Name : Devashish Rai

Roll No. : 20075025

Batch : B.tech CSE 4year

Email Id : devashish.rai.cse20@itbhu.ac.in

Q. Explain how Simulated Annealing is better than Hill Climbing.

Ans. Hill Climbing attempts to reach an optimum value by checking if its current state has the best cost/score in its neighborhood, so as we move for the value which is better than the current state, this makes it prone to getting stuck in local optima.

Simulated Annealing can be considered as an optimized way of hill climbing. In this method, we attempt to overcome this problem by choosing a "bad" move every once in a while. The probability of choosing a "bad" depends on the temperature of the current state. Temperature decreases as time moves on, and eventually, Simulated Annealing becomes Hill Climbing.

If appropriately configured, simulated Annealing has better chances of finding global optima than hill-climbing, whereas hill-climbing sticks on optima nearest to starting state.

Q. Solve the nQueen's problem using Hill Climbing algorithm.

Ans.

Solution for n queen problem in python:

```
import random

n = int(input())

# Create a n * n list to store queens positions.
v = [[0 for _ in range(n)] for _ in range(n)]
for i in range(n):
    p = random.randint(0, n - 1)
    v[i][p] = 1


def check_state(x):
    val = 0
    for i in range(n):
        for j in range(n):
            if x[i][j] == 1:
                for k in range(i + 1, n):
                    if x[k][j] == 1:
                        val += 1

                for k in range(j + 1, n):
                    if x[i][k] == 1:
                        val += 1

                l, m = i + 1, j + 1
                while l < n and m < n:
                    if x[l][m] == 1:
```

```
                    val += 1
                  l += 1
                  m += 1

              l, m = i - 1, j + 1
              while l > 0 and m < n:
                if x[l][m] == 1:
                    val += 1
                  l -= 1
                  m += 1
    return val


cur = check_state(v)
z = []
for i in range(n):
    for j in range(n):
        if v[i][j] == 1:
            z.append([i, j])


def ret_copy():
    d = []
    for i in v:
        d.append(i.copy())

    return d

while True:
    ch = cur
    s = [0, 0, 0, 0]
    for i in range(n):
```

```
w = ret_copy()
q = z[i]
if w[q[0]][(q[1] + 1) % n] == 0:
    w[q[0]][q[1]] = 0
     w[q[0]][(q[1] + 1) % n] = 1
     state = check_state(w)
    if state < ch:
        ch = state
        s = [q[0], q[1], q[0], (q[1] + 1) % n]

w = ret_copy()
r = q[1] - 1
 if r < 0:
    r = n - 1
 if w[q[0]][r] == 0:
    w[q[0]][q[1]] = 0
     w[q[0]][r] = 1
     state = check_state(w)
    if state < ch:
        ch = state
        s = [q[0], q[1], q[0], r]

w = ret_copy()
r = q[0] - 1
 if r < 0:
    r = n - 1
 if w[r][q[1]] == 0:
    w[q[0]][q[1]] = 0
     w[r][q[1]] = 1
     state = check_state(w)
    if state < ch:
        ch = state
```

```
                s = [q[0], q[1], r, q[1]]
        w = ret_copy()
        r = (q[0] + 1) % n
        if w[r][q[1]] == 0:
            w[q[0]][q[1]] = 0
             w[r][q[1]] = 1
             state = check_state(w)
            if state < ch:
                ch = state
                s = [q[0], q[1], r, q[1]]

    if ch < cur:
        cur = ch
        v[s[0]][s[1]] = 0
         v[s[2]][s[3]] = 1
    else:
        if ch > 0:
            v = [[0 for _ in range(n)] for _ in range(n)]
            for i in range(n):
                p = random.randint(0, n - 1)
                v[i][p] = 1
             z = []
            for i in range(n):
                for j in range(n):
                    if v[i][j] == 1:
                        z.append([i, j])
        else:
            break


for i in v:
    print(*i)
```