OPTIMIZATION THEORY

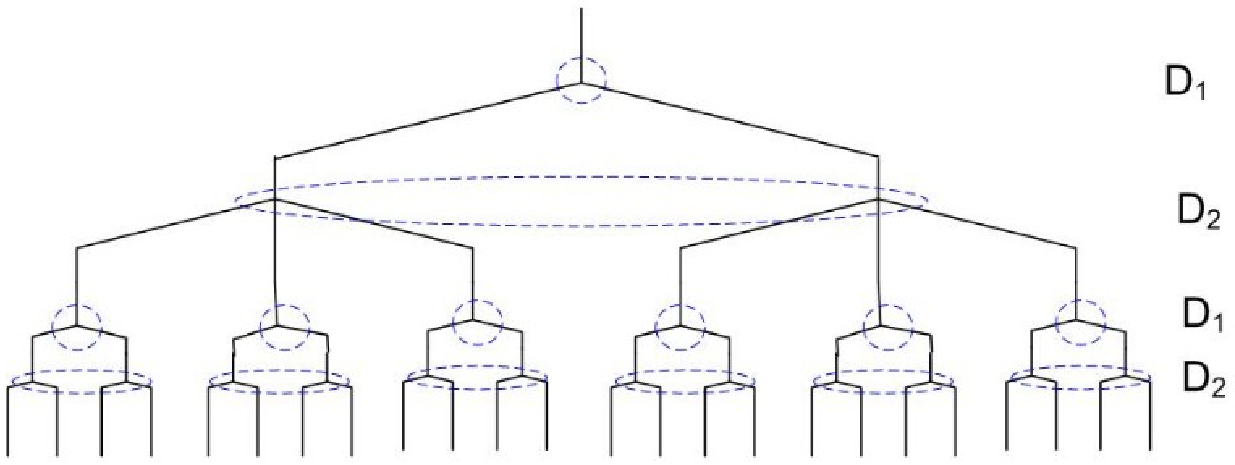
DECISION TREES

Sergio Barbero, Maaz Ashiq

Introduction to the problem

We have to implement the solution for a given structure of decision tree, we decided make ours from a zero-sum game.

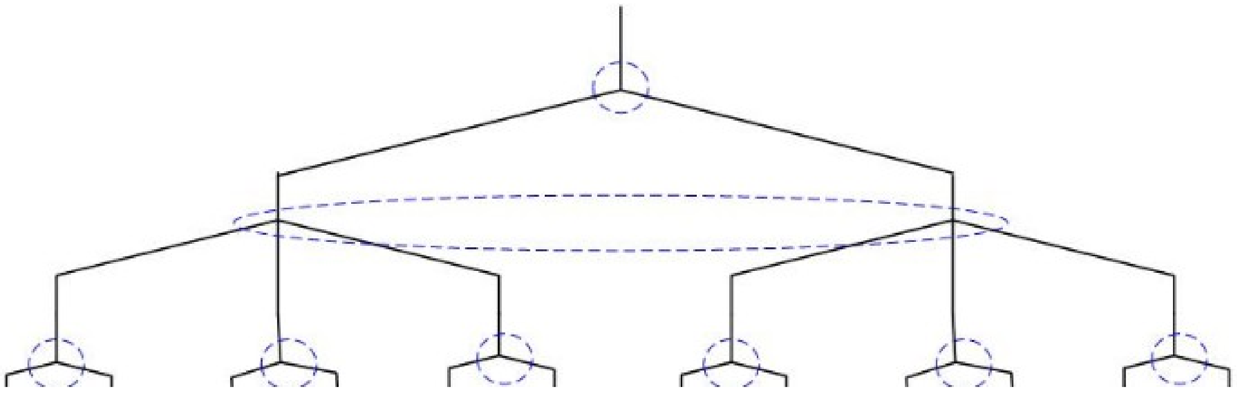
This is the given structure



In our example D1 wants to minimize and D2 to maximize the results

Implementation

This problem is actually several decision trees joined, we can distinguish 2 kinds of decision trees in our example:



1. The top of the game:

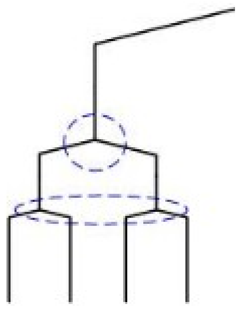
The player D1 (on the top) can’t recognize the movement of the player D2 (under D1), so the matrix will be something like this:

D1\D2	Left	Center	Right
Left			
Right			

The data of this table will come from the next decision trees.

2. The below trees

In this case we have 6 decision trees like this, we have to solve every tree separately, the top of the tree is the player D1 and under it the player D2, the player D1 can't see the movement of the player D1 like before.



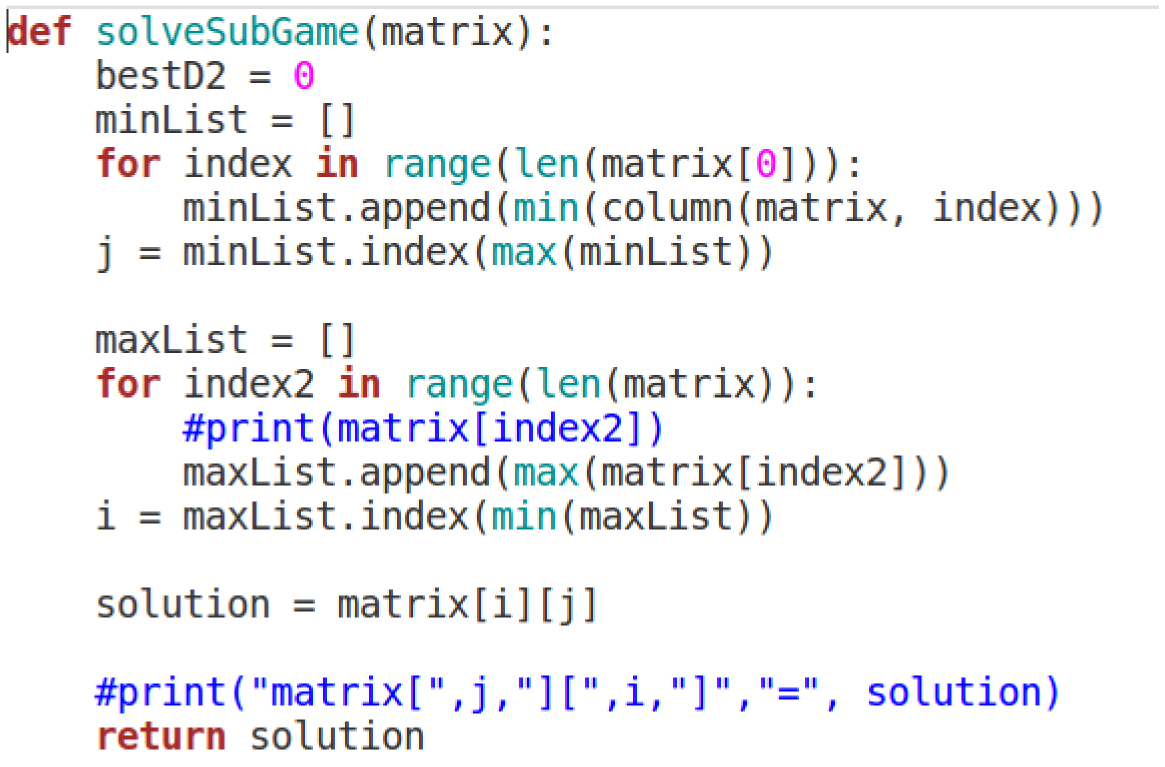
In this case our matrix will have this structure:

D1\D2	Left	Right
Left		
Right		

In order to solve the problem we just need one single function which solves one single tree, no matter how our tree to solve is, so we will invoke to this function whether for our first or second kind of tree.

Furthermore, this function will work for every kind of tree, not only the indicated ones in such a problem.

Let's see the code of such a funcion:



```
def solveSubGame(matrix):
    bestD2 = 0
    minList = []
    for index in range(len(matrix[0])):
        minList.append(min(column(matrix, index)))
    j = minList.index(max(minList))

    maxList = []
    for index2 in range(len(matrix)):
        #print(matrix[index2])
        maxList.append(max(matrix[index2]))
    i = maxList.index(min(maxList))

    solution = matrix[i][j]

    #print("matrix[",j,"][",i,"]","=", solution)
    return solution
```

As we can observe, in our first loop we choose the safe colum for player D2, and in our second loop the same for the player D1 (rows in spite of colums), therefore, our solution is the union of these indexes into our matrix.

So in order to solve the full problem we must call this function as many times as needed.

First of all we call the function 6 times in order to solve the below trees and we almacenate them into variables.

We will create a new matrix called “newTree” with the values returned previously.

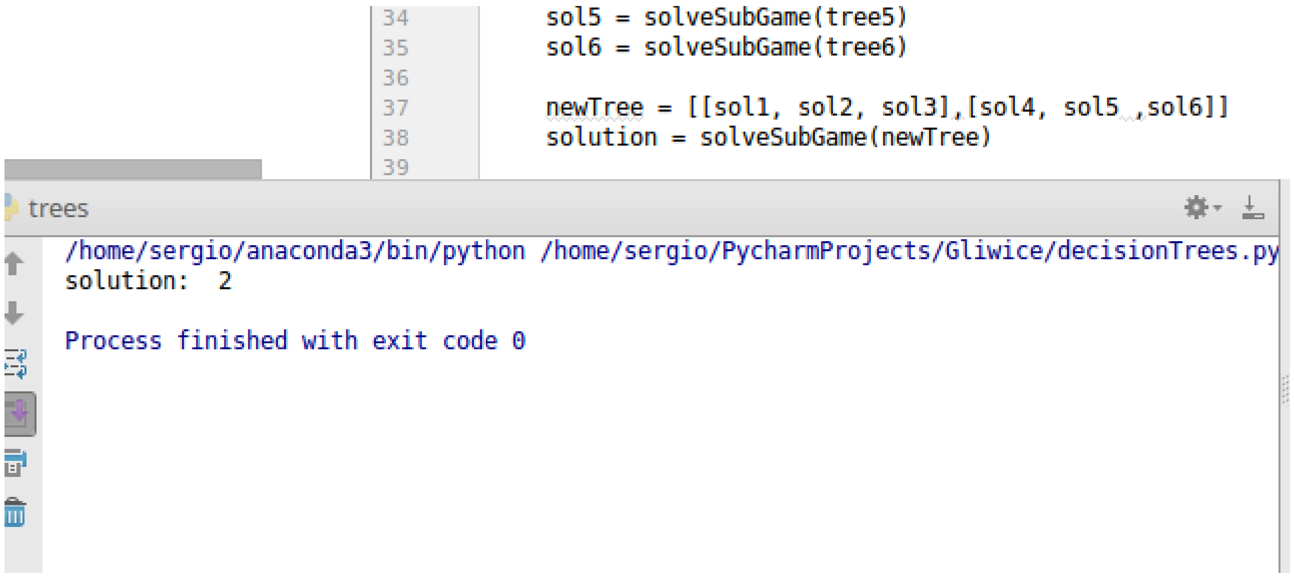
Now we will use this matrix as an input of the function. This last and new call will give us the answer to the full decision tree.

```
def solveGame(list): #5th structure game
    tree1 = [[list[0], list[1]],[list[2], list[3]]]
    tree2 = [[list[4], list[5]],[list[6], list[7]]]
    tree3 = [[list[8], list[9]],[list[10], list[11]]]
    tree4 = [[list[12], list[13]],[list[14], list[15]]]
    tree5 = [[list[16], list[17]],[list[18], list[19]]]
    tree6 = [[list[20], list[21]],[list[22], list[23]]]

    sol1 = solveSubGame(tree1)
    sol2 = solveSubGame(tree2)
    sol3 = solveSubGame(tree3)
    sol4 = solveSubGame(tree4)
    sol5 = solveSubGame(tree5)
    sol6 = solveSubGame(tree6)

    newTree = [[sol1, sol2, sol3],[sol4, sol5 ,sol6]]
    solution = solveSubGame(newTree)

    print("solution: ", solution)

    return solution

leaves = [3, -2, -1, 0, 4, 6, 2, -1, 5, 3, -2, 6, 3, 5, -4, 15, 8, 10, 2, 3, 0, 2, -2, -4]
solveGame(leaves)
```

For an easier introduction of data we will use a list called “leaves”, this refers to the leaves nodes of our problem, if we want to change the data we just need to modify the values of this list.

Results

There is not that much to tell here, the program will return the result of our decision tree:



```
    sol5 = solveSubGame(tree5)
    sol6 = solveSubGame(tree6)

    newTree = [[sol1, sol2, sol3],[sol4, sol5 ,sol6]]
    solution = solveSubGame(newTree)
```

trees

```
/home/sergio/anaconda3/bin/python /home/sergio/PycharmProjects/Gliwice/decisionTrees.py
solution:  2

Process finished with exit code 0
```