IOS-1
GUILLON BONY Rémi
POUPET Vincent

Advanced Data Structure and Algorithms

Mini-Problem Among Us

Table des matières

IOS-1
GUILLON BONY Rémi
POUPET Vincent

Step 1 : To organize the tournament

Propose a data structure to represent a Player and its Score

We first had to define the information we wanted our data structure to contain. We had to be able to access the name of the player, its Score but also know its position in the AVL tree that we will be using. Thus, the data structure had also to link a node of the tree to its children. Therefore, we decided to create a Node Class containing the following attributes: Player, Score, Left_Node and Right_Node. We also added the Height attribute to be able to check for balance in our tree.

But after realizing that multiple players could have the same score, we became concerned about creating a node for each player and using its score as key. Indeed, we would eventually have to delete or retrieve a specific player and this task would not be easy if its key wasn't unique.

So we decided to change our definition of a Node to 'an element containing the list of the players having a defined score (which we would consider as the node's key)'.

Here is an example of the result of an Inorder traversal of a test AVL Tree :

```
Score : 10 ; Player List : ['Samuel', 'Vinhed']
Score : 15 ; Player List : ['Vadim']
Score : 20 ; Player List : ['Markus']
Score : 25 ; Player List : ['Pierre']
Score : 50 ; Player List : ['Henry']
```

We can see that having multiple players with the same score isn't an issue, as the Score value stays unique. If a new player is added, a new node is created for him is no other player has the same score. If someone already has the same score as him, the new player's name is added to the Player's list of this node.

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

We first thought that we could use a generic list of elements of our newly created Node class. But doing a sequential/linear search in this list would have a complexity of O(n). A dichotomic search would have a complexity of only O(logn), but it requires a sorted list and sorting the list would increase the complexity of the algorithm. Indeed, one of the best sorting algorithms is the merge sort which has a complexity of O(nlogn). Thus, using a list would imply using an algorithm of O(nlogn) whereas the problem statement requires us to find a data structure with a complexity of O(logn).

Therefore, we ruled out the possibility of using a list.

We then thought of using a tree and had to decide which type

We chose to use an AVL Tree (as mentioned in the previous question). Indeed, its worst-case time complexity for search, insertion and deletion are O(logn), as asked by the statement and it is the most optimised type of tree that we know of.

We then created an AVL_Tree class containing the 'insert' function which allows us to add an element to the AVL structure. We also added the subsidiaries functions needed to make the first one work correctly

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

We first wondered if we really didn't have to take into account the rules given in the problem for the points scored but after sending an email to the person in charge to confirm it, we settled for a simple function which returns a value between 0 et 12.

After checking the rest of the problem, we realized that we would have to update the score of the players with this new score at some point.

To that extend we created a second function called 'Random_Game_Score_Distribution' which uses the previous method to update the score of all players in a given tournament. This tournament list is given by the functions used to create random/ranked games described below.

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

The previous method returns a dictionary with 10 keys (the names of the players) and 10 values (their score).

We first thought of creating a copy of the AVL_Tree with a new root and inserting in it all the nodes of the original root while just changing the values of the scores of the nodes with a player value equal to one of the dictionary's keys. Thus, we would have always worked with 2 AVL Trees: an up-to-date version and another version used only to temporarily store the other while the results are being updated.

We finally decided against it. Indeed, it would have meant keeping track of 2 AVL Trees sized variables which would have taken a toll on the memory if their size was big enough.

Therefore, we decided to create a Node updating function which would be composed of 2 steps:

- The removal of the player from the player's list of the Node he is in (and the Node deletion if the player was the only element of this list) by using a blank Tree and the insertion function to add to it everything except the element we want to delete and then using this new Tree as reference.
- The addition of the player to the player's list of the Node corresponding to its updated score (or the creation of this Node if none already exists for this score)

We first created the 'delete' function adapted to our project, and then we simply add to combine deletion and insertion in the 'update' function.

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

In order to create our random games, we need to define the number of games that we want as well as who's taking part in them.

Whenever a player is inserted in the Tree, his name is also added to the liste_joueurs attributes of the AVL Tree. When the player is deleted, his name is also removed from this list.

We then randomly assign each player to a random game and return the list of games, each containing the list of the participants

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

We first travel the AVL Tree with an Inorder algorithm in order to obtain a ranked list of players (from the worst to the best).

After that we use a process similar to the one we created in the previous question to obtain a list of games based on the ranking of the players.

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

We first created the 'Drop_Worst' function which return a copy of our current Tree, except that the n worst players have been removed (n being a parameter). If some players have the same score, the algorithm will randomly drop one of them until either all (players having the same score) have been dropped or the number of players to remove has been reached.

To put this method in action, we also created the 'Manche' and 'Jeu_Avant_Finalistes' functions. 'Manche' function is self-explanatory, we can choose if it has to be a random or ranked game. The 'Jeu_Avant_Finalistes' function allow us to play 3 random games then ranked games until the number of players left has reached n (n being a parameter).

NB : We used a lot of parameters in these functions (nb of total players, nb of players to drop each game, nb of finalists, …). Indeed, it was easier to test our functions with smaller and different trees this way. The parameters would obviously have to be set up for our problem if we really wanted to use our program in real life.

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

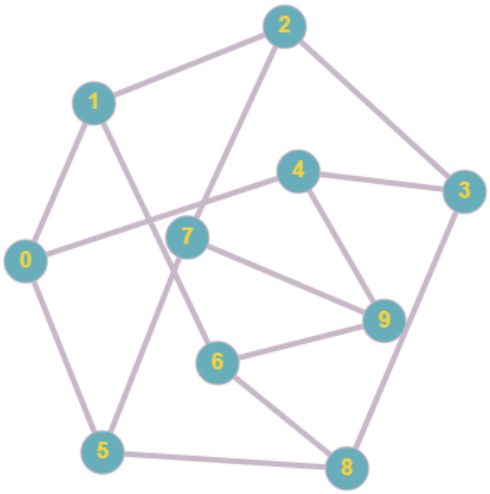
When we are done with the 'Jeu_Avant_Finalistes' function, we can retrieve from it an AVL Tree containing only the TOP X players (in our tests it is 4 but it can be changed to 10). We display their name and set their score to 0.

Afterwards, we launch 5 games, print each time their effects on the scores and finally we display the podium. If some players have the same score, the algorithm will randomly add one of them to the list of winners until either all (players having the same score) have been inserted or the number of winners (=3 because a podium is only three players) has been reached. Because players of the list are ranked from the best to the worst, the first one to be inserted will have the best ranking.

Step 2 : Professor Layton < Guybrush Threepwood < You

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

We can represent the relation have seen by an unordered graph. Indeed we realized that when A has seen B, B has always also seen A. Therefore we can represent the graph as such :



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

If 0 is dead, then he must have been killed by someone who saw him (1, 4 or 5). That means that one of them must be the impostor.

We also know that the 2 impostors never walk together, which means that they haven't seen each other. This means that to create our sets of probable impostors, we must create sets composed of 1,4 or 5 on one side, and another player that the first suspect hasn't seen.

On our graph, this means that if 1 is the impostor, the other impostor can't be represented by a vertex which has an edge between himself and 1. Idem for 4 and 5.

NB : No one else has been killed, but each player has come across other people. This means that the second impostor has seen players but hasn't killed them. Given that 1,4 and 5 haven't seen each other, it is possible that the 2 impostors are among them and he just didn't kill 0 when he had the chance.

Argue about an algorithm solving your problem

To create an algorithm solving this problem, we first need get access to the graph. To do so we need to generate the adjacency matrix and give it as parameter. Then, we can check on the column of the killed player who he has seen during the game. Doing so will give us the options for the left side of the sets of impostors. For each of these options, we'll have to check who the suspect hasn't seen and complete a set with each of these unseen players.

Once this is done, we can return the list of probable sets.

We can even improve it by authorizing another parameter: an existing list of probable sets of impostors. This way we not only determine the sets of probable suspects for one murder, but we can also modify the list of suspects after each murder. Of course, we would also need an updated adjacency list.

Implement the algorithm and show a solution

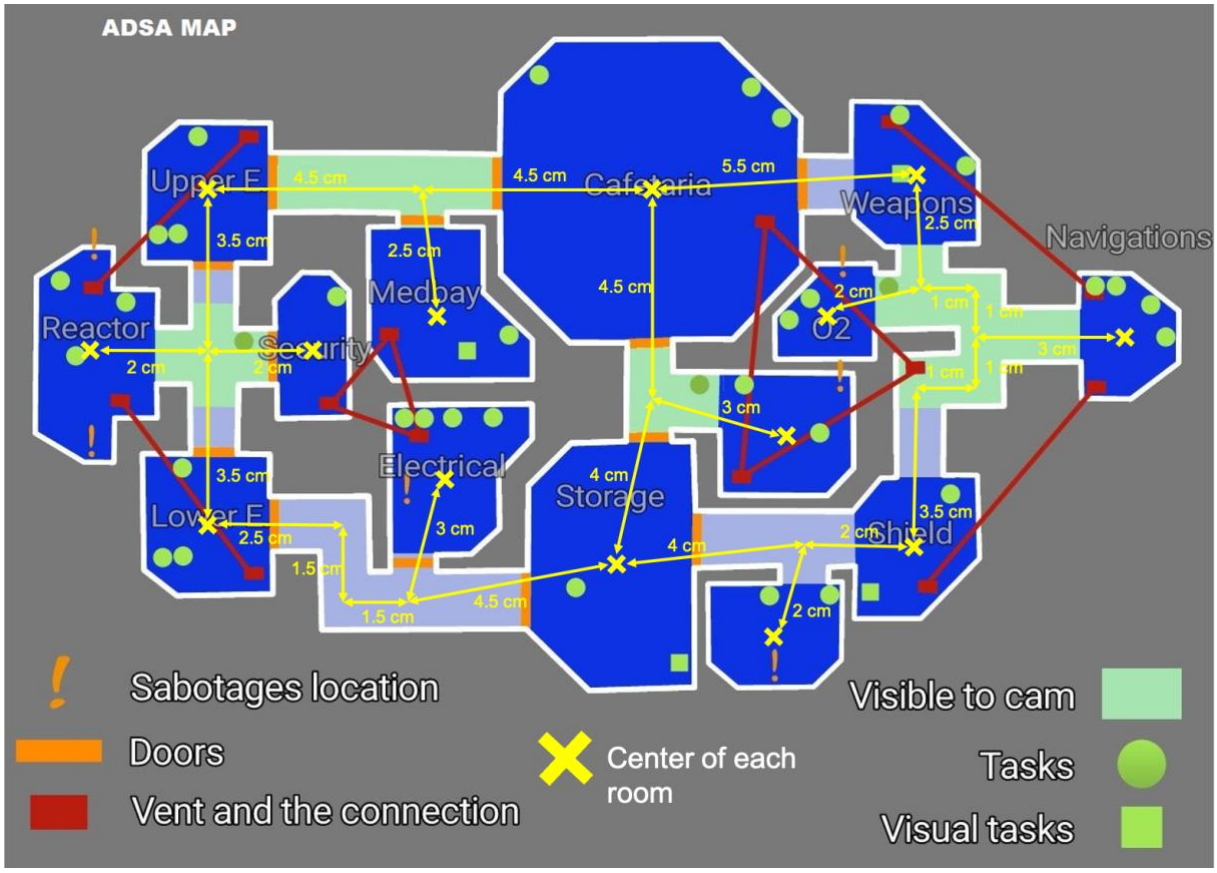
```
[Running] python -u "c:\Users\guill\OneDrive\Documents\OneDrive - De Vinci\Année 4\Advanced Data Structures & Algorithms\
[(1, 3), (1, 4), (1, 5), (1, 7), (1, 8), (1, 9), (4, 2), (4, 5), (4, 6), (4, 7), (4, 8), (5, 2), (5, 3), (5, 6), (5, 9)]
```

Here are all the possible sets of suspects.

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

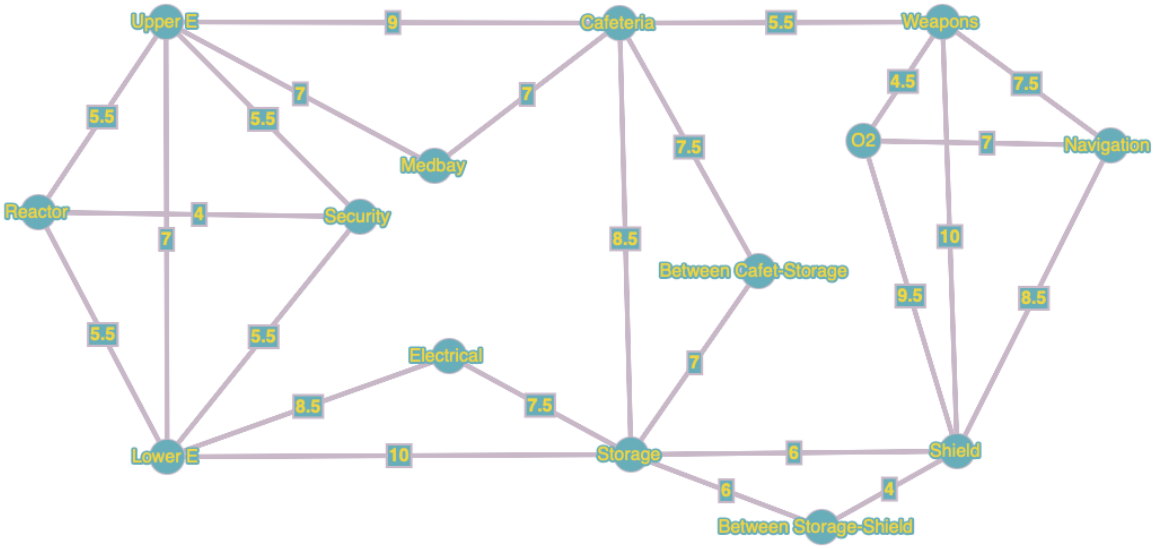
Presents and argue about the two models of the map

First of all, we add some details to the map : the center of each room and some measures to help us design our weighted graph.

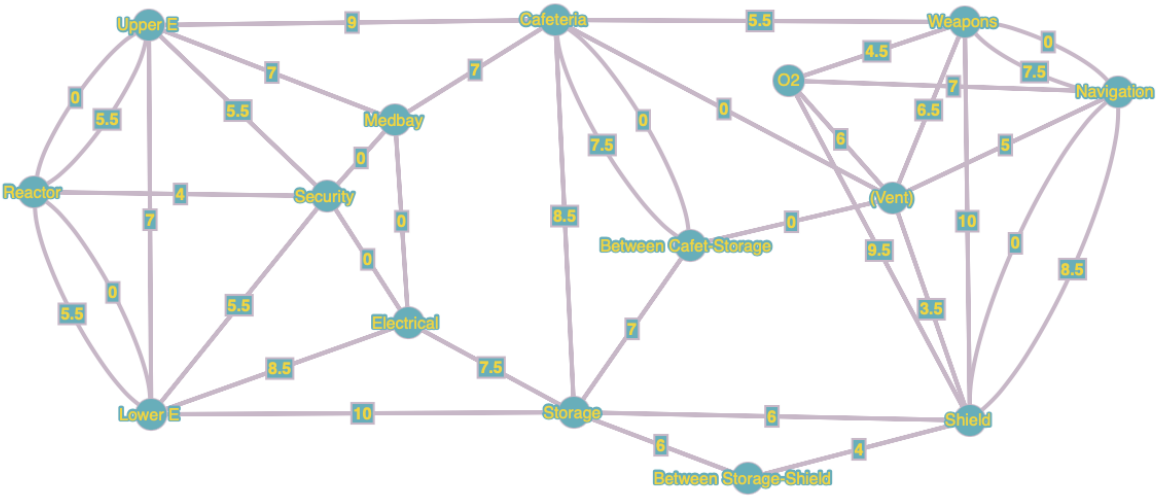


From that image we can now create our two models which will be weighted graphs. We can use the measures as it is to weight our graphs because in the problem, it is said that the time to travel 1cm is 1sec. We consider that player walk in the middle of the corridors.

For the crewmates, we created a graph in which each vertex is the center of a room and they are connected by weighted edges. As said earlier, the weights are the distance in centimeter between each room's center. It thus leads us to the following graph :



For the impostors it is the same graph but with as they can use vents, we had to add several edges. Those edges that we add all have a weight of 0 because in the problem, it is said that taking a vent do not take time. You will also notice that we add a vertex for the vent near the shield room because it is in a corridor. Started from the vent, we also had to add the edges going to the rooms it's connected with. It thus leads us to the following graph:



Argue about a pathfinding algorithm to implement

We first thought that we could use the Dijkstra algorithm because it was the one we knew the most about and we didn't had any negative weight. Thus, we would have created a function with source and destination as parameters.

IOS-1
GUILLON BONY Rémi
POUPET Vincent

We quickly realized that Dijkstra's algorithm wasn't the most suited for this problem. Therefore, we decided to use the Floyd-Warshall algorithm. Like the Bellman-Ford's algorithm or the Dijkstra's algorithm, it computes the shortest path in a graph. However, Bellman-Ford and Dijkstra are both single-source, shortest-path algorithm. This means they only compute the shortest path from a single source. Floyd-Warshall, on the other hand, computes the shortest distances between every pair of vertices in the input graph which is exactly what we need.

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

We first need the adjacency matrix of each graph to give our code a representation of the scenario (crewmate or impostor). Afterwards, we apply to it the Floyd-Warshall algorithm and obtain the following for each scenario:

CREWMATES

	Upper E	Lower E	Reactor	Security	Cafeteria	Medbay	Weapons	O2	Navigation	Shield	Storage	Electrical	Between Cafet-Storage	Between Storage-Shield
Upper E	0.0	7.0	5.5	5.5	9.0	7.0	14.5	19.0	22.0	23.0	17.0	15.5	16.5	23.0
Lower E	7.0	0.0	5.5	5.5	16.0	14.0	21.5	25.5	24.5	16.0	10.0	8.5	17.0	16.0
Reactor	5.5	5.5	0.0	4.0	14.5	12.5	20.0	24.5	27.5	21.5	15.5	14.0	22.0	21.5
Security	5.5	5.5	4.0	0.0	14.5	12.5	20.0	24.5	27.5	21.5	15.5	14.0	22.0	21.5
Cafeteria	9.0	16.0	14.5	14.5	0.0	7.0	5.5	10.0	13.0	14.5	8.5	16.0	7.5	14.5
Medbay	7.0	14.0	12.5	12.5	7.0	0.0	12.5	17.0	20.0	21.5	15.5	22.5	14.5	21.5
Weapons	14.5	21.5	20.0	20.0	5.5	12.5	0.0	4.5	7.5	10.0	14.0	21.5	13.0	14.0
O2	19.0	25.5	24.5	24.5	10.0	17.0	4.5	0.0	7.0	9.5	15.5	23.0	17.5	13.5
Navigation	22.0	24.5	27.5	27.5	13.0	20.0	7.5	7.0	0.0	8.5	14.5	22.0	20.5	12.5
Shield	23.0	16.0	21.5	21.5	14.5	21.5	10.0	9.5	8.5	0.0	6.0	13.5	13.0	4.0
Storage	17.0	10.0	15.5	15.5	8.5	15.5	14.0	15.5	14.5	6.0	0.0	7.5	7.0	6.0
Electrical	15.5	8.5	14.0	14.0	16.0	22.5	21.5	23.0	22.0	13.5	7.5	0.0	14.5	13.5
Between Cafet-Storage	16.5	17.0	22.0	22.0	7.5	14.5	13.0	17.5	20.5	13.0	7.0	14.5	0.0	13.0
Between Storage-Shield	23.0	16.0	21.5	21.5	14.5	21.5	14.0	13.5	12.5	4.0	6.0	13.5	13.0	0.0

IMPOSTORS

	Upper E	Lower E	Reactor	Security	Cafeteria	Medbay	Weapons	O2	Navigation	Shield	Storage	Electrical	Between Cafet-Storage	Between Storage-Shield	Vent btwn Shield-Nav
Upper E	0.0	0.0	0.0	4.0	9.0	4.0	12.5	15.0	12.5	12.5	10.0	4.0	9.0	16.0	9.0
Lower E	0.0	0.0	0.0	4.0	9.0	4.0	12.5	15.0	12.5	12.5	10.0	4.0	9.0	16.0	9.0
Reactor	0.0	0.0	0.0	4.0	9.0	4.0	12.5	15.0	12.5	12.5	10.0	4.0	9.0	16.0	9.0
Security	4.0	4.0	4.0	0.0	7.0	0.0	10.5	13.0	10.5	10.5	7.5	0.0	7.0	13.5	7.0
Cafeteria	9.0	9.0	9.0	7.0	0.0	7.0	3.5	6.0	3.5	3.5	7.0	7.0	0.0	7.5	0.0
Medbay	4.0	4.0	4.0	0.0	7.0	0.0	10.5	13.0	10.5	10.5	7.5	0.0	7.0	13.5	7.0
Weapons	12.5	12.5	12.5	10.5	3.5	10.5	0.0	4.5	0.0	0.0	6.0	10.5	3.5	4.0	3.5
O2	15.0	15.0	15.0	13.0	6.0	13.0	4.5	0.0	4.5	4.5	10.5	13.0	6.0	8.5	6.0
Navigation	12.5	12.5	12.5	10.5	3.5	10.5	0.0	4.5	0.0	0.0	6.0	10.5	3.5	4.0	3.5
Shield	12.5	12.5	12.5	10.5	3.5	10.5	0.0	4.5	0.0	0.0	6.0	10.5	3.5	4.0	3.5
Storage	10.0	10.0	10.0	7.5	7.0	7.5	6.0	10.5	6.0	6.0	0.0	7.5	7.0	6.0	7.0
Electrical	4.0	4.0	4.0	0.0	7.0	0.0	10.5	13.0	10.5	10.5	7.5	0.0	7.0	13.5	7.0
Between Cafet-Storage	9.0	9.0	9.0	7.0	0.0	7.0	3.5	6.0	3.5	3.5	7.0	7.0	0.0	7.5	0.0
Between Storage-Shield	16.0	16.0	16.0	13.5	7.5	13.5	4.0	8.5	4.0	4.0	6.0	13.5	7.5	0.0	7.5
Vent btwn Shield-Nav	9.0	9.0	9.0	7.0	0.0	7.0	3.5	6.0	3.5	3.5	7.0	7.0	0.0	7.5	0.0

NB : We are aware that for each travel time from A to B, you can also fine the same result for the travel time from B to A.

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

If a player's travel time between two rooms belong to the relevant interval, then the player is necessarily an impostor. Please note that for each interval, the value on the right is the minimum time a crewmate can take to travel between the two rooms and the one on the left is the minimum time for an impostor. You can notice that we don't put the row/column 'Vent btwn Shield-Nav' because if you take the vent, you are automatically an impostor.

	Upper E	Lower E	Reactor	Security	Cafeteria	Medbay	Weapons	O2	Navigation	Shield	Storage	Electrical	Between Cafet-Storage	Between Storage-Shield
Upper E	[0.0,0.0[	[0.0,7.0[	[0.0,5.5[	[4.0,5.5[	[9.0,9.0[	[4.0,7.0[	[12.5,14.5[	[15.0,19.0[	[12.5,22.0[	[12.5,23.0[	[10.0,17.0[	[4.0,15.5[	[9.0,16.5[	[16.0,23.0[
Lower E	[0.0,7.0[	[0.0,0.0[	[0.0,5.5[	[4.0,5.5[	[9.0,16.0[	[4.0,14.0[	[12.5,21.5[	[15.0,25.5[	[12.5,24.5[	[12.5,16.0[	[10.0,10.0[	[4.0,8.5[	[9.0,17.0[	[16.0,16.0[
Reactor	[0.0,5.5[	[0.0,5.5[	[0.0,0.0[	[4.0,4.0[	[9.0,14.5[	[4.0,12.5[	[12.5,20.0[	[15.0,24.5[	[12.5,27.5[	[12.5,21.5[	[10.0,15.5[	[4.0,14.0[	[9.0,22.0[	[16.0,21.5[
Security	[4.0,5.5[	[4.0,5.5[	[4.0,4.0[	[0.0,0.0[	[7.0,14.5[	[0.0,12.5[	[10.5,20.0[	[13.0,24.5[	[10.5,27.5[	[10.5,21.5[	[7.5,15.5[	[0.0,14.0[	[7.0,22.0[	[13.5,21.5[
Cafeteria	[9.0,9.0[	[9.0,16.0[	[9.0,14.5[	[7.0,14.5[	[0.0,0.0[	[7.0,7.0[	[3.5,5.5[	[6.0,10.0[	[3.5,13.0[	[3.5,14.5[	[7.0,8.5[	[7.0,16.0[	[0.0,7.5[	[7.5,14.5[
Medbay	[4.0,7.0[	[4.0,14.0[	[4.0,12.5[	[0.0,12.5[	[7.0,7.0[	[0.0,0.0[	[10.5,12.5[	[13.0,17.0[	[10.5,20.0[	[10.5,21.5[	[7.5,15.5[	[0.0,22.5[	[7.0,14.5[	[13.5,21.5[
Weapons	[12.5,14.5[	[12.5,21.5[	[12.5,20.0[	[10.5,20.0[	[3.5,5.5[	[10.5,12.5[	[0.0,0.0[	[4.5,4.5[	[0.0,7.5[	[0.0,10.0[	[6.0,14.0[	[10.5,21.5[	[3.5,13.0[	[4.0,14.0[
O2	[15.0,19.0[	[15.0,25.5[	[15.0,24.5[	[13.0,24.5[	[6.0,10.0[	[13.0,17.0[	[4.5,4.5[	[0.0,0.0[	[4.5,7.0[	[4.5,9.5[	[13.0,23.0[	[6.0,17.5[	[6.0,17.5[	[8.5,13.5[
Navigation	[12.5,22.0[	[12.5,24.5[	[12.5,27.5[	[10.5,27.5[	[3.5,13.0[	[10.5,20.0[	[0.0,7.5[	[4.5,7.0[	[0.0,0.0[	[0.0,8.5[	[6.0,14.5[	[10.5,22.0[	[3.5,20.5[	[4.0,12.5[
Shield	[12.5,23.0[	[12.5,16.0[	[12.5,21.5[	[10.5,21.5[	[3.5,14.5[	[10.5,21.5[	[0.0,10.0[	[4.5,9.5[	[0.0,8.5[	[0.0,0.0[	[6.0,6.0[	[10.5,13.5[	[3.5,13.0[	[4.0,4.0[
Storage	[10.0,17.0[	[10.0,10.0[	[10.0,15.5[	[7.5,15.5[	[7.0,8.5[	[7.5,15.5[	[6.0,14.0[	[10.5,15.5[	[6.0,14.5[	[6.0,6.0[	[0.0,0.0[	[7.5,7.5[	[7.0,7.0[	[6.0,6.0[
Electrical	[4.0,15.5[	[4.0,8.5[	[4.0,14.0[	[0.0,14.0[	[7.0,16.0[	[0.0,22.5[	[10.5,21.5[	[13.0,23.0[	[10.5,22.0[	[10.5,13.5[	[7.5,7.5[	[0.0,0.0[	[7.0,14.5[	[13.5,13.5[
Between Cafet-Storage	[9.0,16.5[	[9.0,17.0[	[9.0,22.0[	[7.0,22.0[	[0.0,7.5[	[7.0,14.5[	[3.5,13.0[	[6.0,17.5[	[3.5,20.5[	[3.5,13.0[	[7.0,7.0[	[7.0,14.5[	[0.0,0.0[	[7.5,13.0[
Between Storage-Shield	[16.0,23.0[	[16.0,16.0[	[16.0,21.5[	[13.5,21.5[	[7.5,14.5[	[13.5,21.5[	[4.0,14.0[	[8.5,13.5[	[4.0,12.5[	[4.0,4.0[	[6.0,6.0[	[13.5,13.5[	[7.5,13.0[	[0.0,0.0[

Step 4: Secure the last tasks

Presents and argue about the model of the map

The map represents the degree of safety/danger of each room. It's very useful to know if a task a safe to do or not. We will then use it to determine the order in which the crewmates must travel each room to complete their tasks and expose themselves as little as possible. Indeed, we will prioritize a path linking all room but beginning by the most dangerous ones because the impostors might eventually split the group at some point.

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

We immediately thought about using the Hamiltonian Algorithm because it allow us to visit all the rooms by passing through each room exactly once.

To allow us to take the safest Hamiltonian path (the one beginning by the most dangerous rooms) we will create every Hamiltonian path possible from any room near the cafeteria which is the starting point after an emergency report. Indeed, starting directly from the cafeteria does not allow the creation of an Hamiltonian path because at some point, you would have to go through this room again.

In order to classify our Hamiltonian paths, we will penalize the ones which make us go through dangerous rooms at the very end for the reason we said earlier.

Argue about an algorithm solving your problem

We must first create a function returning all the possible Hamiltonian paths given a certain starting point. Indeed, we will want to check every possibility starting from the 5 rooms near the cafeteria ('Medbay', 'UpperE', 'Between Cafet-Storage', 'Weapons', 'Storage'). After having gathered every possibility, we will want to classify them according to their dangerosity. This means we will have to implement a penalty system which prevents paths visiting the dangerous rooms at last to have a good score. The path with the best score will be printed as it is the safest.

To penalize a room, we looked at its dangerosity and assigned a number between 0 et 1 based on its color and the scale given in the document.

To penalize a path, we multiplied the penalty of each room composing it by '1 + X/10', with X being its position in the path, and made a sum with them.

IOS-1
GUILLON BONY Rémi
POUPET Vincent

Implement the algorithm and show a solution

Here are all the possible Hamiltonian paths :

```
['Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Weapons', 'Navigation', 'O2', 'Shield', 'Between Storage-Shield']
['Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Weapons', 'O2', 'Navigation', 'Shield', 'Between Storage-Shield']
['Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical', 'Storage', 'Between Storage-Shield', 'Shield', 'O2', 'Navigation', 'Weapons', 'Cafeteria', 'Between Cafet-Storage']
['Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical', 'Storage', 'Between Storage-Shield', 'Shield', 'Navigation', 'O2', 'Weapons', 'Cafeteria', 'Between Cafet-Storage']
['Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Weapons', 'Navigation', 'O2', 'Shield', 'Between Storage-Shield']
['Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Weapons', 'O2', 'Navigation', 'Shield', 'Between Storage-Shield']
['Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical', 'Storage', 'Between Storage-Shield', 'Shield', 'O2', 'Navigation', 'Weapons', 'Cafeteria', 'Between Cafet-Storage']
['Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical', 'Storage', 'Between Storage-Shield', 'Shield', 'Navigation', 'O2', 'Weapons', 'Cafeteria', 'Between Cafet-Storage']
['Between Storage-Shield', 'Shield', 'O2', 'Navigation', 'Weapons', 'Cafeteria', 'Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage']
['Between Storage-Shield', 'Shield', 'O2', 'Navigation', 'Weapons', 'Cafeteria', 'Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage']
['Between Storage-Shield', 'Shield', 'O2', 'Navigation', 'Weapons', 'Cafeteria', 'Between Cafet-Storage', 'Storage', 'Electrical', 'Lower E', 'Reactor', 'Security', 'Upper E', 'Medbay']
['Between Storage-Shield', 'Shield', 'O2', 'Navigation', 'Weapons', 'Cafeteria', 'Between Cafet-Storage', 'Storage', 'Electrical', 'Lower E', 'Security', 'Reactor', 'Upper E', 'Medbay']
['Between Storage-Shield', 'Shield', 'Navigation', 'O2', 'Weapons', 'Cafeteria', 'Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage']
['Between Storage-Shield', 'Shield', 'Navigation', 'O2', 'Weapons', 'Cafeteria', 'Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical', 'Storage', 'Between Cafet-Storage']
['Between Storage-Shield', 'Shield', 'Navigation', 'O2', 'Weapons', 'Cafeteria', 'Between Cafet-Storage', 'Storage', 'Electrical', 'Lower E', 'Reactor', 'Security', 'Upper E', 'Medbay']
['Between Storage-Shield', 'Shield', 'Navigation', 'O2', 'Weapons', 'Cafeteria', 'Between Cafet-Storage', 'Storage', 'Electrical', 'Lower E', 'Security', 'Reactor', 'Upper E', 'Medbay']
['Weapons', 'Navigation', 'O2', 'Shield', 'Between Storage-Shield', 'Storage', 'Electrical', 'Lower E', 'Reactor', 'Security', 'Upper E', 'Medbay', 'Cafeteria', 'Between Cafet-Storage']
['Weapons', 'Navigation', 'O2', 'Shield', 'Between Storage-Shield', 'Storage', 'Electrical', 'Lower E', 'Security', 'Reactor', 'Upper E', 'Medbay', 'Cafeteria', 'Between Cafet-Storage']
['Weapons', 'Navigation', 'O2', 'Shield', 'Between Storage-Shield', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical']
['Weapons', 'Navigation', 'O2', 'Shield', 'Between Storage-Shield', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical']
['Weapons', 'O2', 'Navigation', 'Shield', 'Between Storage-Shield', 'Storage', 'Electrical', 'Lower E', 'Reactor', 'Security', 'Upper E', 'Medbay', 'Cafeteria', 'Between Cafet-Storage']
['Weapons', 'O2', 'Navigation', 'Shield', 'Between Storage-Shield', 'Storage', 'Electrical', 'Lower E', 'Security', 'Reactor', 'Upper E', 'Medbay', 'Cafeteria', 'Between Cafet-Storage']
['Weapons', 'O2', 'Navigation', 'Shield', 'Between Storage-Shield', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical']
['Weapons', 'O2', 'Navigation', 'Shield', 'Between Storage-Shield', 'Storage', 'Between Cafet-Storage', 'Cafeteria', 'Medbay', 'Upper E', 'Security', 'Reactor', 'Lower E', 'Electrical']
```

Here is the safest path :

```
['Medbay', 'Upper E', 'Reactor', 'Security', 'Lower E', 'Electrical', 'Storage', 'Between Storage-Shield', 'Shield', 'Navigation', 'O2', 'Weapons', 'Cafeteria', 'Between Cafet-Storage']
```