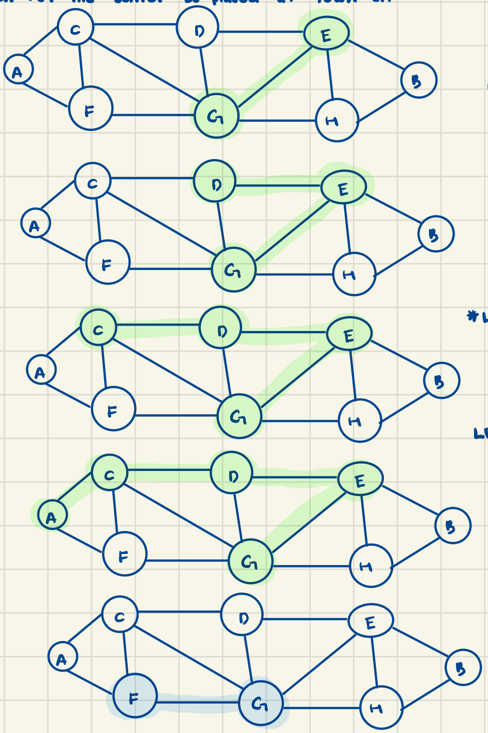
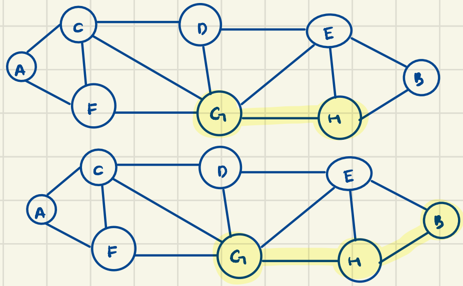
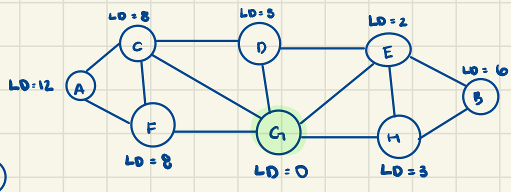
CS 325 : Homework 4

#1 (a) Let the center be placed at town G:





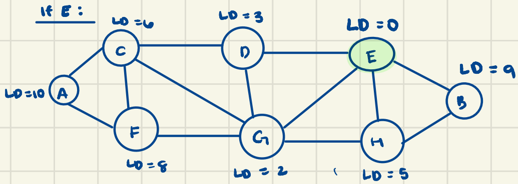
*Lowest distance → LD



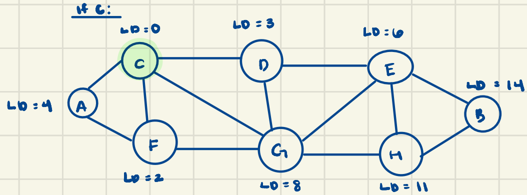
Vertex	distance	parent of vertex	Shortest Path
→ G	0	start	G
A	12	C	G → E → D → C → A
B	6	H	G → H → B
C	8	D	G → E → D → C
D	5	E	G → E → D
E	2	G	G → E
F	8	G	G → F
H	3	G	G → H

(B)

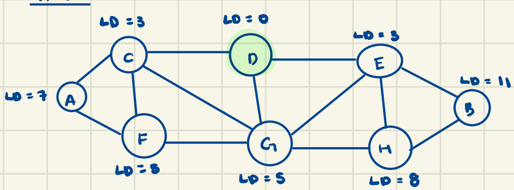
If E:



If C:



If D:

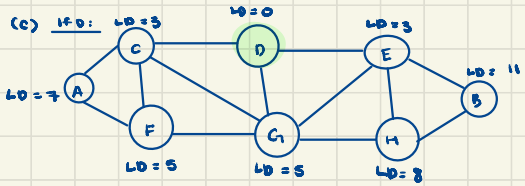


Algorithm:

```
for i in range numLocations :      # loop through diff. locations
    total distance = 0
    distances [ ]
    roads = length (distances)
    for j in range roads - 1 :    # finding sum of roads
        total distance += distance [j]
    if sum > total sum :
        sum = total sum
        good location = i
return goodLocation
```

The run time would be $O(f_r + f^2(\log f))$ →

(c) If D:



Vertex	Distance	Parent vertex	Shortest path from D
D	0	start	D
A	7	C	D → C → A
B	11	H	D → E → G1 → H → B
C	3	D	D → C
E	3	D	D → E
F	5	C	D → C → F
G1	5	E	D → E → G1
H	8	G1	D → E → G1 → H

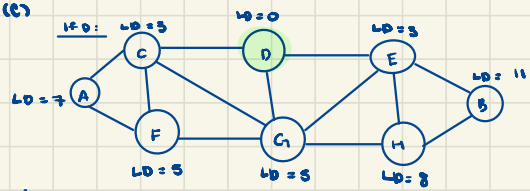
(a) I would place one on D because it is the best place overall and the second place would G1 because it has the best for H, B, E. G1 can handle the right side and D can handle the left. alternatively E would also be nice because it has short paths overall.

An algorithm could be having a system where certain facilities do all the shipping out based on distance. For example G1 should ship to E because it's 2 vs D to E which is 3. Having a structure to store this data so it can be checked which facility would be shipping could help the system go smoother.

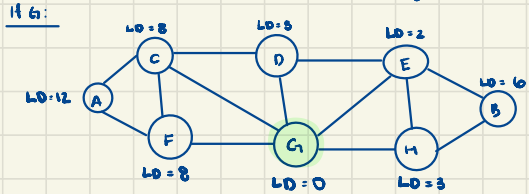
an algorithm to search could be like B except is searching for two. Instead of just searching from one location search from both then record and store which of the two places is better for all the locations

(c)

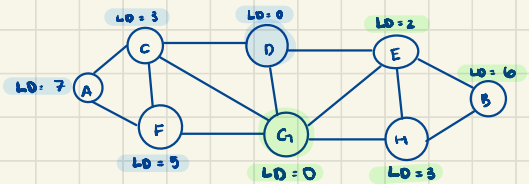
If D:



If G1:



So D and G1:



problem 2:

```
main():
    open("graph.txt")
    num_test = file.readline()
    for the # of tests
        numVert = file.readline()
        for the # of vertices:
            Fill node class
        Weight = mst Weight (vertices)
        Print results
```

```
mst Weight( array):
    for i in range of array length -1:
        if i = 0:
            root = vert array [0]
            pop. vert array
        else:
            root = temp
            pop. temp
        for in of array len:
            find distance with A² + b² = c²
            if distance <= temp dis
                temp dis = distance  # updating closer choice
            total distance += temp dis
    return total distance
```

Following dijkstra's algo so ...

The run time would be $O(f_r + f^2(\log f))$ →