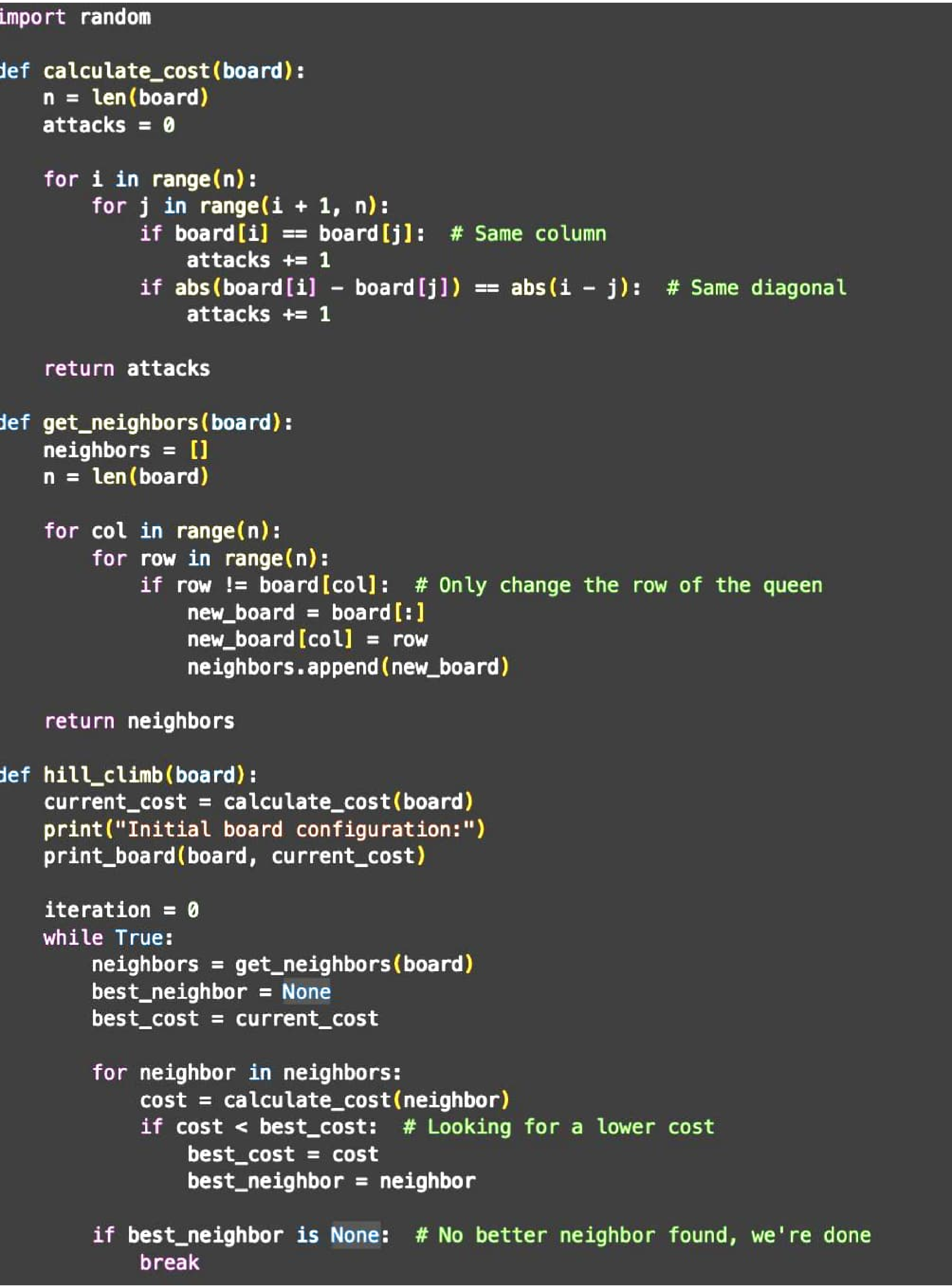
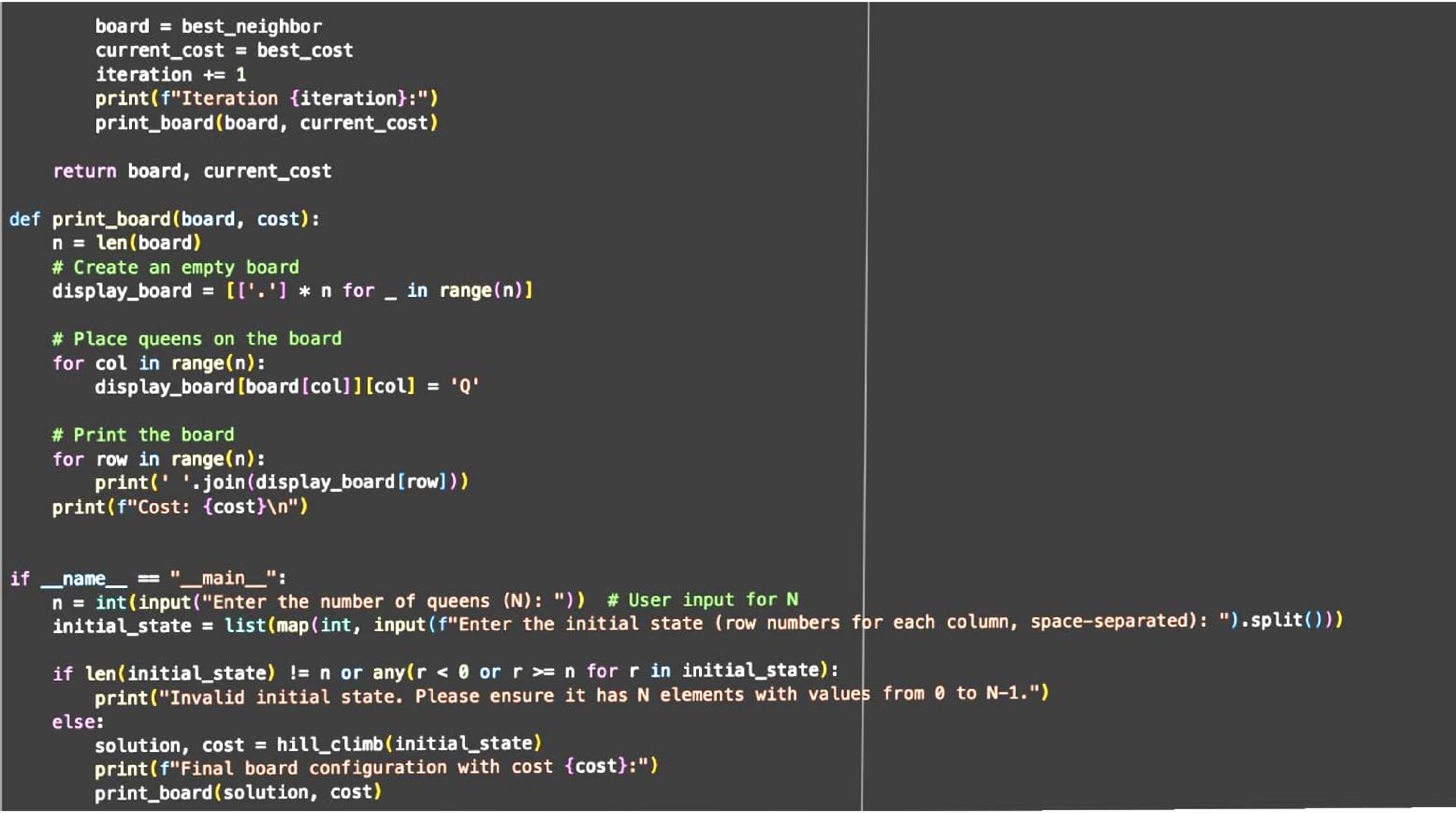
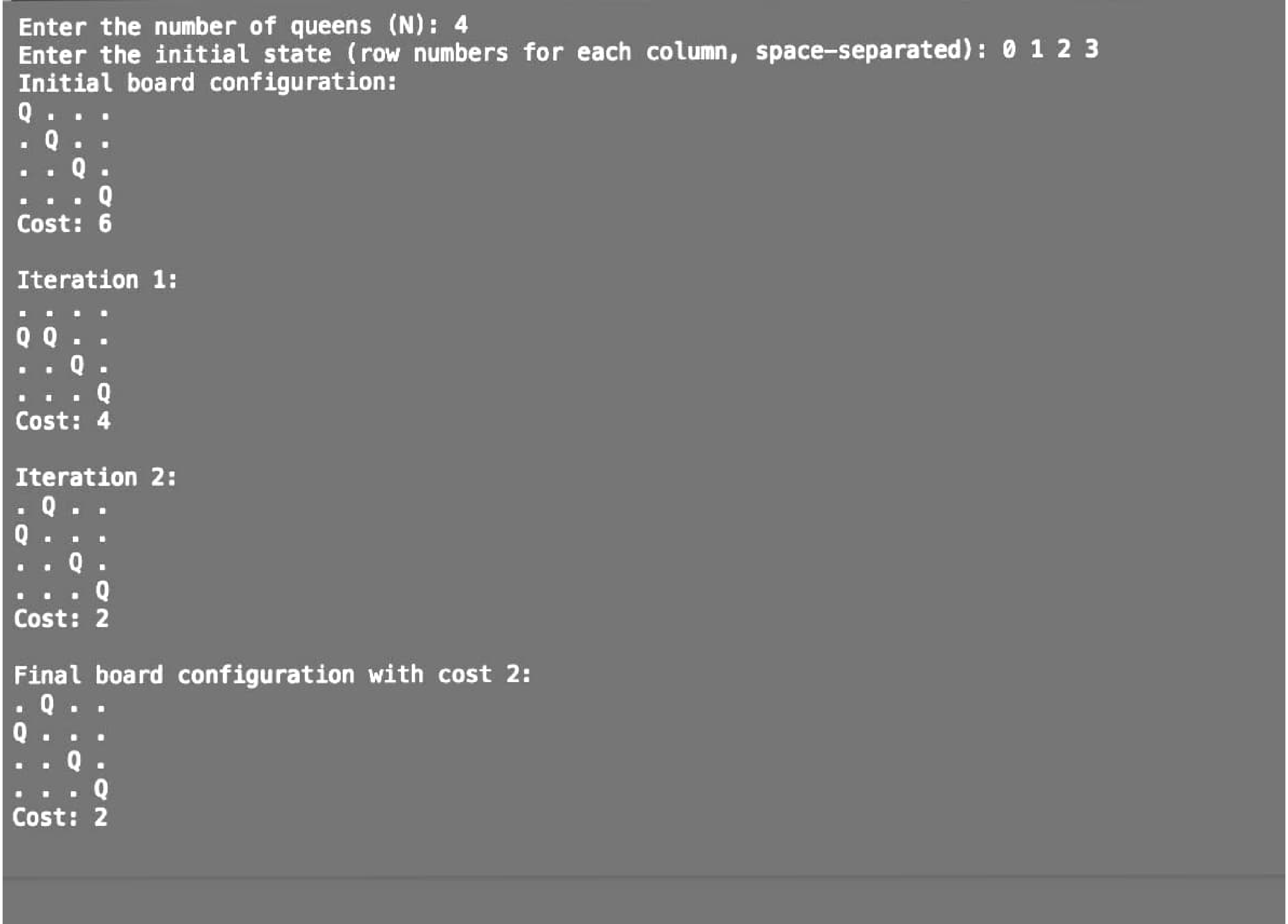


```
import random

def calculate_cost(board):
    n = len(board)
    attacks = 0

    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j]:  # Same column
                attacks += 1
            if abs(board[i] - board[j]) == abs(i - j):  # Same diagonal
                attacks += 1

    return attacks

def get_neighbors(board):
    neighbors = []
    n = len(board)

    for col in range(n):
        for row in range(n):
            if row != board[col]:  # Only change the row of the queen
                new_board = board[:]
                new_board[col] = row
                neighbors.append(new_board)

    return neighbors

def hill_climb(board):
    current_cost = calculate_cost(board)
    print("Initial board configuration:")
    print_board(board, current_cost)

    iteration = 0
    while True:
        neighbors = get_neighbors(board)
        best_neighbor = None
        best_cost = current_cost

        for neighbor in neighbors:
            cost = calculate_cost(neighbor)
            if cost < best_cost:  # Looking for a lower cost
                best_cost = cost
                best_neighbor = neighbor

        if best_neighbor is None:  # No better neighbor found, we're done
            break
```



```
        board = best_neighbor
        current_cost = best_cost
        iteration += 1
        print(f"Iteration {iteration}:")
        print_board(board, current_cost)

    return board, current_cost

def print_board(board, cost):
    n = len(board)
    # Create an empty board
    display_board = [['.'] * n for _ in range(n)]

    # Place queens on the board
    for col in range(n):
        display_board[board[col]][col] = 'Q'

    # Print the board
    for row in range(n):
        print(' '.join(display_board[row]))
    print(f"Cost: {cost}\n")


if __name__ == "__main__":
    n = int(input("Enter the number of queens (N): "))  # User input for N
    initial_state = list(map(int, input(f"Enter the initial state (row numbers for each column, space-separated): ").split()))

    if len(initial_state) != n or any(r < 0 or r >= n for r in initial_state):
        print("Invalid initial state. Please ensure it has N elements with values from 0 to N-1.")
    else:
        solution, cost = hill_climb(initial_state)
        print(f"Final board configuration with cost {cost}:")
        print_board(solution, cost)
```



```
Enter the number of queens (N): 4
Enter the initial state (row numbers for each column, space-separated): 0 1 2 3
Initial board configuration:
Q . . .
. Q . .
. . Q .
. . . Q
Cost: 6

Iteration 1:
. . . .
Q Q . .
. . Q .
. . . Q
Cost: 4

Iteration 2:
. Q . .
Q . . .
. . Q .
. . . Q
Cost: 2

Final board configuration with cost 2:
. Q . .
Q . . .
. . Q .
. . . Q
Cost: 2
```