ESILV ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Report of the project Among us

Advanced Data Structure

AMONG US

NASSAR Ibrahim
TRANG Thomas

Dirigé par DJEBALI SONIA

Contents

Abstract

Zerator asks you to organize the next "Among Us" tournament for the next ZLAN.
The rules are as following:

- Total of 100 players
- 10 players per game
- 3 random games then
- each game regroups the players by a batch of ten following their ranking.
  - o The last 10 players (in the ranking) are ejected to the tournament.
  - o Do it until it remains only 10 players
- For the last 10 players, play 5 games with reinitiated ranking. Update and check the ranking of the 10 players and give the podium.

Here is the ranking model:
- Impostor: 1pts per kill, 3pts per undiscovered murder, 10pts if win.
- Crewmate: 3pts if the argument unmasks an imposter, 1pts if all solo tasks are made, 5pts if win.

Each time a game ended, the score of each player is the mean of all its games.
The players are stored in a structured database with a log complexity to reach an element which corresponds to a score (the most optimized structure presented in the ADSA Course). The goal of this project is to implement the data structure of the game.

Part I
First step : To organize the tournament

6 questions about basic structures of the tournament

The structure we chose for our database is an AVL Tree. In this database, we will stock all of our players and their scores.

Why did we choose the AVL structure ?

AVL Tree is a type of Binary Search Tree (BST). It is a self balancing BST, each vertex is defined by a balancing factor.

$$| \text{Height(left subtree) - Height(right subtree)} | \leq 1$$

It means that the difference between heights of the left and right subtrees can't be more than one, that's how it is balanced.

In fact, the AVL Tree, gives a lot of advantages compared to others structures. Their worst time complexity is $O(\log n)$ for all basic operations of BST. Indeed, insertion and deletion are guaranteed to be a $O(\log n)$ time complexity for average and worst cases.

Furthermore in the project, we sort players by their scores, and other treatments like calculating max and min. The implementation of these operations is simplified by the data structure, as the AVL is alerady sorted.

In our *Database* AVL Tree, one node will represent a player and he will have attributs like 'score' and maybe 'id'. As previously stated, the score will be randomly added at the beginning of each game for each player. We will do treatments which randomize scores to add at each game and update players'scores at the end of a game. And another function for the last round because there are 5 games in it, it doesn't progress the same way.

To create random games on the database, we will take 10 players according to their ranking. Among these 10 players, we will randomly name 2 impostors for the created game.

In order to drop the last 10 players of the rank, we pop 10 players with the lowest scores of the AVL tree at each game. Knowing that the AVL Tree is already sorted, players we will delete are on the left leaves. Repeat this treatment until it remains only 10 players.

At the beginning of the tournament, we create 100 nodes which represent players. We compute the 3 first games and then randomly attribute points to every player. And at the end of the current game, we pop 10 last players. Then we insert the rest of the players in another new AVL tree (for example, at the first game we will insert 90 players). And we repeat this task until it remains only 10 players on the new AVL Tree, this is the last game. At the last game, there are 10 players and they play 5 games to set the winner of the tournament.

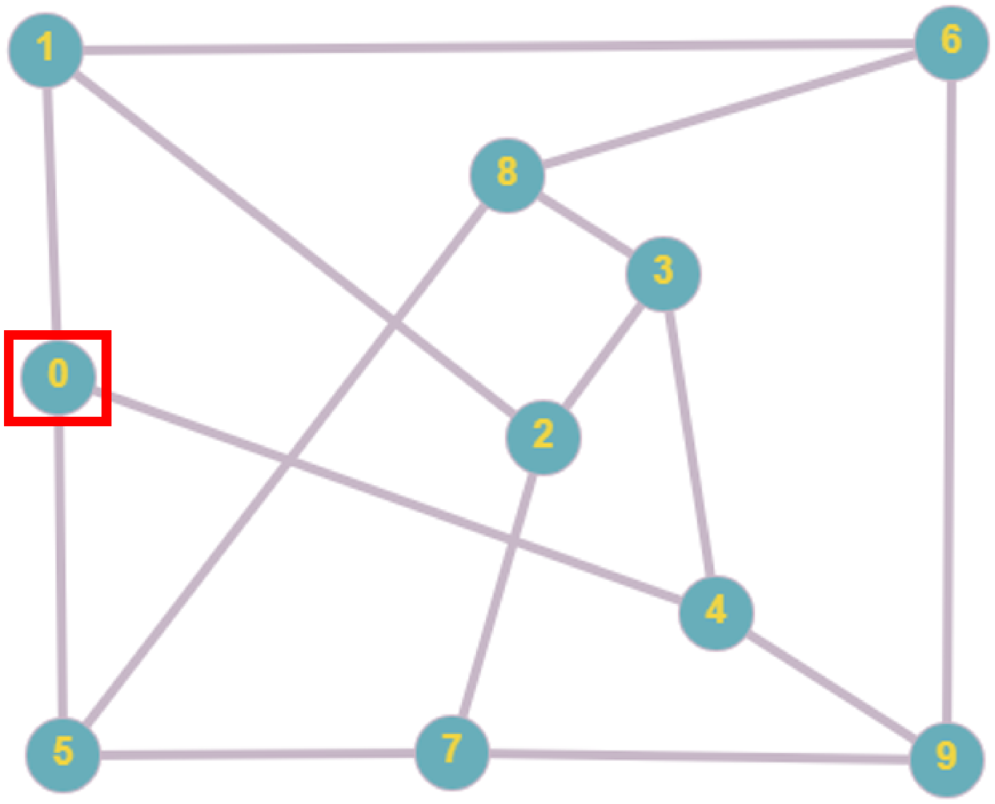
Finally to display the top 10 players, we compute an IN ORDER display of the AVL tree.

Part II
Second step : Professor Layton < Guybrush Threepwood < You

4 questions about the impostors

The representation below is the graph representation of the 10 players and the other players that they all have seen in the game. The player 0 which is red framed is the first killed player. So the aim of this step is to find the impostor.

Figure : Graph representation of players met during the studied game



And in order to find the impostor, we must set a list of probable impostors. Thanks to this representation we can easily see the links between the nodes and who they have visited. The relations between all the players of the game are represented through the adjacent matrix 9x9 below. This graph permits us to see the relation between all the nodes. In this graph, all the nodes represent players and the edges represent visited players during the game.

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

Figure : Adjacency matrix

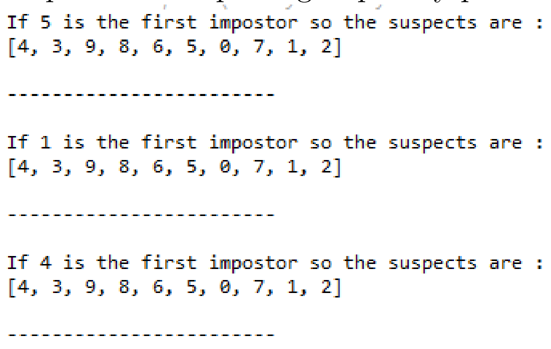
We have to create a graph in which we report all the links between every player. Then we have to set all the suspect who are {1,4,5}. Then we have to get all the adjacent vertices to each of these vertices.
And print the sets of suspects in function of the impostors.

Down below, you can find a pseudo code of the algorithm we used in order to find the set of suspects for each possible impostors.

Algorithm 1 Set of suspects knowing impostors

```
System Initialization
Read the graph
$killed \leftarrow killedCrewmate$
$FirstImpostorsSuspects \leftarrow ProbableFirstImpostors$
**for** each impostor $i$ **do**
    $nonSuspect \leftarrow i.adjacentVertices$
    $SecondImpostorsSuspects \leftarrow allVertices - killed - suspect - nonSuspects$
```

Figure : Set of probable suspects grouped by probable impostors



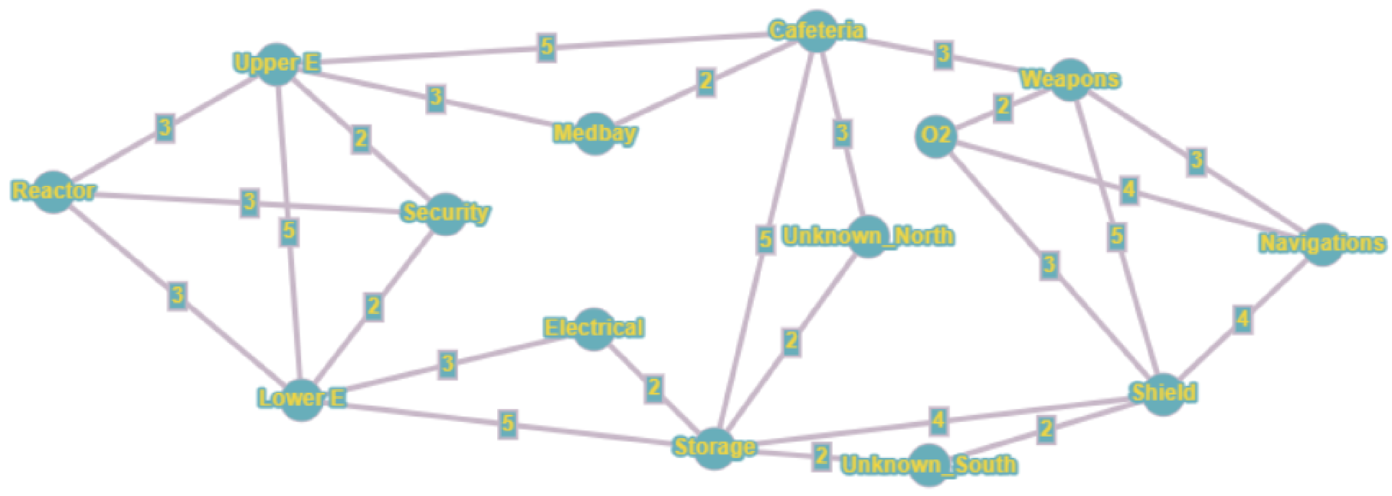
```
If 5 is the first impostor so the suspects are :
[4, 3, 9, 8, 6, 5, 0, 7, 1, 2]

-----------------------

If 1 is the first impostor so the suspects are :
[4, 3, 9, 8, 6, 5, 0, 7, 1, 2]

-----------------------

If 4 is the first impostor so the suspects are :
[4, 3, 9, 8, 6, 5, 0, 7, 1, 2]

-----------------------
```

Part III
Third step : I don't see him, but I can give proofs he vents !

4 questions about the map

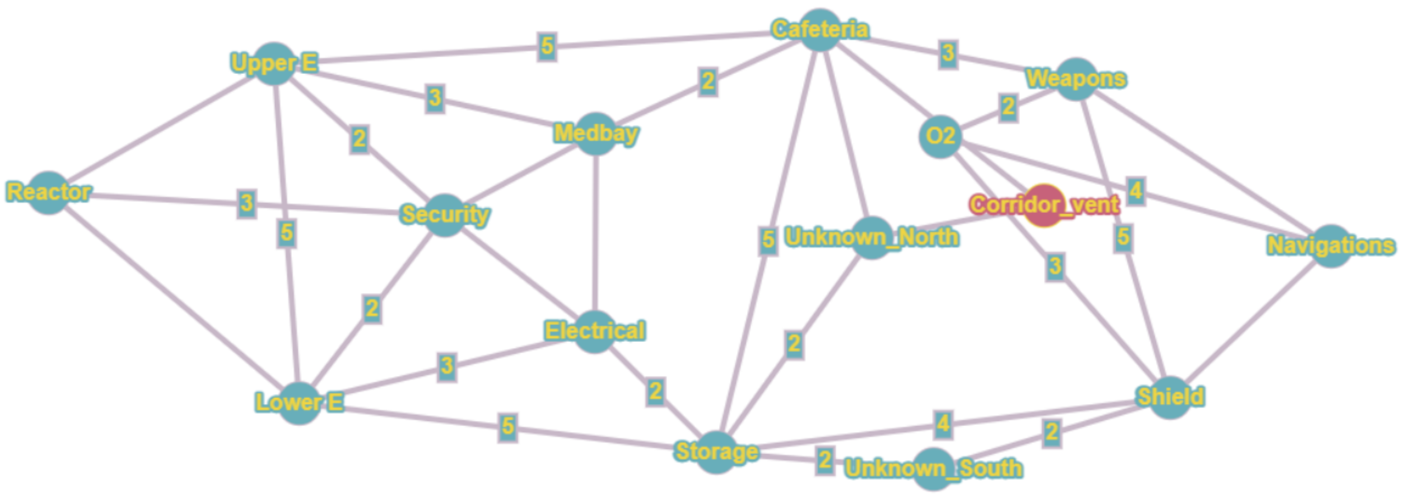
The representation of the map is a graph. Each vertex of the graph represents a room in the map. In order to get the weights between each room, we directly measured on the map.
In order to distinguish, crewmates's and impostors's map we did the maps on the same model but we added some vents and modified some links between some rooms for the impostors's in order to represent impostors's map.

Figure : Crewmates Map representation



Below We can see the impostor's map which is the same as the crewmates's map but we some added vents with no weight.

Figure : Impostors Map representation



The shortest path algorithm we are going to use for this problem is the Floyd Warshall algorithm. It permits to compute all pairs shortest path in a graph.

So why did we choose this one instead of another one ?

We chose the Floyd Warshall algorithm because it can be used to compute the distance between every pair of vertices in the graph. The goal is to calculate the time each players takes to travel in the map. Knowing that the impostors take less time to compute, this can be a hint to find out who are the impostors. We presented all the times to compute between rooms in a DataFrame format.

Part IV
Fourth step : Secure the last tasks

4 questions about the final solution

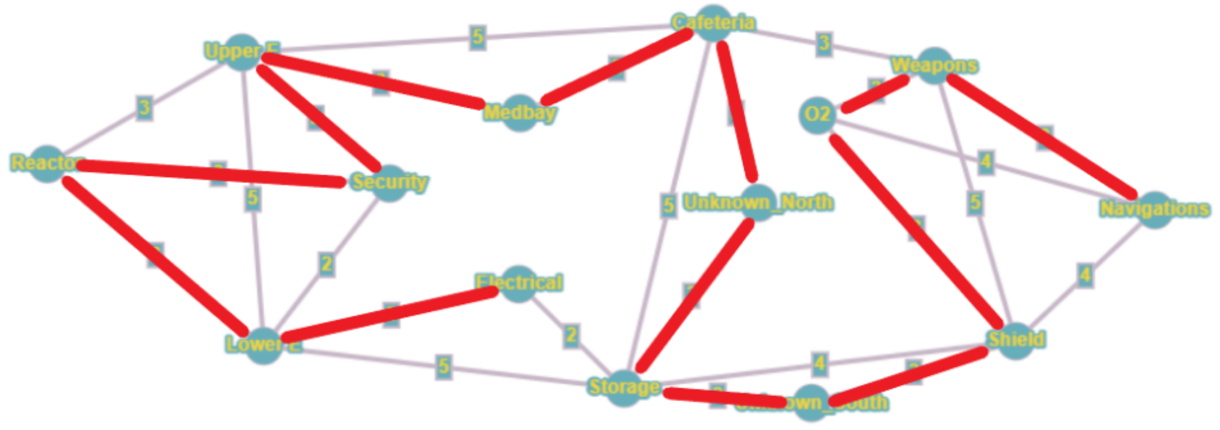
The model of the map that we are going to use for this last step is the same graph map as the graph we used in part 3 for the crewmates map. Because this last step concerns the crewmates and the realization of the last tasks.

To find the route we are going to use the Hamiltonian cycle which allows us to go through a room once and only one per cycle finding the shortest path. We could have chosen a Minimum Spanning Tree algorithm to do this but the problem is that we can go through one node many times in a MST.

So that's why we are going to use a hamiltonian path to do the fourth step even if the complexity is O(n!) which is higher than the complexity of a MST algorithm. But the algorithm that fits well our problem here is the hamiltonian path.

We can count 7 hamiltonian paths which gave the lowest scores. We randomly chose one path among these 7 paths. We decided to represent this path on the creawmates map we saw in the part 3 of this report. Each room is visited only once in the traversal. We can see it in the representation below.

Figure : Hamiltonian path



Algorithm 2 Find hamiltonian path

```
function FindHamiltonianPath(path, nbRooms)
set all vertices as unvisited set start vertex as visited
append start to path
if path.size=nbRooms then return True
end if
for each room i do
    if (i is unvisited) then set i visited append i to path
        if (FindHamiltonianPath(path, nbRooms)) then return True
        end if then set i as unvisited and remove i from path
    end if
return False
```

Part V
Acknowledgement

This project was the opportunity for us to work on a concrete project of Data Structure. It was very pleasant to implement because we did it on a game that we didn't really know but which we found really interesting and clever. Having finished this project, we would like to thank Chendeb Safwan and Djebali Sonia for advicing us when we needed and giving us the opportunity to work on such a project to improve our skills.

Part VI
Conclusion

To conclude, we are satisfied of our work on the realisation of the project, we think it helped us using what we learnt this practical works. Furthermore, we are happy because we reached our goals that we fixed to ourselves at the beginning of the project. This experience was enriching for us because it permitted us to learn a lot and to improve facing the project.