Artificial Intelligence

- VBDS1402
- **MODULE 1 CHAPTER 3: Adversarial Search & Logical Agents**
- **Topic: Alpha-Beta Pruning**

In this session you will learn

- Alpha-Beta Pruning
- Parameter
- Condition for Alpha-beta pruning
- Example

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 that 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 parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.

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

Parameters (2 parameters)

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 **-∞**.

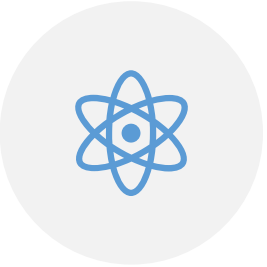
Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

Condition for Alpha-beta pruning

The main condition which required for alpha-beta pruning is: α>=β

The Max player will only update the value of alpha.

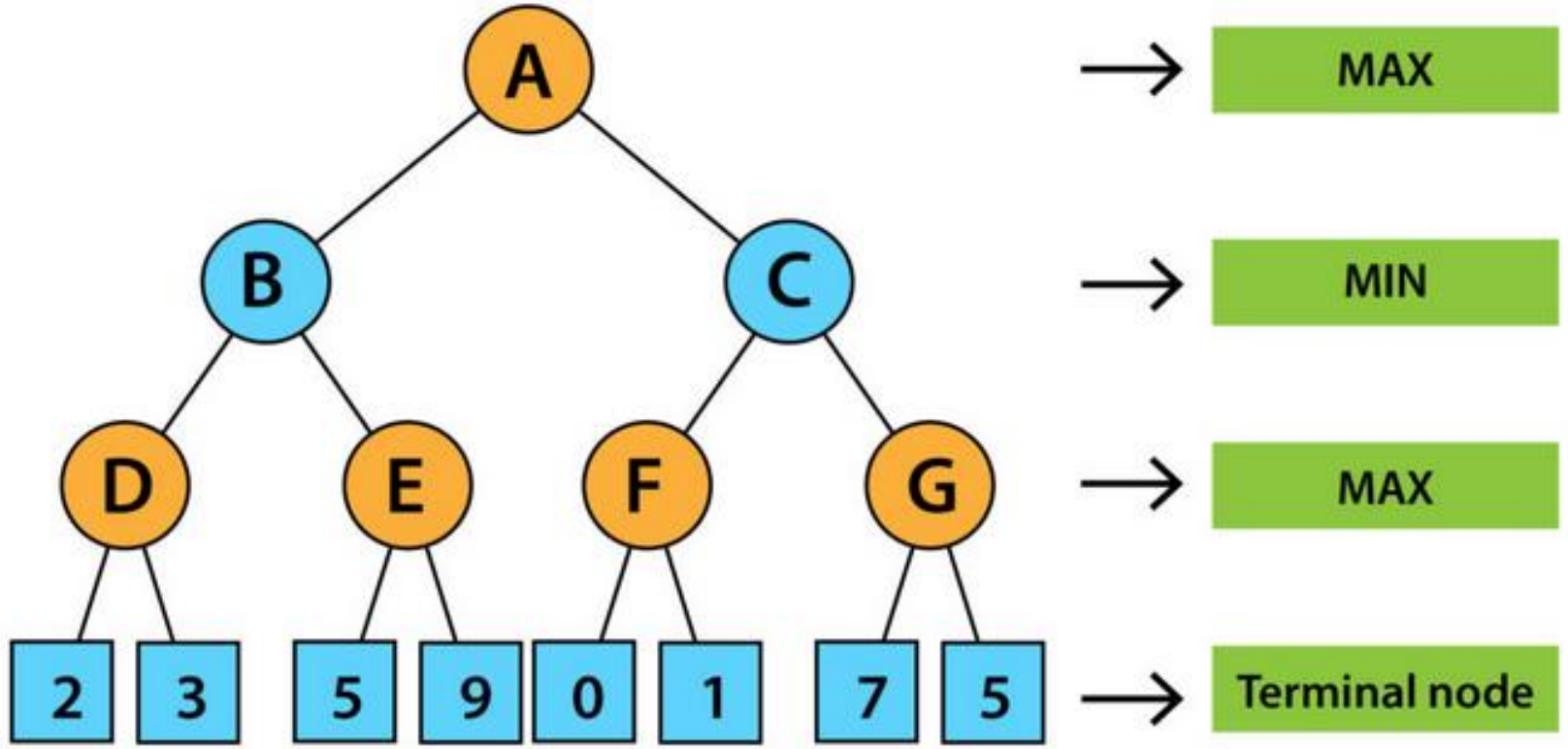
The Min player will only update the value of beta.



While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.

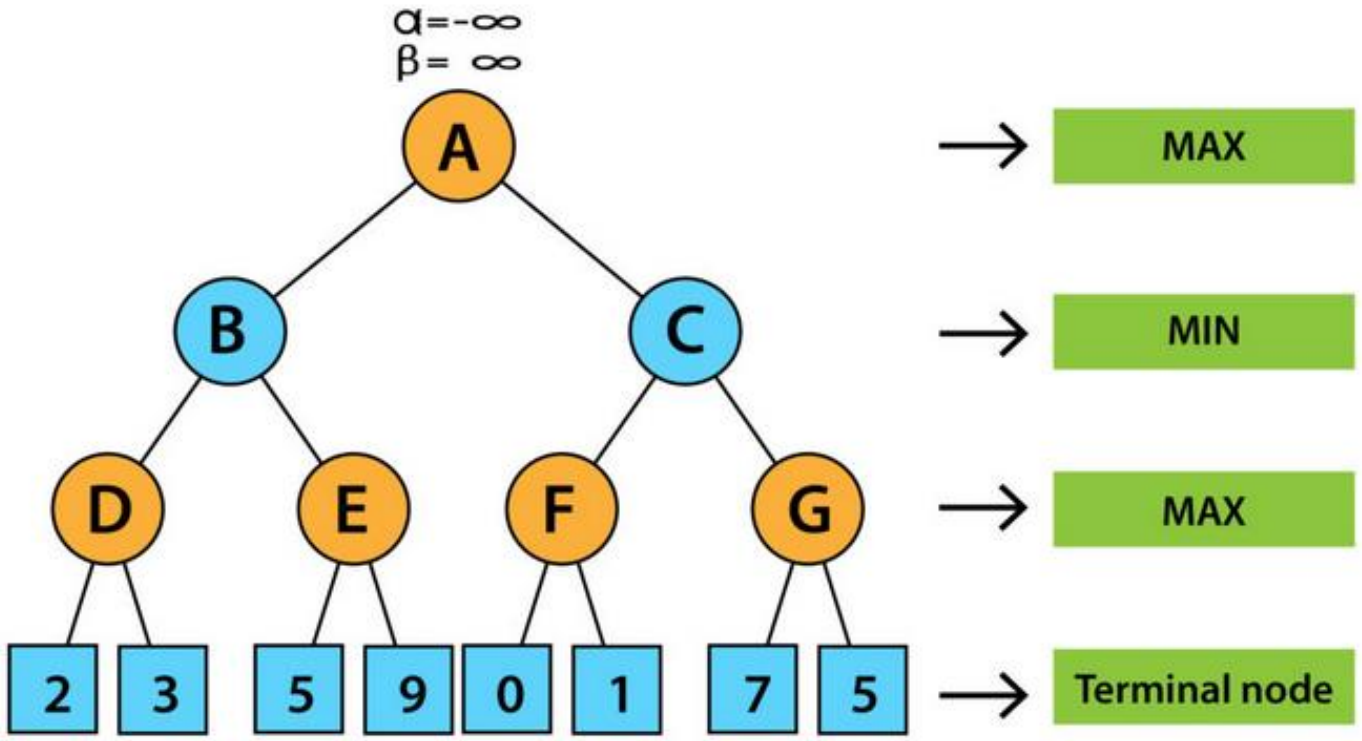
We will only pass the alpha, beta values to the child nodes.

Example



Working of Alpha-beta Pruning

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.



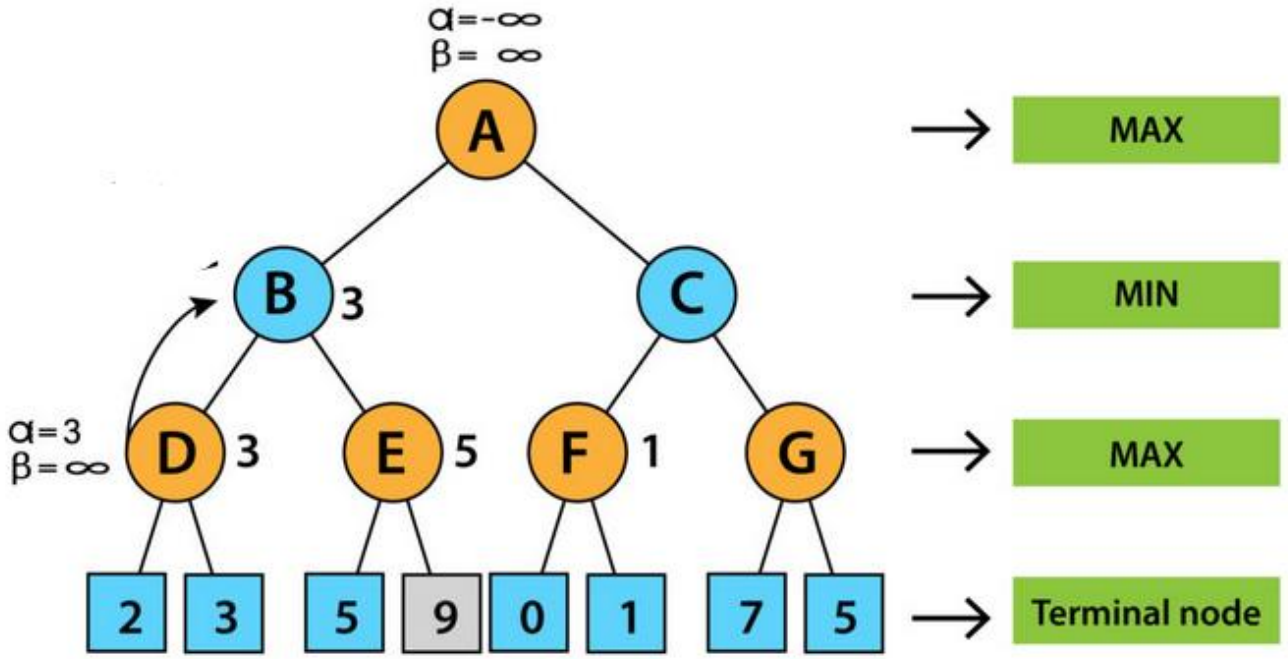
Since the initial value of alpha is less than beta so we didn't prune it.

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).

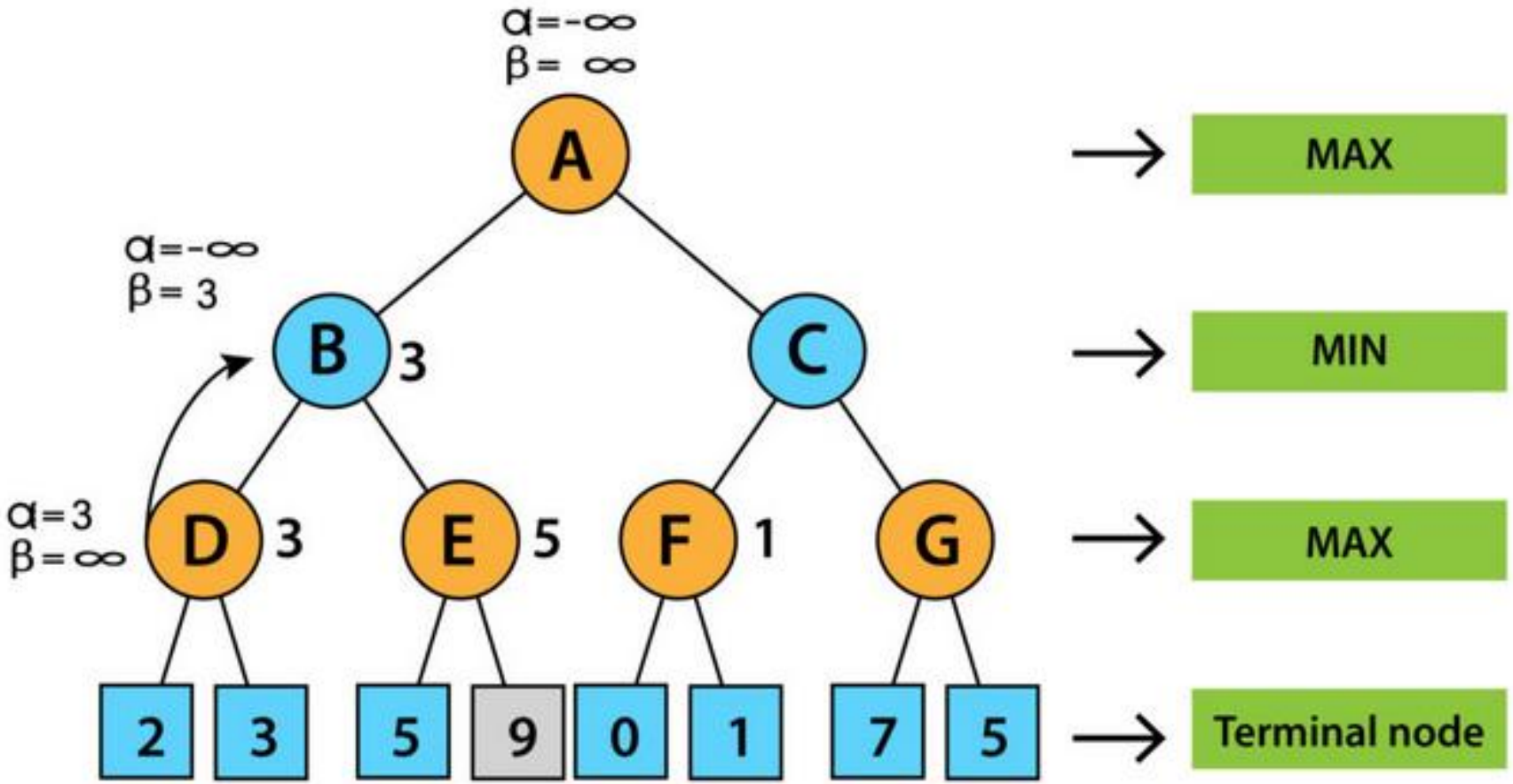
So, value of alpha at node D will be 3.



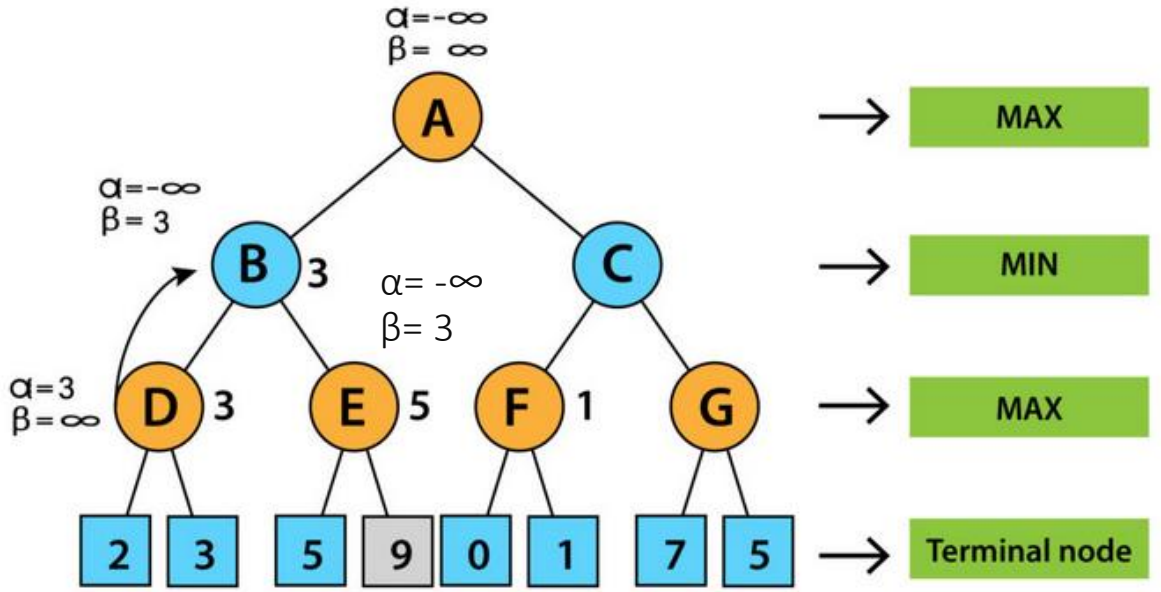
Now the next move will be on node B and its turn for MIN now.

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

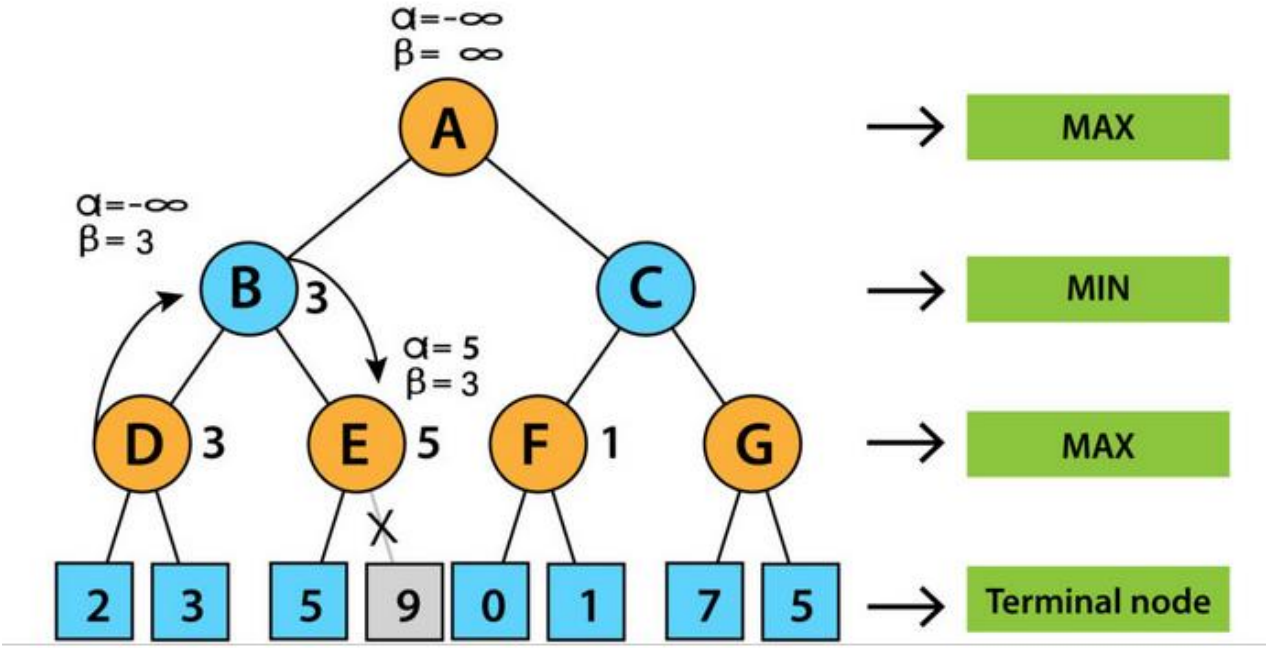
So, at node B values will be alpha= – ∞ and beta will be 3.

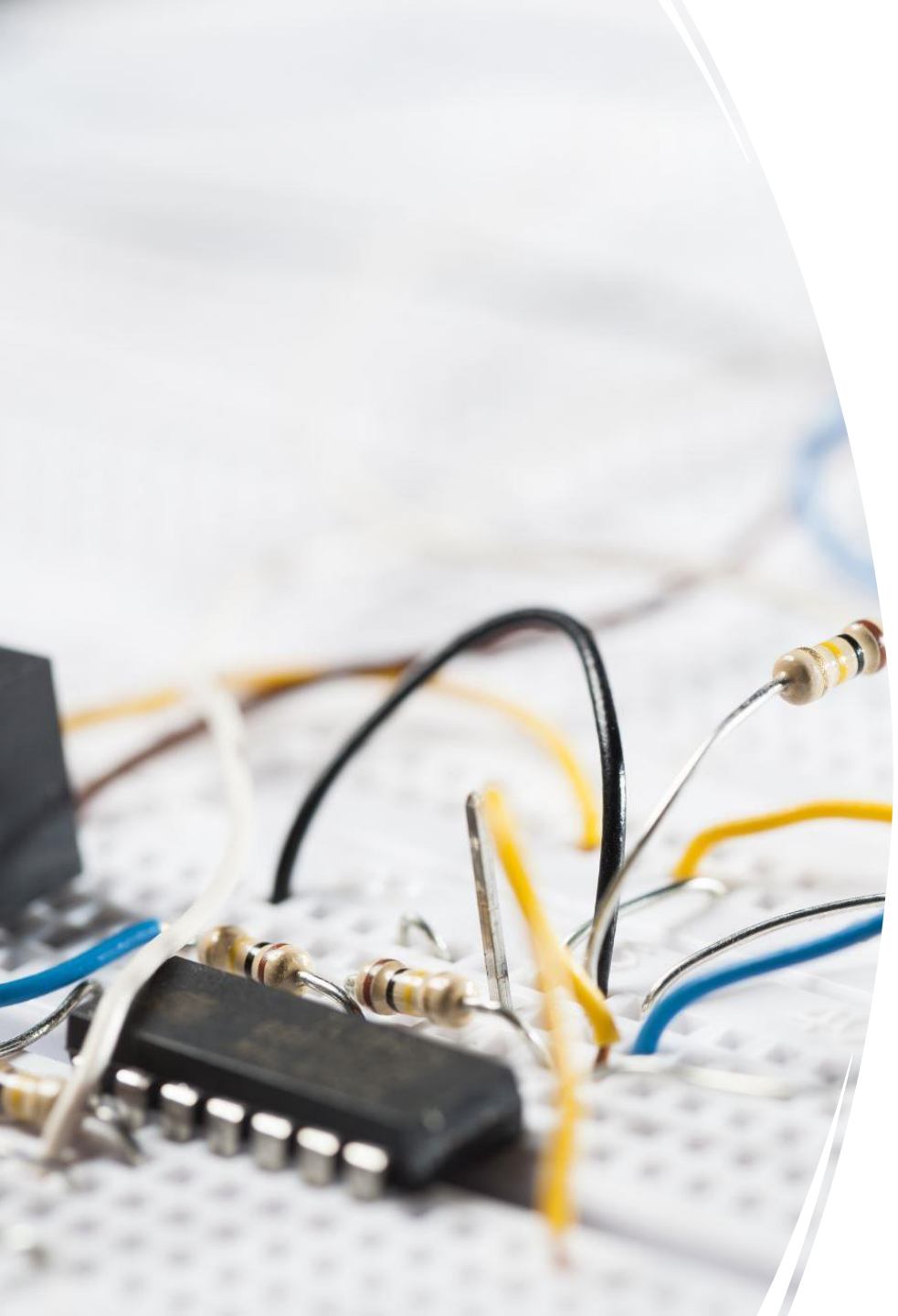


- In the next step, algorithms traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.



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.

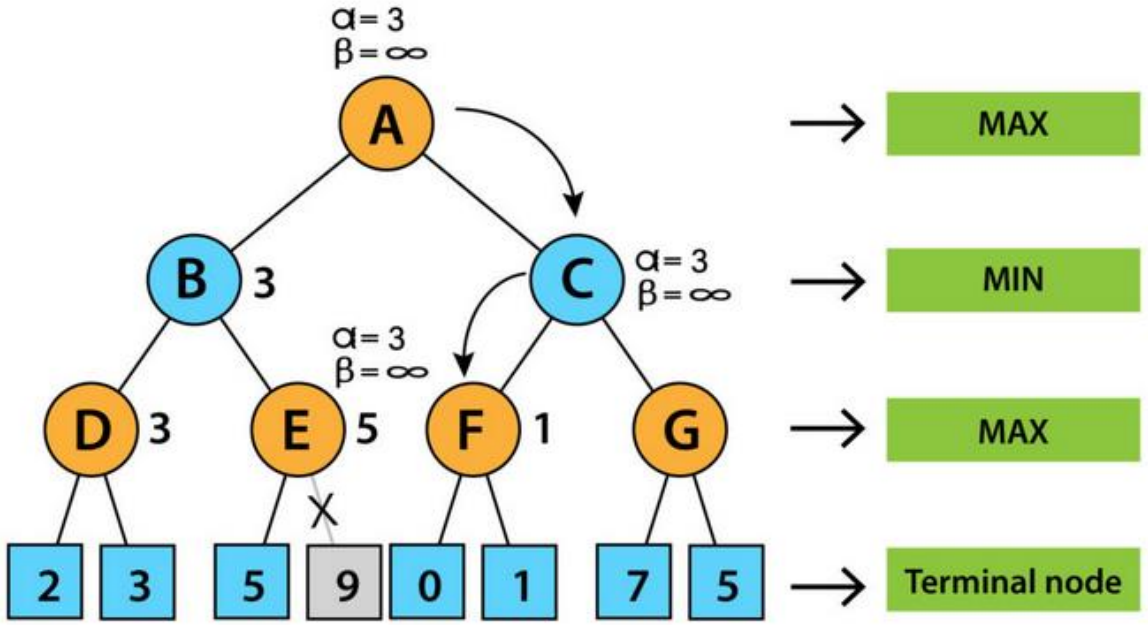




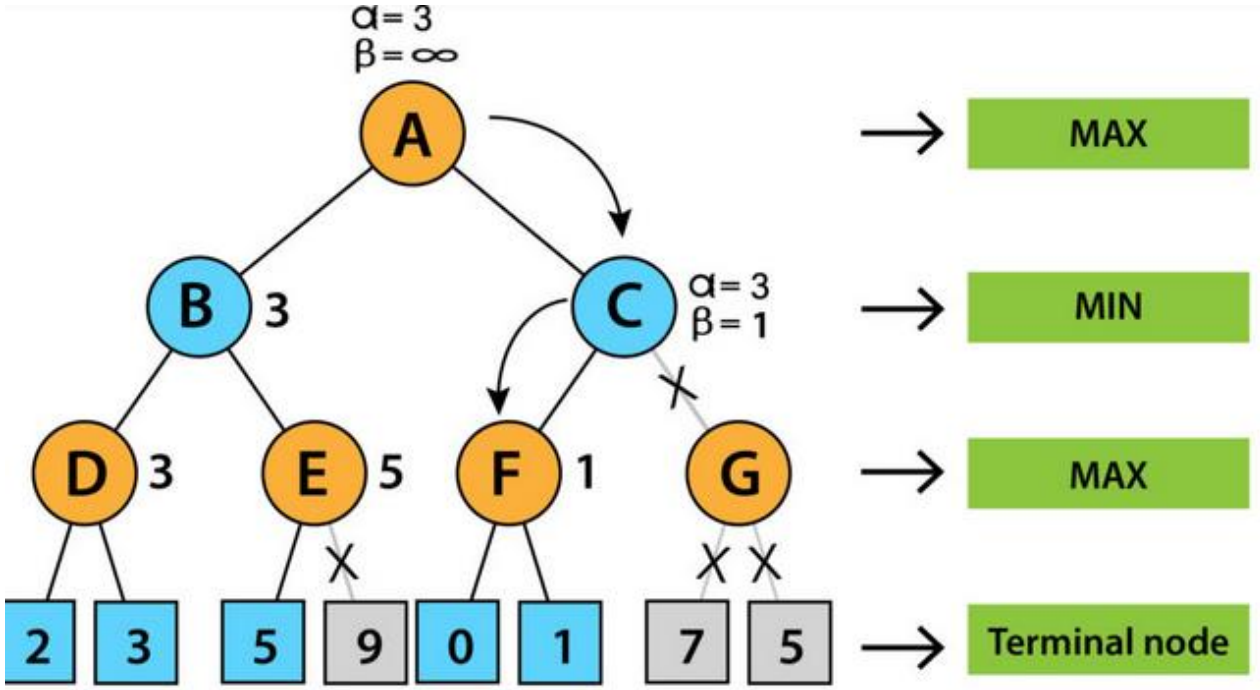
Next Step

- 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.

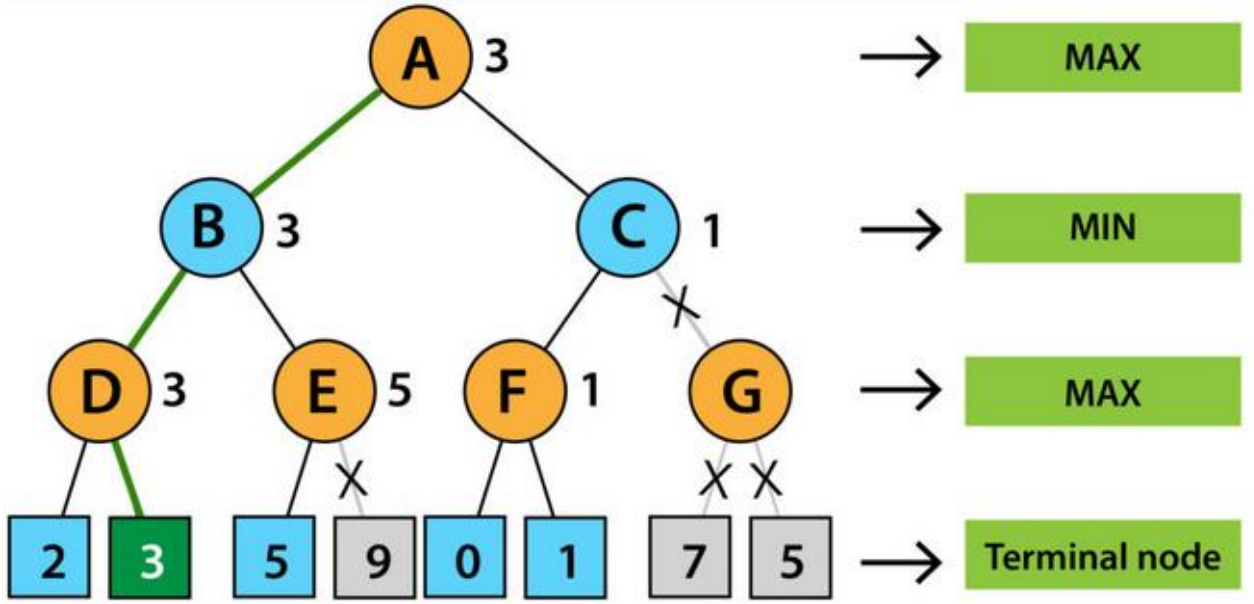
- At node F the value of alpha will be compared to the left branch which is 0. So, MAX (0, 3) will be 3 and then compared with the right child which is 1, and MAX (3,1) = 3 still α remains 3, but the node value of F will become 1.



- 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.

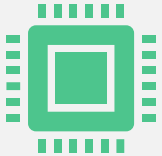


- Now, C will return the node value to A and the best value of A will be MAX (1, 3) will be 3.
- The above represented tree is the final tree which is showing the nodes which are computed and the nodes which are not computed. So, for this example the optimal value of the maximizer will be 3.



Move Ordering in Pruning

The effectiveness of alpha – beta pruning is based on the order in which node is examined. Move ordering plays an important role in alpha beta pruning.



Worst Ordering: In some cases of alpha beta pruning none of the node pruned by the algorithm and works like standard minimax algorithm. This consumes a lot of time as because of alpha and beta factors and also not gives any effective results. This is called Worst ordering in pruning. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.

Ideal Ordering: In some cases of alpha beta pruning lot of the nodes pruned by the algorithm. This is called Ideal ordering in pruning. In this case, the best move occurs on the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```