Data Structures Lab
Assignment: 8

Q1. Upload link: https://www.dropbox.com/request/FhkDD7ev903GZayY80x7

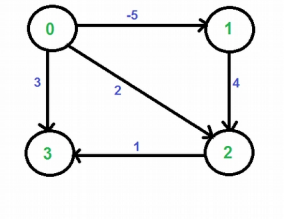
Implement Johnson's algorithm to find all-pairs shortest paths. Jonhson's algorithm is designed to be deployed in a given weighted directed Graph and weights may be negative. Johnson's algorithm uses both Dijkstra and Bellman-Ford algorithms as subroutines. The idea of Johnson's algorithm is to re-weight all edges and make them all positive, then apply Dijkstra's algorithm for every vertex.

Johnson's Algorithm:

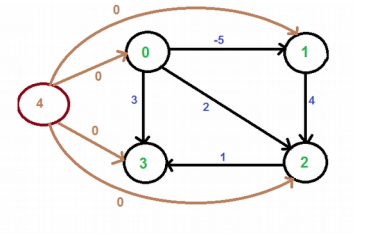
1. Let the given graph be G. Add a new vertex *s* to the graph, add edges from new vertex to all vertices of G. Let the modified graph be G'. Note that there is no edge to *s*, all edges are from *s*.
2. Run Bellman-Ford algorithm on G' with *s* as source. Let the distances calculated by Bellman-Ford be h[0], h[1], .. h[v-1].
3. Reweight the edges of original graph. For each edge (u, v), assign the new weight as "original weight + h[u] – h[v]", where h[u] and h[v] represent the minimum distance from new vertex *s* to *u* and *v* respectively.
4. Remove the added vertex *s* and run Dijkstra's algorithm for every vertex.

Example:

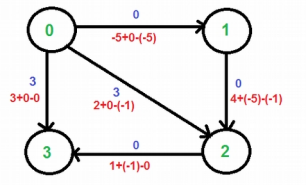
Let us consider the following graph.



We add a source s and add edges from s to all vertices of the original graph. In the following diagram s is 4.



We calculate the shortest distances from 4 to all other vertices using Bellman-Ford algorithm. The shortest distances from 4 to 0, 1, 2 and 3 are 0, -5, -1 and 0 respectively, i.e., h[] = {0, -5, -1, 0}. Once we get these distances, we remove the source vertex 4 and reweight the edges using following formula: w(u, v) = w(u, v) + h[u] - h[v], where h[v] is the vertex number.



Since all weights are positive now, we can run Dijkstra's shortest path algorithm for every vertex as source.

Q2.

In this problem, a rooted tree is a directed graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents.

The given input is a directed graph that started as a rooted tree with N nodes (with distinct values 1, 2, ..., N), with one additional directed edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of edges. Each element of edges is a pair [u, v] that represents a **directed** edge connecting nodes u and v, where u is a parent of child v.

Write a program that returns an edge that can be removed so that the resulting graph is a rooted tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array. Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

Example 1:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given directed graph will be like this:
  1
 / \
v   v
2-->3
```

Example 2:

```
Input: [[1,2], [2,3], [3,4], [4,1], [1,5]]
Output: [4,1]
Explanation: The given directed graph will be like this:
5 <- 1 -> 2
     ^    |
     |    v
     4 <- 3
```