An Application to Network Design

PROJECT 1

Debaspreet Chowdhury
10/7/2014
UTD ID - 2021211268

To implement the basic network design model that is presented in the lecture note entitled "An Application to Network Design" and experiment with it.

TASK 1

1. I have taken the help of the following resources to UNDERSTAND Dijkstras Algorithm and its implementation in JAVA.
   a) http://www.algolist.com/code/java/Dijkstra%27s_algorithm
   b) http://www.vogella.com/tutorials/JavaAlgorithmsDijkstra/article.html
   c) http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
   d) https://www.youtube.com/watch?v=LpJceTbzJFo
   e) http://cs.fit.edu/~ryan/java/programs/graph/Dijkstra-java.html
   f) http://algs4.cs.princeton.edu/44sp/DijkstraSP.java.html
   g) http://codereview.stackexchange.com/questions/32354/implementation-of-dijkstras-algorithm
   h) https://www.youtube.com/watch?v=zXfDYaahsNA
2. The code however has been written by me.
3. For simplification purpose, I have combined the b_{ij} and a_{ij} metrics into one single metric. However the "Fast Solution Method" does indeed instruct to add only the costs over each link. The same model has been incorporated into my JAVA program.
4. The program takes Number of nodes, Costs (in a matrix form) and Source node as INPUT and OUTPUTS the shortest path to each node from the Source node and Overall Network Cost based on the path metrics.
5. Java has been used as the programming

PROGRAM

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.Scanner;
import java.util.Set;

public class shortestpath
{
        private int[] distances;
        private Set<Integer> settled;
        private Set<Integer> unsettled;
        private int nodes;
        private int[][] edges;

        public shortestpath(int nodes)
        {
            this.nodes = nodes;
            distances = new int[nodes + 1];
            settled = new HashSet<Integer>();
            unsettled = new HashSet<Integer>();
            edges = new int[nodes + 1][nodes + 1];
        }
```

```java
public void algorithm(int tempedges[][], int source)
{
    int testNode;
    for (int i = 1; i <= nodes; i++)
        for (int j = 1; j <= nodes; j++)
            edges[i][j] = tempedges[i][j];
    for (int i = 1; i <= nodes; i++)
        distances[i] = Integer.MAX_VALUE;

    unsettled.add(source);
    distances[source] = 0;

    while (!unsettled.isEmpty())
    {
        testNode = minDistUnset();
        unsettled.remove(testNode);
        settled.add(testNode);
        neighbours(testNode);
     }
}

private int minDistUnset()
{
    int min ;
    int node = 0;
    Iterator<Integer> iterator = unsettled.iterator();
    node = iterator.next();
    min = distances[node];
    for (int i = 1; i <= distances.length; i++)
    {
        if (unsettled.contains(i))
        {
            if (distances[i] <= min)
            {
                min = distances[i];
                node = i;
            }
        }
    }
    return node;
}

private void neighbours(int testNode)
{
    int edgeDistance = -1;
    int newDistance = -1;
    for (int destinationNode = 1; destinationNode <= nodes;destinationNode++)
    {
        if (!settled.contains(destinationNode))
        {
            if (edges[testNode][destinationNode] != Integer.MAX_VALUE)
            {
                edgeDistance = edges[testNode][destinationNode];
                newDistance = distances[testNode] + edgeDistance;
```

```
                    if (newDistance < distances[destinationNode])
                    {
                        distances[destinationNode] = newDistance;
                    }
                    unsettled.add(destinationNode);
                }
            }
        }
    }

    public static void main(String args[])
    {
        int[][] edges;
        int vertices;
        int source = 0;
        Scanner userinput = new Scanner(System.in);
            System.out.println("Enter number of nodes");
            vertices = userinput.nextInt();
            edges = new int[vertices + 1][vertices + 1];
            System.out.println("Enter the edges");
            for (int i = 1; i <= vertices; i++)
            {
                for (int j = 1; j <= vertices; j++)
                {
                    edges[i][j] = userinput.nextInt();
                    if (i == j)
                    {
                        edges[i][j] = 0;
                        continue;
                    }
                    if (edges[i][j] == 0)
                    {
                        edges[i][j] =  Integer.MAX_VALUE;
                    }
                }
            }

            int networkCost = 0;
            for (int i = 1; i <= vertices; i++)
            {
                for (int j = 1; j <= vertices; j++)
                {
                     networkCost += edges[i][j];
                     System.out.print("|" +edges[i][j] +"|");
                }
                System.out.println("");
            }

            System.out.println("Enter the source node");
            source = userinput.nextInt();
            shortestpath algObject = new shortestpath(vertices);
            algObject.algorithm(edges, source);

            for (int i = 1; i <= algObject.distances.length - 1; i++)
            {
```

```
                        System.out.println


                 ("Shortest path from "+ source + " to " + i +
                                   " is "+ algObject.distances[i]);
                    }

                    System.out.println("");
                    System.out.println("Total network cost is "+networkCost);

                userinput.close();
            }
        }
```

OUTPUT

```
Enter number of nodes
4
Enter the edges
1
2
3
4
1
2
4
5
6
7
8
9
2
5
7
3
|0||2||3||4|
|1||0||4||5|
|6||7||0||9|
|2||5||7||0|
Enter the source node
1
Shortest path from 1 to 1 is 0
Shortest path from 1 to 2 is 2
Shortest path from 1 to 3 is 3
Shortest path from 1 to 4 is 4

Total network cost is 55
```

TASK 2

The following algorithm is used –

```
Foreach node set distance[node] = HIGH
SettledNodes = empty
UnSettledNodes = empty

Add sourceNode to UnSettledNodes
distance[sourceNode]= 0

while (UnSettledNodes is not empty) {
  evaluationNode = getNodeWithLowestDistance(UnSettledNodes)
  remove evaluationNode from UnSettledNodes
    add evaluationNode to SettledNodes
    evaluatedNeighbors(evaluationNode)
}

getNodeWithLowestDistance(UnSettledNodes){
  find the node with the lowest distance in UnSettledNodes and return it
}

evaluatedNeighbors(evaluationNode){
  Foreach destinationNode which can be reached via an edge from
evaluationNode AND which is not in SettledNodes {
    edgeDistance = getDistance(edge(evaluationNode, destinationNode))
    newDistance = distance[evaluationNode] + edgeDistance
    if (distance[destinationNode]  > newDistance) {
      distance[destinationNode]  = newDistance
      add destinationNode to UnSettledNodes
    }
  }
}
```

TASK 3

The following considerations have gone into creating the following program –

1. Random values of “K” are generated each time the program is run.
2. Each value of “K” has an effect of creating weights for the edges.

PROGRAM

```java
import java.util.HashSet;
import java.util.Iterator;
import java.util.Scanner;
import java.util.Set;

public class shortestpathagain
    {
        private int[] distances;
        private Set<Integer> settled;
        private Set<Integer> unsettled;
        private int nodes;
        private int[][] edges;

        public shortestpathagain(int nodes)
        {
            this.nodes = nodes;
            distances = new int[nodes + 1];
            settled = new HashSet<Integer>();
            unsettled = new HashSet<Integer>();
            edges = new int[nodes + 1][nodes + 1];
        }

        public void algorithm(int tempedges[][], int source)
        {
            int testNode;
            for (int i = 1; i <= nodes; i++)
                for (int j = 1; j <= nodes; j++)
                    edges[i][j] = tempedges[i][j];
            for (int i = 1; i <= nodes; i++)
                distances[i] = Integer.MAX_VALUE;

            unsettled.add(source);
            distances[source] = 0;

            while (!unsettled.isEmpty())
            {
                testNode = minDistUnset();
                unsettled.remove(testNode);
                settled.add(testNode);
                neighbours(testNode);
             }
        }
        private int minDistUnset()
        {
            int min ;
            int node = 0;
            Iterator<Integer> iterator = unsettled.iterator();
            node = iterator.next();
            min = distances[node];
            for (int i = 1; i <= distances.length; i++)
            {
                if (unsettled.contains(i))
                {
                    if (distances[i] <= min)
                    {
                        min = distances[i];
                        node = i;
                    }
                }
            }

            return node;
        }
        private void neighbours(int testNode)
        {
            int edgeDistance = -1;
            int newDistance = -1;
            for (int destinationNode = 1; destinationNode <= nodes;
destinationNode++)
            {
                if (!settled.contains(destinationNode))
                {
                    if (edges[testNode][destinationNode] != Integer.MAX_VALUE)
                    {
                        edgeDistance = edges[testNode][destinationNode];
                        newDistance = distances[testNode] + edgeDistance;
                        if (newDistance < distances[destinationNode])
                        {
                            distances[destinationNode] = newDistance;
                        }
                        unsettled.add(destinationNode);
                    }
                }
            }
        }

        public static void main(String args[])
        {
            int[][] edges;
            int vertices;
            int source = 0;
            Scanner userinput = new Scanner(System.in);

                System.out.println("Enter number of nodes");
                vertices = userinput.nextInt();
                edges = new int[vertices + 1][vertices + 1];
```

```java
//System.out.println("Enter the edges");

// Using random number generators
int[] newmatrix = new int[vertices*vertices];
int a = (int)(3 + Math.random()*14);
int b = (int)(Math.random()*5);
int k=0;
int x=0;
int r=vertices*a;
for(int i = 0;i<r;i++){
    newmatrix[i]=1*b;
}
for(int i = r;i<vertices*vertices;i++){
    newmatrix[i]=250*b;
}
for (int i = 1; i <= vertices; i++)
{
    for (int j = 1; j <= vertices; j++)
    {
        edges[i][j]=newmatrix[k];
        k++;
    }
 }


for (int i = 1; i <= vertices; i++)
{
    for (int j = 1; j <= vertices; j++)
    {
        if (i == j)
        {
            edges[i][j] = 0;
            continue;
        }
        if (edges[i][j] == 0)
        {
            edges[i][j] =  Integer.MAX_VALUE;
        }
    }
}

for (int i = 1; i <= vertices; i++)
{
    for (int j = 1; j <= vertices; j++)
    {
          System.out.print("|" +edges[i][j] +"|");
    }
    System.out.println("");
}

System.out.println("Enter the source node");
source = userinput.nextInt();
shortestpathagain algObject = new shortestpathagain(vertices);
algObject.algorithm(edges, source);
```

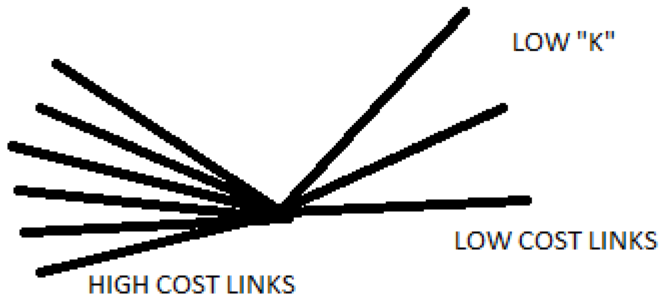
```
                for (int i = 1; i <= algObject.distances.length - 1; i++)
                {
                    System.out.println
                    ("Shortest path from "+ source + " to " + i +
                                 " is "+ algObject.distances[i]);
                }

            userinput.close();
        }
    }
```

OUTPUT

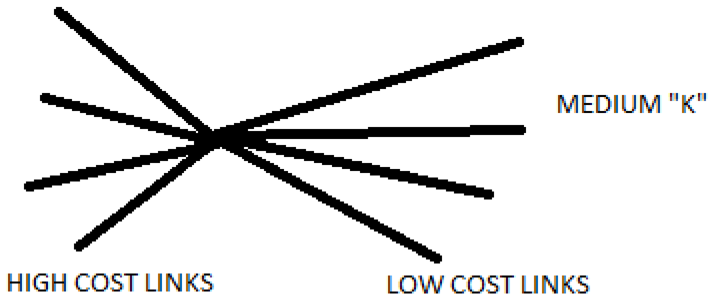
```
Enter number of nodes
20
|0||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4|
|4||0||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4|
|4||4||0||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4||4|
|1000||1000||1000||0||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||0||1000||1000||1000||1000||1000||1000||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||0||1000||1000||1000||1000||1000||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||0||1000||1000||1000||1000||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||0||1000||1000||1000||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||0||1000||1000||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||0||1000||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||0||1000||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||0||1000||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||0||1000||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||0||100
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||
0||1000||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||
1000||0||1000||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||
1000||1000||0||1000||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||
1000||1000||1000||0||1000||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||
1000||1000||1000||1000||0||1000|
|1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||1000||
1000||1000||1000||1000||1000||0|
Enter the source node
2
Shortest path from 2 to 1 is 4
Shortest path from 2 to 2 is 0
Shortest path from 2 to 3 is 4
Shortest path from 2 to 4 is 4
Shortest path from 2 to 5 is 4
Shortest path from 2 to 6 is 4
Shortest path from 2 to 7 is 4
Shortest path from 2 to 8 is 4
Shortest path from 2 to 9 is 4
Shortest path from 2 to 10 is 4
Shortest path from 2 to 11 is 4
Shortest path from 2 to 12 is 4
Shortest path from 2 to 13 is 4
Shortest path from 2 to 14 is 4
Shortest path from 2 to 15 is 4
Shortest path from 2 to 16 is 4
Shortest path from 2 to 17 is 4
Shortest path from 2 to 18 is 4
Shortest path from 2 to 19 is 4
Shortest path from 2 to 20 is 4
```

TASK 4

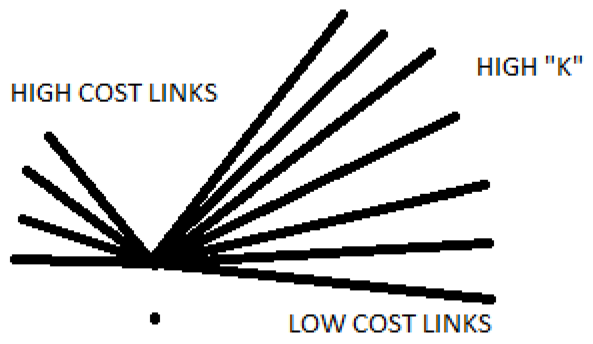
Low “K” value –



Medium “K” value –



High “K” value –



1. More the value of “K”, less is the cost.
2. More the value of “K”, more is the density of low cost links.

TASK 5

Assigning low cost weights to each edge, relaxes the evaluation of the high cost links thus speeding up the process of finding the shortest path. So less the number of low cost links, more easy it'll be for the algorithm to find the shortest path because th algorithm will have lesser links to evaluate the low cost on.