MINIMAX ALGORITHM

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.
- The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning**

The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.

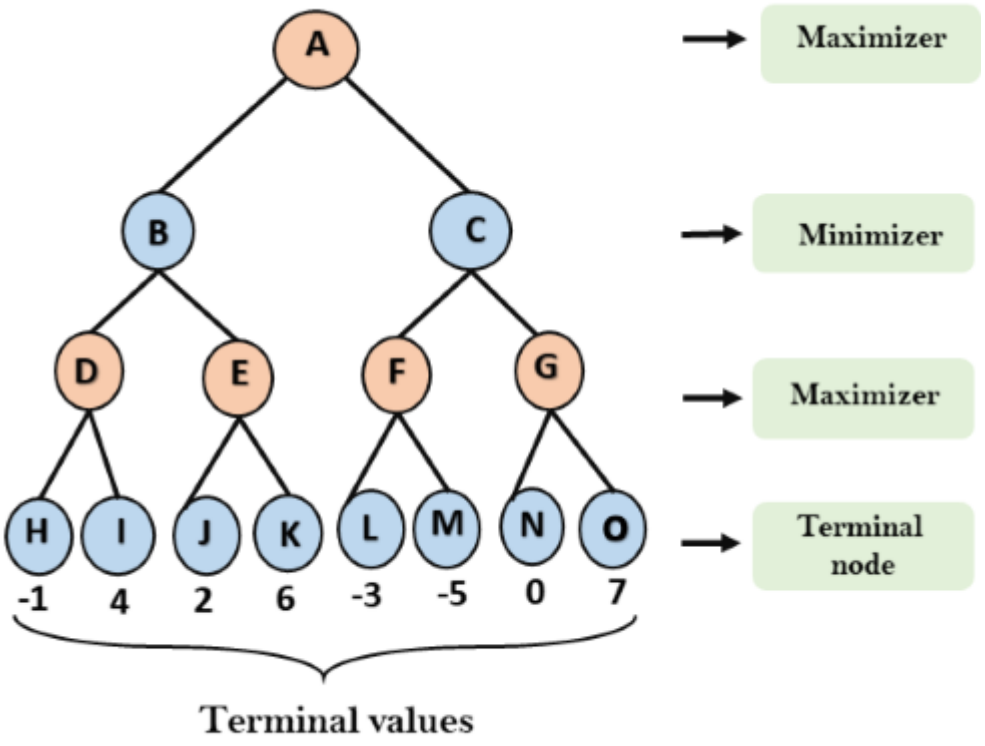
In this example, there are two players one is called Maximizer and other is called Minimizer.

Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.

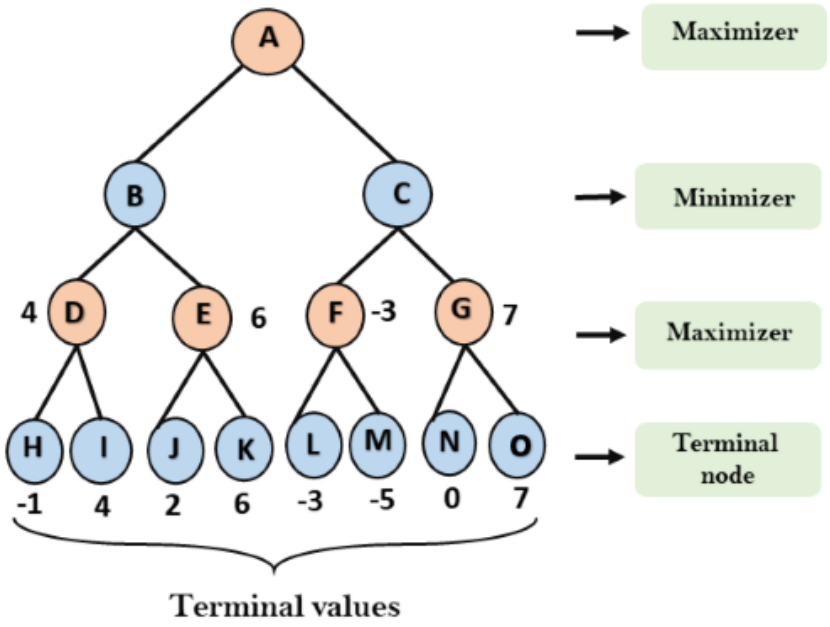
This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.

At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

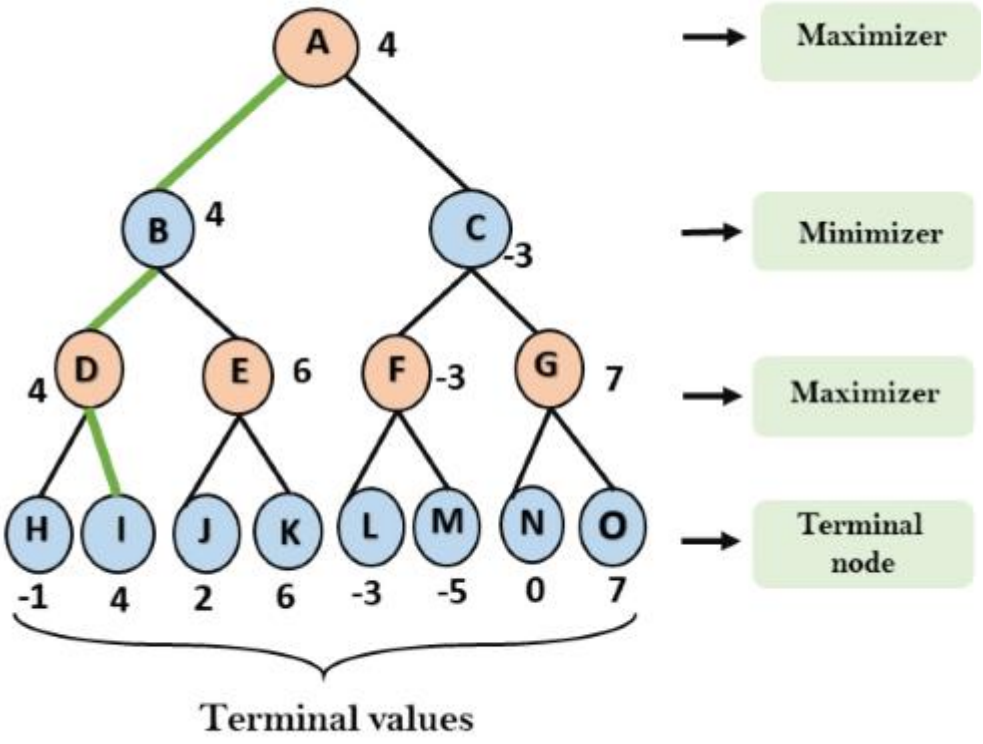
STEP 1



STEP 2


A
B
C
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

STEP 3


A
4
B
4
C
-3
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

CODE

```
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))
# Driver code
scores = [3, 5, 2, 9, 12, 5, 23, 23]


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


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