LAB 4:

Program: Implement Hill Climbing to Solve N-Queens Problem

Code:

```
import random

class NQueens:
    def __init__(self, n, initial_board=None):
        self.n = n
        self.board = initial_board if initial_board is not None else self.initialize_board()

    def initialize_board(self):
        # Place one queen in each row randomly
        return [random.randint(0, self.n - 1) for _ in range(self.n)]

    def calculate_conflicts(self, board):
        conflicts = 0
        for i in range(self.n):
            for j in range(i + 1, self.n):
                # Check for conflicts between queens
                if board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j):
                    conflicts += 1
        return conflicts

    def get_neighbors(self, board):
        neighbors = []
        for row in range(self.n):
            for col in range(self.n):
                if col != board[row]:
                    # Create a new board configuration by moving the queen
                    new_board = board[:]
```

```
                new_board[row] = col
                neighbors.append(new_board)
        return neighbors

    def hill_climbing(self):
        current_board = self.board
        current_conflicts = self.calculate_conflicts(current_board)

        print(f"Initial State: {current_board} | Conflicts: {current_conflicts}")

        while current_conflicts > 0:
            neighbors = self.get_neighbors(current_board)
            next_board = None
            next_conflicts = float('inf')

            for neighbor in neighbors:
                conflicts = self.calculate_conflicts(neighbor)
                print(f"Neighbor: {neighbor} | Conflicts: {conflicts}")

                # Choose the neighbor with the smallest number of conflicts
                # If there's a tie, choose the one with the smallest column number
                if conflicts < next_conflicts or (conflicts == next_conflicts and
self.choose_smaller(neighbor, next_board)):
                    next_board = neighbor
                    next_conflicts = conflicts

            if next_conflicts >= current_conflicts:
                # No better neighbors found
                return None
```

```
            print(f"Moving to: {next_board} | Conflicts: {next_conflicts}")
            current_board = next_board
            current_conflicts = next_conflicts

        return current_board

    def choose_smaller(self, new_board, current_board):
        if current_board is None:
            return True
        for i in range(self.n):
            if new_board[i] != current_board[i]:
                return new_board[i] < current_board[i]
        return False

    def print_board(self, board):
        for row in range(self.n):
            line = ['Q' if board[i] == row else '.' for i in range(self.n)]
            print(' '.join(line))
        print()

if __name__ == "__main__":
    n = int(input("Enter the size of the board (N): "))
    initial_board = []

    print("Enter the initial positions of the queens (0-indexed, one number per row):")
    for i in range(n):
        while True:
            try:
                pos = int(input(f"Row {i} (0 to {n-1}): "))
```

```
            if 0 <= pos < n:
                initial_board.append(pos)
                break
            else:
                print(f"Invalid position! Please enter a value between 0 and {n-1}.")
        except ValueError:
            print("Invalid input! Please enter an integer.")

    solver = NQueens(n, initial_board)
    solution = solver.hill_climbing()

    if solution:
        print("\nSolution found:")
        solver.print_board(solution)
    else:
        print("No solution found.")
```

Algorithm:

Output:

```
Enter the size of the board (N): 4
Enter the initial positions of the queens (0-indexed, one number per row):
Row 0 (0 to 3): 3
Row 1 (0 to 3): 1
Row 2 (0 to 3): 2
Row 3 (0 to 3): 0
Initial State: [3, 1, 2, 0] | Conflicts: 2
Neighbor: [0, 1, 2, 0] | Conflicts: 4
Neighbor: [1, 1, 2, 0] | Conflicts: 2
Neighbor: [2, 1, 2, 0] | Conflicts: 3
Neighbor: [3, 0, 2, 0] | Conflicts: 2
Neighbor: [3, 2, 2, 0] | Conflicts: 4
Neighbor: [3, 3, 2, 0] | Conflicts: 3
Neighbor: [3, 1, 0, 0] | Conflicts: 3
Neighbor: [3, 1, 1, 0] | Conflicts: 4
Neighbor: [3, 1, 3, 0] | Conflicts: 2
Neighbor: [3, 1, 2, 1] | Conflicts: 3
Neighbor: [3, 1, 2, 2] | Conflicts: 2
Neighbor: [3, 1, 2, 3] | Conflicts: 4
No solution found.
```

22/10/24

Lab-4

Implement Hill climbing to solve N-Queens problem.

Algorithm:

```
FUNCTION NQueens (N, initial_board)
  SET board to initial_board
FUNTION Calculate_Conflicts (board)
  SET Conflicts to 0
FOR i FROM 0 TO N-1 DO
  FOR j FROM i+1 TO N-1 DO
IF board[i] == board[j] OR abs(board[i]-
board[j]) == abs(i-j) THEN
    INCREMENT Conflicts.
  RETURN conflicts.

FUNCTION get_neighbors (board)
  SET neighbors to []
FOR row FROM 0 TO N-1 DO
FOR col FROM 0 TO N-1 DO
  IF col != board[row] THEN
SET new_board to COPY of board
  new_board[row] = col
APPEND new_board to neighbors
RETURN neighbors.
FUNCTION hill_climbing()
  SET Current_board to board
SET Current Conflicts to calculate_
    conflicts (current_board)
PRINT "Initial State:", current board, "
    Conflicts:", Current_conflicts.
WHILE Current_conflicts > 0 DO
SET neighbors to get_neighbors (current_board)
```

```
SET next_board to None
SET next_conflicts to INFINITY
FOR Each neighbour IN neighbord DO
SET Conflicts to Calculate_Conflicts(neighbor)
PRINT "Neighbor:", neighbor, "| conflicts :"Conflicts
IF Conflicts < next_Conflicts OR (conflicts ==
next_Conflicts AND Choose_Smaller(neighbor,
next_board)) THEN
SET next_board to neighbor
SET next_Conflicts to Conflicts.
IF next_Conflicts >= Current_Conflicts THEN
RETURN None
PRINT "moving to :", next_board, "| conflicts :",
next_conflicts.
SET Current_board to next_board
SET Current_Conflicts to next_conflicts.
RETURN Current_board.

FUNCTION Choose_Smaller (new_board, Current_board)
If Current_board is None THEN
RETURN True
FOR i FROM 0 TO N-1 DO
IF new_board[i] != Current_board[i] THEN
RETURN new_board[i] < Current_board[i]
RETURN False.

Solution = hill_climbing()
IF Solution THEN
PRINT "Solution Found:", Solution
Else
PRINT "NO Solution found."
```

29/10/24 Tuesday

Output:

Enter Size of board (N): 4
Enter initial position of queens (0 indexed, one number per row):
Row 0 (0 to 3): 1
Row 1 (0 to 3): 3
Row 2 (0 to 3): 2
Row 3 (0 to 3): 0
Initial State: [1, 3, 2, 0] | Conflicts: 1
neighbour: [0, 3, 2, 0] | Conflicts: 3
neighbour: [2, 3, 2, 0] | Conflicts: 3
neighbour: [3, 3, 2, 0] | Conflicts: 3
neighbour: [1, 0, 2, 0] | Conflicts: 2
neighbour: [1, 1, 2, 0] | Conflicts: 2
neighbour: [1, 2, 2, 0] | Conflicts: 3
neighbour: [1, 3, 0, 0] | Conflicts: 1
neighbour: [1, 3, 1, 0] | Conflicts: 2
neighbour: [1, 3, 3, 0] | Conflicts: 2
neighbour: [1, 3, 2, 1] | Conflicts: 4
neighbour: [1, 3, 2, 2] | Conflicts: 2
neighbour: [1, 3, 2, 3] | Conflicts: 3
No Solution found.

9/2/24