NAME:

Roll No.:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Aim: Concurrent Implementation of travelling salesman problem.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.InputMismatchException;

import java.util.Scanner;

import java.util.Stack;

public class TSPNearestNeighbour

{

private int numberOfNodes;

private Stack<Integer> stack;

public TSPNearestNeighbour()

{

stack = new Stack<Integer>();

}

public void tsp(int adjacencyMatrix[][])

{

numberOfNodes = adjacencyMatrix[1].length - 1;

int[] visited = new int[numberOfNodes + 1];

visited[1] = 1;

stack.push(1);

int element, dst = 0, i;

int min = Integer.MAX\_VALUE;

boolean minFlag = false;

System.out.print(1 + "\t");

while (!stack.isEmpty())

{

element = stack.peek();

i = 1;

min = Integer.MAX\_VALUE;

while (i <= numberOfNodes)

{

if (adjacencyMatrix[element][i] > 1 && visited[i] == 0)

{

if (min > adjacencyMatrix[element][i])

{

min = adjacencyMatrix[element][i];

dst = i;

minFlag = true;

}

}

i++;

}

if (minFlag)

{

visited[dst] = 1;

stack.push(dst);

System.out.print(dst + "\t");

minFlag = false;

continue;

}

stack.pop();

}

}

public static void main(String... arg)

{

int number\_of\_nodes;

Scanner scanner = null;

try

{

System.out.println("Enter the number of nodes in the graph");

scanner = new Scanner(System.in);

number\_of\_nodes = scanner.nextInt();

int adjacency\_matrix[][] = new int[number\_of\_nodes + 1][number\_of\_nodes + 1];

System.out.println("Enter the adjacency matrix");

for (int i = 1; i <= number\_of\_nodes; i++)

{

for (int j = 1; j <= number\_of\_nodes; j++)

{

adjacency\_matrix[i][j] = scanner.nextInt();

}

}

for (int i = 1; i <= number\_of\_nodes; i++)

{

for (int j = 1; j <= number\_of\_nodes; j++)

{

if (adjacency\_matrix[i][j] == 1 && adjacency\_matrix[j][i] == 0)

{

adjacency\_matrix[j][i] = 1;

}

}

}

System.out.println("the citys are visited as follows");

TSPNearestNeighbour tspNearestNeighbour = new TSPNearestNeighbour();

tspNearestNeighbour.tsp(adjacency\_matrix);

} catch (InputMismatchException inputMismatch)

{

System.out.println("Wrong Input format");

}

scanner.close();

}

}

\*/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Output:

C:\Users\K!SH>cd\

C:\>javac TSPNearestNeighbour.java

C:\>javac TSPNearestNeighbour

error: Class names, 'TSPNearestNeighbour', are only accepted if annotation proce

ssing is explicitly requested

1 error

C:\>javac TSPNearestNeighbour.java

C:\>java TSPNearestNeighbour

Enter the number of nodes in the graph

5

Enter the adjacency matrix

0 11 10 9 6

8 0 7 3 4

8 4 0 4 8

11 10 5 0 5

6 9 5 5 0

the citys are visited as follows

1 5 3 2 4

C:\>

/\*