Data Structures and Algorithms Design

Assignment 02 | PS4 | Pharmacy Run

Submitted By: Group 308

Group Contribution

Assignment Set Number:

Problem Statement 4

Group Name:

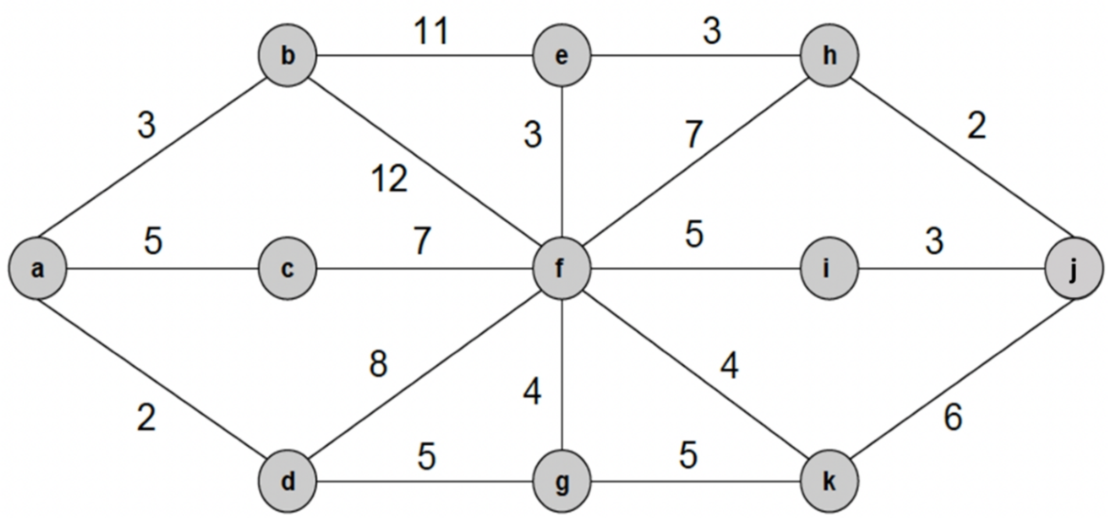
Group Number: 308

Contribution Table:

S.No.	Name (as appears in Canvas)	ID	Contribution (%)
1	Kumar Anurag	2021FC04016	100/3
2	Santoshi Nayak	2021FC04007	100/3
3	Maneesh Kumar Shrivastava	2021FC04011	100/3

Problem Statement

In light of the current pandemic, Harsh has been working from home and is taking extra precaution to make sure he stays safe. However, he has to make an emergency run to the pharmacy as his son hurt himself while playing. There are a couple of pharmacies near his house and he has to decide which one to go to. Every road in his neighborhood has a couple of containment zones. If there are two pharmacies located at specific junctions in his locality, you have to help Harsh identify which is the safer option such that he crosses the least number of containment zones to reach the pharmacy. A map of his locality and the containment zones on each road is given below.



Requirements

1. Formulate an efficient algorithm to help Harsh identify which pharmacy is safer.

Algorithm

We are making use of Dijkstra's Algorithm for finding out Pharmacy with minimum containment zones

STEP 01:

+ Mark all the Pharmacies as temporary
+ Assign all Pharmacies to have infinite containment zones except the Harsh's house (as Harsh's house is source)

STEP 02:

+ Calculate the total no of containment zones for all the adjacent Pharmacies
+ Choose any temporary Pharmacy (u) having the minimum no of containment zone(s)
+ Mark that Pharmacy (u) as permanent

STEP 03:

+ If no temporary Pharmacies left then STOP
+ Otherwise, Go to Step 02

STEP 04:

+ Finally find out the desired Pharmacy having the minimum containment zones.

Pseudocode

- G denotes given Graph
- V denotes each Vertex
- E denotes each Edge
- Q denotes a priority queue of all nodes in the graph
- S denotes a set to indicate which nodes have been visited by the algorithm

—------------------------

G = (V, E): the input graph
source: the source node
/* Initialization
dist[source] ⟵ 0

initialize dist[v] = 0, for all v $\in$ V and v ≠ source

add v to Q, for all v $\in$ V

S ⟵ Φ
/* the main loop for the algorithm
while Q ≠ Φ do

        v ⟵ vertex in Q with min dist[v]
        remove v from Q
        add v to S
        foreach neighbor u of v do

                if u $\in$ S then

                        continue;   // if u is visited then ignore it

                alt ⟵ dist[v] + weight(v, u)
                if alt < dist[u] then

                        dist[u] ⟵ alt;   // update distance of u
                        update Q

        return dist

2. Provide a description about the design strategy used.

Description

For finding out the Pharmacy having the minimum containment zones, we're making use of Dijkstra's Single Source - Shortest Path Algorithm.

First of all, we will consider all the Pharmacies as the node and all the containment zones as the distances. We are making Harsh's house as the source node.

Now our task is to find an immediate adjacent node (Pharmacy) having the minimum distance (minimum containment zones). We will continue this process until we reach the desired Pharmacies that the user has provided in the input file.

Finally, we will compare which Pharmacy took minimum containment zones while traveling.

3. Analyze the time complexity of the algorithm and show that it is an "efficient" one.

Analysis

Let the given **Graph (G) = (V, E)** is represented as an adjacency matrix. Here **cz[u, v]** denotes the no containment zones lying on the way from **V** to **E**, where **V** and **E** are vertices and edges respectively.

Also, the priority queue **Q** is represented as an unordered list.

Let **|E|** and **|V|** be the number of edges and vertices in the graph respectively. Then the time complexity is as follows:

(i) Adding all the vertices to **Q** takes **O(|V|)** times.

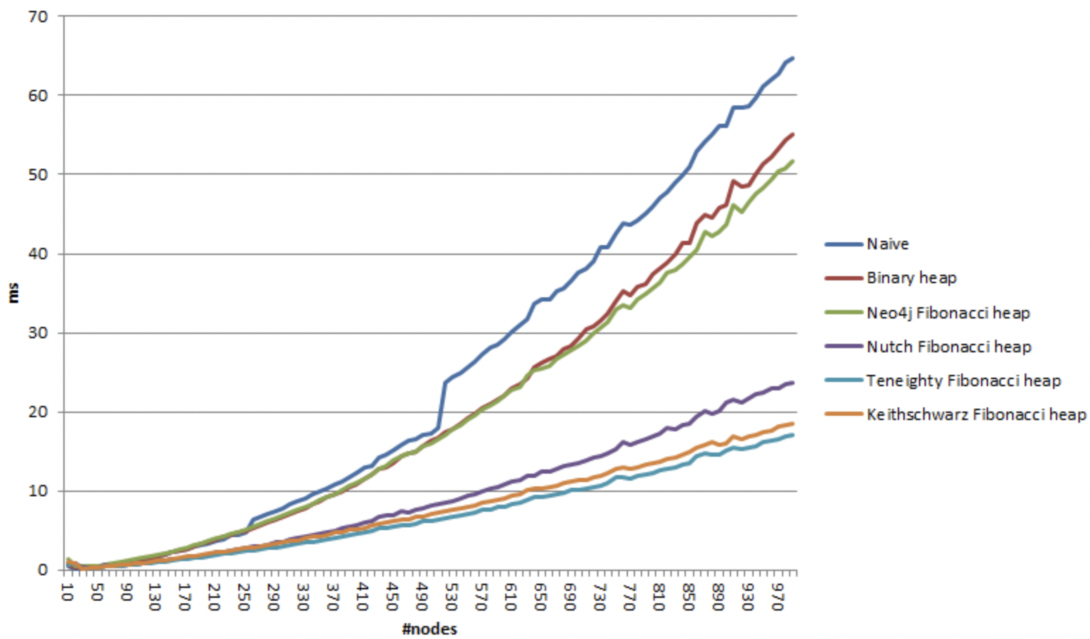
(ii) Removing the node with minimal dist takes **O(|V|)** time, and we only need **0(1)** to recalculate **dist[u]** and update **Q**. Since we use an adjacency matrix here, we'll need to loop for **|V|** vertices to update the dist array.

(iii) The time taken for each iteration of the loop is **O(|V|)**, as one vertex is deleted from **Q** per loop.

(iv) Thus, the total time complexity becomes:

$$O(|V|) + O(|V|) \times O(|V|) = O(|V|^2)$$

Graph Comparison



By comparing different approaches in graphical analysis, we found that our algorithm is the most efficient one.

4. Implement the above problem statement using Python 3.7

Source Code

```
import re

'''
find_pharmacy(): will find all the possible available
Pharmacies with the minimum containment zones
                with respect to the Harsh's House
(which is a source node)

                This function takes 3 parameters:
                    (i) nodes - means total no of
Pharmacies
                    (ii) edges - consists of no of
containment zones lying between 2 Pharmacies
                    (iii) source_index - Harsh's House
'''

def find_pharmacy(nodes, edges, source_index = 1):

   # marking all Pharmacies having infinite
containment zones initially
   path_lengths = {v: float('inf') for v in nodes}
   path_trace = {v: 'a' for v in nodes}

```

```
    # marking Harsh's house having 0 containment zone
as it's the source node
    path_lengths[source_index] = 0

    # storing all the Pharmacy and containment zone
details in 2D dictionary format
    adjacent_nodes = {v: {} for v in nodes}
    for (u,v), c_uv in edges.items():
        adjacent_nodes[u][v] = c_uv
        adjacent_nodes[v][u] = c_uv

    # calculating actual containment zones lying
between each pair of Pharmacies
    temporary_nodes = [v for v in nodes]
    while len(temporary_nodes) > 0:
        upper_bounds = {v: path_lengths[v] for v in
temporary_nodes}

        # finding a Pharmacy having the minimum
containment zones and then removing that node from
temporary nodes
        u = min(upper_bounds, key=upper_bounds.get)
        temporary_nodes.remove(u)

        # updating the containment zone no if we found
another path having minimum containment zone value
        for v, c_uv in adjacent_nodes[u].items():
```

```
            if path_lengths[v] > (path_lengths[u] +
c_uv):
                path_trace[v] = path_trace[u] + " " +
chr(v+96)
            path_lengths[v] = min(path_lengths[v],
path_lengths[u] + c_uv)

   # finally returning details of all Pharmacies along
with their minimum containment zones
   return path_lengths, path_trace


# reading the input file
input_file = open("inputPS4.txt","r")
src = input_file.read()

# Data Extraction from Input File
pattern01 = "([a-z])\s/\s([a-z])\s/\s(\d+)"
pattern02 = "Harsh\Ss House:\s([a-z])"
pattern03 = "Pharmacy\s(\d+):\s([a-z])"
details01 = re.findall(pattern01, src)
details02 = re.findall(pattern02, src)
details03 = re.findall(pattern03, src)
input_file.close()

# storing the edges and nodes details from the input
file
```

```
edges = {}
nodes = []
for x,y,z in details01:
   edges[(ord(x)-96,ord(y)-96)] = float(z)
   if ord(x)-96 not in nodes:
       nodes.append(ord(x)-96)
   if ord(y)-96 not in nodes:
       nodes.append(ord(y)-96)

# storing harsh house, pharmacy 1 and pharmacy 2
details
harsh_house = ord(details02[0])-96
pharmacy_data = {k: v for k,v in details03}

# executing the algorithm
containment_zone, trace = find_pharmacy(nodes, edges,
harsh_house)
final_containment_zone = {}
final_trace = {}
for k,v in containment_zone.items():
   final_containment_zone[chr(k+96)] = v
for k,v in trace.items():
   final_trace[chr(k+96)] = v

# generating the output file
output_file = open("outputPS4.txt","w")
```

```
safer_pharmacy = pharmacy_data[min(pharmacy_data,
key=pharmacy_data.get)]
safer_pharmacy_no = "Pharmacy " +
str(min(pharmacy_data, key=pharmacy_data.get))

path_to_follow = final_trace[safer_pharmacy]
total_containment_zones =
str(int(final_containment_zone[safer_pharmacy]))

final_result = "Safer Pharmacy is: " +
safer_pharmacy_no + "\n" + "Path to follow: " +
path_to_follow + "\n" + "Containment zones on this
path: " + total_containment_zones

output_file.write(final_result)
output_file.close()
```

-x- Thank You !! -x-