CS263: Design and Analysis of Algorithm

1 Problem

Given a weighted directed graph with few negative edges, find the shortest path from the source to all other vertices in the given graph. Write the code and complexity of your algorithm.

Algorithm:

1. At first we will initialize distances from the source to all vertices as infinite and distance to the source itself as 0. Now we will create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

2. The following step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.

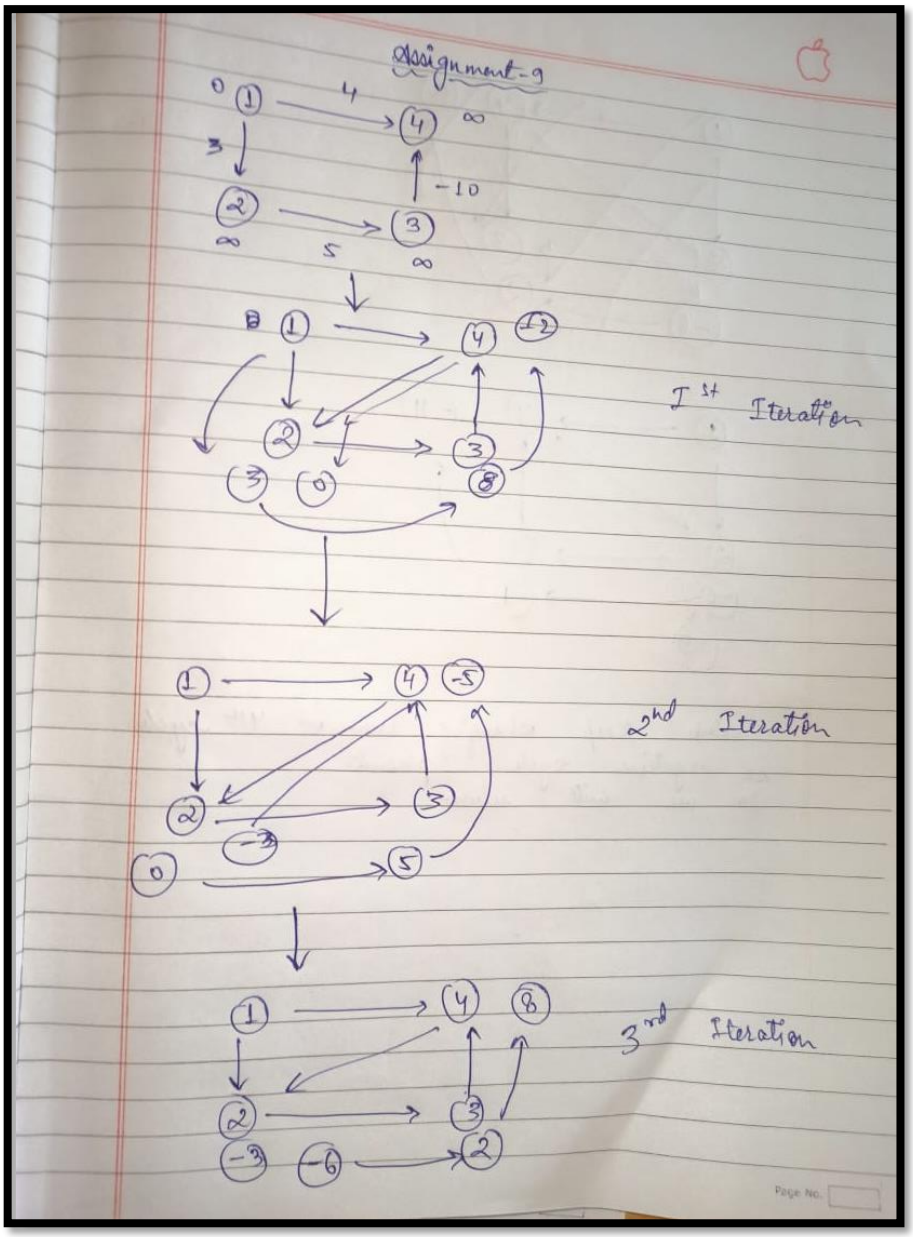
    - Do following for each edge u-v
      If dist[v] > dist[u] + weight of edge uv, then update dist[v]
      dist[v] = dist[u] + weight of edge uv

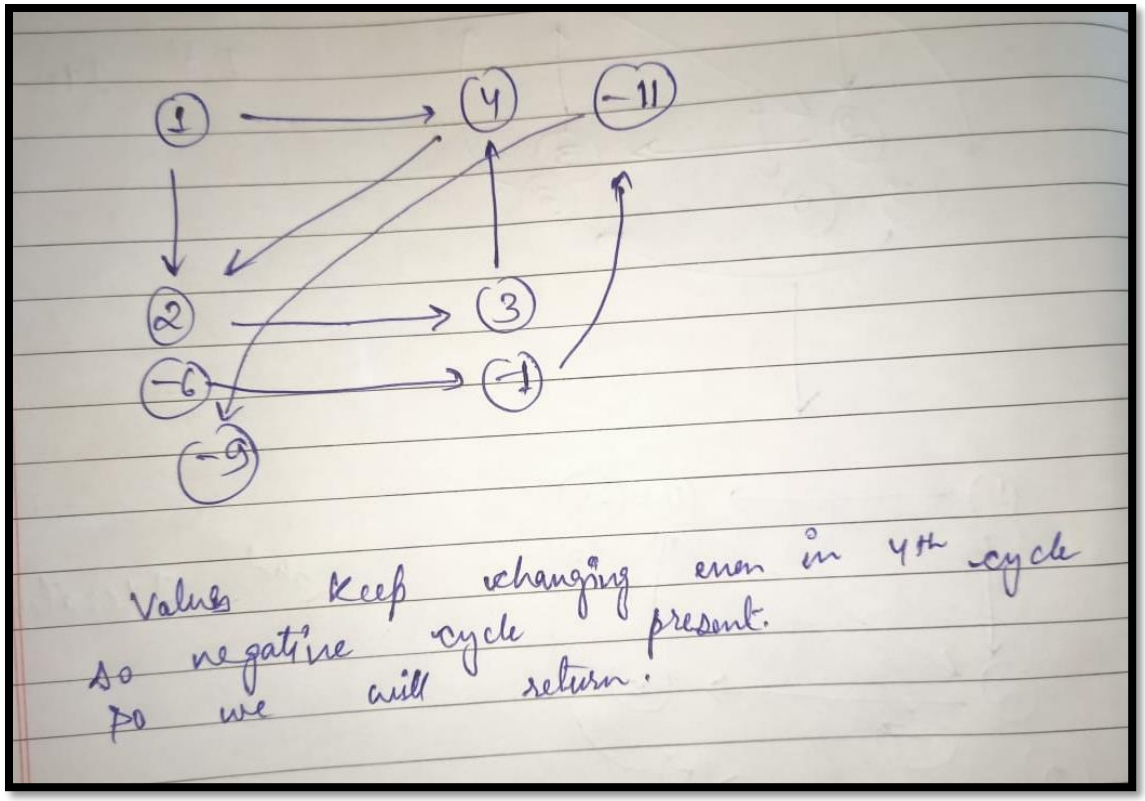
3. This step reports if there is a negative weight cycle in graph. Do following for each edge u-v

    - If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”

    - The idea of step 3 is, step 2 guarantees the shortest distances if the graph doesn’t contain a negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

Example:





Working:

This algorithm initializes the distance to the source to 0 and all other nodes to infinity. Then for all edges, if the distance to the destination can be shortened by taking the edge, the distance is updated to the new lower value. At each iteration *i* that the edges are scanned, the algorithm finds all shortest paths of at most length *i* edges (and possibly some paths longer than *i* edges). Since the longest possible path without a cycle can be edges, the edges must be scanned times to ensure the shortest path has been found for all nodes. A final scan of all the edges is performed and if any distance is updated, then a path of length edges has been found which can only occur if at least one negative cycle exists in the graph.

Time Complexity:

Time Complexity of bellman-ford is **O(E.n)** where E is no of edges and n is no of vertices

and in complete graph no of edges is:

$$\mathbf{E=n(n-1)/2}$$

where n is no of vertices

here

$$\mathbf{E=O(n^2)}$$

so, time complexity of bellman-ford algorithm on complete graph with n vertices is $\mathbf{O(n^3)}$**.**