PRACTICAL10

```
# Alpha-Beta Pruning function
def alpha_beta(node, depth, alpha, beta, maximizingPlayer, game_tree, values):
    """
    node            -> current node (state in the game tree)
    depth           -> remaining depth to explore
    alpha           -> best value the maximizer has found so far
    beta            -> best value the minimizer has found so far
    maximizingPlayer  -> True if it's MAX's turn, False if it's MIN's turn
    game_tree       -> dictionary representing game tree (parent -> [children])
    values          -> dictionary of heuristic values for leaf nodes
    """

    # 1. Base case: stop when depth = 0 or node has no children (terminal node)
    if depth == 0 or node not in game_tree:
        return values[node]   # return heuristic (leaf node value)

    # 2. If current player is the Maximizer (trying to maximize score)
    if maximizingPlayer:
        value = float("-inf")     # Start with -∞ (worst case for maximizer)

        # Explore all children of current node
        for child in game_tree[node]:
            # Recursive call: switch to Minimizer
            value = max(value, alpha_beta(child, depth - 1, alpha, beta, False, game_tree, values))

            # Update alpha with the best value found so far
            alpha = max(alpha, value)

            # Pruning condition: if beta ≤ alpha, stop exploring further
            if beta <= alpha:
                break   # β cut-off (no need to check remaining children)

        return value   # Return best value found for Maximizer

    # 3. If current player is the Minimizer (trying to minimize score)
    else:
        value = float("inf")      # Start with +∞ (worst case for minimizer)

        # Explore all children of current node
        for child in game_tree[node]:
            # Recursive call: switch to Maximizer
            value = min(value, alpha_beta(child, depth - 1, alpha, beta, True, game_tree, values))
```

```
            # Update beta with the best value found so far
            beta = min(beta, value)

            # Pruning condition: if beta ≤ alpha, stop exploring further
            if beta <= alpha:
                break  # α cut-off

        return value  # Return best value found for Minimizer


# ---------- Example Usage ----------
if __name__ == "__main__":
    # Define a simple game tree as adjacency list
    game_tree = {
        'A': ['B', 'C'],    # Root A has two children: B and C
        'B': ['D', 'E'],    # Node B has two children: D and E
        'C': ['F', 'G']     # Node C has two children: F and G
    }

    # Define heuristic values (leaf nodes only)
    values = {
        'D': 3,  # Leaf node D has value 3
        'E': 5,  # Leaf node E has value 5
        'F': 6,  # Leaf node F has value 6
        'G': 9   # Leaf node G has value 9
    }

    # Call alpha-beta pruning starting at root node 'A'
    # depth = 3 (A -> B/C -> D/E/F/G)
    # alpha = -∞, beta = +∞, maximizingPlayer = True (root is MAX's turn)
    best_value = alpha_beta('A', 3, float("-inf"), float("inf"), True, game_tree, values)

    # Print result
    print("Best achievable value for root A:", best_value)
```