Maxime Bourgeois
Julien Fink

Report Project Advanced Data Structures & Algorithms: Among Us

Step 1: Organize the tournament

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

In order to represent a Player and its Score we decided to create a Player class, its attributes are score (int), an id (int) to identify players, a player count (int) to increment at each time a new player is created.

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

First, we thought about a Binary Search Tree for the tournament. This data structure is the most adapted for ranking scores. Thanks to the BST we can find a player easily with its score.

Secondly, we needed a data structure which allows multiple equal score values. That's why we decided to create a Node class. This Node class is composed of a score, a left node, a right node, a Player list. This is the basic form of nodes, but we added a Player list which in case there are players with equal score values.

Finally, we want the most optimized data structure. That's why we chose to build an AVL Tree. This is like BS Tree but more optimised.

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

The easier way to do this was to create a method in the Player class. This method just set the Player score at (random.Next(0, 13) + this.score) / 2.

The new score is the mean between the last score and the new game score.

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

To update Player scores and the database we thought about an iterative method. This one browses all the Game table formed previously based on database. At each iteration, we change the score of the player and put it in a new tournament object. At the end, we replace the old tournament by the new fulfil tournament.

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

To create random games based on the database, we thought about a recursive method. This one browses all Nodes in the database by a pre-order traversal. A Node object has his own score value, so when we're changing the score of a player, we don't change the score of the node which avoid browsing problems. At each "iteration", we browse the Player list, foreach of them, we set a random number which correspond to a game index. After checking if the games[index] contains less player than 10 and that the Game tab is not full (all games contain 10 players), we add the player within this game. Otherwise, if games[index] contains already 10 players, we change index while the player is added. Finally, the method returns a Game table which contains all players divided in different games. Game class is a util class. This one contains a list of players.

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

The recursive method is the following. We browse the database by an in-order traversal. First, call to node.right and finally call to node.left. This allows to browse the database by a decreasing order. Between those to call we can perform our actions. Foreach node, we browse the player list, and put those elements into games. The index to set the player is get by a util function.

Finally the function returns the Game table.

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

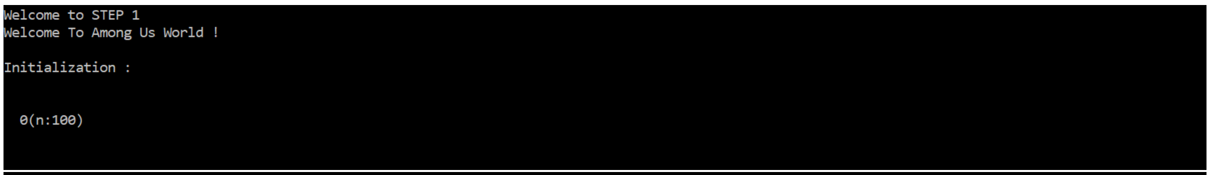
For dropping the 10 last Players we opted for a while loop which are waiting for number of deleted elements is 10. At each loop, we find the lower score (go successively at node.left) and delete this element. This method returns nothing.

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

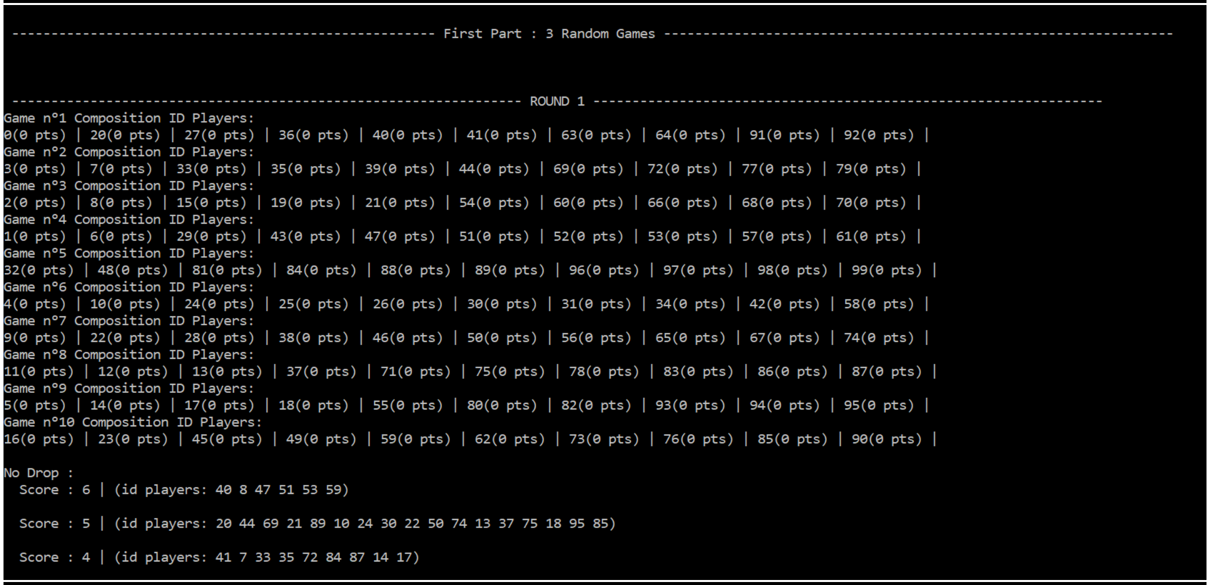
At the end, there are 10 players in the tournament. So, by an in – order traversal we print all players. They are in a decreasing order.

For the podium displayer, we get the game tab and display just the three first players.

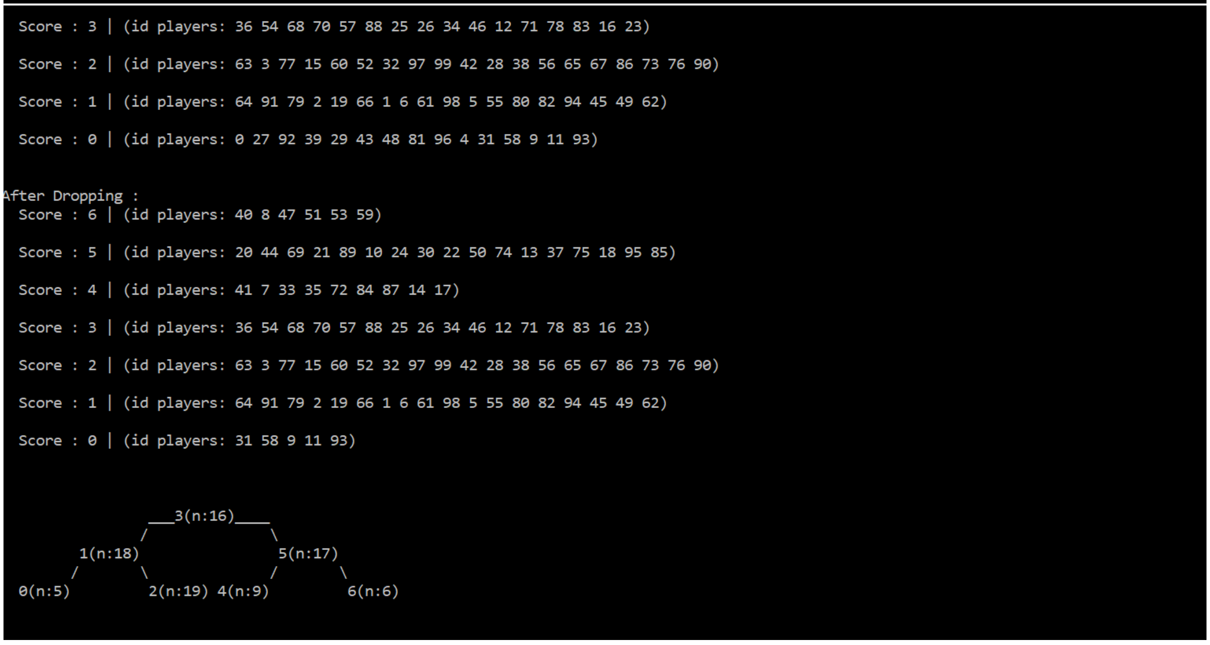
Final results: We built a Main method wich allow to play and print all games/rounds. Let's see what happen in console.



```
Welcome to STEP 1
Welcome To Among Us World !

Initialization :


  0(n:100)
```

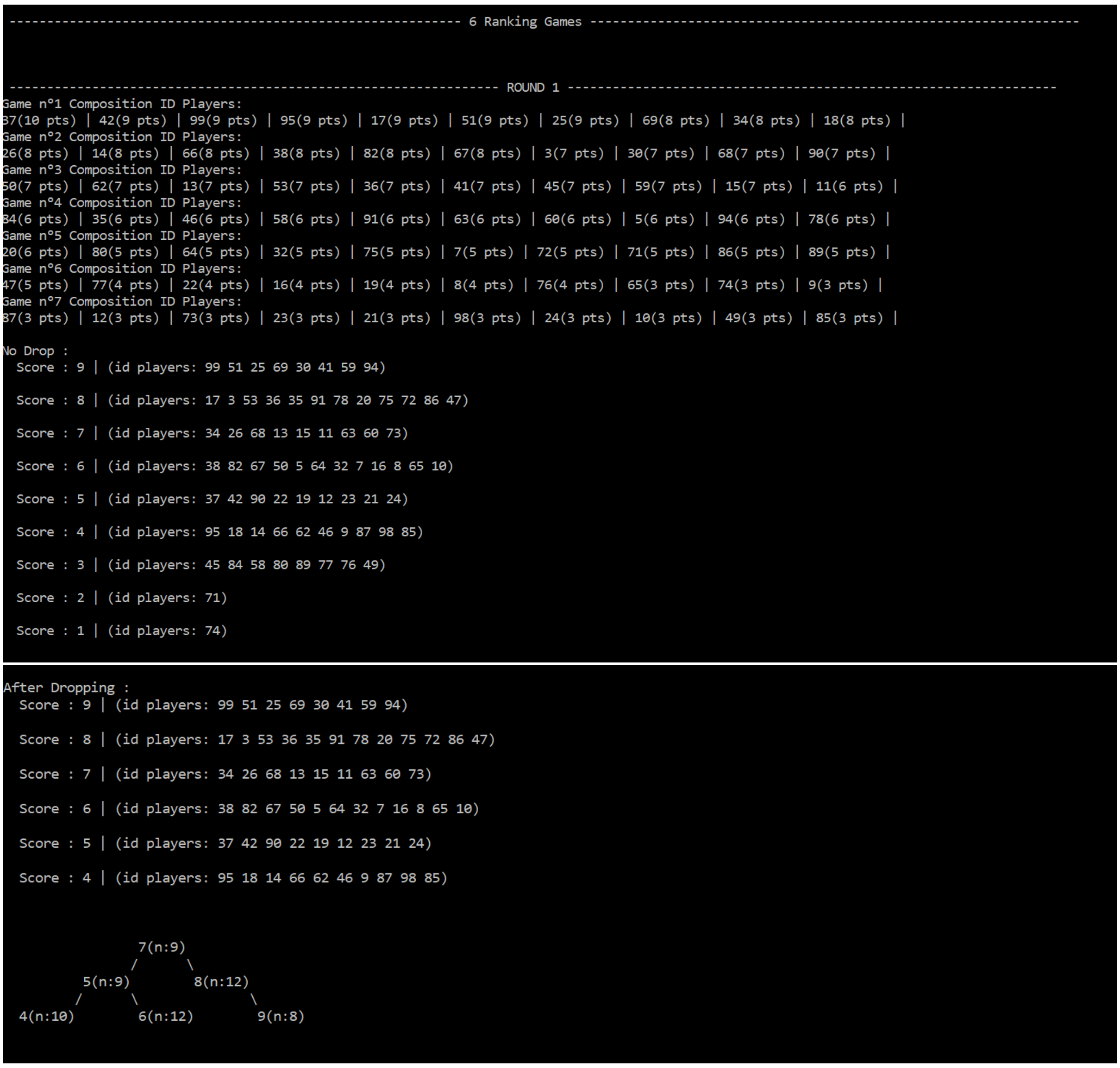


```
 --------------------------------------------------- First Part : 3 Random Games -------------------------------------------------------------



 ---------------------------------------------------------------- ROUND 1 ----------------------------------------------------------------
Game n°1 Composition ID Players:
0(0 pts) | 20(0 pts) | 27(0 pts) | 36(0 pts) | 40(0 pts) | 41(0 pts) | 63(0 pts) | 64(0 pts) | 91(0 pts) | 92(0 pts) |
Game n°2 Composition ID Players:
3(0 pts) | 7(0 pts) | 33(0 pts) | 35(0 pts) | 39(0 pts) | 44(0 pts) | 69(0 pts) | 72(0 pts) | 77(0 pts) | 79(0 pts) |
Game n°3 Composition ID Players:
2(0 pts) | 8(0 pts) | 15(0 pts) | 19(0 pts) | 21(0 pts) | 54(0 pts) | 60(0 pts) | 66(0 pts) | 68(0 pts) | 70(0 pts) |
Game n°4 Composition ID Players:
1(0 pts) | 6(0 pts) | 29(0 pts) | 43(0 pts) | 47(0 pts) | 51(0 pts) | 52(0 pts) | 53(0 pts) | 57(0 pts) | 61(0 pts) |
Game n°5 Composition ID Players:
32(0 pts) | 48(0 pts) | 81(0 pts) | 84(0 pts) | 88(0 pts) | 89(0 pts) | 96(0 pts) | 97(0 pts) | 98(0 pts) | 99(0 pts) |
Game n°6 Composition ID Players:
4(0 pts) | 10(0 pts) | 24(0 pts) | 25(0 pts) | 26(0 pts) | 30(0 pts) | 31(0 pts) | 34(0 pts) | 42(0 pts) | 58(0 pts) |
Game n°7 Composition ID Players:
9(0 pts) | 22(0 pts) | 28(0 pts) | 38(0 pts) | 46(0 pts) | 50(0 pts) | 56(0 pts) | 65(0 pts) | 67(0 pts) | 74(0 pts) |
Game n°8 Composition ID Players:
11(0 pts) | 12(0 pts) | 13(0 pts) | 37(0 pts) | 71(0 pts) | 75(0 pts) | 78(0 pts) | 83(0 pts) | 86(0 pts) | 87(0 pts) |
Game n°9 Composition ID Players:
5(0 pts) | 14(0 pts) | 17(0 pts) | 18(0 pts) | 55(0 pts) | 80(0 pts) | 82(0 pts) | 93(0 pts) | 94(0 pts) | 95(0 pts) |
Game n°10 Composition ID Players:
16(0 pts) | 23(0 pts) | 45(0 pts) | 49(0 pts) | 59(0 pts) | 62(0 pts) | 73(0 pts) | 76(0 pts) | 85(0 pts) | 90(0 pts) |

No Drop :
  Score : 6 | (id players: 40 8 47 51 53 59)

  Score : 5 | (id players: 20 44 69 21 89 10 24 30 22 50 74 13 37 75 18 95 85)

  Score : 4 | (id players: 41 7 33 35 72 84 87 14 17)
```



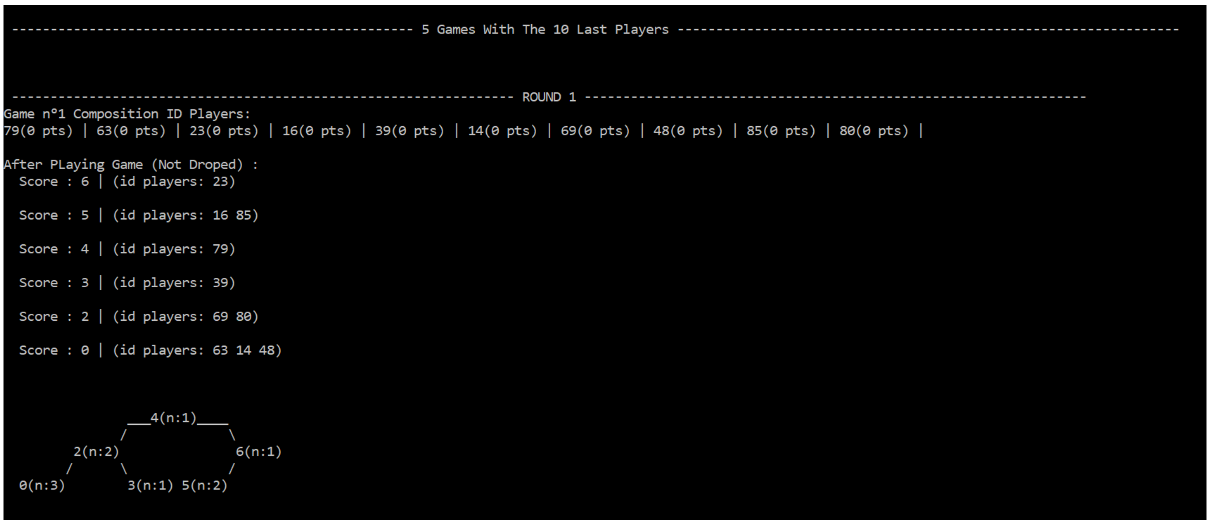
```
  Score : 3 | (id players: 36 54 68 70 57 88 25 26 34 46 12 71 78 83 16 23)

  Score : 2 | (id players: 63 3 77 15 60 52 32 97 99 42 28 38 56 65 67 86 73 76 90)

  Score : 1 | (id players: 64 91 79 2 19 66 1 6 61 98 5 55 80 82 94 45 49 62)

  Score : 0 | (id players: 0 27 92 39 29 43 48 81 96 4 31 58 9 11 93)


After Dropping :
  Score : 6 | (id players: 40 8 47 51 53 59)

  Score : 5 | (id players: 20 44 69 21 89 10 24 30 22 50 74 13 37 75 18 95 85)

  Score : 4 | (id players: 41 7 33 35 72 84 87 14 17)

  Score : 3 | (id players: 36 54 68 70 57 88 25 26 34 46 12 71 78 83 16 23)

  Score : 2 | (id players: 63 3 77 15 60 52 32 97 99 42 28 38 56 65 67 86 73 76 90)

  Score : 1 | (id players: 64 91 79 2 19 66 1 6 61 98 5 55 80 82 94 45 49 62)

  Score : 0 | (id players: 31 58 9 11 93)



                  ___3(n:16)____
                 /              \
         1(n:18)                 5(n:17)
        /        \              /       \
  0(n:5)          2(n:19) 4(n:9)         6(n:6)
```

… 3 times (Random games)



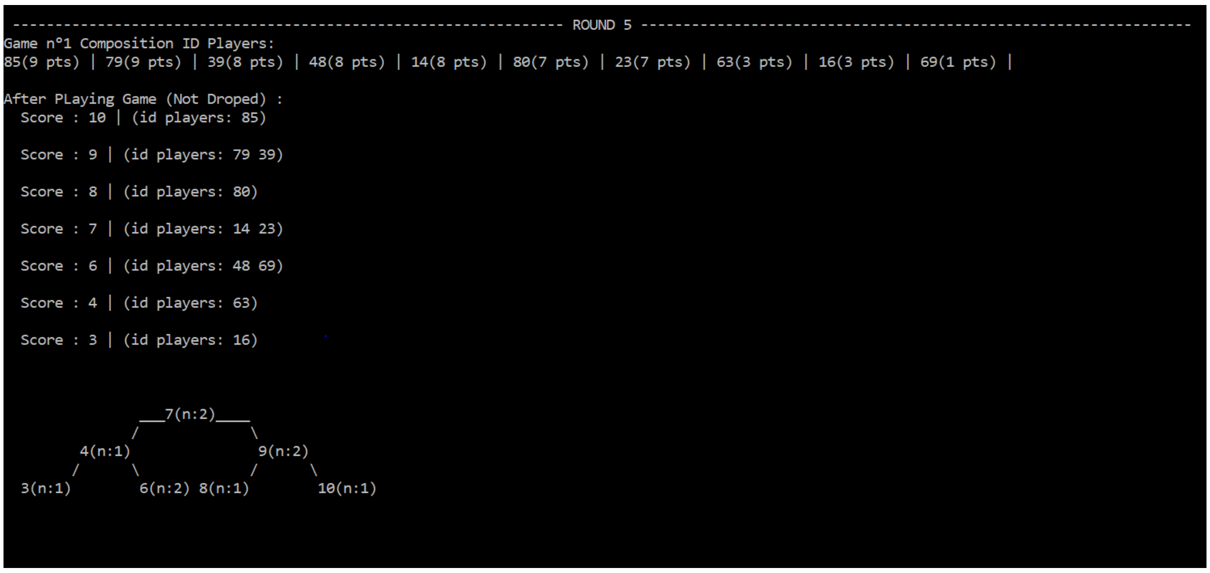
```
----------------------------------------------------------- 6 Ranking Games ----------------------------------------------------------------



 ---------------------------------------------------------------- ROUND 1 --------------------------------------------------------------
Game n°1 Composition ID Players:
37(10 pts) | 42(9 pts) | 99(9 pts) | 95(9 pts) | 17(9 pts) | 51(9 pts) | 25(9 pts) | 69(8 pts) | 34(8 pts) | 18(8 pts) |
Game n°2 Composition ID Players:
26(8 pts) | 14(8 pts) | 66(8 pts) | 38(8 pts) | 82(8 pts) | 67(8 pts) | 3(7 pts) | 30(7 pts) | 68(7 pts) | 90(7 pts) |
Game n°3 Composition ID Players:
50(7 pts) | 62(7 pts) | 13(7 pts) | 53(7 pts) | 36(7 pts) | 41(7 pts) | 45(7 pts) | 59(7 pts) | 15(7 pts) | 11(6 pts) |
Game n°4 Composition ID Players:
34(6 pts) | 35(6 pts) | 46(6 pts) | 58(6 pts) | 91(6 pts) | 63(6 pts) | 60(6 pts) | 5(6 pts) | 94(6 pts) | 78(6 pts) |
Game n°5 Composition ID Players:
20(6 pts) | 80(5 pts) | 64(5 pts) | 32(5 pts) | 75(5 pts) | 7(5 pts) | 72(5 pts) | 71(5 pts) | 86(5 pts) | 89(5 pts) |
Game n°6 Composition ID Players:
47(5 pts) | 77(4 pts) | 22(4 pts) | 16(4 pts) | 19(4 pts) | 8(4 pts) | 76(4 pts) | 65(3 pts) | 74(3 pts) | 9(3 pts) |
Game n°7 Composition ID Players:
37(3 pts) | 12(3 pts) | 73(3 pts) | 23(3 pts) | 21(3 pts) | 98(3 pts) | 24(3 pts) | 10(3 pts) | 49(3 pts) | 85(3 pts) |

No Drop :
  Score : 9 | (id players: 99 51 25 69 30 41 59 94)

  Score : 8 | (id players: 17 3 53 36 35 91 78 20 75 72 86 47)

  Score : 7 | (id players: 34 26 68 13 15 11 63 60 73)

  Score : 6 | (id players: 38 82 67 50 5 64 32 7 16 8 65 10)

  Score : 5 | (id players: 37 42 90 22 19 12 23 21 24)

  Score : 4 | (id players: 95 18 14 66 62 46 9 87 98 85)

  Score : 3 | (id players: 45 84 58 80 89 77 76 49)

  Score : 2 | (id players: 71)

  Score : 1 | (id players: 74)

After Dropping :
  Score : 9 | (id players: 99 51 25 69 30 41 59 94)

  Score : 8 | (id players: 17 3 53 36 35 91 78 20 75 72 86 47)

  Score : 7 | (id players: 34 26 68 13 15 11 63 60 73)

  Score : 6 | (id players: 38 82 67 50 5 64 32 7 16 8 65 10)

  Score : 5 | (id players: 37 42 90 22 19 12 23 21 24)

  Score : 4 | (id players: 95 18 14 66 62 46 9 87 98 85)


                  7(n:9)
                 /      \
          5(n:9)         8(n:12)
         /      \               \
  4(n:10)        6(n:12)         9(n:8)
```

… 6 times (Ranking Games)

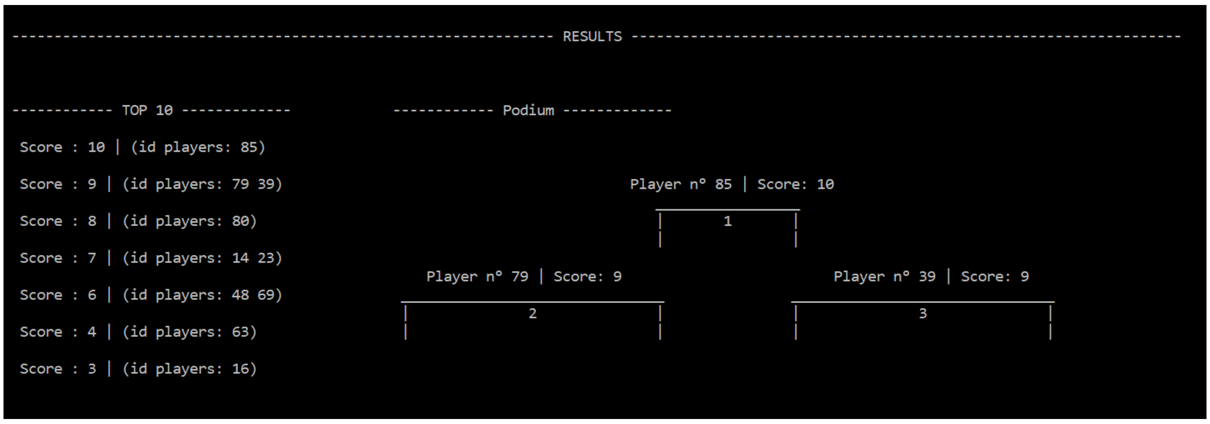


```
 ------------------------------------------------- 5 Games With The 10 Last Players ----------------------------------------------------------------


 --------------------------------------------------------------- ROUND 1 --------------------------------------------------------------
Game n°1 Composition ID Players:
79(0 pts) | 63(0 pts) | 23(0 pts) | 16(0 pts) | 39(0 pts) | 14(0 pts) | 69(0 pts) | 48(0 pts) | 85(0 pts) | 80(0 pts) |

After PLaying Game (Not Droped) :
  Score : 6 | (id players: 23)

  Score : 5 | (id players: 16 85)

  Score : 4 | (id players: 79)

  Score : 3 | (id players: 39)

  Score : 2 | (id players: 69 80)

  Score : 0 | (id players: 63 14 48)



                ___4(n:1)____
               /             \
        2(n:2)               6(n:1)
       /      \             /
  0(n:3)       3(n:1) 5(n:2)
```

… 5 times (Games with 10 last)



```
---------------------------------------------------------------- ROUND 5 ----------------------------------------------------------------
Game n°1 Composition ID Players:
85(9 pts) | 79(9 pts) | 39(8 pts) | 48(8 pts) | 14(8 pts) | 80(7 pts) | 23(7 pts) | 63(3 pts) | 16(3 pts) | 69(1 pts) |

After PLaying Game (Not Droped) :
  Score : 10 | (id players: 85)

  Score : 9 | (id players: 79 39)

  Score : 8 | (id players: 80)

  Score : 7 | (id players: 14 23)

  Score : 6 | (id players: 48 69)

  Score : 4 | (id players: 63)

  Score : 3 | (id players: 16)



                ___7(n:2)____
               /              \
         4(n:1)                9(n:2)
        /      \              /      \
  3(n:1)        6(n:2) 8(n:1)        10(n:1)
```



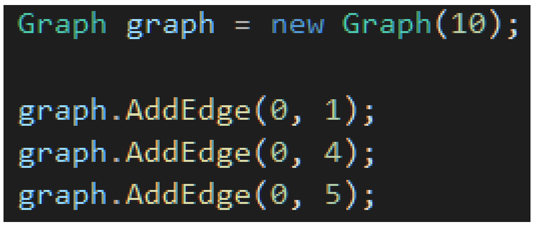
```
--------------------------------------------------------------- RESULTS ---------------------------------------------------------------


------------ TOP 10 -------------             ------------ Podium -------------

 Score : 10 | (id players: 85)

 Score : 9 | (id players: 79 39)                           Player n° 85 | Score: 10
                                                              ________________
 Score : 8 | (id players: 80)                                |        1       |
                                                             |                |
 Score : 7 | (id players: 14 23)
                                         Player n° 79 | Score: 9                   Player n° 39 | Score: 9
 Score : 6 | (id players: 48 69)       ______________________________           ______________________________
                                      |              2               |         |              3               |
 Score : 4 | (id players: 63)         |                              |         |                              |

 Score : 3 | (id players: 16)
```

Step 2: Professor Layton < Guybrush Threepwood < You

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

For this second part, Maxime and I decided to stock players (and the players seen by a player) in a `LinkedList<int>[]`. We created a class Graph to do that, featuring 3 functions : `public void set_of_impostors()`, `public void AddEdge(int u, int v)` and `public void PrintAdjanceyList()`.

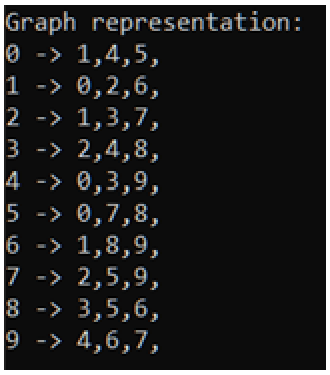
AddEdge allows us to add a Player to the datastructure and Players that have been seen by the specific Player :
The following example add the first Player (among 9 players ; 10 – 1) that has seen players 1, 4 and 5.



```
Graph graph = new Graph(10);

graph.AddEdge(0, 1);
graph.AddEdge(0, 4);
graph.AddEdge(0, 5);
```

The function **`PrintAdjanceyList`** displays all Players and their seen-relation :

"Player 6 has seen Player 1, 8 and 9"



```
Graph representation:
0 -> 1,4,5,
1 -> 0,2,6,
2 -> 1,3,7,
3 -> 2,4,8,
4 -> 0,3,9,
5 -> 0,7,8,
6 -> 1,8,9,
7 -> 2,5,9,
8 -> 3,5,6,
9 -> 4,6,7,
```

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

We know that Player 0 has been found dead. Player 0 has seen players 1, 4 and 5 before dying and we know that impostors rarely see each other in a game, since they don't want to arouse suspicions. If we consider that the second impostor didn't see player 1, 4 or 5, we can define a probable set of impostors :

If Player 2 has seen 1, 3 and 7, we can therefore say that 2 probable se of impostors are (2,4) and (2,5).

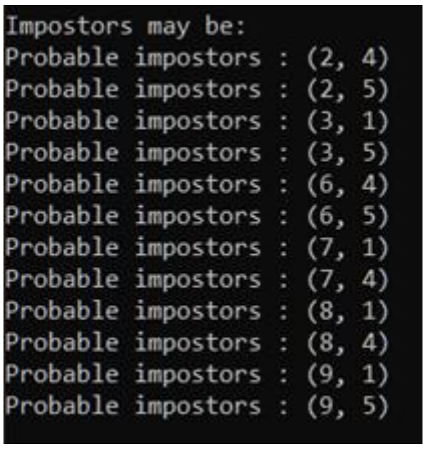
3. **Argue about an algorithm solving your problem.**

To solve our problem, we have thought about the following algorithm :

- We browse the created datastructure in **1.**
- If we are currently dealing with Player 1, 4 or 5, we go to the next iteration
- If not, we try to see if player 1, 4 or 5 has been seen by the iterated Player.
- If that's the case, then the seen Player and the iterated one can't be impostors, so there are only 2 cases left...

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

The function **`set_of_impostors()`** allows us to define a set of impostors knowing that Player 0 was killed.

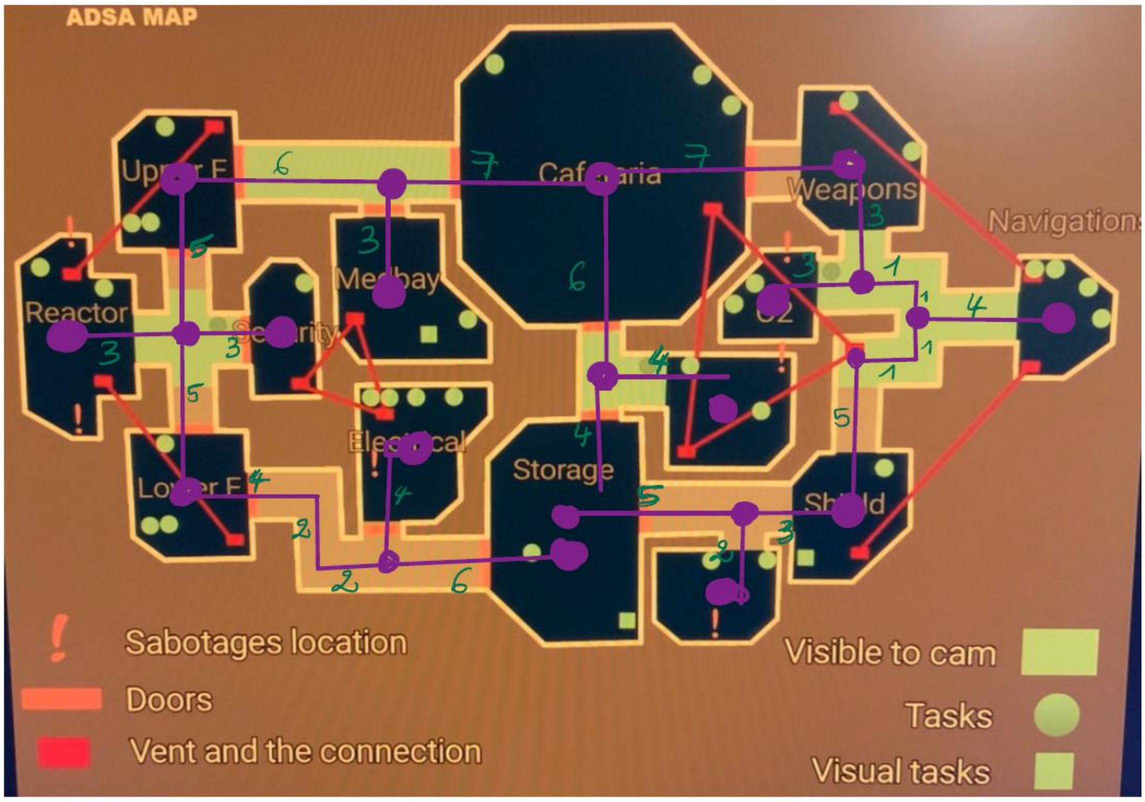


```
Impostors may be:
Probable impostors : (2, 4)
Probable impostors : (2, 5)
Probable impostors : (3, 1)
Probable impostors : (3, 5)
Probable impostors : (6, 4)
Probable impostors : (6, 5)
Probable impostors : (7, 1)
Probable impostors : (7, 4)
Probable impostors : (8, 1)
Probable impostors : (8, 4)
Probable impostors : (9, 1)
Probable impostors : (9, 5)
```

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

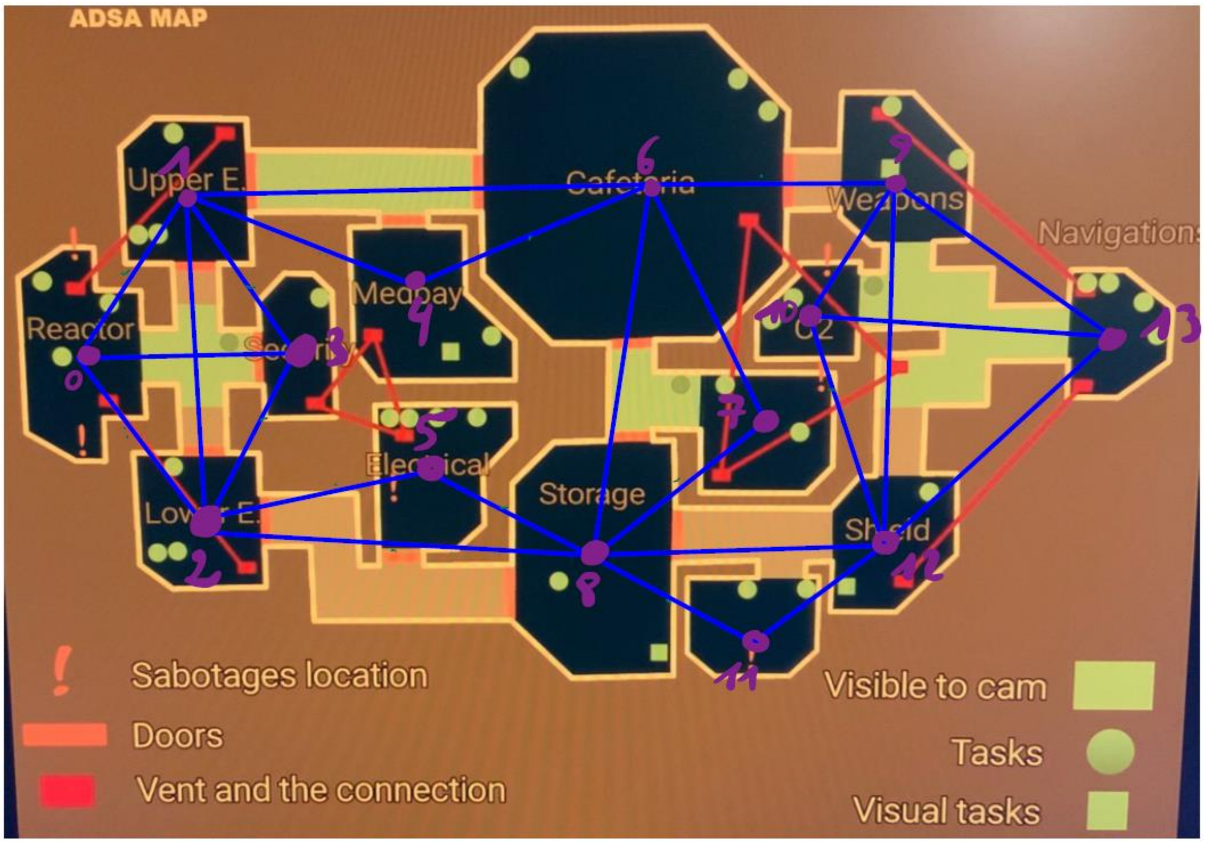
a) Get the distances

We get the distances (in cm) by measuring with a rule distances on the map. The result was the following :

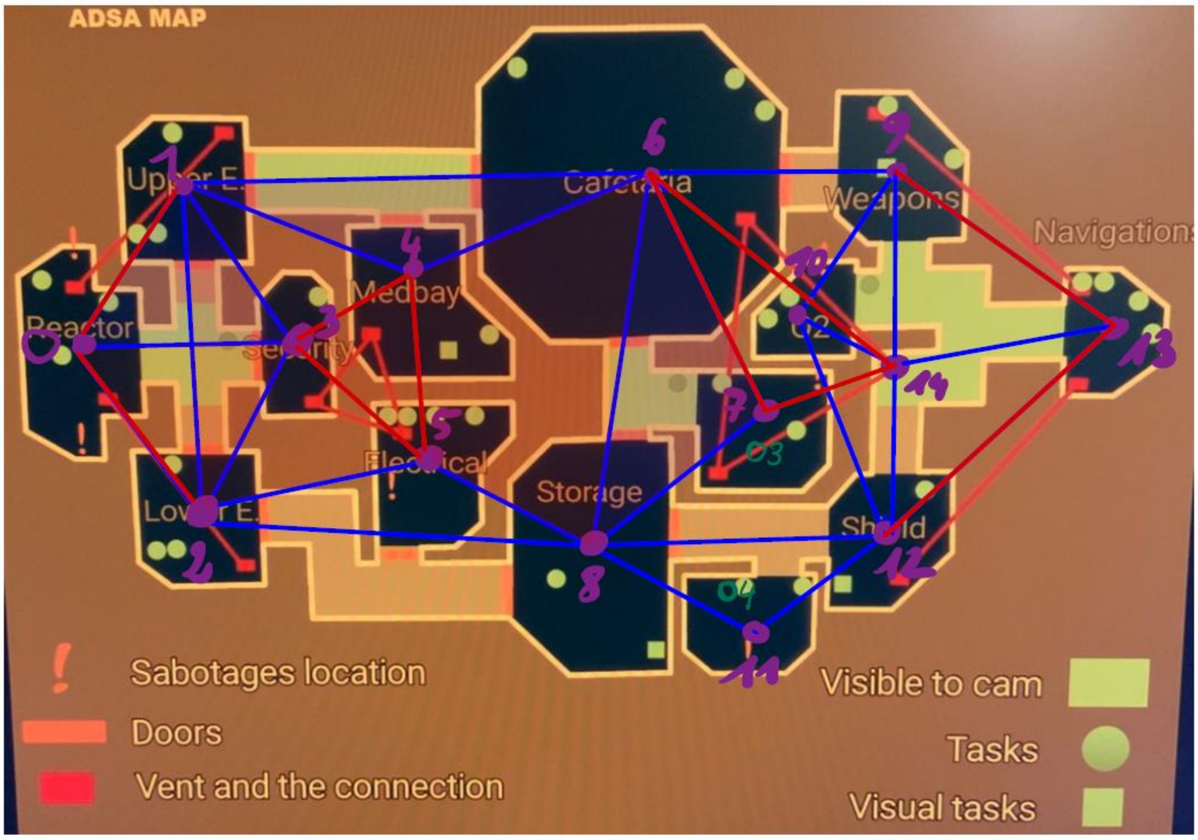


b) Design graphs for Crewmate and Impostor

Graph for Crewmate (without vent). Node values are 0 to 13.

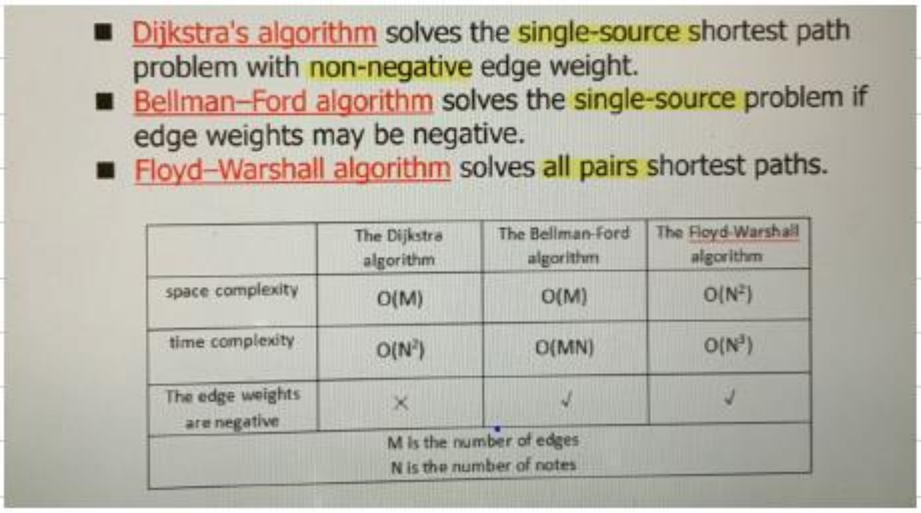


Graph for Impostor (with vents). Also we added the Node n°14 because a vent door is in a corridor.



c) What algorithm are we going to choose for pathfinding problem?

According to the subject, we need to show the time to travel for any pair of rooms for both models.
The course gives us :



- Dijkstra's algorithm solves the single-source shortest path problem with non-negative edge weight.
- Bellman–Ford algorithm solves the single-source problem if edge weights may be negative.
- Floyd–Warshall algorithm solves all pairs shortest paths.

	The Dijkstra algorithm	The Bellman-Ford algorithm	The Floyd-Warshall algorithm
space complexity	O(M)	O(M)	O(N^2)
time complexity	O(N^2)	O(MN)	O(N^3)
The edge weights are negative	×	✓	✓
M is the number of edges N is the number of notes			

Only one can solves all pairs shortest path : **The Floyd-Warhall Algorithm**.

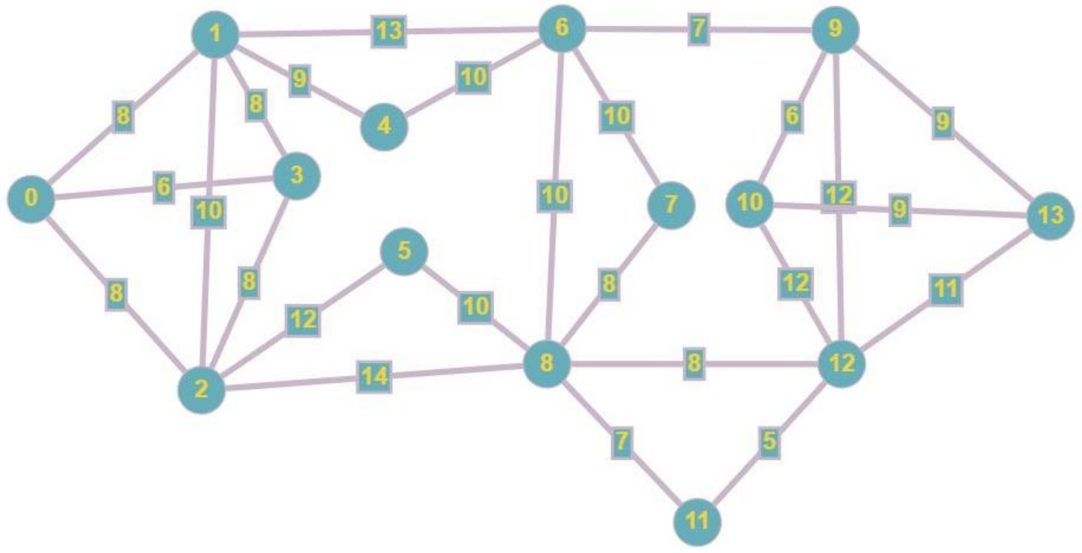
That’s why we will implement this one.

d) build the Adjency Matrix (INPUT ALGO)

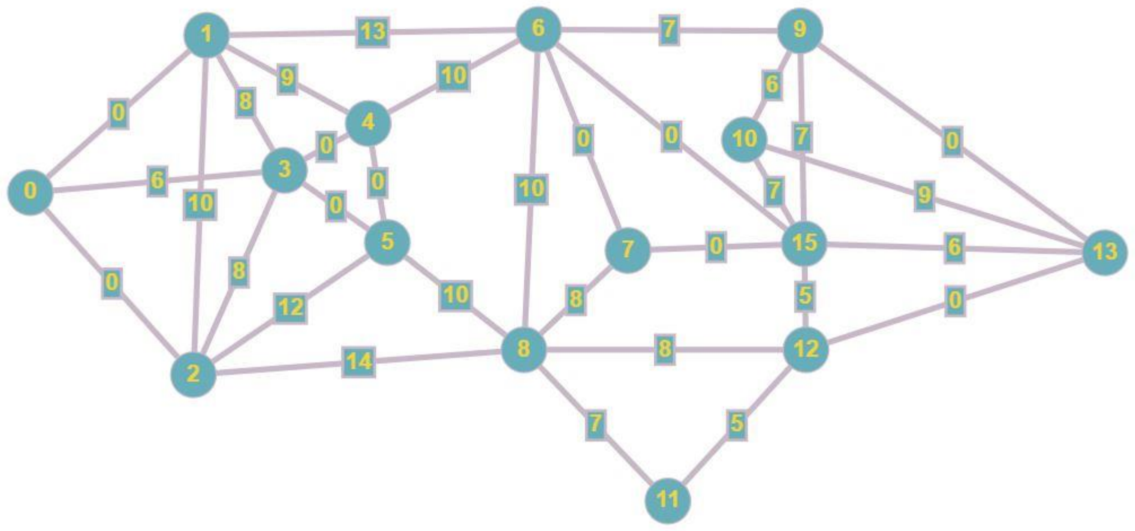
In order to gain times, we used an online graph visualizer which gives an adjacency matrix.

https://graphonline.ru/en/

Graph for Crewmate :

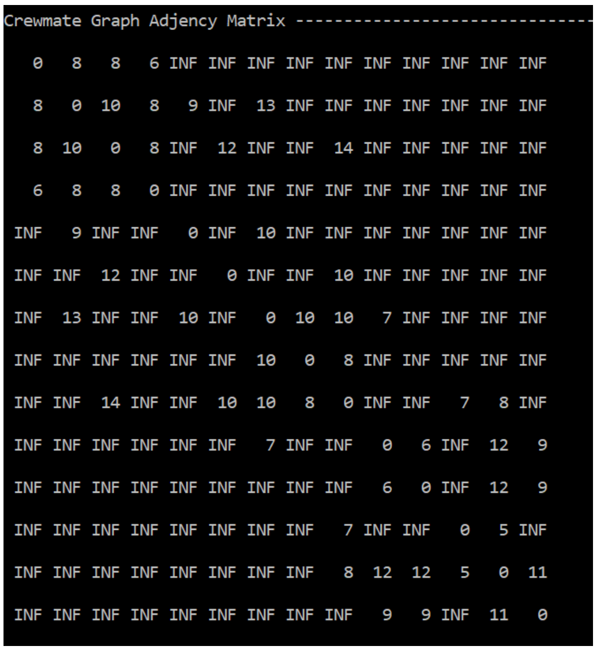


Graph for impostor :

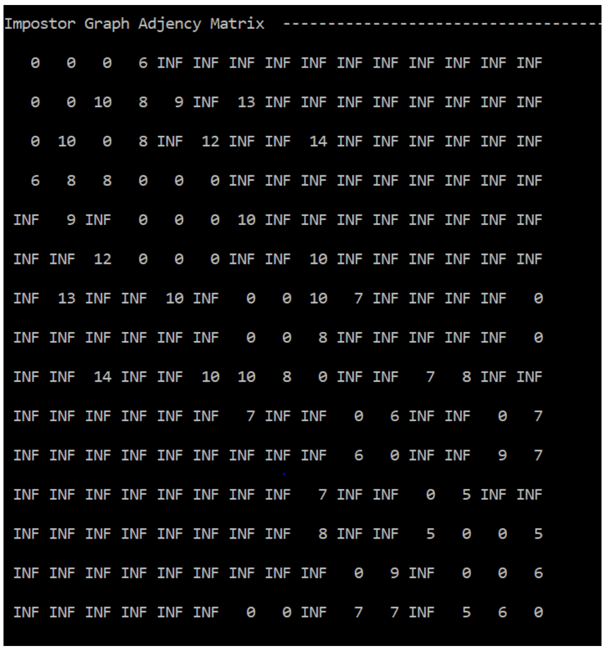


The adjacency matrix given by the website is not yet usable. Why ? if there is no link between two nodes, a 0 is set in the matrix, however, we want an infinite value there. By coding a C# program, we changed those zero values by infinite values (10 000 = INF).

Matrix are all the time read/saved in a .txt file under the folder :
.\ADSA_Project_AmongUs\ADSA_Project_AmongUs\bin\Debug\netcoreapp3.1



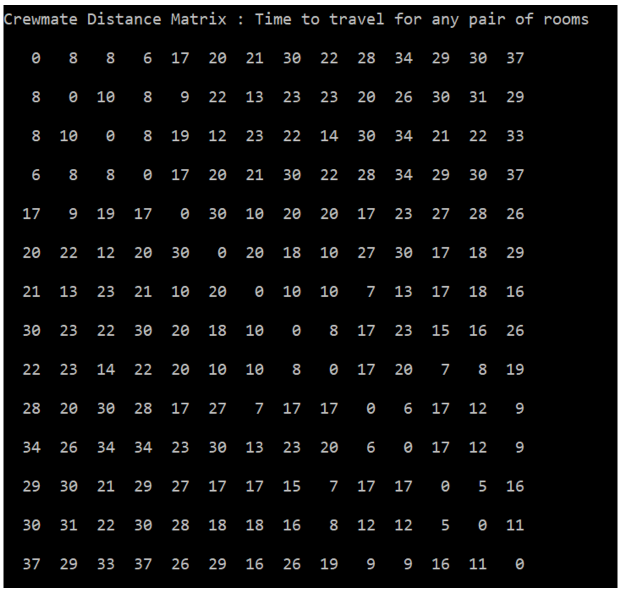
```
Crewmate Graph Adjency Matrix ------------------------------

   0   8   8   6 INF INF INF INF INF INF INF INF INF INF

   8   0  10   8   9 INF  13 INF INF INF INF INF INF INF

   8  10   0   8 INF  12 INF INF  14 INF INF INF INF INF

   6   8   8   0 INF INF INF INF INF INF INF INF INF INF

 INF   9 INF INF   0 INF  10 INF INF INF INF INF INF INF

 INF INF  12 INF INF   0 INF INF  10 INF INF INF INF INF

 INF  13 INF INF  10 INF   0  10  10   7 INF INF INF INF

 INF INF INF INF INF INF  10   0   8 INF INF INF INF INF

 INF INF  14 INF INF  10  10   8   0 INF INF   7   8 INF

 INF INF INF INF INF INF   7 INF INF   0   6 INF  12   9

 INF INF INF INF INF INF INF INF INF   6   0 INF  12   9

 INF INF INF INF INF INF INF INF   7 INF INF   0   5 INF

 INF INF INF INF INF INF INF INF   8  12  12   5   0  11

 INF INF INF INF INF INF INF INF INF   9   9 INF  11   0
```



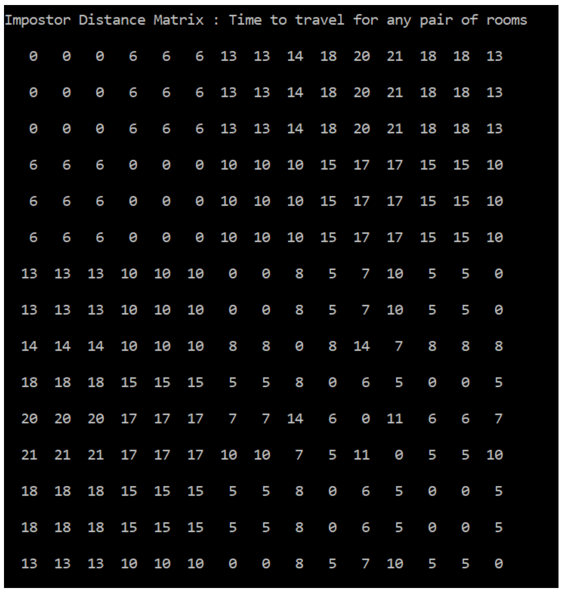
```
Impostor Graph Adjency Matrix  ----------------------------------

   0   0   0   6 INF INF INF INF INF INF INF INF INF INF INF

   0   0  10   8   9 INF  13 INF INF INF INF INF INF INF INF

   0  10   0   8 INF  12 INF INF  14 INF INF INF INF INF INF

   6   8   8   0   0   0 INF INF INF INF INF INF INF INF INF

 INF   9 INF   0   0   0  10 INF INF INF INF INF INF INF INF

 INF INF  12   0   0   0 INF INF  10 INF INF INF INF INF INF

 INF  13 INF INF  10 INF   0   0  10   7 INF INF INF INF   0

 INF INF INF INF INF INF   0   0   8 INF INF INF INF INF   0

 INF INF  14 INF INF  10  10   8   0 INF INF   7   8 INF INF

 INF INF INF INF INF INF   7 INF INF   0   6 INF INF   0   7

 INF INF INF INF INF INF INF INF INF   6   0 INF INF   9   7

 INF INF INF INF INF INF INF INF   7 INF INF   0   5 INF INF

 INF INF INF INF INF INF INF INF   8 INF INF   5   0   0   5

 INF INF INF INF INF INF INF INF INF   0   9 INF   0   0   6

 INF INF INF INF INF INF   0   0 INF   7   7 INF   5   6   0
```

e) implement the **The Floyd-Warhall Algorithm** (OUTPUT ALGO)

The followings matrix give the distances between all pairs of room for a Crewmate or an impostor. This is the output of **Floyd-Warhall Algorithm.**



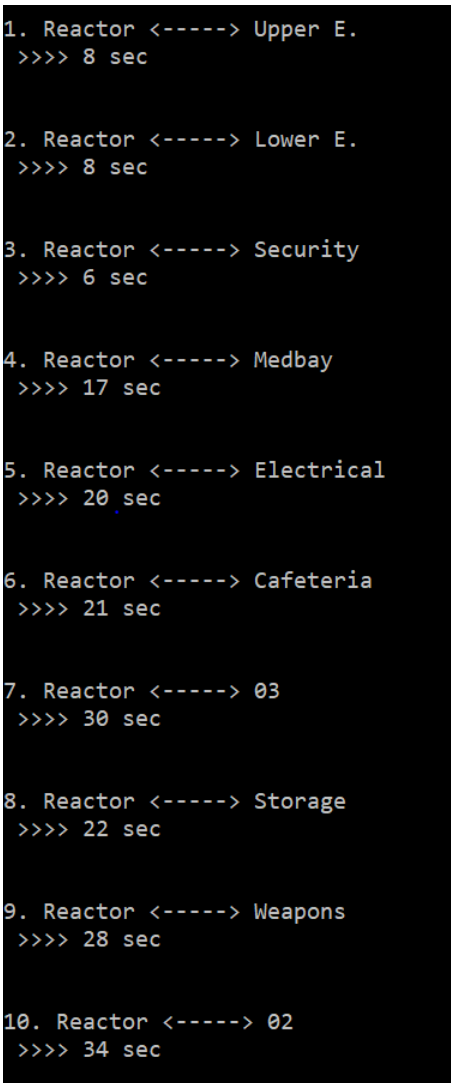
```
Crewmate Distance Matrix : Time to travel for any pair of rooms

   0   8   8   6  17  20  21  30  22  28  34  29  30  37

   8   0  10   8   9  22  13  23  23  20  26  30  31  29

   8  10   0   8  19  12  23  22  14  30  34  21  22  33

   6   8   8   0  17  20  21  30  22  28  34  29  30  37

  17   9  19  17   0  30  10  20  20  17  23  27  28  26

  20  22  12  20  30   0  20  18  10  27  30  17  18  29

  21  13  23  21  10  20   0  10  10   7  13  17  18  16

  30  23  22  30  20  18  10   0   8  17  23  15  16  26

  22  23  14  22  20  10  10   8   0  17  20   7   8  19

  28  20  30  28  17  27   7  17  17   0   6  17  12   9

  34  26  34  34  23  30  13  23  20   6   0  17  12   9

  29  30  21  29  27  17  17  15   7  17  17   0   5  16

  30  31  22  30  28  18  18  16   8  12  12   5   0  11

  37  29  33  37  26  29  16  26  19   9   9  16  11   0
```



```
Impostor Distance Matrix : Time to travel for any pair of rooms

   0   0   0   6   6   6  13  13  14  18  20  21  18  18  13

   0   0   0   6   6   6  13  13  14  18  20  21  18  18  13

   0   0   0   6   6   6  13  13  14  18  20  21  18  18  13

   6   6   6   0   0   0  10  10  10  15  17  17  15  15  10

   6   6   6   0   0   0  10  10  10  15  17  17  15  15  10

   6   6   6   0   0   0  10  10  10  15  17  17  15  15  10

  13  13  13  10  10  10   0   0   8   5   7  10   5   5   0

  13  13  13  10  10  10   0   0   8   5   7  10   5   5   0

  14  14  14  10  10  10   8   8   0   8  14   7   8   8   8

  18  18  18  15  15  15   5   5   8   0   6   5   0   0   5

  20  20  20  17  17  17   7   7  14   6   0  11   6   6   7

  21  21  21  17  17  17  10  10   7   5  11   0   5   5  10

  18  18  18  15  15  15   5   5   8   0   6   5   0   0   5

  18  18  18  15  15  15   5   5   8   0   6   5   0   0   5

  13  13  13  10  10  10   0   0   8   5   7  10   5   5   0
```

For more understanding, we coded a printing function which gives time for all pairs of rooms :

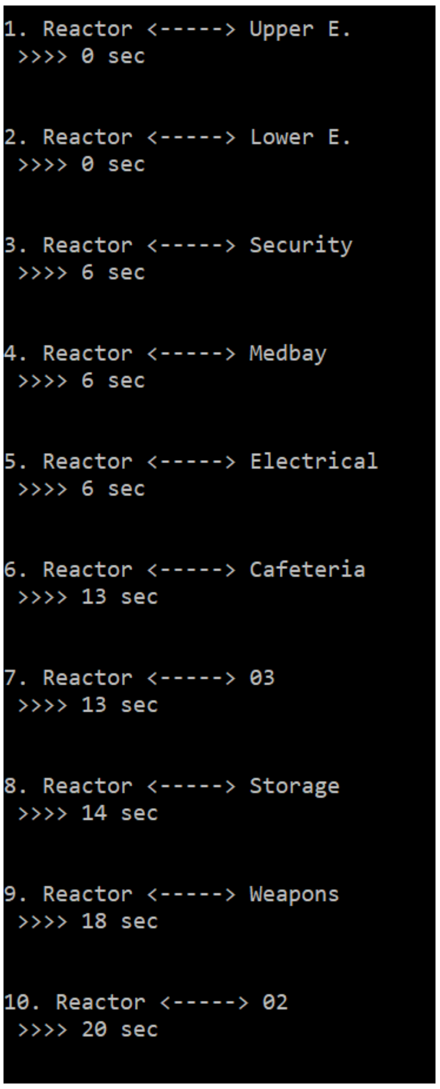
For Crewmate :



```
1. Reactor <-----> Upper E.
 >>>> 8 sec


2. Reactor <-----> Lower E.
 >>>> 8 sec


3. Reactor <-----> Security
 >>>> 6 sec


4. Reactor <-----> Medbay
 >>>> 17 sec


5. Reactor <-----> Electrical
 >>>> 20 sec


6. Reactor <-----> Cafeteria
 >>>> 21 sec


7. Reactor <-----> O3
 >>>> 30 sec


8. Reactor <-----> Storage
 >>>> 22 sec


9. Reactor <-----> Weapons
 >>>> 28 sec


10. Reactor <-----> O2
 >>>> 34 sec
```

…

For Impostor:



```
1. Reactor <-----> Upper E.
 >>>> 0 sec


2. Reactor <-----> Lower E.
 >>>> 0 sec


3. Reactor <-----> Security
 >>>> 6 sec


4. Reactor <-----> Medbay
 >>>> 6 sec


5. Reactor <-----> Electrical
 >>>> 6 sec


6. Reactor <-----> Cafeteria
 >>>> 13 sec


7. Reactor <-----> O3
 >>>> 13 sec


8. Reactor <-----> Storage
 >>>> 14 sec


9. Reactor <-----> Weapons
 >>>> 18 sec


10. Reactor <-----> O2
 >>>> 20 sec
```

…

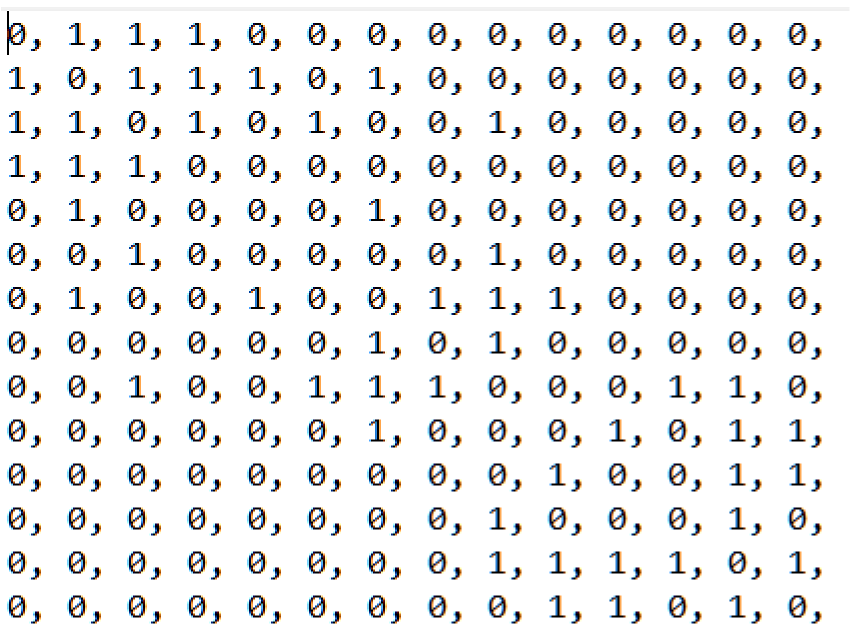
Step 4: Secure the last tasks

1. Presents and argue about the model of the map + 2. Thanks to a graph theory problem, present how to find a route passing through each room only one time.

The map has 14 rooms (14 vertex), 7 corridors (7 branches) and 14 vents.

The bets way we thought of to represent the map was by using a matrix composed of 0s and 1s. 1 represents the fact you can walk from a room to another, 0 you can't.

This matrix was placed a .txt file and read when needed.



```
0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
```

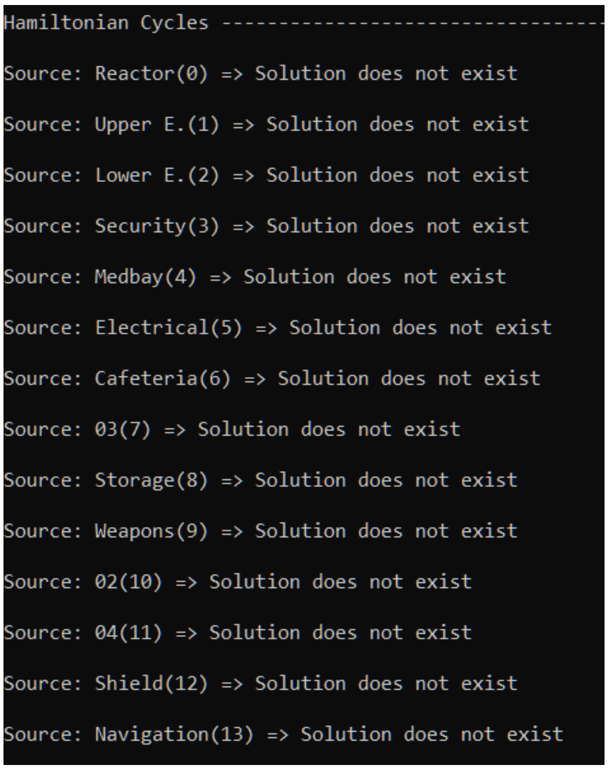
3. Argue about an algorithm solving your problem.

A **Hamiltonian** cycle is a closed loop on a graph where every node (vertex) is visited exactly once. That's what we need.

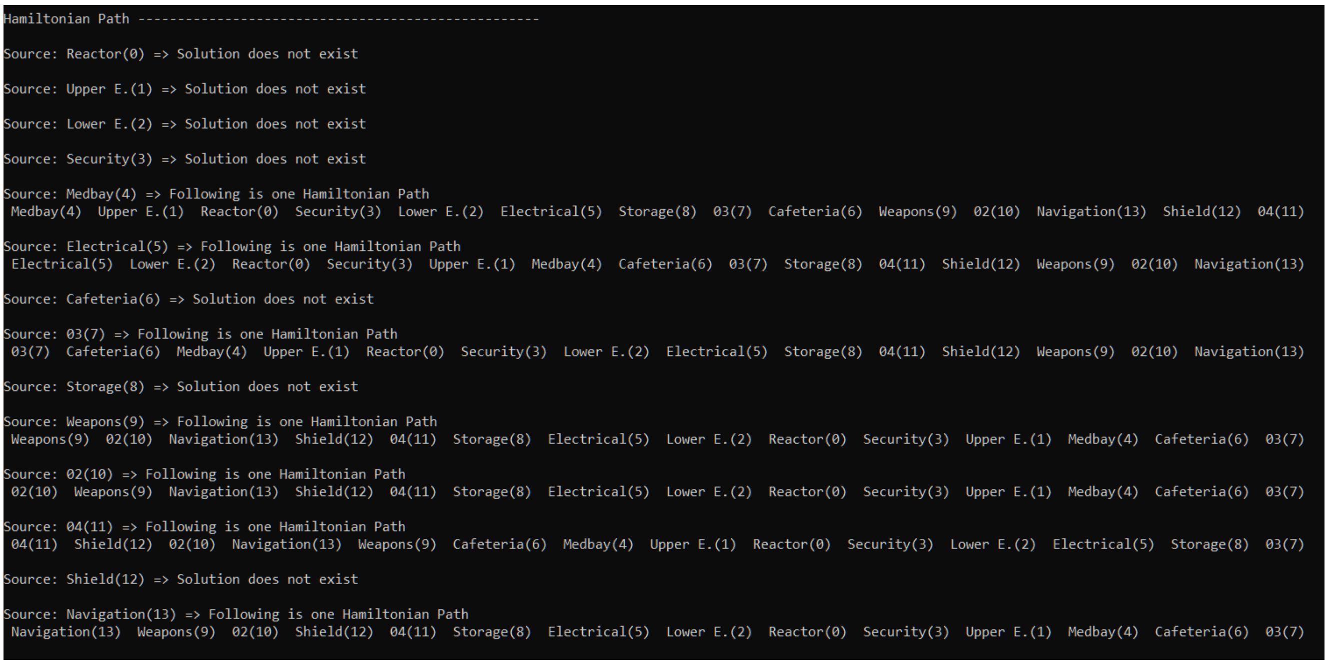
Thus, we implemented 2 functions: **hamCycle** and **hamCycleUtil**. **HamCycle** uses the **hamCycleUtil** function to determine if a cycle exists or not in our graph. However, as we displayed it in the console line, there is no Hamiltonian cycle in our ADSA graph.

Hopefully, we can also use the Hamiltonian path algorithm to check if a graph has a path between two vertices of a graph that visits each vertex exactly once. You can retrieve this in **hamPath** and **hamPathUtil** functions.

4. Implement the algorithm and show a solution.



```
Hamiltonian Cycles ---------------------------------

Source: Reactor(0) => Solution does not exist

Source: Upper E.(1) => Solution does not exist

Source: Lower E.(2) => Solution does not exist

Source: Security(3) => Solution does not exist

Source: Medbay(4) => Solution does not exist

Source: Electrical(5) => Solution does not exist

Source: Cafeteria(6) => Solution does not exist

Source: 03(7) => Solution does not exist

Source: Storage(8) => Solution does not exist

Source: Weapons(9) => Solution does not exist

Source: 02(10) => Solution does not exist

Source: 04(11) => Solution does not exist

Source: Shield(12) => Solution does not exist

Source: Navigation(13) => Solution does not exist
```



```
Hamiltonian Path --------------------------------------------------

Source: Reactor(0) => Solution does not exist

Source: Upper E.(1) => Solution does not exist

Source: Lower E.(2) => Solution does not exist

Source: Security(3) => Solution does not exist

Source: Medbay(4) => Following is one Hamiltonian Path
 Medbay(4)  Upper E.(1)  Reactor(0)  Security(3)  Lower E.(2)  Electrical(5)  Storage(8)  03(7)  Cafeteria(6)  Weapons(9)  02(10)  Navigation(13)  Shield(12)  04(11)

Source: Electrical(5) => Following is one Hamiltonian Path
 Electrical(5)  Lower E.(2)  Reactor(0)  Security(3)  Upper E.(1)  Medbay(4)  Cafeteria(6)  03(7)  Storage(8)  04(11)  Shield(12)  Weapons(9)  02(10)  Navigation(13)

Source: Cafeteria(6) => Solution does not exist

Source: 03(7) => Following is one Hamiltonian Path
 03(7)  Cafeteria(6)  Medbay(4)  Upper E.(1)  Reactor(0)  Security(3)  Lower E.(2)  Electrical(5)  Storage(8)  04(11)  Shield(12)  Weapons(9)  02(10)  Navigation(13)

Source: Storage(8) => Solution does not exist

Source: Weapons(9) => Following is one Hamiltonian Path
 Weapons(9)  02(10)  Navigation(13)  Shield(12)  04(11)  Storage(8)  Electrical(5)  Lower E.(2)  Reactor(0)  Security(3)  Upper E.(1)  Medbay(4)  Cafeteria(6)  03(7)

Source: 02(10) => Following is one Hamiltonian Path
 02(10)  Weapons(9)  Navigation(13)  Shield(12)  04(11)  Storage(8)  Electrical(5)  Lower E.(2)  Reactor(0)  Security(3)  Upper E.(1)  Medbay(4)  Cafeteria(6)  03(7)

Source: 04(11) => Following is one Hamiltonian Path
 04(11)  Shield(12)  02(10)  Navigation(13)  Weapons(9)  Cafeteria(6)  Medbay(4)  Upper E.(1)  Reactor(0)  Security(3)  Lower E.(2)  Electrical(5)  Storage(8)  03(7)

Source: Shield(12) => Solution does not exist

Source: Navigation(13) => Following is one Hamiltonian Path
 Navigation(13)  Weapons(9)  02(10)  Shield(12)  04(11)  Storage(8)  Electrical(5)  Lower E.(2)  Reactor(0)  Security(3)  Upper E.(1)  Medbay(4)  Cafeteria(6)  03(7)
```