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

Шеретов Марк Алексеевич

Проверил: ст. преп. кафедры ИТАС

Тютюных Артём Александрович

Пермь 2023

**ЗАДАНИЕ**

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

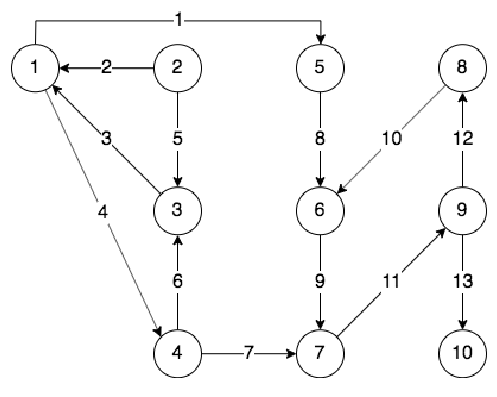


Рисунок 1 – описание системы

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.  
**Задача 2.** Выполнить топологическую декомпозицию одной системы из назначенного варианта (рис. 2), используя разработанную программу. Привести результаты работы программы.

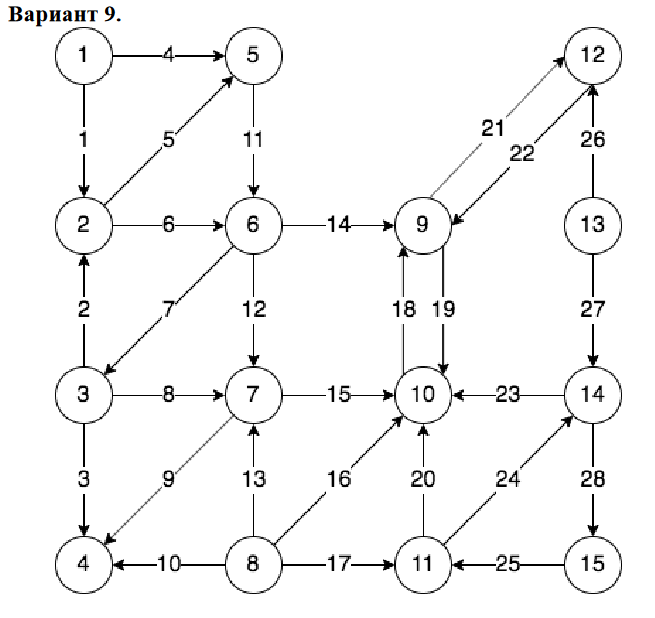


Рисунок 2 – задание для варианта 9

**ВЫПОЛНЕНИЕ РАБОТЫ**

**Задача 1:**

Матрица смежности графа:

0001100000

1010000000

1000000000

0010001000

0000010000

0000001000

0000000010

0000010000

0000000101

0000000000

Множественное представление 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:

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

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

V(1) = 1, 3, 4;

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

Q(2) = 2;

V(2) = 2;

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

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

V(5) = 5;

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

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

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

R(10) = 10;

Q(10) = 10;

V(10) = 10;

Ответ:

G(1) = 5, 6;

G(2) = 1;

G(5) = 6;

G(6) = 10;

G(10) = 0;

На рисунке 3 отмечены сильно связанные подграфы.

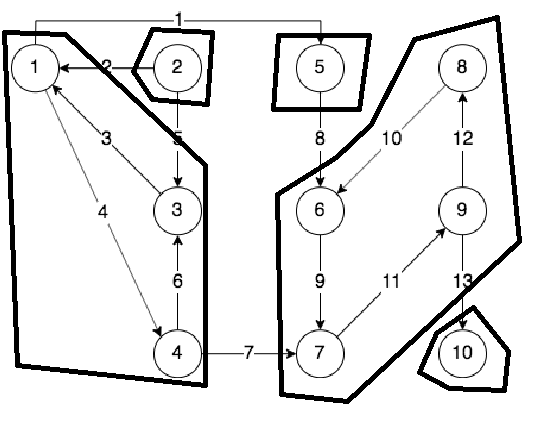


Рисунок 3 – граф с отмеченными сильно связанными подграфами

Результат работы программы (листинг в приложении А) представлен на рисунке 4. Входные данные поступают в формате матрицы смежности из текстового файла. Сначала программа приводит множественное описание входного графа, далее производит топологическую декомпозицию и выводит множественное описание результата.

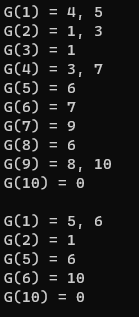


Рисунок 4 – результат работы программы для задачи 1

**Задача 2:**

Результат работы программы для системы варианта 9 представлен на рисунке 5.

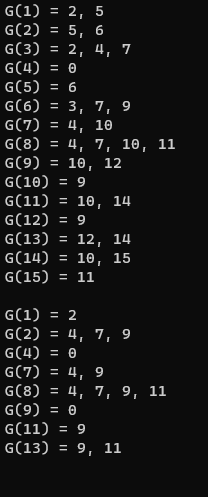


Рисунок 5 – результат работы программы для задачи 2

**ПРИЛОЖЕНИЕ А**

Листинг 1 – код программы на языке программирования C#

using System.Collections.Immutable;

using System.IO;

internal class Program

{

private static void Main(string[] args)

{

var graph = ReadGraphFromFile("input.txt");

graph.Print();

var decomposedGraph = graph.TopologicalDecompose();

decomposedGraph.Print();

}

private static Graph ReadGraphFromFile(string path)

{

var lines = File.ReadAllLines(path);

var verticles = new int[lines.Length];

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

for (int i = 0; i < verticles.Length; i++)

{

verticles[i] = i + 1;

var line = lines[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(verticles, edges);

}

}

internal struct Edge

{

public Edge(int outV, int inV)

{

this.outV = outV;

this.inV = inV;

}

public int outV { get; init; }

public int inV { get; init; }

}

internal class Graph

{

public void Print()

{

foreach (var vert in Verticles)

{

var outSeq = Edges.Where(e => e.outV == vert).Select(e => e.inV);

Console.WriteLine($"G({vert}) = {(outSeq.Any() ? string.Join(", ", outSeq) : 0)}");

}

Console.ReadLine();

}

private class Subgraph

{

public int Index => Verticles.First();

public IEnumerable<int> Verticles { get; init; }

public Subgraph(IEnumerable<int> verticles)

{

Verticles = verticles;

}

}

public Graph(IEnumerable<int> verticles, IEnumerable<Edge> edges)

{

Verticles = verticles.ToImmutableSortedSet();

Edges = edges;

}

public IEnumerable<Edge> Edges { get; private set; }

public IEnumerable<int> Verticles { get; private set; }

public Graph TopologicalDecompose()

{

var Subgraphes = new List<Subgraph>();

var availableVerticles = Verticles;

Subgraph currentSubgraph;

while (availableVerticles.Any())

{

var verticle = availableVerticles.First();

var R = GetR(verticle);

var Q = GetQ(verticle);

currentSubgraph = new Subgraph(R.Intersect(Q));

if (!currentSubgraph.Verticles.Any())

{

var lastSubgraph = new Subgraph(availableVerticles);

Subgraphes.Add(lastSubgraph);

break;

}

Subgraphes.Add(currentSubgraph);

availableVerticles = availableVerticles.Except(currentSubgraph.Verticles);

}

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

foreach (var subgraph in Subgraphes)

{

var outEdges = Edges.Where(e => subgraph.Verticles.Contains(e.outV) && !subgraph.Verticles.Contains(e.inV));

var otherSubgraphs = Subgraphes.Except(new Subgraph[] { subgraph });

foreach (var edge in outEdges)

{

var inIndex = otherSubgraphs.FirstOrDefault(s => s.Verticles.Contains(edge.inV))?.Index;

if (inIndex.HasValue)

{

var newEdge = new Edge(subgraph.Index, inIndex.Value);

if (!newEdges.Contains(newEdge))

{

newEdges.Add(newEdge);

}

}

}

}

return new Graph(Subgraphes.Select(s => s.Index), newEdges);

}

private IEnumerable<int> GetR(int v)

{

var res = new List<int>() { v };

IEnumerable<int> achievable;

for (int i = 0; i < res.Count; i++)

{

List<int> alreadyAdded = new List<int>();

do

{

achievable = Edges.Where(edge => edge.outV == res[i]).Select(edge => edge.inV).Except(alreadyAdded).ToArray();

res = res.Union(achievable).ToList();

alreadyAdded.AddRange(achievable);

} while (achievable.Any());

}

return res.Distinct();

}

private IEnumerable<int> GetQ(int v)

{

Stack<int> stack = new Stack<int>();

List<int> res = new List<int>();

stack.Push(v);

while (stack.TryPop(out int vert))

{

foreach (var outVert in Edges.Where(e => e.inV == vert).Select(e => e.outV).Except(res))

{

if (!stack.Contains(outVert))

{

stack.Push(outVert);

res.Add(outVert);

}

}

}

res.Add(v);

return res;

}

}