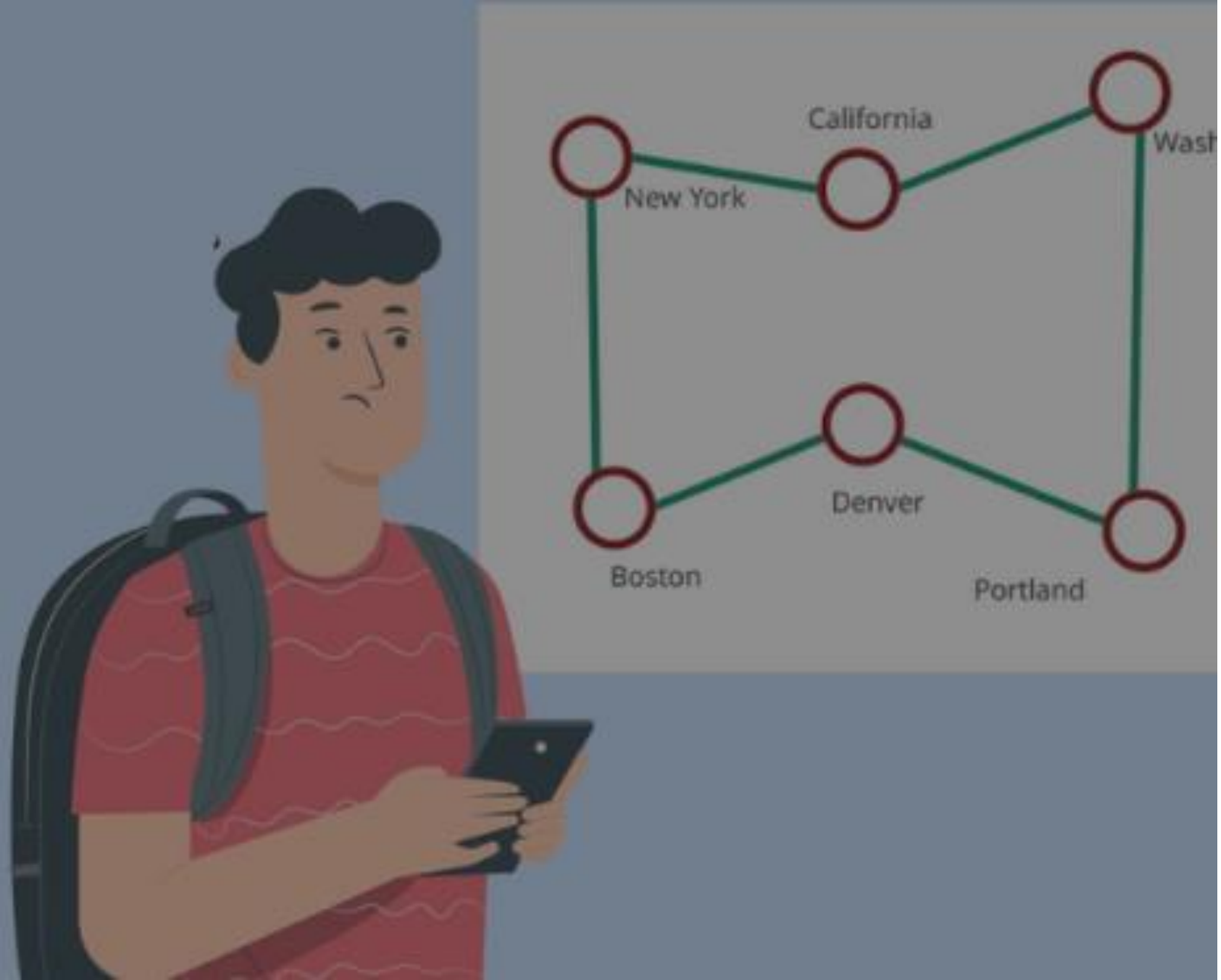
```cpp
void LoadToyGraphs(Graph * g , const std::string& path , const int& graph){
    std::string file_name;
    int edgeId = 0;
    switch (graph) {
        case 0:
            file_name = "shipping.csv";
            break;
        case 1:
            file_name = "stadiums.csv";
            break;
        case 2:
            file_name = "tourism.csv";
            break;
        case 3:
            file_name = "myGraph.csv";
            break;
        default:
            std::cerr << "Choose a valid graph";
            return;
    }
    std::string full_path = path + '/' + file_name;
    std::ifstream file(full_path);
    if (!file.is_open()) {
        std::cerr << "Failed to open the CSV file." << std::endl;
    }
    std::string line;
    getline(file, line);
    while (getline(file, line)) {
        line.erase(std::remove(line.begin(), line.end(), '\r'), line.end());
        std::istringstream lineStream(line);
        std::string id_v1 , id_v2 , distance;
        getline(lineStream , id_v1 , ',');
        getline(lineStream , id_v2 , ',');
        getline(lineStream , distance , ',');
        int id_V1 = stoi(id_v1);
        int id_V2 = stoi(id_v2);
        double v_distances = stod(distance);
        g->addVertex(id_V1 , DBL_MAX , DBL_MAX);
        g->addVertex(id_V2 , DBL_MAX , DBL_MAX);
        g->addEdge(id_V1 , id_V2 ,edgeId, v_distances);
        edgeId++;
        g->addEdge(id_V2 , id_V1 ,edgeId, v_distances);
        edgeId++;
    }
    file.close();
}
```

```cpp
void LoadRealWorldGraphs(Graph * g , const std::string& path , const int& graph){
    std::string file_name;
    int edgeId = 0;
    switch (graph) {
        case 0:
            file_name = "/graph1/nodes.csv";
            break;
        case 1:
            file_name = "/graph2/nodes.csv";
            break;
        case 2:
            file_name = "/graph3/nodes.csv";
            break;
        default:
            std::cerr << "Choose a valid graph";
            return;
    }
    std::string full_path = path+ '/' + file_name;
    std::ifstream file(full_path);
    if (!file.is_open()) {
        std::cerr << "Failed to open the CSV file." << std::endl;
    }
    std::string line;
    getline(file, line);
    while (getline(file, line)) {
        line.erase(std::remove(line.begin(), line.end(), '\r'), line.end());
        std::istringstream lineStream(line);
        std::string id , intitude , latitude;
        getline(lineStream , id , ',');
        getline(lineStream , intitude , ',');
        getline(lineStream , latitude , ',');
        int id_V1 = stoi(id);
        double intitude_ = stod(intitude);
        double latitude_ = stod(latitude);
        g->addVertex(id_V1 , intitude_ , latitude_);
    }
    file.close();
    switch (graph) {
        case 0:
            file_name = "/graph1/edges.csv";
            break;
        case 1:
            file_name = "graph2/edges.csv";
            break;
        case 2:
            file_name = "/graph3/edges.csv";
            break;
        default:
            std::cerr << "Choose a valid graph";
            return;
    }
    full_path = path+ '/' + file_name;
    std::ifstream file2(full_path);
    if (!file2.is_open()) {
        std::cerr << "Failed to open the CSV file." << std::endl;
    }
    std::string line2;
    getline(file2, line2);
    while (getline(file2, line2)) {
        line2.erase(std::remove(line2.begin(), line2.end(), '\r'), line2.end());
        std::istringstream lineStream(line2);
        std::string id_v1 , id_v2 , distance;
        std::getline(lineStream , id_v1 , ',');
        std::getline(lineStream , id_v2 , ',');
        std::getline(lineStream , distance , ',');
        int id_V1 = stoi(id_v1);
        int id_V2 = stoi(id_v2);
        double v_distances = stod(distance);
        Vertex v1 = Vertex(id_V1 , 0 , 0);
        Vertex v2 = Vertex(id_V2 , 0 , 0);
        Edge edge = Edge(&v1 , &v2 ,edgeId, v_distances);
        g->addBidirectionalEdge(v1.getId() , v2.getId() , edge.getId() , edge.getWeight());
        edgeId += 2;
    }
    file2.close();
}
```

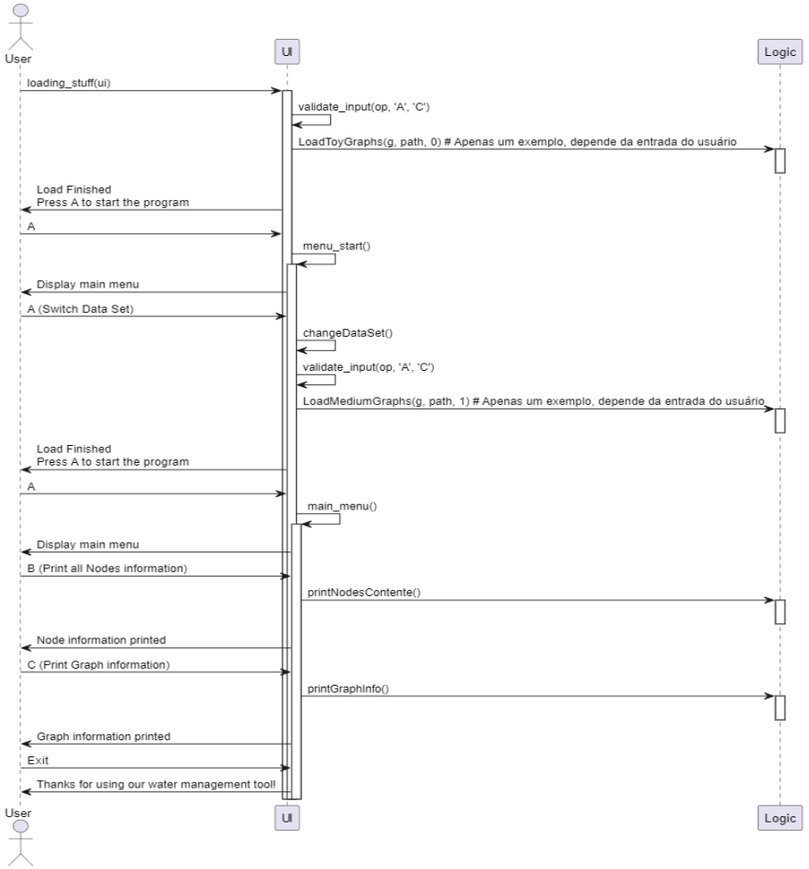
Função responsável por carregar os dados das estações de bombeamento a partir de um arquivo CSV e criar objetos DeliverySite para representar essas estações.

Ui

- **Menu Interativo:**
- O sistema conta com um menu interativo implementado no arquivo UI.cpp. Este menu oferece uma interface de usuário amigável e fácil de usar, que contém a implementação das funções necessárias para interagir com o usuário e conduzir as operações principais do sistema de análise de gerenciamento de abastecimento de água

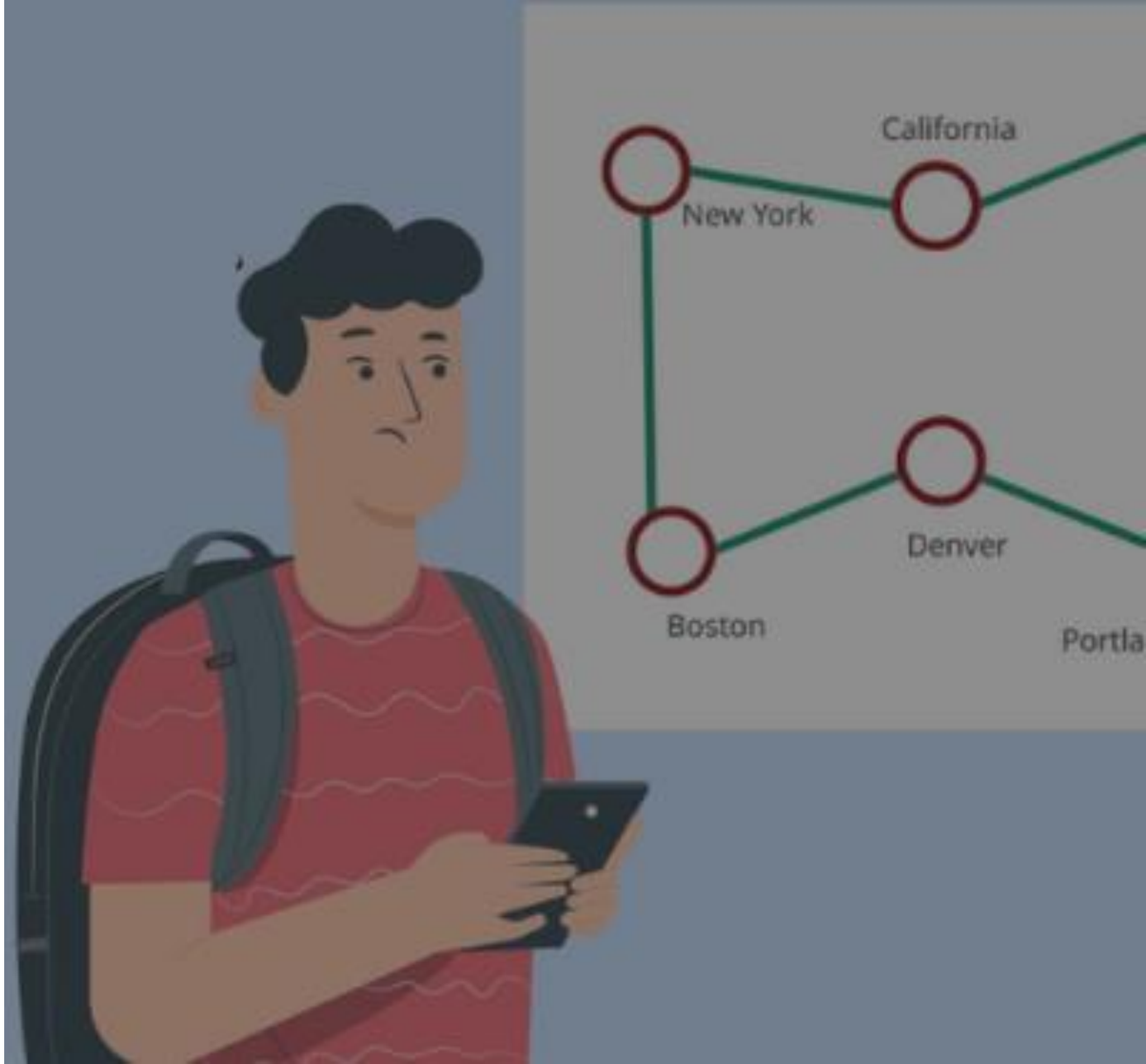

California
New York
Denver
Boston
Portland

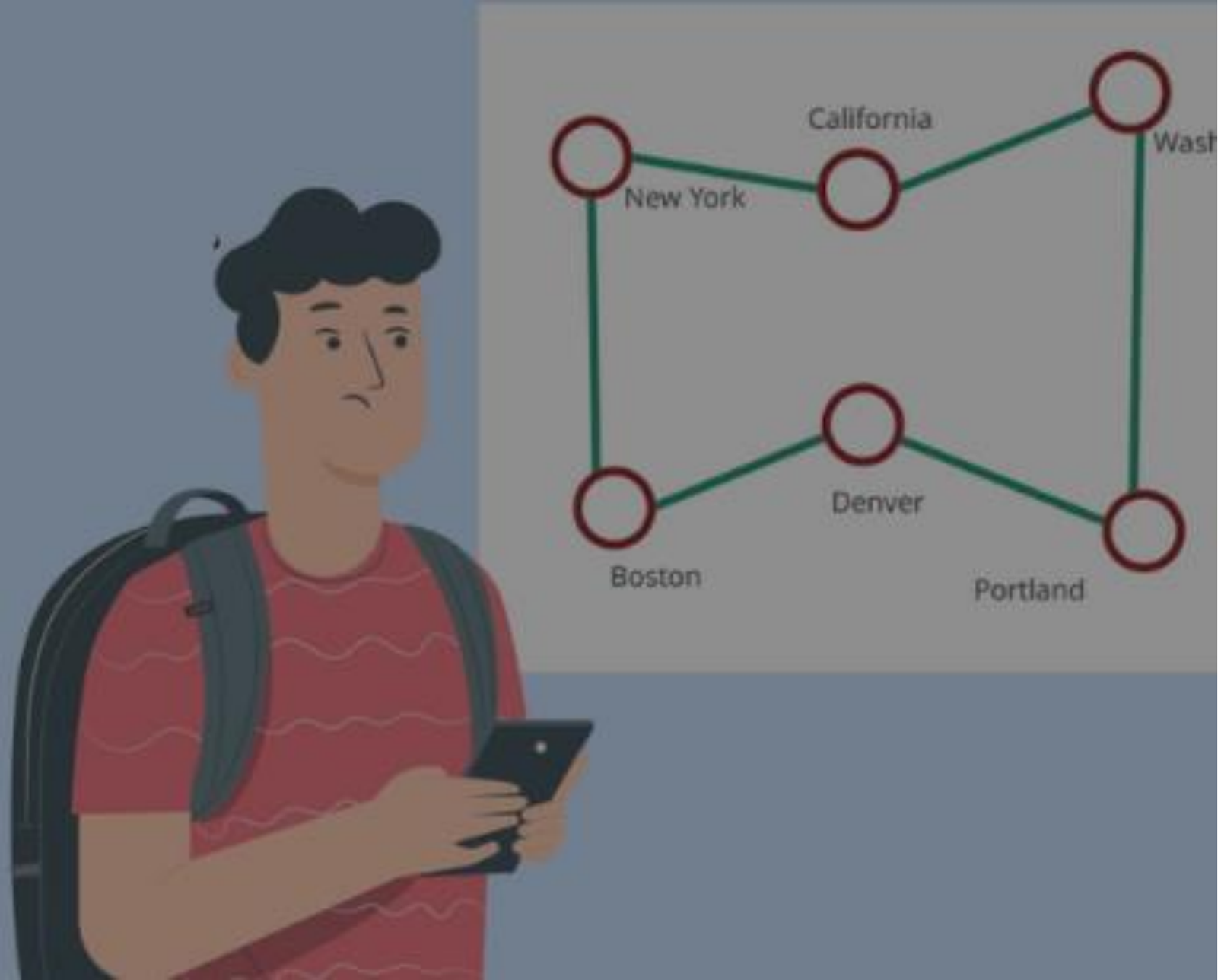
Ui


User
UI
Logic
loading_stuff(ui)
validate_input(op, 'A', 'C')
LoadToyGraphs(g, path, 0) # Apenas um exemplo, depende da entrada do usuário
Load Finished
Press A to start the program
A
menu_start()
Display main menu
A (Switch Data Set)
changeDataSet()
validate_input(op, 'A', 'C')
LoadMediumGraphs(g, path, 1) # Apenas um exemplo, depende da entrada do usuário
Load Finished
Press A to start the program
A
main_menu()
Display main menu
B (Print all Nodes information)
printNodesContente()
Node information printed
C (Print Graph information)
printGraphInfo()
Graph information printed
Exit
Thanks for using our water management tool!
User
UI
Logic

TWOOPT

- **O algoritmo 2-opt tenta otimizar o caminho trocando pares de arestas, sempre que isso resulta em um caminho mais curto. Ele continua fazendo isso até que nenhuma melhoria adicional seja encontrada ou até que o número máximo de iterações seja atingido. Finalmente, ele calcula e retorna o custo total do caminho otimizado.**



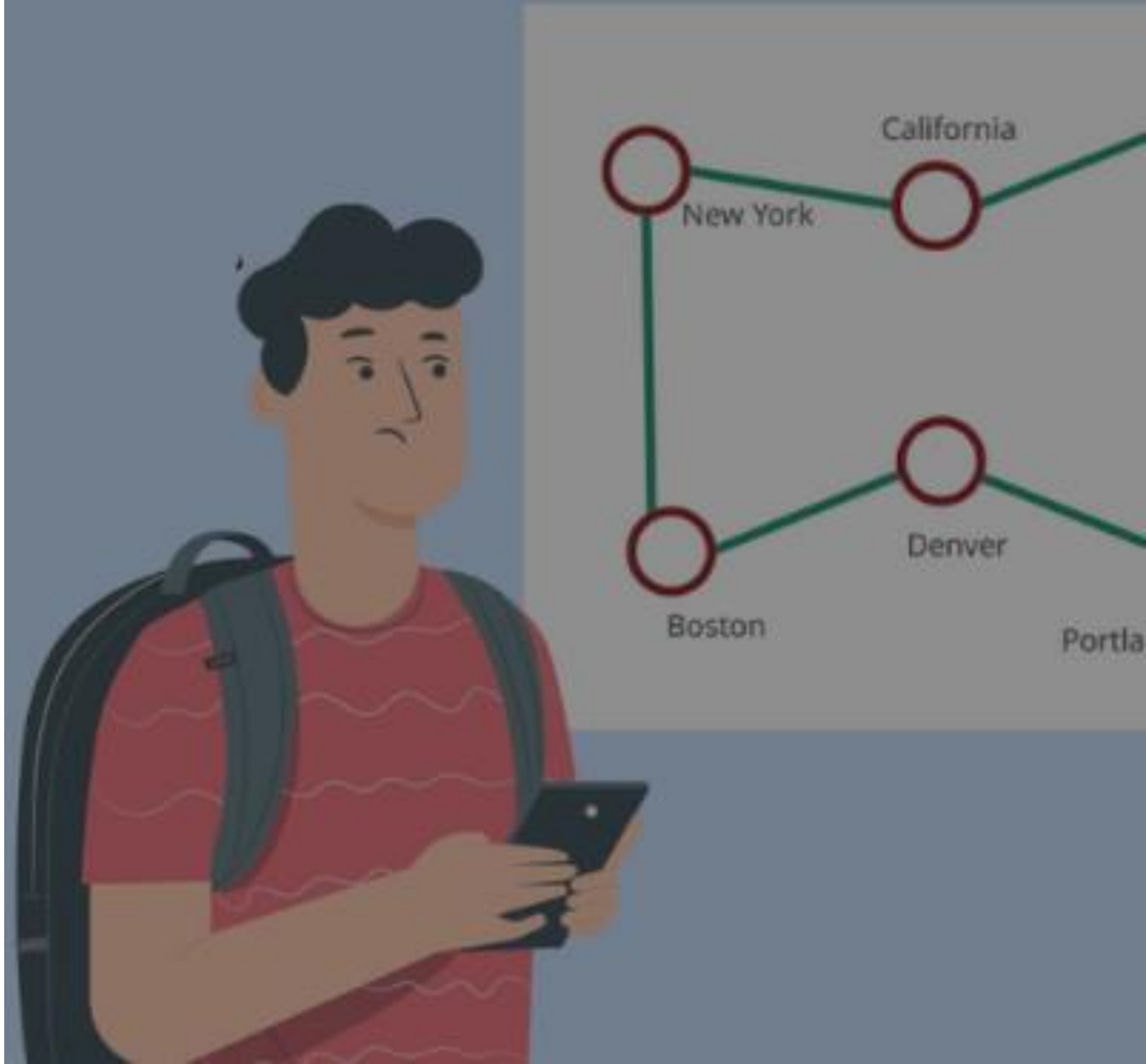


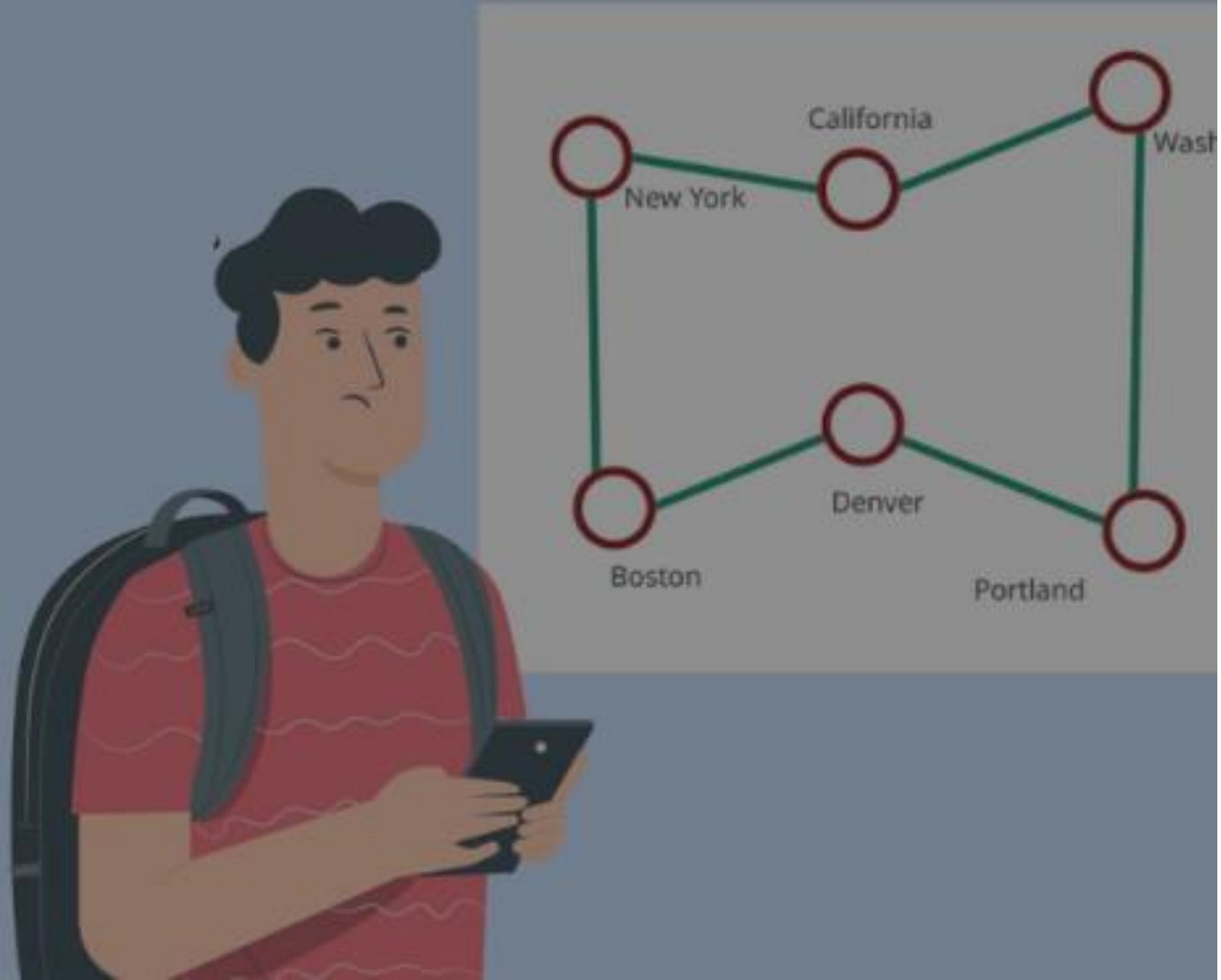
Triangular Aproximation

- **O Triangular Aproximation Tenta conectar todos os vértices do grafo, garantindo que exista um caminho entre quaisquer dois vértices. Tambem Implementa o algoritmo de Prim para construir a MST, minimizando o custo total das conexões entre vértices. Realiza uma travessia em pré-ordem na MST, gerando uma sequência de vértices que representa uma rota inicial aproximada. Procura uma aresta existente entre dois vértices. Se a aresta não existir, uma nova aresta é criada usando a distância calculada pela fórmula de Haversine. Calcula a distância entre dois pontos geográficos usando a fórmula de Haversine, que considera a curvatura da Terra.**

Backtracking

o backtracking é responsável por explorar todas as permutações possíveis de vértices para encontrar o caminho mínimo que visita cada cidade exatamente uma vez e retorna à cidade inicial. Ele garante que todas as soluções viáveis sejam consideradas e que a solução ótima seja encontrada.

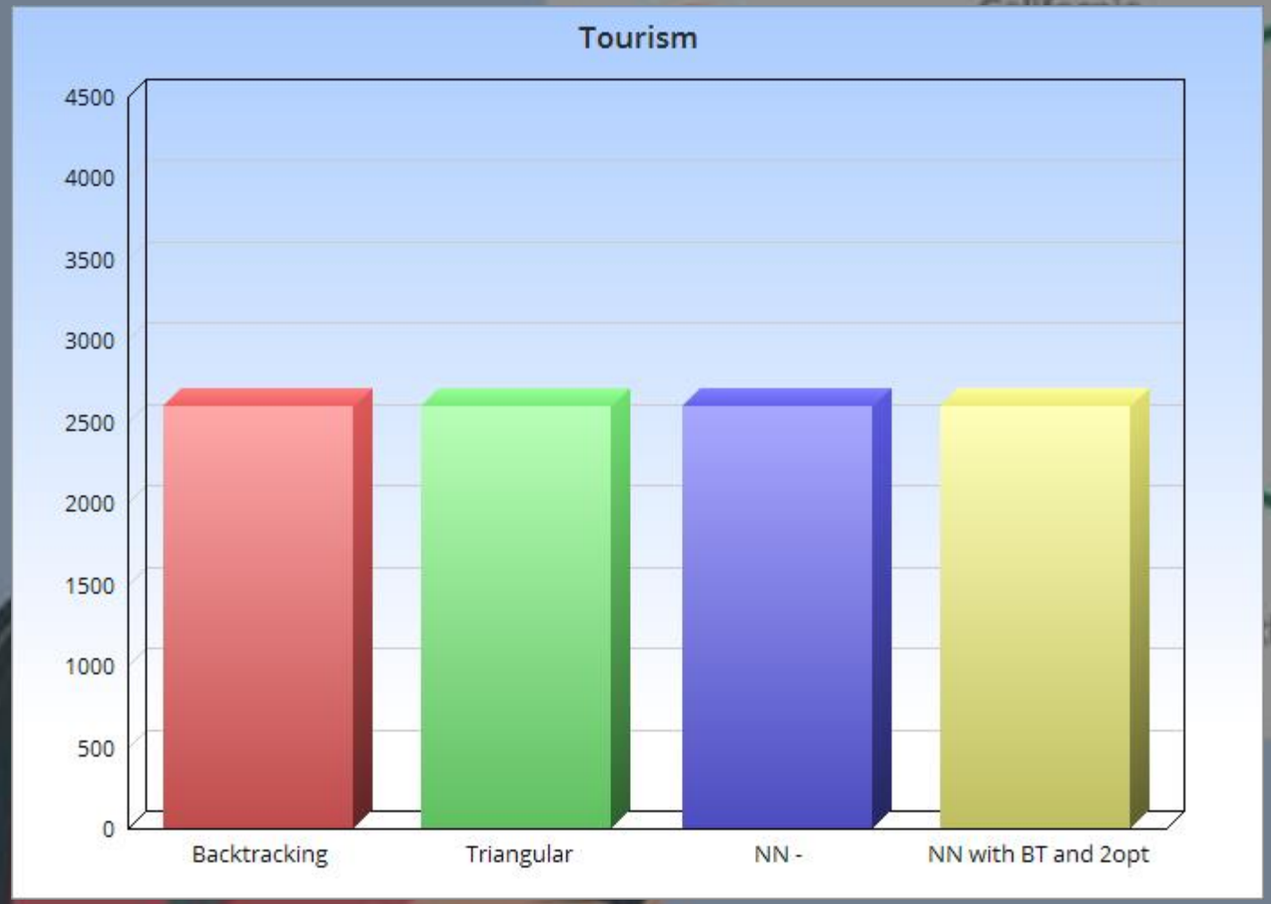


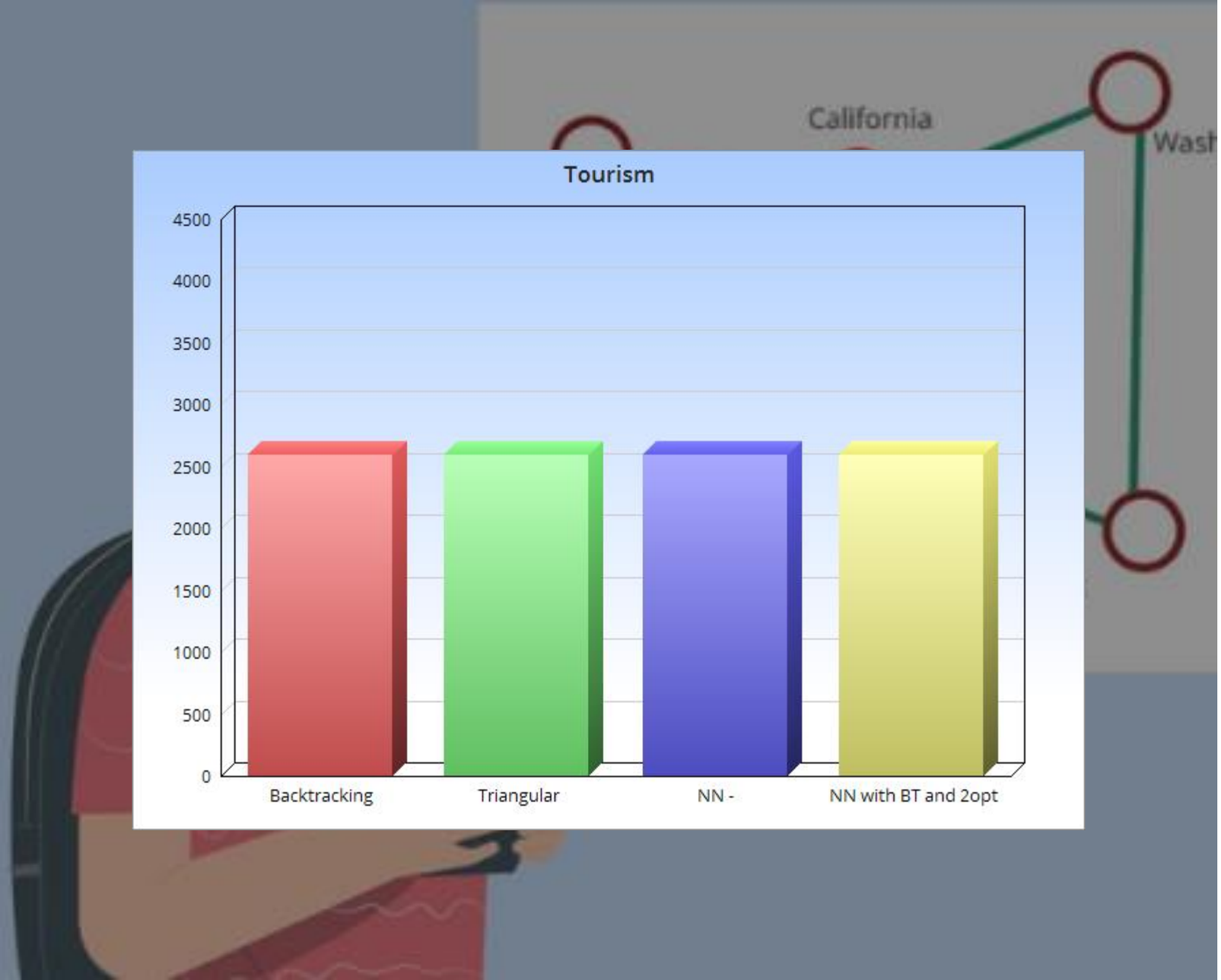


Nearest Neighbour

O algoritmo NearestNeighbour funciona bem para obter uma solução rápida e aproximada para o TSP, mas não garante a solução ótima. Sua simplicidade e rapidez fazem dele uma boa escolha para problemas grandes onde soluções exatas são impraticáveis. A lógica central é sempre escolher o próximo vértice mais próximo ainda não visitado, o que pode não resultar no caminho mais curto, mas geralmente fornece uma solução razoavelmente boa.

Backtracking





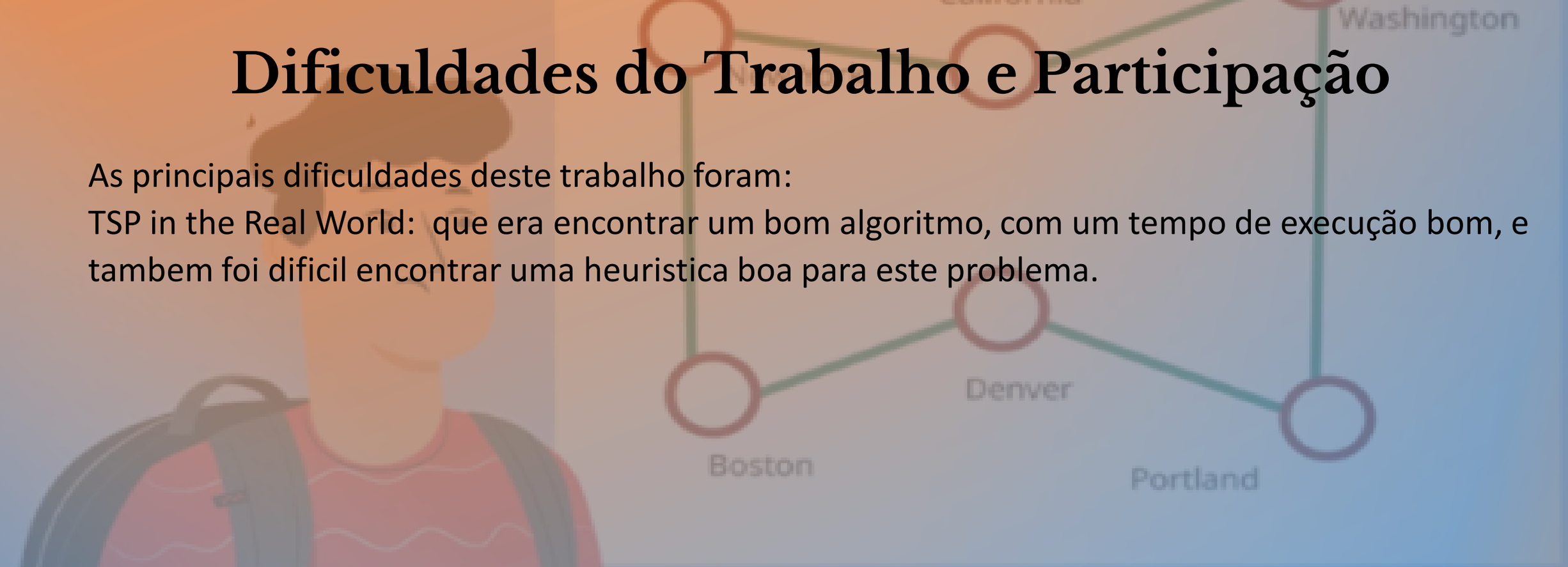
Some results

Destaque de Funcionalidades

O Destaque neste Projeto foram:
TSP in the Real World: Foram usados 3 algoritmos que foram: Nearest Neighbour com Bactracking e TwoOpt, com estes algoritmos Podemos ver como a combinação inteligente de diferentes técnicas pode resultar em um algoritmo poderoso e eficiente, capaz de resolver problemas desafiadores como o TSP com alta eficácia.

Dificuldades do Trabalho e Participação

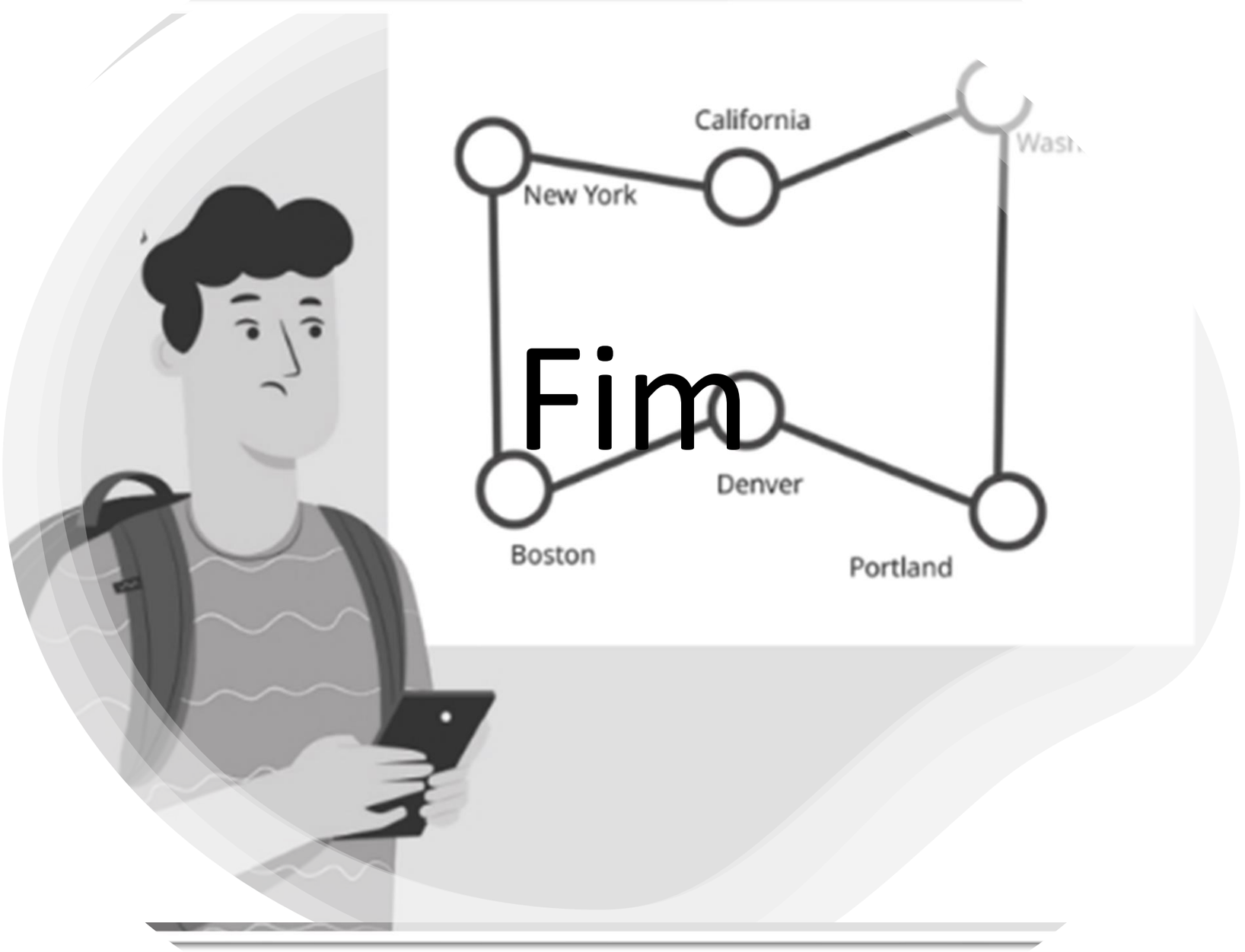
As principais dificuldades deste trabalho foram:
TSP in the Real World: que era encontrar um bom algoritmo, com um tempo de execução bom, e tambem foi dificil encontrar uma heuristica boa para este problema.



Up202207798 – Ângelo Oliveira– 33,3%
Up202209730 – Bruno Fortes – 33,3%
Up202207871 – José Costa – 33,3%

Considerações finais

Este projeto foi uma oportunidade valiosa para aplicar os algoritmos que aprendemos em sala de aula na resolução de problemas do mundo real. Foi uma prova concreta de que esses algoritmos não são apenas teoria académica, mas ferramentas poderosas que têm aplicações práticas em diversos contextos. Pudemos também perceber como os algoritmos de grafos, cálculo de rotas mais curtas e determinação de fluxos máximos podem ser utilizados para resolver problemas reais. Essa experiência preparou-nos para enfrentar desafios semelhantes no futuro, seja na academia, em ambientes profissionais ou até mesmo em projetos pessoais. Além disso, essa vivência mostrou-nos a importância de compreender não apenas a teoria por trás dos algoritmos, mas também sua aplicação prática. Isso permite-nos não apenas resolver problemas de forma eficiente, mas também nos dá uma base sólida para propor soluções inovadoras e eficazes em diversos cenários


California
New York
Boston
Denver
Portland
Fim