DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC-350 Artificial Intelligence and Decision Support System

LAB MANUAL – 08

Course Instructor: Assoc Prof Dr Arsalan Shaukat

Lab Engineer: Kashaf Raheem

Student Name: _____

Degree/ Syndicate: _____

LAB # 8: MINIMAX WITH ALPHA BETA PRUNING

Lab Objective:

- To implement genetic algorithm in python

Hardware/Software required:

Hardware: Desktop/ Notebook Computer
Software Tool: Python 3.10.0

Lab Description:

Minimax is a decision rule used in decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario. When dealing with gains, it is referred to as "maximin"—to maximize the minimum gain.

Use: Originally formulated for two-player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision-making in the presence of uncertainty.

```
Node= Node of which MinMax value to be found

Depth: depth of tree

MaximizingPlayer= True  (IF THE NODE IS MAX NODE)

MaximizingPlayer= False (IF THE NODE IS MIN NODE)
```

The code for defining Node of tree:

```
class Node:

    def __init__(self,val):

        self.left = None

        self.right = None

        self.data = val
```

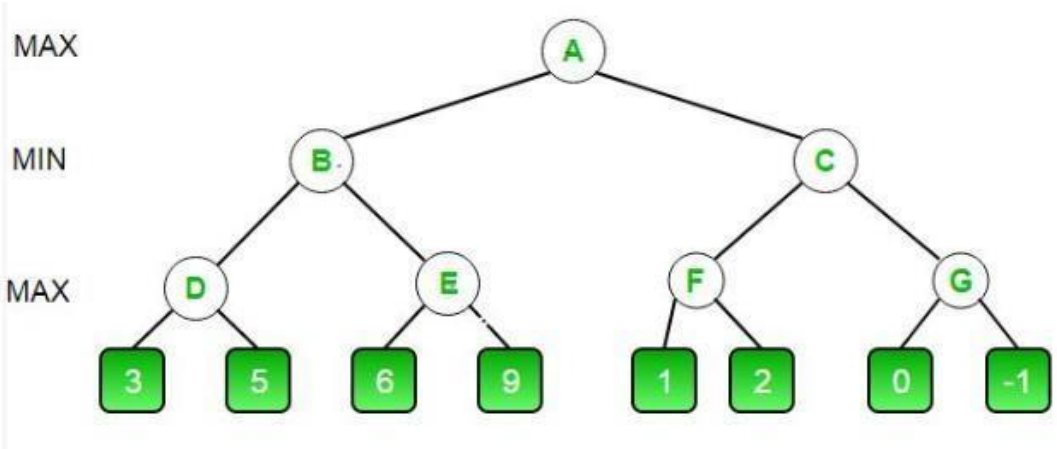
The pseudocode for minimax searching is:

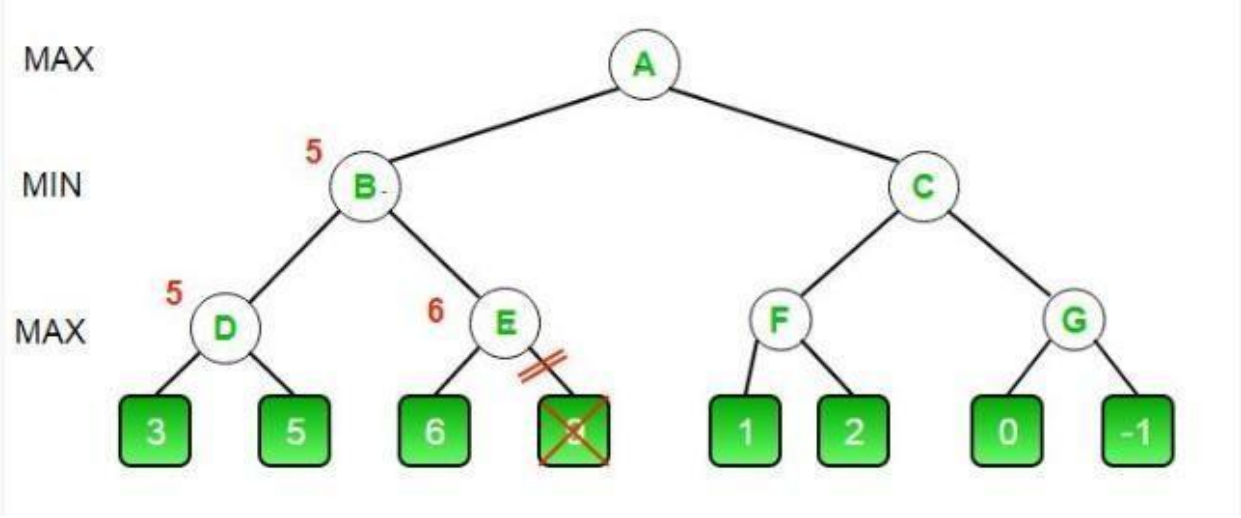
```
function minimax(node, depth, maximizingPlayer)
    if depth = max or node is a terminal node
        return the heuristic value of node
```

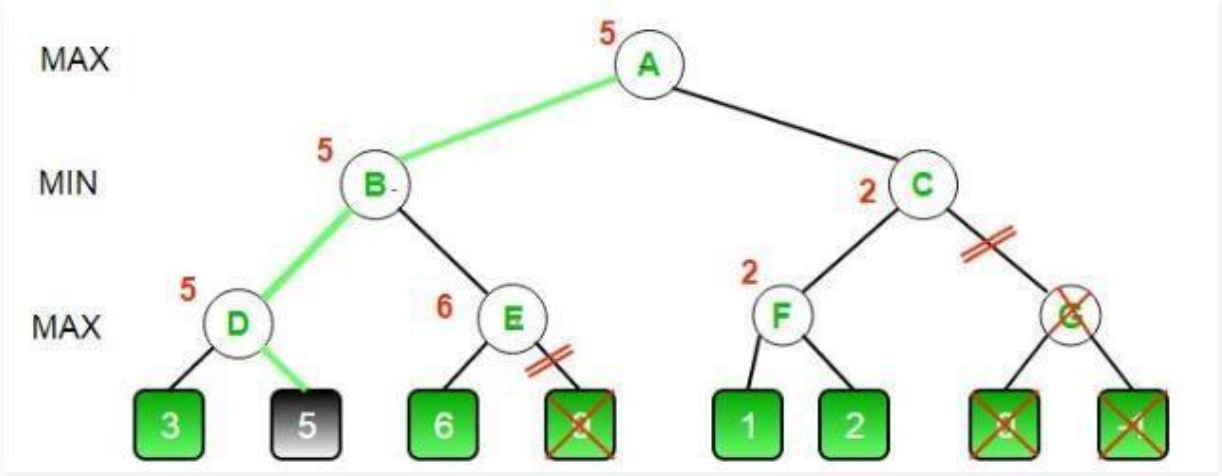
```
    if maximizingPlayer
        bestValue := −∞
        for each child of node
            v := minimax(child, depth + 1, FALSE)
            bestValue := max(bestValue, v)
        return bestValue

    else    (* minimizing player *)
        bestValue := +∞
        for each child of node
            v := minimax(child, depth + 1, TRUE)
            bestValue := min(bestValue, v)
        return bestValue

minimax(origin, depth, TRUE)
```

Minimax with Alpha-Beta Pruning:

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.







The pseudocode for the minimax with alpha-beta pruning is:

```
function alphabeta(node, depth, α, β, maximizingPlayer)
    if depth = max
        return the heuristic value of node

    if maximizingPlayer
        v := -∞
        for each child of node
            v := max(v, alphabeta(child, depth + 1, α, β, FALSE))
            α := max(α, v)
            if β ≤ α
                break (* β cut-off *)
```

```
        return v

    else
        v := +∞
        for each child of node
            v := min(v, alphabeta(child, depth +1, α, β, TRUE))
            β := min(β, v)
            if β ≤ α
                break (* α cut-off *)
        return v

alphabeta(origin, depth, -∞, +∞, TRUE)
```

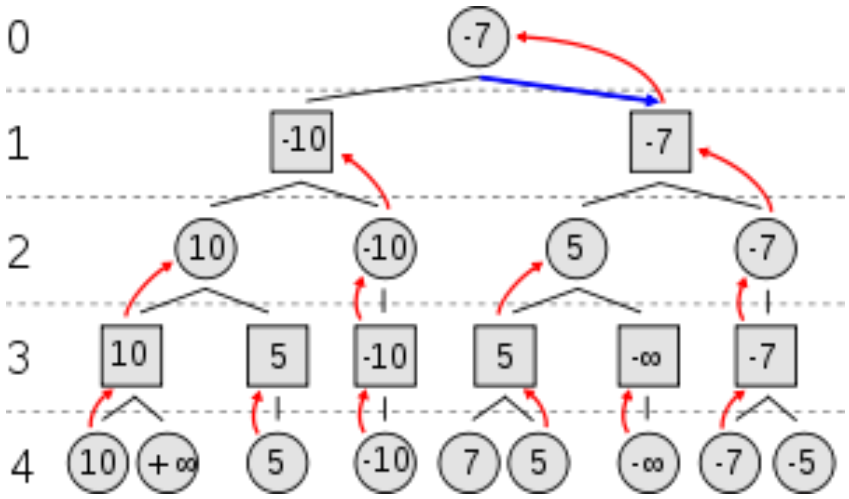
Note: We are not covering the case in which only terminal values are given so we'll have to find the values of nodes at the runtime.

Lab Tasks:

1. Implement minimax algorithm for Tree 1 to find the MinMAx value of the given node, Import the time module and calculate the total time taken by your minimax algorithm.

2. Implement minimax with alpha-beta pruning for Tree 1.

3. Compare the execution time of Question # 2 with Question # 1.



Tree 1