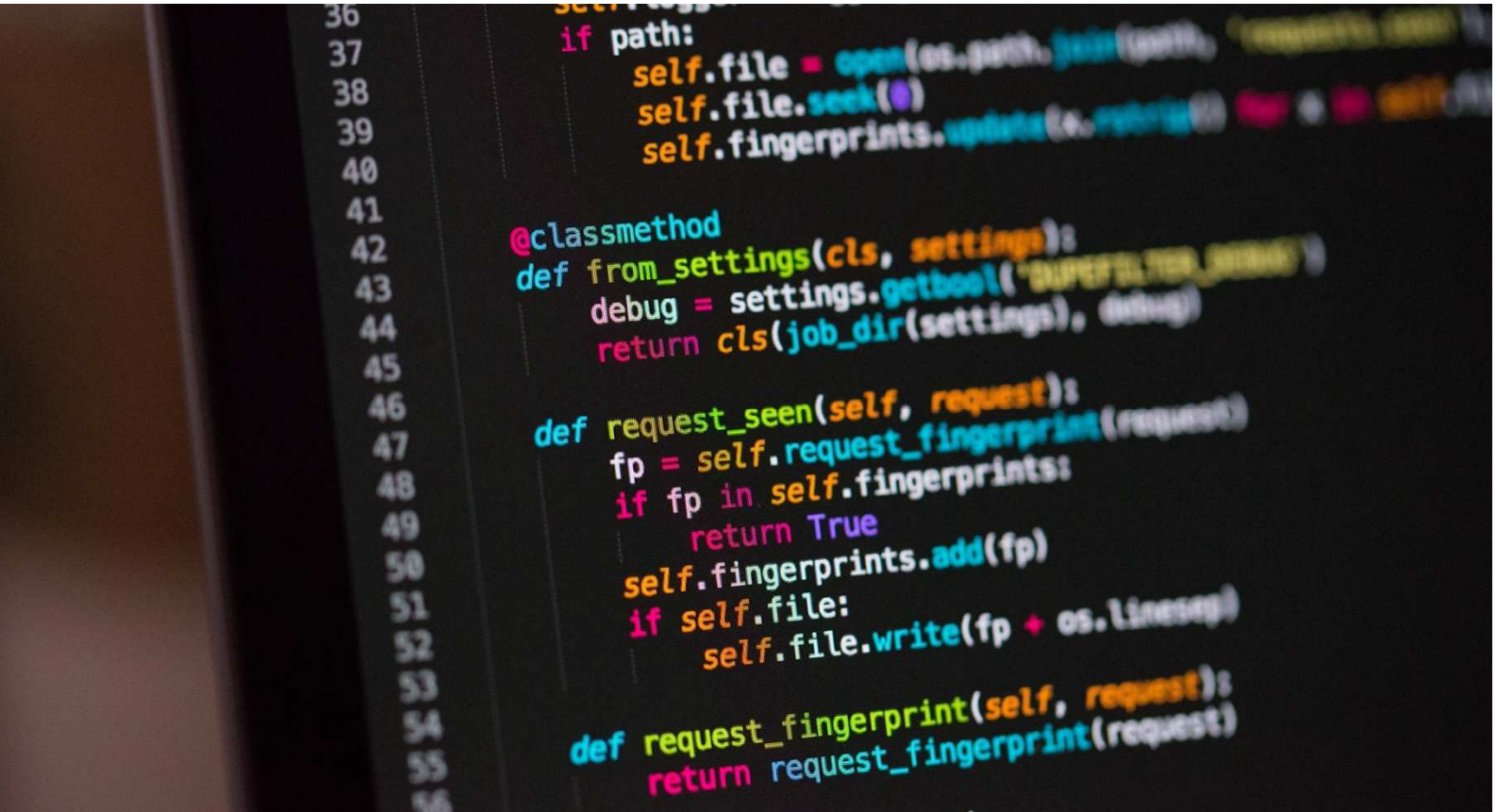
AMONG US

Advanced Data Structures & Algorithms

Bilal SELMI & Yanis FADILI DIA5

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

SUMMARY



Step 1 : To organize the tournament

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

To represent a player and its score, we've chosen an AVL tree because it's more adapted than a list, a dictionary, or other data structures viewed in class.

Therefore, the Player will be stocked in a Node from the AVL Tree that will contain 100 nodes like that (so 100 players).

Moreover, we had to respect the condition regarding the complexity that had to be log(n).

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

For us, the most optimized data structures to illustrate the tournament is the AVL tree as said previously, indeed, it checks all the condition of a good data structure and the complexity asked. Even if we first genuinely thought of a "classic" list such it is simpler to implement.

Moreover, it has the advantages of a Binary Search Tree, but it is balanced so it facilitates the research of an element knowing that they are sorted in a way.

The problem was when it comes to place players in the Tree because we use to do it with ints, so it was easy to place players in left or right whether the value was bigger or not than the root, but with Players it was more difficult.

To overcome this problem, we decided to redefine the "<" and ">" operators in the class by using __lt__ and __gt__ (less than and greater than) (we did the same with "<=" and ">=" with __le__ and __ge__ (less or equal and greater or equal).

So, we give a definition and made our program able to understand what Player A > Player B means, he had to compare their scores to place them in our AVL Tree.

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

To randomize the players, score between 0 to 12 we've simply import the library random which contains the function randint(0,12) on Python3.

By implementing the score as an attribute of the class Player, we just had to recall the method at each game such that a new score was generated and we just had to add it to the previous one to have the cumulate scores (except when there are only 10 players left we reset all the scores).

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

To update the score, we use the getMinvaluenode() method from our class Tree which return the player with the lower score. By using this method in a loop, we are able to get the 10 players with the worst scores of the tournament and by adding the delete method, we can delete from the tournament these 10 worst players. To update the score, we store all the players in a list, and we apply the update_score() method which returns the mean of the 3 games for their new score. We do it till it remains only 10 players, then we reset all the scores and finally we play 5 more games before announcing the podium.

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

To create random games based on the database, we create an AVL Tree with 100 nodes which correspond to the 100 Players. The players have all a score and a unique pseudo.

Then we have to play games and to kick out the 10 players with the worst scores, for that, we execute 9 times the method delete_last_ten which, as its name indicates, will delete the 10 last players regarding their scores by using the getMinValuenode() method that we couple with the delete function.

Finally, when there are ten players left, we reset the scores and we launch the 5 last games by generating new random scores.

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

After all that, when we have 10 players left with their final score, we just store them into a list and then we just sort the list by scores in decreasing order such that the first element of the list will be the winner of the tournament and we can display his name.

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

As said previously, we use the getMinvaluenode() method to return the Player with the least score and then we delete it, by doing this in a for loop that iterates 10 times, we can delete the 10 least players. By doing this whole thing 9 times, we can be sure that we will have delete 90 players such that there are only 10 players.

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

We use our Inorder method to store the 10 last players in a List that we have created before, when the final game is done, we will therefore have a list with these 10 players with their names and scores and we just have to sort it in order to have the podium and print the list such that we know who the 10 last players and their last rank are and the first one being the winner.

Step 2: To organize the tournament

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

We created a graph class that will have as an attribute a dictionary of the vertices such that each key will be a player and the values will be the players seen by the key player.
We also did that as a matrix to be clearer in the display and such that we could use our method further to catch the eventual impostors knowing which players could be ones regarding who they saw during the game.

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

We used a Graph coloring approach to solve this problem. Knowing who can be an impostor because of the reported kill, we can know, by analyzing in our graph and matrix the neighbors vertices who can be an impostor. By doing that with every suspect, we are sure that the impostor is among a list. Still remains to discover who this is!

3. Argue about an algorithm solving your problem.

By researching in the matrix of who saw who among the players and knowing that impostors never walk together so it's impossible that they saw themselves, and that Player 0 has been found dead after seeing Players 1, 4 and 5, we knew that the impostor was among these Players, so we had to look for who saw these players with our graph coloring approach to define a list of probable impostors.

4. Implement the algorithm and show a solution.

```
Knowing that Player 1 may be an Impostor, the other one may be among those players :
Player 3 || Player 4 || Player 5 || Player 7 || Player 8 || Player 9 ||

Knowing that Player 4 may be an Impostor, the other one may be among those players :
Player 1 || Player 2 || Player 5 || Player 6 || Player 7 || Player 8 ||

Knowing that Player 5 may be an Impostor, the other one may be among those players :
Player 1 || Player 2 || Player 3 || Player 4 || Player 6 || Player 9 ||
```

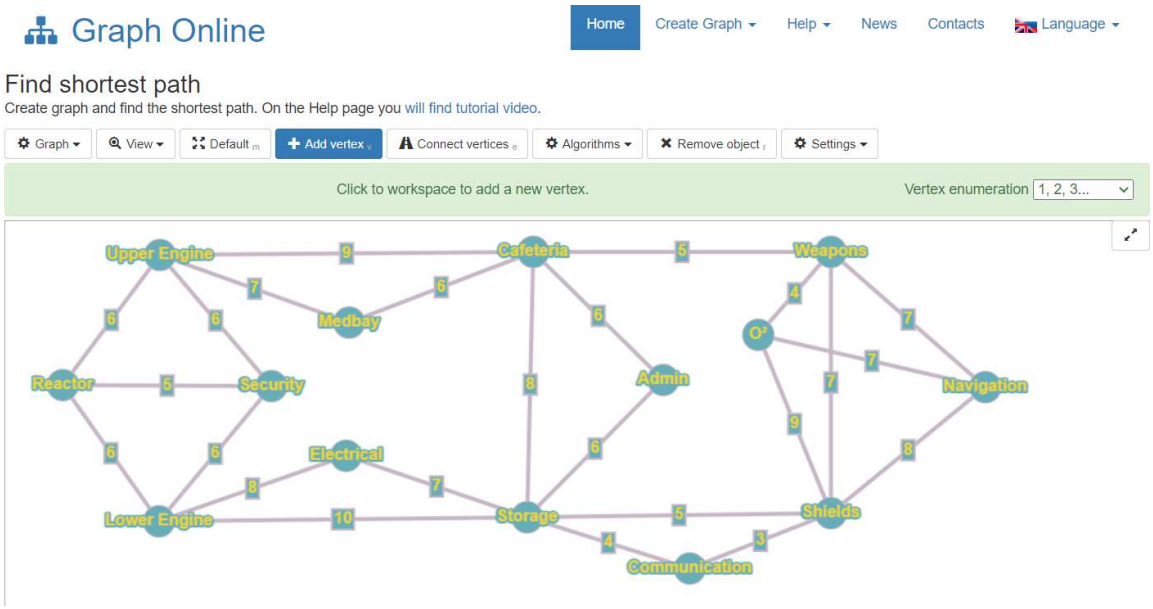
Here is what we found, in fact this just tell us that basically, everyone can be an impostor, indeed even if we are sure that among the players 1 4 5 there is 1 impostor, we do not know who his partner is, but, if along the game we figure out who among these players is an impostor, we can know who among the other players cannot be one of them. (For example, if we discover Player 4 as an impostor, we can be sure that player 3 is a Crewmate).

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 had to implement either the Crewmate Graph or the Impostor Graph, to do so, it was necessary to measure the different distances between the rooms. We used a ruler to do so and put all the values into 2 matrices, one for Crewmates and one for Impostors (note that by using a classic ruler, the distances are approximative but it doesn't change our approach of the problem it's only a matter of precision).
The difference between Crewmates and Impostors is that Impostors are able to take vents which allow them to travel between given rooms with a time of 0, it is immediate.

To visualize well the graph, we used the graphonline website which allow us to create a graph with distances and also give us the adjacency and distance matrix so that it avoids us to write it number by number in the python script.



Here is the Crewmate matrix that we have at the end for example:

```
Crewmate_Graph = [[0, 6, 5, 6, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF],
[6, 0, 6, INF, 7, 9, INF, INF, INF, INF, INF, INF, INF, INF],
[5, 6, 0, 6, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF],
[6, INF, 6, 0, INF, INF, 10, 8, INF, INF, INF, INF, INF, INF],
[INF, 7, INF, INF, 0, 6, INF, INF, INF, INF, INF, INF, INF, INF],
[INF, 9, INF, INF, 6, 0, 8, INF, 6, 5, INF, INF, INF, INF],
[INF, INF, INF, 10, INF, 8, 0, 7, 6, INF, INF, INF, 5, 4],
[INF, INF, INF, 8, INF, INF, 7, 0, INF, INF, INF, INF, INF, INF],
[INF, INF, INF, INF, INF, 6, 6, INF, 0, INF, INF, INF, INF, INF],
[INF, INF, INF, INF, INF, 5, INF, INF, INF, 0, 4, 7, 7, INF],
[INF, INF, INF, INF, INF, INF, INF, INF, INF, 4, 0, 7, 9, INF],
[INF, INF, INF, INF, INF, INF, INF, INF, INF, 7, 7, 0, 8, INF],
[INF, INF, INF, INF, INF, INF, 5, INF, INF, 7, 9, 8, 0, 3],
[INF, INF, INF, INF, INF, INF, 4, INF, INF, INF, INF, INF, 3, 0]]
```

INF means that the two rooms are not connected, thus, the distance between them is infinite (math.inf). Each line and column correspond to a room that's why in the diagonal there is only zeros between the distance between a room and itself is null.

2. Argue about a pathfinding algorithm to implement.

Following what we have seen during this course, we had the choice between Dijkstra, Bellman-Ford and Floyd-Warshall. We decided to choose Floyd-Warshall because it's the only one that calculates the (minimum) distances between every rooms by pair.

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

```
Crewmate_Graph = [[0, 6, 5, 6, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF],
[6, 0, 6, INF, 7, 9, INF, INF, INF, INF, INF, INF, INF, INF],
[5, 6, 0, 6, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF],
[6, INF, 6, 0, INF, INF, 10, 8, INF, INF, INF, INF, INF, INF],
[INF, 7, INF, INF, 0, 6, INF, INF, INF, INF, INF, INF, INF, INF],
[INF, 9, INF, INF, 6, 0, 8, INF, 6, 5, INF, INF, INF, INF],
[INF, INF, INF, 10, INF, 8, 0, 7, 6, INF, INF, INF, 5, 4],
[INF, INF, INF, 8, INF, INF, 7, 0, INF, INF, INF, INF, INF, INF],
[INF, INF, INF, INF, INF, 6, 6, INF, 0, INF, INF, INF, INF, INF],
[INF, INF, INF, INF, INF, 5, INF, INF, INF, 0, 4, 7, 7, INF],
[INF, INF, INF, INF, INF, INF, INF, INF, INF, 4, 0, 7, 9, INF],
[INF, INF, INF, INF, INF, INF, INF, INF, INF, 7, 7, 0, 8, INF],
[INF, INF, INF, INF, INF, INF, 5, INF, INF, 7, 9, 8, 0, 3],
[INF, INF, INF, INF, INF, INF, 4, INF, INF, INF, INF, INF, 3, 0]]

Impostor_Graph = [[0, 0, 5, 0, INF, INF, INF, INF, INF, INF, INF, INF, INF, INF],
[0, 0, 6, INF, 7, 9, INF, INF, INF, INF, INF, INF, INF, INF],
[5, 6, 0, 6, 0, INF, INF, 0, INF, INF, INF, INF, INF, INF],
[0, INF, 6, 0, INF, INF, 10, 8, INF, INF, INF, INF, INF, INF],
[INF, 7, 0, INF, 0, 6, INF, 0, INF, INF, INF, INF, INF, INF],
[INF, 9, INF, INF, 6, 0, 8, INF, 0, 5, 4, 4, INF, INF],
[INF, INF, INF, 10, INF, 8, 0, 7, 6, INF, INF, INF, 5, 4],
[INF, INF, 0, 8, 0, INF, 7, 0, INF, INF, INF, INF, INF, INF],
[INF, INF, INF, INF, INF, 0, 6, INF, 0, INF, INF, INF, INF, INF],
[INF, INF, INF, INF, INF, 5, INF, INF, INF, 0, 4, 0, 7, INF],
[INF, INF, INF, INF, INF, INF, INF, INF, INF, 4, 0, 7, 9, INF],
[INF, INF, INF, INF, INF, INF, INF, INF, INF, 0, 7, 0, 0, INF],
[INF, INF, INF, INF, INF, 5, INF, INF, 7, 9, 0, 0, 3],
[INF, INF, INF, INF, INF, INF, 4, INF, INF, INF, INF, 3, 0]]
```

For the vent that start from Cafeteria and Admin, the third one is not a room in particular, so we decided to state that by taking this vent, we had to remeasure the time between this particular point and the rooms that surrounds it (Weapons, Navigation and Shield), indeed, to go to Storage for example it is preferable to go by foot. And the result was that it almost divides by 2 all these distances.

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

As an indicative way, here is the distance between certain rooms for an impostor for example. Time between Medbay and itself is null such as between Medbay and Security or Electrical due to the presence of a vent.

```
Start = Medbay | Stop = Reactor | Time = 5
Start = Medbay | Stop = Upper Engine | Time = 5
Start = Medbay | Stop = Security | Time = 0
Start = Medbay | Stop = Lower Engine | Time = 5
Start = Medbay | Stop = Medbay | Time = 0
Start = Medbay | Stop = Cafeteria | Time = 6
Start = Medbay | Stop = Storage | Time = 7
Start = Medbay | Stop = Electrical | Time = 0
Start = Medbay | Stop = Admin | Time = 6
Start = Medbay | Stop = Weapons | Time = 10
Start = Medbay | Stop = O² | Time = 10
Start = Medbay | Stop = Navigation | Time = 10
Start = Medbay | Stop = Shields | Time = 10
Start = Medbay | Stop = Communication | Time = 11
```


Step 4: Secure the last tasks

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

We represented the map as a graph in which we precise to the program every edge meaning all the rooms that are linked. Note that it was not necessary to recense the weight on the graph because we did not take into account the distances between the map because we just want to travel all by passing through each room only one time.

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

We had the choice between Kruskal, Prim and Hamiltonian paths. We chose Hamilton because it was the only one that provided us the capability to travel into every room without ever passing more than one time in one of them. We also tried Kruskal, but it was no able to respond to our problem.

3. Argue about an algorithm solving your problem.

We created a Graph class, give it all the connection between the rooms and then we just had to iterate the algorithm starting from any room (given by the user) and store every vertices into a list and try all the possibilities and verify each time if this list contains only unique values meaning that it never passed through a room more than one time.
Some rooms are impossible to start with meaning there is no Hamiltonian path starting from this room, but it is precised when running the program we give the possibility for the user to enter the starting room by himself but we show which room does not provide Hamilton path.

4. Implement the algorithm and show a solution.

```
Here are the different rooms, choose one to start with :


The room that can start a Hamiltonian Path are 4,7,8,9,10,11,13. Otherwise it won't be possible to
start a Hamiltonian Path.
|Reactor : 0|
|Upper Engine : 1|
|Security : 2|
|Lower Engine : 3|
|Medbay : 4|
|Cafeteria : 5|
|Storage : 6|
|Electrical : 7|
|Admin : 8|
|Weapons : 9|
|O² : 10|
|Navigation : 11|
|Shields : 12|
|Communication : 13|

Enter a valid room (number between 1 and 14): 8
Here are the different Hamiltonian Paths possible, starting from Admin :

['Admin', 'Cafeteria', 'Medbay', 'Upper Engine', 'Reactor', 'Security', 'Lower Engine', 'Electrical',
'Storage', 'Communication', 'Shields', 'Weapons', 'O²', 'Navigation']


['Admin', 'Cafeteria', 'Medbay', 'Upper Engine', 'Reactor', 'Security', 'Lower Engine', 'Electrical',
'Storage', 'Communication', 'Shields', 'Weapons', 'Navigation', 'O²']


['Admin', 'Cafeteria', 'Medbay', 'Upper Engine', 'Reactor', 'Security', 'Lower Engine', 'Electrical',
'Storage', 'Communication', 'Shields', 'O²', 'Weapons', 'Navigation']


['Admin', 'Cafeteria', 'Medbay', 'Upper Engine', 'Reactor', 'Security', 'Lower Engine', 'Electrical',
'Storage', 'Communication', 'Shields', 'O²', 'Navigation', 'Weapons']
```

(and other paths…)

We just display every Hamilton Paths available without taking into account the distance, indeed it does just depend on the tasks that remain and the fact that we have to travel into every room only one time and not the time taken to finish these.