```
var fs = require('fs');
var storedCollabsByPersonFile = "./collabsByPerson.json";
var initVertices = JSON.parse(fs.readFileSync(storedCollabsByPersonFile, 'utf8'));

var args = process.argv.slice(2);
var source = args[0];
var target = args[1];
bellmanFord = function (source, target, vertices){

        var cycles = false;
        var dists = {};
        var prev = {}
        var path = []

        //Set inital distances to infinity assuming they are next to each other and set neighbor on
        //shortest path from the source to undefined for all vertices
        Object.keys(vertices).forEach((key) => {
                dists[key] = Infinity;
                prev[key] = undefined;
        });

        //set distance from source to source to be 0
        dists[source] = 0;
        //start algorithm at the source
        //while vertices are still graph

        //for 0 to v-1 (do this v-1 times)
        for(var i=0; i<Object.keys(vertices).length-1; i++){
                //for each vertex
                Object.keys(vertices).forEach((currentVertex)=>{
                        //store edges before deleting object
                        //edges are between neighbors
                        var currentEdges = vertices[currentVertex];
                        //find new shortest paths to all neighboring vertices if available
                        Object.keys(currentEdges).forEach((neighbor) => {
                                //distance is -1/number Of times Collaborated,
                                //because 1/ number of times collaborated gives smaller distances
                                //to lots of collabs with same person
                                //negated for longest path
                                var testDist = (-1.0/currentEdges[neighbor])+dists[currentVertex];
                                //update if it makes sense (if you have a smaller distance than
                                //previously stored, then update.)
                                if(testDist < dists[neighbor]){
                                        //storing previous's allows us to recurse back
                                        //on the path once we find the "shortest" (actually longest bc of negation)
                                        prev[neighbor] = currentVertex;
```

```
                                        dists[neighbor] = testDist;
                                }
                        });
                })
        }
        //prepend the target to the list
        currentVertex = target;
        path.unshift(currentVertex);

        //recurse back down the path starting from target
        //go to each previous until you get to the source.
        while(prev[currentVertex] != source){
                // console.log("VERTEX:",currentVertex);
                // console.log("PREV:",prev[currentVertex]);

                //preprend prev to list and set new current to be the previous
                //This if else clause means that as you go through the previous's
                //and build up the path, if you come across a node you've already seen,
                //you have a cycle. So, in the else clause, we delete this last edge
                //in the cycle.
                if(path.indexOf(prev[currentVertex])==-1){
                        path.unshift(prev[currentVertex])
                        currentVertex = prev[currentVertex]
                } else{
                        // console.log("CYCLE IS:",currentVertex, "to", prev[currentVertex]);

                        //delete the cycle
                        delete vertices[prev[currentVertex]][currentVertex] //= -10;

                        cycles = true;
                        break
                }
        }

        //if we have no more cycles, we can give the path.
        if(!cycles){
                path.unshift(source)
                return path;
        } else {
                // console.log("RECURRRRRRRR")
                return bellmanFord(source, target, vertices)
        }
}
console.log(bellmanFord(source, target, initVertices))
module.exports = bellmanFord;
```