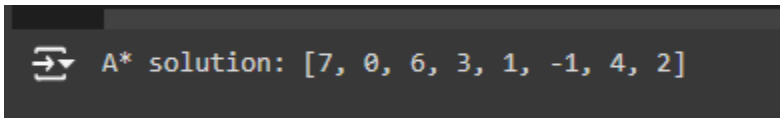
LAB-6 - Implementing A* and Hill Climbing Algorithm on 8 Queens.

Code: A*

```
import numpy as np
import heapq

class Node:
    def __init__(self, state, g, h):
        self.state = state  # current state of the board
        self.g = g          # cost to reach this state
        self.h = h          # heuristic cost to reach goal
        self.f = g + h      # total cost
    def __lt__(self, other):
        return self.f < other.f

def heuristic(state):
    # Count pairs of queens that can attack each other
    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 a_star_8_queens():
    initial_state = [-1] * 8  # -1 means no queen placed
    open_list = []
    closed_set = set()

    initial_h = heuristic(initial_state)
    heapq.heappush(open_list, Node(initial_state, 0, initial_h))

    while open_list:
        current_node = heapq.heappop(open_list)
        current_state = current_node.state
        closed_set.add(tuple(current_state))

        # Check if we reached the goal
```

```
        if current_node.h == 0:
            return current_state

        for col in range(8):
            for row in range(8):
                if current_state[col] == -1:  # Only place a queen if none is present in this column
                    new_state = current_state.copy()
                    new_state[col] = row
                    if tuple(new_state) not in closed_set:
                        g_cost = current_node.g + 1
                        h_cost = heuristic(new_state)
                        heapq.heappush(open_list, Node(new_state, g_cost, h_cost))

    return None

solution = a_star_8_queens()
print("A* solution:", solution)
```

Output:

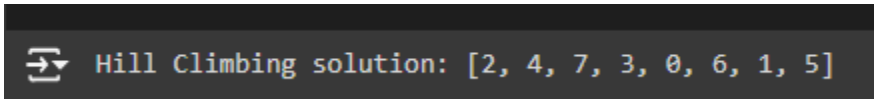


Code: Hill Climbing

```
import random

def heuristic(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)]  # Random initial state
    while True:
        current_h = heuristic(state)
        if current_h == 0:  # Found a solution
            return state

        next_state = None
        next_h = float('inf')

        for col in range(8):
            for row in range(8):
                if state[col] != row:  # Only consider moving the queen
                    new_state = state.copy()
                    new_state[col] = row
                    h = heuristic(new_state)
                    if h < next_h:
                        next_h = h
                        next_state = new_state

        if next_h >= current_h:  # No better neighbor found
            return None  # Stuck at local maximum
        state = next_state

solution = hill_climbing_8_queens()
print("Hill Climbing solution:", solution)
```

Output:

If there is a solution:



If there is no a solution: