TSP Management
Project 2 DA 23/24

•••

G18_3:
David Ranito
Pedro João
Tiago Torres

Classes Diagram

Management -> Management.cpp, Management.h

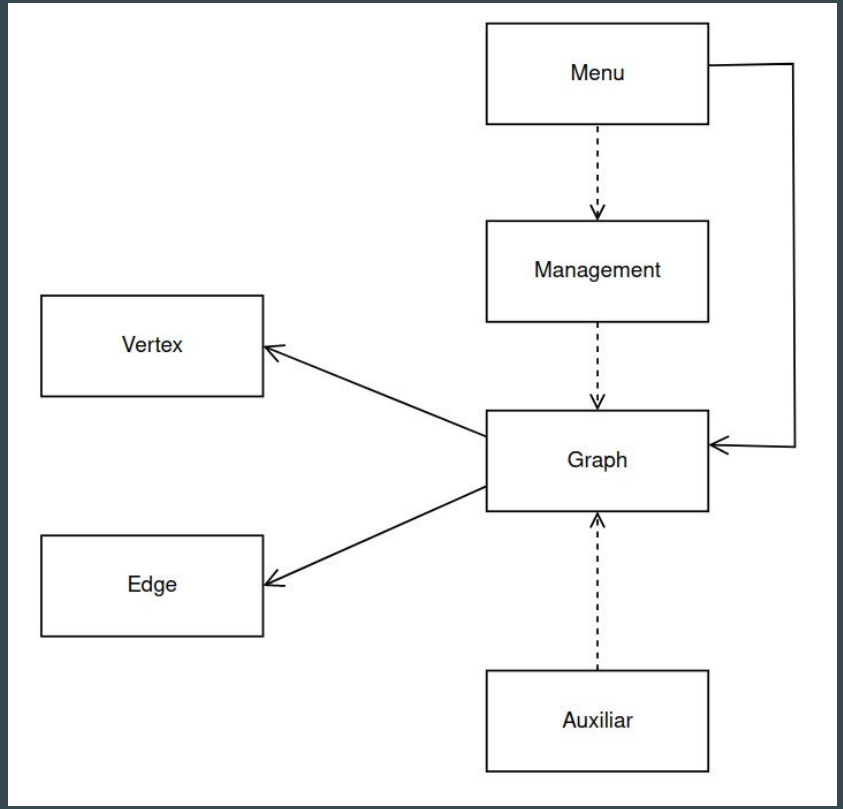
Graph -> Graph.cpp, Graph.h

Menu -> Menu.cpp, Menu.h

Auxiliar -> Auxiliar.cpp, Auxiliar.h, Folders -> Toy_Graphs, Extra_Fully_Connected_Graphs, Real_World_Graphs

Vertex-> Vertex.cpp, Vertex.h

Edge -> Edge.cpp, Edge.h



Reading the Dataset

To read the given data, 5 function were added in the “Auxiliar” class:

- readDataset
- readSmall
- readMedium
- readLarge
- initMatrix

These functions read the respective file line by line, create the appropriate object and adds them to the graph.

```
std::ifstream file( s: filename);
std::string line, orig, dest, distance;
getline( &: file,  &: line);

std::string auxFilename = filename.substr( pos: 43,  n: filename.length());
int nrVertex = std::stoi( str: auxFilename.substr( pos: 0,  n: auxFilename.length() - 4));

 g->setMatrix( newMatrix: Auxiliar::initMatrix( n: nrVertex));

while (std::getline( &: file,  &: line)){
    std::istringstream ss( str: line);
    getline( &: ss,  &: orig,  delim: ',');
    getline( &: ss,  &: dest,  delim: ',');
    getline( &: ss,  &: distance,  delim: '\r');
    g->addVertex( in: std::stoi( str: orig));
    g->addVertex( in: std::stoi( str: dest));
    g->addBidirectionalEdge( sourc: std::stoi( str: orig),  dest: std::stoi( str: dest),  w: std::stod( str: distance));
    g->addToDistMatrix( v1: stoi( str: orig),  v2: stoi( str: dest),  dist: stod( str: distance));
}
```

Reading the Dataset

While reading the datasets, the distances between vertices that don't have an edge between the in the file are calculated using the Haversine distance and added to the distMatrix.

We only needed iterate through half of the matrix (once per bidirectional edge), because the distance from A >B and B->A is the same.

```
for (int i = 0; i < nrVertex - 1; i++) {
    for (int j = 0; j < i + 1; j++){
        if ((g->getDist( v1: i,  v2: j) == 0)) {
            g->addToDistMatrix( v1: i,  v2: j,  dist: Management::getHaversineDist( v1: g->findVertex( in: i),  v2: g->findVertex( in: j)));
```

Graph Description

The graph is built with vertices and and edges.

It has vectors to store the vertices and a matrix with the distances.

```
protected:
    std::vector<Vertex *> vertexSet;
    double** distMatrix;
```

Each Vertex has a group of adjacent and incoming Edges.

Also has some auxiliary attributes to help compute the algorithms.

```
int info;                // info node
std::vector<Edge *> adj;  // outgoing edges

// auxiliary fields
bool visited = false;
bool processing = false;
unsigned int indegree;
double lat = 0;
double lon = 0;
double dist;
Vertex *parent;
std::vector<Vertex *> children;
Edge *path = nullptr;

std::vector<Edge *> incoming; // incoming edges
```

Graph Description - continuation

Each Edge has a origin and a destination Vertex and auxiliary fields.

```
Vertex * dest; // destination vertex
double weight; // edge weight, can also be used for capacity

// auxiliary fields
bool selected = false;

// used for bidirectional edges
Vertex *orig;
Edge *reverse = nullptr;
```

Backtracking Algorithm

Time Complexity: O(V!)

Other important aspects:

- For the time it takes, is infeasible for graphs with highest number of vertices.

```
TSP using a Backtracking Algorithm
 - For graph: Shipping, starting in node 0

Cost: 86.7
Execution time: 151ms
Press 'm' to go back to the main menu.
Press 'q' to quit.
```

```
TSP using a Backtracking Algorithm
 - For graph: Stadiums, starting in node 0

Cost: 341
Execution time: 13617ms
Press 'm' to go back to the main menu.
Press 'q' to quit.
```

```
TSP using a Backtracking Algorithm
 - For graph: Tourism, starting in node 0

Cost: 2600
Execution time: 0ms
Press 'm' to go back to the main menu.
Press 'q' to quit.
```

Triangular Approximation Heuristic

Algorithms used: Prim's for MST and 2-opt triangular approx

Time Complexity: $O(V^2 \log(V))$

Other important aspects:

- The implementation of the min heap for the mst algorithm was not ideal, given that c++ doesn't have an updatable priority queue;

```
TSP using the Triangular Approximation Algorithm
 - For graph: Stadiums, starting in node 0

Cost: 398.1
Execution time: 0ms
Press 'm' to go back to the main menu.
Press 'q' to quit.
```

```
TSP using the Triangular Approximation Algorithm
 - For graph: Graph 1, starting in node 0

Cost: 1.14166e+06
Execution time: 29ms
Press 'm' to go back to the main menu.
Press 'q' to quit.
```

Other Heuristics

Algorithms used: Nearest neighbour algorithm

Time Complexity: $O(V^2)$

Other important aspects:

- Always improved solution, by time or cost

```
TSP using Other Heuristics
 - For graph: Graph 1, starting in node 0

Cost: 1.01724e+06
Execution time: 23ms
Press 'm' to go back to the main menu.
Press 'q' to quit.
```

TSP in the Real World

Algorithms used: Branch and bound

Time Complexity: O(V!)

Other important aspects:

- Because the graphs given are not fully connected and we need to know if there isn't a tour available, it was not possible to achieve a polynomial time;
- If the graph had at least one vertex with less than 2 edges, automatically returns no tour
- A branch would stop from going further if the cost was already higher than one achieved before.

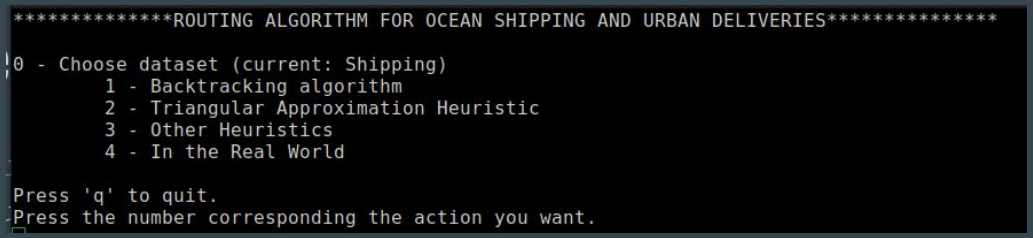
User Interface

The interface is implemented with the Menu class, that is responsible for managing the execution of the program as wished by the user.

All functionalities are displayed in the console by menus.

For each menu the user can choose one option with the corresponding string and then follow the steps that are shown.

The output of the program is saved in a file.



Highlights

- A good planning and a good graph structure contributed for the success of this project.
- The nearest neighbour algorithm improved in time and cost the triangular approximation, especially for large graphs.
- For the complexity involved, the speed and responsiveness of the program is very good.
- The interface has a good look, is intuitive and flexible for user input. The dataset can be chosen and easily changed.

Conclusion

During the project the biggest struggle was to find an algorithm that is successful in the Real World Graphs.

The nearest neighbour algorithm improved the triangular approximation and seems to be a very good choice of heuristics to the tsp.

In general, we developed our critical thinking and realized the importance of good heuristics for this type of problems and also the importance of the time complexity for larger datasets.

All elements equally contributed to the project.