МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО МГТУ «СТАНКИН»)

Институт
информационных технологий

Кафедра
информационных систем

Отчет по лабораторной работе №1

по дисциплине «**Интеллектуальные и экспертные системы**»
на тему: «Изучение методов поиска на явно заданном графе»

Студент **Мустафаева П.М.**
группа ИДБ–22–06

подпись

Руководитель **Быстрикова В. А.**
старший преподаватель

подпись

Москва 2025 г.

ВВЕДЕНИЕ

Цель работы является изучение алгоритмов поиска на явно заданном графе, а также различных форм организации хранения и обработки данных. Разработка программы, реализующей алгоритм поиска в глубину, и программы, выполняющей поиск в ширину.

ХОД РАБОТЫ

Разработать программную реализацию алгоритмов поиска на явно заданном графе с использованием Visual Studio.

В качестве исходных данных используется:

а) представление явно заданного графа способом, выбранным студентом;

б) начальная вершина;

в) целевая вершина;

г) указание метода поиска на графе.

Результатом выполнения программы является:

а) определение факта наличия пути между начальной и целевой вершинами;

б) при существовании пути необходимо вывести этот путь в виде последовательности вершин, в которой каждая вершина соединена ребром со следующей вершиной;

в) определение числа шагов, за которое была найдена искомая вершина графа.

Тестовый пример графа представлен на рис. 1.

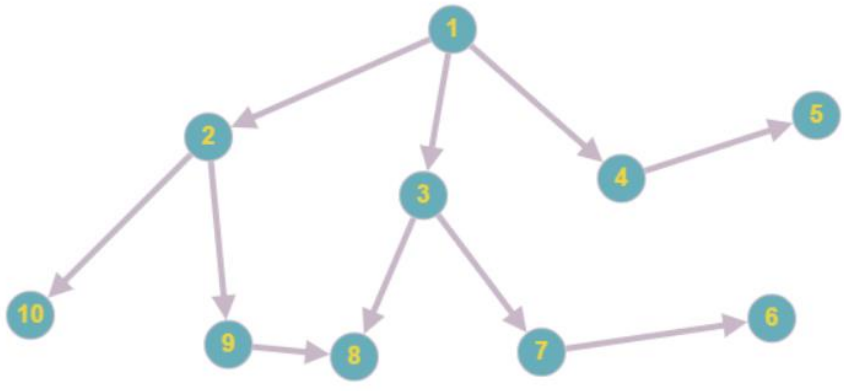

Рис. 1 Тестовый пример графа в графическом виде

Граф для алгоритма будет представлен в виде матрицы смежности:

0, 1, 1, 1, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 1, 1,

0, 0, 0, 0, 0, 0, 1, 1, 0, 0,

0, 0, 0, 0, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Для поиска решение на графе будут использованы 2 алгоритма: поиск в ширину и поиск в глубину.

Алгоритм поиска в ширину:

CLOSED - список исследованных (раскрытых) вершин.

Шаг 1. Поместить начальную вершину в список Open. Создать пустой список Closed.

Шаг 2. Если список Open пуст, то конец алгоритма с сообщением о неудаче поиска, иначе перейти к Шагу 3.

Шаг 3. Выбрать первую вершину X из списка Open и перенести ее в список Closed, удалив из Open.

Шаг 4. Если X - целевая вершина, то конец алгоритма и выдача решения, иначе перейти к Шагу 5.

Шаг 5. Образовать потомков X. Не находящихся в Open и Closed добавить в конец списка Open. Построить указатели от этих вершин к X. Перейти к Шагу 2.

Алгоритм поиска в глубину:

Шаг 1. Поместить начальную вершину в список Open. Создать пустой список Closed.

Шаг 2. Если список Open пуст, то конец алгоритма с сообщением о неудаче поиска, иначе перейти к Шагу 3.

Шаг 3. Выбрать первую вершину X из списка Open и перенести ее в список Closed, удалив из Open.

Шаг 4. Если X - целевая вершина, то конец алгоритма и выдача решения, иначе перейти к Шагу 5.

Шаг 5. Образовать потомков X. Не находящихся в Open и Closed добавить в начало списка Open. Построить указатели от этих вершин к X. Перейти к Шагу 2.

Программа для алгоритма поиска путей на графе написана с использованием Windows Forms и языка программирования C#. Программный код представлен в листинге 1.

Листинг 1 – Программный код

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace LR1_graf
{
   public partial class Graph : Form
   {
      private int vertexCount;

      public Graph()
      {
         InitializeComponent();
         InitializeDataMatrix();
      }

      private void Form1_Load(object sender, EventArgs e)
      {

      }
      private void InitializeDataMatrix()
      {
         vertexCount = 5;
         dataMatrix.ColumnCount = vertexCount;
         dataMatrix.RowCount = vertexCount;
         ConfigureDataMatrixStyle();
      }

      private void ConfigureDataMatrixStyle()
      {
         dataMatrix.DefaultCellStyle.Font = new System.Drawing.Font("Arial", 9);
         dataMatrix.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
         dataMatrix.ColumnHeadersDefaultCellStyle.Font = new System.Drawing.Font("Arial",
7, System.Drawing.FontStyle.Bold);
```

```
        dataMatrix.ColumnHeadersDefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
        dataMatrix.RowHeadersDefaultCellStyle.Font = new System.Drawing.Font("Arial", 7,
System.Drawing.FontStyle.Bold);
        dataMatrix.RowHeadersDefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;

        for (int i = 0; i < vertexCount; i++)
        {
            dataMatrix.Columns[i].Name = "V" + (i + 1);
            dataMatrix.Columns[i].Width = 24;
            dataMatrix.Rows[i].HeaderCell.Value = "V" + (i + 1);
        }
    }


    private int[,] GetAdjacencyMatrix()
    {
        int[,] matrix = new int[vertexCount, vertexCount];

        for (int i = 0; i < vertexCount; i++)
        {
            for (int j = 0; j < vertexCount; j++)
            {
                if (int.TryParse(dataMatrix.Rows[i].Cells[j].Value?.ToString(), out int value))
                {
                    matrix[i, j] = value;
                }
            }
        }

        return matrix;
    }

    private void butBFS_Click(object sender, EventArgs e)
    {
        if (!int.TryParse(txtStart.Text, out int start) || !int.TryParse(txtTarget.Text, out int target))
        {
            MessageBox.Show("Введите корректные номера вершин.");
            return;
        }

        start--;
        target--;

        int[,] adjacencyMatrix = GetAdjacencyMatrix();
        List<int> path = BFS(adjacencyMatrix, start, target, out int steps);

        if (path.Count > 0 && path[0] != -1)
        {
            txtPathBFS.Text = string.Join(" -> ", path);
            txtStepsBFS.Text = steps.ToString();
```

```
        }
        else
        {
            txtPathBFS.Text = "Путь не найден";
            txtStepsBFS.Text = "0";
        }
    }

    private List<int> BFS(int[,] adjacencyMatrix, int start, int target, out int steps)
    {
        Queue<int> open = new Queue<int>();
        HashSet<int> closed = new HashSet<int>();
        Dictionary<int, int> parent = new Dictionary<int, int>();

        open.Enqueue(start);
        steps = 0;

        while (open.Count > 0)
        {
            int x = open.Dequeue();
            closed.Add(x);
            steps++;

            if (x == target)
            {
                return ConstructPath(parent, start, target);
            }

            for (int i = 0; i < vertexCount; i++)
            {
                if (adjacencyMatrix[x, i] == 1 && !closed.Contains(i) && !open.Contains(i))
                {
                    open.Enqueue(i);
                    parent[i] = x;
                }
            }
        }

        return new List<int> { -1 };
    }

    private List<int> ConstructPath(Dictionary<int, int> parent, int start, int target)
    {
        List<int> path = new List<int>();

        if (parent.ContainsKey(target))
        {
            int current = target;
            while (current != start)
            {
                path.Add(current + 1);
                current = parent[current];
```

```
            }
            path.Add(start + 1);
            path.Reverse();
        }

        return path;
    }

    private void butDFS_Click(object sender, EventArgs e)
    {
        if (!int.TryParse(txtStart.Text, out int start) || !int.TryParse(txtTarget.Text, out int target))
        {
            MessageBox.Show("Введите корректные номера вершин.");
            return;
        }

        start--;
        target--;

        int[,] adjacencyMatrix = GetAdjacencyMatrix();
        List<int> path = DFS(adjacencyMatrix, start, target, out int steps);

        if (path.Count > 0 && path[0] != -1)
        {
            txtPathDFS.Text = string.Join(" -> ", path);
            txtStepsDFS.Text = steps.ToString();
        }
        else
        {
            txtPathDFS.Text = "Путь не найден";
            txtStepsDFS.Text = "0";
        }
    }

    private List<int> DFS(int[,] adjacencyMatrix, int start, int target, out int steps)
    {
        Stack<int> open = new Stack<int>();
        HashSet<int> closed = new HashSet<int>();
        Dictionary<int, int> parent = new Dictionary<int, int>();

        open.Push(start);
        steps = 0;

        while (open.Count > 0)
        {
            int x = open.Pop();
            steps++;

            if (!closed.Contains(x))
            {
                closed.Add(x);
            }
```

```
            if (x == target)
            {
                return ConstructPath(parent, start, target);
            }

            for (int i = vertexCount - 1; i >= 0; i--)
            {
                if (adjacencyMatrix[x, i] == 1 && !closed.Contains(i))
                {
                    open.Push(i);
                    parent[i] = x;
                }
            }
        }

        return new List<int> { -1 };
    }

    private void butClear_Click(object sender, EventArgs e)
    {
        txtPathBFS.Clear();
        txtStepsBFS.Clear();
        txtPathDFS.Clear();
        txtStepsDFS.Clear();
    }

    private void butUpdateMatrix_Click(object sender, EventArgs e)
    {
        if (int.TryParse(txtVertexCount.Text, out int newVertexCount) && newVertexCount >
0)
        {
            vertexCount = newVertexCount;

            dataMatrix.ColumnCount = vertexCount;
            dataMatrix.RowCount = vertexCount;

            ConfigureDataMatrixStyle();
        }
        else
        {
            MessageBox.Show("Введите корректное количество вершин.");
        }
    }
  }
}
```

Результаты поиска в ширину и глубину представлены на рис. 2.

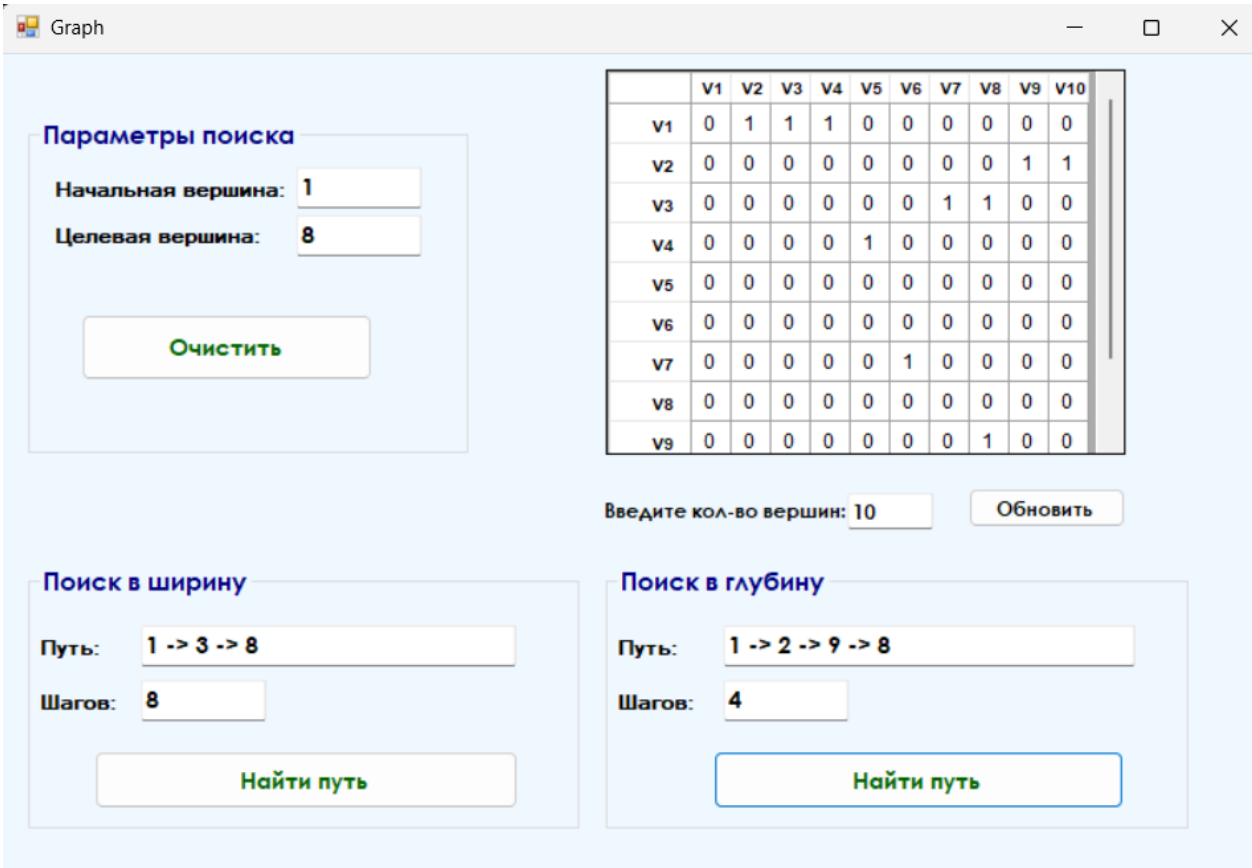

Рис. 2 Экранная форма с результатами работы алгоритмов

После проверки работоспособности программы необходимо проверить эффективность их работы. Для этого использовался тестовый граф, построенный в редакторе. Сравнив полученные результаты в программе и результаты на тестовом графе можно увидеть, что они сходятся (рис. 3-4).

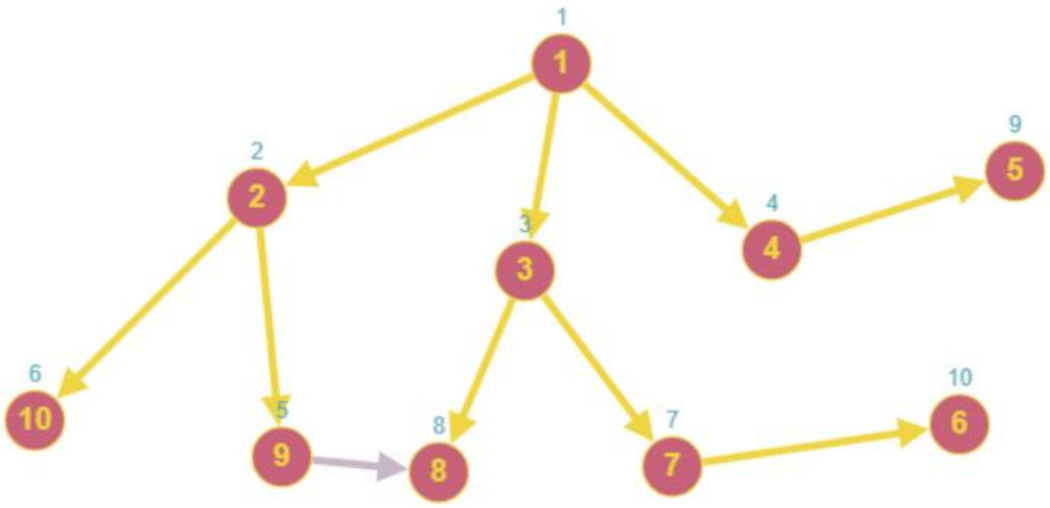

Рис. 3 Поиск в ширину

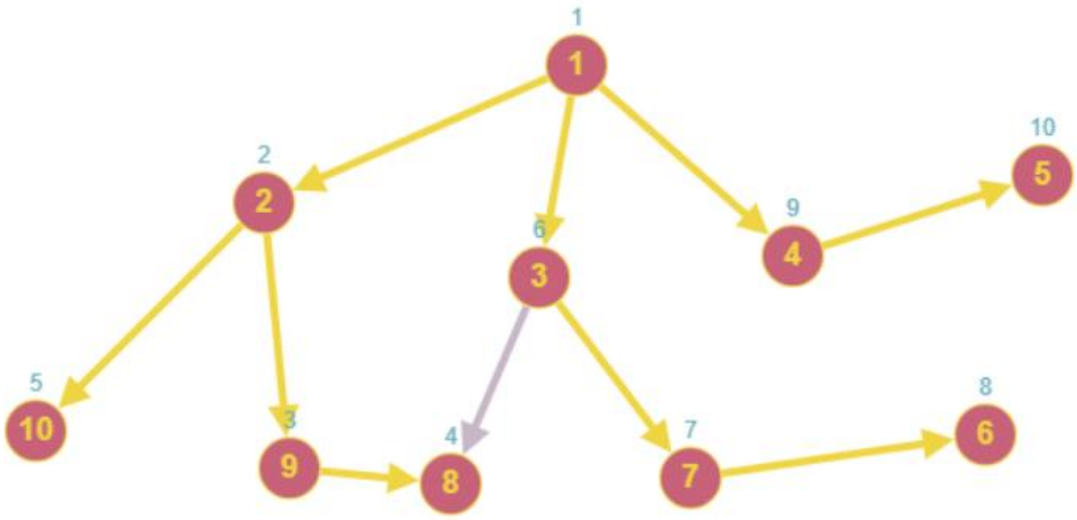

Рис. 4 Поиск в глубину

ВЫВОД

В ходе выполнения работы была разработана программа для реализации алгоритма поиска в глубину и для алгоритма поиска в ширину. Оба алгоритма были визуализированы с помощью Windows Forms, что позволило наглядно продемонстрировать их работу и сравнить эффективность.