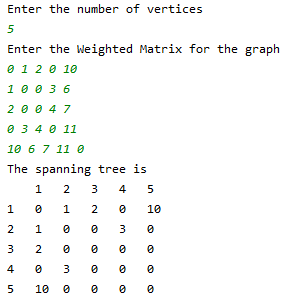
**Листинг практической работы № 13**

package ru.mirea.ikbo20.AOD.pr13;  
import java.util.Comparator;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Scanner;  
import java.util.Stack;  
  
public class KruskalAlgorithm  
{  
 private List<Edge> edges;  
 private int numberOfVertices;  
 public static final int *MAX\_VALUE* = 999;  
 private int visited[];  
 private int spanning\_tree[][];  
  
 public KruskalAlgorithm(int numberOfVertices)  
 {  
 this.numberOfVertices = numberOfVertices;  
 edges = new LinkedList<Edge>();  
 visited = new int[this.numberOfVertices + 1];  
 spanning\_tree = new int[numberOfVertices + 1][numberOfVertices + 1];  
 }  
  
 public void kruskalAlgorithm(int[][] adjacencyMatrix)  
 {  
 boolean finished = false;  
 for (int source = 1; source <= numberOfVertices; source++)  
 {  
 for (int destination = 1; destination <= numberOfVertices; destination++)  
 {  
 if (adjacencyMatrix[source][destination] != *MAX\_VALUE* && source != destination)  
 {  
 Edge edge = new Edge();  
 edge.sourcevertex = source;  
 edge.destinationvertex = destination;  
 edge.weight = adjacencyMatrix[source][destination];  
 adjacencyMatrix[destination][source] = *MAX\_VALUE*;  
 edges.add(edge);  
 }  
 }  
 }  
 edges.sort(new EdgeComparator());  
 CheckCycle checkCycle = new CheckCycle();  
 for (Edge edge : edges)  
 {  
 spanning\_tree[edge.sourcevertex][edge.destinationvertex] = edge.weight;  
 spanning\_tree[edge.destinationvertex][edge.sourcevertex] = edge.weight;  
 if (checkCycle.checkCycle(spanning\_tree, edge.sourcevertex))  
 {  
 spanning\_tree[edge.sourcevertex][edge.destinationvertex] = 0;  
 spanning\_tree[edge.destinationvertex][edge.sourcevertex] = 0;  
 edge.weight = -1;  
 continue;  
 }  
 visited[edge.sourcevertex] = 1;  
 visited[edge.destinationvertex] = 1;  
 for (int j : visited) {  
 if (j == 0) {  
 finished = false;  
 break;  
 } else {  
 finished = true;  
 }  
 }  
 if (finished)  
 break;  
 }  
 System.*out*.println("The spanning tree is ");  
 for (int i = 1; i <= numberOfVertices; i++)  
 System.*out*.print("\t" + i);  
 System.*out*.println();  
 for (int source = 1; source <= numberOfVertices; source++)  
 {  
 System.*out*.print(source + "\t");  
 for (int destination = 1; destination <= numberOfVertices; destination++)  
 {  
 System.*out*.print(spanning\_tree[source][destination] + "\t");  
 }  
 System.*out*.println();  
 }  
 }  
  
 public static void main(String... arg)  
 {  
 int[][] adjacency\_matrix;  
 int number\_of\_vertices;  
  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.println("Enter the number of vertices");  
 number\_of\_vertices = scan.nextInt();  
 adjacency\_matrix = new int[number\_of\_vertices + 1][number\_of\_vertices + 1];  
  
 System.*out*.println("Enter the Weighted Matrix for the graph");  
 for (int i = 1; i <= number\_of\_vertices; i++)  
 {  
 for (int j = 1; j <= number\_of\_vertices; j++)  
 {  
 adjacency\_matrix[i][j] = scan.nextInt();  
 if (i == j)  
 {  
 adjacency\_matrix[i][j] = 0;  
 continue;  
 }  
 if (adjacency\_matrix[i][j] == 0)  
 {  
 adjacency\_matrix[i][j] = *MAX\_VALUE*;  
 }  
 }  
 }  
 KruskalAlgorithm kruskalAlgorithm = new KruskalAlgorithm(number\_of\_vertices);  
 kruskalAlgorithm.kruskalAlgorithm(adjacency\_matrix);  
 scan.close();  
 }  
}  
  
class Edge  
{  
 int sourcevertex;  
 int destinationvertex;  
 int weight;  
}  
  
class EdgeComparator implements Comparator<Edge>  
{  
 @Override  
 public int compare(Edge edge1, Edge edge2)  
 {  
 if (edge1.weight < edge2.weight)  
 return -1;  
 if (edge1.weight > edge2.weight)  
 return 1;  
 return 0;  
 }  
}  
  
class CheckCycle  
{  
 private Stack<Integer> stack;  
  
 public CheckCycle()  
 {  
 stack = new Stack<Integer>();  
 }  
  
 public boolean checkCycle(int[][] adjacency\_matrix, int source)  
 {  
 boolean cyclepresent = false;  
 int number\_of\_nodes = adjacency\_matrix[source].length - 1;  
  
 int[][] adjacencyMatrix = new int[number\_of\_nodes + 1][number\_of\_nodes + 1];  
 for (int sourcevertex = 1; sourcevertex <= number\_of\_nodes; sourcevertex++)  
 {  
 System.*arraycopy*(adjacency\_matrix[sourcevertex], 1,  
 adjacencyMatrix[sourcevertex], 1, number\_of\_nodes);  
 }  
  
 int[] visited = new int[number\_of\_nodes + 1];  
 int element = source;  
 int i = source;  
 visited[source] = 1;  
 stack.push(source);  
  
 while (!stack.isEmpty())  
 {  
 element = stack.peek();  
 i = element;  
 while (i <= number\_of\_nodes)  
 {  
 if (adjacencyMatrix[element][i] >= 1 && visited[i] == 1)  
 {  
 if (stack.contains(i))  
 {  
 cyclepresent = true;  
 return cyclepresent;  
 }  
 }  
 if (adjacencyMatrix[element][i] >= 1 && visited[i] == 0)  
 {  
 stack.push(i);  
 visited[i] = 1;  
 adjacencyMatrix[element][i] = 0;*// mark as labelled;* adjacencyMatrix[i][element] = 0;  
 element = i;  
 i = 1;  
 continue;  
 }  
 i++;  
 }  
 stack.pop();  
 }  
 return cyclepresent;  
 }  
}

**Демонстрация работы программы**

****