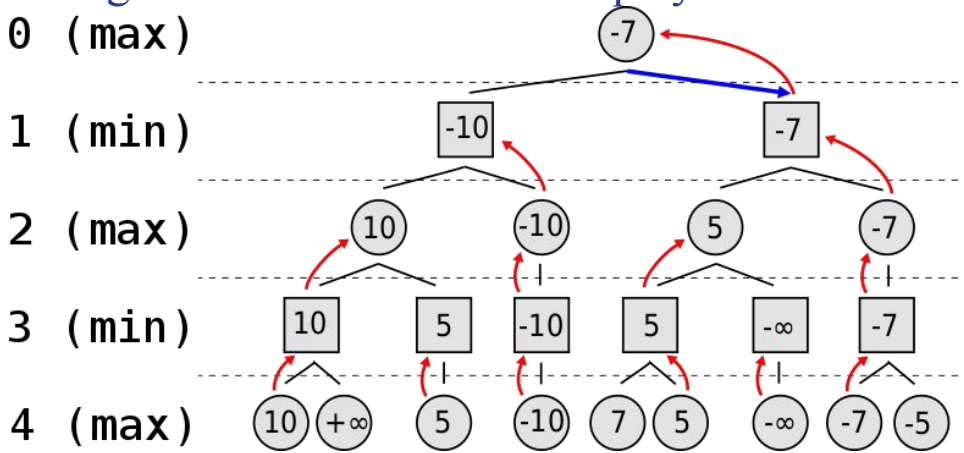
# Alpha-Beta Pruning

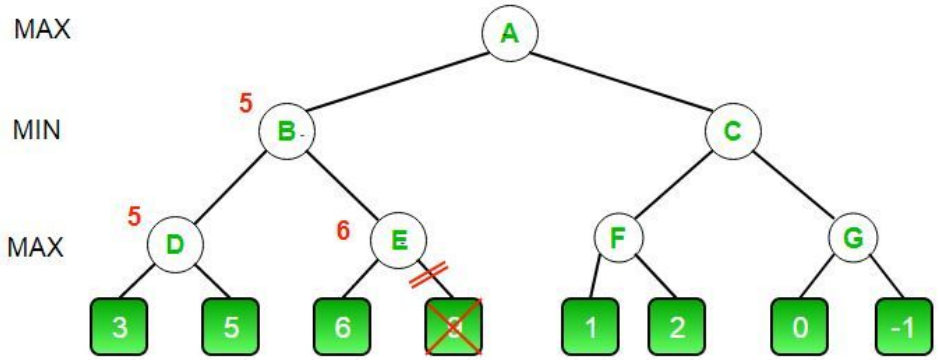
# Introduction: History

The alpha-beta pruning algorithm is a search algorithm commonly used in decision trees and game trees to find the best possible move for a player in a game. It is a variant of the minimax algorithm, which is a backtracking algorithm that searches through all possible moves of a game and evaluates them based on a static evaluation function. The goal of the minimax algorithm is to find the move that maximizes the score for the current player while minimizing the score for the opponent. This is achieved by recursively exploring the tree of possible moves, alternating between maximizing the score for the current player and minimizing the score for the opponent at each level.



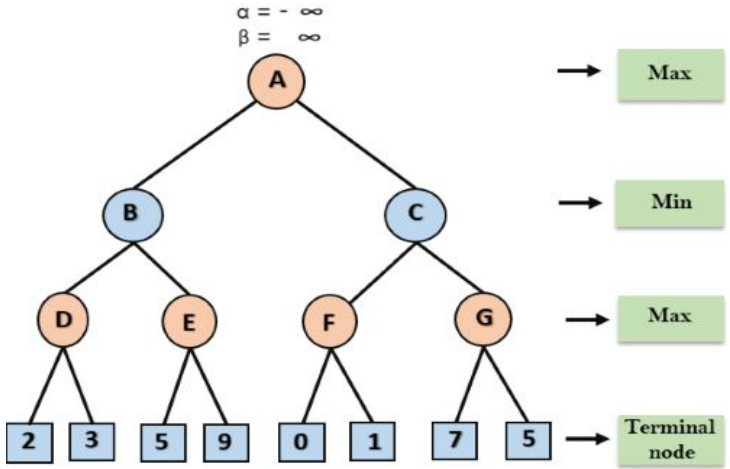
# Introduction: Improvements

- The alpha-beta pruning algorithm addresses this issue by cutting off branches of the game tree that are determined to be irrelevant to the final decision. It does this by maintaining two values, alpha and beta, that represent the best score the current player can achieve and the best score the opponent can achieve, respectively, at any given point in the search.



# Intuition

During the search, the algorithm compares the current branch's score to alpha and beta, pruning it if it falls outside of the range between them. By cutting off these irrelevant branches, the alpha-beta pruning algorithm significantly reduces the number of nodes visited by the algorithm, making it much more efficient than other search algorithms, particularly for games with large branching factors. The alpha-beta pruning algorithm uses techniques such as recursion and backtracking to explore the game tree, keeping track of the best possible outcome at each level of the tree.



# PseudoCode

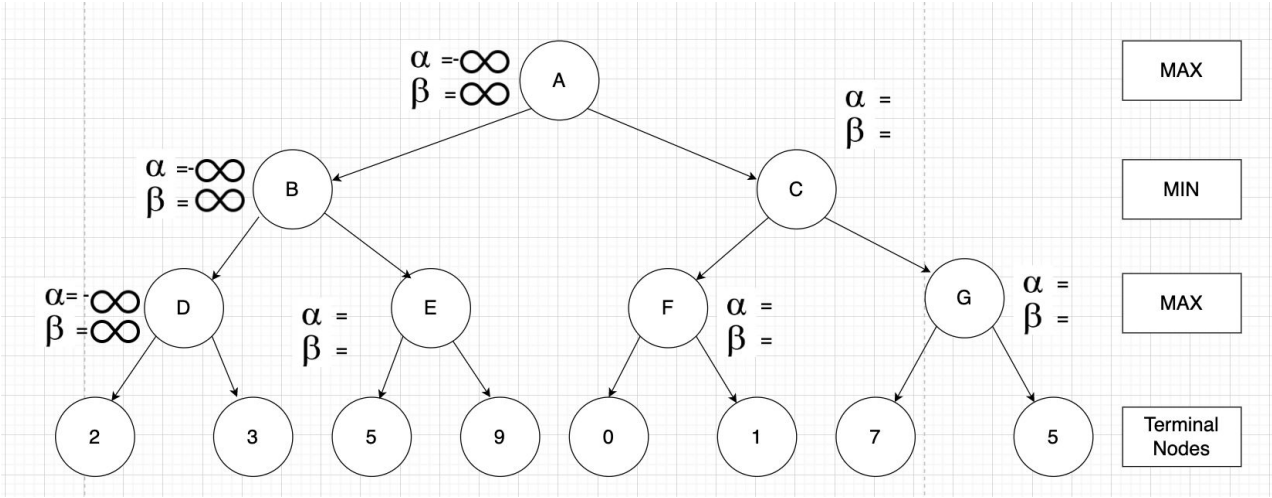
```
function GenerateTree(node, depth, max_depth, branching_factor):
    if depth == max_depth:
        return
    for i in range(branching_factor):
        child = new Node()
        node.children.add(child)
        GenerateTree(child, depth+1, max_depth, branching_factor)
```

```
function SetTree(root, options, index)
    if node is a leaf node
        node.value = options[index]
        index = index + 1
        return node
    for each child of node
        return SetTree(child, options, index)
```

```
function AlphaBetaPruning(node, max_depth, alpha, beta,maximizingPlayer):
    if depth = 0:
        return node.value
    if maximizingPlayer:
        value = -infinity
        for child in node.children:
            value = max(value, AlphaBetaPruning(child, max_depth - 1, alpha, beta, False))
            alpha = max(alpha, value)
            if beta <= alpha:
                break
            return value
    else:
        value = infinity
        for child in node.children:
            value = min(value, AlphaBetaPruning(child, depth - 1, alpha, beta, True))
            beta = min(beta, value)
        if beta <= alpha:
            break
        return value
```

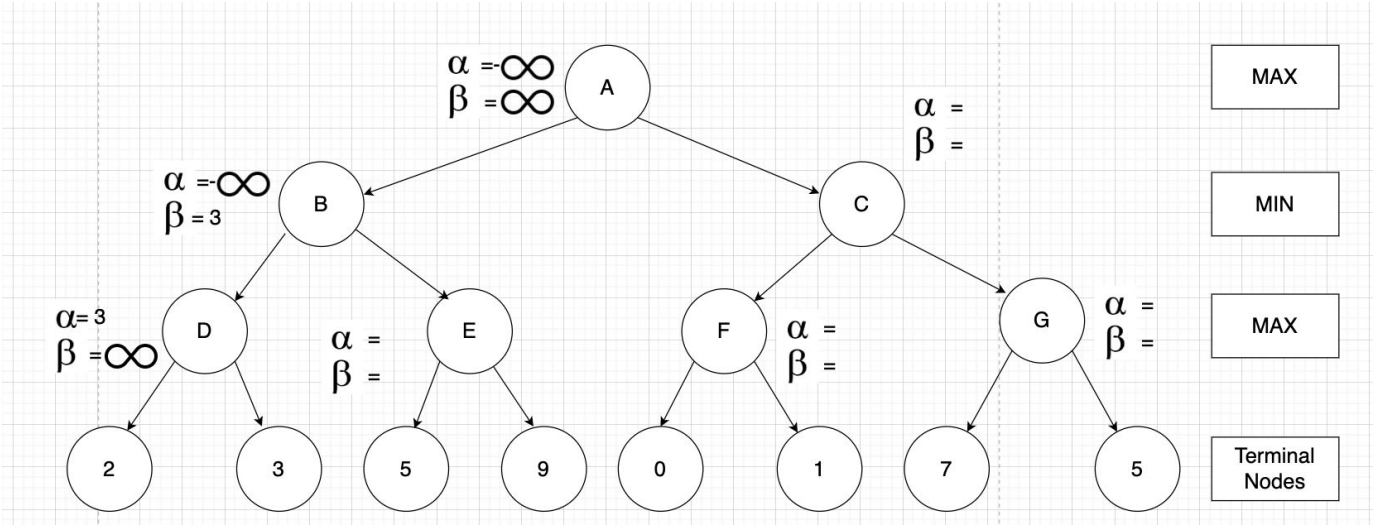
# Example Step 1

- 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.



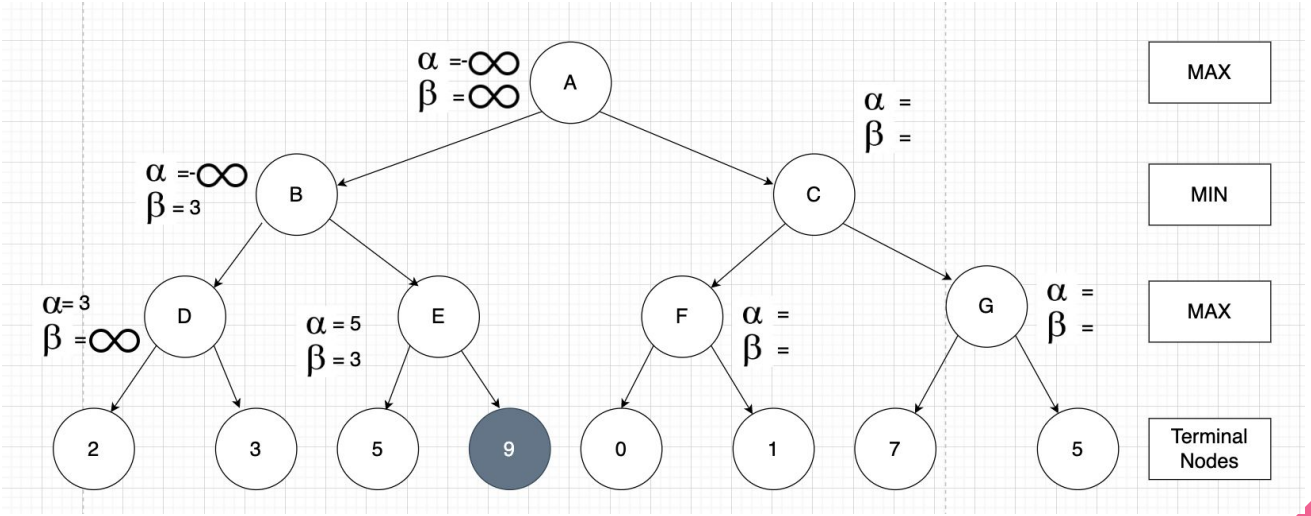
# Example Step 2

- 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 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.



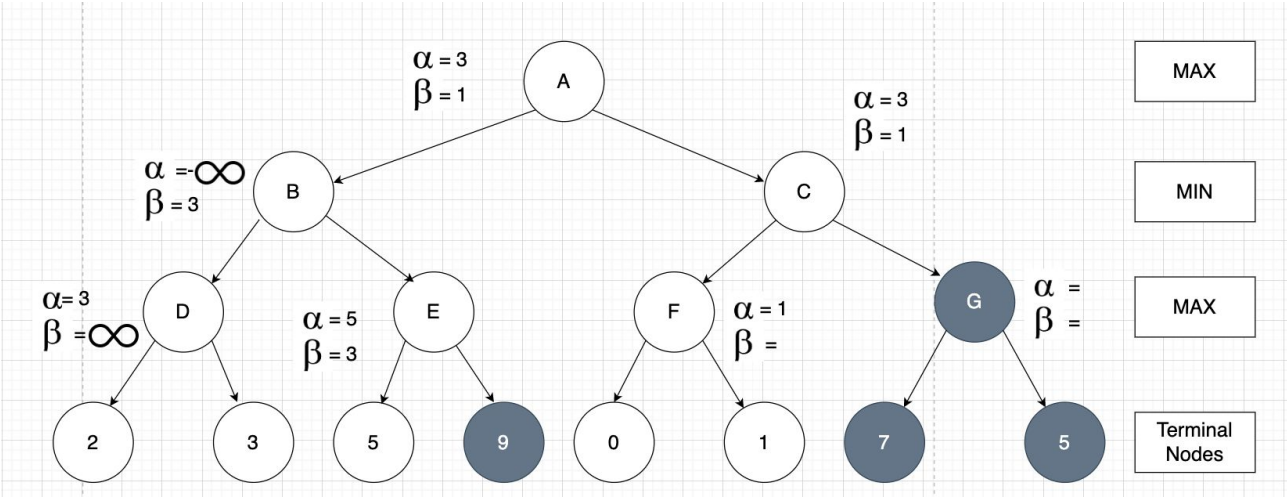
# Example Step 3

- 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.



# Example Step 4

- Repeat the previous steps until you end up with a tree like this. We see that the best option is 3 so we trace back to 3 to get the best possible option while not having to explore the whole tree.



# Analysis Part 1

The time complexity of the Alpha Beta Pruning algorithm is typically expressed in terms of the size of the game tree being searched. The size of the game tree is determined by the number of nodes it contains. The time complexity of the algorithm can be expressed as the number of nodes that need to be evaluated during the search. In the worst-case scenario, the game tree is a complete tree where every non-leaf node has b children. In this case, the total number of nodes in the tree can be expressed as follows:

$$N = 1 + b + b^2 + ... + b^{(d-1)}$$

where N is the total number of nodes in the tree, b is the branching factor of the tree, and d is the maximum depth of the tree. This is a geometric series, which can be simplified as follows:

$$N = (b^d - 1) / (b - 1)$$

# Analysis Part 2

Therefore, the total number of nodes in the tree is exponential in the depth of the tree, with a base of the branching factor.

During the alpha-beta pruning algorithm, the search space is pruned based on the current best move found so far. If the algorithm discovers that a node does not improve the current best move, it does not need to evaluate its children. Therefore, the actual number of nodes evaluated during the algorithm is often much smaller than the total number of nodes in the tree. In practice, the actual number of nodes evaluated depends on the specific structure of the tree and the quality of the utility function being used.

In the best case, where the tree is perfectly balanced, the time complexity is O(b^(d/2)). In the worst case, where the tree is a linear chain, the time complexity is O(b^d), as defined by the geometric series above. However, due to the pruning technique used in the algorithm, it can often achieve a significant reduction in the search space compared to the minimax algorithm so b^d is more often than worst case than the average time.

# Questions/Feedback?