WEEK-10

Implement Alpha-Beta Pruning

import math

def minimax(node, depth, is\_maximizing):

    """

    Implement the Minimax algorithm to solve the decision tree.

    Parameters:

    node (dict): The current node in the decision tree, with the following structure:

        {

            'value': int,

            'left': dict or None,

            'right': dict or None

        }

    depth (int): The current depth in the decision tree.

    is\_maximizing (bool): Flag to indicate whether the current player is the maximizing player.

    Returns:

    int: The utility value of the current node.

    """

    # Base case: Leaf node

    if node['left'] is None and node['right'] is None:

        return node['value']

    # Recursive case

    if is\_maximizing:

        best\_value = -math.inf

        if node['left']:

            best\_value = max(best\_value, minimax(node['left'], depth + 1, False))

        if node['right']:

            best\_value = max(best\_value, minimax(node['right'], depth + 1, False))

        return best\_value

    else:

        best\_value = math.inf

        if node['left']:

            best\_value = min(best\_value, minimax(node['left'], depth + 1, True))

        if node['right']:

            best\_value = min(best\_value, minimax(node['right'], depth + 1, True))

        return best\_value

# Example usage

decision\_tree = {

    'value': 5,

    'left': {

        'value': 6,

        'left': {

            'value': 7,

            'left': {

                'value': 4,

                'left': None,

                'right': None

            },

            'right': {

                'value': 5,

                'left': None,

                'right': None

            }

        },

        'right': {

            'value': 3,

            'left': {

                'value': 6,

                'left': None,

                'right': None

            },

            'right': {

                'value': 9,

                'left': None,

                'right': None

            }

        }

    },

    'right': {

        'value': 8,

        'left': {

            'value': 7,

            'left': {

                'value': 6,

                'left': None,

                'right': None

            },

            'right': {

                'value': 9,

                'left': None,

                'right': None

            }

        },

        'right': {

            'value': 8,

            'left': {

                'value': 6,

                'left': None,

                'right': None

            },

            'right': None

        }

    }

}

# Find the best move for the maximizing player

best\_value = minimax(decision\_tree, 0, True)

print(f"The best value for the maximizing player is: {best\_value}")

OUTPUT:

