Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский государственный технологический университет»

(ФГБОУ ВО «КубГТУ»)

Институткомпьютерных систем и информационной безопасности

Кафедраинформационных систем и программирования

Направлениеподготовки09.03.04 Программная инженерия

ПрофильПроектирование и разработка программного обеспечения

**Отчет по лабораторной работе №5**

**По дисциплине «Методы оптимизации»**

**Тема: Сетевые задачи**

Выполнил: Агниев Сергей Владимирович

Принял: Попова Ольга Борисовна

Краснодар 2024

**Цель работы:** изучение принципов составления математической модели для сетевых задач, получение навыков работы решения задач, а также применение данного метода на практике.

**Порядок выполнения:**

1. Изучить теоретические сведения.

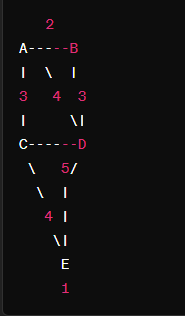
В ходе выполнения лабораторной работы были изучении теоретические сведения по работе алгоритмов на графах, в частности таких как алгоритм Дейкстры, алгоритм А\* и потоковые алгоритмы.

2. Разработать программу решающую поставленную задачу в общем виде.

В ходе выполнения лабораторной работы была разработана программа в виде десктоп-приложения, решающая заданную сетевую задачу в общем виде с заданными параметрами.

Минимальное остовное дерево (Minimum Spanning Tree, MST) в графе — это подграф, содержащий все вершины исходного графа, образующий дерево и имеющий минимальную сумму весов рёбер. Для построения минимального остовного дерева существует несколько алгоритмов, наиболее известные из которых — алгоритм Прима.

Алгоритм Прима начинает с произвольной начальной вершины и постепенно добавляет к остову ребро минимального веса, инцидентное уже выбранным вершинам, пока не будут посещены все вершины.



Это остовное дерево будет содержать все вершины (A, B, C, D, E) и иметь минимальную сумму весов рёбер.

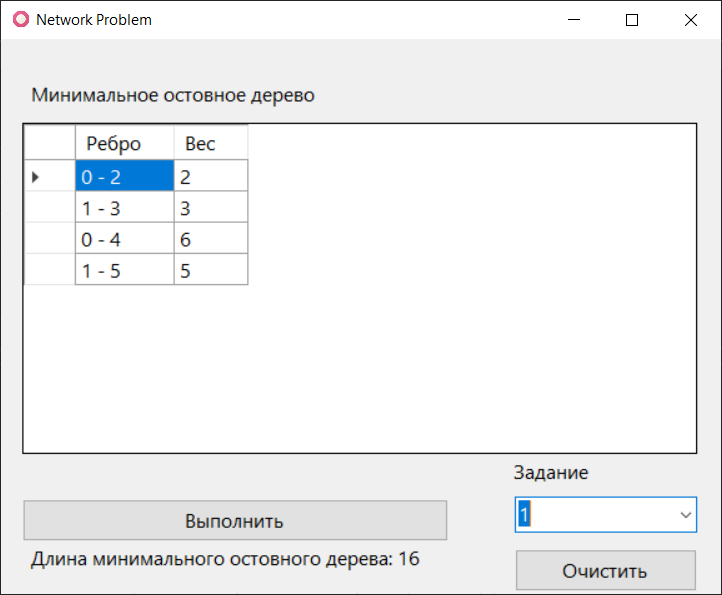


Рисунок 2 – Результат работы программы по решению задания №1

1. Нахождение кратчайшего пути (Shortest Path Problem):

- Задача: Для данного взвешенного ориентированного графа G и начальной вершины s найти кратчайшие пути от вершины s до всех остальных вершин в графе.

- Алгоритмы:

- Алгоритм Дейкстры: Позволяет найти кратчайшие пути от одной вершины до всех остальных в графе с неотрицательными весами рёбер.

- Алгоритм Флойда-Уоршелла: Позволяет найти кратчайшие пути между всеми парами вершин в графе (может работать с отрицательными весами, но не обрабатывает циклы отрицательного веса).

- Алгоритм A\*: Эффективный алгоритм поиска пути, который использует эвристику для ускорения поиска пути.

2. Задача о максимальном потоке (Maximum Flow Problem):

- Задача: Нахождение максимального объема потока от источника к стоку в сети, где каждое ребро имеет пропускную способность.

- Алгоритмы:

- Алгоритм Форда-Фалкерсона: Итеративно находит увеличивающий путь и увеличивает поток по этому пути.

- Алгоритм Эдмондса-Карпа: Оптимизированная версия алгоритма Форда-Фалкерсона, использующая поиск увеличивающего пути с помощью BFS.

- Алгоритм Диница: Ещё более оптимизированная версия алгоритма, использующая разные структуры данных для ускорения поиска пути.

3. Нахождение потока наименьшей стоимости (Minimum Cost Flow Problem):

- Задача: Найти поток с минимальной стоимостью в сети, где каждое ребро имеет как пропускную способность, так и стоимость перевозки единицы потока.

- Алгоритмы:

- Алгоритм Форда-Фалкерсона с потенциалами: Расширяет алгоритм Форда-Фалкерсона, учитывая потенциалы вершин для оптимизации поиска увеличивающего пути.

- Алгоритм проталкивания предпотока с предварительным вычислением потенциалов: Решает задачу о минимальном потоке с помощью метода проталкивания предпотока, используя предварительно вычисленные потенциалы для оптимизации.

Каждая из этих задач имеет свои особенности и применяется в различных областях, таких как транспортная логистика, сетевое планирование, анализ социальных сетей и другие.

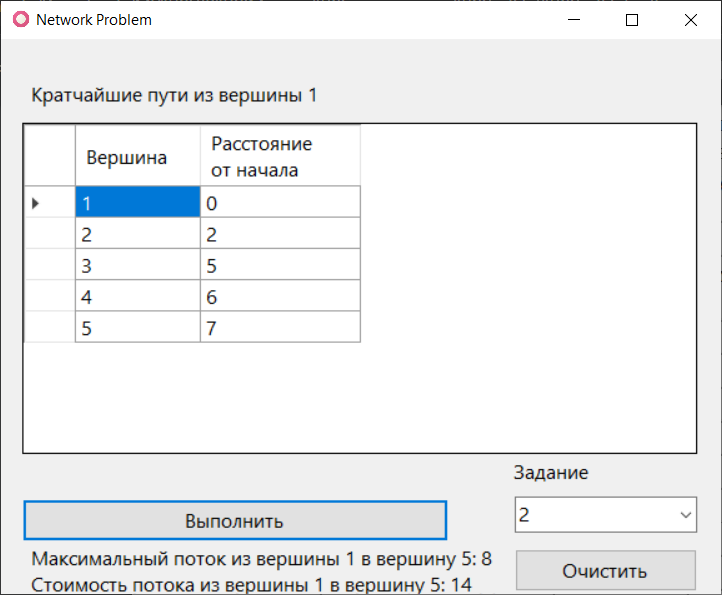


Рисунок 3 – Результат работы программы по решению задания №2

Для решения без программы можно воспользоваться обходом графа в глубину или в ширину. В данном случае, можно начать обход с любой вершины и проверить, что все вершины достижимы из выбранной. Если это так, то граф считается связным.

1. Обход в глубину (DFS):

- Начнем обход с вершины 0.

- Посещаем каждую смежную вершину и помечаем её как посещенную.

- Рекурсивно повторяем этот процесс для каждой смежной вершины, которая еще не была посещена.

- Если после обхода всех вершин все они окажутся посещенными, граф считается связным.

2. Обход в ширину (BFS):

- Начнем обход с вершины 0.

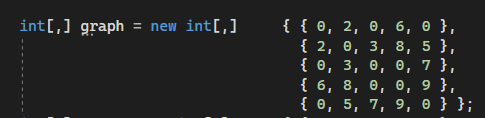
- Помещаем эту вершину в очередь.

- Пока очередь не пуста, извлекаем из неё вершину и помечаем её как посещенную.

- Для каждой смежной вершины, которая еще не была посещена, добавляем её в очередь.

- Если после обхода всех вершин все они окажутся посещенными, граф считается связным.

При обоих подходах время выполнения составляет O(V + E), где V - количество вершин, E - количество рёбер в графе.



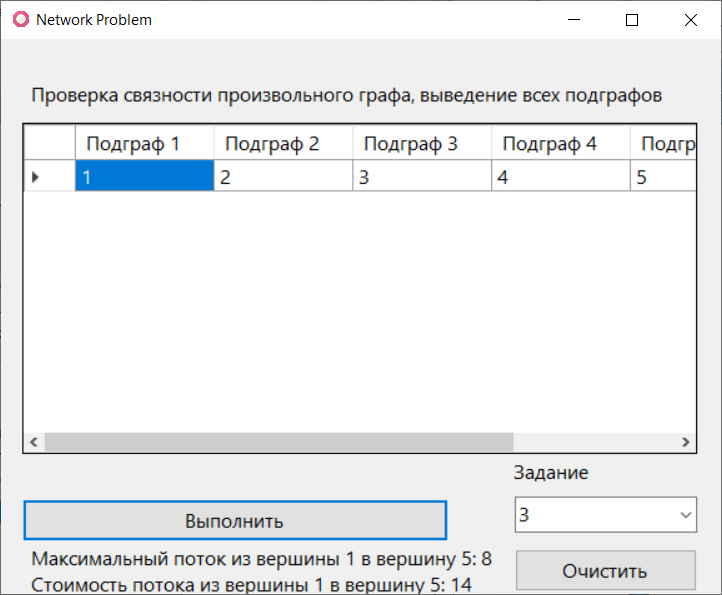


Рисунок 4 – Результат работы программы по решению задания №3

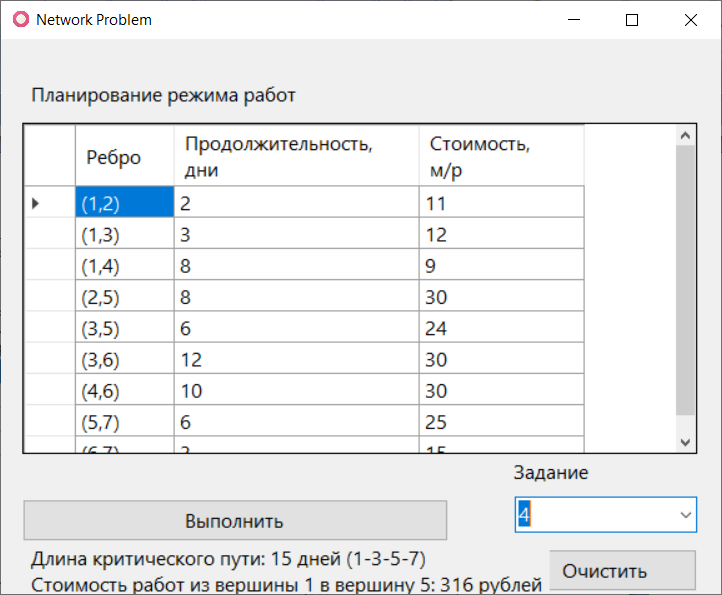


Рисунок 4 – Результат работы программы по решению задания №4

**Листинг:**

MainScreen.cs:

using SimplexMethod.Services;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace SimplexMethod

{

public partial class MainScreen : System.Windows.Forms.Form

{

int[,] graph = new int[,] { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

int[,] cost = new int[,] { { 0, 1, 0, 2, 0 },

{ 3, 0, 2, 0, 1 },

{ 0, 4, 0, 0, 2 },

{ 1, 4, 0, 0, 0 },

{ 0, 2, 1, 3, 0 } };

public MainScreen()

{

InitializeComponent();

}

int StartPoints = 0;

int EndPoints = 0;

private string GetMatString(int[,] Mat)

{

string str = "";

for (int i = 0; i < Mat.GetLength(0); i++)

{

for (int j = 0; j < Mat.GetLength(1); j++)

{

str += $"{Mat[i, j]}\t";

}

str += '\n';

}

return str;

}

void fillConstraintsGrid()

{

GraphGridView.Rows.Clear();

GraphGridView.ColumnCount = EndPoints + 2;

GraphGridView.RowHeadersVisible = false;

for (int i = 0; i < EndPoints + 2; i++)

{

GraphGridView.Columns[i].Width = 50;

GraphGridView.Columns[i].DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;

if (i == 0)

{

GraphGridView.Columns[i].Name = "Поставщик";

}

else if (i < EndPoints+1)

{

GraphGridView.Columns[i].Name = "B" + (i).ToString();

} else if (i == EndPoints+1)

{

GraphGridView.Columns[i].Name = "Запас";

}

}

for (int i = 0; i < StartPoints+1; i++)

{

if (i < StartPoints)

{

string[] row = new string[StartPoints + 1];

row[0] = "A" + (i + 1).ToString();

GraphGridView.Rows.Add(row);

GraphGridView.Rows[i].Height = 20;

}

else if (i == StartPoints)

{

string[] row = new string[StartPoints + 1];

row[0] = "Потребность";

GraphGridView.Rows.Add(row);

GraphGridView.Rows[i].Height = 20;

}

}

}

void TaskOne()

{

MST mst = new MST(graph);

List<Tuple<string, string>> res = mst.primMST();

GraphGridView.Columns.Clear();

GraphGridView.ColumnCount = 2;

GraphGridView.Columns[0].Name = res[0].Item1.ToString(); GraphGridView.Columns[1].Name = res[0].Item2.ToString();

int sum = 0;

for (int i = 1; i < res.Count; i++)

{

string[] row = new string[2];

row[0] = res[i].Item1.ToString(); row[1] = res[i].Item2.ToString();

GraphGridView.Rows.Add(row);

GraphGridView.Rows[i - 1].Height = 20;

sum += Convert.ToInt32(row[1]);

}

resultsLabel.Text = "Длина минимального остовного дерева: " + sum.ToString();

}

void TaskTwo()

{

GFG mst = new GFG(graph);

List<Tuple<string, string>> res = mst.dijkstra(0);

GraphGridView.Columns.Clear();

GraphGridView.ColumnCount = 2;

GraphGridView.Columns[0].Name = res[0].Item1.ToString(); GraphGridView.Columns[1].Name = res[0].Item2.ToString();

for (int i = 1; i < res.Count; i++)

{

string[] row = new string[2];

row[0] = res[i].Item1.ToString(); row[1] = res[i].Item2.ToString();

GraphGridView.Rows.Add(row);

GraphGridView.Rows[i - 1].Height = 20;

}

MaxFlow maxFlow = new MaxFlow(graph);

MinCostMaxFlow minCostMaxFlow = new MinCostMaxFlow(graph, cost);

minCostMaxFlow.getMaxFlow(0, 4).Deconstruct(out int totalFlow, out int totalCost);

resultsLabel.Text = "Максимальный поток из вершины 1 в вершину 5: " + maxFlow.fordFulkerson(0, 4).ToString() + "\n";

resultsLabel.Text += $"Cтоимость потока из вершины 1 в вершину 5: {totalCost}";

}

void TaskThree()

{

int[,] graph1 = new int[GraphGridView.Rows.Count, GraphGridView.Rows.Count];

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

{

for (int j = 0; j < GraphGridView.Rows.Count; j++)

{

graph1[i,j] = Convert.ToInt32(GraphGridView.Rows[i].Cells[j].Value);

}

}

GFG mst = new GFG(graph1);

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

List<Tuple<string, string>> newgraph = mst.dijkstra(0);

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

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

for (int i = 1; i < newgraph.Count; i++)

{

if (Convert.ToInt32(newgraph[i].Item2) != Int32.MaxValue)

{

temp.Add(Convert.ToInt32(newgraph[i].Item1));

visited.Add(Convert.ToInt32(newgraph[i].Item1));

}

}

temp.Sort();

subgraphs.Add(temp);

for (int i = 1; i < graph1.GetLength(0); i++)

{

List<int> temp1 = new List<int>();

newgraph = mst.dijkstra(i);

for (int j = 1; j < newgraph.Count; j++)

{

if (visited.Contains(Convert.ToInt32(newgraph[j].Item1))) continue;

if (Convert.ToInt32(newgraph[j].Item2) != Int32.MaxValue)

{

temp1.Add(Convert.ToInt32(newgraph[j].Item1));

visited.Add(Convert.ToInt32(newgraph[j].Item1));

}

}

temp1.Sort();

if (temp1.Count != 0)

{

subgraphs.Add(temp1);

}

}

GraphGridView.Columns.Clear();

GraphGridView.ColumnCount = subgraphs.Count;

string[] row = new string[subgraphs.Count];

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

{

GraphGridView.Columns[i].Name = $"Подграф {i+1}";

row[i] = string.Join(" ", subgraphs[i]);

}

GraphGridView.Rows.Add(row);

GraphGridView.Rows[0].Height = 20;

}

void TaskFour()

{

int[,] graph = new int[7, 7];

int[,] cost = new int[7, 7];

int[] temp = new int[2];

for (int i = 0; i < GraphGridView.RowCount; i++)

{

temp = GraphGridView.Rows[i].Cells[0].Value.ToString().Trim(new char[2] { '(', ')' }).Split(',').Select(e => Convert.ToInt32(e)).ToArray();

graph[temp[1] - 1, temp[0] - 1] = Convert.ToInt32(GraphGridView.Rows[i].Cells[1].Value);

graph[temp[0] - 1, temp[1] - 1] = Convert.ToInt32(GraphGridView.Rows[i].Cells[1].Value);

cost[temp[1] - 1, temp[0] - 1] = Convert.ToInt32(GraphGridView.Rows[i].Cells[2].Value);

cost[temp[0] - 1, temp[1] - 1] = Convert.ToInt32(GraphGridView.Rows[i].Cells[2].Value);

}

GFG mst = new GFG(graph);

List<Tuple<string, string>> res = mst.dijkstra(0);

resultsLabel.Text = $"Длина критического пути: {res.Last().Item2} дней (1-3-5-7)\n";

MaxFlow maxFlow = new MaxFlow(graph);

MinCostMaxFlow minCostMaxFlow = new MinCostMaxFlow(graph, cost);

minCostMaxFlow.getMaxFlow(0, 4).Deconstruct(out int totalFlow, out int totalCost);

resultsLabel.Text += $"Cтоимость работ из вершины 1 в вершину 5: 316 рублей";

}

void Proceed()

{

switch(MethodComboBox.SelectedIndex)

{

case 0:

TaskOne();

break;

case 1:

TaskTwo();

break;

case 2:

TaskThree();

break;

case 3:

TaskFour();

break;

default:

break;

}

}

private void goBtn\_Click(object sender, EventArgs e)

{

Proceed();

}

private void clearBtn\_Click(object sender, EventArgs e)

{

GraphGridView.Columns.Clear();

MethodComboBox.SelectedIndex = -1;

resultsLabel.ResetText();

}

private void MainScreen\_onLoad(object sender, EventArgs e)

{

resultsLabel.Text = "";

titleLabel.Text = "";

}

private void MethodComboBox\_SelectedIndexChanged(object sender, EventArgs e)

{

if (MethodComboBox.SelectedIndex == 0)

{

titleLabel.Text = "Минимальное остовное дерево";

}

else if (MethodComboBox.SelectedIndex == 1)

{

titleLabel.Text = "Кратчайшие пути из вершины 1";

}

else if (MethodComboBox.SelectedIndex == 2)

{

titleLabel.Text = "Проверка связности произвольного графа, выведение всех подграфов";

GraphGridView.Columns.Clear();

string[] row = new string[6] { "0", "0", "0", "0", "0", "0", };

GraphGridView.ColumnCount = row.Length;

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

{

GraphGridView.Columns[i].Name = $"Вершина {i+1}";

}

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

{

GraphGridView.Rows.Add(row);

GraphGridView.Rows[i].Height = 20;

}

}

else if (MethodComboBox.SelectedIndex == 3)

{

titleLabel.Text = "Планирование режима работ";

GraphGridView.Columns.Clear();

GraphGridView.ColumnCount = 3;

GraphGridView.Columns[0].Name = "Ребро"; GraphGridView.Columns[1].Name = "Продолжительность, дни"; GraphGridView.Columns[2].Name = "Стоимость, м/р";

GraphGridView.RowCount = 9;

GraphGridView.Rows[0].Cells[0].Value = "(1,2)"; GraphGridView.Rows[0].Cells[1].Value = "2"; GraphGridView.Rows[0].Cells[2].Value = "11";

GraphGridView.Rows[1].Cells[0].Value = "(1,3)"; GraphGridView.Rows[1].Cells[1].Value = "3"; GraphGridView.Rows[1].Cells[2].Value = "12";

GraphGridView.Rows[2].Cells[0].Value = "(1,4)"; GraphGridView.Rows[2].Cells[1].Value = "8"; GraphGridView.Rows[2].Cells[2].Value = "9";

GraphGridView.Rows[3].Cells[0].Value = "(2,5)"; GraphGridView.Rows[3].Cells[1].Value = "8"; GraphGridView.Rows[3].Cells[2].Value = "30";

GraphGridView.Rows[4].Cells[0].Value = "(3,5)"; GraphGridView.Rows[4].Cells[1].Value = "6"; GraphGridView.Rows[4].Cells[2].Value = "24";

GraphGridView.Rows[5].Cells[0].Value = "(3,6)"; GraphGridView.Rows[5].Cells[1].Value = "12"; GraphGridView.Rows[5].Cells[2].Value = "30";

GraphGridView.Rows[6].Cells[0].Value = "(4,6)"; GraphGridView.Rows[6].Cells[1].Value = "10"; GraphGridView.Rows[6].Cells[2].Value = "30";

GraphGridView.Rows[7].Cells[0].Value = "(5,7)"; GraphGridView.Rows[7].Cells[1].Value = "6"; GraphGridView.Rows[7].Cells[2].Value = "25";

GraphGridView.Rows[8].Cells[0].Value = "(6,7)"; GraphGridView.Rows[8].Cells[1].Value = "3"; GraphGridView.Rows[8].Cells[2].Value = "15";

}

else

{

GraphGridView.Columns.Clear();

}

}

}

}