Ai lab 3

22k-4595

Q1.

class GBA:

def \_\_init\_\_(self, graph, goal, depth\_limit):

self.graph = graph

self.goal = goal

self.depth\_limit = depth\_limit

def depth\_limited\_search(self, node, depth):

if node == self.goal:

return [node]

if depth == 0:

return None

for neighbor in self.graph.get(node, []):

path = self.depth\_limited\_search(neighbor, depth - 1)

if path:

return [node] + path

return None

def search(self, start):

return self.depth\_limited\_search(start, self.depth\_limit)

graph = {

'A': ['B', 'C'],

'B': ['D', 'E'],

'C': ['F', 'G'],

'D': [],

'E': ['H'],

'F': [],

'G': [],

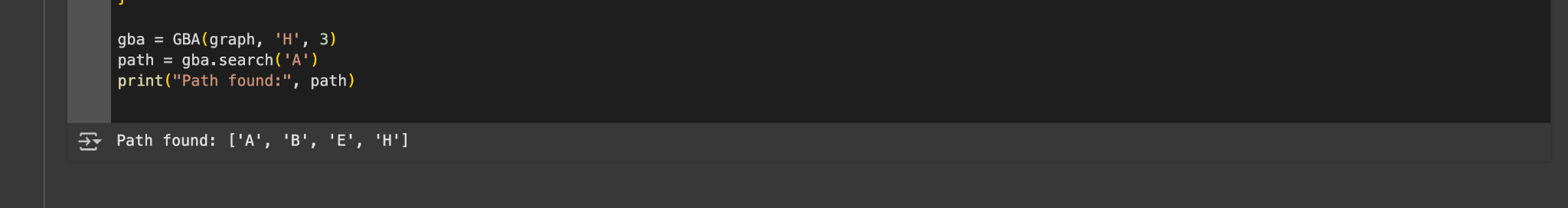
'H': []

}

gba = GBA(graph, 'H', 3)

path = gba.search('A')

print("Path found:", path)



Q1b.

class UCSAgent:

def \_\_init\_\_(self, graph, goal):

self.graph = graph

self.goal = goal

def uniform\_cost\_search(self, start):

queue = [(0, start, [])]

visited = set()

while queue:

queue.sort()

cost, node, path = queue.pop(0)

if node in visited:

continue

visited.add(node)

path.append(node)

if node == self.goal:

return path, cost

for neighbor, weight in self.graph.get(node, []):

queue.append((cost + weight, neighbor, path.copy()))

return None, float("inf")

graph = {

'A': [('B', 1), ('C', 4)],

'B': [('D', 2), ('E', 5)],

'C': [('F', 3)],

'D': [],

'E': [('H', 2)],

'F': [],

'G': [],

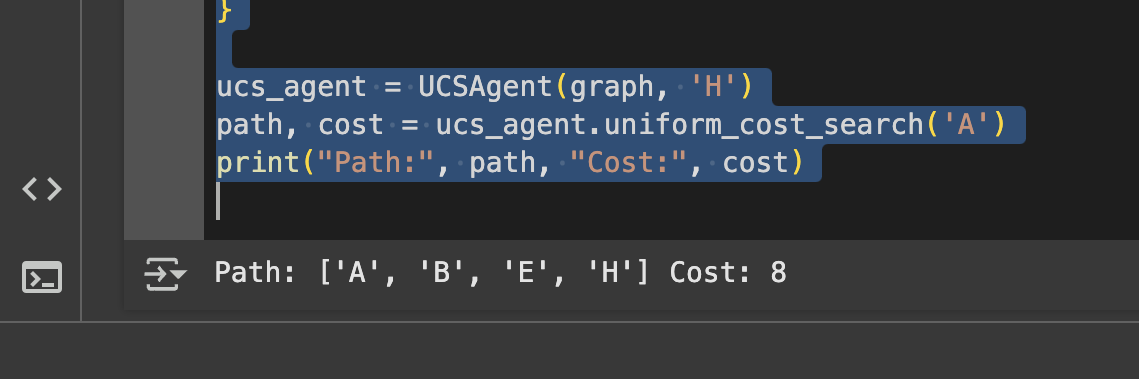
'H': []

}

ucs\_agent = UCSAgent(graph, 'H')

path, cost = ucs\_agent.uniform\_cost\_search('A')

print("Path:", path, "Cost:", cost)



Q2.

def tsp(graph):

n = len(graph)

visited = [False] \* n

path = [0] # Start from city 0

visited[0] = True

cost = 0

for \_ in range(n - 1):

last = path[-1]

nearest\_city = -1

min\_dist = float('inf')

for city in range(n):

if not visited[city] and graph[last][city] < min\_dist:

min\_dist = graph[last][city]

nearest\_city = city

path.append(nearest\_city)

visited[nearest\_city] = True

cost += min\_dist

path.append(0) # Return to start

cost += graph[path[-2]][0]

return path, cost

graph = [

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

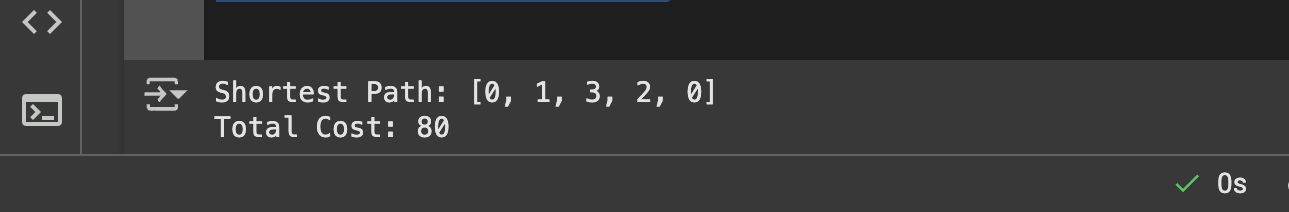
[20, 25, 30, 0]

]

path, cost = tsp(graph)

print("Shortest Path:", path)

print("Total Cost:", cost)



Q3.

def dls(node, goal, graph, depth, visited):

if depth < 0:

return False

if node == goal:

return True

visited.add(node)

for neighbor in graph.get(node, []):

if neighbor not in visited and dls(neighbor, goal, graph, depth - 1, visited):

return True

return False

def iddfs(start, goal, graph, max\_depth):

for depth in range(max\_depth):

visited = set()

if dls(start, goal, graph, depth, visited):

return True

return False

# Example Graph (Adjacency List Representation)

graph = {

0: [1, 2],

1: [3, 4],

2: [5, 6],

3: [],

4: [],

5: [],

6: []

}

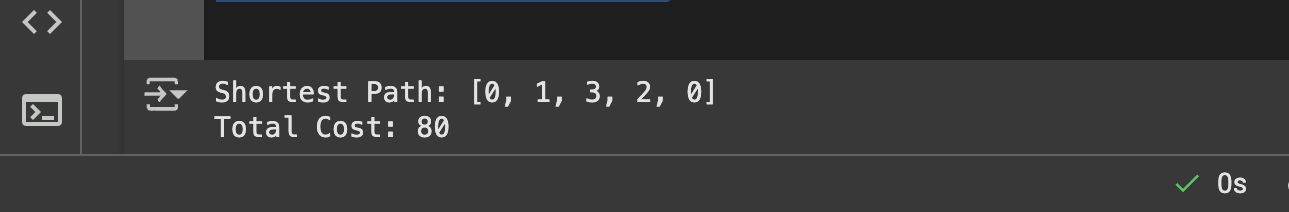
start\_node = 0

goal\_node = 5

max\_depth = 4

found = iddfs(start\_node, goal\_node, graph, max\_depth)

print("Goal Found:" if found else "Goal Not Found")



Q4.

def bfs(grid, start, goal):

rows, cols = len(grid), len(grid[0])

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] # Up, Down, Left, Right

queue = [(start, [start])]

visited = set()

visited.add(start)

while queue:

current, path = queue.pop(0) # Use list instead of deque

if current == goal:

return path # Return the shortest path

x, y = current

for dx, dy in directions:

nx, ny = x + dx, y + dy

if 0 <= nx < rows and 0 <= ny < cols and grid[nx][ny] == 0 and (nx, ny) not in visited:

queue.append(((nx, ny), path + [(nx, ny)]))

visited.add((nx, ny))

return None # No path found

# Example Grid (0 = open area, 1 = obstacle, S = start, G = goal)

grid = [

[0, 0, 1, 0, 0],

[0, 1, 0, 1, 0],

[0, 1, 0, 1, 0],

[0, 0, 0, 1, 0],

[1, 1, 0, 0, 0]

]

start = (0, 0) # 'S' position

goal = (4, 4) # 'G' position

path = bfs(grid, start, goal)

print("Shortest Path:", path if path else "No Path Found")

