```
Enter File Name:
C:\Users\Jamie\Desktop\AI Files\file5

6 2 3 5 8 4 7 1 6 : 141
----------------------------------------------------------------------
--------

Enter File Name:
C:\Users\Jamie\Desktop\AI Files\file6

4 18 10 15 6 9 14 5 13 20 2 8 11 17 19 3 12 7 16 1 4 : 448
```

```java
import sun.awt.image.ImageWatched;

import java.util.LinkedList;

//Solves the travelling salesman problem
public class Travel{

    //path class
    private class Path{
        private LinkedList<Integer> list;               //Vertices in path
        private int cost;                               //Cost of path

        //Constructor
        private Path(){
            list = new LinkedList<>();                  //Empty list of vertices
            cost = 0;                                   //Cost is 0
        }

        //Copy constructor
        private Path(Path other){
            list = new LinkedList<>();                  //Empty list of vertices

            for(int i = 0; i < other.list.size(); i++)
                list.addLast(other.list.get(i));        //Copy to list

            cost = other.cost;                          //Copy cost
        }

        //Method finds last vertex of path
        private void add(int vertex, int weight){
            list.addLast(vertex);                       //add vertex at the end
            cost += weight;                             //increment cost
        }

        //Finds last vertex
        private int last(){
            return list.getLast();
        }

        //returns cost
        private int cost(){
            return cost;
        }

        //returns length
        private int size(){
            return list.size();
        }

        //Decide whether path contains a given vertex
        private boolean contains(int vertex){
            for(int i = 0; i < list.size(); i++)         //compare vertex with Vertices of path
                if(list.get(i) == vertex)
                    return true;

            return false;
        }

        //displays path and cost
        private void display(){
            for(int i = 0; i < list.size(); i++)         //print path
                System.out.print(list.get(i)+1 + " " );
            System.out.println(": " + cost);            //cost
        }
    }

    private int size;                                   //Number of vertices of graph
    private int[][] matrix;                             //adjacency matrix of graph
    private int initial;                                //starting/ending vertex

    //Constructor
    public Travel(int vertices, int[][] edges){
        size = vertices;                                //assign vertices

        matrix = new int[size][size];                   //initialize adjacency matrix
        for(int i = 0; i < size; i++)
            for(int j = 0; j < size; j++)
                matrix[i][j] = 0;

        for(int i = 0; i < edges.length; i++){
            int u = edges[i][0];                        //place weights
            int v = edges[i][1];
            int weight = edges[i][2];
            matrix[u][v] = weight;
            matrix[v][u] = weight;
        }

        initial = edges[0][0];                          //Pick a origin
    }

    //Finds shortest cycle
    public void solve(){
        Path shortestPath = null;                       //initialize shortest
        int minimumCost = Integer.MAX_VALUE;            //minimum
```

```java
cost

        LinkedList<Path> list = new LinkedList<>();     //list of paths

        Path path = new Path();                         //Create initial path and add to list
        path.add(initial,0);
        list.addFirst(path);

        //While list has paths
        while (!list.isEmpty()){
            path  = list.removeFirst();                 //Remove first path

            if(complete(path)){                         //if Path has a cycle
                if(path.cost() < minimumCost){
                    minimumCost = path.cost();          //update
                    shortestPath =  path;
                }
            }
            else{
                //generate children of path
                LinkedList<Path> children = generate(path);

                //add children to beginning of list
                for(int i = 0; i < children.size(); i++)
                    list.addFirst(children.get(i));
            }
        }
        if(shortestPath == null)                        //if no cycle
            System.out.println("No Solution");          //no solution
        else
            shortestPath.display();
    }

    //Generates Children
    private LinkedList<Path> generate(Path path){
        LinkedList<Path> children = new LinkedList<>();
        int lastVertex = path.last();

        for(int i = 0; i < size; i++){          //Iterate
            if(matrix[lastVertex][i] != 0){     //if vertex is neighbor
                if(i == initial){
                    if(path.size() == size){    //if path has size vertices
                        Path child = new Path(path);
                        child.add(i, matrix[lastVertex][i]);
                                                //add vertex to path
                        children.addLast(child);//add extend path to child list
                    }
                }
                else{                               //If vertex is not initial
                    if(!path.contains(i)) {      //if vertex is not in path
                        Path child = new Path(path);
                        child.add(i, matrix[lastVertex][i]);
                        children.addLast(child);
                    }
                }
            }
        }
        return children;
    }
    //decides if path is complete
    boolean complete(Path path){
        return path.size() == size + 1; //check path has size + 1
    }
}

```

```java
import java.util.Scanner;
import java.io.*;

public class TravelTester {
    public static void main(String[] args)throws IOException{
        Scanner keyIn = new Scanner(System.in);

        System.out.println("Enter File Name Using 0 as Blanks:");
        String fileName = keyIn.nextLine();

        File file = new File(fileName);
        Scanner sc = new Scanner(file);

        String firstLine[] = sc.nextLine().split(" ");
        int vertices = Integer.parseInt(firstLine[0]);
        int edgesCount = Integer.parseInt(firstLine[1]);
        sc.nextLine();
        System.out.println(vertices + " - " + edgesCount);
        int[][] edges = new int[edgesCount][3];
        for(int i = 0; i < edgesCount; i++){
            String line[] = sc.nextLine().split("\\s+");
            System.out.println(line[0] + " " + line[1] + " " + line[2]
);
            edges[i][0] = Integer.parseInt(line[0])-1;
            edges[i][1] = Integer.parseInt(line[1])-1;
            edges[i][2] = Integer.parseInt(line[2]);
        }
        Travel t = new Travel(vertices, edges);
        t.solve();
    }
}

```