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

dijk = function (source, target, graph){

        var dists = {};
        var prev = {}
        var path = []
        if (graph) {
                vertices = graph;
        }
        vertices = JSON.parse(JSON.stringify(vertices));

        //Set inital distances to infinity (and beyond) 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
        var currentVertex = source;
        //while vertices are still graph
        while (Object.keys(vertices).length >0){
                //preset minDistance not visted yet to infinity
                var minDist = Infinity;

                //find unvisited node with minimum distance from source
                Object.keys(vertices).forEach((vertex) =>{
                        var vertDist = dists[vertex];
                        if(dists[vertex] < minDist) {
                                minDist = dists[vertex];
                                currentVertex = vertex;
                        }
                });

                //visit this minimum distance node by checking it's edges and updating
                //it's neighbors if necessary.

```

```javascript
		//store edges before deleting object
		var currentEdges = vertices[currentVertex];
		//delete the current vertex from graph because we have now visited it
		delete vertices[currentVertex];

		// If no more edges, no path is possible
		if (!currentEdges) {
			return [];
		} else {
			//find new shortest paths to all neighboring vertices if available
			Object.keys(currentEdges).forEach((neighbor) => {
				//1.0/currentEdges[neighbor] is the distance to this neighbor node
				//the reason we use this reciprocal is to give shorter distance
				//between 2 nodes with lots of collabs with eachother.
				//adding the distance up to this point
				var testDist = (1.0/currentEdges[neighbor])+dists[currentVertex];
				if(testDist < dists[neighbor]){
					prev[neighbor] = currentVertex;
					dists[neighbor] = testDist;
				}
			});

			//if our current vertex is the target, go down
			//the prev tree to find the whole path
			if(currentVertex === target){
				//prepend the target to the list
				path.unshift(currentVertex);

				while(prev[currentVertex] != undefined){
					//preprend prev to list and set new current to be the previous
					path.unshift(prev[currentVertex])
					currentVertex = prev[currentVertex]
				}
				return path;
			}
		}
	}
}

module.exports = dijk;
```