LAB 6

A\* Algorithm code for 8 Queens:

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)

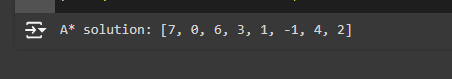
heapq.heappush(open\_list, Node(new\_state, g\_cost, h\_cost))

return None

solution = a\_star\_8\_queens()

print("A\* solution:", solution)

output:-



Hill Climbing for 8 queens

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



Observation book

