Minimax Algorithm in Game Theory | (Alpha-Beta Pruning)

The word 'pruning' means cutting down branches and leaves. Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.

Define the parameters alpha and beta as follows:
Alpha is the best value that the **maximizer** currently can guarantee at that level or above.
Beta is the best value that the **minimizer** currently can guarantee at that level or above.

Condition for Alpha-beta pruning

- Alpha: At any point along the Maximizer path, Alpha is the best option or the highest value we've discovered. The initial value for alpha is $-\infty$.
- Beta: At any point along the Minimizer path, Beta is the best option or the lowest value we've discovered.. The initial value for alpha is $+\infty$.
- The condition for Alpha-beta Pruning is that $\alpha >= \beta$.
- The alpha and beta values of each node must be kept track of. Alpha can only be updated when it's MAX's time, and beta can only be updated when it's MIN's turn.
- MAX will update only alpha values and the MIN player will update only beta values.
- The node values will be passed to upper nodes instead of alpha and beta values during going into the tree's reverse.
- Alpha and Beta values only are passed to child nodes.

Minimax algorithm

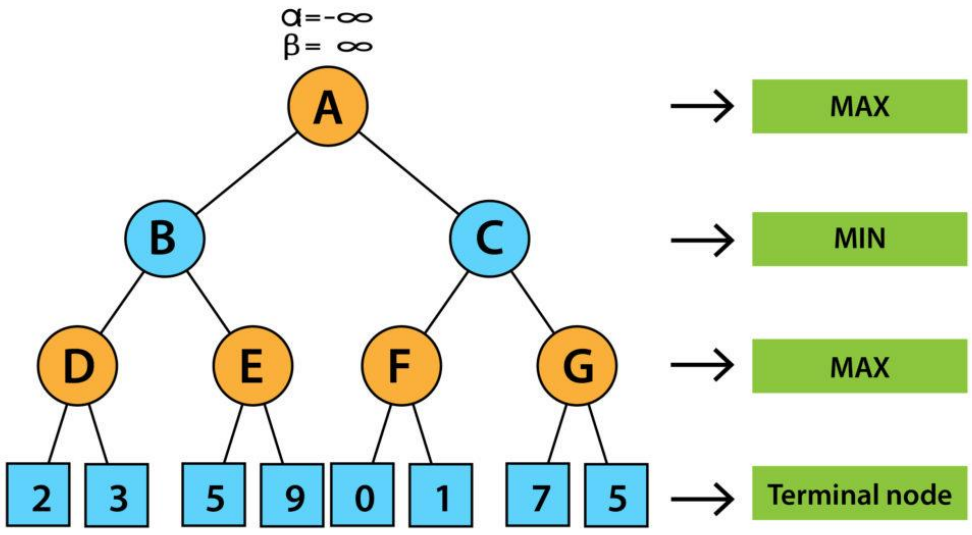
Minimax is a classic depth-first search technique for a sequential two-player game. The two players are called MAX and MIN. The minimax algorithm is designed for finding the optimal move for MAX, the player at the root node. The search tree is created by recursively expanding all nodes from the root in a depth-first manner until either the end of the game or the maximum search depth is reached.

Pseudocode :

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal

// Calling the function for the first time.
minimax(0, 0, true, -INFINITY, +INFINITY)
```

Working of Alpha-beta Pruning

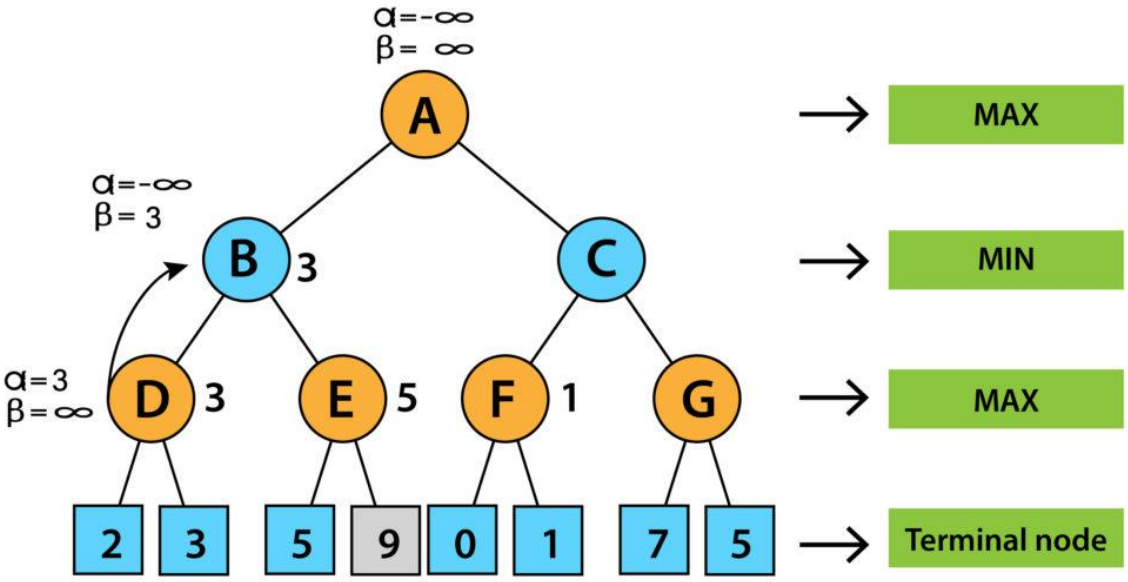
1. We will first start with the initial move. We will initially define the alpha and beta values as the worst case i.e. $\alpha = -\infty$ and $\beta = +\infty$. We will prune the node only when alpha becomes greater than or equal to beta.



2. Since the initial value of alpha is less than beta so we didn't prune it. Now it's turn for MAX. So, at node D, value of alpha will be calculated. The value of alpha at node D will be max (2, 3). So, value of alpha at node D will be 3.

3. Now the next move will be on node B and its turn for MIN now.

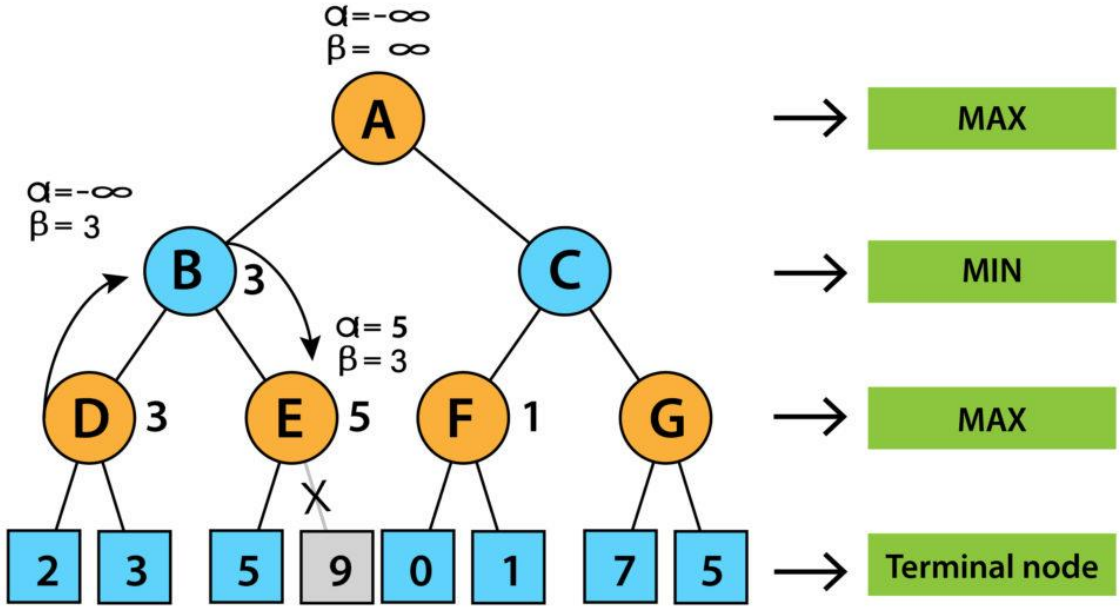
So, at node B, the value of alpha beta will be min (3, ∞). So, at node B values will be alpha= – ∞ and beta will be 3.



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

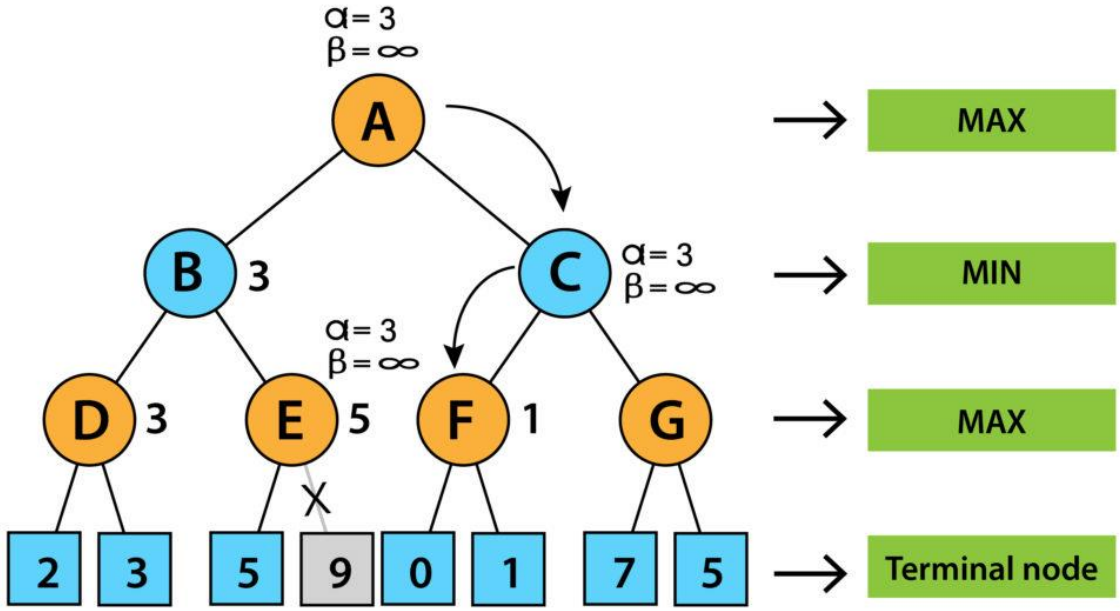
4. Now it's turn for MAX. So, at node E we will look for MAX. The current value of alpha at E is – ∞ and it will be compared with 5. So, MAX (- ∞, 5) will be 5. So, at node E, alpha = 5, Beta = 5.

Now as we can see that alpha is greater than beta which is satisfying the pruning condition so we can prune the right successor of node E and algorithm will not be traversed and the value at node E will be 5.

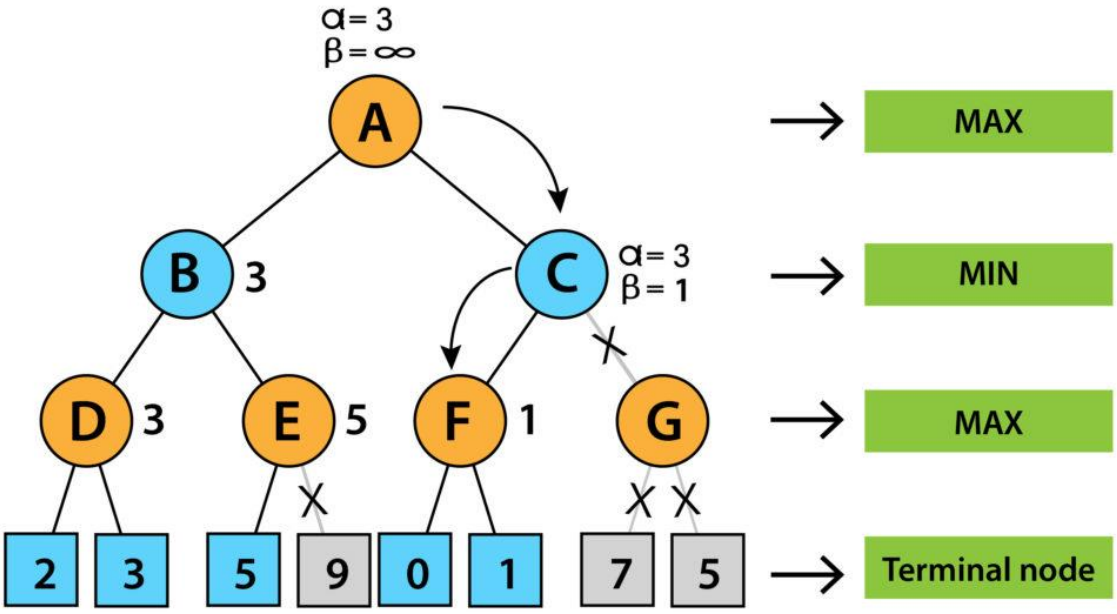


6. In the next step the algorithm again comes to node A from node B. At node A alpha will be changed to maximum value as MAX (- ∞, 3). So now the value of alpha and beta at node A will be (3, + ∞) respectively and will be transferred to node C. These same values will be transferred to node F.

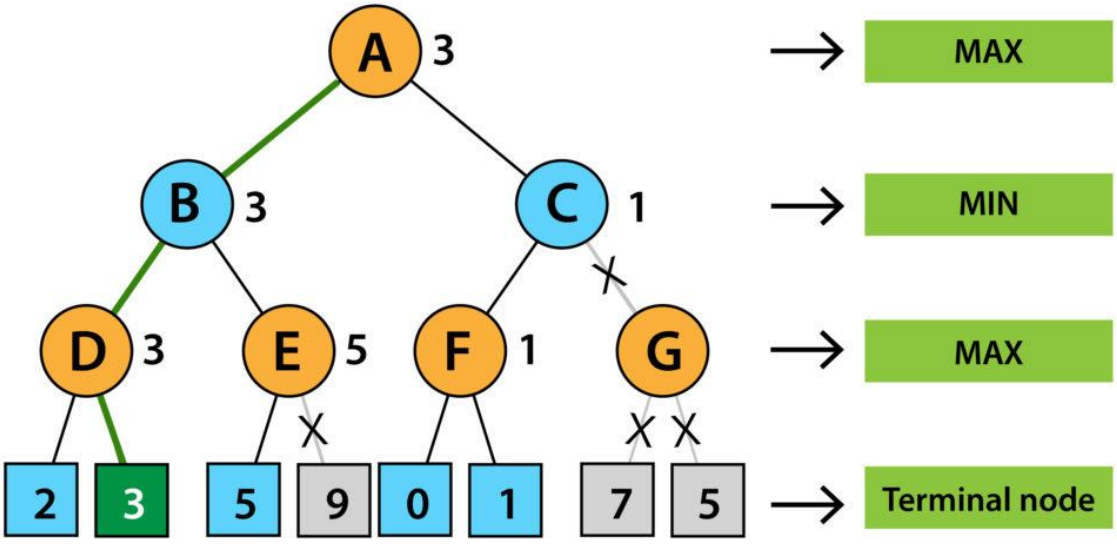
7. At node F the value of alpha will be compared to the left branch which is 0. So, MAX(0, 3) will be 3 and then compared with the right child which is 1, and MAX (3,1) = 3 still α remains 3, but the node value of F will become 1.



8. Now node F will return the node value 1 to C and will compare to beta value at C. Now its turn for MIN. So, MIN (+ ∞, 1) will be 1. Now at node C, α= 3, and β= 1 and alpha is greater than beta which again satisfies the pruning condition. So, the next successor of node C i.e. G will be pruned and the algorithm didn't compute the entire subtree G.



Now, C will return the node value to A and the best value of A will be MAX (1, 3) will be 3.



The above represented tree is the final tree which is showing the nodes which are computed and the nodes which are not computed. So, for this example the optimal value of the maximizer will be 3.