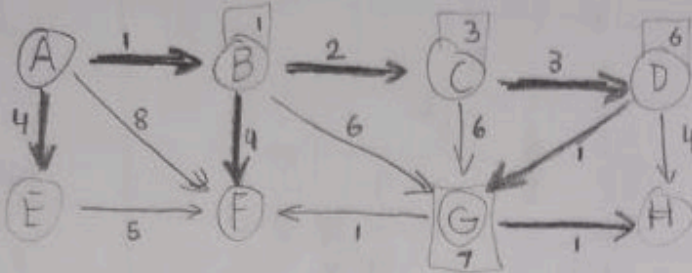


Assignment 14

1) Dijkstra's Algorithm



A ∞ 0
 B ∞ 1
 C ∞ 3
 D ∞ 6
 E ∞ 4
 F ∞ 5
 G ∞ 7
 H ∞ 8

Visited
 [A, B, C, D, E, F, G, H]

PQ:

A-B	1
A-E	4
A-F	8

 →

B-C	3
A-E	4
B-F	5
B-G	7
A-F	8

 →

A-E	4
B-F	5
C-D	6
B-G	7
A-F	8
C-G	9

 →

B-F	5
C-D	6
B-G	7
A-F	8
E-F	9
E-G	9

both nodes
 have now
 been visited
 so delete

→

C-D	6
B-G	7
C-G	9
E-F	9
E-G	9

 →

D-G	7
B-G	7
C-G	9
E-F	7
C-F	9

 →

G-H	8
-----	---

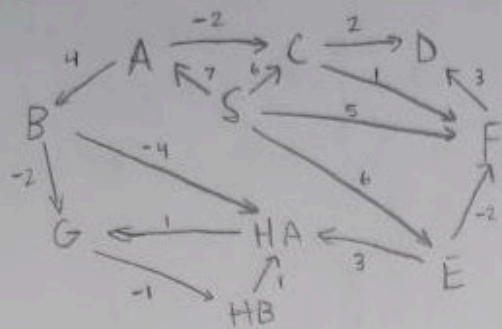
all nodes have been visited.

Final:

A → B → C → D
 ↓ ↓ ↙
 E F G → H

cost = 16

2) Bellman Ford

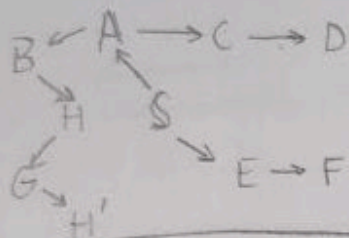


	S	A	C	E	B	H	G
A	7						
B		11					
C		5					
D			1				
E	6						
F	5			4			
G				9	8		
HA				9	7		
HB						7	

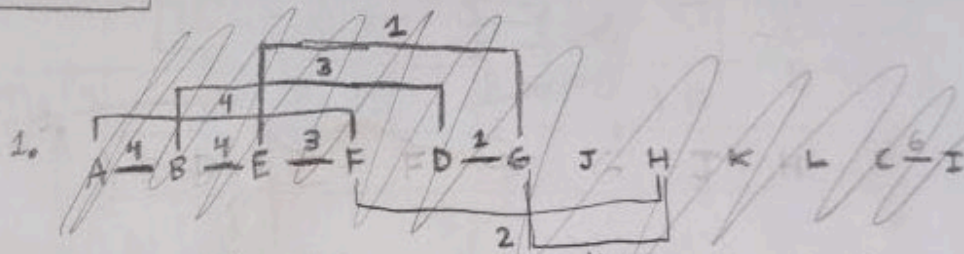
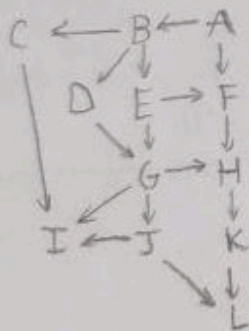
(cost, EdgeTo)

(7, S)
(15, A)
(5, A)
(7, C)
(6, S)
(4, E)
(8, H)
(7, B)
(7, 6)

Final Tree (Cost: 11)



3) DAG



can solve using a queue based approach:

Topological sort:

DFS (A → B → D → G → I) — I

DFS (J, L) — L, J

DFS (H, K) — K, H, G, D

DFS (E, F) — F, E

DFS (C) — C, B, A

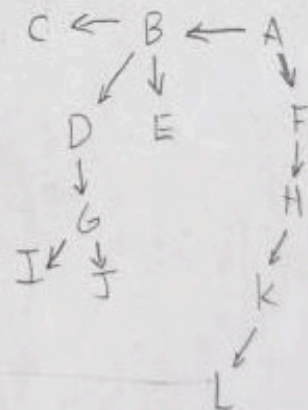
resulting stack: A B C D E F D G H K J L I

relax →

relaxing:	A	B	C	D	E	F	D	G	H	K	Final
A	0, A										0, A
B	4, A										4, A
C		9, B									9, B
D		7, B									7, B
E		8, B									8, B
F	4, A										9, A
G					9, E		8, D				8, D
H						6, F					6, F
I					15, C			10, G			10, G
J								11, G			11, G
K									9, H		9, H
L										10, K	10, K

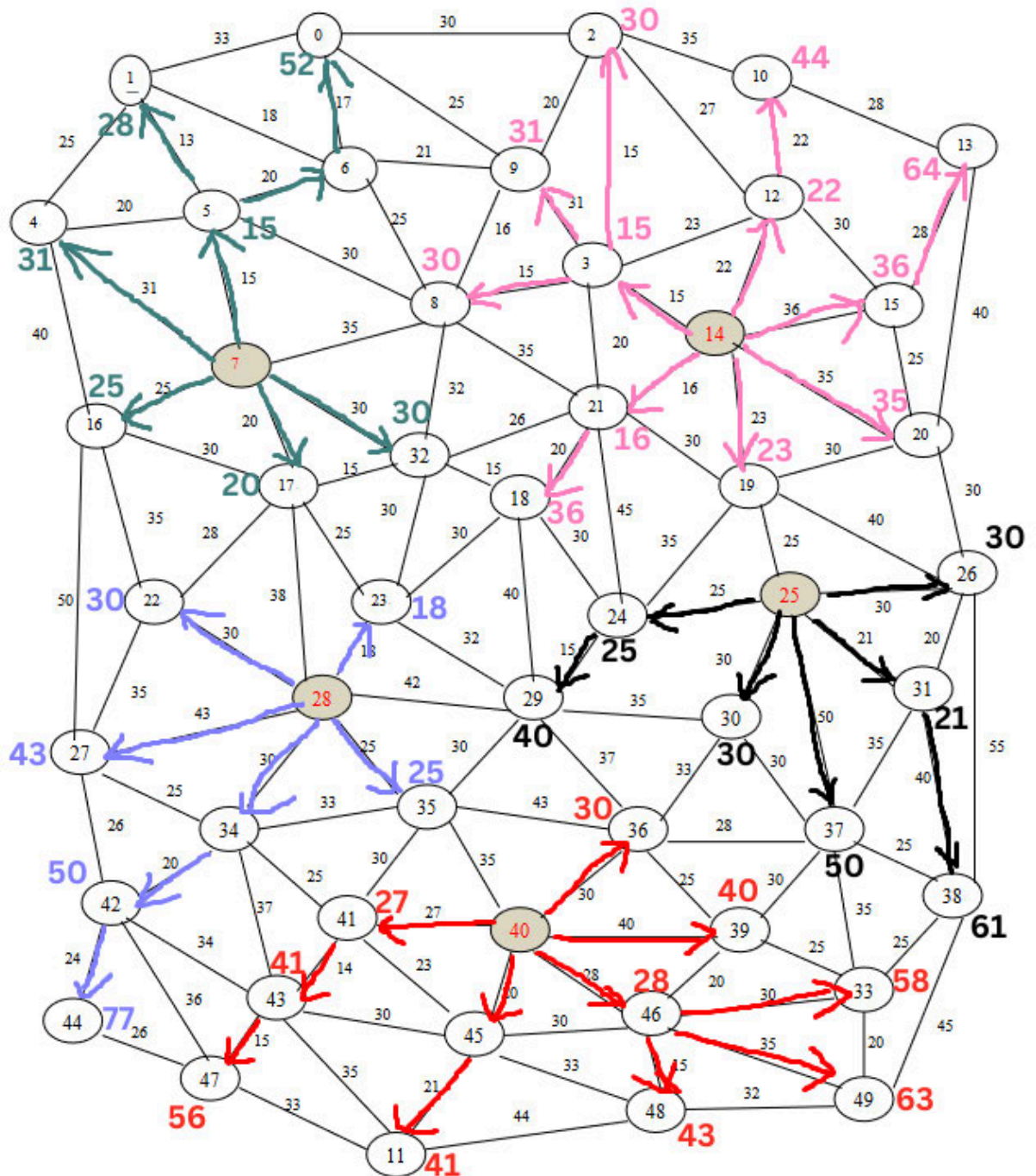
results in shortest path of:

$$\text{cost} = 32$$



FOR PROGRAMMING ASSIGNMENT VVVV

Customer can efficiently serve through the following stations and paths:



If I had to chose a node to be a station, I would likely pick 18 as it is central and has quite cheap options.

Just incase I missed any code, here is my output:

The screenshot shows an IDE with a Java file named `GraphTester.java`. The code implements a shortest path algorithm (Dijkstra's) for a graph with 50 nodes. It includes a `printShortestPaths` method that iterates through all nodes and prints the shortest path and total distance for each. The output on the right shows the results for nodes 41 through 49.

```
public class GraphTester {
    // ----- //
    // printShortestPaths
    // (where stuff goes down!!)
    // ----- //

    // Method to print the shortest paths to all nodes
    private static void printShortestPaths(DijkstraSP dijkstra)
    {
        // Loop through all nodes
        for (int i = 0; i < 50; i++)
        {
            // Get the shortest path to node i
            Iterable<DirectedEdge> path = dijkstra.pathTo(i);
            // If a path exists
            if (path != null)
            {
                // Calculate the total distance of the path
                double totalDistance = calculateTotalDistance(path);
                // If the total distance is not 0
                if (totalDistance != 0.0)
                {
                    // Print the header for the node
                    System.out.println("%s-----");
                    System.out.println("Shortest path to node " + i + ":");
                    System.out.println("Total Distance: " + totalDistance);
                    System.out.println("Path:");
                    // Print each edge in the path
                    for (DirectedEdge edge : path)
                    {
                        if (edge.weight() != 0.0)
                        {
                            System.out.println("From Node " + edge.from() + " to Node " + edge.to() + ", Distance: " + edge.weight());
                        }
                    }
                    // Print the footer for the node
                    System.out.println("%s=====");
                }
            }
            else
            {
                // Print a message if no path is found
                System.out.println("No path found to node " + i);
            }
        }
    }
}
```

Shortest path to node 41:
Total Distance: 27.0
Path:
From Node 40 to Node 41, Distance: 27.0
=====

Shortest path to node 42:
Total Distance: 33.0
Path:
From Node 28 to Node 34, Distance: 33.0
From Node 34 to Node 42, Distance: 20.0
=====

Shortest path to node 43:
Total Distance: 41.0
Path:
From Node 40 to Node 41, Distance: 27.0
From Node 41 to Node 43, Distance: 14.0
=====

Shortest path to node 44:
Total Distance: 77.0
Path:
From Node 28 to Node 34, Distance: 33.0
From Node 34 to Node 42, Distance: 20.0
From Node 42 to Node 44, Distance: 24.0
=====

Shortest path to node 45:
Total Distance: 20.0
Path:
From Node 40 to Node 45, Distance: 20.0
=====

Shortest path to node 46:
Total Distance: 28.0
Path:
From Node 40 to Node 46, Distance: 28.0
=====

Shortest path to node 47:
Total Distance: 56.0
Path:
From Node 40 to Node 41, Distance: 27.0
From Node 41 to Node 43, Distance: 14.0
From Node 43 to Node 47, Distance: 15.0
=====

Shortest path to node 48:
Total Distance: 43.0
Path:
From Node 40 to Node 46, Distance: 28.0
From Node 46 to Node 48, Distance: 15.0
=====

Shortest path to node 49:
Total Distance: 63.0
Path:
From Node 40 to Node 46, Distance: 28.0
From Node 46 to Node 49, Distance: 35.0
=====

I hope this algorithm is efficient and suits your needs