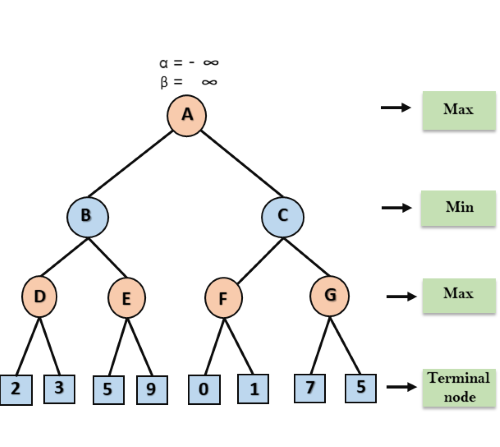
**Program 9**

# Write a Program to Implement Alpha-Beta Pruning using Python.



**Description**

Alpha-Beta pruning is an optimization technique for the minimax algorithm that helps reduce the number of nodes evaluated in the search tree. It does this by pruning branches in the search tree that cannot possibly influence the final decision. It does this by maintaining two values, alpha and beta, that help in pruning the tree:

* **Alpha** is the best value that the maximizing player can guarantee so far (the highest value encountered).
* **Beta** is the best value that the minimizing player can guarantee so far (the lowest value encountered).

**How Alpha-Beta Pruning Works:**

* If at any point during the search, the value of a node becomes worse than the current alpha or beta, we prune (stop exploring) that node and move to another branch of the tree.
* If the value of the node falls outside of the alpha-beta bounds, no further exploration is done for that branch.

**Basic Algorithm:**

1. **Maximizing Player**: Tries to maximize the score.
2. **Minimizing Player**: Tries to minimize the score.
3. **Alpha** and **Beta** are passed down the tree and updated at each node.

**Alpha-Beta Pruning Implementation:**

Let's implement Alpha-Beta Pruning for a simple game tree.

**Step-by-Step:**

1. **alpha\_beta\_pruning()**:
   * The function takes in a node, the current depth, alpha and beta values, and a boolean maximizing\_player to decide whether it's the maximizing player's or minimizing player's turn.
   * At the terminal nodes (leaf nodes), the function returns the node's value.
   * The function recursively explores the children of the current node and applies the pruning logic:
     + If it's a maximizing player's turn, we maximize the value and update the alpha value.
     + If it's a minimizing player's turn, we minimize the value and update the beta value.
     + Pruning occurs if the current node's value is worse than the alpha or beta values, i.e., no need to explore further.
2. **Sample Game Tree**:
   * The example tree has three branches, each representing a possible move.
   * The leaves have values: 3, 5, 6, 9, 1, and 2.
   * We apply Alpha-Beta pruning with the root as a maximizing player, alternating between maximizing and minimizing players.

**How it Works:**

* nitially, alpha is set to -inf and beta is set to inf.
* The algorithm explores the game tree:
* It starts with maximizing at the root, then minimizes at the first child, maximizes at the second child, and so on.
* During this exploration, if it encounters a situation where further exploration is pointless (based on the alpha and beta values), it prunes that branch.
* The final result is the optimal value (6 in this case) based on the Alpha-Beta pruning algorithm.

**SOURCE CODE :**

"""

Alpha Beta Pruning :

-------------------

depth (int): Current depth in the game tree.

node\_index (int): Index of the current node in the values array.

maximizing\_player (bool): True if the current player is maximizing, False otherwise.

values (list): List of leaf node values.

alpha (float): Best value for the maximizing player.

beta (float): Best value for the minimizing player.

Returns:

int: The optimal value for the current player.

"""

import math

def alpha\_beta\_pruning(depth, node\_index, maximizing\_player, values, alpha, beta):

# Base case: leaf node

if depth == 0 or node\_index >= len(values):

return values[node\_index]

if maximizing\_player:

max\_eval = -math.inf

for i in range(2): # Each node has two children

eval = alpha\_beta\_pruning(depth - 1, node\_index \* 2 + i, False, values, alpha, beta)

max\_eval = max(max\_eval, eval)

alpha = max(alpha, eval)

if beta <= alpha:

break # Beta cutoff

return max\_eval

else:

min\_eval = math.inf

for i in range(2): # Each node has two children

eval = alpha\_beta\_pruning(depth - 1, node\_index \* 2 + i, True, values, alpha, beta)

min\_eval = min(min\_eval, eval)

beta = min(beta, eval)

if beta <= alpha:

break # Alpha cutoff

return min\_eval

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Leaf node values for a complete binary tree

values = [3, 5, 6, 9, 1, 2, 0, -1]

depth = 3 # Height of the tree

optimal\_value = alpha\_beta\_pruning(depth, 0, True, values, -math.inf, math.inf)

print(f"The optimal value is: {optimal\_value}")

**OUTPUT :**

The optimal value is: 5