NICOLAS Kevin – YAYA-OYE Ikhlass – DIA 1

ADSA Mini Problem Report

Summary

Step 1: Organize the tournament

1. **Propose a data structure to represent a Player and its Score**

To represent a Player, we will use a class. A player is initialized with a name that is a string with 5 letters randomly chosen. It is also initialized with a score equal to 0 and a list of all its scores, empty at the beginning.

The Player class has also a function that allows us to display it, a function that can update the score of a player and a function that can reset the score of a player to 0 and its score's list to an empty list.

2. **Propose a most optimized data structures for the tournament (called database in the following questions)**

For the data structure of the tournament, we chose to use a binary search tree in which the root left sub-tree contains the ten-last player and the root right sub-tree is an AVL tree.

For the nodes of the tree, we use a class and where the data of the node is a player. The left and right child are nodes too.

Also, we used a list of Players in our database because we use it to create and update our tree tournament.

3. **Present and argue about a method that randomize player score at each game (between 0 point to 12 points)**

We want to create a method that simulate the score that a player can have at each game. We use the Python's random library *randint()* and as suggested in the question, the player can have a score between 0 and 12 at each random game.

4. **Present and argue about a method to update Players score and the database**

At the end of each game, we store each score for each player in a list so we can access them to do the mean easily.

Thank to our method in the Player class *update_score,* we can update the score by adding the score of the just played game to the list of scores of the previous games. After that, we sum the scores of the list for each player and we do the mean of this list and this is the new score of the player.

When the Players' score is updated, we create a new tree with the new scores and by doing so, we are updating the database (same complexity).

5. Present and argue about a method to create random games based on the database

We initialized the score of each player at 0. We create a method that make 3 randomized games for each group of 10 players. In this method, we just update 3 times the score of each player, the score of each player being randomly chosen by our method called *random_score*.

6. Present and argue about a method to create games based on ranking

At the end of each game, we have to do the mean of the score of each player. We suppose that before each round of 3 random games based on the ranking, we use an in-order method to store the players in a list of lists. By doing, we have all the players ranked by their score and each list correspond to a "room" of 10 players. Then we have created the games based on ranking.

7. Present and argue about a method to drop the players and to play game until the last 10 players

Ten last players are placed on the left sub tree of the binary search tree and all the remainders are on the right sub tree. We added a drop method in our Node class : at the end of each 3 random games, we update the tree by deleting all the nodes on the left sub tree.

So, we added a drop method in our Node class. We create a method that initialize the players, a counter and that display the first tree of the tournament. After that, we do a loop where we rank the players before each round of 3 games, then we split them in rooms where they will do their games and after that we rank them by their score. We display the results in the tournament tree, we drop the 10 last for the previous round, and we display a tree with the players left in the tournament. We do this loop until we have our 10 finalists.

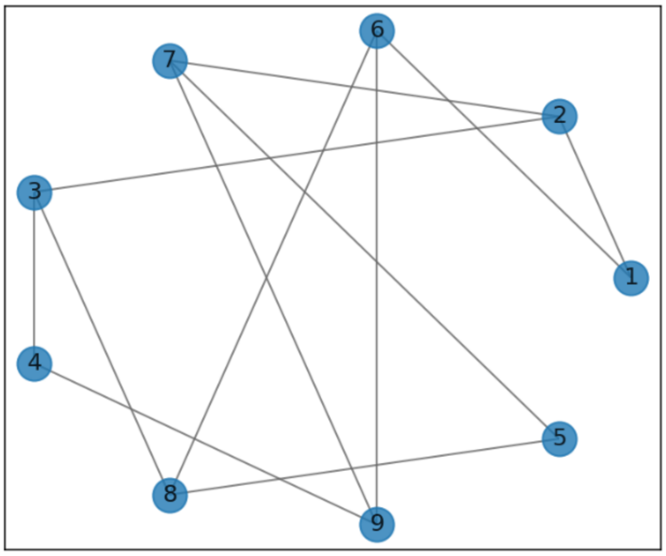
8. Present and argue about a method which display the TOP10 players and the podium after the final game

For the final game, we have to play 5 games and we need to reinitiate their ranking. We randomize the players score with our method, we update each player' score (and their ranking) and we start a new game. We do the same process until we have 5 games played. At each game, we do the mean of each player' score. At the end, we update the players' score, and we display the score, the name of each player, ranked with the player with the higher score at the 1st position and a podium.

Step 2: Professor Layton < Guybrush Threepwood < You

1. **Represent the relation (have seen) between players as a graph, argue about your model.**

The first strategy was to create a directed graph between the players in which an arrow from A to B means "A have seen B". However, according to the given information in the subject after the first kill, we can see that when A saw B, B saw A. So, an undirected graph should be enough. Here is the graph after removing 0.



2. **Thanks to a graph theory problem, present how to find a set of probable impostors.**

Like said previously, each edge between two vertices means that the players have seen each other. Here, our set of probably imposters are 1, 4 and 5 who have seen 0 before he died. And we want all possible pairs of imposters knowing the two imposters have not seen them before the emergency meeting.

A solution can be using a graph theory problem, graph labelling in this case. For this case, Graph coloring is an appropriate graph labelling problem. Indeed, each vertex connect to another should have a different color.

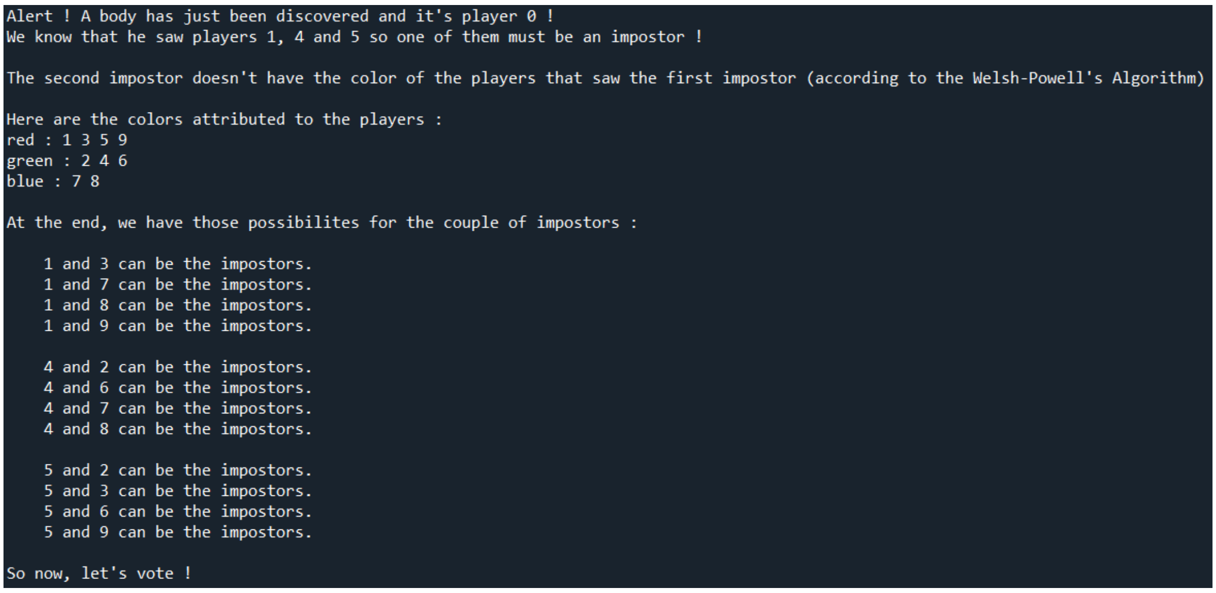
For our case, we suppose that one impostor is 1 or 4 or 5. So we should start with 3 colors and see if it is possible to color all nodes with 3 different color like no one edge links two vertices with the same color.)

3. Argue about an algorithm solving your problem.

For solve our problem, we choose to use Welsh Powell Algorithm. It is based on the greedy algorithm and coloring vertex based on the decreasing order of their degree. We first choose a color and go through the node which are in a list. If this node doesn't have a color and no one of its adjacent nodes have the current color, we assign the color to the node. If we cannot color the node, we continue and try to color the next node. After this, we try the same process with another color until all nodes are colored.

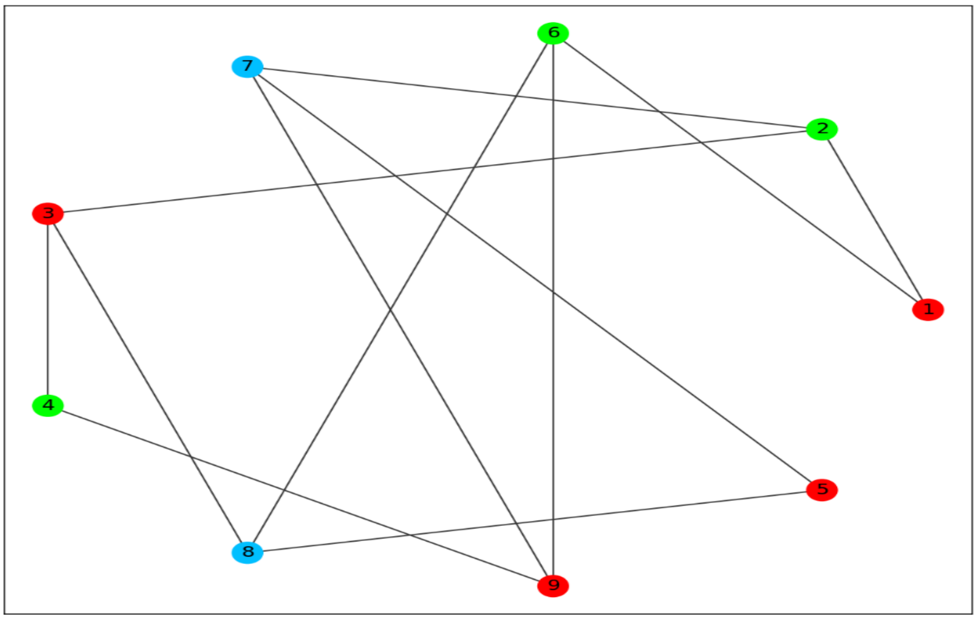
At the end, we obtain a colored graph. For each node, we can find a set of probable impostors: all nodes which have not the same color of the selected node's neighbors can be an impostor with this node.

4. Implement the algorithm and show a solution.



```
Alert ! A body has just been discovered and it's player 0 !
We know that he saw players 1, 4 and 5 so one of them must be an impostor !

The second impostor doesn't have the color of the players that saw the first impostor (according to the Welsh-Powell's Algorithm)

Here are the colors attributed to the players :
red : 1 3 5 9
green : 2 4 6
blue : 7 8

At the end, we have those possibilites for the couple of impostors :

    1 and 3 can be the impostors.
    1 and 7 can be the impostors.
    1 and 8 can be the impostors.
    1 and 9 can be the impostors.

    4 and 2 can be the impostors.
    4 and 6 can be the impostors.
    4 and 7 can be the impostors.
    4 and 8 can be the impostors.

    5 and 2 can be the impostors.
    5 and 3 can be the impostors.
    5 and 6 can be the impostors.
    5 and 9 can be the impostors.

So now, let's vote !
```

Colors of the players and the links that they have between them according to who they saw

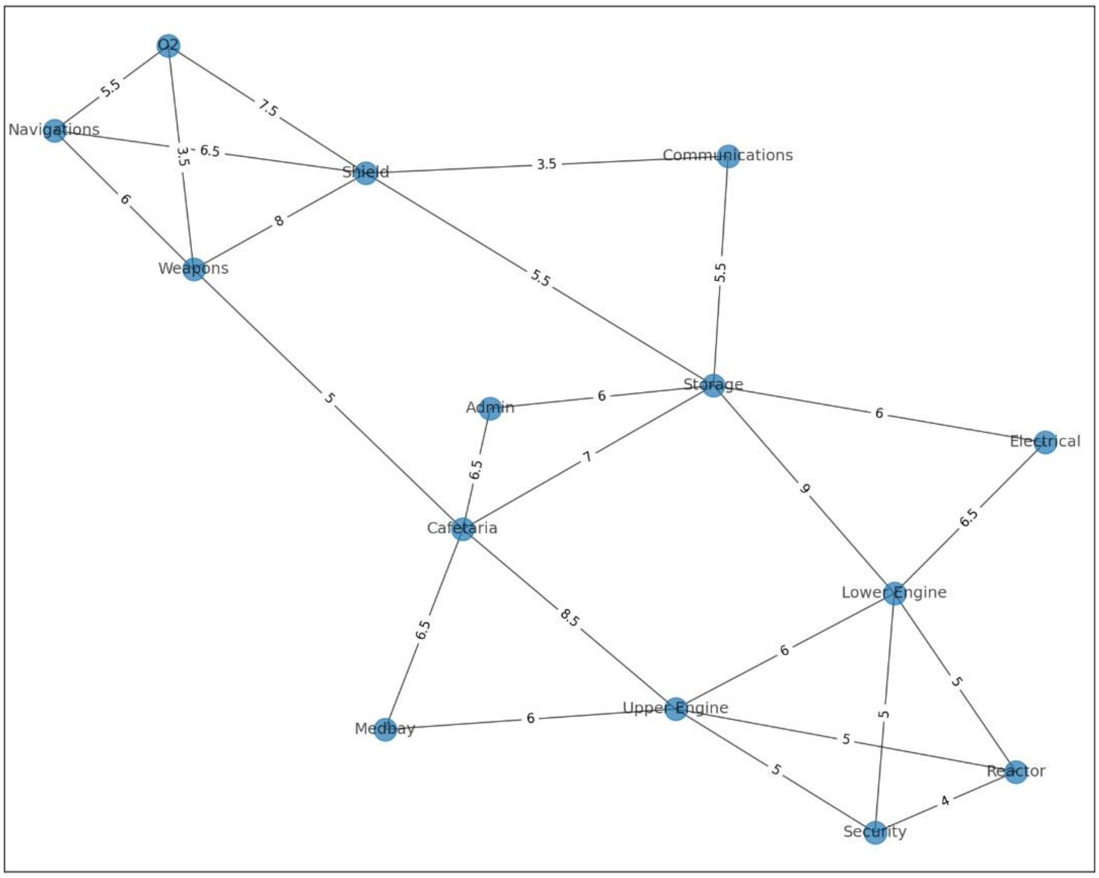


Step 3: I don't see him, but I can give proofs that he vents!

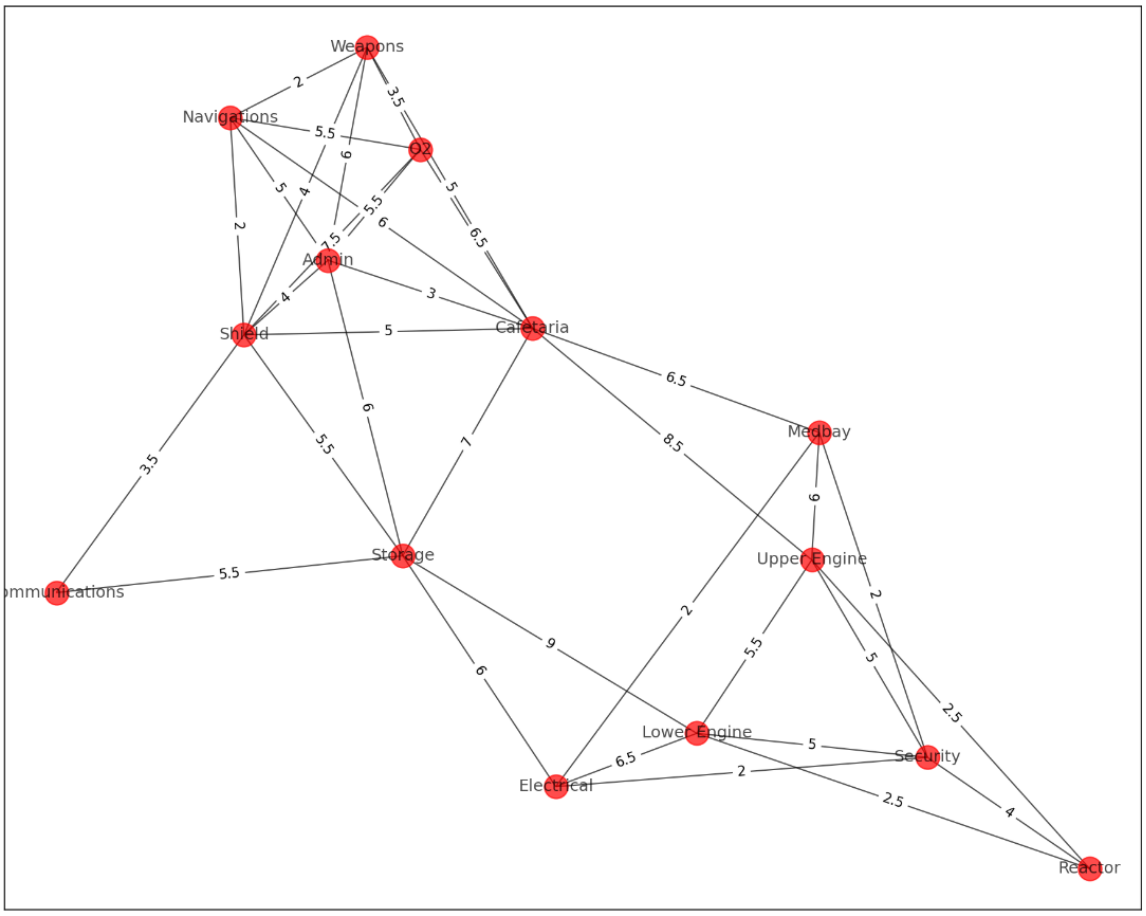
1. **Presents and argue about the two models of the map.**

Two models, one for the crewmates and one for the impostors. We measured from the centers of each room the distance between rooms. For the impostors, we started our measures from the center of the room but if there was a vent in this room that can rely him to another room, then we measure the distance between the center of his actual room to the vent and then the distance between the arrival vent to the center of the room where he wants to go.

<u>Crewmate Map</u>



Impostor Map



2. Argue about a pathfinding algorithm to implement.

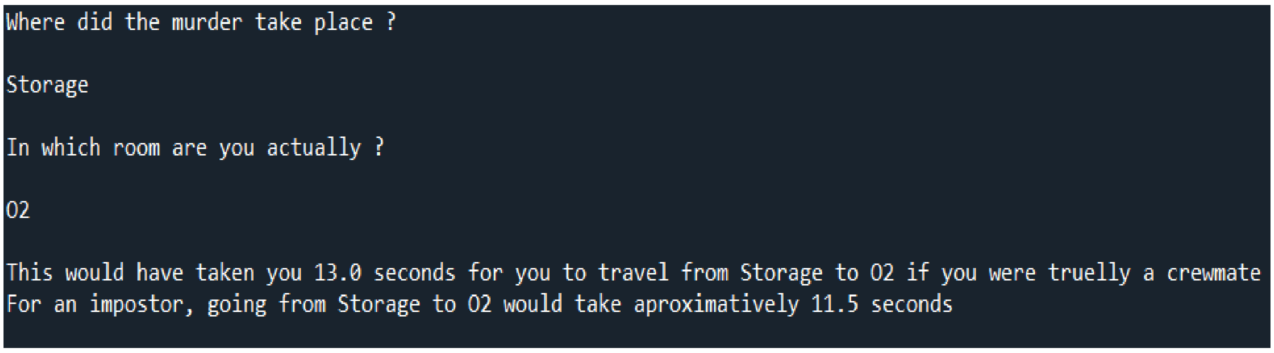
We have to find a way that permits us to have the shortest path from all rooms to all other rooms of the map. We compute the shortest path from a room A to all other and we do it for all room. For this, we use the Floyd-Warshall Algorithm.

The algorithm compares all possible path through the map between each pair of room and store in a matrix the shortest paths. The complexity of the algorithm is $O(V^3)$.

If *i* represents a room in the index and *j* a room in the columns, we use a variable *k* as an intermediate for select rooms in the {1, 2, …, k} subset. Then the shortest path for each pair is the minimum between the path from *i* to *k-1* and the path from *i* to *k* then from *k* to *j*.

3. Implement the method and show the time to travel for any pair of rooms for both models.

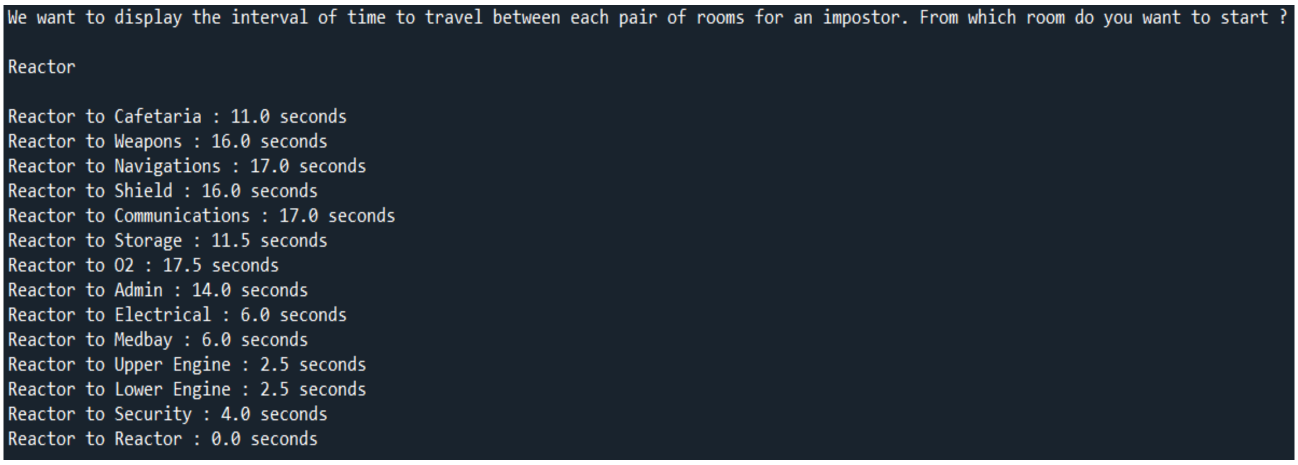
First, we ask where the dead body was found. After that, we ask to the player in which room he was when the dead body was reported. Then we want to display the time that the player would have taken to go from the room where the body was found to his actual room and we display it for the crewmate and the impostor cases.



```
Where did the murder take place ?

Storage

In which room are you actually ?

O2

This would have taken you 13.0 seconds for you to travel from Storage to O2 if you were truelly a crewmate
For an impostor, going from Storage to O2 would take aproximatively 11.5 seconds
```

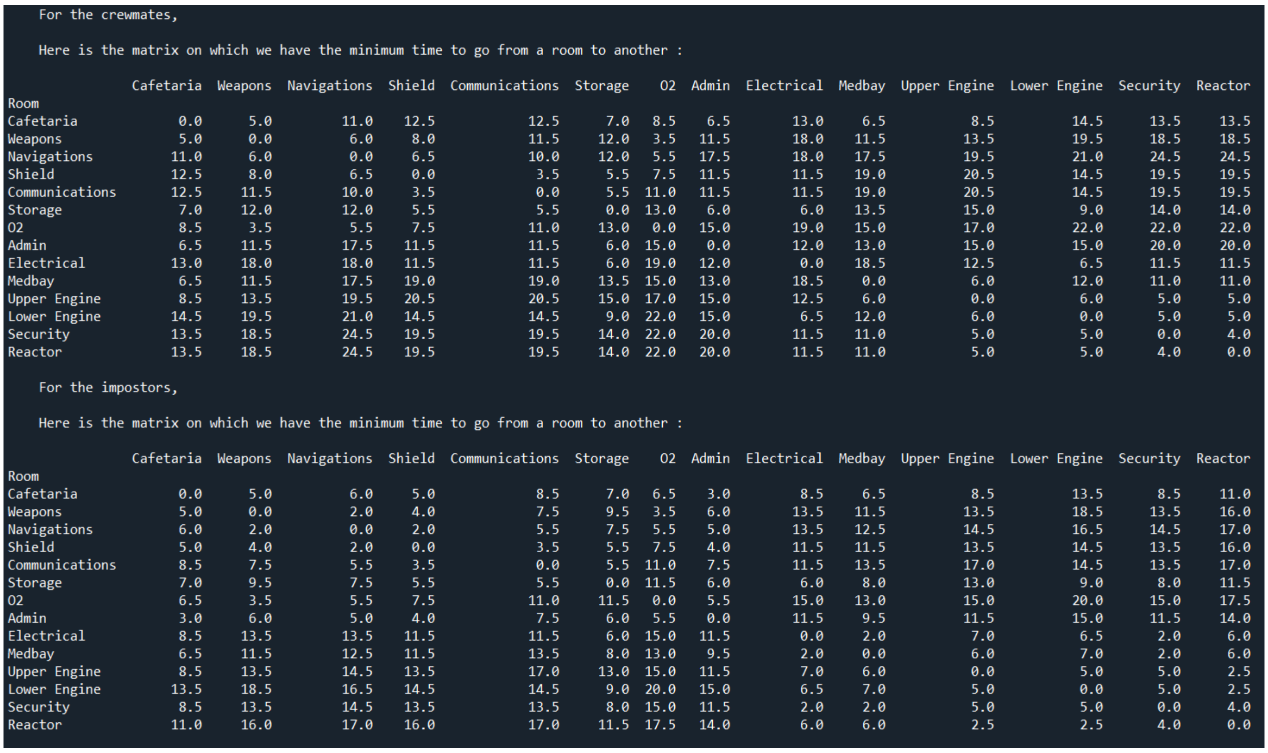
4. Display the interval of time for each pair of room where the traveler is an impostor.

For this task, we ask from which room do we want to start to calculate the time it would take to travel between the starting room and all the other rooms.



```
We want to display the interval of time to travel between each pair of rooms for an impostor. From which room do you want to start ?

Reactor

Reactor to Cafetaria : 11.0 seconds
Reactor to Weapons : 16.0 seconds
Reactor to Navigations : 17.0 seconds
Reactor to Shield : 16.0 seconds
Reactor to Communications : 17.0 seconds
Reactor to Storage : 11.5 seconds
Reactor to O2 : 17.5 seconds
Reactor to Admin : 14.0 seconds
Reactor to Electrical : 6.0 seconds
Reactor to Medbay : 6.0 seconds
Reactor to Upper Engine : 2.5 seconds
Reactor to Lower Engine : 2.5 seconds
Reactor to Security : 4.0 seconds
Reactor to Reactor : 0.0 seconds
```

In addition, we wanted to display the 2 matrices, for crewmates and impostors, that show the time travelling between 2 rooms:



```
    For the crewmates,

    Here is the matrix on which we have the minimum time to go from a room to another :

                Cafetaria  Weapons  Navigations  Shield  Communications  Storage    O2  Admin  Electrical  Medbay  Upper Engine  Lower Engine  Security  Reactor
Room
Cafetaria             0.0      5.0         11.0    12.5            12.5      7.0   8.5    6.5        13.0     6.5           8.5          14.5      13.5     13.5
Weapons               5.0      0.0          6.0     8.0            11.5     12.0   3.5   11.5        18.0    11.5          13.5          19.5      18.5     18.5
Navigations          11.0      6.0          0.0     6.5            10.0     12.0   5.5   17.5        18.0    17.5          19.5          21.0      24.5     24.5
Shield               12.5      8.0          6.5     0.0             3.5      5.5   7.5   11.5        11.5    19.0          20.5          14.5      19.5     19.5
Communications       12.5     11.5         10.0     3.5             0.0      5.5  11.0   11.5        11.5    19.0          20.5          14.5      19.5     19.5
Storage               7.0     12.0         12.0     5.5             5.5      0.0  13.0    6.0         6.0    13.5          15.0           9.0      14.0     14.0
O2                    8.5      3.5          5.5     7.5            11.0     13.0   0.0   15.0        19.0    15.0          17.0          22.0      22.0     22.0
Admin                 6.5     11.5         17.5    11.5            11.5      6.0  15.0    0.0        12.0    13.0          15.0          15.0      20.0     20.0
Electrical           13.0     18.0         18.0    11.5            11.5      6.0  19.0   12.0         0.0    18.5          12.5           6.5      11.5     11.5
Medbay                6.5     11.5         17.5    19.0            19.0     13.5  15.0   13.0        18.5     0.0           6.0          12.0      11.0     11.0
Upper Engine          8.5     13.5         19.5    20.5            20.5     15.0  17.0   15.0        12.5     6.0           0.0           6.0       5.0      5.0
Lower Engine         14.5     19.5         21.0    14.5            14.5      9.0  22.0   15.0         6.5    12.0           6.0           0.0       5.0      5.0
Security             13.5     18.5         24.5    19.5            19.5     14.0  22.0   20.0        11.5    11.0           5.0           5.0       0.0      4.0
Reactor              13.5     18.5         24.5    19.5            19.5     14.0  22.0   20.0        11.5    11.0           5.0           5.0       4.0      0.0

    For the impostors,

    Here is the matrix on which we have the minimum time to go from a room to another :

                Cafetaria  Weapons  Navigations  Shield  Communications  Storage    O2  Admin  Electrical  Medbay  Upper Engine  Lower Engine  Security  Reactor
Room
Cafetaria             0.0      5.0          6.0     5.0             8.5      7.0   6.5    3.0         8.5     6.5           8.5          13.5       8.5     11.0
Weapons               5.0      0.0          2.0     4.0             7.5      9.5   3.5    6.0        13.5    11.5          13.5          18.5      13.5     16.0
Navigations           6.0      2.0          0.0     2.0             5.5      7.5   5.5    5.0        13.5    12.5          14.5          16.5      14.5     17.0
Shield                5.0      4.0          2.0     0.0             3.5      5.5   7.5    4.0        11.5    11.5          13.5          14.5      13.5     16.0
Communications        8.5      7.5          5.5     3.5             0.0      5.5  11.0    7.5        11.5    13.5          17.0          14.5      13.5     17.0
Storage               7.0      9.5          7.5     5.5             5.5      0.0  11.5    6.0         6.0     8.0          13.0           9.0       8.0     11.5
O2                    6.5      3.5          5.5     7.5            11.0     11.5   0.0    5.5        15.0    13.0          15.0          20.0      15.0     17.5
Admin                 3.0      6.0          5.0     4.0             7.5      6.0   5.5    0.0        11.5     9.5          11.5          15.0      11.5     14.0
Electrical            8.5     13.5         13.5    11.5            11.5      6.0  15.0   11.5         0.0     2.0           7.0           6.5       2.0      6.0
Medbay                6.5     11.5         12.5    11.5            13.5      8.0  13.0    9.5         2.0     0.0           6.0           7.0       2.0      6.0
Upper Engine          8.5     13.5         14.5    13.5            17.0     13.0  15.0   11.5         7.0     6.0           0.0           5.0       5.0      2.5
Lower Engine         13.5     18.5         16.5    14.5            14.5      9.0  20.0   15.0         6.5     7.0           5.0           0.0       5.0      2.5
Security              8.5     13.5         14.5    13.5            13.5      8.0  15.0   11.5         2.0     2.0           5.0           5.0       0.0      4.0
Reactor              11.0     16.0         17.0    16.0            17.0     11.5  17.5   14.0         6.0     6.0           2.5           2.5       4.0      0.0
```

Step 4: Secure the last tasks

1. **Presents and argue about the model of the map.**

The model of the map will be the same that we use in Step 3. We use the crewmate version of the map because we want to pack the players and the aim is to act like crewmates to counter the final impostor.

2. **Thanks to a graph theory problem, present how to find a route passing through each room only one time.**

Our objective in this step is to go through each room of the map in the quickest way for finish latest missions. As the map is assimilated to a graph, we must print a path in which a vertex is not used more than once. We overview multiple algorithms to choose a correct one for our problem. We selected Hamilton's path that is a path in our undirected weighted graph that visits each vertex exactly once. In our case, we don't need to do a Hamilton cycle because when the players have finished their tasks, they don't need to go back to the beginning room to end the game. This means that the first room and the last room don't have to be adjacent.

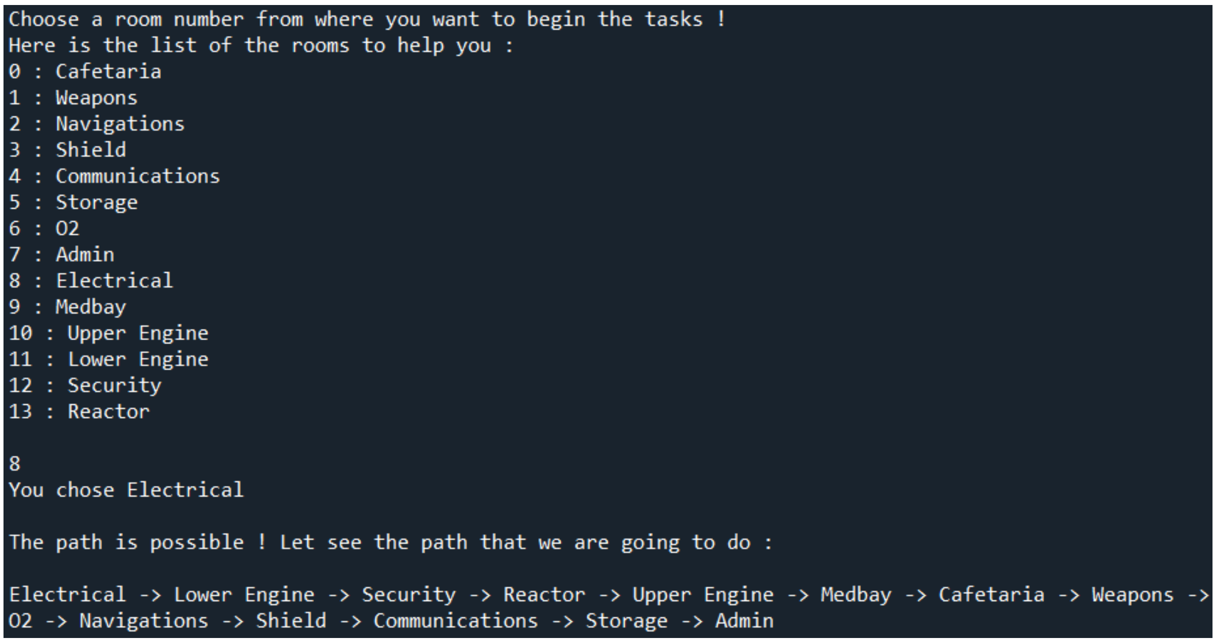
3. **Argue about an algorithm solving your problem.**

Our algorithm for solving the problem is a recursive function. We choose a vertex as starting point. The base condition is if the path length is equal to the number of vertices in the graph (here we have 14 rooms).

If not, we order the neighbors by the weight of the edge which linked them to the current vertex. Then for each of them we check it is not already in the path and if not, we add it in the path and apply the function on it: the added vertex become the new current vertex.

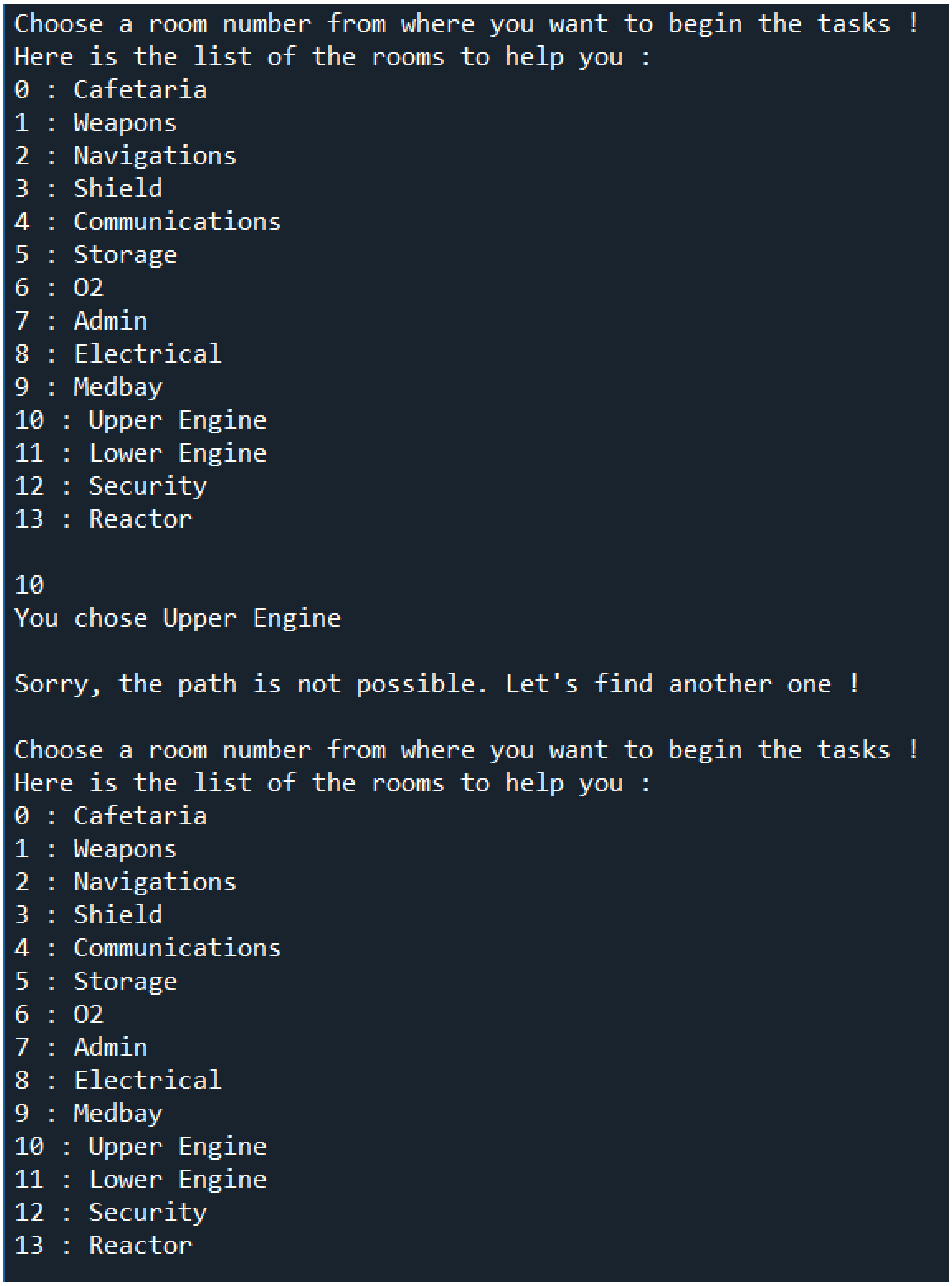
4. **Implement the algorithm and show a solution.**

This is what our code display if the path is possible :



```
Choose a room number from where you want to begin the tasks !
Here is the list of the rooms to help you :
0 : Cafetaria
1 : Weapons
2 : Navigations
3 : Shield
4 : Communications
5 : Storage
6 : O2
7 : Admin
8 : Electrical
9 : Medbay
10 : Upper Engine
11 : Lower Engine
12 : Security
13 : Reactor

8
You chose Electrical

The path is possible ! Let see the path that we are going to do :

Electrical -> Lower Engine -> Security -> Reactor -> Upper Engine -> Medbay -> Cafetaria -> Weapons ->
O2 -> Navigations -> Shield -> Communications -> Storage -> Admin
```

If a path isn't possible, the code display that the path wanted is not possible and ask another room from where to start until we have a path that is possible :



```
Choose a room number from where you want to begin the tasks !
Here is the list of the rooms to help you :
0 : Cafetaria
1 : Weapons
2 : Navigations
3 : Shield
4 : Communications
5 : Storage
6 : O2
7 : Admin
8 : Electrical
9 : Medbay
10 : Upper Engine
11 : Lower Engine
12 : Security
13 : Reactor

10
You chose Upper Engine

Sorry, the path is not possible. Let's find another one !

Choose a room number from where you want to begin the tasks !
Here is the list of the rooms to help you :
0 : Cafetaria
1 : Weapons
2 : Navigations
3 : Shield
4 : Communications
5 : Storage
6 : O2
7 : Admin
8 : Electrical
9 : Medbay
10 : Upper Engine
11 : Lower Engine
12 : Security
13 : Reactor
```