DIA1 - LEBLANC César & ADJARIAN Stéphan

Advanced Data Structure and Algorithms Mini-Problem

Everything has been programmed in Google Collab with Python, hence we have submitted a .ipynb which has a nice display and easy package installation. Here is the link :
https://colab.research.google.com/drive/10IAB2f-7Tasd2Nr9v_1M3ZQFk0kLFZC9?usp=sharing
You can make a copy of the notebook to execute it easily or import the .ipynb in Collab.

Step 1: To organize the tournament

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

We propose to store player's data in a dictionary to have access to the element in $O(1)$. The dictionary is composed of :

- the player's name : a *string* (player_1, player_2, …, player_100)
- his score : *an array* containing all his games' score and the mean of all its games (the first value is the average score and the next values are the scores of the player in each game so if a player has played 2 games in which he scored 6 points and then 10 points, his score will be [8, 6, 10])
- whether he is an impostor or not : *a Boolean* (True stands for Impostor, False stands for Crewmate)

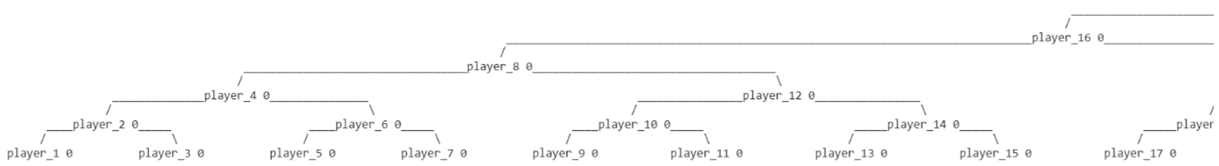
2. Propose a most optimized data structures for the tournament (called database in the following questions)

The most optimized data structure for the database seems to be an AVL tree. Each node is the player's dictionary and the AVL tree is sorted using the player's average score. AVL tree allows us to search, access, insert and delete elements in $O(log\ n)$. Moreover, the data are stored sorted.

The height h (counted as number of edges on the longest path) of an AVL tree with n nodes lies in the interval :

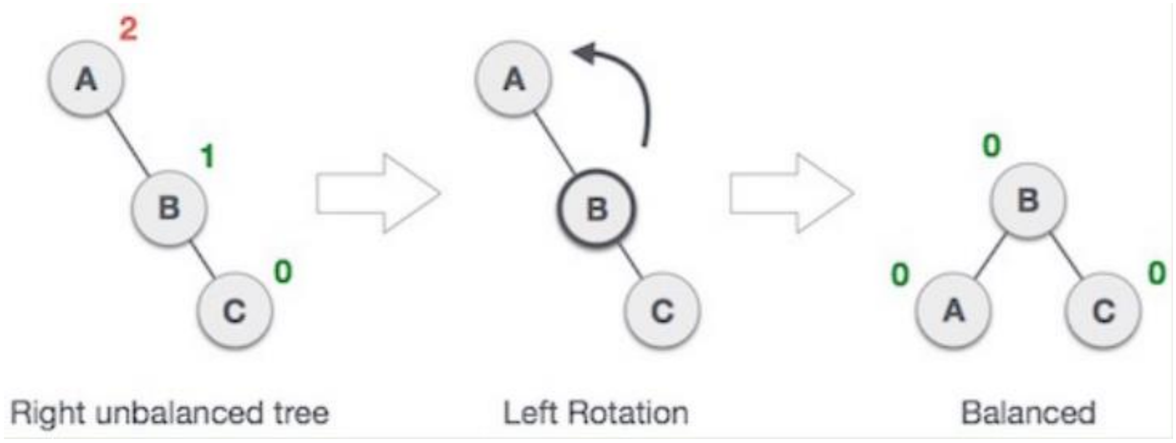
$$log_2(n+1) - 1 \ \leq \ h < \ log_{\phi}(n+2) + b \ \ where \ \ b \approx \ -1.3277$$

In our case we have 100 nodes therefore our AVL tree's height will remain between 6 and 8. The tree is too big to be correctly displayed, but here is for example a small part of the left subtree :

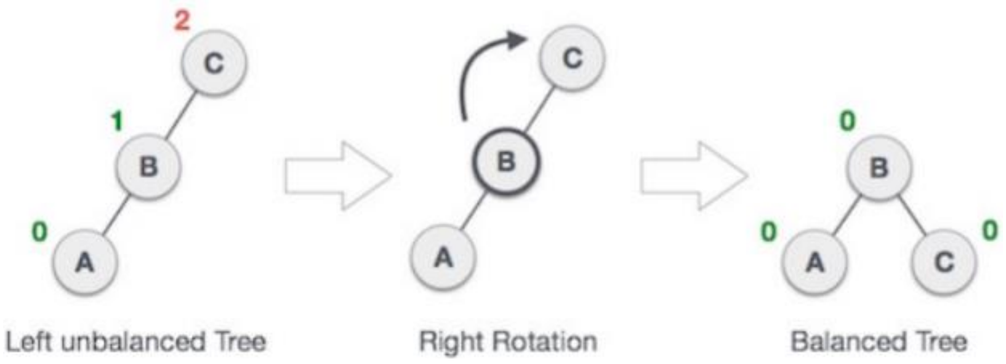


```
                                                                                                                                                                 ______________________
                                                                                                                                                                /
                                                                            _______________________________________________________________________________player_16 0_________________
                                                                           /
                                    __________________________________player_8 0_____________________________________
                                   /                                                                                 \
                ______________player_4 0_______________                                          ________________player_12 0________________
               /                                       \                                        /                                           \                                           /
      ____player_2 0_____                     ____player_6 0_____                     ____player_10 0_____                     _____player_14 0_____                       _____player
     /                   \                   /                   \                   /                    \                   /                     \                     /
player_1 0          player_3 0          player_5 0          player_7 0          player_9 0           player_11 0          player_13 0           player_15 0          player_17 0
```

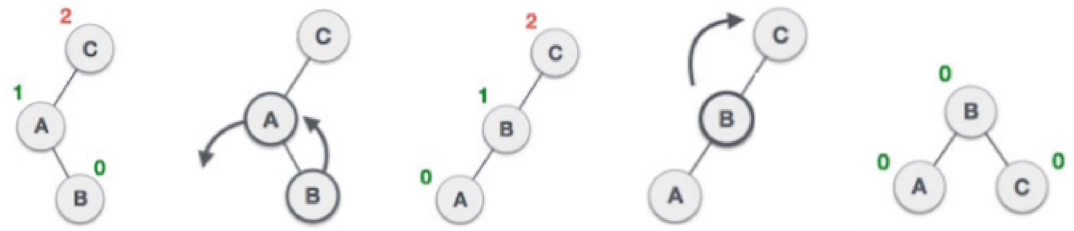
Using an AVL Tree comes with certain obligations. Each time we will insert or delete a player in the tournament, we will have to make sure that our tree is still balanced. if the difference in height between its left sub-tree and right sub-tree is not between -1 and +1, we will have to balance the tree again by making rotation. If the tree is unbalanced because of a player that has a better score than a second player that himself have a better score than a third player, we have to perform a single left rotation :



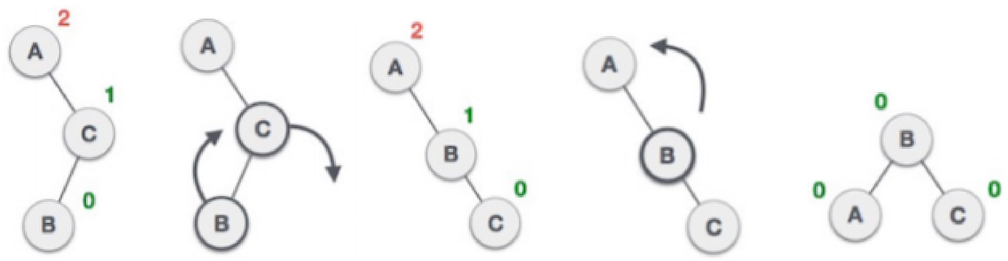
If the tree is unbalanced because of a player that has a worst score than a second player that himself have a worst score than a third player, we have to perform a single right rotation :



If the tree is unbalanced because of a player that has a better score than a second player that himself have a worst score than a third player, we have to perform a left right rotation :



And last but not least, if the tree is unbalanced because of a player that has a worst score than a second player that himself have a better score than a third player, we have to perform a right left rotation :



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

For all player we append a new random number between 0 and 12 to the score list and then we update the mean of all games which is the first element of the list. It is done easily with the library random available in Python.

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

Using Tree Traversal algorithm elements can be retrieved in-order by recursively traversing the left subtree of the root node accessing the node itself then recursively traversing the right subtree of the node continuing this pattern with each node in the tree as it is recursively accessed. Traversal is performed in $O(n)$.
Hence our method consists in creating another AVL tree and applying the traversal algorithm to our database. For each node we copy the node and append to the player's array a randomize int between 0 to 12 and we compute and store the mean of all his games in the first element of this array (initially set to 0). Then we insert this node to the new AVL tree. We chose to create a new AVL tree to avoid passing twice on the same node. The whole process is then done in $O(n * log\ n)$.

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

We propose to create 10 empty AVL trees numbered from 1 to 10. Then we use Tree Traversal algorithm. For each node we pick a random int in a dictionary whose keys are the numbers corresponding to each AVL tree and the value is the number of player already added to the tree we increment the value corresponding the AVL tree in the dictionary. When an element reaches 10 we remove it from the dictionary it means the tree is full (10 players max in a game). Once trees are completely filled we pick 2 distinct numbers in {1 … 10} and we attribute to the corresponding player the role of impostor by using search algorithm in $O(log\ n)$. Hence the whole process is also done in O(n*log n).

Here is an example of three games among ten, and we can see that the repartition of players is heterogeneous (players inside one game have very different scores) :

```
                    __________________player_88 5.0_______________________________________________
                   /                                                                              \
      ______player_3 1.0______                                                  ____________________player_43 12.0______
     /                        \                                                /                                        \
player_38 0.0            player_82 1.0                           ______player_94 9.0______                       player_51 12.0______
                                                                /                         \                                          \
                                                      player_39 8.0              player_86 11.0                                player_85 12.0
                    ___________________player_54 8.0________________________________________________
                   /                                                                                \
      ______player_70 3.0______                                                   ____________________player_72 11.0______
     /                         \                                                 /                                        \
player_67 3.0             player_23 5.0                           ______player_20 11.0______                       player_19 12.0______
                                                                 /                          \                                          \
                                                       player_92 8.0               player_40 11.0                                player_25 12.0
                    ___________________player_59 5.0____________________________________________
                   /                                                                            \
      ______player_15 3.0______                                                 __________________player_21 9.0______
     /                         \                                               /                                     \
player_64 2.0             player_30 5.0                          ______player_41 7.0______                     player_31 9.0______
                                                                /                         \                                      \
                                                       player_18 6.0               player_1 9.0                            player_75 11.0
```

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

We create an empty AVL tree and a global counting variable. Then once again with Tree Traversal algorithm we add each node to the new tree and add 1 to the count variable. Once it reaches 10 we create another tree and do the same until with finish the traversal.

Here is an example of three games among ten, and we can see that the repartition of players is now homogenous (players inside one game have more or less the same score) :

```
                    ___________________player_3 1.5_______________________________________________
                   /                                                                              \
      _____player_64 1.0______                                                     ___________________player_27 2.0______
     /                        \                                                   /                                      \
player_6 1.0             player_98 1.0                            ______player_42 1.5______                        player_70 2.0______
                                                                 /                         \                                         \
                                                       player_84 1.5              player_56 2.0                              player_15 2.5
                    ___________________player_59 2.5_______________________________________________
                   /                                                                               \
      ______player_89 2.5______                                                    ___________________player_30 3.0______
     /                         \                                                  /                                      \
player_52 2.5             player_23 2.5                             ______player_38 3.0______                      player_12 3.5______
                                                                   /                         \                                       \
                                                         player_81 2.5              player_24 3.0                              player_65 3.5
                    ___________________player_96 4.5_____________________________________________
                   /                                                                             \
      ______player_29 4.0______                                                   ___________________player_91 4.5______
     /                         \                                                 /                                      \
player_37 4.0             player_10 4.5                            ______player_61 4.5______                      player_21 4.5______
                                                                  /                         \                                      \
                                                         player_5 4.5               player_62 4.5                            player_53 5.0
```

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

At the end of each game we update all scores. Then we delete 10 times the leftmost element using the simple deletion algorithm of AVL trees. Once it is done we add 10 to a global counting variable and when it reaches 90 we stop the process.

Here is an example of a final possible tree :

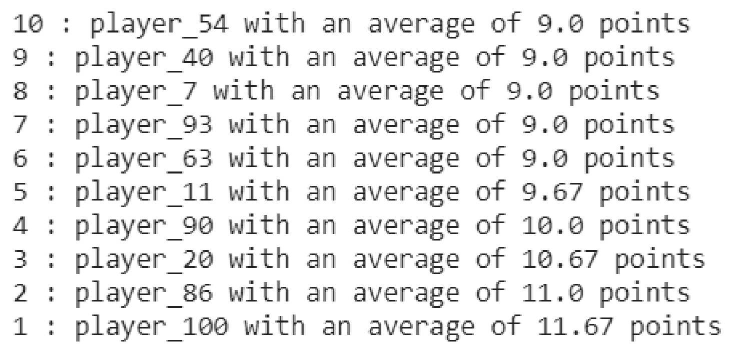
```
                                 _________________player_63 9.0____________________
                                /                                                  \
          __________________player_7 9.0______                              ______player_90 10.0_____________________
         /                                    \                            /                                         \
player_54 9.0______                      player_93 9.0              player_11 9.67                        ______player_86 11.0_______
                   \                                                                                     /                          \
              player_40 9.0                                                                     player_20 10.67            player_100 11.67
```

(please note that the fact that player_100 is the one with the best average score is a pure coincidence !)

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

Using Tree Traversal algorithm on the remaining AVL tree we display players in increasing order.

Here is an example of a possible outcome :

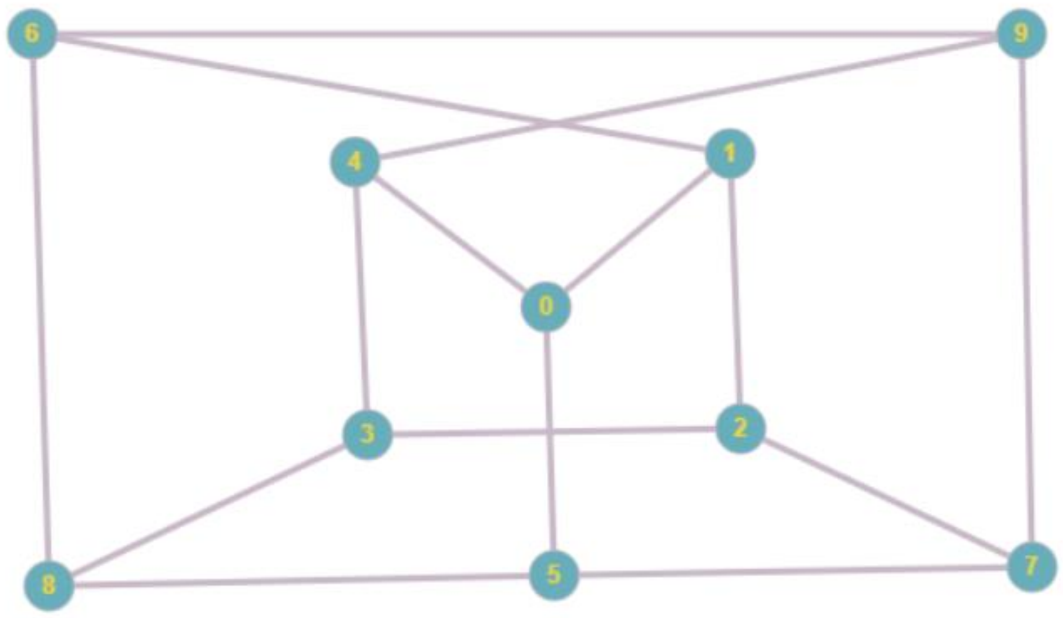


```
10 : player_54 with an average of 9.0 points
9 : player_40 with an average of 9.0 points
8 : player_7 with an average of 9.0 points
7 : player_93 with an average of 9.0 points
6 : player_63 with an average of 9.0 points
5 : player_11 with an average of 9.67 points
4 : player_90 with an average of 10.0 points
3 : player_20 with an average of 10.67 points
2 : player_86 with an average of 11.0 points
1 : player_100 with an average of 11.67 points
```

Step 2: Professor Layton < Guybrush Threepwood < You

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

We chose to use an adjacency matrix, 1 correspond to have seen and 0 have not seen. Here is the model :



According to us, this model is a best because it's very simple to use and we can use a lot of different algorithms with graphs, but also because graphs are very easy to understand, even for people who don't know anything about mathematics.

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

We could use the graph coloring in order to solve this problem. However, graph coloring uses the adjacency matrix. So in order to have a better and faster complexity, we will directly use the adjacency matrix to find a set of probable impostors.

3. Argue about an algorithm solving your problem.

Step 1 : With the information about who the players have seen, create the adjacency matrix.

Step 2 : Select the players that have seen the dead player. We can double check by looking at the adjacency matrix.

Step 3 : For each of these players, look at their column in the adjacency matrix. When you come cross a 0 (meaning that the 2 players haven't seen each other) that doesn't belong to a player that have also seen the dead player, it means that this player is a probable impostor. You can add the tuple of the original player and this one to the set of probable impostors.

We chose this method because the complexity is O(N x M) with N the number of players that have seen the dead player and M the total number of players alive minus N.

4. Implement the algorithm and show a solution.

Step 1 :

$$\begin{pmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0
\end{pmatrix}$$

Step 2 :

The players that have seen 0 are 1, 4 and 5.

We can double check and see that in the first column of the adjacency matrix (for player 0), there are a 1 on lines 2, 5 and 6 (for players 1, 4 and 5).

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Step 3 :

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

The set of probable impostors is :

{ 1, 3}, {1, 7}, {1, 8}, {1, 9}, {4, 2}, {4, 6}, {4, 7}, {4, 8}, {5, 2}, {5, 3}, {5, 6}, {5, 9}}

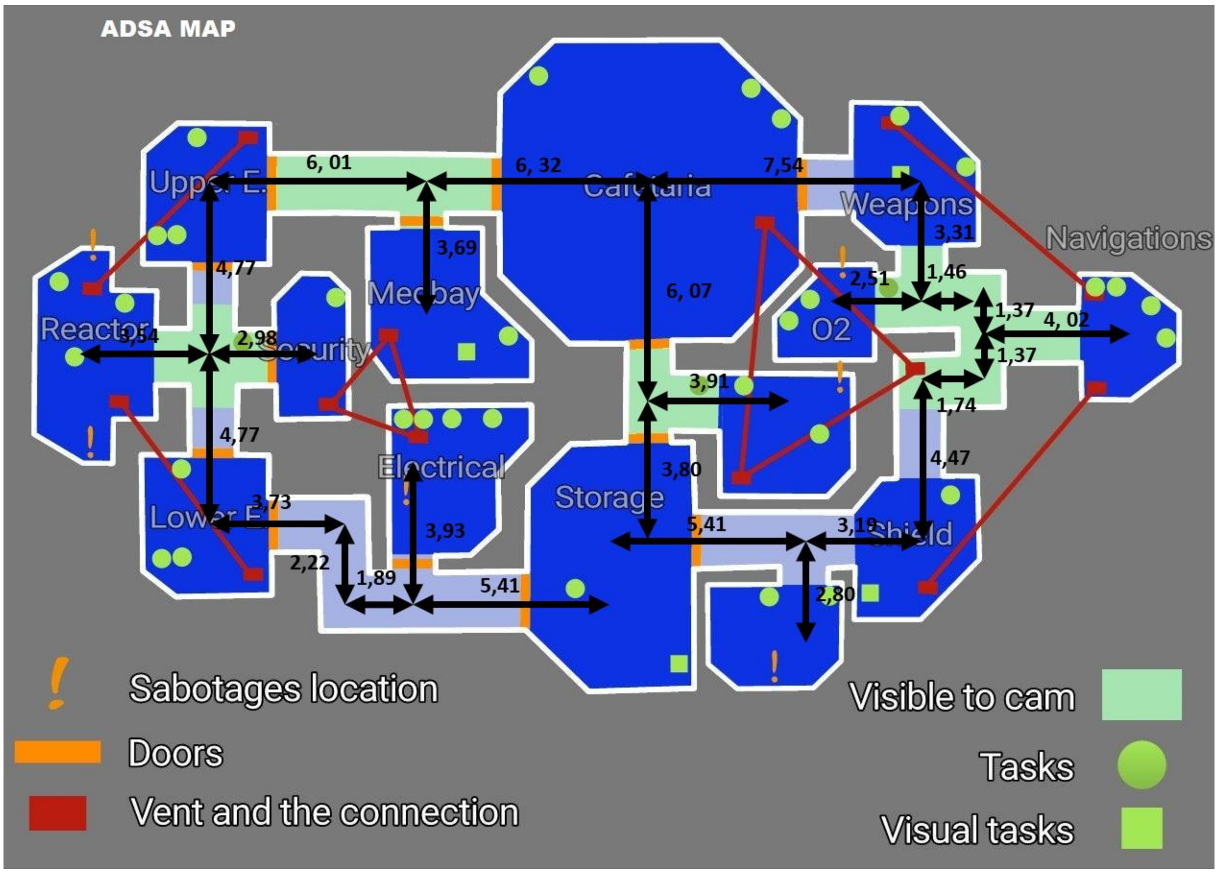
In our case, the complexity was O(N x M) with N = 3 and M = 6.

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

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

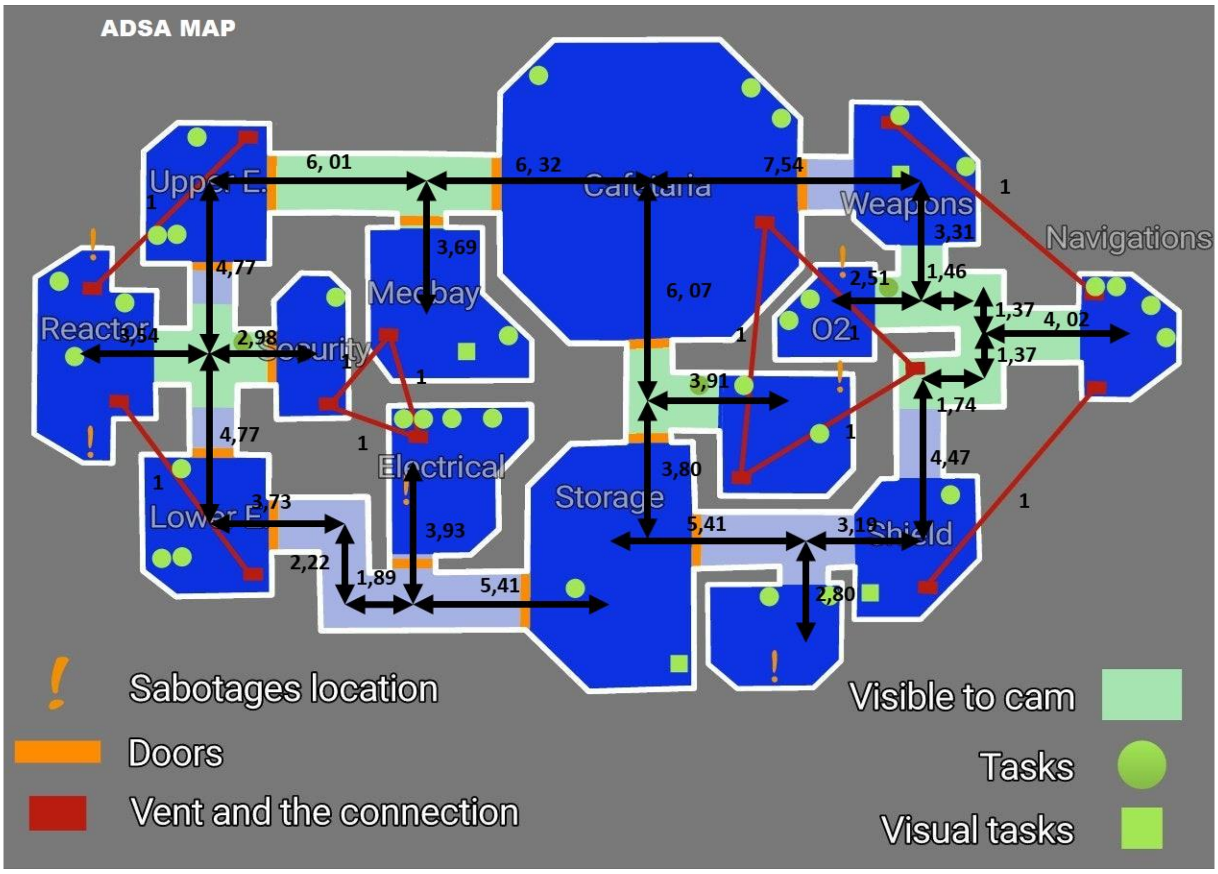
We have two maps : one for crewmates and one for impostors (the same than the previous one but with vents). We took the picture of the Among Us map and we decided to take 1cm = 1sec. Moreover, we decided to say that taking a vent would not be instantaneous but would take 1sec (which is still obviously a lot faster than walking).

Here is the map that crewmates can use :



We can see that for example to go from Upper E. to Reactor a crewmate has to walk 4.77 cm + 3.54cm = 8.31cm. It will the player 8.31sec to go from Upper E. to Reactor.

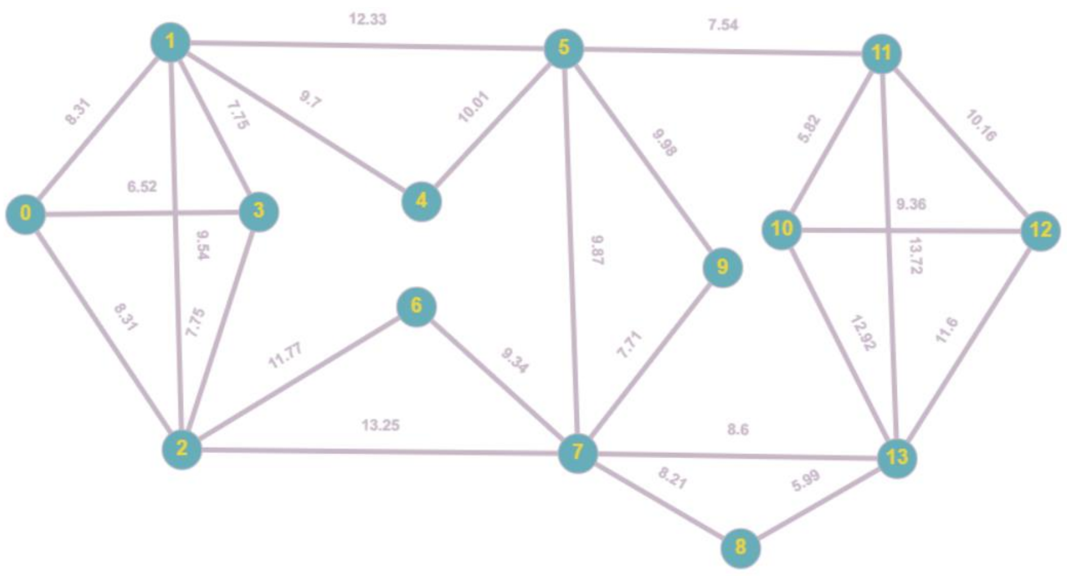
Now let's see the map for impostors :



Now if we take the same example than for crewmates, we can see that to go from Upper E. to Reactor, an impostor has 2 choices : he can either pretend to be a crewmate and walk between the two rooms (that will take him 8.31sec, same as before) or he can take the vent (that will take him only 1sec).

2. Argue about a pathfinding algorithm to implement.

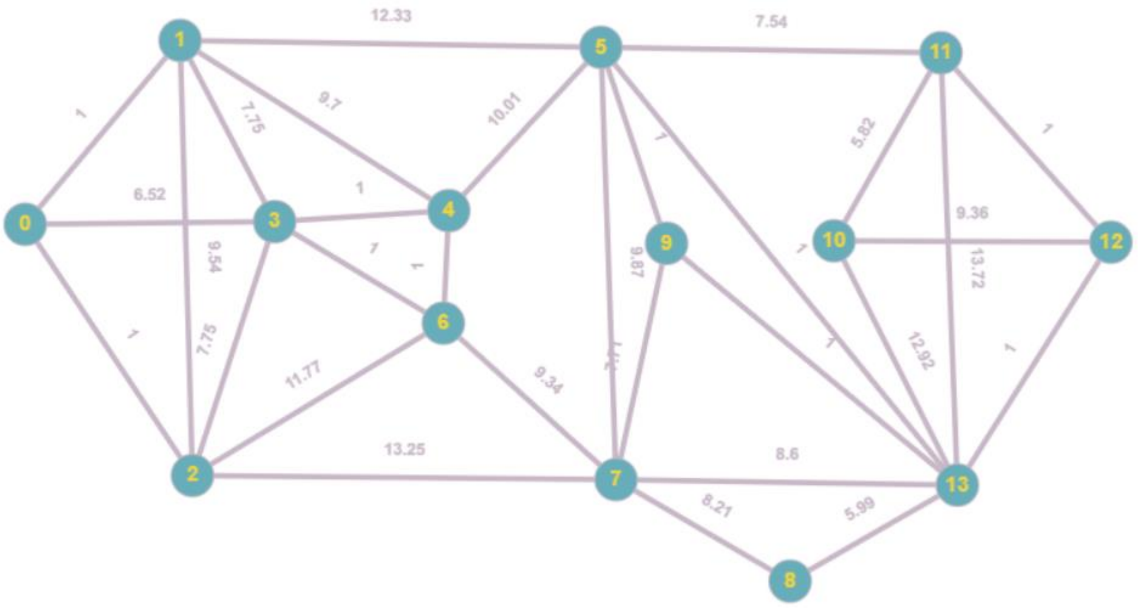
The goal to compare the time to travel between any pair of rooms, so we will use the Floyd-Warshall Algorithm because it computes the shortest distances between every pair of vertices in the input graph. If we write V the number of Vertices and E the number of Edges, the space complexity of this algorithm is $O(V^2)$ and time complexity of this algorithm is $O(V^3)$. We will have to create two graphs : one for the crewmates and their time to travel between each room and one for the impostors with the minimal time to travel between each room (so if they can walk and use a vent to go from one room to another, we will only include the vent in the graph, because otherwise if they decide to walk then we can use the crewmate graph). So here is the crewmates' graph :



It is obviously unoriented because if you can walk from room A to room B you can also walk from room B to room A.

For the impostor graph, we decided to do something a little different. Because it was specified in the exercise that “we don’t have to be precise”, we decided to put the vent in the Shield room (which is the room 13 in our graph). This way, impostors can directly go from Cafeteria to Shield instead of walking in the corridor for example.

So here is the impostors’ graph :



It is also an unoriented graph because impostors can walk and vent in any direction they want.

Here is how the algorithm of Floyd-Warshall works :

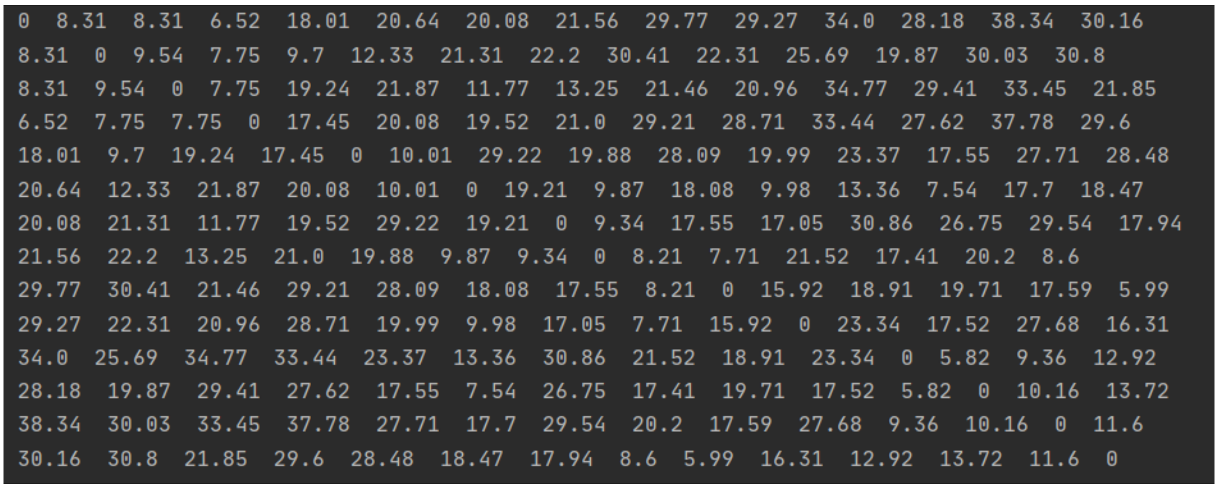
Step 1 : We create a matrix of dimension 14 by 14 (14 being the number of rooms in the map).

Step 2 : We iterate through each cell of the matrix. If the cell is in the diagonal, then we put a 0 here (because the distance to go from a room to the same room will obviously always be 0). If the cell is an edge (meaning that there is a path directly leading from the first room to the second), then we put the weight of the edge in this cell. Otherwise, we put 999 (a large number).

Step 3 : We iterate from 1 to 14 (the number of rooms). In each iteration, we iterate once more from 1 to 14, and again we iterate from 1 to 14 (meaning we have 3 nested loops). Then if the cell representing the distance to go from the room of the second loop to the room of the third loop is bigger than the sum of the distance to go from the room of the second loop to the room of the first loop and the distance to go from the room of the first loop to the room of the third loop, then this first distance becomes the sum.

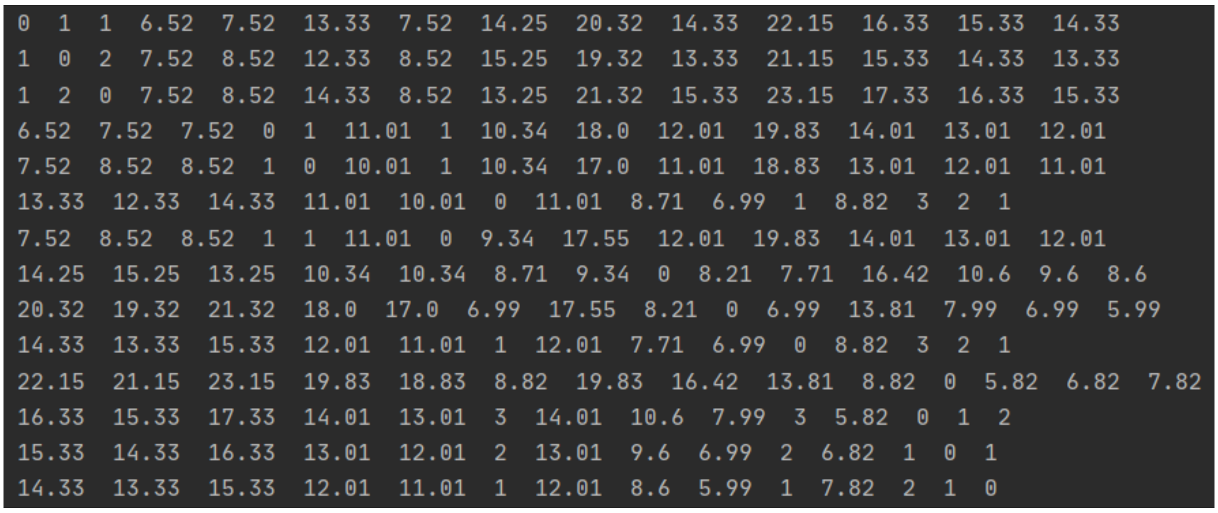
3. Implement the method and show the time to travel for any pair of rooms for both models.

Here is the result for the crewmates. How can we read that ? We must take the room from which we want to start as the number of our line (+1 because it starts as 0) and the room from which we want to end as the number of our column (+1 because it starts as 0) and check the intersection. So, for example, let's say a crewmate wants to go from Cafeteria to Electrical. We look at the graph from question 2 and we see that Cafeteria is room 5 and Electrical room 6. So, we look at the intersection from line 6 and column 7 and we find 19.21 which means it's the distance in cm and the time in sec that it will take to a crewmate to go (in the fastest way possible) from Cafeteria to Electrical.



```
0  8.31  8.31  6.52  18.01  20.64  20.08  21.56  29.77  29.27  34.0  28.18  38.34  30.16
8.31  0  9.54  7.75  9.7  12.33  21.31  22.2  30.41  22.31  25.69  19.87  30.03  30.8
8.31  9.54  0  7.75  19.24  21.87  11.77  13.25  21.46  20.96  34.77  29.41  33.45  21.85
6.52  7.75  7.75  0  17.45  20.08  19.52  21.0  29.21  28.71  33.44  27.62  37.78  29.6
18.01  9.7  19.24  17.45  0  10.01  29.22  19.88  28.09  19.99  23.37  17.55  27.71  28.48
20.64  12.33  21.87  20.08  10.01  0  19.21  9.87  18.08  9.98  13.36  7.54  17.7  18.47
20.08  21.31  11.77  19.52  29.22  19.21  0  9.34  17.55  17.05  30.86  26.75  29.54  17.94
21.56  22.2  13.25  21.0  19.88  9.87  9.34  0  8.21  7.71  21.52  17.41  20.2  8.6
29.77  30.41  21.46  29.21  28.09  18.08  17.55  8.21  0  15.92  18.91  19.71  17.59  5.99
29.27  22.31  20.96  28.71  19.99  9.98  17.05  7.71  15.92  0  23.34  17.52  27.68  16.31
34.0  25.69  34.77  33.44  23.37  13.36  30.86  21.52  18.91  23.34  0  5.82  9.36  12.92
28.18  19.87  29.41  27.62  17.55  7.54  26.75  17.41  19.71  17.52  5.82  0  10.16  13.72
38.34  30.03  33.45  37.78  27.71  17.7  29.54  20.2  17.59  27.68  9.36  10.16  0  11.6
30.16  30.8  21.85  29.6  28.48  18.47  17.94  8.6  5.99  16.31  12.92  13.72  11.6  0
```

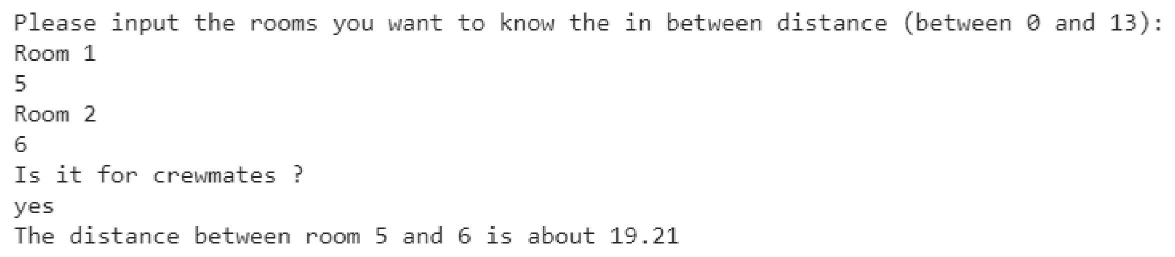
And here is the result for the impostors. We can see that for an impostor, it's now faster to go from Cafeteria to Electrical, because it now takes only 11.01 (both cm and sec).



```
0  1  1  6.52  7.52  13.33  7.52  14.25  20.32  14.33  22.15  16.33  15.33  14.33
1  0  2  7.52  8.52  12.33  8.52  15.25  19.32  13.33  21.15  15.33  14.33  13.33
1  2  0  7.52  8.52  14.33  8.52  13.25  21.32  15.33  23.15  17.33  16.33  15.33
6.52  7.52  7.52  0  1  11.01  1  10.34  18.0  12.01  19.83  14.01  13.01  12.01
7.52  8.52  8.52  1  0  10.01  1  10.34  17.0  11.01  18.83  13.01  12.01  11.01
13.33  12.33  14.33  11.01  10.01  0  11.01  8.71  6.99  1  8.82  3  2  1
7.52  8.52  8.52  1  1  11.01  0  9.34  17.55  12.01  19.83  14.01  13.01  12.01
14.25  15.25  13.25  10.34  10.34  8.71  9.34  0  8.21  7.71  16.42  10.6  9.6  8.6
20.32  19.32  21.32  18.0  17.0  6.99  17.55  8.21  0  6.99  13.81  7.99  6.99  5.99
14.33  13.33  15.33  12.01  11.01  1  12.01  7.71  6.99  0  8.82  3  2  1
22.15  21.15  23.15  19.83  18.83  8.82  19.83  16.42  13.81  8.82  0  5.82  6.82  7.82
16.33  15.33  17.33  14.01  13.01  3  14.01  10.6  7.99  3  5.82  0  1  2
15.33  14.33  16.33  13.01  12.01  2  13.01  9.6  6.99  2  6.82  1  0  1
14.33  13.33  15.33  12.01  11.01  1  12.01  8.6  5.99  1  7.82  2  1  0
```

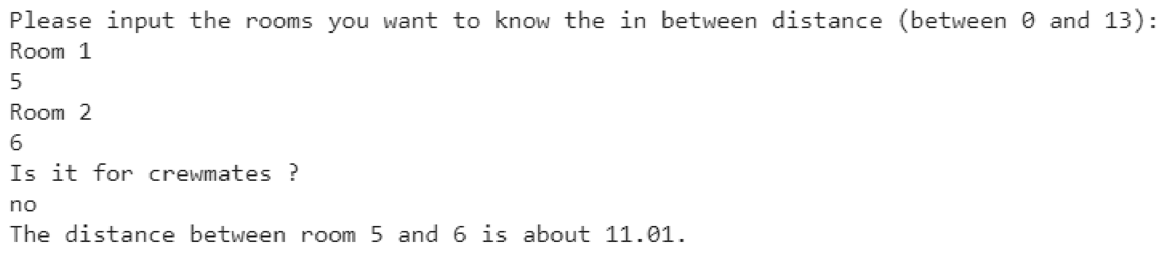
We decided to implement a user-friendly method that will allow you to check the shortest distance between two rooms in a cleaner way.

Here is the same example than before for crewmates :



```
Please input the rooms you want to know the in between distance (between 0 and 13):
Room 1
5
Room 2
6
Is it for crewmates ?
yes
The distance between room 5 and 6 is about 19.21
```

And here is the same example but for impostors :



```
Please input the rooms you want to know the in between distance (between 0 and 13):
Room 1
5
Room 2
6
Is it for crewmates ?
no
The distance between room 5 and 6 is about 11.01.
```

We can see that we found the same distance/time than when we directly looked into the matrix.

Step 4: Secure the last tasks

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

In this section we will use the same adjacency matrix as step 3 for crewmates with the same weight between rooms since it contains all travelled times between rooms.

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

This is a well-known problem in graph theory called travelling salesman problem. We need to find an hamiltonian path minimizing the travelled distance. An hamiltonian path is a path in graph passing through all nodes only once.

3. Argue about an algorithm solving your problem.

To solve this problem, we decided to use a recursive backtracking algorithm to find all Hamiltonian paths in the graph and then to sort all the paths founded with respect to the travelled distance and pick the best one (meaning the one minimizing the distance) !

4. Implement the algorithm and show a solution.

We implement a graph class with some function to create easily the adjacency matrix (add edge function). We needed a function to get the neighbors of a node to compute the hamiltonian in the recursive function to get Hamilton paths.

We don't want to use a brute force algorithm, because it would be very slow (if we note r the number of rooms in the map, there are r! different sequences of rooms that could be Hamiltonian paths). So, we used a faster approach.

Here is the algorithm we used to get all the Hamiltonian paths of the graph beginning on one node (note that we do this for every node and then we sort them by ascending order to get the lowest distance/time first).

Step 1 : First we start with just one node and an empty path.

Step 2 : We check if the same node is twice in the path. If yes, we exit the algorithm and no such Hamiltonian path exists.

Step 3 : If the path thus far is valid, we calculate the distance by adding the weight between each node (which represents the distance in cm and the time in sec between two rooms in our case).

Step 4 : Now we get all the neighbors of the last node of our current path, and we start again our algorithm at step 2 for all of the neighbors (meaning it's a recursive algorithm, an algorithm which calls himself several times).

Step 5 : At the end, when we calculated all of the possible Hamiltonian paths starting at one room, we start again but starting at another room (we do that for the 14 rooms of the map).

Step 5 : We sort the Hamiltonian paths in an ascending order according to the distance, and the path at the top of our list will be the optimal path because it will be the path that travels in every room exactly once with the lowest distance and time.

When we run the algorithm, it returns 80 lines sorted. Here is the final result (truncated) :

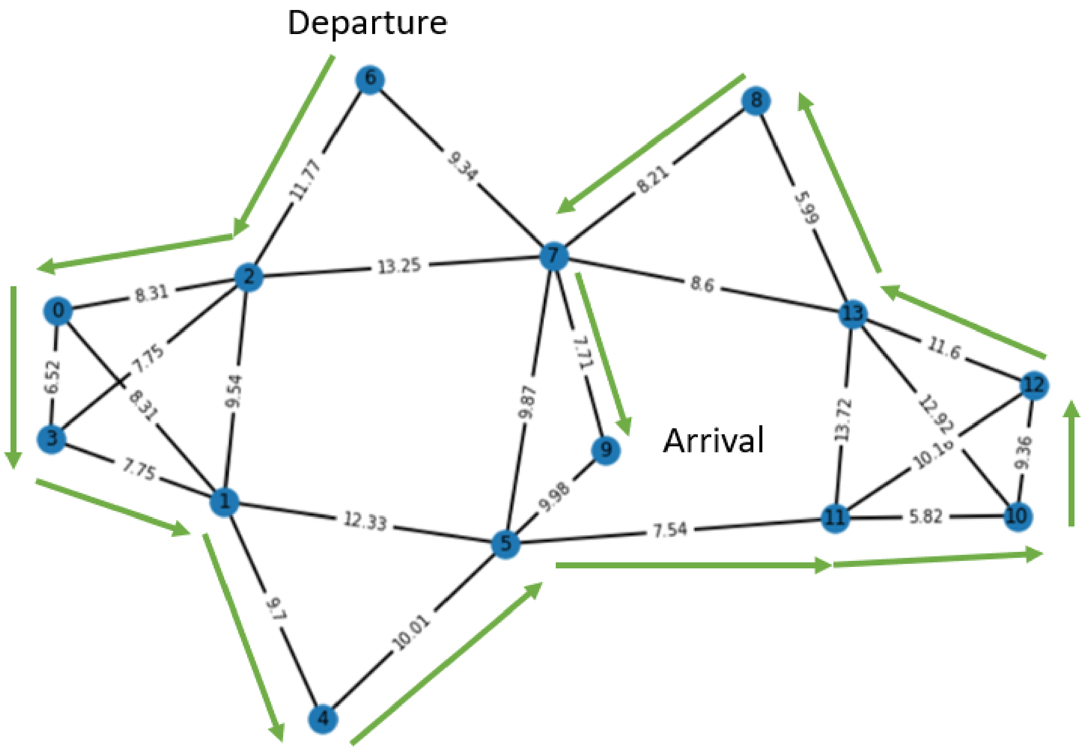
```
[[[6, 2, 0, 3, 1, 4, 5, 11, 10, 12, 13, 8, 7, 9], 110.28999999999998],
 [[6, 2, 3, 0, 1, 4, 5, 11, 10, 12, 13, 8, 7, 9], 110.28999999999998],
 [[9, 7, 8, 13, 12, 10, 11, 5, 4, 1, 0, 3, 2, 6], 110.29],
 [[9, 7, 8, 13, 12, 10, 11, 5, 4, 1, 3, 0, 2, 6], 110.29],
 [[4, 1, 0, 3, 2, 6, 7, 9, 5, 11, 10, 12, 13, 8], 111.38999999999999],
 [[4, 1, 3, 0, 2, 6, 7, 9, 5, 11, 10, 12, 13, 8], 111.38999999999999],
 [[8, 13, 12, 10, 11, 5, 9, 7, 6, 2, 0, 3, 1, 4], 111.38999999999999],
 [[8, 13, 12, 10, 11, 5, 9, 7, 6, 2, 3, 0, 1, 4], 111.38999999999999],
 [[8, 13, 12, 10, 11, 5, 4, 1, 0, 3, 2, 6, 7, 9], 111.41999999999999],
 [[8, 13, 12, 10, 11, 5, 4, 1, 3, 0, 2, 6, 7, 9], 111.41999999999999],
 [[9, 7, 6, 2, 0, 3, 1, 4, 5, 11, 10, 12, 13, 8], 111.42000000000002],
 [[9, 7, 6, 2, 3, 0, 1, 4, 5, 11, 10, 12, 13, 8], 111.42000000000002],
```

Note that some paths are identical but doesn't begin on the same vertex (1 and 4 for instance).

Hence, the shortest path to go through each room and finish all the remaining tasks as fast as possible is the first one with a distance of 110,289cm or about 110 seconds.

We can see that the shortest path is : 6 → 2 → 0 → 3 → 1 → 4 → 5 → 11 → 10 → 12 → 13 → 8 → 7 → 9

Which corresponds to : Electrical → Lower E. →Reactor → Security → Upper E. → Medbay → Cafeteria → Weapons → O2 → Navigations → Shield → The room without a name between Shield and Storage → Storage → The room without a name between Storage and Cafeteria.


Departure
Arrival
6
2
0
3
1
4
5
7
8
9
10
11
12
13
11.77
9.34
8.21
5.99
13.25
8.6
8.31
7.75
6.52
8.31
9.54
7.71
9.87
11.6
13.72
12.92
10.16
9.36
7.75
12.33
9.98
7.54
5.82
9.7
10.01