Martina Carretta – 1673930 and Meritxell Carvajal – 1671647

PROJECT 2 REPORT

DIFFICULTIES OF THE PROJECT AND HOW WE SOLVED THEM

- **Finding the shortest path method.** Instead of looking for the shortest path in a heuristic way, we did it with the Dijkstra algorithm, which is what actually finds the path with the least weight; but it happened that this method only worked for us with some examples and we decided do it the heuristic way. What's wrong? We had trouble finding the shortest route because the code went into a loop. So to fix this, instead of doing it all in one function, we made the heuristic function call the function smallestNonZero() (it doesn't return the cost, it returns the city ID); then, what it receives is a city, and of the unvisited cities it seeks the lowest cost. The heuristic is then called recursively with that new city. → so fixed

- **Doing the partial road map and the total road map.** (important: two pointers: the head, which is the pointer for the total road map, and the path, which is a pointer that is reused for each partial road map. Then, when a partial road map ends, what it does is adding itself to the head and empty the path so that it is ready for the next one. We have another pointer for the partial road maps, which is the last one. We didn't have much trouble making the prints of the two road maps: the only thing that happened was that there were 3 cities that were printed in reverse, but we realized that it was a problem of how we were calling the function.

- **printPartialMap() →** When the tree is made, first the whole tree must be printed and then all the partial road maps. Problem: partial roadmaps are made and saved in the head and then deleted, so we can't save them all. So, instead of printing from the path, which is the pointer for the partial road maps, what we decided was to print from the head until the city we want to print is the same as the next one (when you see that the first of the next partial road map is equal to the last of the current partial road map)

- **DFS.** The best way we have found to do this has been with recursivity, where the function receives as an argument the ID of the city we are in. Firstc, a traversal of the list is done and a node is created to add it to the tree (the head of the tree is called HeadTree). The thing is that we create a familytree node and it receives the information. So, we have two variables, which perhaps we could have saved (not created) but it was the best way we saw how to do it and they are: id_forthemothercity (citym) and id_forthefathercity (cityf); we have used these variables to check if it is the last node, equating it to != -1 (it is the same if we match citym or cityf because they both end at the same time). The problem we had was to make the execution end when it was supposed to end.

- **printTree() →** We only use it to print the DFS. To print it we are traversing. It receives the node and the level of the node, and this is used for the arrows of the tree, which has more arrows the further down it is. We have also done this function with recursivity.

- **Array →** We use it because in the DFS when we build the tree what we do is keep the IDs of the cities we visit, not those of the path, but those of each node. What this does is that when we print the partial road maps, for each element of the array that is not a -1, it looks for the shortest path between array[i] and array[i+1].

- **BFS.** As we already have the DFS done, we don't need to make the tree again, but to print it in a different way. We have not made this function recursive.

TESTING

To test that the code works, we've tried with the small.h file and compared the results with the pdf provided. We haven't executed the code for the medium.h and large.h versions, since there is no matrix provided.

We will show you what the final output is with the small.h proposed file.

To test it, this time we are not executing through the terminal, we are just running the code. However, it can also be executed writing this in the terminal (without the need of any argument):

gcc Filename.c -o executablename

./ executablename

ONLY TRIAL: small.h

input

```
This is a small case of the program
Ancestors' tree:

DFS -> Names:
Maria and Jordi (Barcelona)
--> Louise and Paul (Paris)
--> --> Anna and Kazimierz (Varsovia)
--> --> Agnese and Leonardo (Rome)
--> Eva and Albert (Zurich)
--> --> Madalena and Lourenço (Lisbon)
--> --> Amber and Finn (Amsterdam)

Partial road map:
Barcelona - Madrid - Paris - 180
Paris - Berlin - Varsovia - 170
Varsovia - Berlin - Viena - Zurich - Rome - 230
Rome - Zurich - 70
Zurich - Viena - Berlin - Paris - Lisbon - 270
Lisbon - Barcelona - Madrid - Rome - Amsterdam - 340

Total road map:
Barcelona - Madrid - Paris - Berlin - Varsovia - Berlin - Viena - Zurich - Rome - Zurich - Viena - Berlin
 - Paris - Lisbon - Barcelona - Madrid - Rome - Amsterdam

Total cost: 1260
```

```
------------------------------------------------------------------------
BFS -> Names:
Maria and Jordi (Barcelona)
--> Louise and Paul (Paris)
--> Eva and Albert (Zurich)
--> --> Anna and Kazimierz (Varsovia)
--> --> Agnese and Leonardo (Rome)
--> --> Madalena and Lourenço (Lisbon)
--> --> Amber and Finn (Amsterdam)

Partial road map:
Barcelona - Madrid - Paris - 180
Paris - Berlin - Viena - Zurich - 150
Zurich - Viena - Varsovia - 140
Varsovia - Berlin - Viena - Zurich - Rome - 230
Rome - Zurich - Viena - Berlin - Paris - Lisbon - 340
Lisbon - Barcelona - Madrid - Rome - Amsterdam - 340

Total road map:
Barcelona - Madrid - Paris - Berlin - Viena - Zurich - Viena - Varsovia - Berlin - Viena - Zurich - Rome
- Zurich - Viena - Berlin - Paris - Lisbon - Barcelona - Madrid - Rome - Amsterdam

Total cost: 1380
```