Advanced Data Structure and Algorithms Report

AMONG US PROJECT

BUE MATHIEU
IOS 1 – PROMO 2022

STEP 1: TO ORGANIZE THE TOURNAMENT

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

In order to represent the players, I have chosen to create a class Player.

The Player class has two attributes:

- An id which identifies each player
- A score which enable a ranking

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

Considering “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).”, I have chosen to use an AVL tree, with this data structure it is possible to have a log complexity search.

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

I use random python library for this function.

```
Random_score(game)
        for each player in game
                player.score <- player.score + randint(0,12)
        return game
```

In this function I just add a random number to the score of each player in the game.

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

```
Update_score(tree, player_list)
        tree.root <- None
        for each player in player_list
                tree.root <- tree.insert(tree.root,player)
        return tree.root
```

The easiest way to update the tree to recreating it from scratch. All the values are changing so there is no reason to change them 1 by 1.

Considering N is the length of the player list.

Considering M is the number of nodes in the tree.

I make the following approximation M ~ N

The complexity of updating the tree is N*log(N)

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

In order to make this, I have used 2 functions and the random python library.

```
Random_games(tree)
        game_list <- []
        for i from 0 to 9
                game_list.append([])
        Random_assigment(tree.root,game_list)
        return game_list


Random_assigment(root,game_list)
        curr <- root
        if curr == None
                return
        else
                Random_assigment(curr.left,game_list)
                Random_assigment(curr.right,game_list)
                index <- randrange(0,10)
                While game_list.length == 10
                        Index <- randrange(0,10)
                game_list[index]append(curr.val)
```

The first step is to create a list. The size of the list is 10 because there are 10 players per game (players total is 100) and there are no eliminated players yet.
The idea is to generate a random number between 0 and 9 and to check if the game corresponding to this number is not full, if it is not full, I add the player to the game.
This method is working in this problem because the maximum number of games is 10. But if the number of games becomes higher, another method should be implemented.
Indeed completing the last game is longer because there is 1/10 chance to have the index of the last game.

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

```
Ranking_games(tree,nb_games)
        game_list <- []
        for i from 0 to nb_games
                game_list.appen([])
        Ranking_assigment(tree.root,game_list)
        return game_list

Ranking_assigment(root,game_list)
        curr <- root
        index <- 0
        ranking_assigment(curr.left,game_list)
        while(len(game_list[index]==10):
                index <- index + 1
        game_list[index].append(curr.val)
        ranking_assigment(curr.right,game_list)
```

To create games based on ranking, I use an InOrder traversal but instead of printing the value I add it to a game. When the game is full, I complete the next one. The number of players can change so I put a nb_game value which corresponds to the number of games.

The worst players are always in the game 0 and the best players in the game 9 with this method.

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

```
Eliminated_players_list(root,eliminated_players)
        curr <- root
        if curr == None
                return
        else
                eliminated_player_list(curr.left,eliminated_players)
                if len(eliminated_players) < 10
                        eliminated_players.append(curr.val)
                eliminated_players_list(curr.right,eliminated_players)
        return eliminated_players

Drop_players(tree,eliminated_players,players_list)
        for each p in eliminated_players
                players_list.remove(p)
        tree.root = update_score(tree,players_list)
        return tree.root
```

The function Eliminated_players selects the 10 worst scores in the tree and return a list containing these players. The function Drop_players removes the eliminated players from the tree.

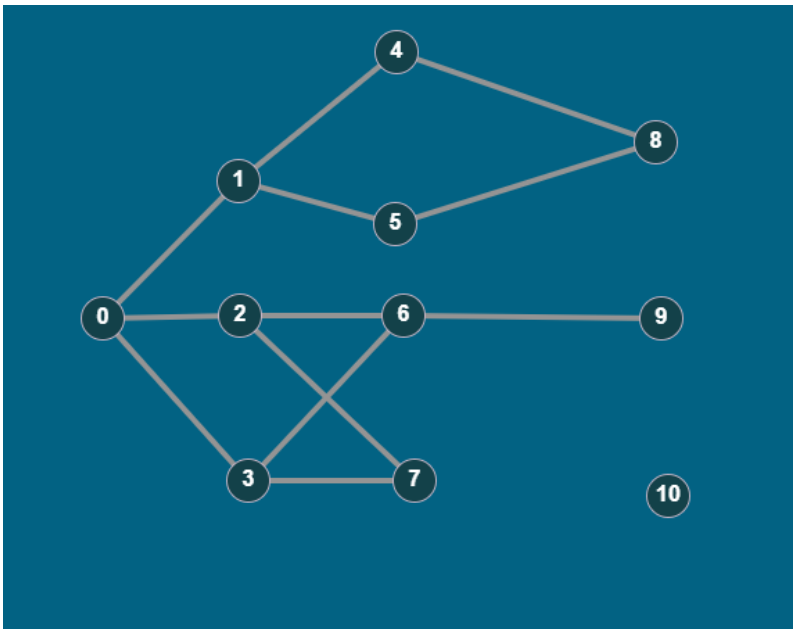
8. Present and argue about a method which display the TOP10 players and the podium after the final game.

In order to print the last final 10 players ranked by score, I do an InOrder traversal, but I swap left and right. I'm able to print the winner first, then the second … etc

```
Winners(root)
        curr <- root
        if curr == None
                return
        else
                Winners(curr.right)
                print(curr.val.to_string())
                Winners(curr.left)
```

STEP 2: PROFESSOR LAYTON < GUYBRUSH THREEPWOOD < YOU

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



I represent the players as nodes and when they see each other I make a bridge between them.

For instance, player 0 has seen Player 1, 2 and 3 and Player 10 has seen nobody.

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

This problem is a coloring graph problem.

The idea is to find the color of the dead player, the suspected killer players, the safe players and the suspected 2nd impostor.

First I consider everyone as a suspected 2nd impostor.

I start by analyzing the bridge of the dead player, I have the list of suspected killers, then by analyzing the bridge of the suspected killers I have the list of safe players.

I put the color of the safe players, but it remains killers and dead player tag as safe. So I overwrite the killers and then the dead. Now I'm sure to have a graph correctly colored.

3. Argue about an algorithm solving your problem.

```
Suspicious(dead_player,players)
    sus <- []
    safe <- []
    for each player in dead_player.bridge
        sus.append(p)
    for player in sus
        for b in players.bridge
            safe.append(b)
    for player in safe
        player.color <- "safe" #it could tag killer and it tags dead
    for player in sus
        player.color <- "sus-killer" #there is no more sus considered as safe
    dead_player.color <- "dead"
```

4. Implement the algorithm and show a solution.

Cf part2.py

The graph is generated randomly. The dead player is always player 0.

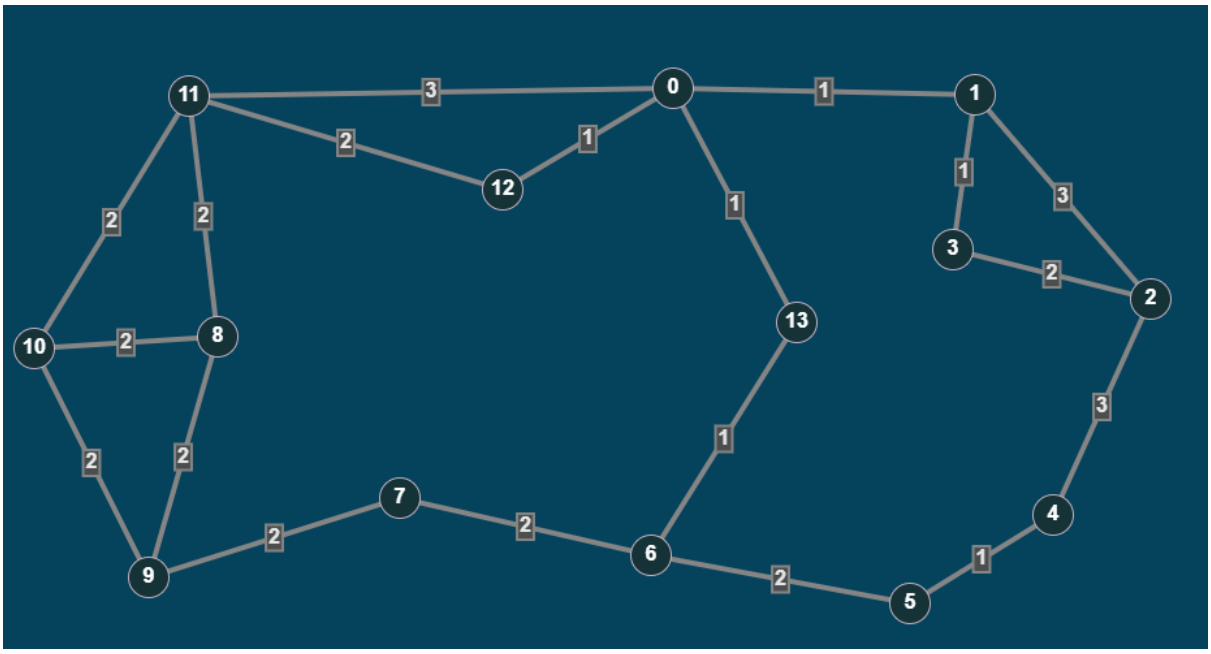
Output :

```
possible impostors :
Id : 1
Bridge : Id : 7 ,  Id : 7 ,  Id : 9 ,
Color :sus-2nd-imp

Id : 3
Bridge : Id : 2 ,  Id : 0 ,  Id : 3 ,  Id : 3 ,  Id : 5 ,  Id : 5 ,  Id : 6 ,  Id : 7 ,  Id : 7 ,
Color :sus-killer

Id : 4
Bridge : Id : 2 ,  Id : 8 ,  Id : 2 ,
Color :sus-2nd-imp

Id : 6
Bridge : Id : 0 ,  Id : 3 ,  Id : 7 ,  Id : 9 ,
Color :sus-killer

Id : 8
Bridge : Id : 4 ,  Id : 5 ,  Id : 9 ,
Color :sus-2nd-imp
```

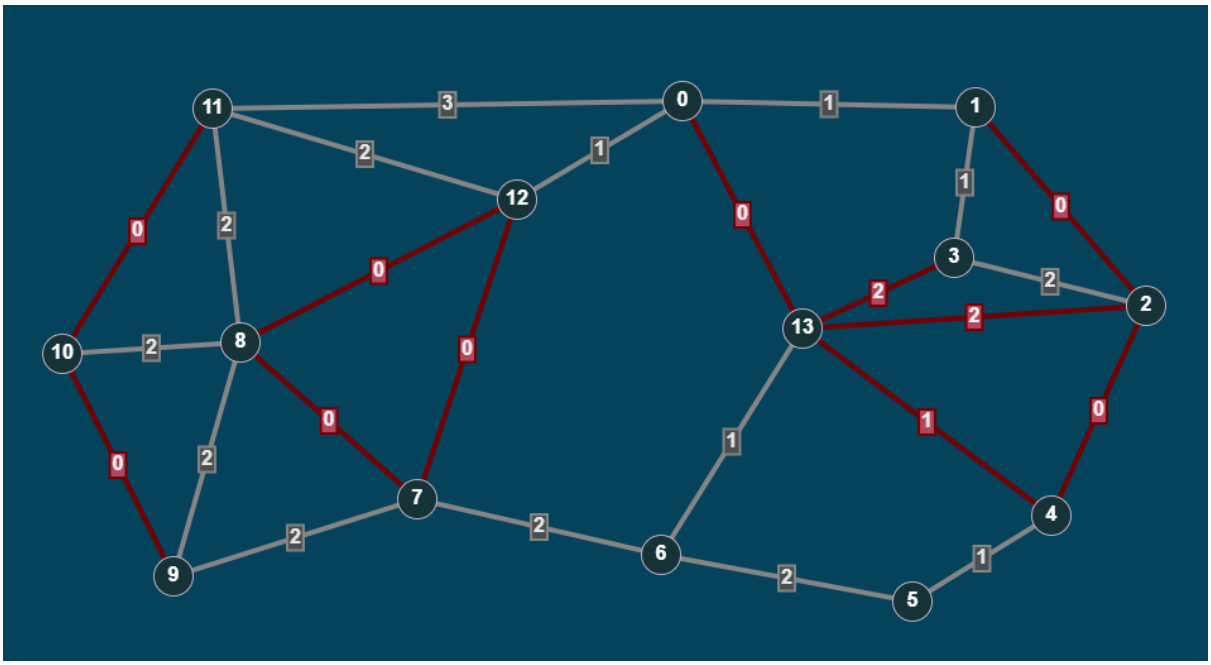
STEP 3: I DON'T SEE HIM, BUT I CAN GIVE PROOFS HE VENTS!

1. **Presents and argue about the two models of the map.**

Crewmate Maps :



Impostors map:



I have considered that the vent from admin(13) enable to travel faster to o2(3),nav(2) and shield(4).

2. Argue about a pathfinding algorithm to implement.

Considering I must print the distance for all the pair of rooms. The best algorithm is Floyd – Warshall because it calculates the distance from a node to all the others.

```
For each cell (i, j) in M:
  If i == j:
     M[i][j] = 0
   If (i, j) is an edge in E:
     M[i][j] = weight(i, j)
   Else:
     M[i][j] = infinity
For k from 1 to |V|:
    For i from 1 to |V|:
        For j from 1 to |V|:
            If M[i][j] > M[i][k] + M[k][j]:
                 M[i][j] = M[i][k] + M[k][j]
```

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

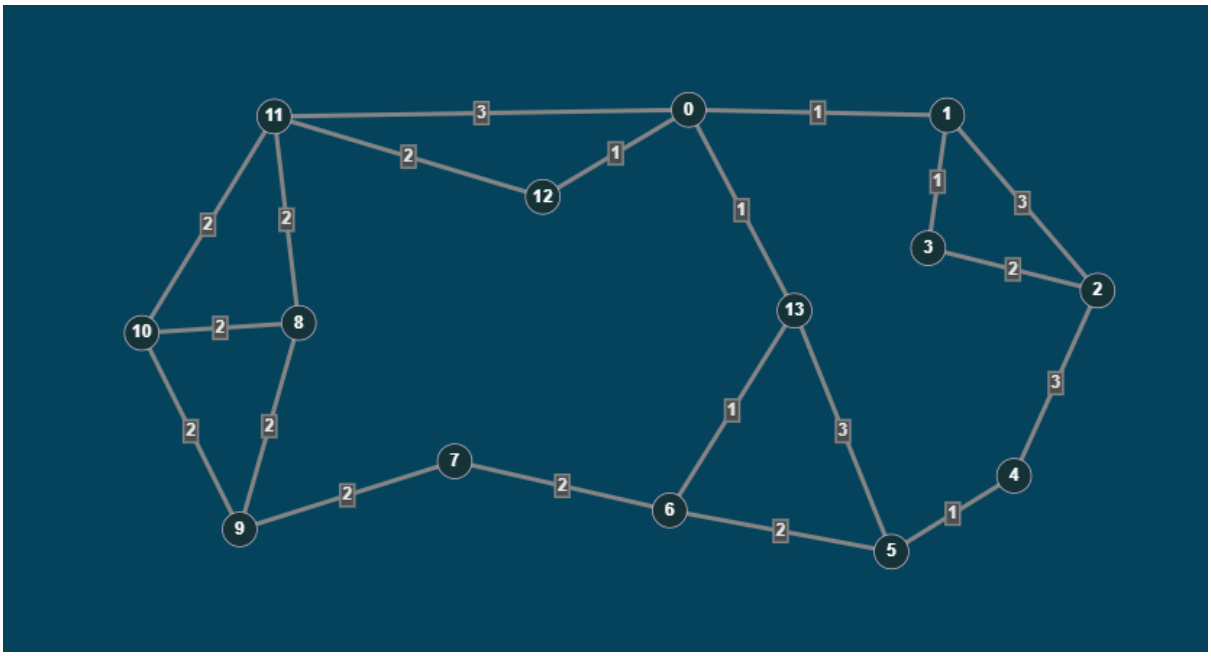
Cf part3.py

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

```
In [106]: runfile('C:/Users/M BUE/Desktop/Esilv/A4/S1/Algo/Projet Among Us/part3report.py', wdir='C:/Users/M BUE/
Crewmate
Following matrix shows the shortest distances between every pair of vertices :
      0       1       4       2       5       4       2       4       5       6       5       3       1       1
      1       0       3       1       6       5       3       5       6       7       6       4       2       2
      4       3       0       2       3       4       6       8       9      10       9       7       5       5
      2       1       2       0       5       6       4       6       7       8       7       5       3       3
      5       6       3       5       0       1       3       5       9       7       9       8       6       4
      4       5       4       6       1       0       2       4       8       6       8       7       5       3
      2       3       6       4       3       2       0       2       6       4       6       5       3       1
      4       5       8       6       5       4       2       0       4       2       4       6       5       3
      5       6       9       7       9       8       6       4       0       2       2       2       4       6
      6       7      10       8       7       6       4       2       2       0       2       4       6       5
      5       6       9       7       9       8       6       4       4       2       0       2       4       6
      3       4       7       5       8       7       5       6       2       4       2       0       2       4
      1       2       5       3       6       5       3       5       4       6       4       2       0       2
      1       2       5       3       4       3       1       3       6       5       6       4       2       0
Impostors
Following matrix shows the shortest distances between every pair of vertices :
      0       1       1       2       1       2       1       1       1       3       3       3       1       0
      1       0       0       1       0       1       2       2       2       4       4       4       2       1
      1       0       0       1       0       1       2       2       2       4       4       4       2       1
      2       1       1       0       1       2       3       3       3       5       5       5       3       2
      1       0       0       1       0       1       2       2       2       4       4       4       2       1
      2       1       1       2       1       0       2       3       3       5       5       5       3       2
      1       2       2       3       2       2       0       2       2       4       4       4       2       1
      1       2       2       3       2       3       2       0       0       2       2       2       0       1
      1       2       2       3       2       3       2       0       0       2       2       2       0       1
      3       4       4       5       4       5       4       2       2       0       0       0       2       3
      3       4       4       5       4       5       4       2       2       0       0       0       2       3
      3       4       4       5       4       5       4       2       2       0       0       0       2       3
      1       2       2       3       2       3       2       0       0       2       2       2       0       1
      0       1       1       2       1       2       1       1       1       3       3       3       1       0
diff
Following matrix shows the shortest distances between every pair of vertices :
      0       0       3       0       4       2       1       3       4       3       2       0       0       1
      0       0       3       0       6       4       1       3       4       3       2       0       0       1
      3       3       0       1       3       3       4       6       7       6       5       3       3       4
      0       0       1       0       4       4       1       3       4       3       2       0       0       1
      4       6       3       4       0       0       1       3       7       3       5       4       4       3
      2       4       3       4       0       0       0       1       5       1       3       2       2       1
      1       1       4       1       1       0       0       0       4       0       2       1       1       0
      3       3       6       3       3       1       0       0       4       0       2       4       5       2
      4       4       7       4       7       5       4       4       0       0       0       0       4       5
      3       3       6       3       3       1       0       0       0       0       2       4       4       2
      2       2       5       2       5       3       2       2       2       2       0       2       2       3
      0       0       3       0       4       2       1       4       0       4       2       0       0       1
      0       0       3       0       4       2       1       5       4       4       2       0       0       1
      1       1       4       1       3       1       0       2       5       2       3       1       1       0
```

STEP 4: SECURE THE LAST TASKS

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



The algorithm is not able to find a Hamiltonian path starting at point 0. So I had to add a bridge from admin(13) to com(5).

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

The graph theory problem is to find a Hamiltonian path in the graph starting from cafe(0).

3. Argue about an algorithm solving your problem.

```
HamCycle(map)
        path <- [-1] * map.V    #map.V is the number of vertices
        path[0] <- 0 #starting node
        if map.hamCycleUtil(path,1) == False
                return -1
        map.printSolution(path) # print the Hamiltonian cycle path
        return 0


HamCycleUtil(map,path,pos)
        If pos == map.V         #test if the path is completed
                If map.graph[ path[pos-1][path[0]] == 1       #test if the path makes a cycle
                        return True
                else
                        return False
        for v from 0 to map.V
                if map.isSafe(v,pos,path)  == True # test if current vertice is not already in the path
                        path[pos] <- v
                if map.hamCycleUtil(path,pos+1) == True #test if the next step is correct
                        return True
                path[pos] = -1
        return False
```

4. Implement the algorithm and show a solution.

Cf part4.py

```
In [107]: runfile('C:/Users/M BUE/Desktop/Esil
Solution Exists: Following is one Hamiltonian
0 12 11 8 10 9 7 6 13 5 4 2 3 1 0
```