Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра: Информационные технологии и автоматизированные системы

Дисциплина: «Математические методы теории систем»

Лабораторная работа № 1

на тему: «Модели структуры систем с использованием теории графов.

Топологическая декомпозиция»

Выполнил: студент группы АСУ8-23-1м

Зиятдинов Расиль Артурович

Проверил: ст. преп. кафедры ИТАС

Тютюных Артём Александрович

Пермь 2023

**ЗАДАНИЕ**

**Задача 1.** Дано описание системы на рисунке 1.

1. Выполнить матричное и множественное описание графа топологии системы.

2. Выполнить топологическую декомпозицию системы.

3. Разработать алгоритм решения задачи топологической декомпозиции на одном из языков программирования. Привести результаты работы программы.

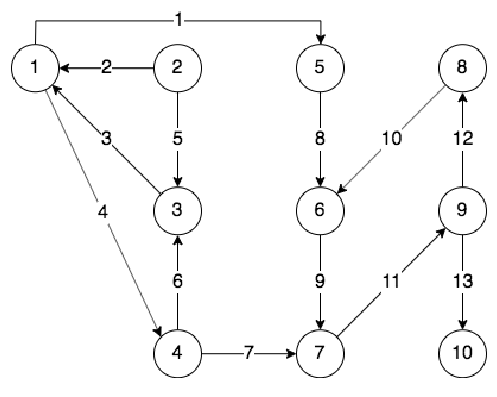


Рисунок 1 – Описание системы

**Задача 2.** Выполнить топологическую декомпозицию системы одного из вариантов (рисунок 2), используя разработанную программу. Привести результаты работы программы.

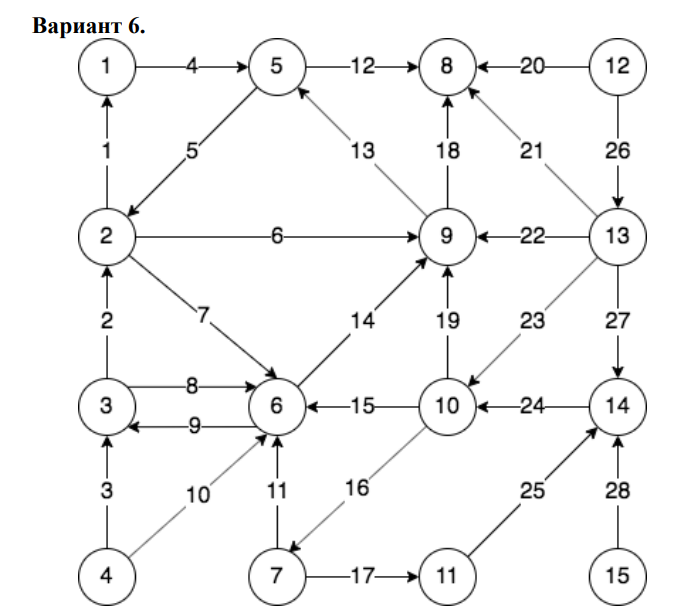


Рисунок 2 – Задание для варианта 6

**ВЫПОЛНЕНИЕ РАБОТЫ**

**Задача 1:**

Матричное описание графа

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j  i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 |  |  |  | 1 | 1 |  |  |  |  |  |
| 2 | 1 |  | 1 |  |  |  |  |  |  |  |
| 3 | 1 |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 1 |  |  |  | 1 |  |  |  |
| 5 |  |  |  |  |  | 1 |  |  |  |  |
| 6 |  |  |  |  |  |  | 1 |  |  |  |
| 7 |  |  |  |  |  |  |  |  | 1 |  |
| 8 |  |  |  |  |  | 1 |  |  |  |  |
| 9 |  |  |  |  |  |  |  | 1 |  | 1 |
| 10 |  |  |  |  |  |  |  |  |  |  |

Множественное описание графа

G(1) = 4, 5

G(2) = 1, 3

G(3) = 1

G(4) = 3, 7

G(5) = 6

G(6) = 7

G(7) = 9

G(8) = 6

G(9) = 8, 10

G(10) = 0

Топологическая декомпозиция графа

R(1) = 1, 3, 4, 5, 6, 7, 8, 9, 10

Q(1) = 1, 2, 3, 4

V(1) = 1, 3, 4

R(2) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Q(2) = 2

V(2) = 2

R(5) = 5, 6, 7, 8, 9, 10

Q(5) = 1, 2, 3, 4, 5

V(5) = 5

R(6) = 6, 7, 8, 9, 10

Q(6) = 1, 2, 3, 4, 5, 6, 7, 8, 9

V(6) = 6, 7, 8, 9

R(10) = 10

Q(10) = 10

V(10) = 10

Результат декомпозиции

G(1) = 5, 6

G(2) = 1

G(5) = 6

G(6) = 10

G(10) = 0

На рисунке 3 отмечены сильно связанные подграфы.

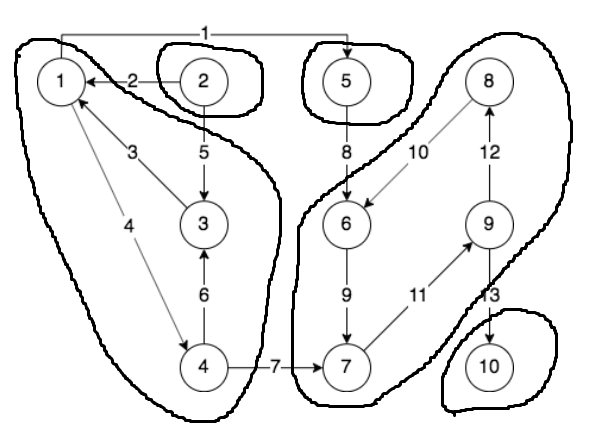


Рисунок 3 – Вид сильно связных подграфов

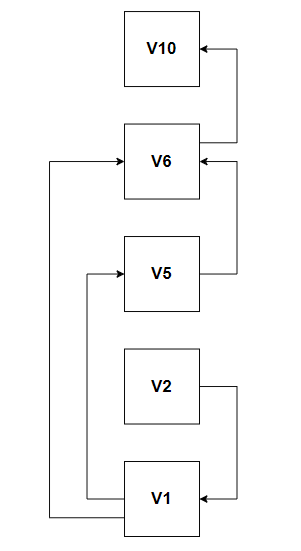


Рисунок 4 – Результат декомпозиции исходного графа

Результат работы программы (листинг в приложении А) представлен на рисунке 5. Входные данные поступают в формате матрицы смежности из текстового файла. Сначала программа приводит множественное описание входного графа, далее производит топологическую декомпозицию, выводит список сильно связных подграфов и множественное описание результата.

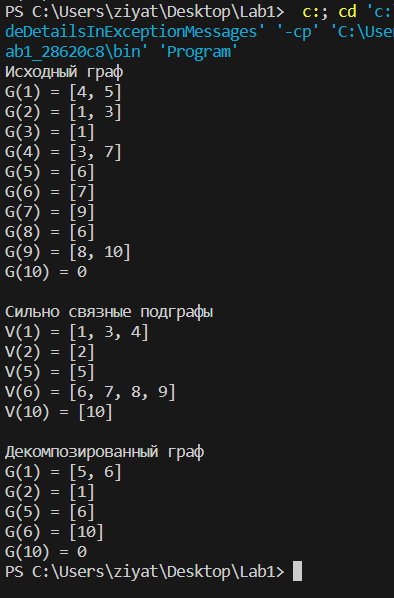


Рисунок 5 – Результат работы программы для задачи 1

**Задача 2:**

Результат работы программы для варианта 6 представлен на рисунке 6.

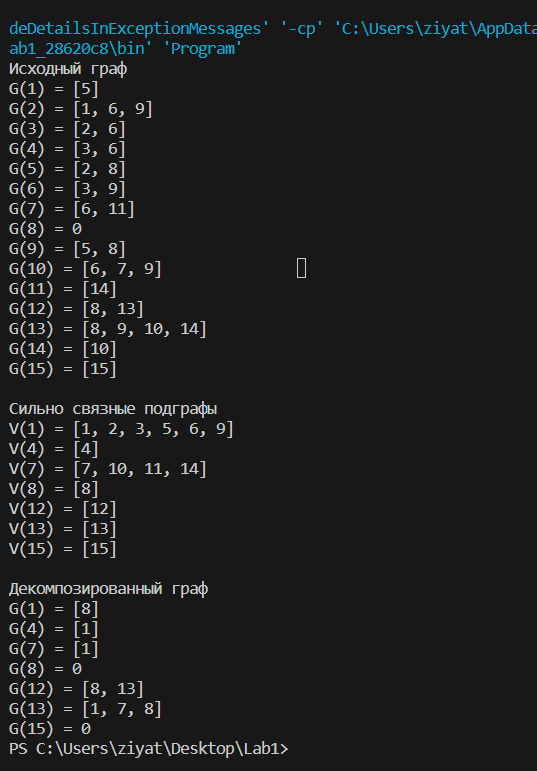


Рисунок 6 – Результат работы программы для задачи 2

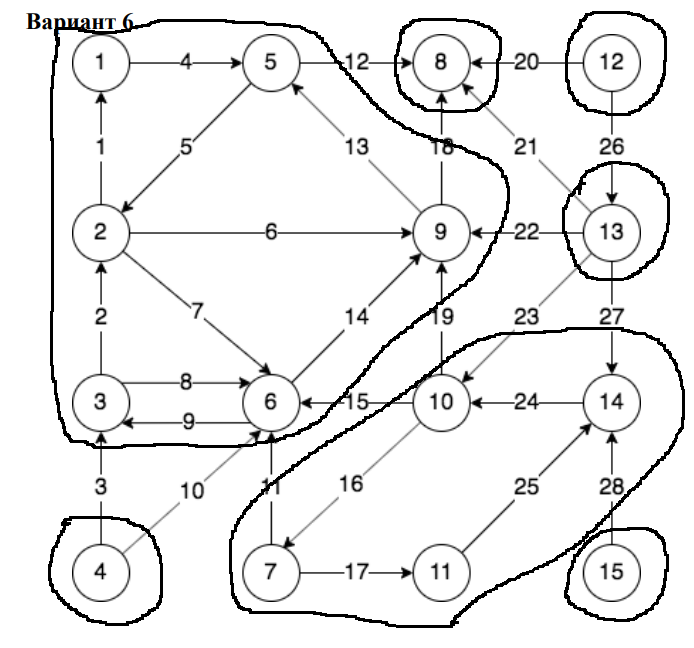


Рисунок 7 – Вид сильно связных подграфов

**ПРИЛОЖЕНИЕ А**

Листинг 1 – Код программы на языке программирования Java

**import java.io.IOException;**

**import java.nio.file.Files;**

**import java.nio.file.Paths;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.Collections;**

**import java.util.List;**

**import java.util.stream.Collectors;**

**class Program {**

**private static Graph readFromFile(String path) {**

**try {**

**List<String> lines = Files.readAllLines(Paths.get(path));**

**int[] vertices = new int[lines.size()];**

**List<Edge> edges = new ArrayList<>();**

**for (int i = 0; i < vertices.length; i++) {**

**vertices[i] = i + 1;**

**char[] line = lines.get(i).toCharArray();**

**for (int j = 0; j < line.length; j++) {**

**if (line[j] == '1') {**

**edges.add(new Edge(i + 1, j + 1));**

**}**

**}**

**}**

**return new Graph(vertices, edges);**

**} catch (IOException e) {**

**e.printStackTrace();**

**return null;**

**}**

**}**

**public static void main(String[] args) {**

**Graph graph = readFromFile("graph.txt");**

**graph.print("Исходный граф");**

**System.out.println();**

**Graph decomposedGraph = graph.decompose();**

**System.out.println();**

**decomposedGraph.print("Декомпозированный граф");**

**}**

**}**

**class Edge {**

**private final int source;**

**private final int dest;**

**public Edge(int source, int dest) {**

**this.source = source;**

**this.dest = dest;**

**}**

**public int getSource() {**

**return source;**

**}**

**public int getDest() {**

**return dest;**

**}**

**@Override**

**public boolean equals(Object o) {**

**if (o == this) {**

**return true;**

**}**

**if (!(o instanceof Edge)) {**

**return false;**

**}**

**Edge e = (Edge) o;**

**return Integer.compare(source, e.getSource()) == 0**

**&& Integer.compare(dest, e.getDest()) == 0;**

**}**

**}**

**class Graph {**

**private class Subgraph {**

**private final int index;**

**private final List<Integer> vertices;**

**public Subgraph(List<Integer> \_vertices) {**

**index = \_vertices.get(0);**

**vertices = \_vertices;**

**}**

**public int getIndex() {**

**return index;**

**}**

**public List<Integer> getVertices() {**

**Collections.sort(vertices);**

**return vertices;**

**}**

**}**

**private final List<Edge> edges;**

**private final List<Integer> vertices;**

**public Graph(int[] vertices, List<Edge> edges) {**

**this.vertices = new ArrayList<>();**

**for (int vertex : vertices) {**

**this.vertices.add(vertex);**

**}**

**this.edges = edges;**

**}**

**public void print(String text) {**

**System.out.println(text);**

**for (int vert : vertices) {**

**List<Integer> outSeq = new ArrayList<>();**

**for (Edge e : edges) {**

**if (e.getSource() == vert) {**

**outSeq.add(e.getDest());**

**}**

**}**

**Collections.sort(outSeq);**

**System.out.printf("G(%d) = %s%n", vert, outSeq.isEmpty() ? "0" : String.join(", ", outSeq.toString()));**

**}**

**}  
    public Graph decompose() {**

**List<Subgraph> subgraphes = new ArrayList<>();**

**List<Integer> accessVerts = vertices;**

**Subgraph currentSubgraph;**

**System.out.println("Сильно связные подграфы");**

**while (!accessVerts.isEmpty()) {**

**var vertex = accessVerts.get(0);**

**var R = findR(vertex);**

**var Q = findQ(vertex);**

**currentSubgraph = new Subgraph(R.stream()**

**.distinct().filter(Q::contains).collect(Collectors.toList()));**

**System.out.println("V(" + currentSubgraph.getIndex() + ") = " + Arrays.toString(currentSubgraph.getVertices().toArray()));**

**if (currentSubgraph.getVertices().isEmpty()) {**

**var lastSubgraph = new Subgraph(accessVerts);**

**subgraphes.add(lastSubgraph);**

**break;**

**}**

**subgraphes.add(currentSubgraph);**

**accessVerts.removeAll(currentSubgraph.getVertices());**

**}**

**List<Edge> newEdges = new ArrayList<Edge>();**

**int[] newVerts = new int[subgraphes.size()];**

**int i = 0;**

**for (Subgraph subgraph : subgraphes) {**

**newVerts[i] = subgraph.getIndex();**

**i++;**

**List<Edge> outEdges = new ArrayList<>();**

**for (Edge e : edges) {**

**if (subgraph.vertices.contains(e.getSource()) && !subgraph.vertices.contains(e.getDest())) {**

**outEdges.add(e);**

**}**

**}**

**var otherSubgraphs = subgraphes.stream()**

**.filter(j -> subgraph != j)**

**.collect(Collectors.toList());**

**for (Edge edge : outEdges) {**

**int inIndex = -1;**

**for (Subgraph s : otherSubgraphs) {**

**if (s.vertices.contains(edge.getDest())) {**

**inIndex = s.getIndex();**

**break;**

**}**

**}**

**if (inIndex != -1) {**

**var newEdge = new Edge(subgraph.getIndex(), inIndex);**

**if (!newEdges.contains(newEdge)) {**

**newEdges.add(newEdge);**

**}**

**}**

**}**

**}**

**return new Graph(newVerts, newEdges.stream()**

**.distinct()**

**.collect(Collectors.toList()));**

**}**

**private List<Integer> findR(int start) {**

**List<Integer> resultSet = new ArrayList<Integer>() {**

**{**

**add(start);**

**}**

**};**

**List<Integer> achieveVerts = new ArrayList<Integer>();**

**for (int i = 0; i < resultSet.size(); i++) {**

**List<Integer> addedVerts = new ArrayList<Integer>();**

**do {**

**for (Edge e : edges) {**

**if (e.getSource() == resultSet.get(i)) {**

**achieveVerts.add(e.getDest());**

**}**

**}**

**achieveVerts.removeAll(addedVerts);**

**resultSet.addAll(achieveVerts);**

**resultSet = resultSet.stream().distinct().collect(Collectors.toList());**

**addedVerts.addAll(achieveVerts);**

**addedVerts = addedVerts.stream().distinct().collect(Collectors.toList());**

**} while (!achieveVerts.isEmpty());**

**}**

**return resultSet.stream().distinct().collect(Collectors.toList());**

**}**

**private List<Integer> findQ(int end) {**

**List<Integer> resultSet = new ArrayList<Integer>() {**

**{**

**add(end);**

**}**

**};**

**List<Integer> achieveVerts = new ArrayList<Integer>();**

**for (int i = 0; i < resultSet.size(); i++) {**

**List<Integer> addedVerts = new ArrayList<Integer>();**

**do {**

**for (Edge e : edges) {**

**if (e.getDest() == resultSet.get(i)) {**

**achieveVerts.add(e.getSource());**

**}**

**}**

**achieveVerts.removeAll(addedVerts);**

**resultSet.addAll(achieveVerts);**

**resultSet = resultSet.stream().distinct().collect(Collectors.toList());**

**addedVerts.addAll(achieveVerts);**

**addedVerts = addedVerts.stream().distinct().collect(Collectors.toList());**

**} while (!achieveVerts.isEmpty());**

**}**

**return resultSet.stream().distinct().collect(Collectors.toList());**

**}**

**}**