LAB REPORT 08

SUBMITTED TO: MR. MUBASHIR IQBAL

SUBMITTED BY: AHMED SALEEM RANA

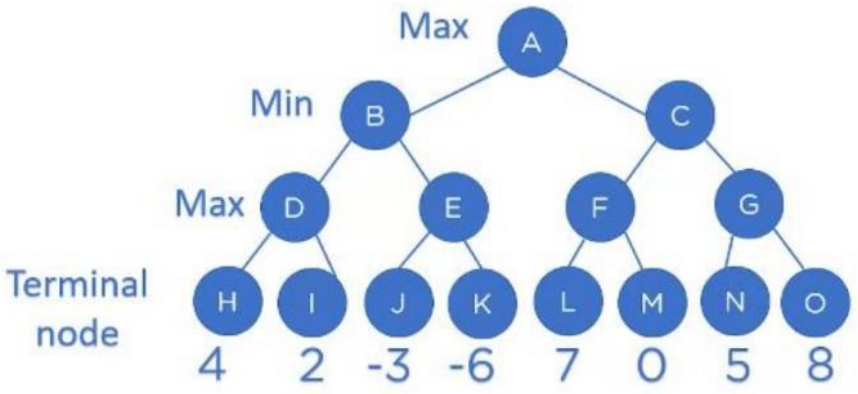
REG ID: 24-CYS-023

SECTION: B

HITEC UNIVERSITY TAXILA, CANTT

TASK 01

1. Apply Alpha-Beta Pruning to the graph.



CODE:

```
# Alpha-Beta Pruning Implementation
import math
# Terminal node values
terminal_values = {
    "H": 4, "I": 2,
    "J": -3, "K": -6,
    "L": 7, "M": 0,
    "N": 5, "O": 8
}
# Tree structure
tree = {
    "A": ["B", "C"],
    "B": ["D", "E"],
    "C": ["F", "G"],
    "D": ["H", "I"],
    "E": ["J", "K"],
    "F": ["L", "M"],
    "G": ["N", "O"]
}
def alphabeta(node, depth, alpha, beta, is_max):
    # If terminal node
    if node not in tree:
        return terminal_values[node]

    if is_max:
        max_eval = -math.inf
```

```
            for child in tree[node]:
                value = alphabeta(child, depth + 1, alpha, beta, False)
                max_eval = max(max_eval, value)
                alpha = max(alpha, value)
                if beta <= alpha:
                    break  # Beta cut-off
            return max_eval

        else:
            min_eval = math.inf
            for child in tree[node]:
                value = alphabeta(child, depth + 1, alpha, beta, True)
                min_eval = min(min_eval, value)
                beta = min(beta, value)
                if beta <= alpha:
                    break  # Alpha cut-off
            return min_eval
    # Run Alpha-Beta Pruning from root A
    result = alphabeta("A", 0, -math.inf, math.inf, True)
    print("Optimal value at root (A):", result)
```

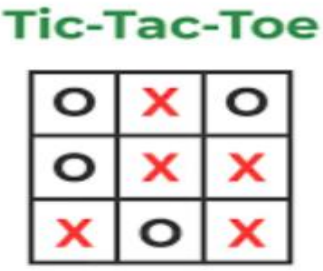
OUTPUT:

```
...   Optimal value at root (A): 7
```

Generate + Code + Markdown

TASK 02

2. Apply the minimax algorithm in the Tic-Tac-Toe game.



CODE:

```
import math
# Initialize board
board = [
    [' ', ' ', ' '],
    [' ', ' ', ' '],
    [' ', ' ', ' ']
]
AI = 'X'
HUMAN = 'O'
# Print board
def print_board(b):
    for row in b:
        print(row)
    print()
# Check winner
def check_winner(b):
    # Rows and Columns
    for i in range(3):
        if b[i][0] == b[i][1] == b[i][2] != ' ':
            return b[i][0]
        if b[0][i] == b[1][i] == b[2][i] != ' ':
            return b[0][i]
    # Diagonals
    if b[0][0] == b[1][1] == b[2][2] != ' ':
        return b[0][0]
    if b[0][2] == b[1][1] == b[2][0] != ' ':
        return b[0][2]
    return None
            for j in range(3):
                if b[i][j] == ' ':
                    b[i][j] = HUMAN
                    score = minimax(b, depth + 1, True)
                    b[i][j] = ' '
                    best = min(best, score)
        return best
# Find best move for AI
def best_move():
    best_score = -math.inf
    move = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = AI
                score = minimax(board, 0, False)
                board[i][j] = ' '
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move
# Example Run
print("Initial Board:")
print_board(board)
row, col = best_move()
board[row][col] = AI
print("Board After AI Move:")
print_board(board)
```

```python
# Check draw
def is_draw(b):
    for row in b:
        if ' ' in row:
            return False
    return True
# Minimax Algorithm
def minimax(b, depth, is_max):
    winner = check_winner(b)
    if winner == AI:
        return 10 - depth
    if winner == HUMAN:
        return depth - 10
    if is_draw(b):
        return 0
    if is_max:
        best = -math.inf
        for i in range(3):
            for j in range(3):
                if b[i][j] == ' ':
                    b[i][j] = AI
                    score = minimax(b, depth + 1, False)
                    b[i][j] = ' '
                    best = max(best, score)
        return best
    else:
        best = math.inf
        for i in range(3):
```

OUTPUT:

```
Initial Board:
[' ', ' ', ' ']
[' ', ' ', ' ']
[' ', ' ', ' ']

Board After AI Move:
['X', ' ', ' ']
[' ', ' ', ' ']
[' ', ' ', ' ']
```