**Assignment 7**

**Problem 1 (4+4 marks). For this problem, use tables for illustration, as shown on the handout provided in class. Additionally, list the final obtained shortest (weighted or unweighted) path length for every vertex, after termination of the algorithm. NOTE: It is enough if you list all the tables only for one of the two problems (you may choose which one). For the other one, you can opt to show only the final results.**

1. **For the graph shown in Figure 1, find the shortest weighted paths from A to the other vertices in the graph (one path for each vertex).**

Ans:

1. First step will be to an array of distances from source to all nodes like so [0, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
2. Now we will set up a loop which goes through every node in the graph.
   1. In this loop we will check if graph[ A ][ n ] > 0 and distance[ n ] > distance[ A ] + graph[ A ][ n ]
   2. If it is true we set the distance[n] to distance[ A ] + graph[ A ][ n ]
3. After the first iteration of this loop we will get the distance array like so [0, 5, 3, Infinity, Infinity, Infinity, Infinity]
4. Now we change the from A to C as the source and reevaluate the distance from all unknown nodes.
5. After 7 iterations we get a fully known distance array as [0, 5, 3, 9, 7, 8, 6]
   1. A = 0, B = 5, C = 3, D = 9, E = 7, F = 8, G = 6

Steps followed:

Initial distance array:

[0, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]

if [ 0 > 0 and 0 > 0 + 0 ]-> False

if [ 5 > 0 and Infinity > 0 + 5 ]-> True

distance[ B ] = 0 + 5 = 5

if [ 3 > 0 and Infinity > 0 + 3 ]-> True

distance[ C ] = 0 + 3 = 3

if [ 0 > 0 and Infinity > 0 + 0 ]-> False

if [ 0 > 0 and Infinity > 0 + 0 ]-> False

if [ 0 > 0 and Infinity > 0 + 0 ]-> False

if [ 0 > 0 and Infinity > 0 + 0 ]-> False

Distance array after iteration: 7

[0, 5, 3, Infinity, Infinity, Infinity, Infinity]

if [ 0 > 0 and 0 > 3 + 0 ]-> False

if [ 0 > 0 and 5 > 3 + 0 ]-> False

if [ 0 > 0 and 3 > 3 + 0 ]-> False

if [ 7 > 0 and Infinity > 3 + 7 ]-> True

distance[ D ] = 3 + 7 = 10

if [ 7 > 0 and Infinity > 3 + 7 ]-> True

distance[ E ] = 3 + 7 = 10

if [ 0 > 0 and Infinity > 3 + 0 ]-> False

if [ 0 > 0 and Infinity > 3 + 0 ]-> False

Distance array after iteration: 7

[0, 5, 3, 10, 10, Infinity, Infinity]

if [ 0 > 0 and 0 > 5 + 0 ]-> False

if [ 0 > 0 and 5 > 5 + 0 ]-> False

if [ 2 > 0 and 3 > 5 + 2 ]-> False

if [ 0 > 0 and 10 > 5 + 0 ]-> False

if [ 3 > 0 and 10 > 5 + 3 ]-> True

distance[ E ] = 5 + 3 = 8

if [ 0 > 0 and Infinity > 5 + 0 ]-> False

if [ 1 > 0 and Infinity > 5 + 1 ]-> True

distance[ G ] = 5 + 1 = 6

Distance array after iteration: 7

[0, 5, 3, 10, 8, Infinity, 6]

if [ 0 > 0 and 0 > 6 + 0 ]-> False

if [ 0 > 0 and 5 > 6 + 0 ]-> False

if [ 0 > 0 and 3 > 6 + 0 ]-> False

if [ 0 > 0 and 10 > 6 + 0 ]-> False

if [ 1 > 0 and 8 > 6 + 1 ]-> True

distance[ E ] = 6 + 1 = 7

if [ 0 > 0 and Infinity > 6 + 0 ]-> False

if [ 0 > 0 and 6 > 6 + 0 ]-> False

Distance array after iteration: 7

[0, 5, 3, 10, 7, Infinity, 6]

if [ 0 > 0 and 0 > 7 + 0 ]-> False

if [ 0 > 0 and 5 > 7 + 0 ]-> False

if [ 0 > 0 and 3 > 7 + 0 ]-> False

if [ 2 > 0 and 10 > 7 + 2 ]-> True

distance[ D ] = 7 + 2 = 9

if [ 0 > 0 and 7 > 7 + 0 ]-> False

if [ 1 > 0 and Infinity > 7 + 1 ]-> True

distance[ F ] = 7 + 1 = 8

if [ 0 > 0 and 6 > 7 + 0 ]-> False

Distance array after iteration: 7

[0, 5, 3, 9, 7, 8, 6]

if [ 0 > 0 and 0 > 8 + 0 ]-> False

if [ 0 > 0 and 5 > 8 + 0 ]-> False

if [ 0 > 0 and 3 > 8 + 0 ]-> False

if [ 0 > 0 and 9 > 8 + 0 ]-> False

if [ 0 > 0 and 7 > 8 + 0 ]-> False

if [ 0 > 0 and 8 > 8 + 0 ]-> False

if [ 0 > 0 and 6 > 8 + 0 ]-> False

Distance array after iteration: 7

[0, 5, 3, 9, 7, 8, 6]

if [ 0 > 0 and 0 > 9 + 0 ]-> False

if [ 0 > 0 and 5 > 9 + 0 ]-> False

if [ 0 > 0 and 3 > 9 + 0 ]-> False

if [ 0 > 0 and 9 > 9 + 0 ]-> False

if [ 0 > 0 and 7 > 9 + 0 ]-> False

if [ 6 > 0 and 8 > 9 + 6 ]-> False

if [ 0 > 0 and 6 > 9 + 0 ]-> False

Distance array after iteration: 7

[0, 5, 3, 9, 7, 8, 6]

1. **For the graph shown in Figure 1, find the shortest unweighted paths from B to the other vertices in the graph (one path for each vertex).**

Ans: Following same steps we can get the result as

[0, 1, 3, 5, 4, 3, 6, 6, 12, 6, 9]

i.e s = 0, A = 1, B = 3, C = 5, D = 4, E = 3, F = 6, G = 6, H = 12, I = 6, t = 9

**Problem 2 (4 marks). Give an example of a directed weighted graph G that satisfies the following three properties simultaneously. (1) G has no cycle with negative cost; (2) G has an edge with negative cost; (3) Dijkstra's algorithm, applied to G, gives an incorrect answer.**

Ans: 

1. The cycle does not exist because of negative value.
2. The edge D to B has a negative cost of -3
3. When Dijkstra’s algorithm is applied to this graph, it will not give us the shortest distance because considering A as Source and C as Sink, its shortest distance will be 0 at the end result. This will be directly from A to C. But the shortest distance is through D as it will cost -1 which is less than 0. This will not be found by Dijkstra’s algorithm.

**Problem 3 (6 marks). Provide a modification of Dijkstra's algorithm for the case that the range of the weight function c equals {1, 2, 3, …, k} for some k ∈ N. Your algorithm must have a worst-case running time of O(k . |V| + |E|). Explain the correctness of your modified algorithm and its running time.**

Ans: Bucket Implementation can be used to make the modifications in the algorithm. As this method will take the worst case running time of O(k . |V| + |E|) for some k ∈ N. The vertex V updates |V|-1 times and the bound is O(k . |V|) on the path which has the shortest distance. We can use the following implementation;

Input: Graph G(V, E) where weight of edges is Wg, Max weight is W and some vertex which belongs to V.

Implementation Algorithm: First step will be to consider that the distance from source to all the nodes is infinite and the previous pointer has null value. Also initializing the bucket variable to null value. Now create an array A of size k|V| with counter i. This is to store the vertex with distance [i] in A[i].

Let the source value be a = A[0] and the bucket value from the source be bucket(s) = 0. Now start the while loop with the constraints as it will end the loop when the value of counter exceeds the value of k|V|. Inside the loop, edges(a V) belong to E store the distance to v, dist(v), in a temporary variable. Check if the new distance is less than the current distance, if it is update the current distance with new and set the previous pointer, prev(V) to a. Increment the counter.

**Problem 4 (4+3 marks).**

1. **Provide an algorithm that solves the following problem.**

**given: an undirected graph G = (V,E) that contains no cycles; a vertex s ∈ V .**

**task: for each vertex v ∈ V , determine the length of the longest path from s to v in G.**

**Your algorithm must have a worst-case running time of O(|V| + |E|). Explain the correctness of your algorithm and its running time.**

Ans: Longest path can be achieved using following steps:

1. Initializing distance of all vertices to 0 from source s.
2. Create an array of all the vertices in a topological fashion
3. Start a loop for every vertex x in the array. Check if the distance from source to y (vertex adjacent to x) is less than the distance from source to x adding weight of this edge to it. If it is, then update the distance of y to the greater value.
4. Resulting distance array will have all the longest path.

The time complexity will be the addition of topological sorting plus the loop which executes for every vertex in the array. Which gives us O(V + E) + O(E) (almost equal to O(V + E)).

1. **Why does the problem in (a) become more dicult if the input graph is no longer known to be acyclic and we ask for the lengths of the longest simple paths? What is the running time cost of the most efficient algorithm you can think of?**

Ans: If we have a cyclic graph then every time a vertex is visited it will increase the longest distance of that vertex. This will result in a never ending loop. We can achieve a time complexity if O(n \* (2 pow n)) if we use Dynamic programming.