Q1

from queue import PriorityQueue

class Node:

def \_\_init\_\_(self, position, parent=None):

self.position = position

self.parent = parent

self.g = 0

self.h = 0

self.f = 0

def \_\_lt\_\_(self, other):

return self.f < other.f

def heuristic(pos, goals):

return min(abs(pos[0] - g[0]) + abs(pos[1] - g[1]) for g in goals)

def best\_first\_search\_multiple\_goals(maze, start, goals):

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

path = []

for goal in goals:

start\_node = Node(start)

frontier = PriorityQueue()

frontier.put(start\_node)

visited = set()

while not frontier.empty():

current\_node = frontier.get()

current\_pos = current\_node.position

if current\_pos == goal:

temp\_path = []

while current\_node:

temp\_path.append(current\_node.position)

current\_node = current\_node.parent

path.extend(temp\_path[::-1])

start = goal

break

visited.add(current\_pos)

for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:

new\_pos = (current\_pos[0] + dx, current\_pos[1] + dy)

if 0 <= new\_pos[0] < rows and 0 <= new\_pos[1] < cols and maze[new\_pos[0]][new\_pos[1]] == 0 and new\_pos not in visited:

new\_node = Node(new\_pos, current\_node)

new\_node.g = current\_node.g + 1

new\_node.h = heuristic(new\_pos, goals)

new\_node.f = new\_node.h

frontier.put(new\_node)

return path

# Example run for Task #1

maze = [

[0, 0, 1, 0],

[0, 1, 0, 0],

[0, 0, 0, 1],

[1, 0, 0, 0]

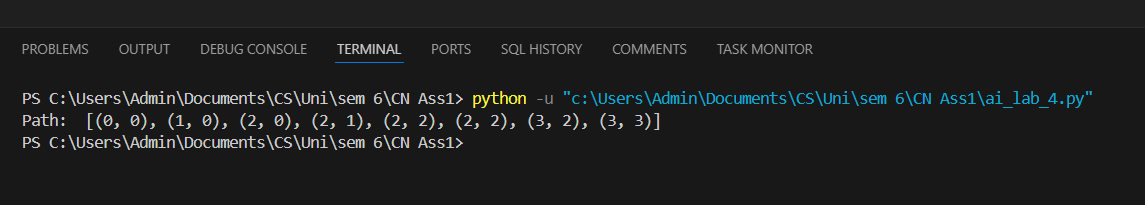
]

start = (0, 0)

goals = [(2, 2), (3, 3)]

path = best\_first\_search\_multiple\_goals(maze, start, goals)

print("Task #1 Path through multiple goals:", path)



Q2

# TASK #2: Dynamic A\* Search with Changing Edge Costs

import random

def a\_star\_dynamic(graph, start, goal, heuristic):

frontier = [(start, 0 + heuristic[start])]

visited = set()

g\_costs = {start: 0}

came\_from = {start: None}

while frontier:

frontier.sort(key=lambda x: x[1])

current\_node, \_ = frontier.pop(0)

if current\_node == goal:

path = []

while current\_node is not None:

path.append(current\_node)

current\_node = came\_from[current\_node]

return path[::-1]

visited.add(current\_node)

for neighbor in graph[current\_node]:

cost = graph[current\_node][neighbor] + random.randint(-2, 2)

new\_g\_cost = g\_costs[current\_node] + cost

f\_cost = new\_g\_cost + heuristic[neighbor]

if neighbor not in g\_costs or new\_g\_cost < g\_costs[neighbor]:

g\_costs[neighbor] = new\_g\_cost

came\_from[neighbor] = current\_node

frontier.append((neighbor, f\_cost))

return None

# Example run for Task #2

graph = {

'A': {'B': 2, 'C': 3},

'B': {'D': 4},

'C': {'D': 2},

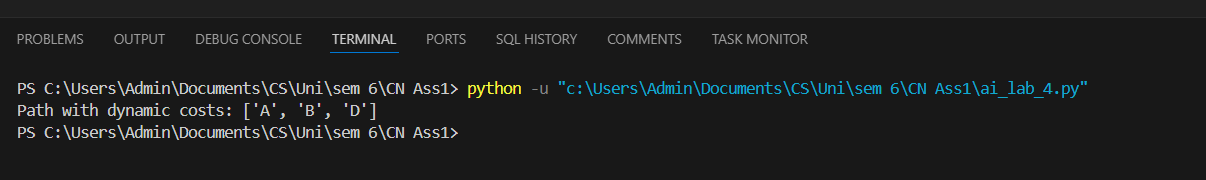
'D': {}

}

heuristic = {'A': 6, 'B': 4, 'C': 3, 'D': 0}

path = a\_star\_dynamic(graph, 'A', 'D', heuristic)

print("Task #2 Path with dynamic costs:", path)



Q3

def greedy\_bfs\_with\_time(graph, start, goal, heuristic, time\_windows):

frontier = [(start, heuristic[start])]

visited = set()

came\_from = {start: None}

while frontier:

frontier.sort(key=lambda x: x[1])

current\_node, \_ = frontier.pop(0)

if current\_node in visited:

continue

print(f"Visiting {current\_node} within time window {time\_windows.get(current\_node, 'N/A')}")

visited.add(current\_node)

if current\_node == goal:

path = []

while current\_node is not None:

path.append(current\_node)

current\_node = came\_from[current\_node]

return path[::-1]

for neighbor in graph[current\_node]:

if neighbor not in visited:

came\_from[neighbor] = current\_node

frontier.append((neighbor, heuristic[neighbor]))

return None

graph = {

'A': {'B': 2, 'C': 3},

'B': {'D': 4},

'C': {'D': 2},

'D': {}

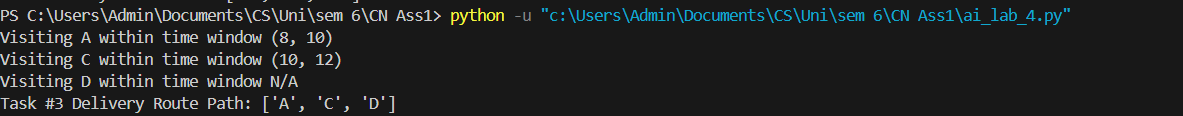
}

heuristic = {'A': 6, 'B': 4, 'C': 3, 'D': 0}

time\_windows = {'A': (8, 10), 'B': (9, 11), 'C': (10, 12)}

path = greedy\_bfs\_with\_time(graph, 'A', 'D', heuristic, time\_windows)

print("Task #3 Delivery Route Path:", path)



Q4

def a\_star\_with\_traffic(graph, start, goal, heuristic, traffic\_updates):

frontier = [(start, 0 + heuristic[start])]

visited = set()

g\_costs = {start: 0}

came\_from = {start: None}

while frontier:

frontier.sort(key=lambda x: x[1])

current\_node, \_ = frontier.pop(0)

if current\_node == goal:

path = []

while current\_node is not None:

path.append(current\_node)

current\_node = came\_from[current\_node]

return path[::-1]

visited.add(current\_node)

for neighbor in graph[current\_node]:

traffic\_cost = graph[current\_node][neighbor] + traffic\_updates.get((current\_node, neighbor), 0)

new\_g\_cost = g\_costs[current\_node] + traffic\_cost

f\_cost = new\_g\_cost + heuristic[neighbor]

if neighbor not in g\_costs or new\_g\_cost < g\_costs[neighbor]:

g\_costs[neighbor] = new\_g\_cost

came\_from[neighbor] = current\_node

frontier.append((neighbor, f\_cost))

return None

graph = {

'A': {'B': 2, 'C': 3},

'B': {'D': 4},

'C': {'D': 2},

'D': {}

}

heuristic = {'A': 6, 'B': 4, 'C': 3, 'D': 0}

# Example run for Task #4

traffic\_updates = {('A', 'B'): 3, ('B', 'D'): -1}

path = a\_star\_with\_traffic(graph, 'A', 'D', heuristic, traffic\_updates)



print("Task #4 Path with traffic updates:", path)