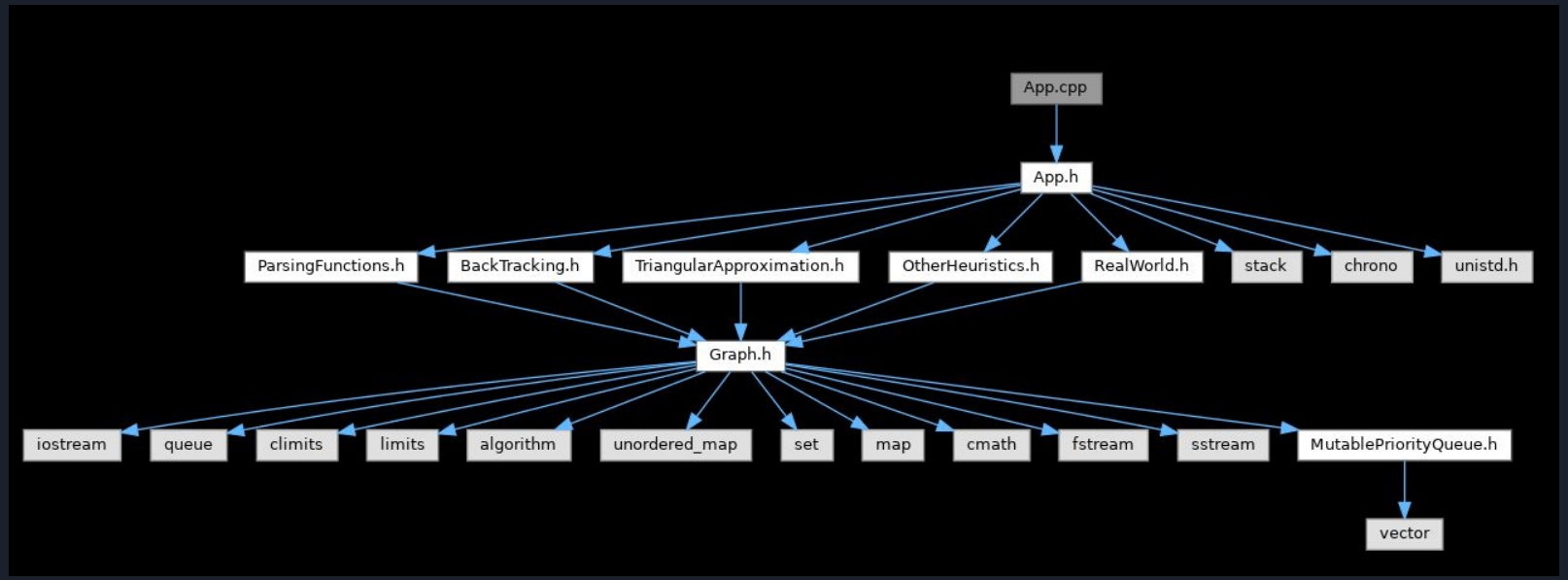
L.EIC Ocean Shipping and Urban Deliveries

Grupo 3 2LEIC13

Alexandre Ramos up202208028@up.pt
Filipa Fidalgo up202208039@up.pt
Francisco Afonso up202208115@up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Class Diagram



Parsing Functions

Foram criadas funções para ler os diferentes tipos de grafos fornecidos:

`read_from_toy_graph`

`read_from_fully_connected_graph`

`read_from_real_graph`

`read_from_nodes_fully_connected`

`read_from_nodes_real`

`read_from_edges`

Todos os grafos foram adicionadas edges não direcionadas.
Foi utilizada uma função `check_vertices()` na construção dos fully connected graphs.

Graph

Foi utilizado um grafo com construtores capazes de suportar os diferentes formatos de CSV fornecidos.

Estruturas de Dados

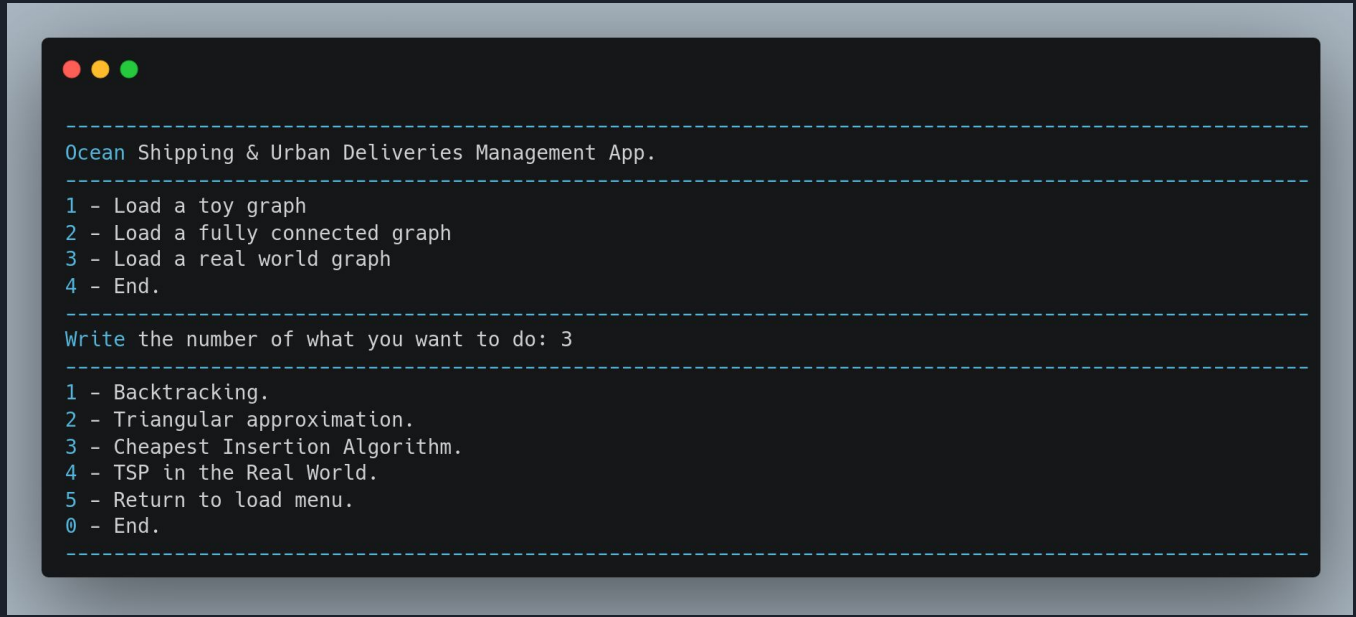
Stack
Utilizado para a troca de menus.

HashMap
Utilizado em funções de procura.

Vector e Set
Utilizado para armazenar os vértices do grafo.

Priority Queue
Utilizado no algoritmo de Prim.

Menu Principal



Funcionalidades

BackTracking

$V!$

Triangular Approximation

$(V + E) \log(V)$

Cheapest Insertion

$V^2 * E$

Nearest Neighbor

$V * E * \log^2(V)$

Backtracking

`findTSP(Graph* graph);`

Function that finds a solution to the TSP using the backtracking algorithm.

Time Complexity: O(V!)

```
void tspUtil(Graph* graph, Vertex *v, double currentWeight, int count, double &minWeight,
std::vector<Vertex*>& currentPath, std::vector<Vertex*>& bestPath) {
    currentPath.push_back(v);

    if (count == graph->getNumVertex() && v->findEdge(graph->getVertexSet().at(0)->getInfo())) {
        double totalWeight = currentWeight + v->findEdge(graph->getVertexSet().at(0)->getInfo())-
>getWeight();
        if (totalWeight < minWeight) {
            minWeight = totalWeight;
            bestPath = currentPath;
            bestPath.push_back(graph->getVertexSet().at(0));
        }
    } else {
        for (auto edge : v->getAdj()) {
            if (!(edge->getDest()->isVisited())) {
                edge->getDest()->setVisited(true);
                tspUtil(graph, edge->getDest(),  currentWeight + edge->getWeight(), count + 1,
minWeight, currentPath, bestPath);
                edge->getDest()->setVisited(false);
            }
        }
    }

    currentPath.pop_back();
}


TourResult findTSP(Graph* graph) {

    TourResult result;
    std::vector<Vertex*> bestPath;
    double minWeight = INT_MAX;
    std::vector<Vertex*> currentPath;

    graph->getVertexSet().at(0)->setVisited(true);
    tspUtil(graph, graph->getVertexSet().at(0), 0, 1, minWeight, currentPath, bestPath);
    graph->getVertexSet().at(0)->setVisited(false);

    result.totalDistance = minWeight;
    result.tour = bestPath;
    return result;
}
```

Triangular Approximation

`triangularApproximation(Graph* graph);`

Function that finds a solution to the TSP using the triangular approximation algorithm.

Time Complexity: O((V + E) * log(V))

```
TourResult triangularApproximation(Graph* graph){
    TourResult result;
    auto order = prim(graph);
    if (order.empty()) return result;
    result.totalDistance = 0.0;

    for(auto v = 0; v < order.size()-1; v++){
        double distance = order[v]->getDist();

        result.totalDistance += distance;
    }

    result.tour = order;
    return result;

}
```

Cheapest Insertion

```
TourResult cheapestInsertion(Graph* graph) {
    vector<Vertex*> tour;

    tour.push_back(graph->getVertexSet()[0]);

    double minDist = INT_MAX;
    Vertex* aux = nullptr;

    for (auto v : graph->getVertexSet()) { // V * E
        double weight = getWeight(*graph, graph->getVertexSet()[0], v);
        if (weight < minDist) {
            minDist = weight;
            aux = v;
        }
    }

    tour.push_back(aux);

    while (tour.size() < graph->getVertexSet().size()) {
        aux = graph->findVertex(0);
        minDist = INT_MAX;

        for (auto v : graph->getVertexSet()) {
            if (find(tour.begin(), tour.end(), v) == tour.end()) {
                double nearestDist = INT_MAX;

                for (auto j : tour) {
                    double dist = getWeight(*graph, j, v);
                    if (dist < nearestDist) {
                        nearestDist = dist;
                        aux = v;
                    }
                }

                if (nearestDist < minDist) {
                    minDist = nearestDist;
                }
            }
        }
```

```
        unsigned int idx = 0;
        double minWeight = INT_MAX;

        for (unsigned int i = 0; i < tour.size(); i++) {
            unsigned int nextIndex = i + 1;
            if (nextIndex > tour.size() - 1) nextIndex = 0;
            double weight = getWeight(*graph, tour[i], aux) + getWeight(*graph, aux, tour[nextIndex]) -
getWeight(*graph, tour[i], tour[nextIndex]);
            if (weight < minWeight) {
                minWeight = weight;
                idx = nextIndex;
            }
        }

        tour.insert(tour.begin() + idx, aux);
    }


    TourResult result;
    result.tour = tour;

    for (unsigned int i = 0; i < tour.size() - 1; i++) {
        result.totalDistance += getWeight(*graph, tour[i], tour[i + 1]);
    }

    return result;
}
```

Cheapest Insertion

```
cheapestInsertion(Graph* graph);
```

Function that finds a solution to the TSP using the cheapest insertion algorithm.

Time Complexity: O(V^2 * E)

Vantagem:

- A solução obtida por este algoritmo é bastante melhor em comparação com o triangular approximation.

Desvantagem:

- O tempo de execução é significativamente maior.

Real World TSP

- O objetivo deste tópico era desenvolver um algoritmo que, caso exista um caminho, forneça um caminho o mais próximo possível do ideal em tempo útil para o problema tsp, mesmo em grafos que não têm garantia de estarem totalmente conectados.
- A nosso ver, isto só seria possível com backtracking, isto porque este algoritmo verifica todas as opções.
- Mas, como os real world graphs são grafos com diversos nós, o algoritmo não correria em tempo útil.

Real World TSP

- Existem algoritmos que, apesar de poderem encontrarem ciclos hamiltonianos, não garantem que o encontrem, como o Nearest Neighbour Algorithm.

- O algoritmo que implementámos é baseado no triangular approximation algorithm.
- Este fornece o caminho correto se este for a MST. No caso de o caminho fornecido pela MST ter edges em falta o algoritmo retorna que não é possível encontrar um caminho, mesmo que haja um caminho que não passe pela MST.

Real World TSP

`realWorld(Graph& graph, double firstVertex);`

Function that finds a solution to the TSP using the mst of the graph (based on triangular approximation) and prints the resulting tour as well as it's total distance.

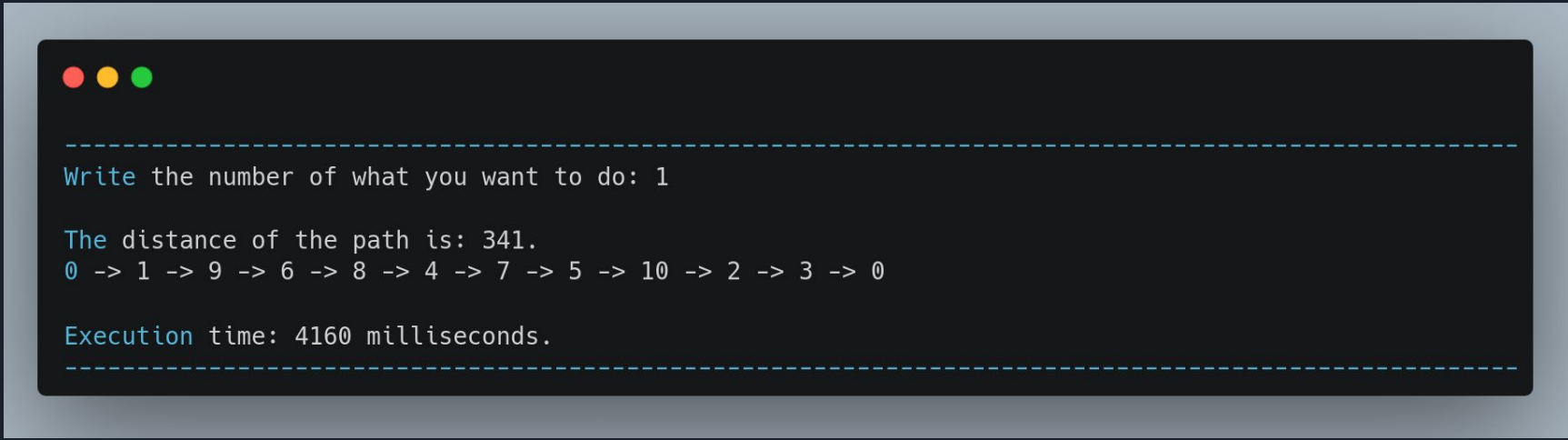
Time Complexity: $O(V * E * \log^2(v))$.

```
void realWorld(Graph &graph, double firstVertex) {
    TourResult result;
    vector<Vertex *> vertexSet = graph.getVertexSet();

    result.tour = prim(graph, firstVertex);

    bool exists = false;
    Vertex *last = graph.findVertex(result.tour.back()->getInfo());
    for (auto e: last->getAdj()) {
        if (e->getDest()->getInfo() == firstVertex) {
            exists = true;
            result.tour.push_back(e->getDest());
            result.totalDistance += e->getWeight();
            break;
        }
    }

    if (!exists) {
        result.totalDistance = 0;
        cout << "There is no feasible path" << endl;
        return;
    }

    double distanceSum = 0;
    for (int i = 0; i < result.tour.size() - 1; i++) {
        cout << result.tour[i]->getInfo() << /*" - " << distance << */" -> ";

        double distance = result.tour[i]->getDist();
        distanceSum += distance;
    }
    cout << result.tour.back()->getInfo() << endl;
    cout << "Total distance covered: " << distanceSum << endl;
}
```

Resultados e tempos de execução

		BACKTRACKING		TRIANGULAR		CHEAPEST INSERTION	
		TEMPO	DISTÂNCIA	TEMPO	DISTÂNCIA	TEMPO	DISTÂNCIA
TOY	STADIUMS	2247	341	0	398	1	348,6
	SHIPPING	40	86,7				
	TOURISM	0	2600	0	2600	0	2600
FULLY CONNECTED	25			0	348706	17	296062
	50			0	551982	127	453410
	75			1	648225	591	582239
	100			2	699188	1921	582039
	200			6	914653	2922	749396
	300			15	1,20E+06	131013	1,01E+06
	400			20	1,37E+06	399221	1,20E+06
	500			45	1,56E+06	975745	1,25E+06
	600			50	1,67E+06	2151133	1,37E+06
	700			78	1,84E+06	3696885	1,52E+06
	800			100	1,97E+06	7535833	1,65E+06
	900			123	2,13E+06	9614287	1,77E+06
REAL WORLD	1			153	1,18E+06		
	2			1841	3,72E+06		
	3			2850	6,50E+06		

Exemplo de Interface



Dificuldades

Tivemos as seguintes dificuldades ao fazer o projeto:

- Interpretação do enunciado;
- TSP in the Real World

Contribuição:

- Alexandre Ramos - 33.3%
- Filipa Fidalgo - 33.3%
- Francisco Afonso - 33.3%

FIM