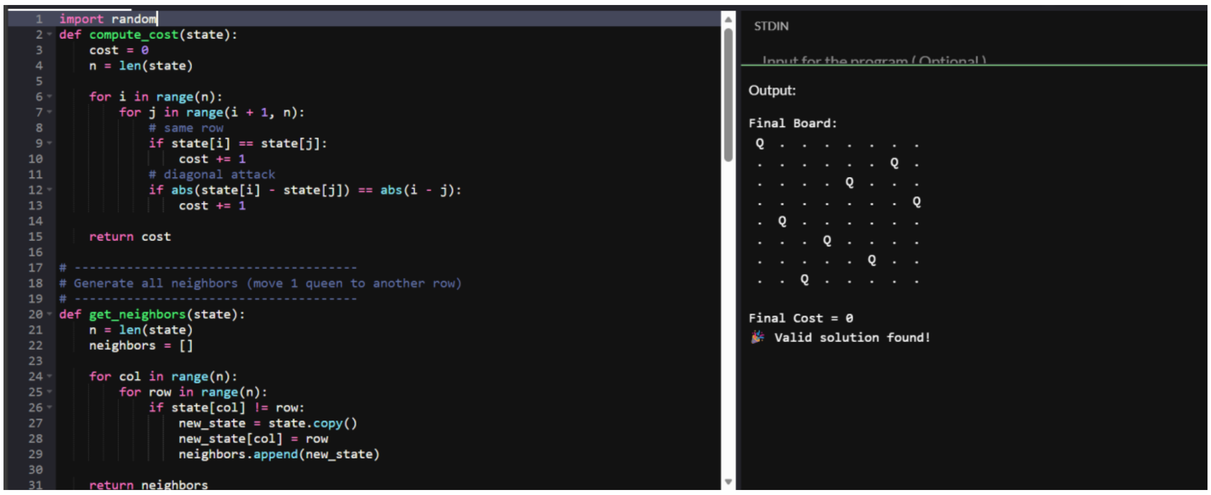
MLA0109 – Artificial Intelligence and Expert System

Python Programs – Day 3

1. Implement a Python program to solve the N-Queens problem using Hill Climbing. The program should:
   a) Represent a state as positions of N queens (one per column).
   b) Define a cost function = number of attacking pairs of queens.
   c) Generate neighbors by moving one queen within its column to a different row.
   d) Move to the best neighbor if it strictly improves the cost.
   e) Stop when no better neighbor exists (local optimum).
   f) (Optional) Use random restarts to increase the chance of finding a global optimum.
   g) Print the final board, the final cost, and whether a valid solution (cost = 0) was found.



```python
import random
def compute_cost(state):
    cost = 0
    n = len(state)

    for i in range(n):
        for j in range(i + 1, n):
            # same row
            if state[i] == state[j]:
                cost += 1
            # diagonal attack
            if abs(state[i] - state[j]) == abs(i - j):
                cost += 1

    return cost

# -------------------------------------
# Generate all neighbors (move 1 queen to another row)
# -------------------------------------
def get_neighbors(state):
    n = len(state)
    neighbors = []

    for col in range(n):
        for row in range(n):
            if state[col] != row:
                new_state = state.copy()
                new_state[col] = row
                neighbors.append(new_state)

    return neighbors
```

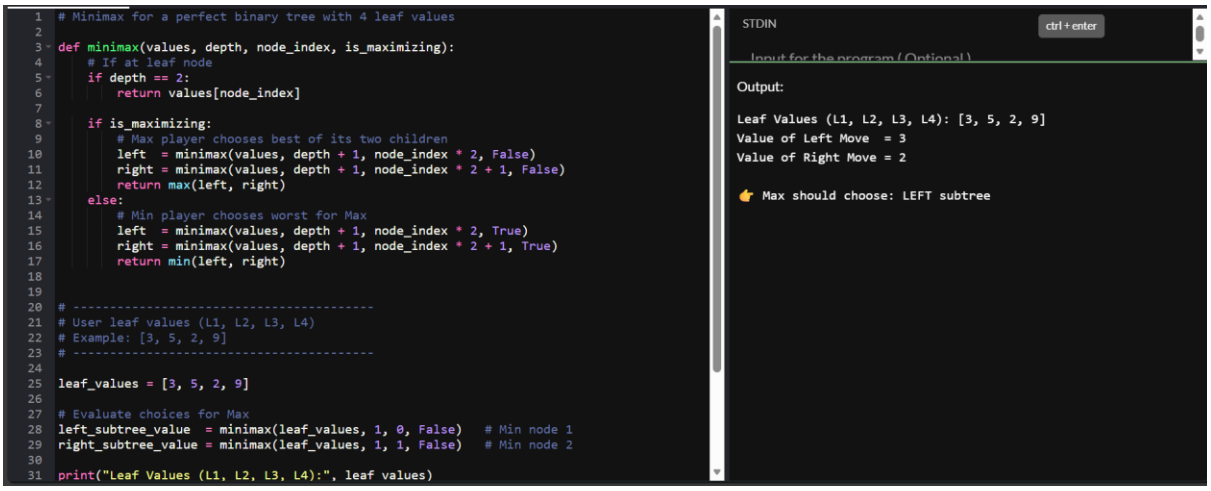
STDIN

Input for the program (Optional)

Output:

```
Final Board:
 Q  .  .  .  .  .  .  .
 .  .  .  .  .  .  Q  .
 .  .  .  .  Q  .  .  .
 .  .  .  .  .  .  .  Q
 .  Q  .  .  .  .  .  .
 .  .  .  Q  .  .  .  .
 .  .  .  .  .  Q  .  .
 .  .  Q  .  .  .  .  .

Final Cost = 0
🎉 Valid solution found!
```

2. Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level. Using python create the program, which move you would make as a maximizing player considering that your opponent also plays optimally?

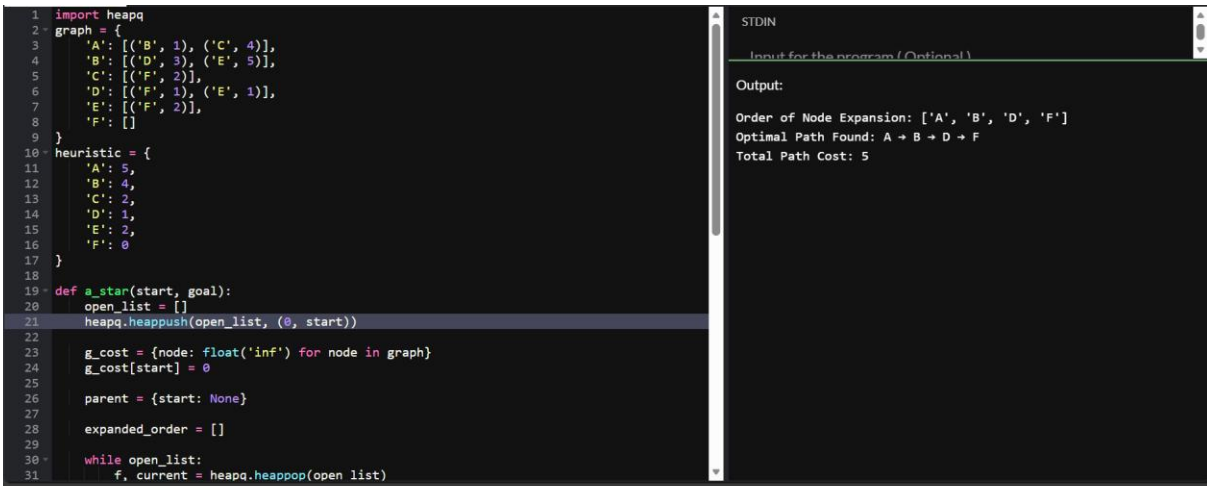


```python
# Minimax for a perfect binary tree with 4 leaf values

def minimax(values, depth, node_index, is_maximizing):
    # If at leaf node
    if depth == 2:
        return values[node_index]

    if is_maximizing:
        # Max player chooses best of its two children
        left  = minimax(values, depth + 1, node_index * 2, False)
        right = minimax(values, depth + 1, node_index * 2 + 1, False)
        return max(left, right)
    else:
        # Min player chooses worst for Max
        left  = minimax(values, depth + 1, node_index * 2, True)
        right = minimax(values, depth + 1, node_index * 2 + 1, True)
        return min(left, right)


# ----------------------------------------
# User leaf values (L1, L2, L3, L4)
# Example: [3, 5, 2, 9]
# ----------------------------------------

leaf_values = [3, 5, 2, 9]

# Evaluate choices for Max
left_subtree_value  = minimax(leaf_values, 1, 0, False)   # Min node 1
right_subtree_value = minimax(leaf_values, 1, 1, False)   # Min node 2

print("Leaf Values (L1, L2, L3, L4):", leaf_values)
```

Output:

```
Leaf Values (L1, L2, L3, L4): [3, 5, 2, 9]
Value of Left Move  = 3
Value of Right Move = 2

👉 Max should choose: LEFT subtree
```

3. Implement a Python program to solve a pathfinding problem using the A* algorithm. The program should:
   a) Represent the environment as a graph or a grid with weighted edges.
   b) Take a start node and a goal node as inputs.
   c) Use a suitable heuristic function (e.g., Manhattan distance for grids, straight-linedistance for graphs).
   d) Explore nodes using the A* evaluation function.
   e) Print the order of node expansion and the final optimal path with its total cost.
   f) Start Node: A Goal Node: F Nodes: A, B, C, D, E, F Edges with weights: (A, B, 1), (A, C, 4), (B, D, 3), (B, E, 5), (C, F, 2), (D, F, 1), (D, E, 1), (E, F, 2).



```python
import heapq
graph = {
    'A': [('B', 1), ('C', 4)],
    'B': [('D', 3), ('E', 5)],
    'C': [('F', 2)],
    'D': [('F', 1), ('E', 1)],
    'E': [('F', 2)],
    'F': []
}
heuristic = {
    'A': 5,
    'B': 4,
    'C': 2,
    'D': 1,
    'E': 2,
    'F': 0
}

def a_star(start, goal):
    open_list = []
    heapq.heappush(open_list, (0, start))

    g_cost = {node: float('inf') for node in graph}
    g_cost[start] = 0

    parent = {start: None}

    expanded_order = []

    while open_list:
        f, current = heapq.heappop(open_list)
```

Output:

```
Order of Node Expansion: ['A', 'B', 'D', 'F']
Optimal Path Found: A → B → D → F
Total Path Cost: 5
```