Lucas Witvoet

Anais Moussouni

Project Among Us

ADVANCED DATA STRUCTURE AND ALGORITHMS

Group IOS 3

Step 1: To organize the tournament

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

In order to represent a Player, we decided to create a class because it seems the most appropriate representation. Each player shares the same fundamentals attributes and methods, whatever their roles are, so we used a class named Player in order to keep our program organized and optimized. This class define the object Player and its different attributes implemented as follow.

Name	Type	Meaning
score	Int	It is the score of each player after one game.
finalScore	Int	Represent the mean of the scores earned by each player during the tournament.
impostor	Boolean	Define if a player is an impostor or not, it is settled at false by default.
player_id	Int	Each player has a unique id which allows us to identify them.
count_games	Int	It is a counter to indicate how many games did a player play.

So, the score is defined as an attribute to the object Player instantiated because every player has a score. We differentiated the score for each game played and the final score which is the mean of all the scores won after every game of the tournament.

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

   Then, we defined two different methods to calculate the score for each impostor and each crewmate, and randomly assign it.
   After that, to complete each game, we did a function to choose which team wins the game: the impostors or the crewmates.
   Finally, after each game we needed to reset the role for every player, so a new role can be assigned to them at the beginning of the next game.

3. Present and argue about a method that randomize player score at each game.

   To do that we defined two different methods, one to set the score for the impostors and one for the crewmates. Indeed, the score must be calculated differently depending on the role the player has. For an impostor, we set the method "*setScore_impostors*" which define randomly how many murders are committed by each impostor and how many of them stay undiscovered by the crewmates. Accordingly, to these numbers and the rules we attribute points to each impostor.
   Then, we define the method "*setScore_crewmate*" which just attribute a random score to every crewmate by respecting the rules.
   Finally, we know that if a team wins, the impostors gain 10 points and the crewmates 5 points. To respect that, we created a method named "*are_winner*" and we randomly decide which team is the winner and so gain the points.

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

   We created a method named "*finalScoreCalculation*" that define the final score after each game by calculating the mean thanks to the counter 'count_games' that we defined as an attribute to player. In order to do so, we update the final score by calling and apply the mean formula on the attributes from the class Player and store the mean calculated in "*finalScore*".

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

   In order to create an optimized data structure for the tournament, we created a class Game, which define every game played during the tournament. Thanks to this class, we simulate each game and to do so we randomly set two impostors among ten players.

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

   To create games based on ranking we used the fact that our AVL tree can be changed into an array of players. Thus, we can divide this array and obtain several arrays with 10 players each. Then, after each game the AVL tree is recreated and we update the ranking accordingly to the score each player gains during the game he just plays.
   .

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

   To drop the last 10 players after each game, we decide to reverse the order of the ranking and delete the first 10 players. Thus, the last 10 players left are the best one and to display the podium thanks to the "*podium*" method we just must reverse again. So, in our method "playTournament", when our ranking is back in order, we can apply the method "delete" that we created, which is used to delete nodes in our graph.

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

   The method named "*podium*" is the one that allows us to display the ranking of the 10 best players of the tournament. After deleting every player whose rank is superior to 10, we apply the method "*podium*" which display the player's id and his final score. Then, this method is called in the "*playTournament*" method on the last 10 players of the ranking.

Step 2: Professor Layton < Guybrush Threepwood < You

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

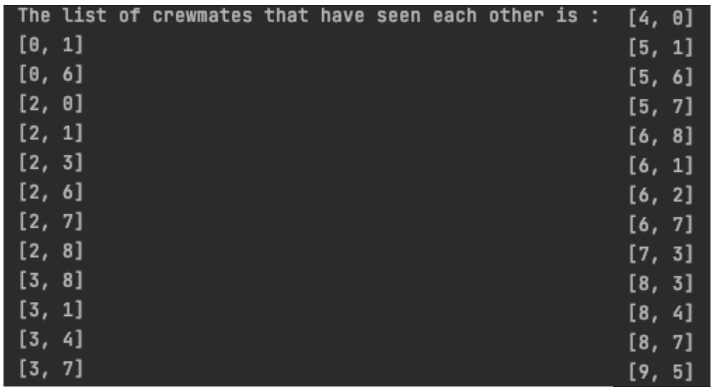
   We represented the relation between players as a graph in the form of a matrix.in this matrix each line represents the link between a crewmate that seen another crewmate and, in the columns, we have the crewmate represented by numbers from 0 to 9. There is no weight because a crewmate is sure he has seen the other crewmate.
   Example: If : Crewmate 1 have seen Crewmate 0,4,8 before the murder
   Crewmate 2 have seen Crewmate 4,9,1 before the murder
   We will have the following matrix representing the graph :

1	0
1	4
1	8
2	4
2	9
2	1

2. Thanks to a graph theory problem, present how to find a set of probable impostors.
   We can look at the link that the murdered player had with the other players. If it had a link with other players, they could be considered as impostor because they were in contact, we've him and could have killed him. We have to browse the graph to look for the links between the

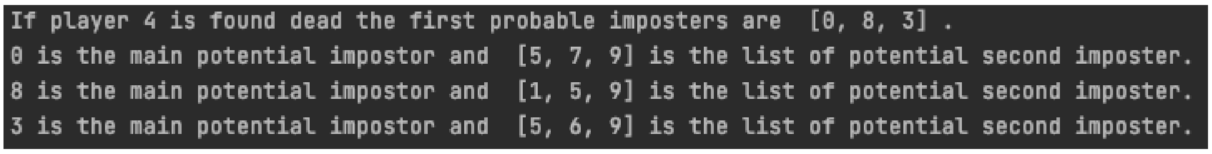
3. Argue about an algorithm solving your problem.
   We could implement two algorithms. The first on named finds_connection take in argument a player (represented by a node) and the graph represented by an array. We make two loops to browse the nodes of the graphs and the elements in those nodes. In order not to miss a lead of potential impostor we made a second function that helps us find which player was in contact with the potential impostor and return them

4. Implement the algorithm and show a solution.
   For the following example automatically, random graph generated by a function we created



```
The list of crewmates that have seen each other is :   [4, 0]
[0, 1]                                                 [5, 1]
[0, 6]                                                 [5, 6]
[2, 0]                                                 [5, 7]
[2, 1]                                                 [6, 8]
[2, 3]                                                 [6, 1]
[2, 6]                                                 [6, 2]
[2, 7]                                                 [6, 7]
[2, 8]                                                 [7, 3]
[3, 8]                                                 [8, 3]
[3, 1]                                                 [8, 4]
[3, 4]                                                 [8, 7]
[3, 7]                                                 [9, 5]
```

The function returns:



```
If player 4 is found dead the first probable imposters are  [0, 8, 3] .
0 is the main potential impostor and  [5, 7, 9] is the list of potential second imposter.
8 is the main potential impostor and  [1, 5, 9] is the list of potential second imposter.
3 is the main potential impostor and  [5, 6, 9] is the list of potential second imposter.
```

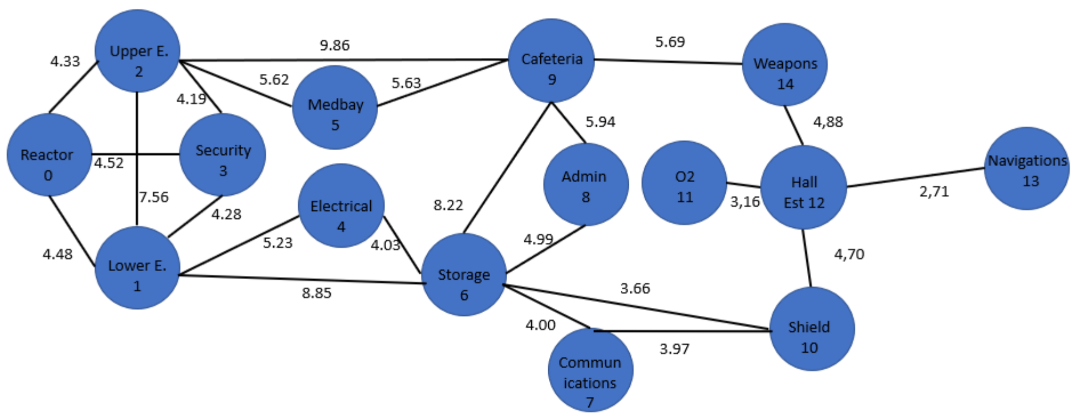
We can conclude that the impostor is probably one of the following players : the 0, 3 or 8

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.

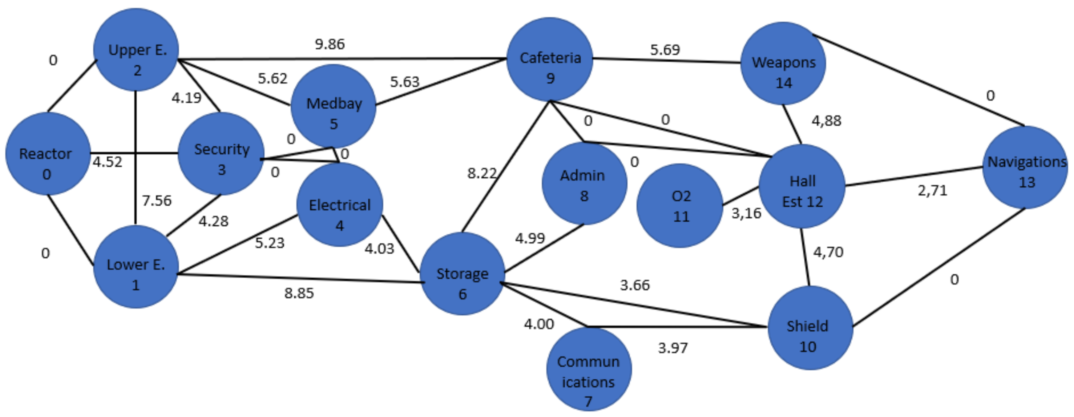
We choose to make two different graphs. The first one is the graph of the map for the crewmate. In the graph each room is represented by a vertex with a name and a unique number and the nodes are represented by a black line and the weight correspond to the distance between each center of rooms. We used this website to have more precise measure of the map and tried to have a scale of 1cm = 1 as described in the subject.

Graph Crewmate



The next graph represents the map but only for the impostor, the main difference is the existence of vent that allows impostors to travel from specific room instantly.

Graph Impostor



2. Argue about a pathfinding algorithm to implement.

We need to find an algorithm that allows us to find the shortest path between each room. We chose the Dijkstra algorithm because even though we have to run it n times with n the number of vertices ,its complexity will be lower. We will have O(|V*(E+V*log(V)|) instead of O(|V^3|) and this complexity is smaller because |E| <|V^2|. This algorithm computes the shortest path between one room to each room. We then need to run the algorithm with each room in argument one by one. We will then have what we are looking for : the shortest path between each room.

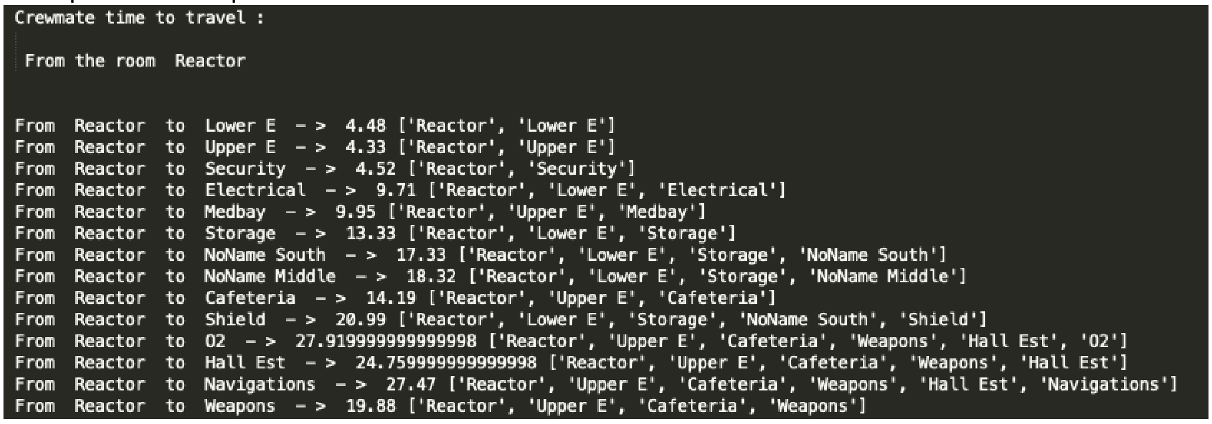
3. Implement the method and show the time to travel for any pair of rooms for both models.

To implement this method, I wrote 2 files representing the 2 graph the node and the vertices present in both graphs. One function called “read” read the graph and add the edges to the python graph structure. We can then apply the Dijkstra function to each. The output of the function is in a text file named TravelDistances.txt in the same repository then the code.

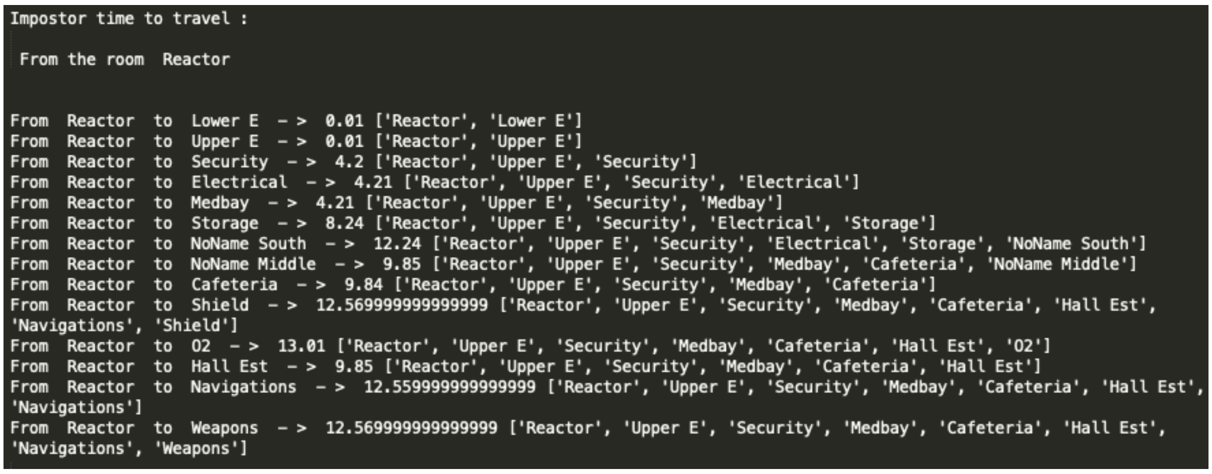
Dijkstra's algorithm :

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:
        distance[v] = infinity
        previous[v] := undefined
    distance[source] = 0
    S = the set of all nodes in Graph
    while Q is not empty:
        u = node in S with smallest distance[ ]
        remove u from S
        for each neighbor v of u:
            alt = distance[u] + distance_between(u, v)
            if alt < distance[v]
                distance[v] = alt
                previous[v] = u
return previous[ ]
```

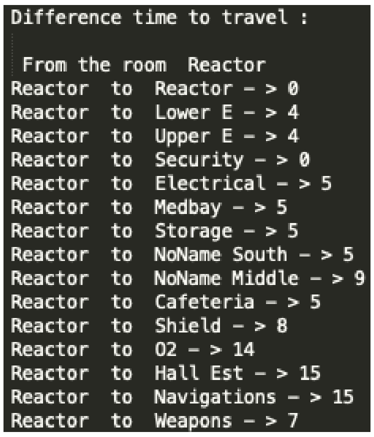
Example of the output file for one room :



```
Crewmate time to travel :

 From the room  Reactor


From  Reactor  to  Lower E  - >  4.48 ['Reactor', 'Lower E']
From  Reactor  to  Upper E  - >  4.33 ['Reactor', 'Upper E']
From  Reactor  to  Security  - >  4.52 ['Reactor', 'Security']
From  Reactor  to  Electrical  - >  9.71 ['Reactor', 'Lower E', 'Electrical']
From  Reactor  to  Medbay  - >  9.95 ['Reactor', 'Upper E', 'Medbay']
From  Reactor  to  Storage  - >  13.33 ['Reactor', 'Lower E', 'Storage']
From  Reactor  to  NoName South  - >  17.33 ['Reactor', 'Lower E', 'Storage', 'NoName South']
From  Reactor  to  NoName Middle  - >  18.32 ['Reactor', 'Lower E', 'Storage', 'NoName Middle']
From  Reactor  to  Cafeteria  - >  14.19 ['Reactor', 'Upper E', 'Cafeteria']
From  Reactor  to  Shield  - >  20.99 ['Reactor', 'Lower E', 'Storage', 'NoName South', 'Shield']
From  Reactor  to  O2  - >  27.919999999999998 ['Reactor', 'Upper E', 'Cafeteria', 'Weapons', 'Hall Est', 'O2']
From  Reactor  to  Hall Est  - >  24.759999999999998 ['Reactor', 'Upper E', 'Cafeteria', 'Weapons', 'Hall Est']
From  Reactor  to  Navigations  - >  27.47 ['Reactor', 'Upper E', 'Cafeteria', 'Weapons', 'Hall Est', 'Navigations']
From  Reactor  to  Weapons  - >  19.88 ['Reactor', 'Upper E', 'Cafeteria', 'Weapons']
```



```
Impostor time to travel :

 From the room  Reactor


From  Reactor  to  Lower E  - >  0.01 ['Reactor', 'Lower E']
From  Reactor  to  Upper E  - >  0.01 ['Reactor', 'Upper E']
From  Reactor  to  Security  - >  4.2 ['Reactor', 'Upper E', 'Security']
From  Reactor  to  Electrical  - >  4.21 ['Reactor', 'Upper E', 'Security', 'Electrical']
From  Reactor  to  Medbay  - >  4.21 ['Reactor', 'Upper E', 'Security', 'Medbay']
From  Reactor  to  Storage  - >  8.24 ['Reactor', 'Upper E', 'Security', 'Electrical', 'Storage']
From  Reactor  to  NoName South  - >  12.24 ['Reactor', 'Upper E', 'Security', 'Electrical', 'Storage', 'NoName South']
From  Reactor  to  NoName Middle  - >  9.85 ['Reactor', 'Upper E', 'Security', 'Medbay', 'Cafeteria', 'NoName Middle']
From  Reactor  to  Cafeteria  - >  9.84 ['Reactor', 'Upper E', 'Security', 'Medbay', 'Cafeteria']
From  Reactor  to  Shield  - >  12.569999999999999 ['Reactor', 'Upper E', 'Security', 'Medbay', 'Cafeteria', 'Hall Est',
'Navigations', 'Shield']
From  Reactor  to  O2  - >  13.01 ['Reactor', 'Upper E', 'Security', 'Medbay', 'Cafeteria', 'Hall Est', 'O2']
From  Reactor  to  Hall Est  - >  9.85 ['Reactor', 'Upper E', 'Security', 'Medbay', 'Cafeteria', 'Hall Est']
From  Reactor  to  Navigations  - >  12.559999999999999 ['Reactor', 'Upper E', 'Security', 'Medbay', 'Cafeteria', 'Hall Est',
'Navigations']
From  Reactor  to  Weapons  - >  12.569999999999999 ['Reactor', 'Upper E', 'Security', 'Medbay', 'Cafeteria', 'Hall Est',
'Navigations', 'Weapons']
```



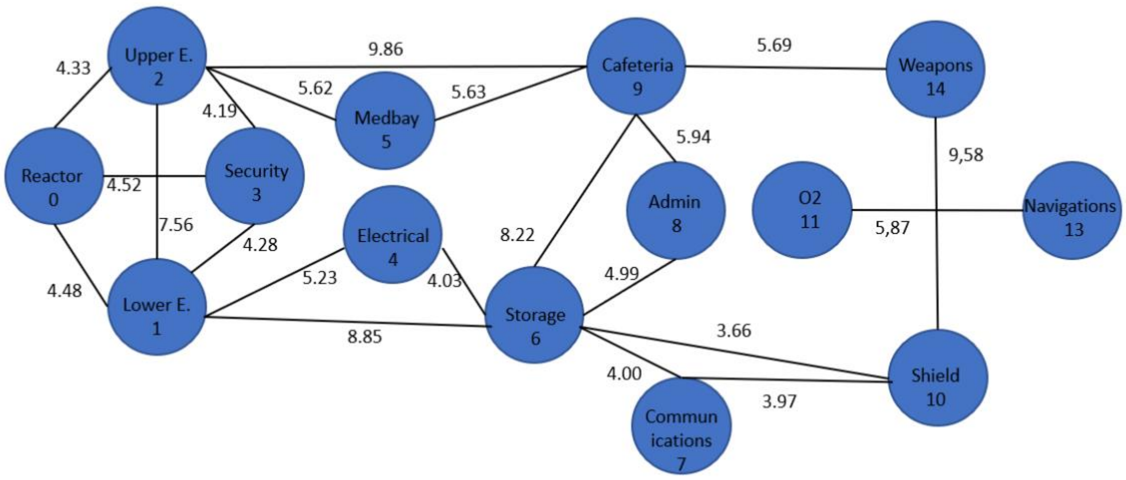
```
Difference time to travel :

 From the room  Reactor
Reactor  to  Reactor - > 0
Reactor  to  Lower E - > 4
Reactor  to  Upper E - > 4
Reactor  to  Security - > 0
Reactor  to  Electrical - > 5
Reactor  to  Medbay - > 5
Reactor  to  Storage - > 5
Reactor  to  NoName South - > 5
Reactor  to  NoName Middle - > 9
Reactor  to  Cafeteria - > 5
Reactor  to  Shield - > 8
Reactor  to  O2 - > 14
Reactor  to  Hall Est - > 15
Reactor  to  Navigations - > 15
Reactor  to  Weapons - > 7
```

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

   To display the interval of time for each pair of room ,I used two list to stock the different time of travel for crewmate and impostor. Then in the function called writeres, we browse both list and calculate the difference between the time of a crewmate and an impostor.

Step 4: To organize the tournament

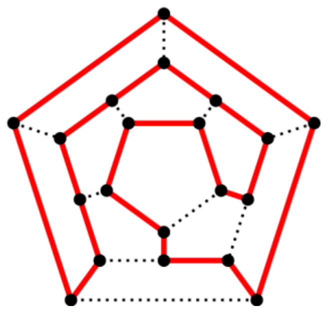
1. Presents and argue about the model of the map.

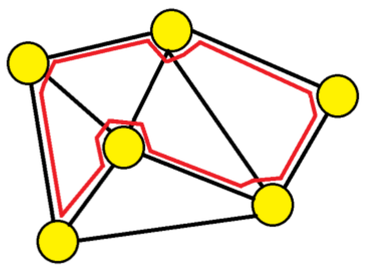


For this part, we needed to represent a graph for the crewmates. So, we based it on the precedent graph, but we choose to delete the "Halle Est" that we represented earlier, because it was useful when we have to compare to the impostor's graph but not anymore.
To answer to the problem, we needed to represent the map in a way that can be both practical and clear. So, we choose to represent it in a text file, with three columns: start, arrival, time. This way, information is easy to read and to use. Thanks to the function "*create_graph*" we store the data into a matrix; thus, it can be easily run through to find a path.

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

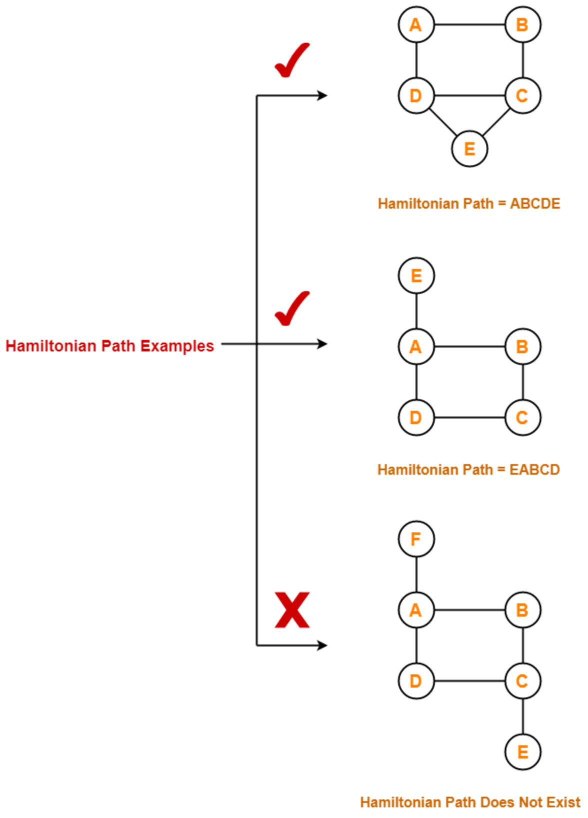
   In the mathematical field of graph theory, a Hamiltonian path is a path in an undirected or directed graph that visits each vertex exactly once. We found out that we can implement what is called a Hamiltonian cycle or a Hamiltonian path to resolve our problem.


Hamiltonian path


Hamiltonian cycle

3. Argue about an algorithm solving your problem.

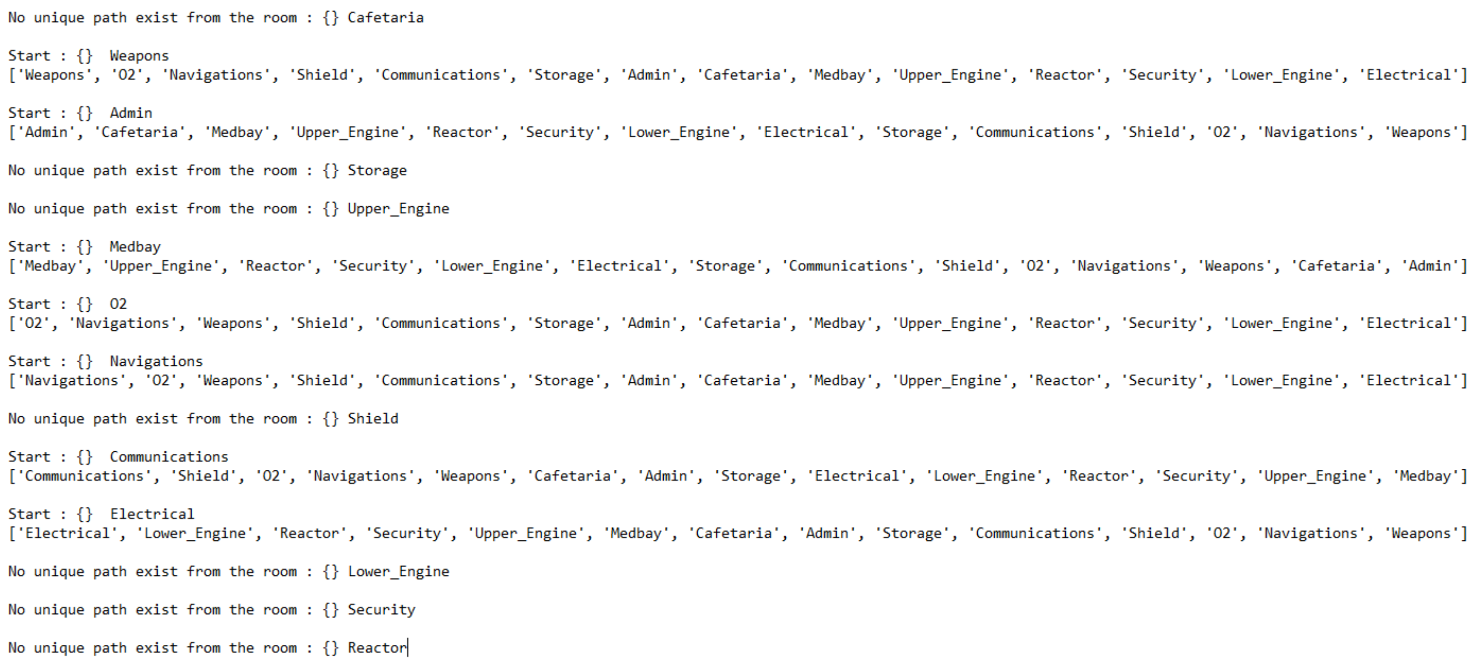
   In order to solve our problem, we needed to make a smart choice between these two possibilities: Hamiltonian cycle and Hamiltonian path. A Hamiltonian cycle is a Hamiltonian path that is a cycle, so it must return to the starting node, in our case we can eliminate this



algorithm because it doesn’t fit with our graph. Finally, determining whether unique paths exist in a graph like ours is the Hamiltonian path problem.

4. Implement the algorithm and show a solution.

   We implemented a function named “*hamilton_path*” that run through our graph and return a unique path which pass by every node once.
   To print the result, we choose to do another function named “step4” which display whether it exists a Hamiltonian path. In order to do that, we used the returned path from the present function and if its length is superior to 1, the path does exist. Then we can display the starting node, and every node passing through, else we choose to display the sentence: “"No unique path exist from the room:” with the name of the room.
   We got the following result:



```
No unique path exist from the room : {} Cafetaria

Start : {}  Weapons
['Weapons', 'O2', 'Navigations', 'Shield', 'Communications', 'Storage', 'Admin', 'Cafetaria', 'Medbay', 'Upper_Engine', 'Reactor', 'Security', 'Lower_Engine', 'Electrical']

Start : {}  Admin
['Admin', 'Cafetaria', 'Medbay', 'Upper_Engine', 'Reactor', 'Security', 'Lower_Engine', 'Electrical', 'Storage', 'Communications', 'Shield', 'O2', 'Navigations', 'Weapons']

No unique path exist from the room : {} Storage

No unique path exist from the room : {} Upper_Engine

Start : {}  Medbay
['Medbay', 'Upper_Engine', 'Reactor', 'Security', 'Lower_Engine', 'Electrical', 'Storage', 'Communications', 'Shield', 'O2', 'Navigations', 'Weapons', 'Cafetaria', 'Admin']

Start : {}  O2
['O2', 'Navigations', 'Weapons', 'Shield', 'Communications', 'Storage', 'Admin', 'Cafetaria', 'Medbay', 'Upper_Engine', 'Reactor', 'Security', 'Lower_Engine', 'Electrical']

Start : {}  Navigations
['Navigations', 'O2', 'Weapons', 'Shield', 'Communications', 'Storage', 'Admin', 'Cafetaria', 'Medbay', 'Upper_Engine', 'Reactor', 'Security', 'Lower_Engine', 'Electrical']

No unique path exist from the room : {} Shield

Start : {}  Communications
['Communications', 'Shield', 'O2', 'Navigations', 'Weapons', 'Cafetaria', 'Admin', 'Storage', 'Electrical', 'Lower_Engine', 'Reactor', 'Security', 'Upper_Engine', 'Medbay']

Start : {}  Electrical
['Electrical', 'Lower_Engine', 'Reactor', 'Security', 'Upper_Engine', 'Medbay', 'Cafetaria', 'Admin', 'Storage', 'Communications', 'Shield', 'O2', 'Navigations', 'Weapons']

No unique path exist from the room : {} Lower_Engine

No unique path exist from the room : {} Security

No unique path exist from the room : {} Reactor
```