Experiment No.5
Study and Implementation of Min-Max algorithm.
Date of Performance:
Date of Submission:

Aim: Study and Implementation of Min-Max algorithm.

Objective: To study the adversarial search technique and its implementation in game playing.
Theory:

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.
In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.
Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Mini-Max Algorithm in Artificial Intelligence

- In decision-making and game theory, the mini-max algorithm is a recursive or backtracking method. It suggests the best move for the player, provided that the opponent is likewise playing well.
- In AI, the Min-Max algorithm is mostly employed for game play. Chess, checkers, tic-tac-toe, go, and other two-player games are examples. This Algorithm calculates the current state's minimax choice.
- The game is played by two players, one named MAX and the other named MIN, in this algorithm.
- Both players FIGHT it, since the opponent player receives the smallest benefit while they receive the greatest profit.
- Both players in the game are adversaries, with MAX selecting the maximum value and MIN selecting the minimum value.
- For the exploration of the entire game tree, the minimax method uses a depth-first search strategy.
- For the exploration of the entire game tree, the minimax method uses a depth-first search strategy.
- The minimax algorithm descends all the way to the tree's terminal node, then recursively backtracks the tree.

Pseudo-code for MinMax Algorithm:

```
 import math

def minimax (curDepth, nodeIndex,
                      maxTurn, scores,
                      targetDepth):

       if (curDepth == targetDepth):
              return scores[nodeIndex]

       if (maxTurn):
              return max(minimax(curDepth + 1, nodeIndex * 2,
                                    False, scores, targetDepth),
                             minimax(curDepth + 1, nodeIndex * 2 + 1,
                                    False, scores, targetDepth))

       else:
              return min(minimax(curDepth + 1, nodeIndex * 2,
                                    True, scores, targetDepth),
                             minimax(curDepth + 1, nodeIndex * 2 + 1,
                                    True, scores, targetDepth))

scores = [-1,4,2,6,-3,-5,0,7]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")
print(minimax(0, 0, True, scores, treeDepth))


output:

The optimal value is : 4
```

Initial call:

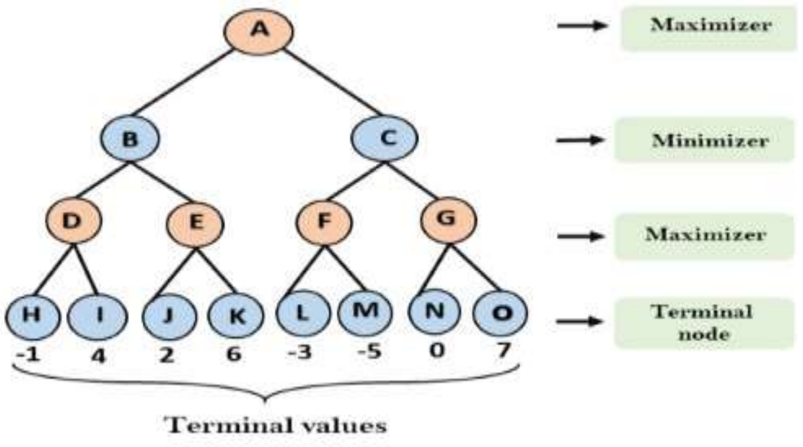
minimax(node, 3, true)

Working of Min-Max Algorithm:

- A simple example can be used to explain how the minimax algorithm works. We've included an example of a game-tree below, which represents a two-player game.
- There are two players in this scenario, one named Maximizer and the other named Minimizer.
- Maximizer will strive for the highest possible score, while Minimizer will strive for the lowest possible score.
- Because this algorithm uses DFS, we must go all the way through the leaves to reach the terminal nodes in this game-tree.

- The terminal values are given at the terminal node, so we'll compare them and retrace the tree till we reach the original state. The essential phases in solving the two-player game tree are as follows:

Step 1: The method constructs the whole game-tree and applies the utility function to obtain utility values for the terminal states in the first step. Let's assume A is the tree's initial state in the diagram below. Assume that the maximizer takes the first turn with a worst-case initial value of -infinity, and the minimizer takes the second turn with a worst-case initial value of +infinity.

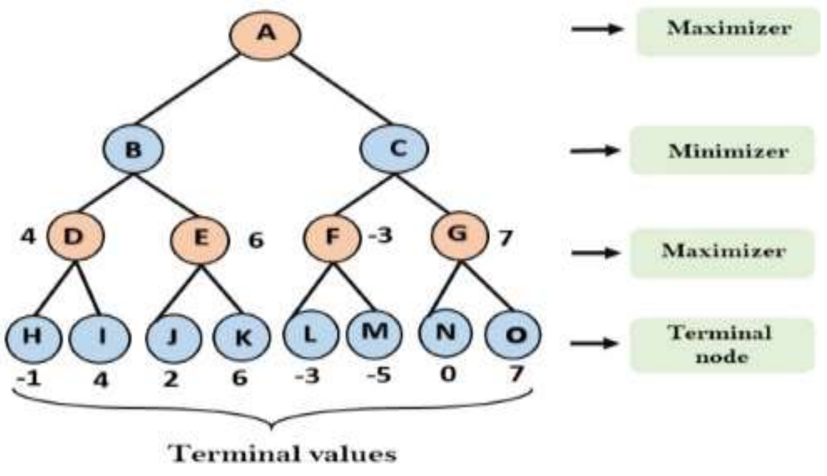


Example: Working of Min-Max Algorithm

step-1

Step 2: Next, we'll locate the Maximizer's utilities value, which is -, and compare each value in the terminal state to the Maximizer's initial value to determine the upper nodes' values. It will select the best option from all of them.

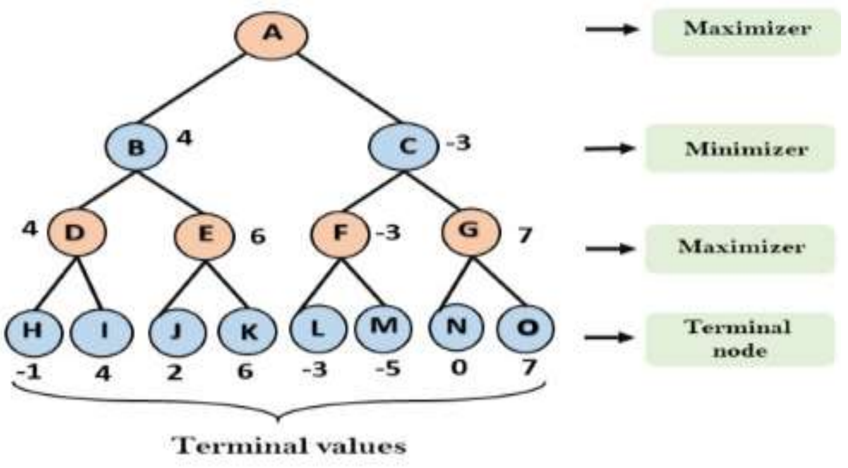
- **For node D max(-1,- -∞) => max(-1,4)= 4**
- **For Node E max(2, -∞) => max(2, 6)= 6**
- **For Node F max(-3, -∞) => max(-3,-5) = -3**
- **For node G max(0, -∞) = max(0, 7) = 7**



Step-2

Step 3: Now it's the minimizer's time, thus it'll compare all nodes' values with + and determine the 3rd layer node values.

- **For node B = min(4,6) = 4**
- **For node C = min (-3, 7) = -3**

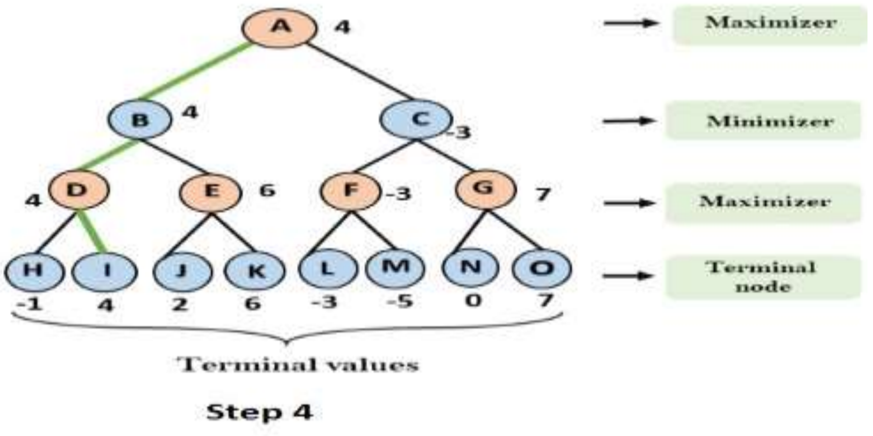


Step 3

In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α = -∞, and β = 3 will also be passed.

Step 4: Now it's Maximizer's turn, and it'll choose the maximum value of all nodes and locate the root node's maximum value. There are only four layers in this game tree, so we can go to the root node right away, but there will be more layers in real games.

For node A max(4, -3)= 4



Properties of Mini-Max algorithm:

- **Complete –**The Min-Max algorithm is finished. In the finite search tree, it will undoubtedly locate a solution (if one exists).
- **Optimal-** If both opponents are playing optimally, the Min-Max algorithm is optimal.
- **Time complexity-** Because it executes DFS for the game-tree, the time complexity of the Min-Max algorithm is **O(b^m)**, where b is the game-branching tree's factor and m is the tree's maximum depth.
- **Space Complexity-** Mini-max method has a space complexity that is similar to DFS, which is **O(bm)**.

Limitation of the minimax Algorithm:

The biggest disadvantage of the minimax algorithm is that it becomes extremely slow while playing complex games like chess or go. This style of game contains a lot of branching, and the player has a lot of options to choose from. The minimax algorithm's drawback can be alleviated by using **alpha-beta pruning**, which we will explore in the next section. the depth to which the tree can grow.

Conclusion: In conclusion, the implementation of the Minimax algorithm equips systems with the ability to make strategic decisions that maximize their chances of success, even in complex and competitive environments. However, it's important to note that Minimax's effectiveness heavily depends on factors such as the depth of the search tree, the accuracy of the evaluation function, and the computational resources available. Therefore, while Minimax serves as a foundational algorithm, its performance can be enhanced through optimizations such as alpha-beta pruning, heuristic evaluations, and parallelization techniques.