# AI LAB ASSESSMENT

Aritro Pal

101803560

Q1.

import random import math

def get\_cost(state):

dec = int(str(state), 2)

cost = (dec\*\*3) - (60\*(dec\*\*2)) + (900\*dec) + 100 return cost

def get\_random\_neighbour(state):

res = [int(x) for x in str(state)]

if(len(res)<5):

n = 5 - len(res) for i in range(n):

res.insert(0, 0)

i = random.choice(range(len(res))) if res[i] == 1:

res[i] = 0 else:

res[i] = 1

result = int("".join(str(i) for i in res)) return(result)

def simulated\_annealing(init\_state):

T = 500

delta\_T = 0.9

n = 500

cur\_state = init\_state

print('Initial cost: ' + str(get\_cost(cur\_state))) for i in range(n):

neighbour = get\_random\_neighbour(cur\_state)

cost\_diff = get\_cost(neighbour) - get\_cost(cur\_state) if cost\_diff > 0:

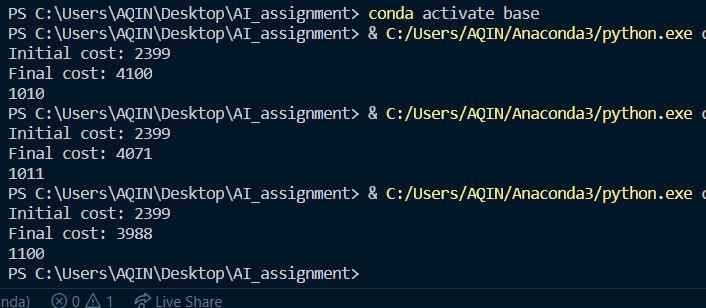
cur\_state = neighbour else:

if random.uniform(0,1) < math.exp(cost\_diff/T): cur\_state = neighbour

T = T - delta\_T

print('Final cost: ' + str(get\_cost(cur\_state))) return cur\_state

print(simulated\_annealing(10011))



Q2.

|  |  |  |
| --- | --- | --- |
| from queue import Queue, PriorityQueue  def get\_weight(from\_node, to\_node, weights=None):  return weights.get((from\_node, to\_node), 10e100) if weights else 1  def dfs\_cost(graph, start, goal, weights, max\_depth): visited = []  path = [start]  pq = PriorityQueue()  pq.put((0, start, path, visited)) depth = 0  while not pq.empty() and depth <= max\_depth: cost, current\_node, path, visited = pq.get() if current\_node == goal:  print(path) return True  visited = visited + [current\_node] child\_nodes = graph[current\_node]  for node in child\_nodes: if node not in visited:  cost\_of\_node = cost + get\_weight(current\_node, node, weights) pq.put((cost\_of\_node, node, path + [node], visited))  depth = depth + 1  def ids(graph, start, goal, weights, depth): for i in range(depth):  if(dfs\_cost(graph, start, goal, weights, i)): return  return False  graph = {  'S': ['A', 'B', 'C'],  'A': ['S', 'D'],  'B': ['S', 'G', 'D'],  'C': ['S', 'G'],  'D': ['A', 'B'],  'G': ['B', 'C']  }  weights = { | | |
| ('S', | 'A'): | 10, |
| ('S', | 'B'): | 8, |
| ('S', | 'C'): | 9, |
| ('A', | 'D'): | 1, |
| ('A', | 'S'): | 10, |
| ('B', | 'S'): | 8, |
| ('B', | 'G'): | 5, |
| ('B', | 'D'): | 4, |
| ('C', | 'S'): | 9, |
| ('C', | 'G'): | 5, |
| ('D', | 'A'): | 1, |

('D', 'B'): 4,

('G', 'B'): 5,

('G', 'C'): 5,

}

ids(graph, 'S', 'G', weights, 10)

