# Routing Algorithm for Ocean Shipping and Urban Deliveries

Work by:
Guilherme Guerra, nº202205140
Tomás Pereira, nº202108845
Sofia Sousa, nº202005932

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Index

# Inner Architecture

For this project, we used the MVC (Model - View - Controller) architecture.

To do this, we used the following classes:

# Parsing

For parsing the datasets, we used 2 ways of doing it .

- Symmetric -> In the symmetric we process both x to y edge , and y to x edge.
- Real mode -> In the real mode we only use what is given on the datasets.

Additional Note: In the future, we might want to improve the parsing time for the real world graphs, since the 2 bigger graphs take 5 and 12 minutes respectively to be processed.  The algorithms efficiency is still very fast for them, but we maybe could change to a graph with matrix representation to faster processing.

# Menu Class

For the Menu, we created a class that holds a pointer to the current menu's functions, and uses that to dynamically switch between menus.

We also use a stack to save previous menus the user has been in, and then use the function goBack to return to a previous menu.

This class is also responsible for the parsing of the information, as we can see in the image.

**All** data-sets presented in the project description are available in this program.

```
-------------------------------------------
            Menu -> Algorithm

    Algorithms (Recommended for Small Graphs)

            1. Backtracking

    Algorithms (Recommended for Large Graphs)

        2. Triangular Approximation - Prim
        3. Other Heuristic (Cristofides)
        4. TSP in real World

         Extra Algorithms and Metrics

                5. Extra

              Go back option
                6. Go back
-------------------------------------------
Enter your choice (1-6):
```

Img 1. Main Menu

# Backtracking

This algorithm guarantees the optimal solution for our problem, but is only recommend to use in the toy graphs.  In our algorithm we also take into account, the coordinates of the vertices, to find new paths, if needed. For this special test, we created a folder for different graphs, where we testes with small non complete graphs.

When we parse the graph with real-mode in the toy graphs is not expected to give any solution since the graphs don’t have coordinates. For symmetric parsing all 3 gave the optimal result as intended.

```
carmo-> clerigos
clerigos-> se
se-> dLuis
dLuis-> bolsa
bolsa-> carmo
Distance: 2600
CPU Time: 0.000213392 Real Time: 0.000217576
```

Img 2. Graph tourism.csv output

# Triangular Approximation Heuristic

For exercise 4.2, we created a function that performs the following steps:

1. First, we call a function that generates an MST using Prim's algorithm, starting at a specified vertex;
    a. We also support incomplete graphs, but using the vertex's coordinates and the distance between them;
2. Then, we order the edges in the MST to create a way to traverse every single one without going back to a previously visited edge;
3. After that, we add an extra edge to connect the last edge to the starting edge, finishing the tour;

```
24-> 8
8-> 12
12-> 9
9-> 5
5-> 15
15-> 18
18-> 10
10-> 19
19-> 3
3-> 21
21-> 13
13-> 1
1-> 2
2-> 16
16-> 14
14-> 23
23-> 0
Distance: 348706
CPU Time: 0.000362423 Real Time: 0.000365494
```

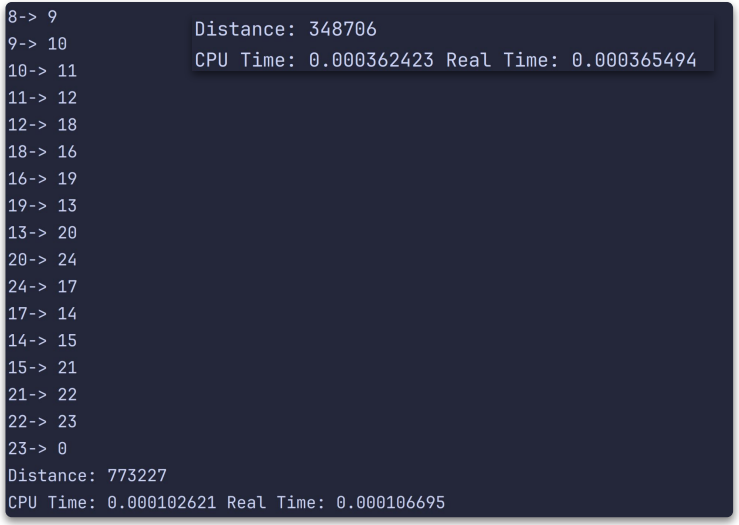
Img 3. Triangular Approximation output for 25 vertices.

# Other Heuristic - Greedy Heuristic

For exercise 4.3, we chose to do a greedy solution to the problem:

1. We first choose the starting vertex of the graph;
2. Then, we iterate through all it's adjacent Edges to find the closest one;
    a. If all adjacent edges have already been visited, we use a haversinedistance function to find the closest edge distance wise;
3. After choosing a new Vertex, we set the old one as visited and restart the process with the new Vertex;
4. We do this until all Vertices of the graph has been added to the result variable of the heuristic;

This algorithm, while not yielding the best possible answer to the problem at hand, is considerably faster than the Triangular Approximation Heuristic, specially for larger graphs.




Img 4. Greedy Heuristic Output Compared to Triangular Approximation for 25 vertices.

# TSP in the Real World

To resolve this point, we used the same greedy algorithm used before, but this time, we don’t use the distances between Vertices to generate the Tour.

We only use the Edges provided by the graph.

```
234-> 251
251-> 157
157-> 470
470-> 416
416-> 462
462-> 287
287-> 351
351-> 265
265-> 739
739-> 968
968-> 959
959-> 756
756-> 197
197-> 826
826-> 770
770-> 772
772-> 0
Distance: 1.00516e+06
CPU Time: 0.0415143 Real Time: 0.0415168
```

Img 5. Real world algorithm for real world graphs, graph 1.

# Conclusion

From this project, we learned that some problems really are mind boggling.

It was really challenging to implement all these algorithms, and even worse were those we had to scrap because we genuinely couldn't understand how to write them.

But in the end, we are happy with the solutions we wrote and hope we'll never have to encounter anything like this in the real world… hopefully.

For future developments, we might consider using other heuristics like Cristofide's, lin-kernighan's or even held-karp's by using dynamic programming.

**The End.**