29-10-24

Lab - 06

★ Solve 8 queen's problem using hill-climbing algorithm

1) Function to calculate the number of attacks in each current state

```
def calc_attack(state):
    for i in range(8):
        for j in range(i+1, 8):
            if state[i] == state[j]: attack++
            if abs(state[i] - state[j]) == abs(i - j): attack++
    return attack
```

2) Function → main fn

```
def hill_climb():
    state = (random.randint(0, 7) for i in range(8))
    curr_attack = calc_attack(state)

    for iter in range(100):
        neighbours = []
        for row in range(8):
            for col in range(8):
                if state[row] != col:
                    neighbour = state
                    neighbour[row] = col

        next_state = min(neighbours, key = calc_attack)
        next_attack = calc_attack(next_state)
        if next_attack >= curr_attack: break
        state = nex_state
        attack = next_attack
```

3) Function to display the board

```
def display(state):
    for i in range(8):
        for j in range(8):
            if state[i] == j: print("Q")
            else: print(".")
```

29/10/24

* Solve the 8 queen's problem using A* algorithm

① Function to calculate the heuristic value of states
→ The heuristic value for 8 queens is the number of attacks

```
def cale_attacks (state):
    for i in range(8):
        for j in range(8):
            if state[i] == state[j]: attack++
            if abs(state[i] - state[j]) == (i - j)  attack++
return attack
```

② Function to implement A* algorithm

```
def A*():
    import heapq  # import a heap data structure
    state = []
    g = 8  # queens left
    for i in range(8):
        for j in range(8):
            f = cale_attacks (state) + g
            heap_push (state, f)
            g --
    for i in range(8):
        curr_state = heap_pop()
        state.append (curr_state)
    if (g == 0): break
```

③ Display the board

```
for i in range(8):
    for j in range(8):
        if state[i] == state[j]: print("Q")
        else: print(".")
```