Національний авіаційний університет

Факультет кібербезпеки, комп’ютерної та програмної інженерії

**ЗВІТ**

**по лабораторній роботі No 1**

**Алгоритми Дейкстри та Флойда пошуку найкоротших шляхів в мережі.**

Дисципліна: «Методи оптимізацій та дослідження операцій»

Кафедра: прикладної математики

ОС: бакалавр

Спеціальність: 113 «Прикладна математика»

ОПП: «Прикладне програмне забезпечення»

Виконав: здобувач вищої освіти 4 курсу. 451 групи

Архіпов Олексій Тімурович

Перевірив: Хребет Валерій Григорович

Київ 2024

**Тема:** Алгоритми Дейкстри та Флойда пошуку найкоротших шляхів в мережі.

**Мета:** Набуття практичних навичок розробки та налагодження програм відшукання найкоротших шляхів у зважених графах.

**Завдання:**

Для заданих графів (згідно варіанту) виконати наступні задачі:

Побудуйте модель задачі найкоротшого шляху:

а) від початкової вершини до всіх інших вершин (табл. а) ) за алгоритмом Дейкстри та знайти його;

б) між усіма парами вершин (табл. б)) за алгоритмом Флойда та знайти їх;

**Порядок обчислень**

Метод Дейкстри - це алгоритм для пошуку найкоротшого шляху з однієї вершини графа (або мережі) до всіх інших вершин. Цей метод був розроблений нідерландським вченим Едсгером Дейкстрой в 1956 році. Він є одним із найвідоміших алгоритмів найкоротшого шляху і широко використовується у різних галузях, таких як телекомунікації, транспорт, GPS-навігація та інші.

**Принцип роботи:**

1. **Ініціалізація**: Алгоритм починається з початкової вершини та масиву відстаней, де відстань від початкової вершини до всіх інших вершин встановлюється як нескінченність. Розсташування відстаней від початкової вершини до неї ж самої встановлюється як 0.
2. **Вибір вершини**: Обирається вершина, до якої найкоротший шлях відомий, але ще не закритий.
3. **Оновлення відстаней**: Для кожної сусідньої вершини поточної вершини перераховується довжина найкоротшого відомого шляху до неї через поточну вершину. Якщо такий шлях коротший за поточний відомий найкоротший шлях до сусідньої вершини, відстань до неї оновлюється.
4. **Закриття вершини**: Після того як всі сусідні вершини поточної вершини перевірені, поточна вершина вважається закритою.
5. **Повторення**: Кроки 2-4 повторюються для всіх вершин графа.
6. **Визначення найкоротшого шляху**: Після завершення алгоритму для кожної вершини знаходиться найкоротший шлях від початкової вершини до неї.

Цей алгоритм має часову складність O(V^2) для реалізації з використанням матриці суміжності та O((V+E)logV) для реалізації з використанням черги пріоритетів, де V - кількість вершин, E - кількість ребер.

Метод Флойда (також відомий як алгоритм Флойда-Уоршелла) - це алгоритм для пошуку найкоротших шляхів між кожною парою вершин у напіввагованому або вагованому графі. Цей метод був винайдений Робертом Флойдом в 1959 році та незалежно Штанлі Уоршеллом в 1962 році. Метод Флойда-Уоршелла є одним із найбільш загальновживаних алгоритмів для знаходження найкоротших шляхів і зазвичай використовується у випадках, коли кількість вершин у графі невелика, або коли необхідно знайти найкоротший шлях між всіма парами вершин.

1. **Ініціалізація**: Початкова матриця відстаней (D) заповнюється відстанями між сусідніми вершинами, або нескінченностями, якщо вершини не з'єднані прямо. Тобто, якщо вершини i та j з'єднані ребром з вагою w, то D[i][j] буде рівне w, а якщо вершини не з'єднані, то D[i][j] буде рівне нескінченності. Якщо ж i == j, то D[i][j] = 0.

Початкова матриця шляхів (P) заповнюється порожніми значеннями або нулями.

1. **Шлях через проміжну вершину (k)**: Для кожної вершини k виконується оновлення матриці відстаней D та матриці шляхів P. Для кожної пари вершин (i, j) перевіряємо, чи можемо скоротити шлях від вершини i до вершини j через вершину k. Це робиться наступним чином:

D[i][j] = min(D[i][j], D[i][k] + D[k][j])

Де D[i][j] - відстань між вершинами i та j, D[i][k] - відстань між вершинами i та k, а D[k][j] - відстань між вершинами k та j.

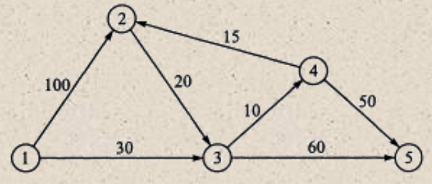
При цьому, якщо шлях від i до j через вершину k виявиться коротшим, ніж поточно відомий найкоротший шлях, то оновлюється відстань та шлях від i до j.

1. **Повторення**: Кроки 2 та 3 повторюються для всіх вершин графа у якості проміжної вершини.
2. **Кінцевий результат**: Після завершення алгоритму матриця відстаней D містить найкоротші відстані між усіма парами вершин у графі. І матриця шляхів P містить інформацію про найкоротший шлях між кожною парою вершин.

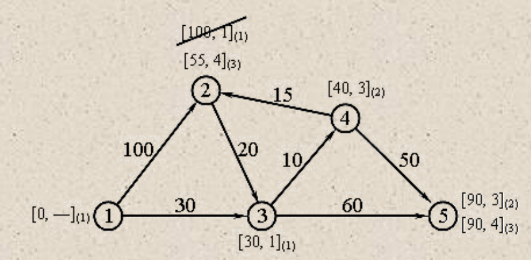
Цей алгоритм гарантує знаходження найкоротших шляхів між усіма парами вершин у графі, але він має складність O(V^3), де V - кількість вершин у графі. Також, він може працювати з графами, що містять від'ємні ваги, за умови, що в графі немає від'ємного циклу.

**Результати тестування**

Приклад 1:



Результат:



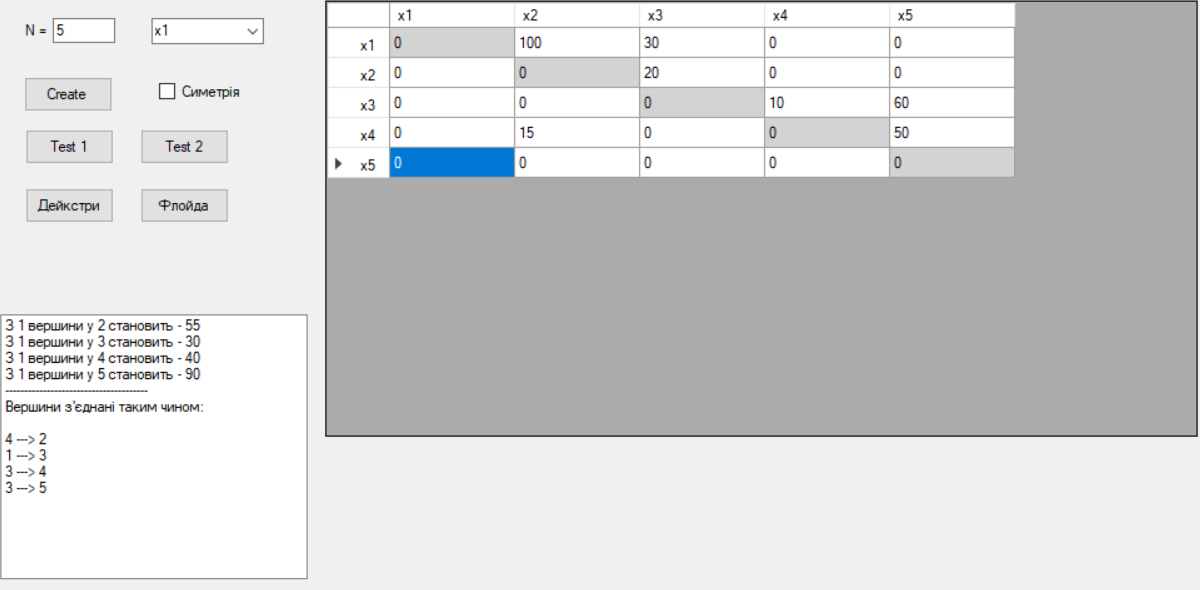


Рис 1. Алгоритм Дейкстри

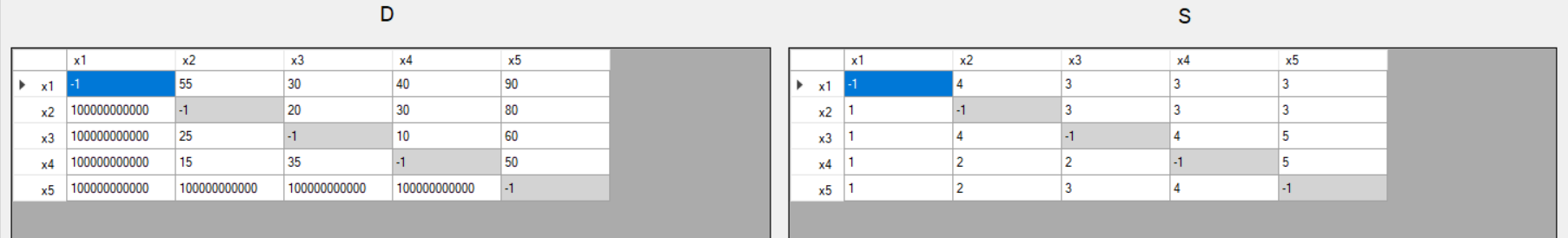
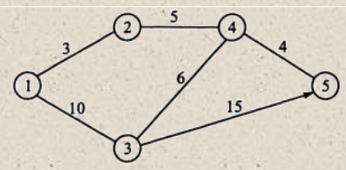


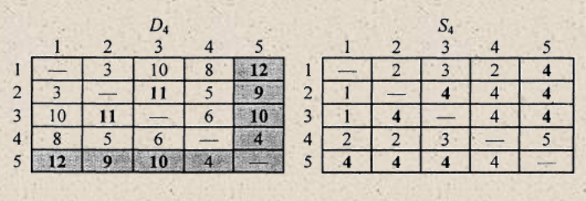
Рис 2. Алгоритм Флойда

Як видно з рис 1 і рис 2 результати співпадають.

Приклад 2:



Результат:



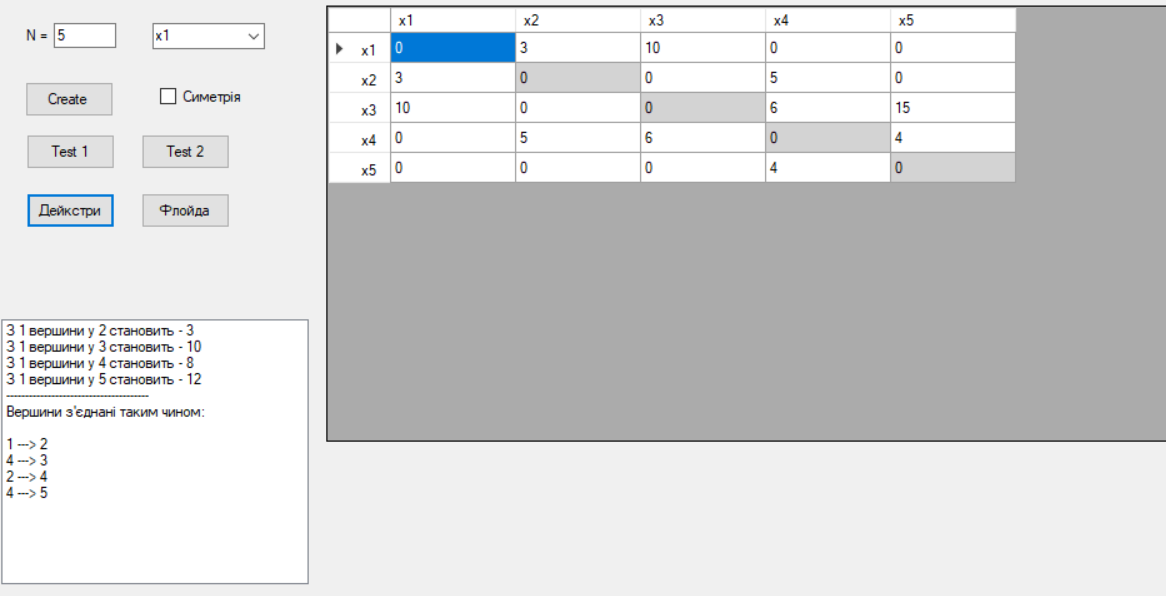


Рис 3. Алгоритм Дейкстри

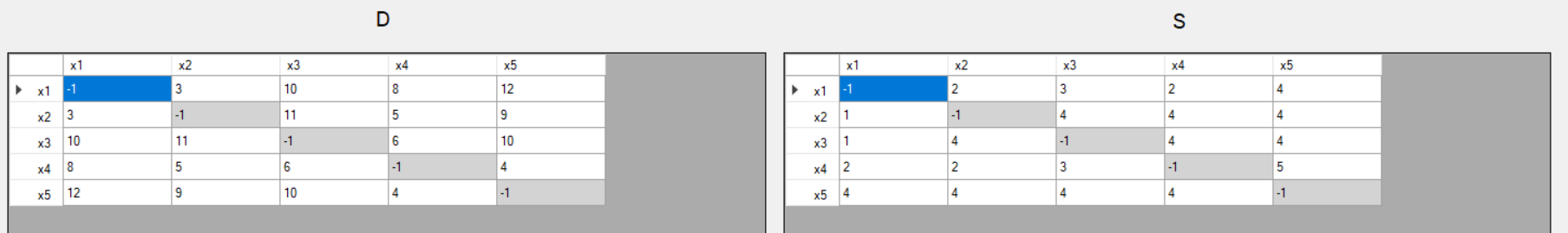


Рис 4. Алгоритм Флойда

Як видно з рис 3 і рис 4 результати співпадають.

**Висновоки**

1. Набув практичних навичок розробки та налагодження програм відшукання найкоротших шляхів у зважених графах.
2. Реалізував алгоритми Дейкстри та Флойда для пошуку найкоротших шляхів в мережі.

**Код програми**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

dg.AllowUserToAddRows = false;

listBox1.HorizontalScrollbar = true;

}

int n = 0, firstpoint = 0;

double[,] matrix;

private void dg\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

void CreateMatrix2()

{

cb1.Items.Clear();

n = Convert.ToInt32(tbn.Text);

dg.Rows.Clear();

dg.Columns.Clear();

for (int i = 0; i < n; i++)

{

cb1.Items.Add("x" + (i+1).ToString());

dg.Columns.Add("x" + (i + 1).ToString(), "x" + (i + 1).ToString());

}

cb1.SelectedIndex = 0;

dg.Rows.Add(n);

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

{

dg.Rows[i].HeaderCell.Value = "x" + (i + 1).ToString();

}

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if(i == j) dg.Rows[i].Cells[i].Style.BackColor = Color.LightGray;

dg.Rows[i].Cells[j].Value = "0";

}

}

}

void CreateMatrixTest()

{

cb1.Items.Clear();

n = 6;

tbn.Text = n.ToString();

dg.Rows.Clear();

dg.Columns.Clear();

for (int i = 0; i < n; i++)

{

cb1.Items.Add("x" + (i + 1).ToString());

dg.Columns.Add("x" + (i + 1).ToString(), "x" + (i + 1).ToString());

}

cb1.SelectedIndex = 0;

dg.Rows.Add(n);

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

{

dg.Rows[i].HeaderCell.Value = "x" + (i + 1).ToString();

}

double[,] M = new double[n, n];

M[0, 1] = M[1, 0] = 7;

M[0, 2] = M[2, 0] = 9;

M[0, 5] = M[5, 0] = 14;

M[1, 2] = M[2, 1] = 10;

M[1, 3] = M[3, 1] = 15;

M[2, 3] = M[3, 2] = 11;

M[2, 5] = M[5, 2] = 2;

M[4, 3] = M[3, 4] = 6;

M[4, 5] = M[5, 4] = 9;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i == j) dg.Rows[i].Cells[i].Style.BackColor = Color.LightGray;

dg.Rows[i].Cells[j].Value = M[i,j].ToString();

}

}

}

void CreateMatrixTest2()

{

cb1.Items.Clear();

n = 5;

tbn.Text = n.ToString();

dg.Rows.Clear();

dg.Columns.Clear();

for (int i = 0; i < n; i++)

{

cb1.Items.Add("x" + (i + 1).ToString());

dg.Columns.Add("x" + (i + 1).ToString(), "x" + (i + 1).ToString());

}

cb1.SelectedIndex = 0;

dg.Rows.Add(n);

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

{

dg.Rows[i].HeaderCell.Value = "x" + (i + 1).ToString();

}

double[,] M = new double[n, n];

M[0, 1] = M[1, 0] = 3;

M[0, 2] = M[2, 0] = 10;

M[1, 3] = M[3, 1] = 5;

M[2, 3] = M[3, 2] = 6;

M[4, 3] = M[3, 4] = 4;

M[2, 4] = 15;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i == j) dg.Rows[i].Cells[i].Style.BackColor = Color.LightGray;

dg.Rows[i].Cells[j].Value = M[i, j].ToString();

}

}

}

private void dg\_CellValueChanged(object sender, DataGridViewCellEventArgs e)

{

if (checkBox1.Checked)

{

if (dg.Columns.Count > 0 && dg.Rows.Count > 0)

{

if (e.ColumnIndex >= 0 && e.RowIndex >= 0)

{

var newValue = dg[e.ColumnIndex, e.RowIndex].Value;

dg[e.RowIndex, e.ColumnIndex].Value = newValue;

}

}

}

}

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

{

CreateMatrix2();

}

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

{

CreateMatrixTest();

}

void Read()

{

n = Convert.ToInt32(tbn.Text);

matrix = new double[n,n];

firstpoint = cb1.SelectedIndex;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

matrix[i,j] = Convert.ToDouble(dg.Rows[i].Cells[j].Value);

}

}

}

void Deikstru()

{

listBox1.Items.Clear();

Read();

List<Tuple<double, int, bool>> All = new List<Tuple<double, int, bool>>();

for(int i = 0; i < n; i++)

{

if (i == firstpoint) All.Add(new Tuple<double, int, bool>(0, firstpoint, true));

else All.Add(new Tuple<double, int, bool>(100000000000000, -1, false));

}

int nextpoint = firstpoint;

do

{

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

for (int i = 0; i < n; i++)

{

if (matrix[nextpoint, i] != 0 && All[i].Item3 == false) index.Add(i);

}

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

{

double temp = matrix[nextpoint, index[i]] + All[nextpoint].Item1;

List<double> temp2 = new List<double>();

if (temp < All[index[i]].Item1) All[index[i]] = new Tuple<double, int, bool>(temp, nextpoint, false);

}

double min = All.Where(item => !item.Item3).Min(item => item.Item1);

int minIndex = All.FindIndex(item => item.Item1 == min && !item.Item3);

if (matrix[minIndex, nextpoint] != 0 && All[minIndex].Item1 > All[nextpoint].Item1 && (All.Count - All.Count(item => item.Item3)) > 1) All[minIndex] = new Tuple<double, int, bool>(All[minIndex].Item1, nextpoint, true);

else All[minIndex] = new Tuple<double, int, bool>(All[minIndex].Item1, All[minIndex].Item2, true);

nextpoint = minIndex;

if (All.Count == All.Count(item => item.Item3)) break;

} while (true);

for(int i = 0; i < n; i++)

{

if(i != firstpoint)

{

listBox1.Items.Add("З " + (firstpoint + 1).ToString() + " вершини у " + (i + 1).ToString() + " становить - " + All[i].Item1.ToString());

}

}

listBox1.Items.Add("--------------------------------------");

listBox1.Items.Add("Вершини з'єднані таким чином:");

listBox1.Items.Add("");

for (int i = 0; i < n; i++)

{

if (i != firstpoint)

{

listBox1.Items.Add((All[i].Item2 + 1).ToString() + " ---> " + (i + 1).ToString());

}

}

}

void Floida()

{

listBox1.Items.Clear();

Read();

double[,] D = new double[n, n];

double[,] S = new double[n, n];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i == j)

{

D[i, j] = -1;

S[i, j] = -1;

}

else if (matrix[i, j] == 0)

{

D[i, j] = 100000000000;

S[i, j] = j;

}

else

{

D[i, j] = matrix[i, j];

S[i, j] = j;

}

}

}

for(int k = 0; k < n; k++)

{

for(int i = 0; i < n; i++)

{

for(int j = 0; j < n; j++)

{

if(i != j && j != k && i != k)

{

if (D[i,k] + D[k,j] < D[i, j])

{

D[i, j] = D[i, k] + D[k, j];

S[i, j] = k;

}

}

}

}

}

Form2 form = new Form2(D, S, n);

form.ShowDialog();

}

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

{

Floida();

}

private void btnTest2\_Click(object sender, EventArgs e)

{

CreateMatrixTest2();

}

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

{

Deikstru();

}

}

}