Travelling Salesman Problem

2º Projeto de Desenho de Algoritmos

Trabalho realizado por:

António Rama (up202108801@fe.up.pt)

José Veiga(up202108753@fe.up.pt)

Pedro Marcelino(up202108754@fe.up.pt)

Problema

- Implementar diversos algoritmos e heurísticas para resolver o TSP, tendo em conta diversos grafos, onde a abordagem tem de ser diferente para cada um deles

- Leitura e “parsing” dos ficheiros csv dos grafos

- Criação de um menu user-friendly para o user testar os diferentes algoritmos e comparar resultados

Solução

Criação das classes Utils, Menu e Graph

```
Utils::Utils() {...}

void Utils::buildGraphs() {...}

Graph* Utils::buildToyGraph(const string& filename, const string& path) {...}

Graph* Utils::buildRealWorldGraph(unsigned number) {...}

void Utils::buildRealWorldGraphNodes(unsigned number) {...}

void Utils::buildRealWorldGraphEdges(unsigned number) {...}

const vector<Graph*>& Utils::getGraphs() {...}
```

```
class Menu {

private:
    Utils utils;
    std::string command;

public:

    /**...*/
    Menu();

    /**...*/
    void run();


    /**...*/
    static void cleanTerminal();

    /**...*/
    void enterOption(unsigned n);

    /**...*/
    void press0ToContinue();

    /**...*/
    void mainMenu();

    /**...*/
    void graphsMenu();

    /**...*/
    void graphMenu(Graph* graph);

    /**...*/
    static void printGraph(Graph* graph);

    /**...*/
    void algorithmsMenu();

    /**...*/
    void tSPGameMenu();

    /**...*/
    void tSPGame(Graph* graph);
};
```

```
class Graph {

private:

    std::string name;
    bool realOrToy;

    struct Node;

    struct Edge {
        Node* first;
        Node* second;
        double distance;
        bool visited;
    };

    struct Node {
        unsigned id;
        std::vector<Edge*> adj = {};
        double latitude = 0;
        double longitude = 0;
        std::string label;
        bool visited = false;
        Node* parent;
        int degree = 0;
    };

    std::vector<Node*> nodes;
```

Algoritmos implementados

- TSP Backtracking

Devido à sua complexidade temporal de O(|N|!), é eficiente apenas para grafos pequenos

```
void Graph::tSPBacktracking(unsigned currNode, vector<unsigned>& currPath, double currDistance, double& minDistance, vector<unsigned>& bestPath) {

    // Base case: all nodes visited
    if (currPath.size() == nodes.size()) {
        // Check if it forms a better tour
        Edge* edge = findEdge( first: 0, second: currNode);
        edge != nullptr ? currDistance += edge->distance : currDistance = numeric_limits<double>::max();
        if (currDistance < minDistance) {
            minDistance = currDistance;
            bestPath = currPath;
            bestPath.push_back(0);
        }
        return;
    }

    // Try all possible next nodes
    for (auto i :Edge* : nodes[currNode]->adj) {
        unsigned next = i->first->id == currNode ? i->second->id : i->first->id;

        // If the next node hasn't been visited
        if (!nodes[next]->visited && currDistance + i->distance < minDistance) {

            nodes[next]->visited = true;
            currPath.push_back(next);
            currDistance += i->distance;

            // Recursively check next nodes
            tSPBacktracking( currNode: next, &: currPath, currDistance, &: minDistance, &: bestPath);

            nodes[next]->visited = false;
            currPath.pop_back();
            currDistance -= i->distance;
        }
    }
}
```

- TSP 2Approximation Heuristic

Heurística que, através da DFS tree e da desigualdade triangular, encontra uma aproximação não maior que o dobro da solução ótima.

```
double Graph::tSP2Approximation(vector<unsigned>& path) {

    path.clear();
    if (getName() == "shipping") return 0; // Exclude shipping graph

    vector<pair<unsigned, unsigned>> mST;
    mSTPrim( &: mST);

    Graph mSTGraph( name: "mSTGraph", realOrToy);

    for (auto node :Node* : nodes) {
        mSTGraph.addNode(node->id);
    }

    for (auto edge :pair<...> : mST) {
        mSTGraph.addEdge(edge.first, edge.second, findEdge(edge.first, edge.second)->distance);
    }

    vector<unsigned> tree;
    mSTGraph.dfsTree( node: 0, &: tree);
    path.push_back(0);
    double distance = 0;
    unsigned last = 0;

    for (unsigned i = 1; i < tree.size() - 1; i++) {
        if (find( first: path.begin(), last: path.end(), val: tree[i]) == path.end()) {
            path.push_back(tree[i]);
            auto edge :Graph::Edge* = findEdge( first: last, second: tree[i]);
            edge == nullptr ? distance += haversine( first: nodes[last], second: nodes[tree[i]]) : distance += edge->distance;
            last = tree[i];
        }
    }
    path.push_back(0);
    auto edge :Graph::Edge* = findEdge( first: last, second: 0);
    edge == nullptr ? distance += haversine( first: nodes[0], second: nodes[last]) : distance += edge->distance;

    return distance;
}
```

Outros Algoritmos

- TSP Nearest Neighbor Heuristic (escolhe o node mais próximo e “cria edges” caso necessário)
- TSP Greedy Heuristic (escolhe o edge mais barato)
- TSP 1Tree Lower Bound (encontra a melhor estimativa inferior à solução
- Christofides Algorithm (talvez a melhor heurística criada, no máximo 1.5 vezes superior à solução ótima
- TSP 2Opt Improvement (Melhora um path através de local search e swaps de pares de nodes)
- Prim e Kruskal (para calcular MST e ajudar noutros algoritmos
- DFS, Connected Components, Articulation Points, etc (importantes para auxiliar outros algorithmos

Menu interativo e intuitivo

```
-----------------------------------------------
|                  MAIN MENU                  |
|---------------------------------------------|
| 1. GRAPHS MENU                              |
| 2. ALGORITHMS MENU                          |
| 3. TSP GAME MENU                            |
| 4. EXIT                                     |
-----------------------------------------------

   -OPTION:
```

```
-----------------------------------------------
| shipping MENU                               |
|---------------------------------------------|
| 1. N OF NODES, EDGES AND ARTICULATION POINTS  |
| 2. IS IT COMPLETE, CONNECTED AND REAL/TOY?    |
| 3. PRINT GRAPH                              |
| 4. GO BACK                                  |
-----------------------------------------------
```

```
-------------------------------------------------
|                ALGORITHMS MENU                |
|-----------------------------------------------|
| 1. TSP BACKTRACKING                           |
| 2. TSP 2APPROXIMATION HEURISTIC               |
| 3. TSP NEAREST NEIGHBOR HEURISTIC             |
| 4. TSP GREEDY HEURISTIC                       |
| 5. TSP 1TREE LOWER BOUND                      |
| 6. CHRISTOFIDES HEURISTIC                     |
| 7. APPLY ALL ALGORITHMS TO ALL GRAPHS         |
| 8. GO BACK                                    |
-------------------------------------------------
```

Alguns resultados

Indicação do resultado, tempo demorado e caminho percorrido, de forma a comparar resultados e planear abordagens aos diferentes grafos.

TSP BACKTRACKING

GRAPH	RESULT	TIME	PATH
shipping	86.7	14 ms	{0, 1, 11, 10, 12, 13, 3, 2, 4, 6, 9, 7, 8, 5, 0}
stadiums	341	174 ms	{0, 3, 2, 10, 5, 7, 4, 8, 6, 9, 1, 0}
tourism	2600	0 ms	{0, 3, 2, 1, 4, 0}

CHRISTOFIDES

GRAPH	RESULT	TIME	PATH
stadiums	371.3	0 ms	{0, 1, 2, 10, 5, 4, 8, 9, 6, 7, 3, 0}
tourism	2600	0 ms	{0, 3, 2, 1, 4, 0}
edges_25	297900	0 ms	
edges_50	484531	1 ms	
edges_75	599035	2 ms	
edges_100	628415	4 ms	
edges_200	785669	11 ms	
edges_300	1.04566e+06	26 ms	
edges_400	1.2059e+06	51 ms	
edges_500	1.33399e+06	99 ms	
edges_600	1.48668e+06	176 ms	
edges_700	1.57863e+06	258 ms	
edges_800	1.74797e+06	365 ms	
edges_900	1.90542e+06	500 ms	
graph1	1.07561e+06	649 ms	

Testing TSP2Approximation()

GRAPH	RESULT	TIME	PATH
stadiums	398.1	0 ms	{0, 1, 2, 10, 5, 4, 8, 7, 6, 9, 3, 0}
tourism	2600	0 ms	{0, 3, 2, 1, 4, 0}
edges_25	349573	0 ms	
edges_50	554134	0 ms	
edges_75	627035	0 ms	
edges_100	681458	0 ms	
edges_200	909414	2 ms	
edges_300	1.19689e+06	5 ms	
edges_400	1.34421e+06	7 ms	
edges_500	1.49618e+06	10 ms	
edges_600	1.61821e+06	15 ms	
edges_700	1.75767e+06	28 ms	
edges_800	1.8649e+06	33 ms	
edges_900	2.05297e+06	37 ms	
graph1	1.14154e+06	47 ms	
graph2	3.99489e+06	499 ms	
graph3	6.19269e+06	1429 ms	

Todos os resultados!

	tspBacktracking		MSTPrim		tspNNHeuristic		tspGreedyHeuristic		tsp1TreeLowerBound		tspChristofides		tsp2approximation		ChristofidesOpt2improvement	NNopt2improvement
	Resultado	Tempo	Resultado	Tempo	Resultado	Tempo	Resultado	Tempo	Resultado	Tempo	Resultado	Tempo	Resultado	Tempo	Resultado	Resultado
shipping	86.7	6 ms	66.9	0ms	----------	----------	----------	----------	80.5	0ms	----------	----------	----------	----------	----------	----------
stadiums	341	163ms	231.2	0ms	407.4	0ms	368.9	0ms	312.1	0ms	371.3	0ms	398.1	0ms	371.3	374.6
tourism	2600	0ms	1850	0ms	2600	0ms	2600	0ms	2600	0ms	2600	0ms	2600	0ms	2600	2600
edges_25	----------	----------	210 875	0ms	300 952	0ms	322 933	0ms	261 705	1ms	297 900	0ms	349 573	0ms	290 023	293 443
edges_50	----------	----------	329 259	0ms	534 149	0ms	499 841	0ms	357 689	10ms	484 531	0ms	554 134	0ms	472 335	517 292
edges_75	----------	----------	392 094	0ms	613 487	0ms	615 861	1ms	418 677	21ms	599 035	0ms	627 035	0ms	587 664	584 240
edges_100	----------	----------	428 235	0ms	705 267	0ms	627 325	2ms	447 838	51ms	628 415	1ms	681 458	0ms	609 558	689 598
edges_200	----------	----------	547 845	2ms	848 895	0ms	838 268	11ms	562 943	458ms	785 669	10ms	909 414	2ms	772 316	828 094
edges_300	----------	----------	741 702	4ms	1.10M	1ms	1.05M	29ms	758 945	1.33s	1.05M	24ms	1.20M	5ms	1.00M	1.08M
edges_400	----------	----------	834 121	6ms	1.41M	3ms	1.28M	57ms	854 360	1.99s	1.21M	47ms	1.34M	7ms	1.17M	1.31M
edges_500	----------	----------	907 053	8ms	1.37M	5ms	1.32M	96ms	920 972	3.83s	1.33M	93ms	1.50M	10ms	1.30M	1.33M
edges_600	----------	----------	988 822	12ms	1.60M	7ms	1.49M	148ms	1.00M	6.32s	1.49M	166ms	1.62M	15ms	1.42M	1.57M
edges_700	----------	----------	1.08M	24ms	1.72M	10ms	1.68M	222ms	1.09M	14.72s	1.58M	247ms	1.76M	28ms	1.52M	1.67M
edges_800	----------	----------	1.14M	27ms	1.84M	13ms	1.72M	348ms	1.16M	18.90s	1.75M	340ms	1.87M	33ms	1.68M	1.78M
edges_900	----------	----------	1.26M	29ms	1.89M	20ms	1.89M	461ms	1.27M	22.28s	1.91M	467ms	2.05M	37ms	1.81M	1.85M
graph1	----------	----------	653 415	37ms	1.00M	25ms	974 286	634ms	682 289	33.57s	1.08M	604ms	1.14M	47ms	1.02M	975 802
graph2	----------	----------	4.52M	335ms	9.11M	258ms	----------	----------	4.54M	25.58min	----------	----------	3.99M	499ms	----------	----------
graph3	----------	----------	8.09M	856ms	16.75M	678ms	----------	----------	8.11M	2.28h	----------	----------	6.19M	1.4s	----------	----------

Bónus – pequeno jogo interativo de TSP onde o user pode escolher o caminho a percorrer (útil para testar o Nearest Neighbor Heuristic)

```
-Path: 0 -> 1 -> 2 -> 3 -> 4 -> 0
-Distance: 3750
-You have visited all nodes!
```

```
   -Path: 0 -> 1
   -Distance: 1300

Choose a node to visit:
   [2]se Distance: 450
   [3]clerigos Distance: 950
   [4]bolsa Distance: 450
```