National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

CS-272 Artificial Intelligence

BSDS-01A

Lab 9: Open Ended Lab

Date: 21-11-2024

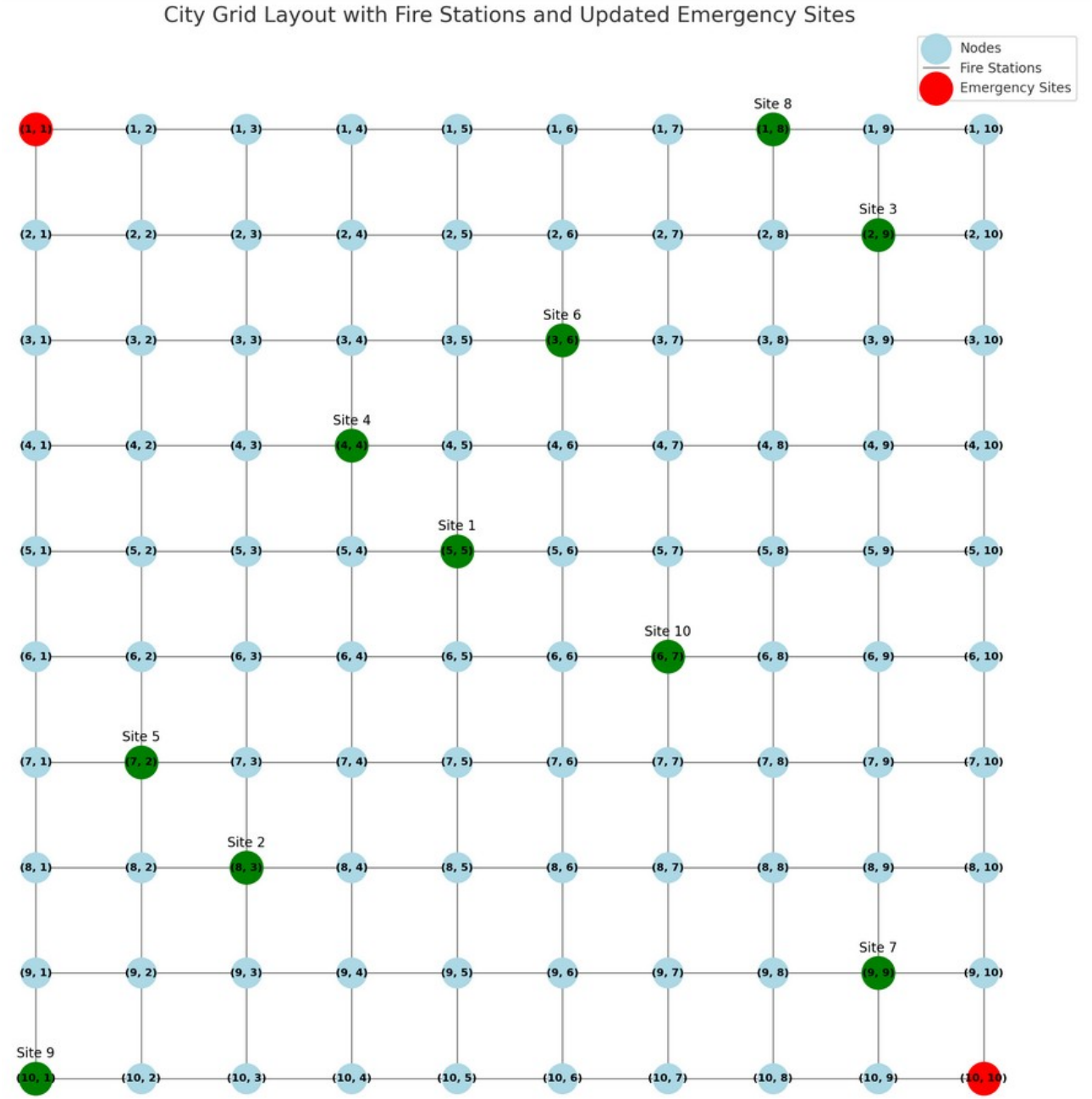
Lab Engineer: Mr Junaid Sajid

Instructor: Dr Seemab Latif

Name	Qalam
Abdul Mateen	457052

Problem:

We had road map in which we have nearly 100 sites which are linked together in the form of grid. There are several emergency sites and two fire stations. The road map is shown below:



Here you can see the linkage of sites with one another. But all the sites are not open at one time. At a particular time, there are sites that are blocked. At a specific time we are informed about an emergency site and we have to send help to that site. For that purpose we have to determine the path. Determining the path is our task here. We have two files data.xlsx and emergencySite.xlsx which contain the data about open paths in time interval and emergency site in time interval respectively.

Solution:

We will use concepts of dijkstra algorithm to solve this problem.

Approach to Solve the Problem:

Step 1 : Observing the data of excel sheets

First step that we are going to take is the observation of excel sheets. If you observe the emergency site sheet thats look totally fine. The problem we encounter is in data.xlsx. The problem is that we once a path from (a,b) to (x,y) is mentioned. Then we are not going to see (x,y) to (a,b). So, we have to built our algorithm while keeping this thing in mind.

Step : 2

Retrive the data and store them in the form of lists

Code Snippet:

```
import pandas as pd
import datetime
import heapq  # Using heapq for priority queue functionality

# Load the Excel files
file_path_1 = 'data.xlsx'
file_path_2 = 'emergencySiteData.xlsx'

data_1 = pd.read_excel(file_path_1)
data_2 = pd.read_excel(file_path_2)

# Select the required columns for roads
columns_1 = ['Road Segment Start', 'Road Segment End', 'Status', 'Current Speed (km/h)', 'Time']
selected_data_1 = data_1[columns_1]
data_list_1 = selected_data_1.values.tolist()

# Select the required columns for emergency sites
columns_2 = ['Emergency Site', 'Coordinates', 'Time']
selected_data_2 = data_2[columns_2]
data_list_2 = selected_data_2.values.tolist()
```

Step:3 Algorithm

In algorithm we need these things:

→ A function that give us the paths from specific site to other sites that are open at that time.

Code Snippet:

```
def get_neighbors(current_node, time, data_list):
    neighbors = []
    for road in data_list:
        start, end, status, speed, road_time = road
        if start == current_node:
            if isinstance(road_time, datetime.time):
                # Check if the road is open at the given time
                if road_time == time and status == 'Open':
                    neighbors.append((end, speed))
        elif end == current_node:
```

```
            if isinstance(road_time, datetime.time):
                # Check if the road is open at the given time
                if road_time == time and status == 'Open':
                    neighbors.append((start, speed))
    return neighbors
```

→ A function that checks the site is goal site or not. Calculate the time taken to reach the site. Keep record of the paths.

Code Snippet:

```
def dijkstra(start, target, time, data_list):
    # Min-heap for the priority queue: stores (cumulative_time, current_node)
    frontier = []
    heapq.heappush(frontier, (datetime.timedelta(0), start))  # Start with 0 time and the initial node
    explored = set()
    times = {start: datetime.timedelta(0)}  # Starting time is 0
    paths = {start: [start]}  # Path taken to reach each node

    while frontier:
        current_time, current_node = heapq.heappop(frontier)
        if current_node == target:
            return current_time, paths[current_node]

        neighbors = get_neighbors(current_node, time, data_list)
        for neighbor, speed in neighbors:
            if neighbor not in explored:
                travel_time = datetime.timedelta(hours=1) / speed  # Time taken to travel 1 km
                new_time = current_time + travel_time

                if neighbor not in times or new_time < times[neighbor]:
                    times[neighbor] = new_time
                    paths[neighbor] = paths[current_node] + [neighbor]
                    heapq.heappush(frontier, (new_time, neighbor))  # Push to the frontier
                explored.add(current_node)
    return None, None  # Return None if no path found
```

→ A function that determine path from both emergency sites and then compare them and give us the optimal path.

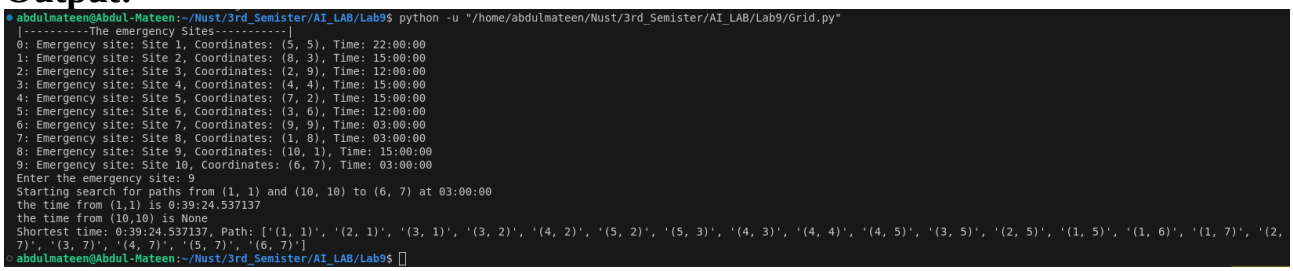
Code Snippet:

```
def findPath(data_list, emergency_site):
    start1 = '(1, 1)'
    start2 = '(10, 10)'
    target = emergency_site[1]
    time = emergency_site[2]

    print(f"Starting search for paths from {start1} and {start2} to {target} at {time}")
```

```
# Find the path from the first fire station (1, 1) to the emergency site
time_from_start1, path_from_start1 = dijkstra(start1, target, time, data_list)
# Find the path from the second fire station (10, 10) to the emergency site
time_from_start2, path_from_start2 = dijkstra(start2, target, time, data_list)
print("the time from (1,1) is", time_from_start1)
print("the time from (10,10) is", time_from_start2)
# Determine the shortest path and time
if time_from_start1 and time_from_start2:
    if time_from_start1 < time_from_start2:
        return time_from_start1, path_from_start1
    else:
        return time_from_start2, path_from_start2
elif time_from_start1:
    return time_from_start1, path_from_start1
elif time_from_start2:
    return time_from_start2, path_from_start2
else:
    return "No path available", None
```

Output:



Git Hub Link:
https://github.com/AbdulMateen12344567/Small-AI-model.git