That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

Complete- Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.

Optimal- Min-Max algorithm is optimal if both opponents are playing optimally.

Time complexity- As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is O(bm), where b is the branching factor of the game-tree, and m is the maximum depth of the tree.

Space Complexity- Space complexity of Minimax algorithm is also similar to DFS which is O(bm).

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of game 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

• Alpha-Beta Pruning

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.

As we have seen in the minimax search algorithm, the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameters Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called the Alpha-Beta Algorithm.

Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prunes the tree leaves but also the entire sub-tree.

The two-parameter can be defined as:

Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.

Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making the algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

α>=β

Key points about alpha-beta pruning:

- ➔ The Max player will only update the value of alpha.
- ➔ The Min player will only update the value of beta.
- ➔ While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- ➔ We will only pass the alpha, beta values to the child nodes.

Pseudo-code for Alpha-beta Pruning:

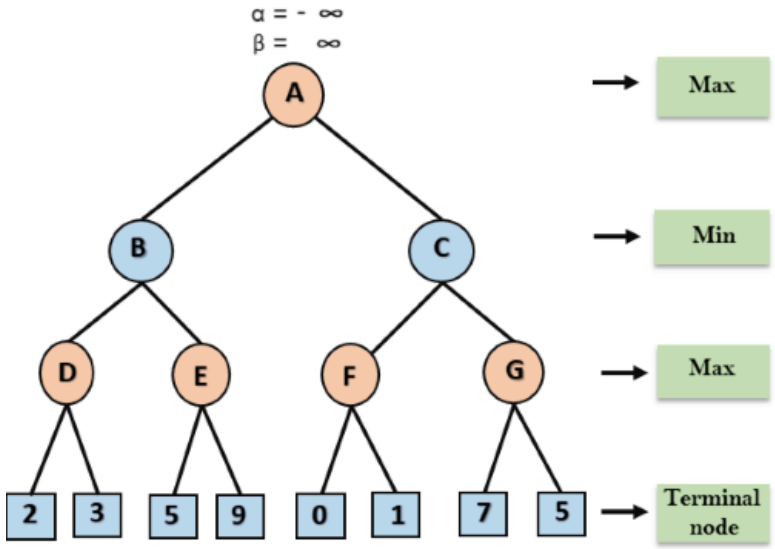
```
function minimax(node, depth, alpha, beta, maximizingPlayer) is

if depth ==0 or node is a terminal node then

return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player

  maxEva= -infinity

  for each child of node do

  eva= minimax(child, depth-1, alpha, beta, False)
```

```
 maxEva= max(maxEva, eva)
 alpha= max(alpha, maxEva)
  if beta<=alpha
break
return maxEva
else                        // for Minimizer player
   minEva= +infinity
   for each child of node do
   eva= minimax(child, depth-1, alpha, beta, true)
   minEva= min(minEva, eva)
   beta= min(beta, eva)
    if beta<=alpha
 break
return minEva
```

Working of Alpha-Beta Pruning:

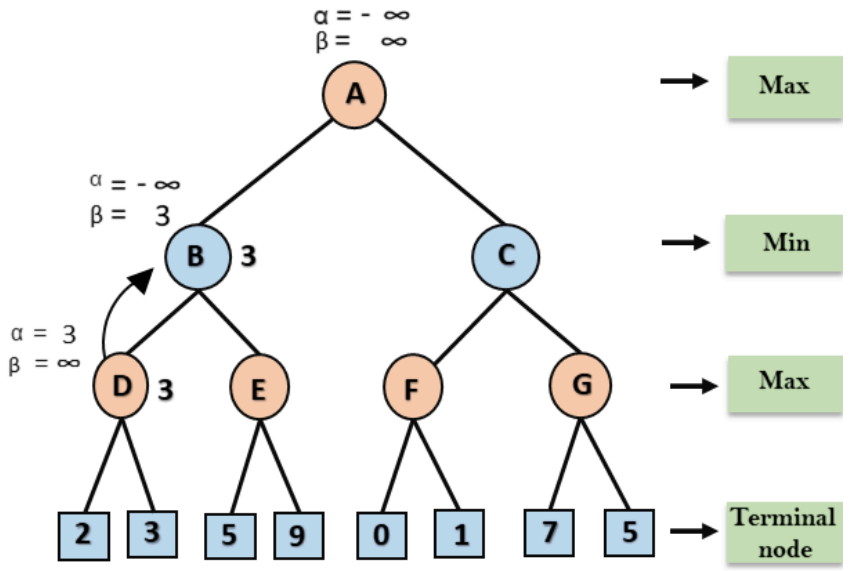
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



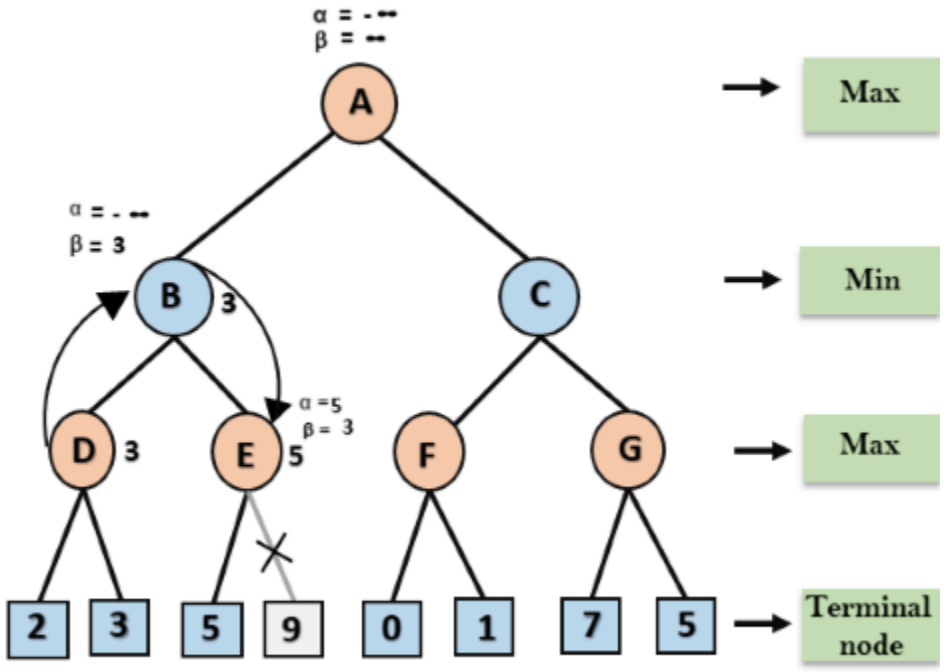
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also be 3

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



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

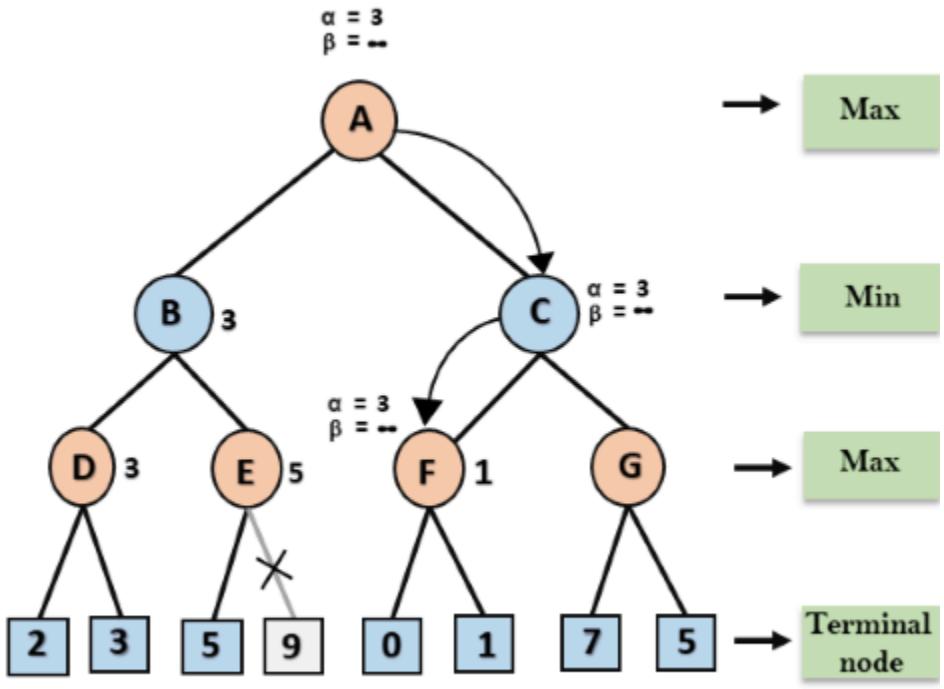
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



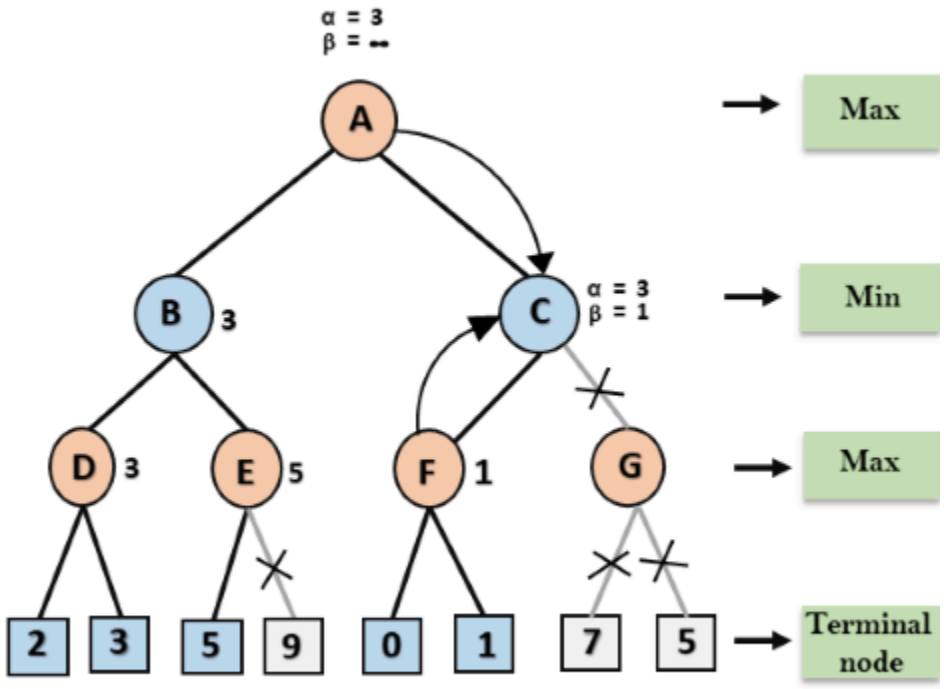
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

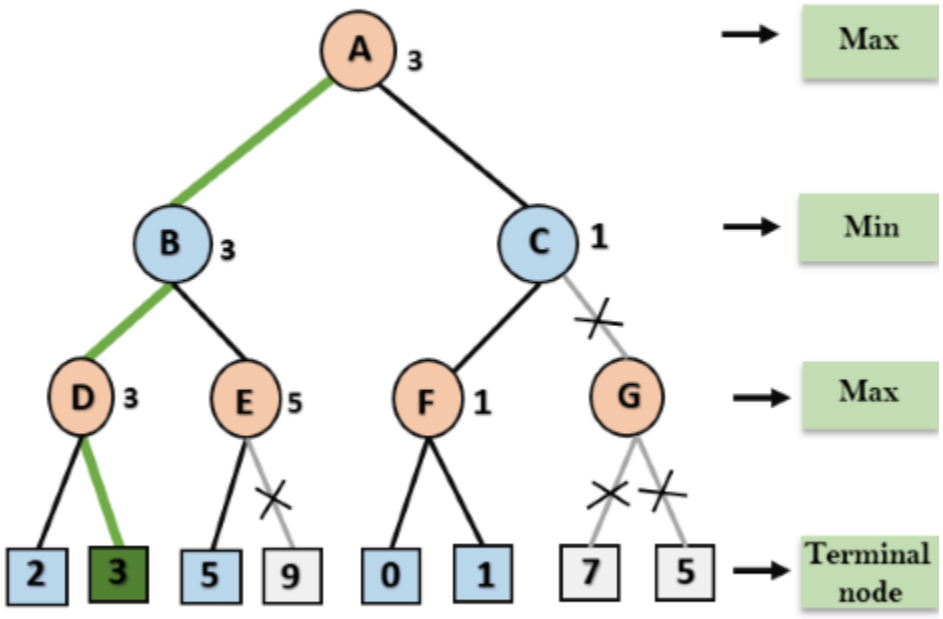
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire subtree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which shows the nodes which are computed and nodes which have never computed. Hence the optimal value for the maximizer is 3 for this example.


A
3
B
3
C
1
D
3
E
5
F
1
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node