

## Programming Assignment 3

CS610-851

Author: Haard Shah

This assignment implements Dijkstra's algorithm to find shortest path between nodes, given a graph.

Main submission file: ***Assignment3.java***

Link to online interpreter with assignment code: <http://tpcg.io/veJrvp>

- To compile the code run:

```
$ javac Assignment3.java
```

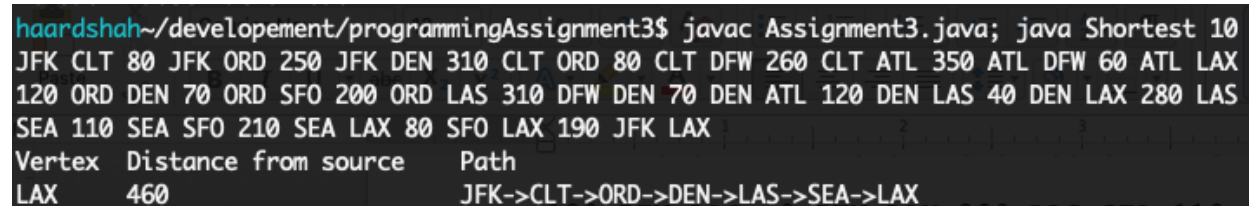
- To execute the code, run:

```
$ java Shortest <# nodes> <src1> <dest1> <distance1>...<srcN>  
                <destN> <distanceN> <srcNode> <destinationNode>
```

Example command:

```
javac Assignment3.java; java Shortest 10 JFK CLT 80 JFK ORD 250  
JFK DEN 310 CLT ORD 80 CLT DFW 260 CLT ATL 350 ATL DFW 60 ATL  
LAX 120 ORD DEN 70 ORD SFO 200 ORD LAS 310 DFW DEN 70 DEN ATL  
120 DEN LAS 40 DEN LAX 280 LAS SEA 110 SEA SFO 210 SEA LAX 80  
SFO LAX 190 JFK LAX
```

Sample run (image):



```
haardshah~/developement/programmingAssignment3$ javac Assignment3.java; java Shortest 10  
JFK CLT 80 JFK ORD 250 JFK DEN 310 CLT ORD 80 CLT DFW 260 CLT ATL 350 ATL DFW 60 ATL LAX  
120 ORD DEN 70 ORD SFO 200 ORD LAS 310 DFW DEN 70 DEN ATL 120 DEN LAS 40 DEN LAX 280 LAS  
SEA 110 SEA SFO 210 SEA LAX 80 SFO LAX 190 JFK LAX  
Vertex Distance from source Path  
LAX 460 JFK->CLT->ORD->DEN->LAS->SEA->LAX
```

## Programming Assignment 3

CS610-851

Author: Haard Shah

### Assignment3.java

```
/*
Programming Assignment 3
CS610-851
Author: Haard Sha
*/

import java.util.*;
import java.lang.*;
import java.io.*;
import java.util.stream.Collectors;

// for priority queue
class NodeComparator implements Comparator<Node> {
    public int compare(Node s1, Node s2) {
        if (s1.getDist() < s2.getDist())
            return -1;
        else if (s1.getDist() > s2.getDist())
            return 1;
        else
            return 0;
    }
}

class Node {
    int idx;
    int dist;
    boolean inCloud;
    ArrayList<Integer> path;

    public Node(int idx) {
        this.idx = idx;
        this.dist = Integer.MAX_VALUE;
        inCloud = false;

        path = new ArrayList<Integer>();
        // add destination node to path
        path.add(this.idx);
    }

    void setDist(int newDist) { this.dist = newDist; }

    int getDist() { return this.dist; }

    void addToCloud() { this.inCloud = true; }
```

### Programming Assignment 3

CS610-851

Author: Haard Shah

```
        boolean isInCloud() { return this.inCloud; }

        void changeNewPath(ArrayList<Integer> newPath) {
            path.clear();
            path.addAll(newPath);
            path.add(this.idx);
        }

        ArrayList<Integer> getPath() { return path; }
    }

    class Shortest {

        int numNodes;
        Node nodes[];
        Boolean cloud[];
        // PriorityQueue<Node> pq;

        public Shortest(int numNodes) {
            this.numNodes = numNodes;
            nodes = new Node[numNodes];
            cloud = new Boolean[numNodes];
            for (int idx = 0; idx < nodes.length; idx++) {
                nodes[idx] = new Node(idx);
                cloud[idx] = false;
            }

            // pq = new PriorityQueue<Node>(numNodes, new
NodeComparator());
        }

        int getMinNode() {
            int min = Integer.MAX_VALUE;
            int midx = -1;
            for (int i = 0; i < numNodes; i++) {
                if (nodes[i].inCloud == false &&
nodes[i].getDist() < min) {
                    min = nodes[i].getDist();
                    midx = i;
                }
            }
            return midx;
        }

        void printNodes() {
```

### Programming Assignment 3

CS610-851

Author: Haard Shah

```
        System.out.println("Vertex      Distance from source
Path");
        for (int i = 0; i < numNodes; i++) {

            System.out.println(i+"\t"+nodes[i].getDist()+"\t\t\t"+nodes
[i].getPath().toString());
        }
    }

    void printShortestDistanceTo(int idx) {
        System.out.println("Vertex      Distance from source
Path");
        int i = idx;

        System.out.println(i+"\t"+nodes[i].getDist()+"\t\t\t"+nodes
[i].getPath().toString());
    }

    private ArrayList<String>
mapPathIdxToNames(Hashtable<Integer, String> mapFromIdx,
ArrayList<Integer> origPath) {
        ArrayList<String> newPath = new ArrayList<String>();
        for (int i = 0; i < origPath.size(); i++) {
            newPath.add(mapFromIdx.get(origPath.get(i)));
        }
        return newPath;
    }

    void printNodesWithKeymap(Hashtable<Integer, String>
mapFromIdx) {
        System.out.println("Vertex      Distance from source
Path");
        for (int i = 0; i < numNodes; i++) {
            String vertexName = mapFromIdx.get(i);
            int dist = nodes[i].getDist();
            ArrayList<Integer> origPath = nodes[i].getPath();
            ArrayList<String> newPath =
mapPathIdxToNames(mapFromIdx, origPath);
            String path = newPath.stream().map(t ->
t).collect(Collectors.joining("->"));
            System.out.println(vertexName + "\t" + dist +
"\t\t\t" + path);
        }
    }
}
```

### Programming Assignment 3

CS610-851

Author: Haard Shah

```
void printPathToDestinationWithKeyMap(Hashtable<Integer,
String> mapFromIdx, int destNodeIdx) {
    System.out.println("Vertex      Distance from source
    Path");
    String vertexName = mapFromIdx.get(destNodeIdx);
    int dist = nodes[destNodeIdx].getDist();
    ArrayList<Integer> origPath =
nodes[destNodeIdx].getPath();
    ArrayList<String> newPath =
mapPathIdxToNames(mapFromIdx, origPath);
    String path = newPath.stream().map(t ->
t).collect(Collectors.joining("->"));
    System.out.println(vertexName + "\t" + dist + "\t\t\t"
+ path);
}

Node[] dijkstra(int graph[][], int src) {
    nodes[src].setDist(0);

    for (int i = 0; i < numNodes-1; i++) {
        // get node idx with min dist not in
        int u = getMinNode();
        nodes[u].addToCloud(); // sets inCloud property
to true

        for (int v = 0; v < numNodes; v++) {
            if (nodes[v].isInCloud()) continue;
            if (graph[u][v] == 0) continue; // not an
edge

            // System.out.println("Print nodes v: "+v);
            // printNodes();

            // perform relaxation on edge (u, v)
            int newDistance = nodes[u].getDist() +
graph[u][v];
            if (newDistance < nodes[v].getDist()) {
                nodes[v].setDist(newDistance);

                nodes[v].changeNewPath(nodes[u].getPath());
            }
        }

        // printNodes();
        return nodes;
    }
}
```

## Programming Assignment 3

CS610-851

Author: Haard Shah

```
static void printGraph(int graph[][]) {
    for (int i = 0; i < graph.length; i++) {
        System.out.println(Arrays.toString(graph[i]));
    }
}

public static void main(String[] args) {
    // test(); System.exit(0);
    if ((args.length - 3) % 3 != 0 && args.length >= 6) {
        System.out.println("Usage: java Shortest <#
nodes> <src1> <dest1> <distance1>...<srcN> <destN> <distanceN>
<srcNode> <destinationNode>");
        System.exit(1);
    }

    // get graph from args
    int numNodes = Integer.parseInt(args[0]);
    Shortest s = new Shortest(numNodes);
    int graph[][] = new int[numNodes][numNodes];
    // System.out.println(Arrays.deepToString(graph));
    System.exit(1);
    int idx = 0;
    Hashtable<String, Integer> mapToIdx = new
Hashtable<String, Integer>();
    for (int i = 1; i < args.length-2; i+=3) {
        String src = args[i];
        String dest = args[i+1];
        int srcIdx;
        int destIdx;
        int distance = Integer.parseInt(args[i+2]);
        if (mapToIdx.containsKey(src)) {
            srcIdx = mapToIdx.get(src);
        } else {
            srcIdx = idx++;
            mapToIdx.put(src, srcIdx);
        }
        if (mapToIdx.containsKey(dest)) {
            destIdx = mapToIdx.get(dest);
        } else {
            destIdx = idx++;
            mapToIdx.put(dest, destIdx);
        }
        // then set graph
        graph[srcIdx][destIdx] = distance;
        graph[destIdx][srcIdx] = distance;
    }
}
```

## Programming Assignment 3

CS610-851

Author: Haard Shah

```
        }
        int srcNodeIdx; int destNodeIdx;
        String srcNodeName = args[args.length-2];
        String destNodeName = args[args.length-1];
        if (!mapToIdx.containsKey(srcNodeName) ||
!mapToIdx.containsKey(destNodeName)) {
            System.out.println("Input error: source and
destination node names must be connected to other components in
the graph provided."); System.exit(1);
        }
        srcNodeIdx = mapToIdx.get(srcNodeName);
        destNodeIdx = mapToIdx.get(destNodeName);

        // printGraph(graph); System.exit(1);
        Hashtable<Integer, String> mapFromIdx = new
Hashtable<Integer, String>();
        for (String key : mapToIdx.keySet()) {
mapFromIdx.put (mapToIdx.get(key), key); }

        s.dijkstra(graph, srcNodeIdx);

        s.printPathToDestinationWithKeyMap (mapFromIdx,
destNodeIdx);
        // print all results
        // s.printNodesWithKeymap (mapFromIdx);
    }
}
```