Week - 4

Implement Hill Climbing

Algorithm:

Week-4

Implement Hill Climbing search algorithm to solve N-Queens problem.

Algorithm/pseudocode:

```
function Hill-Climbing returns a state
that is a local maximum.
current ← Make-node (problem.Initial-state)
loop do
    neighbour ← a highest-valued successor
                of current
    if neighbour.Value ≤ current.value
        then return current.state
    current ← neighbour.
```

Solve 4-Queens:

	0	1	2	3
x_0				Q
x_1		Q		
x_2			Q	
x_3	Q			

$x_0 = 3, x_1 = 1, x_2 = 2, x_3 = 0$

	0	1	2	3
x_0			Q	
x_1		Q		Q
x_2			Q	
x_3	Q			

$x_0 = 2, x_1 = 1$
$x_2 = 2, x_3 = 0$

	0	1	2	3
x_0			Q	
x_1		Q		
x_2				Q
x_3	Q			

$x_0 = 2, x_1 = 1$
$x_2 = 3, x_3 = 0$

	0	1	2	3
x_0			Q	
x_1	Q			
x_2				Q
x_3	Q			

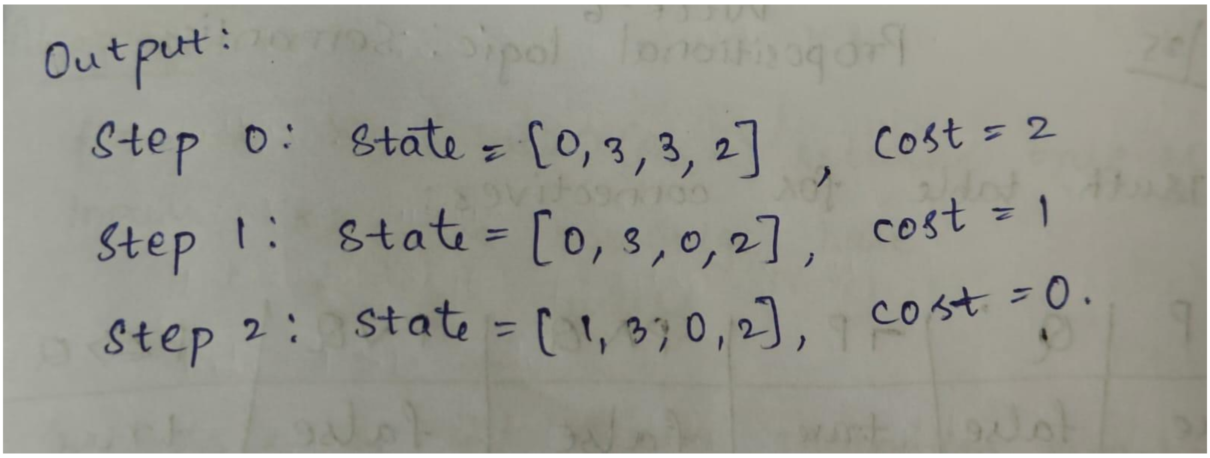
$x_0 = 2, x_1 = 0$
$x_2 = 3, x_3 = 0$

	0	1	2	3
x_0			Q	
x_1	Q			
x_2				Q
x_3		Q		

$x_0 = 2, x_1 = 0$
$x_2 = 3, x_3 = 1$

	0	1	2	3
x_0			Q	
x_1	Q			
x_2				Q
x_3		Q		

$x_0 = 2, x_2 = 3$
$x_1 = 0, x_3 = 1$



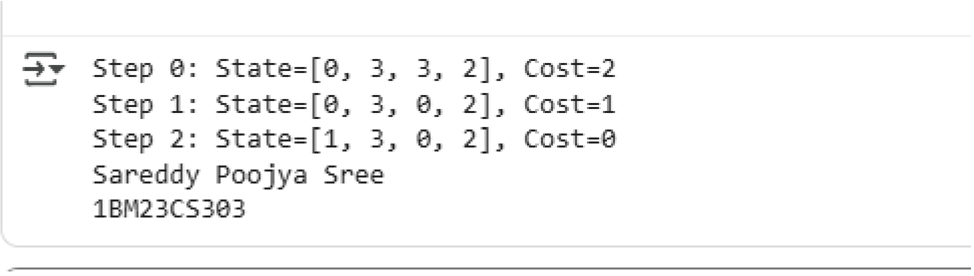
Output:

Step 0: State = [0, 3, 3, 2] , cost = 2

Step 1: State = [0, 3, 0, 2], cost = 1

Step 2: State = [1, 3, 0, 2], cost = 0.

Output:



```
Step 0: State=[0, 3, 3, 2], Cost=2
Step 1: State=[0, 3, 0, 2], Cost=1
Step 2: State=[1, 3, 0, 2], Cost=0
Sareddy Poojya Sree
1BM23CS303
```

Code:

```
def hill_climbing_with_restarts(n):
    max_restarts = 1000  # limit restarts to avoid infinite loop
    restart_count = 0

    while restart_count < max_restarts:
        # Start with a random initial state
        current = [random.randint(0, n-1) for _ in range(n)]
        steps = []
        step_count = 0

        while True:
            current_cost = calculate_cost(current)
            steps.append((current, current_cost))
            print(f"Step {step_count}: State={current},
Cost={current_cost}")
            step_count += 1

            if current_cost == 0:  # solution found

                return steps

            neighbors = generate_neighbors(current)
            neighbor_costs = [(n, calculate_cost(n)) for n in
neighbors]
```

```
            best_neighbor, best_cost = min(neighbor_costs, key=lambda x: x[1])

            if best_cost >= current_cost:
                # no improvement, restart with a new random state
                print(f"Restarting after {step_count} steps, cost stuck at {current_cost}")
                restart_count += 1
                break

            current = best_neighbor

    print("Failed to find a solution after max restarts.")
    return None

steps = hill_climbing_with_restarts(4)
print("Sareddy Poojya Sree\n1BM23CS303")
```