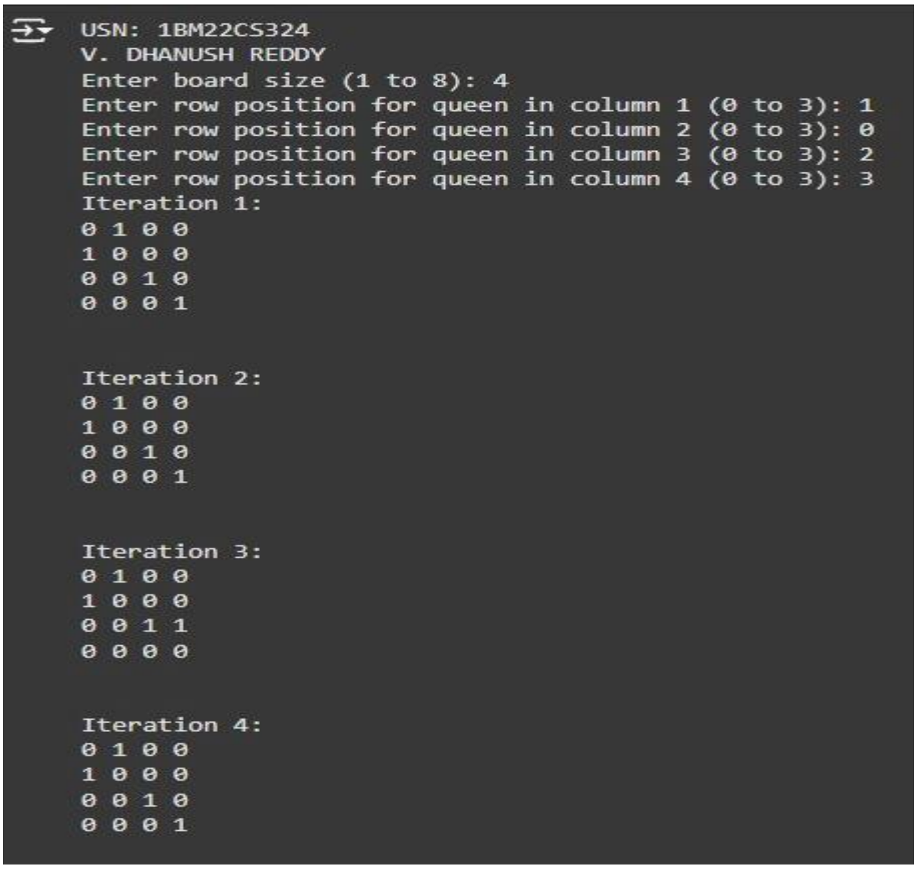
HILL CLIMBING

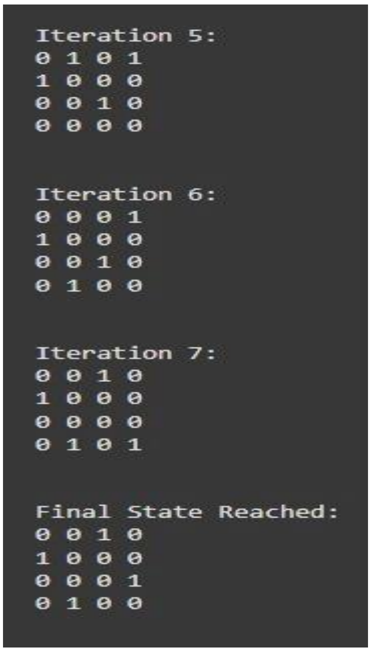
IMPLEMENTATION

```
print("USN: 1BM22CS324")
print("V. DHANUSH REDDY")
def count_conflicts(state):
    conflicts = 0
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j]:
                conflicts += 1
            if abs(state[i] - state[j]) == abs(i - j):
                conflicts += 1
    return conflicts

def generate_neighbors(state):
    neighbors = []
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            neighbor = state[:]
            neighbor[i], neighbor[j] = neighbor[j], neighbor[i]  # Swap positions of queens i and j
            neighbors.append(neighbor)
    return neighbors

def hill_climbing(n, initial_state):
    state = initial_state
    while True:
        current_conflicts = count_conflicts(state)
        if current_conflicts == 0:
            return state
        neighbors = generate_neighbors(state)
        best_neighbor = None
        best_conflicts = float('inf')
        for neighbor in neighbors:
            conflicts = count_conflicts(neighbor)
            if conflicts < best_conflicts:
                best_conflicts = conflicts
                best_neighbor = neighbor
        if best_conflicts < current_conflicts:
            state = best_neighbor
        else:
            return None


def get_user_input(n):
    while True:
        try:
            user_input = input(f"Enter the row positions for the queens (space-separated integers between 0 and {n-1}): ")
            initial_state = list(map(int, user_input.split()))
            if len(initial_state) != n or any(x < 0 or x >= n for x in initial_state):
                print(f"Invalid input. Please enter exactly {n} integers between 0 and {n-1}.")
                continue
            return initial_state
        except ValueError:
            print(f"Invalid input. Please enter a list of {n} integers.")


def print_board(state):
    n = len(state)
    for row in range(n):
        board = ['Q' if col == state[row] else '.' for col in range(n)]
        print(' '.join(board))
    print()  # Add a newline for better readability

# Main program logic
n = 4
initial_state = get_user_input(n)

print("User's Initial Board:")
print_board(initial_state)  # Display the user's initial configuration

solution = hill_climbing(n, initial_state)

if solution:
    print("Solution found!")
    print_board(solution)  # Display the solved board
else:
    print("No solution found (stuck in local minimum).")
```

OUTPUT:

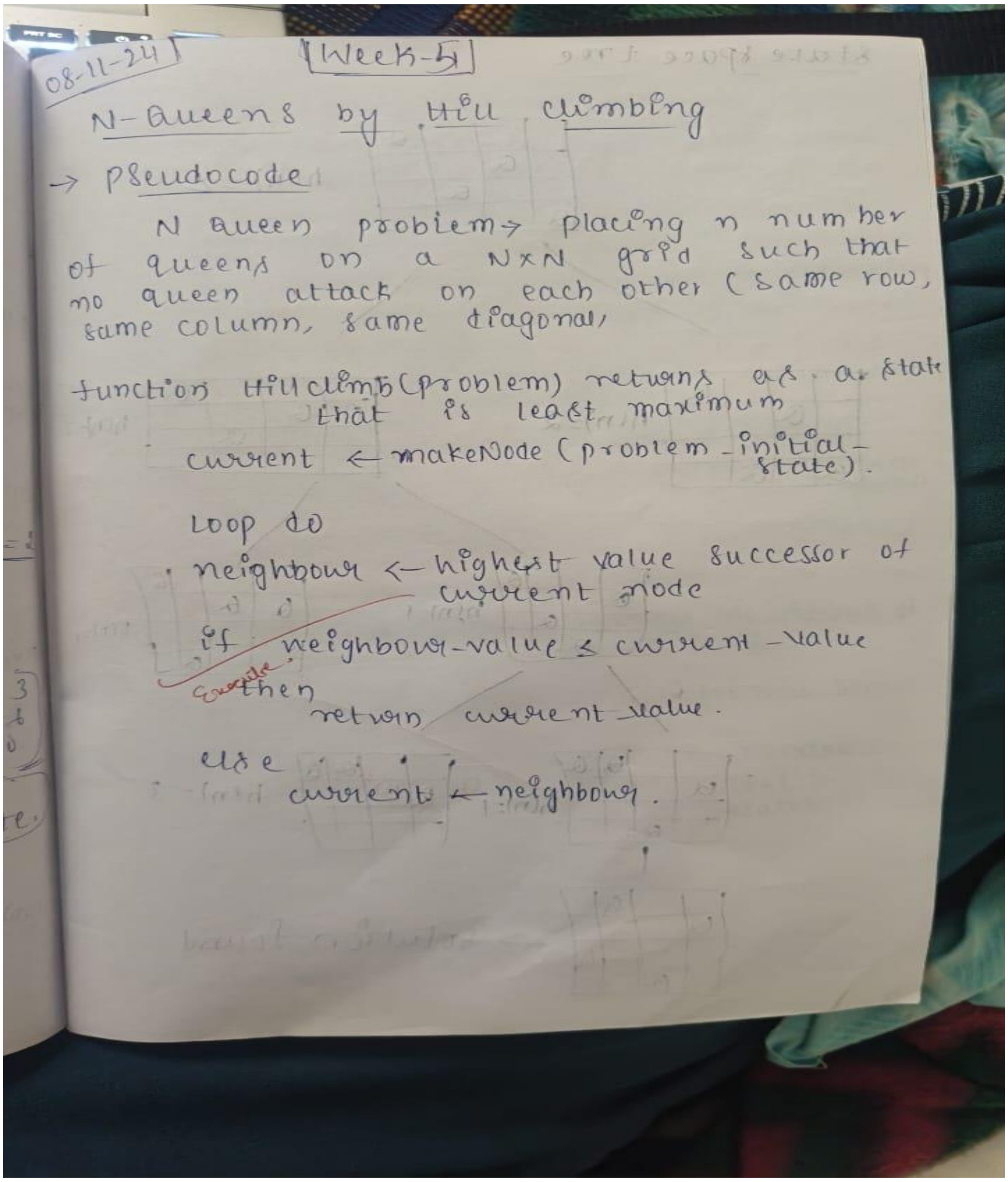


```
USN: 1BM22CS324
V. DHANUSH REDDY
Enter board size (1 to 8): 4
Enter row position for queen in column 1 (0 to 3): 1
Enter row position for queen in column 2 (0 to 3): 0
Enter row position for queen in column 3 (0 to 3): 2
Enter row position for queen in column 4 (0 to 3): 3
Iteration 1:
0 1 0 0
1 0 0 0
0 0 1 0
0 0 0 1


Iteration 2:
0 1 0 0
1 0 0 0
0 0 1 0
0 0 0 1


Iteration 3:
0 1 0 0
1 0 0 0
0 0 1 1
0 0 0 0


Iteration 4:
0 1 0 0
1 0 0 0
0 0 1 0
0 0 0 1
```



```
Iteration 5:
0 1 0 1
1 0 0 0
0 0 1 0
0 0 0 0


Iteration 6:
0 0 0 1
1 0 0 0
0 0 1 0
0 1 0 0


Iteration 7:
0 0 1 0
1 0 0 0
0 0 0 0
0 1 0 1


Final State Reached:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

OBSERVATION



08-11-24

[Week-5]

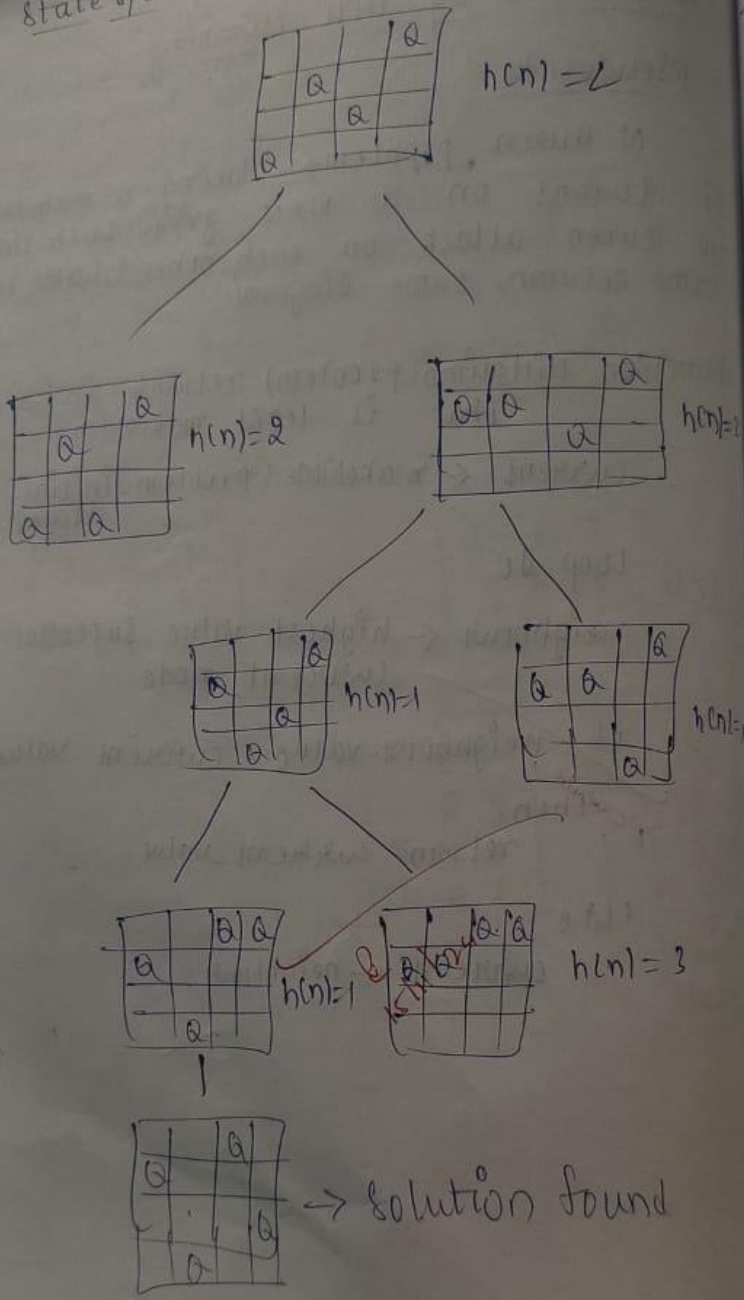
N-Queens by Hill climbing

→ Pseudocode

N Queen problem → placing n number of queens on a N×N grid such that no queen attack on each other (same row, same column, same diagonal,

```
function HillClimb(problem) returns as a state
                that is least maximum
    current ← makeNode (problem _initial_state).

    Loop do
    neighbour ← highest value successor of
                    current node
    if neighbour-value ≤ current -value
    Execute then
        return current_value.
    else
        current ← neighbour.
```

State space tree



15/11/25

Ⓠ Imp

solve

→ Fun

Func

wh

if

ret