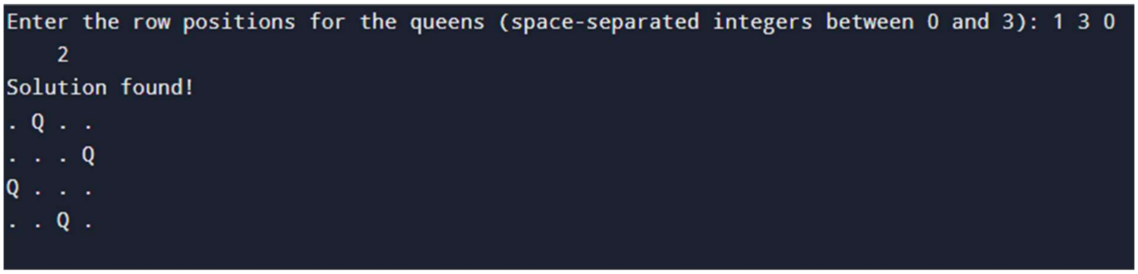
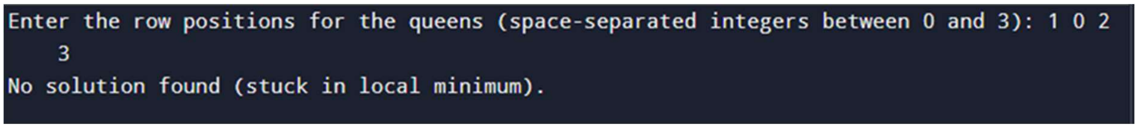
```
def count_conflicts(state):
    conflicts = 0
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j]:  # Same column
                conflicts += 1
            elif abs(state[i] - state[j]) == abs(i - j):  # Same diagonal
                conflicts += 1
    return conflicts

def generate_neighbors(state):
    neighbors = []
    n = len(state)
    for i in range(n):
        # Try moving the queen in column i to a different row
        for j in range(n):
            if j != state[i]:  # Don't move to the same row
                neighbor = state[:]
                neighbor[i] = j  # Move queen to a new row in the same column
                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  # Found a solution

        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  # No improvement, stuck in local minimum

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(solution):
    n = len(solution)
    for row in range(n):
        board = ['Q' if col == solution[row] else '.' for col in range(n)]
        print(' '.join(board))


# Main Execution
n = 4  # You can change this to any n for different sizes of the board
initial_state = get_user_input(n)


solution = hill_climbing(n, initial_state)


if solution:
    print("Solution found!")
    print_board(solution)
else:
    print("No solution found (stuck in local minimum).")
```



```
Enter the row positions for the queens (space-separated integers between 0 and 3): 1 3 0
    2
Solution found!
. Q . .
. . . Q
Q . . .
. . Q .
```



```
Enter the row positions for the queens (space-separated integers between 0 and 3): 1 0 2
    3
No solution found (stuck in local minimum).
```