```
//original code from http://www.sanfoundry.com/java-program-
find-mst-using-prims-algorithm/
//last editied by tanveer bariana
//all assumptions of input come from how input.txt is formated

import java.util.InputMismatchException;//both exceptions handle
input
import java.io.FileNotFoundException;
import java.util.Scanner;// added to handle file
import java.io.File; //added to handle files
public class Prims
{
    private boolean unsettled[];//array of unsettled nodes
    private boolean settled[];// array of handled/ reached nodes
    private int numberofvertices;//# of vertices
    private double adjacencyMatrix[][];//main adjecency martrix
    private double key[];
    public static final int INFINITE = 999;
    private int parent[];


    public Prims(int numberofvertices)//set up all need for
prims algorythm with how many vertices included
    {
        this.numberofvertices = numberofvertices;
        unsettled = new boolean[numberofvertices ];
        settled = new boolean[numberofvertices ];
        adjacencyMatrix = new
double[numberofvertices][numberofvertices];
        key = new double[numberofvertices];//weight of edge
being commpared
        parent = new int[numberofvertices];
    }

    public double getUnsettledCount(boolean unsettled[])//check
how many are unsorted
    {
        int count = 0;
        for (int index = 0; index < unsettled.length; index++)
        {
            if (unsettled[index])//if index has yet to be
```

```
reached count++
            {
                count++;
            }
        }
        return count;
    }

    public void primsAlgorithm(double adjacencyMatrix[][])
   {
        int evaluationVertex;
        for (int source = 0; source < numberofvertices;
source++)
        {
            for (int destination = 0; destination <
numberofvertices; destination++)
            {
             this.adjacencyMatrix[source][destination] =
adjacencyMatrix[source][destination]; //dump adjecncymatrix
scanned into adjacenymatrix of program to work on
            }
        }

        for (int index = 0; index < numberofvertices; index++)
//set key to large number for each vertex
        {
            key[index] = INFINITE;
        }
        key[0] = 1;//declarations of first element on each of
needed arrays
        unsettled[0] = true;
        parent[0] = 1;

        while (getUnsettledCount(unsettled) != 0)
        {
            evaluationVertex =
getMimumKeyVertexFromUnsettled(unsettled);
            unsettled[evaluationVertex] = false;
            settled[evaluationVertex] = true;
            evaluateNeighbours(evaluationVertex);
        }
```

```
    }

    private int getMimumKeyVertexFromUnsettled(boolean[]
unsettled2)//smallest weight to vertex untouched is ?
    {
        double min = Integer.MAX_VALUE;//min is largest possible
        int node = 0;
        for (int vertex = 0; vertex < numberofvertices;
vertex++)//for all vertexs
        {
            if (unsettled[vertex] == true && key[vertex] <
min)//if the untouched vertex is less than key
            {
                node = vertex;// we go to that piont
                min = key[vertex];//record piont
            }
        }
        return node;
    }

    public void evaluateNeighbours(int evaluationVertex)//look
at surroinding wieghts
    {

        for (int destinationvertex = 0; destinationvertex <
numberofvertices; destinationvertex++)//for all vertexes
        {
            if (settled[destinationvertex] == false)//is it
unsettled
            {
                if
(adjacencyMatrix[evaluationVertex][destinationvertex] !=
INFINITE)//is it an actual value passed in
                {
                    if
(adjacencyMatrix[evaluationVertex][destinationvertex] <
key[destinationvertex])//is it smaller than key at this vertex
                    {
                        key[destinationvertex] =
adjacencyMatrix[evaluationVertex][destinationvertex];//set key
to weight form adjacency matrix
```

```
                        parent[destinationvertex] =
evaluationVertex;
                    }
                    unsettled[destinationvertex] = true;
                }
            }
        }
    }

    public void printMST()
    {
        System.out.println("SOURCE  : DESTINATION = WEIGHT");
        for (int vertex = 2; vertex < numberofvertices;
vertex++)
        {
            System.out.println(parent[vertex] + "\t:\t" + vertex
+"\t=\t"+ adjacencyMatrix[parent[vertex]][vertex]);
        }
    }

    public static void main(String... arg)
    {
      try{  //to set up catching exceptions while still allowing
values to be passed to other methods
        double adjacency_matrix[][];//will be used to hold
weights of graph
        int number_of_vertices;//self explanitory
        int number_of_edges;//self explanitory
        File input = new File("input.txt");//set up file to then
be read into scanner
        Scanner scan = new Scanner(input);// scan file
        number_of_vertices = scan.nextInt();//first int is # of
vertices
        number_of_edges = scan.nextInt(); // second int is # of
edges
        adjacency_matrix = new
double[number_of_vertices][number_of_vertices];// declare
array/matrix

        for(int i = 0; i< number_of_vertices;i++){ //set all
values in array to max possible so unchanged values are never
```

```
picked
                for (int j = 0; j < number_of_vertices; j++){
                  adjacency_matrix[i][j] = Integer.MAX_VALUE;
                }
        }
            for (int i = 1; i <= number_of_edges; i++){
//filling in with actual values from file. every third value
will be weight and read in as double
               int k = scan.nextInt();                      // from
this int
               int j = scan.nextInt();                      // to
this int
               adjacency_matrix[k][j]= scan.nextDouble();// with
this being weight
          }

            Prims prims = new Prims(number_of_vertices);
            prims.primsAlgorithm(adjacency_matrix);
            prims.printMST();
            scan.close();
        } catch (InputMismatchException inputMismatch){
            System.out.println("Wrong Input Format");
        }catch(FileNotFoundException e) {
            System.out.println("No Correct File");
        }


    }
}
```