Name: Naga Venkata V Vinnakota
Net ID: nvv13
Course: ECE 16:332:573 DSA

Assignment-4

1. Write a program that answers the following for an undirected graph: Is a graph acyclic? Run your program on graph (linked after Q2).

Solution:

Depth-First search can help us check the connectivity between points or vertices in a graph. A loop exists if a vertex is visited more than once. Any such loop will create a cycle in the graph.

I have used the above fact to check if the graph is cyclic. I have used a Boolean variable, is_cyclic whose value is initially false and whenever a vertex is visited more than once, the is_cyclic value becomes true indicating there is a loop. This value will be true if there exists a cycle or more precisely at least one loop.

Results:

The results I got from the graph provided in the link is as below:

Is the graph acyclic?

No, the graph is cyclic

2. Implement and execute Prim's and Kruskal's algorithms on the graph linked below (the third field is the weight of an edge). Which performs better? Explain your answer.

Use Graph for Q1 and 2: https://content.sakai.rutgers.edu/access/content/group/3c4e126c-962f-4655-97c6-d7299fa5ef53/Homework%20DataSet/mediumEWG.txt

Solution:

Prim's Algorithm tries to connect vertices to a single spanning tree. The algorithm tries to compute the nearest vertex to the spanning tree from the available vertices.

Kruskal's Algorithm checks if two vertices having a minimum distance are connected. If they are not connected the algorithms connects them and make them a set. If connected, checks for other vertex pairs with minimum distance.

Moreover, the time complexity of Kruskal's Algorithm is of order O(Elog(V)). The time complexity of Prims' Algorithm is nearly of order O((V+E)log(V)), where E is the number of vertices and v is the number of vertices in the graph in both the cases.

This clearly gives us an idea that Kruskal's Algorithm takes lesser time compared to Prim's Algorithm. The same can be observed from the results of executing Kruskal's and Prim's algorithm on the same graph as shown below. So Kruskal's algorithm shows better performance.

Results:

Runtime for Kruskal's Algorithm is 10974600 ns

Runtime for Prim's Algorithm is 18910200 ns

3. For the edge-weighted directed acyclic graph given below, compute (i.e., manually trace) both the longest path and the shortest path.

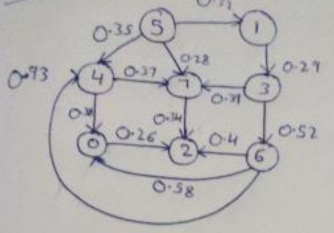
```
8
13
5 4 0.35
4 7 0.37
5 7 0.28
5 1 0.32
4 0 0.38
0 2 0.26
3 7 0.39
1 3 0.29
7 2 0.34
6 2 0.40
3 6 0.52
6 0 0.58
6 4 0.93
```

Solution:

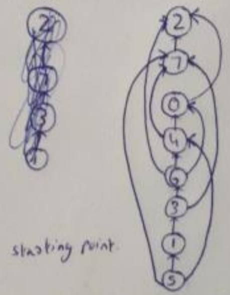
I have used Topological sort for selecting the source. I have selected '5' as the source since the Topological sorted DAG shows us that '5' can reach all the vertices.

Another reason for using topological sort is that we can easily find the shortest and longest path from source just by examining the graph rearranged in topological order. It will give us the information like which edge to be taken to reach a vertex, how many paths are available from a source to a destination etc.

Q(3)



By using Topological sort we get



By making 'S' as starting point.

The shortest path to each other vertex is

0	1	2	3	4	5	6	7
0·73	0·32	0·62	~~0·47~~ 0·61	0·35	0	1·13	0·28

The longest path to other vertices form S is

0	1	2	3	4	5	6	7
2·44	0·32	2·7	0·61	2·06	0	1·13	2·43

Results:

Shortest Path

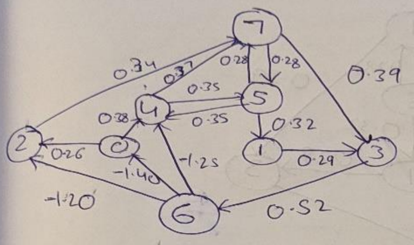
0	1	2	3	4	5	6	7
0.73	0.32	0.62	0.61	0.35	0	1.13	0.28

Longest Path:

0	1	2	3	4	5	6	7
2.44	0.32	2.7	0.61	2.06	0	1.13	2.43

4. (a) For the digraph with negative weights, compute (i.e. manually trace) the progress of the Bellman-Ford Algorithm.

8
15
4 5 0.35
5 4 0.35
4 7 0.37
5 7 0.28
7 5 0.28
5 1 0.32
0 4 0.38
0 2 0.26
7 3 0.39
1 3 0.29
2 7 0.34
6 2 -1.20
3 6 0.52
6 0 -1.40
6 4 -1.25



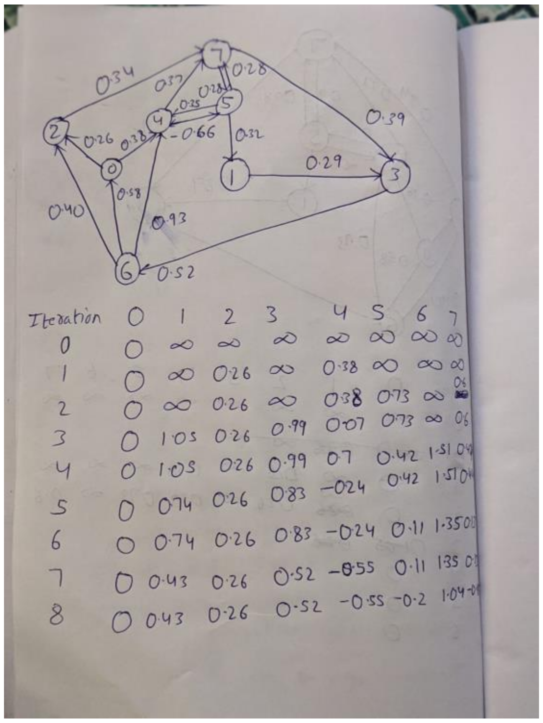
Bellman-Ford Algorithm

0	1	2	3	4	5	6	7	
0	∞	∞	∞	∞	∞	∞	∞	Initially
0	∞	0·26	∞	0·38	∞	∞	∞	1
0	∞	0·26	∞	0·38	0·73	∞	0·6	2
0	1·05	0·26	0·99	0·38	0·73	∞	0·6	3
0	1·05	0·26	0·99	0·38	0·73	1·51	0·6	4
0	0·93	0·26	0·99	0·26	0·61	1·51	0·6	→ final result.

In this case we get final result at 5th iteration due to the fact, the edge 6→4 does not create a cycle and ~~is less~~ the path is less than the initial weight.

4. (b) For the digraph with a negative cycle, compute (i.e. manually trace) the progress of the Bellman-Ford Algorithm.

8
15
4 5 0.35
5 4 -0.66
4 7 0.37
5 7 0.28
7 5 0.28
5 1 0.32
0 4 0.38
0 2 0.26
7 3 0.39
1 3 0.29
2 7 0.34
6 2 0.40
3 6 0.52
6 0 0.58
6 4 0.93



Iteration	0	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	∞	0.26	∞	0.38	∞	∞	∞
2	0	∞	0.26	∞	0.38	0.73	∞	0.6
3	0	1.05	0.26	0.99	0.07	0.73	∞	0.6
4	0	1.05	0.26	0.99	0.7	0.42	1.51	0.4
5	0	0.74	0.26	0.83	-0.24	0.42	1.51	0.
6	0	0.74	0.26	0.83	-0.24	0.11	1.35	0.
7	0	0.43	0.26	0.52	-0.55	0.11	1.35	0.
8	0	0.43	0.26	0.52	-0.55	-0.2	1.04	-0

The result at 8th iteration is

0	1	2	3	4	5	6	7
0.0	0.43	0.26	0.52	-0.55	-0.2	1.04	-0.18

Observation:

For the question, 4-a, there are no negative cycles present in the graph. So, the edge relaxation stops at a point where there is no further possibility of reducing the path length using Bellman-Ford Algorithm. After 5 iterations, the graph stabilizes by the obtained path weights and further relaxation is not possible.

For the question 4-b, however, since there is a negative cycle present in the graph, the edge relaxation does not stop, and we cannot achieve the proper solution of the shortest path.

This gives us a clear idea on how negative cycles impact the Bellman-Ford algorithm.

5. Implement a DFS and BFS traversal for the data-set of the undirected road network of New York City. The graph contains 264346 vertices and 733846 edges. It is connected, contains parallel edges, but no self-loops. The edge weights are travel times and are strictly positive.

Solution:

I have implemented an iterative DFS and BFS, discussed in class for the above provided data set. I choose the iterative approach since the graph contains 264346 vertices and 733846 edges. Using a Recursive approach might cause in exceeding recursion stack limit.

I have implemented a stack to store the vertices in the DFS and a queue to store the vertices in BFS using list.

In both the cases, the result of pop function will be visited.

6. Implement the shortest path using Djikstra's Algorithm for the graph in HW5 Q 4(b). Then run your implementation of Djikstra's on HW5 4(a). What happens? Explain.

Solution:

I have implemented Dijkstra's on 4(b) and 4(b) and the results are as below:

```
Shortest path from source S=0 for graph 4-b
Vertex  Distance from Source:
0           0
1           1.05
2           0.26
3           0.9900000000000001
4           0.38
5           0.73
6           1.5100000000000002
7           0.6000000000000001
```

```
Shortest path from source S=0 for 4-a
Vertex  Distance from Source:
0               0
1               1.05
2               0.26
3               0.9900000000000001
4               0.38
5               0.73
6               1.5100000000000002
7               0.6000000000000001
```

We can observe that the Dijkstra’s Algorithm did not give the results same as the Bellman-Ford Approach we used in 4(a) and 4(b).

This is due to the reason, that Dijkstra's algorithm does not work efficiently for negative edges. Dijkstra’s Algorithm work on the fact that given a graph of non-negative edges, adding an extra edge to the path will not create a shorter path.