# Step 1: To organize the tournament

### Propose a data structure to represent a Player and its Score

Nodes of the tree are structures or dictionary. We need this structure because in order to add nodes to the tree, it needs numerical values. But if we only give him numerical values, it is going to balance the tree just fine but afterwards we won’t be able to assign each score to its player. Therefore, we need this structure always linking the player and his score hence the dictionary (key:value)=(name:score) or a new structure player(int id, string name, int score)

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

Use AVL tree and not BST because the search complexity in AVL is always log2(n) while BST’s can reach a linear complexity of n (if all the new nodes are added on the right for example)

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

A simple randomizer without weighted score can be implemented. However, to keep the simulation as real as possible, we need to make scores like 10, 11 or 12 less probable than 4,5 or 6. We can use the technique of the dice increasing probability for average score and decreasing for extreme score (1,2,11,12)

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

Each player scored a certain amount of points, and depending this amount, its position in the tree is going to change with different rotations (right-right, right-left etc)

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

To create the 3 first random games, we can use a randomizer that is going to give ID to the players giving them a random number between 1 and 100, each number given once and only once. Then, we place the players in an AVL tree according to their ID (1 being on left and 100 on right). Now we can use three different type of search algorithm to manage that each player won’t play against the same players each 3 games. So first we have the in-order search algorithm, then the pre-order algorithm and finally the post-order algorithm. After each game we attribute a random score (between 0 and 12) that we add to their current score. At the end of the 3 random games, the 100 players will be ranked according to their score so that the tournament and the disqualification games can begin. (the score of each player is not reset after the 3 games, it wouldn’t be fair for top players).

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

Since players will be classed in increasing order in the tree and we want to organize games of ten players according to their ranking (player classed 1 to 10 play together, … 90 to 100 play together), all we need to do is to use an in order search on the AVL tree and to class players in a list. Then, each ten players, we create another game with the ten next players and so on.

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

With the previous method, we get a list of players classed in decreasing ranking order (from 100 to 1). Therefore, the ten first players of the list are dropped from it. We repeat this operation (game + update + delete) 9 times until we have 10 players remaining.

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

Then we reset the scores (set to 0) and organize 5 games (also meaning 5 updates/rotations on the AVL tree). With our final AVL tree, we just have to do an in-order tree to class the top 10 players and make the podium for the three best players.

# Step 2: Professor Layton < Guybrush Threepwood < Us

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

Use graph and representation matrix of a graph (CMO2 p25). Matrix of Boolean, n\*n with n the number of players. If the first player has seen the players 2, 4 and 5, then the value of the matrix is set to 1, same for and .

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

In our case, the player who has been seen the least time is more likely to be an impostor. Impostor also never walk with another impostor so if a player has seen every other player except one, we can deduce he is the impostor and the one player he didn’t see is an impostor too.

Mathematically, this corresponds to doing the sum of columns (or row since the matrix is symmetrical) of the matrix. The player whose column has the minimum score is probably an impostor. Likewise, if the sum equals the number of player alive minus the number of impostors, the player might be an impostor.

### Argue about an algorithm solving your problem.

Write the algorithm describing the above text.

### Implement the algorithm and show a solution.

Implement the previous algorithm

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

### Presents and argue about the two models of the map.

**First model : without vents**

Undirected graph : players can go in each rooms in both ways (from B to A and A to B), each way, aka edges, between rooms, aka nodes, is weighted by the amount of time needed to go from one room to another.

To calculate weights, we need to measure the distance between rooms connected to each other.

To estimate who is the impostor, we need to calculate the time a player put to go from one room to another. If this time is less than the weighted edge, then the player is an impostor.

**Second model : with vents**

This graph is a subgraph of the previous one, meaning we are going to take the same one and add edges to it. And the edges we are adding are the vents travels impostors can do between two rooms. Because the time to travel in vents is null, the weights on these new edges are all 0.

### Argue about a **pathfinding algorithm** to implement.

Matrix of size m\*m with m the number of rooms. The mij element of the matrix represents the minimum amount of time to travel from room i to room j.

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

Create a function that takes in parameter a depart room and an arrival room and returns the time of travel with the Dijkstra algorithm (finding the shortest paths between nodes in a graph. (no negative weights, directed or undirected graph, finite number of vertices, defined source))

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

Function using the previous function and calculate the eta for map 1 and 2 and keep the pairs of room where the eta is different.

# Step 4: Secure the last tasks

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

The map is going to be a subgraph of the precedent graph. Indeed, at the end of the game, only few tasks are remaining and rooms are no longer needed because no tasks are to be accomplished in those rooms. Therefore, we need to delete nodes (corresponding to rooms with no tasks), and create new edges weighted with the sum of the previous edges.

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

We are going to use an algorithm of the commerce traveler. Finding the shortest or fastest way/itinerary between multiple tasks remaining.

To do so, we use the Hamiltonian path (cf CMO3) : undirected graph, each place must be visited once and only once (visit every vertex/node exactly once)

### Argue about an algorithm solving your problem.

Explain functioning of Hamiltonian algorithm

### Implement the algorithm and show a solution.

Implement previous algorithm with python.