```
import math

class Node:
    def __init__(self, value=None):
        self.value = value
        self.children = []

def minimax(node, depth, maximizing_player):
    if depth == 0 or not node.children:
        return node.value

    if maximizing_player:
        max_eval = -math.inf
        for child in node.children:
            eval = minimax(child, depth - 1, False)
            max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = math.inf
        for child in node.children:
            eval = minimax(child, depth - 1, True)
            min_eval = min(min_eval, eval)
        return min_eval

def alpha_beta_pruning(node, depth, alpha, beta, maximizing_player):
    if depth == 0 or not node.children:
        return node.value

    if maximizing_player:
        max_eval = -math.inf
        for child in node.children:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = math.inf
        for child in node.children:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval

# Example usage
if __name__ == "__main__":
    root = Node()
    root.children = [Node(3), Node(6), Node(8)]
    root.children[0].children = [Node(4), Node(2)]
    root.children[1].children = [Node(9), Node(1)]
    root.children[2].children = [Node(5), Node(7)]

    print("Minimax result:", minimax(root, 2, True))
    print("Alpha-Beta Pruning result:", alpha_beta_pruning(root, 2, -math.inf, math.inf, True))
```

```
Minimax result: 5
Alpha-Beta Pruning result: 5
```