Lab-6

→ 8 Queens Using Hill Climbing

① import random

② def - calculate_attacks ()

attack = 0

for i in range (len, state)

for j in range (len+1, state)

if (state(i) == state(j)] or abs state [i-j] = abs state (j-1]

attack += 1

return attack

③ def - hill climbing ()

state = random. randint (0,7) for in range 8

Calculat_attacks = current-attacks

④ for _ in range ()

neighbours = ( )

for row in range (8)

for col in range (8)

state (row) != state (col)

neighbours = state

next_state = min (neighbour, Calculat_attack)

next_cuttack = Calculate_attack.

if next_attack > current_attack (It calculates the no of attacks) co.

break

len no of attacks.

⑤ Display current_attack
→ (print)

⑥ if current_attack > next_attack

print

else
break.

→ A* :-

.import heapq

① def - heuristic (self, cost)
self.g = g
self.f = f
self.h = g + f

② def - calculate_attacks()
attack = 0
for i in range (len, state)
for j in range (len +1, state)
if (state(i) == state[j]) or abs (state[i - j]
return attack

③ def - A Star (open set [], neighbours)
for row in range (8)
for col in range (8)
State [row] != State [col]

heap (state) = heap push [open set [], state)
heap pop [min (state)]

④ open set [] → array
take open set & sort the los minimum. (from heuristic fu

⑤ display (open set [])

→ A* :-

1. import heapq

(1) def - heuristic (self, goal):

self.g = g

self.h = h

self.h = g + h

(2) def - calculate_attacks():

attack = 0

for i in range (len, state)

for j in range (len + 1, state)

if (state(i) == state[j]) or abs (state[i - j] = abs (state[j - i])

return attack

(3) def - A star (open set[], neighbours)

for row in range (8)

for col in range (8)

state[row] != state[col]

heapq.heappush(open set[], state)

heapq.heappop(calculate(state))

(4) open set[] → array

take open set & sort the list A & it sorts the minimum. (from heuristic function)

(5) display (open set())

Hill Climb :-

```
import random

def calculate_attacks (state)

    attacks = 0
    for i in range( len(state)):
        for j in range (i+1, len(state))
            if state[i] == state[j] or abs(state[i] - state[j]) == j - i
                attacks += 1
    return attacks

def hill_climbing_8_queens():
    State = [random.randint(0, 7) for _ in range(8)]
    current_attacks = (calculate_attacks (state)

    for _ in range(100):
        neighbours = []
        for row in range(8):
            for col in range(8):
                if state[row] != col:
                    neighbour = state[:]
                    neighbour[row] = col
                    neighbours.append(neighbour)
    next_state = min(neighbours, (calculate_attacks)
    next_attacks = (calculate_attacks(next_state)
    if next_attaks >= current_attacks:
        break
```

```
        state = next_state
        current_attacks = next_attacks
    return state, current_attacks

def display_board(state):
    for row in range(8):
        line = ""
        for col in range(8):
            if state[row] == col:
                line += "Q "
            else:
                line += ". "
        print(line)
    print()

best_solution = None
best_attacks = float('inf')
attempts = 100

for _ in range(attempts):
    solution, attacks = hill_climbing_8_queens()
    if attacks < best_attacks:
        best_solution = solution
        best_attacks = attacks
    if best_attacks == 0:
        break

if best_solution:
    print(f"Best solution ({best_attacks}):")
    display_board(best_solution)
else:
    print("No solution found.")
```