

CS 542

Final Project Report

Team Member:

HUNAG JIE A20279698 (Live)

GU YIZHI A20334298(Live)

Part One

I. Source code

In this source code part, there are total 4 classes included:

1. Application class;
2. Router class;
3. RouterTable class;
4. FindPath class;

Application.java:

```
import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;
public class Application {
    public static void main(String [] args){

        String fileName;
        String li,resp;
        int [] ar,ar2,ar3,ar4,ar5,ar6;
        int rotNum=0, i=0,j=0, source,dest,source1,dest1,fRouter;
        ArrayList<int []> tableArray=new ArrayList<int []>();
        Router soc,det,soc1,det1,downRouter;
        try{
            //input file
            Scanner sc1=new Scanner(System.in);
            System.out.println("Welcome! ");
            System.out.println("Enter the input file name: ");
            fileName=sc1.nextLine();

            //upload the file into matrix
            Scanner sc2=new Scanner(new BufferedReader(new
            FileReader(fileName)));
```

```

        while(sc2.hasNextLine()){
            sc2.nextLine();
            rotNum++;
        }
        int[][] matx= new int[rotNum][rotNum];
        Scanner sc3=new Scanner(new BufferedReader(new
FileReader(fileName)));
        while(sc3.hasNext()){
            li=sc3.next();
            matx[i][j]=Integer.parseInt(li);
            if(j<rotNum-1){
                j++;
            }
            else{
                j=0;
                i++;
            }
        }

        //output the matrix

System.out.println("-----");
System.out.println("The initial matrix is as follows:");
for(int m=0;m<rotNum;m++){
    System.out.println();
    for(int n=0;n<rotNum;n++){
        System.out.print(matx[m][n]+"\\t ");
    }
}

//initial the routers

for(int m=0;m<rotNum;m++){
    ar=new int [rotNum];
    for(int n=0;n<rotNum;n++){
        ar[n]=matx[m][n];
    }
    tableArray.add(ar);
}
//Create a content of routers
RouterTable rTable=new RouterTable();

```

```

        for(int m=0;m<rotNum;m++){
            Router re=new
Router(m,tableArray.get(m));//initialize each router
            rTable.add(re);
        }
        for(int m=0;m<rotNum;m++){
            rTable.pop(m).findNeighb(rTable);//find neighbor of
each router
        }
        for(int m=0;m<rotNum;m++){
            rTable.pop(m).send(rTable);//start update the table of
each router
        }

//output the number of iterations
        System.out.println();

        System.out.println("-----");
        System.out.print("The required number of iterations n = ");
        System.out.print(rTable.pop(0).getIte());

//output the final matrix
        System.out.println();

        System.out.println("-----");
        System.out.println("The final matrix computed by the DV
algorithm is as follows: ");
        for(int m=0;m<rotNum;m++){
            for(int k=0;k<rTable.pop(m).getTable().size();k++){
                if(rTable.pop(m).getTable().get(k)<999){

                    System.out.print(rTable.pop(m).getTable().get(k)+"\t ");
                }else
                {
                    System.out.print("NA"+" \t ");
                }
            }
            System.out.println();
        }

//output the shortest path repeatedly

```

```

do{
    //make the user input the source and destination
    System.out.println();

    System.out.println("-----");
    System.out.println("Enter the source and destination
nodes: ");
    Scanner sc4=new Scanner(System.in);
    source=sc4.nextInt();
    dest=sc4.nextInt();
    soc=rTable.pop(source-1);
    det=rTable.pop(dest-1);
    FindPath fp=new FindPath();
    fp.sPath(soc, det);

    //output the length of path from source to destination

    System.out.println("-----");
    System.out.print("The length of this path is :");
    System.out.println("\t"+fp.pLength(soc,det));

    //provide an other chance to find the path
    System.out.println("Another source-destination
pair ?(yes/no)");
    Scanner sc5=new Scanner (System.in);
    resp=sc5.next();
}while(resp.equals("yes"));

    //make the user enter the router which they want to set down

    System.out.println("-----");
    System.out.println("Enter the number of the router whose
failure is to be simulated:");
    Scanner sc6=new Scanner(System.in);
    fRouter=sc6.nextInt();
    downRouter=rTable.pop(fRouter-1);
    downRouter.setDown(rTable);

    //output the number of iterations
    System.out.println();

```

```

System.out.println("-----");
System.out.print("The required number of iterations n = ");
System.out.print(rTable.pop(0).getIte());

//output the final matrix
System.out.println();

System.out.println("-----");
System.out.println("The final matrix computed by the DV
algorithm is as follows: ");
for(int m=0;m<rotNum;m++){
    for(int k=0;k<rTable.pop(m).getTable().size();k++){
        if(rTable.pop(m).getTable().get(k)<999){

System.out.print(rTable.pop(m).getTable().get(k)+" \t");
        }else
        {
            System.out.print("NA"+" \t");//set the distance of
unreachable router to NA
        }
    }
    System.out.println();
}

//output the shortest path repeatedly
do{
    System.out.println();

System.out.println("-----");
System.out.println("Enter the source and destination
nodes: ");
Scanner sc4=new Scanner(System.in);
source1=sc4.nextInt();
dest1=sc4.nextInt();
soc1=rTable.pop(source1-1);
det1=rTable.pop(dest1-1);
FindPath fp1=new FindPath();
fp1.sPath(soc1, det1);

//output the length of path from source to destination

```

```

System.out.println("-----");
    System.out.print("The length of this path is :");
    System.out.println("\t"+fp1.pLength(soc1, det1));

    //provide an other chance to find the path
    System.out.println("Another source-destination
pair ?(yes/no)");
    Scanner sc5=new Scanner (System.in);
    resp=sc5.next();
    }while(resp.equals("yes"));
    System.out.println();

System.out.println("-----");
    System.out.println("Thanks for using");

    }catch(IOException io){
        System.out.println(io);
    }
}
}

```

Router class.java:

```

import java.util.ArrayList;

public class Router {
    ArrayList <Integer> table=new ArrayList<Integer>();
    ArrayList <Integer> copTb;
    ArrayList <Router> neigb=new ArrayList<Router>();

    Router [] nextHop;
    public static int iter=0;

    int routName;
    boolean flag;

    //constructor of Router class

```

```

public Router(int rtn, int[] ary){
    routName=rtn;
    nextHop=new Router[ary.length];
    for(int i=0;i<ary.length;i++)
    {
        table.add(ary[i]); //add the initial record of router
distance
    }

};

//find the neighbor of router
public void findNeighb(RouterTable rt){
    for(int i=0;i<table.size();i++)
    {
        //Check the initial distance to find the neighbor
        if(table.get(i)!= 999 && table.get(i)!=0)
        {
            neighb.add(rt.pop(i)); //record the neighbor router
            nextHop[i]=rt.pop(i); //update the nextHop
        }
    }

}

//send the record to the neighbor router
public void send(RouterTable rt){
    for(int i=0;i<neighb.size();i++){
        neighb.get(i).updt(routName,table,rt);
    }
}

//update the routing table by using the received record
public void updt(int rName, ArrayList<Integer> tb, RouterTable
rt){
    copTb=new ArrayList<Integer>(); //copy the received record
    flag=false;
    for(int i=0;i<tb.size();i++)
    {
        copTb.add(tb.get(i));
    }
}

```



```

    for (int i=0;i<tb.size();i++)
    {
        copTb.set(i,copTb.get(i)+table.get(rName));
        //check if update is needed
        if(copTb.get(i)<table.get(i)){
            table.set(i,copTb.get(i));
            nextHop[i]=rt.pop(rName);
            flag=true;//if the record is updated, set flag to true
        }else if(copTb.get(i)>table.get(i) &&
nextHop[i]==rt.pop(rName)){
            table.set(i, copTb.get(i));
            flag=true;//if the record is updated, set flag to true
        }

    }
    //if the table is updated, then call the send method
    if(flag==true){
        iter++;
        this.send(rt);
    }
}

//return the record distance of the router
public ArrayList<Integer> getTable()
{
    for(int i=0;i<table.size();i++)
    {
        if(table.get(i)>999)
        {
            table.set(i,999);
        }
    }
    return table;
}

//return the iteration times
public int getIte(){
    return iter;
}

//return the nextHop of destination router

```

```

    public Router getNextHop(int i){
        return nextHop[i];
    }

    //set this router down
    public void setDown(RouterTable rt){
        iter=0;
        for(int i=0;i<table.size();i++)
        {
            table.set(i, 999);
        }
        this.send(rt);
    }
}

```

RouterTable.java:

```

import java.util.ArrayList;

public class RouterTable {
    ArrayList <Router> router=new ArrayList<Router>();

    //the constructor of RouterTable
    public RouterTable(){

    }

    //add a router into the content
    public void add(Router rot)
    {
        router.add(rot);
    }

    //return the i-th router
    public Router pop(int i)
    {
        return router.get(i);
    }
}

```

.....

FindPath.java:

```
import java.util.ArrayList;

public class FindPath {
    Router src, dest;
    RouterTable rTable;
    int rNum1, rNum2;
    int count=0;
    ArrayList<Integer> path=new ArrayList<Integer>();

    //the constructor of FindPath
    public FindPath(){
    }

    //print the path from the source to the destination
    public void sPath(Router source, Router destination){
        count++;
        rNum1=source.routName;
        rNum2=destination.routName;
        System.out.print(rNum1+1+"->");
        if(source.getNextHop(rNum2).equals(destination)){
            System.out.println(rNum2+1);
        }
        else{
            sPath(source.getNextHop(rNum2),destination);
        }
    }

    //return the path length between source and destination
    public int pLength(Router source, Router destination){
        return source.table.get(destination.routName);
    }
}
```

II. Input and Output

1. Input file

datafile1.txt

0	3	999	7	999	999
3	0	6	999	1	999
999	6	0	2	999	5
7	999	2	0	4	999
999	1	999	4	0	9
999	999	5	999	9	0

2. Print out

Welcome!

Enter the input file name:

datafile1.txt

The initial matrix is as follows:

0	3	999	7	999	999
3	0	6	999	1	999
999	6	0	2	999	5
7	999	2	0	4	999
999	1	999	4	0	9
999	999	5	999	9	0

The required number of iterations n = 16

The final matrix computed by the DV algorithm is as follows:

0	3	9	7	4	13
3	0	6	5	1	10
9	6	0	2	6	5
7	5	2	0	4	7
4	1	6	4	0	9

13 10 5 7 9 0

Enter the source and destination nodes:

6 2

6->5->2

The length of this path is : 10

Another source-destination pair ?(yes/no)

yes

Enter the source and destination nodes:

1 3

1->2->3

The length of this path is : 9

Another source-destination pair ?(yes/no)

no

Enter the number of the router whose failure is to be simulated:

5

The required number of iterations n = 35

The final matrix computed by the DV algorithm is as follows:

0	3	9	7	NA	14
3	0	6	8	NA	11
9	6	0	2	NA	5
7	8	2	0	NA	7
NA	NA	NA	NA	NA	NA
14	11	5	7	NA	0

Enter the source and destination nodes:

6 2

6->3->2

The length of this path is : 11

Another source-destination pair ?(yes/no)

yes

Enter the source and destination nodes:

1 3

1->2->3

The length of this path is : 9

Another source-destination pair ?(yes/no)

no

Thanks for using

Part two

I. Design and Test report

In this project, we use Java to implement the Distance Vector Routing Algorithm (DV). As the DV algorithm described, each router needs to update its routing table asynchronously, whenever it has received a record from its neighbor. In other words, each router executes part of the whole algorithm in the Bellman-Ford algorithm. After a router has updated its routing table, it should send the result to its neighbors so that they can also update their routing table.

When we doing the implementation, we create a class of Router because the processing is distributive and we need a unique object for every router. In the router class we create ***table*** to save the record of distance to other routers; we create ***neighb*** to save the neighbors of the router; we create ***nextHop*** to save the next hop to certain destination.

To simulate the DV algorithm in network, we first trigger a ***send*** method in every router. When a router received a record from its neighbor, it will check if update is needed by using ***updt*** method. This send-receive mechanism is implemented by following code:

```
public void send(RouterTable rt){
    for(int i=0;i<neighb.size();i++){
        neighb.get(i).updt(routName,table,rt);
    }
}
```

When a router sends its record to its neighbor, it will trigger the neighbors' **updt** method. And these steps will repeat until there is no update in any router.

If update is needed, we not only change the distance in **table** but also set the corresponding nextHop as the sender. For example:

We have a router R1 receives a record from router R2, and update the table and nextHop.

Original Table for R1

Destination	R1	R2	R3
Distance	0	3	9
NextHop	NA	R2	R4

Now, R1 receives a record from R2

Destination	R1	R2	R3
Distance	3	0	2

After the update, the Table for R1

Destination	R1	R2	R3
Distance	0	3	5
NextHop	NA	R2	R2

In the table of router R1 we update the distance and also the nextHop.

When we finding the shortest path to destination, we first check the

nextHop (Ri) of source router, then check the nextHop of the Ri. Do these steps repeatedly until we find the destination.

And the distance from the source to the destination can be found in the table of source.

We simulate the failure of router X by set all the distance of it to 999 using **setDown** method. Then we send its record to its neighbor to update the whole network. When we doing the update we check an addition condition to see if the update is needed. The code is as below:

```
else if(copTb.get(i)>table.get(i) && nextHop[i]==rt.pop(rName)){
    table.set(i, copTb.get(i));
    flag=true;//if the record is updated, set flag to true
}
```

Assume R2 is down, R2 will send the table below to its neighbor

Destination	R1	R2	R3
Distance	999	999	999

Assume R1 one of R2's neighbors; its original table is as below

Destination	R1	R2	R3
Distance	0	3	9
NextHop	NA	R2	R2

R1 will updated to the table below

Destination	R1	R2	R3
Distance	0	999	999
NextHop	NA	R2	R2

And it will send the record to its neighbor, and the new shortest path will be found by doing the send and update repeatedly.

The second part of line 29 of pseudocode is doing the same check, the mistake is:

R.next==Ti.next

The correct one should be

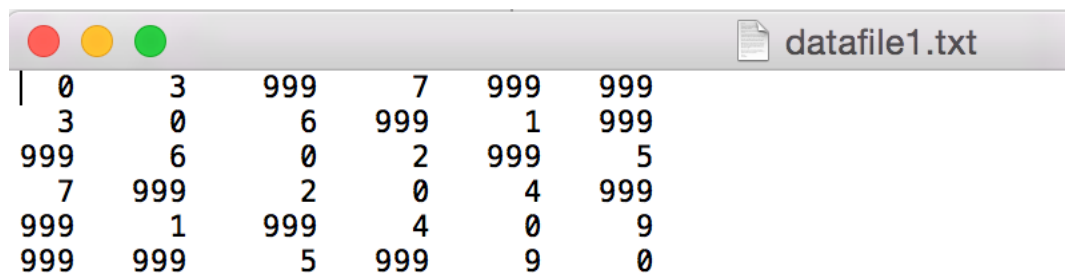
R==Ti.next

It's the same as we did in our code. Router should update its record immediately, when, for some destination, its nextHop is the sender.

Part Three

Program Instruction (ReadMe)

This Java program can be compiled by Eclipse or Command Line. Before you run this program, you should input a **txt** file with format as below:



A screenshot of a text editor window titled 'datafile1.txt'. The window contains a 6x6 matrix of numbers. The first row is: 0, 3, 999, 7, 999, 999. The second row is: 3, 0, 6, 999, 1, 999. The third row is: 999, 6, 0, 2, 999, 5. The fourth row is: 7, 999, 2, 0, 4, 999. The fifth row is: 999, 1, 999, 4, 0, 9. The sixth row is: 999, 999, 5, 999, 9, 0.

0	3	999	7	999	999
3	0	6	999	1	999
999	6	0	2	999	5
7	999	2	0	4	999
999	1	999	4	0	9
999	999	5	999	9	0

In the txt file, there should be an N*N matrix. The numbers inside the matrix represent the distance between routers:

0 means the distance to itself,

999 means that the path to the destination is currently unknown.

And this file is based on the network topology diagram.

After you have the file, you need put it into the project folder. And run this program. The console will instruct you to finish the program.