Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторної роботи №3

На тему:
«Застосування дерев. Властивості відношень»
з дисципліни
«Комп’ютерна дискретна математика»

Лектор:
Журавчак Л.М.
Виконав:
ст. гр. ПЗ-18
Юшкевич А.І.
Прийняв:
Курапов П.Р.

« -- » -- 2022 р.

∑ = _____

Львів – 2022

Тема: Застосування дерев. Властивості відношень.

Мета: Ознайомитись на практиці з алгоритмом Краскала побудови мінімального остовного дерева, з алгоритмом побудови дерева за заданим кодом Прюфера, з поданням арифметичних виразів у вигляді математичного (бінарного) дерева та польськими записами, навчитися обчислювати значення виразів в обох цих записах, освоїти алгоритм бектрекінгу для розфарбування графа мінімально можливою кількістю кольорів. Реалізувати обчислення матриць операцій над двома бінарними відношеннями, а також визначення властивостей бінарного відношення.

Теоретичні відомості

Під час написання програми використовувався об'єктно орієнтований підхід. Кожній задачі відповідає окремий клас, який є самодостатнім і може функціонувати незалежно від інших класів.

Початковим методом у кожному класі відповідної задачі є метод, що у своєму ідентифікаторі містить слово "Task" ("FirstTask", "SecondTask", і т.п.). Щоб запустити кожен приклад окремо треба змінити назву вищезазначеного метода на "Main", або використати наведений нижче клас, для повноцінного функціонування програми. (Вимагає наявності УСІХ класів в межах одного рішення)

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;

namespace KDM_Lab03
{
    class Program
    {
        public static void Main()
        {
            bool isStopped = false;

            while (!isStopped)
            {
                Program program = new Program();

                switch (program.Hello())
                {
                    case 0:
                        isStopped = true;
                        break;
                    case 1:
                        Task01.FirstTask();
                        break;
                    case 2:
                        Task02.SecondTask();
                        break;
                    case 5:
                        Task05.FifthTask();
                        break;
                    case 6:
                        Task06.SixthTask();
                        break;

                    default:

                        Console.ForegroundColor = ConsoleColor.DarkRed;
```

```
                        Console.WriteLine("\nSorry, but we don't have this task. Please,
try again.\n");
                        Console.ForegroundColor = ConsoleColor.Gray;
                        break;
                    }
                }
            }
            int Hello()
            {
                int taskNumber = -1;
                string s = "-1";

                Console.ForegroundColor = ConsoleColor.Green;
                Console.Write("Please, Enter the number of task, you want to check or press
\"S\" to stop process: ");
                s = Console.ReadLine();
                Console.ForegroundColor = ConsoleColor.Gray;

                if (s == "s" || s == "S" || s == "Stop")
                {
                    taskNumber = 0;
                }
                else
                {
                    try
                    {
                        taskNumber = Convert.ToInt32(s);
                    }
                    catch (FormatException)
                    {

                    }
                }

                return taskNumber;
            }
        }
    }
```

Додаток 1 до лабораторної роботи № 3

Завдання 1

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує знаходження мінімального остовного (кістякового) дерева (МОД) неорієнтованого зваженого зв'язного графа, заданого ваговою матрицею (таблиця 3.1) за алгоритмом Краскала. Етапи розв'язання задачі виводити на екран.

Одна з можливих реалізацій алгоритму Краскала (алгоритм пошуку МОД):

Крок 1. Упорядкувати множину ребер у порядку зростання ваг: e1, e2,..., et.

Крок 2. Утворити розбиття множини вершин на одноелементні підмножини: {{v1}, {v2}, …, {vn}}.

Крок 3. Вибирати таке чергове ребро з упорядкованої послідовності ребер, що його кінці містяться в різних множинах розбиття (це забезпечить відсутність простих циклів). Якщо вибрано ребро ei={vi, vj}, то множини розбиття об’єднати в одну множину.

Крок 4. Якщо вже вибрано (п-1) ребро (у такому разі всі підмножини розбиття виявляться об’єднаними в одну), то зупинитися, бо вибрані ребра утворюють мінімальний остов. Інакше перейти до кроку 3.

Алгоритм будує МОД у зваженому графі, послідовно вибираючи ребра найменшої можливої ваги. МОД зберігається в пам’яті комп’ютера як множина Т ребер.

Вимоги до програми

Програма має передбачати такі можливості:

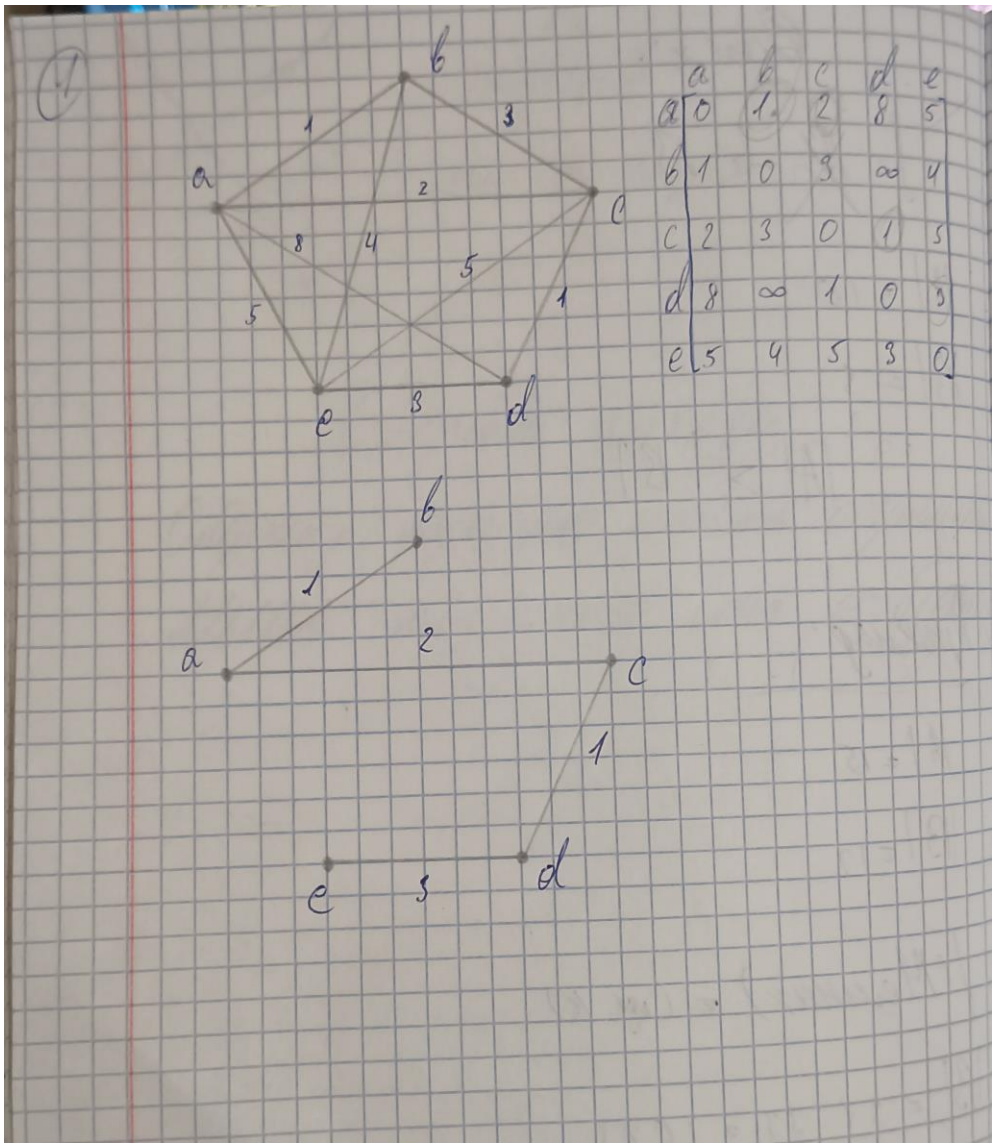
- введення вхідних даних вручну (вершин та ребер графа);
- реалізацію алгоритму Краскала для пошуку МОД;
- виведення на екран результату виконання програми (множин вершин та ребер МОД); перевірку на некоректне введення даних.

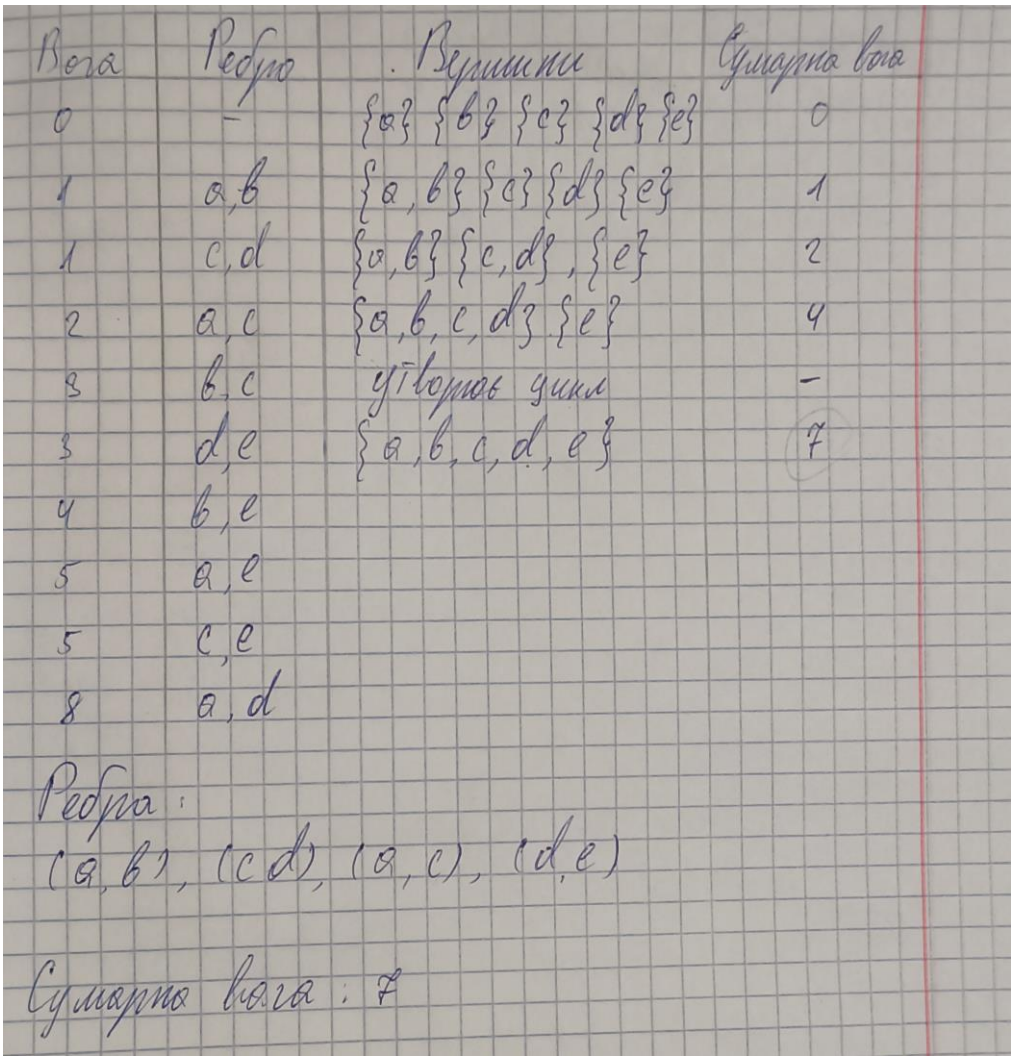
Завдання вважається зарахованим, якщо при тестуванні програми в присутності викладача отримано правильний результат.

Таблиця 3.1

25	$\begin{pmatrix} 0 & 1 & 2 & 8 & 5 \\ 1 & 0 & 3 & \infty & 4 \\ 2 & 3 & 0 & 1 & 5 \\ 8 & \infty & 1 & 0 & 3 \\ 5 & 4 & 5 & 3 & 0 \end{pmatrix}$

Зошит


1
a b c d e
a 0 1 2 8 5
b 1 0 3 ∞ 4
c 2 3 0 1 5
d 8 ∞ 1 0 3
e 5 4 5 3 0
a
b
c
d
e
1
3
2
8
4
5
5
1
3
a
b
c
d
e
1
2
1
3



Реалізація у коді

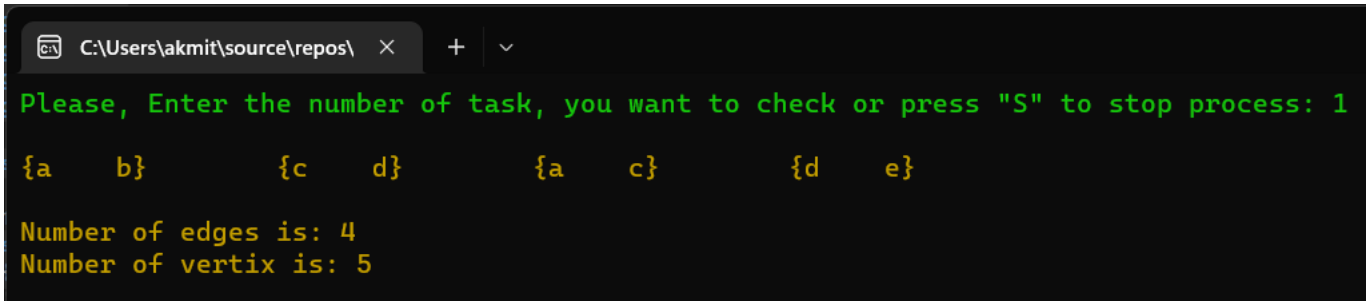
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KDM_Lab03
{
    static class Task01
    {
        public static void FirstTask()
        {
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("Please, Enter graph: ");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] graph = GetMatrix();
            int[,] graph =
            {
                {0,  1,  2,  8, 5},
                {1,  0,  3, -1, 4},
                {2,  3,  0,  1, 5},
                {8, -1,  1,  0, 3},
                {5,  4,  5,  3, 0}
            };
```

```
        //int[,] graph =
        //{
        //    {0,  1,   6,  5, 14},
        //    {1,  0,   3,  4,  6},
        //    {6,  3,   0, 10, 12},
        //    {5,  4,  10,  0,  6},
        //    {14, 6,  12,  6,  0}
        //};

        int[,] edges = FindMOD(graph);
        ShowMatrix(edges);

    }
    static int[,] GetMatrix()
    {
        int size;

        while (true)
        {
            try
            {
                Console.ForegroundColor = ConsoleColor.Blue;
                Console.Write("Please, enter matrix size NxN:\nN = ");
                Console.ForegroundColor = ConsoleColor.Cyan;
                size = Convert.ToInt32(Console.ReadLine());
                Console.ForegroundColor = ConsoleColor.Gray;

                break;
            }
            catch (FormatException)
            {

                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("\nPlease, enter numbers only!\n");
                Console.ForegroundColor = ConsoleColor.Gray;
            }
        }
        Console.WriteLine();

        char[] l = new char[size];
        for (int i = 0; i < l.Length; i++)
        {
            l[i] = (char)(97 + i);
        }

        int[,] myMatrix = new int[size, size];
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                if (j <= i) continue;

                while (true)
                {
                    Console.ForegroundColor = ConsoleColor.Blue;
                    Console.Write("Edge between \"{0}\" and \"{1}\": ", l[i], l[j]);
                    try
                    {
                        Console.ForegroundColor = ConsoleColor.Cyan;
                        myMatrix[i, j] = Convert.ToInt32(Console.ReadLine());
                        Console.ForegroundColor = ConsoleColor.Gray;

                        myMatrix[j, i] = myMatrix[i, j];
                        if (myMatrix[i, j] == 0 || myMatrix[i, j] == 1)
                            break;
                        else
                        {
```

```
                                Console.ForegroundColor = ConsoleColor.Red;
                                Console.WriteLine("\nPlease, enter only \"1\" or
\"0\"\n");
                                Console.ForegroundColor = ConsoleColor.Gray;
                            }
                        }
                        catch (FormatException)
                        {
                            Console.ForegroundColor = ConsoleColor.Red;
                            Console.WriteLine("\nPlease, only digits\n");
                            Console.ForegroundColor = ConsoleColor.Gray;
                        }

                    }
                }
            }

            return myMatrix;
        }
        static int[,] FindMOD(int[,] graph)
        {
            int min = int.MaxValue;
            int iIn = 0, jIn = 0;
            int indexK1 = -1, indexK2 = -1;
            int counter = 0;
            int[,] edges = new int[graph.GetLength(0) - 1, 2];
            int[,] vertices = new int[graph.GetLength(0), graph.GetLength(0)];
            int[] verticesIndexes = new int[graph.GetLength(0)];

            for (int i = 0; i < graph.GetLength(0); i++)
                for (int j = 0; j < graph.GetLength(0); j++)
                {
                    vertices[i, j] = -1;
                    if (j < 2 && i < graph.GetLength(0) - 1)
                        edges[i, j] = -1;
                }

            for (int s = 0; s < graph.GetLength(0) - 1; s++)
            {

                for (int i = 0; i < graph.GetLength(0); i++)
                    for (int j = 0; j < graph.GetLength(0); j++)
                        if (j > i && graph[i, j] > 0 && graph[i, j] < min)
                        {
                            min = graph[i, j];
                            iIn = i;
                            jIn = j;
                        }


                for (int i = 0; i < graph.GetLength(0); i++)
                {
                    for (int j = 0; j < graph.GetLength(0); j++)
                    {
                        if (vertices[i, j] == iIn || vertices[i, j] == jIn)
                            counter++;
                    }

                    if (counter >= 2)
                        break;
                    else if (counter == 1 && indexK1 > -1)
                        indexK2 = i;
                    else if (counter == 1)
                        indexK1 = i;

                    counter = 0;
                }
```

```
                if (counter < 2)
                {
                    edges[s, 0] = iIn;
                    edges[s, 1] = jIn;

                    if (indexK1 == -1)
                    {
                        for (int l = 0; l < graph.GetLength(0); l++)
                        {
                            if (verticesIndexes[l] == 0)
                            {
                                vertices[l, iIn] = iIn;
                                vertices[l, jIn] = jIn;
                                verticesIndexes[l] = 1;
                                break;
                            }

                        }
                    }
                    else if (indexK1 > -1 && indexK2 == -1)
                    {
                        vertices[indexK1, iIn] = iIn;
                        vertices[indexK1, jIn] = jIn;
                    }
                    else if (indexK1 > -1 && indexK2 > -1)
                    {
                        for (int l = 0; l < graph.GetLength(0); l++)
                        {
                            if (vertices[indexK2, l] != -1)
                                vertices[indexK1, l] = vertices[indexK2, l];
                        }
                    }
                }
                else
                    --s;

                counter = 0;
                indexK1 = -1;
                indexK2 = -1;
                graph[iIn, jIn] = int.MaxValue;
                min = int.MaxValue;
            }
            return edges;
        }
        static void ShowMatrix(int[,] matrix)
        {
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.DarkYellow;

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                Console.Write("{");
                for (int j = 0; j < 2; j++)
                {
                    Console.Write((char)(97 + matrix[i, j]));
                    if(j + 1 < 2)
                        Console.Write("    ");
                }
                Console.Write("}\t");
            }
            Console.WriteLine("\n\nNumber of edges is: {0}\nNumber of vertix is: {1}\n",
matrix.GetLength(0), matrix.GetLength(0) + 1);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Вивід у консоль



Завдання 2

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує побудову дерева за заданим кодом Прюфера (таблиця 3.2). Етапи розв'язання задачі виводити на екран.

Вимоги до програми

Програма має передбачати такі можливості:

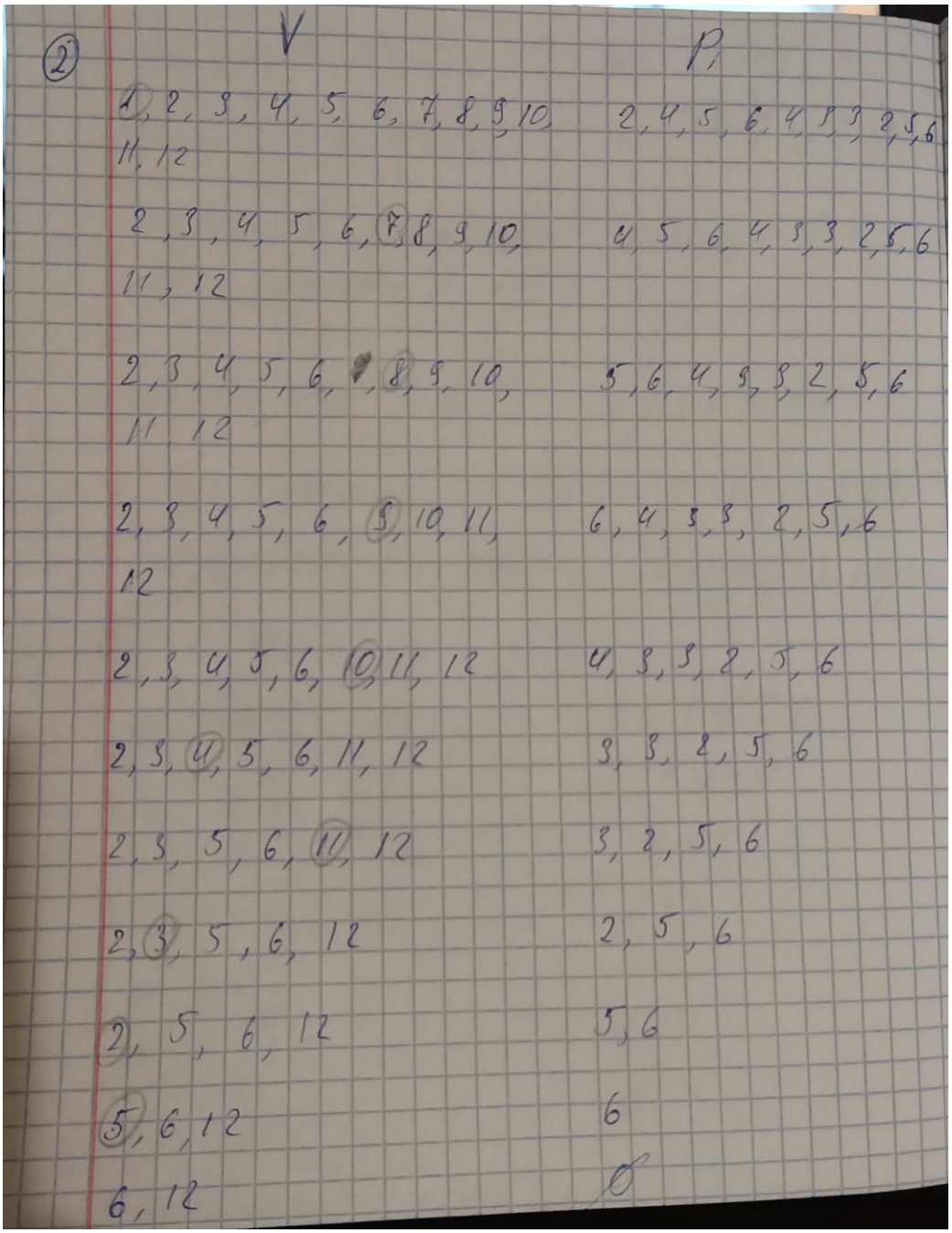
- введення вхідних даних вручну (коду дерева);
- реалізацію алгоритму побудови дерева за заданим кодом Прюфера;
- виведення на екран результату виконання програми (ребра побудованого дерева); перевірку на некоректне введення даних.

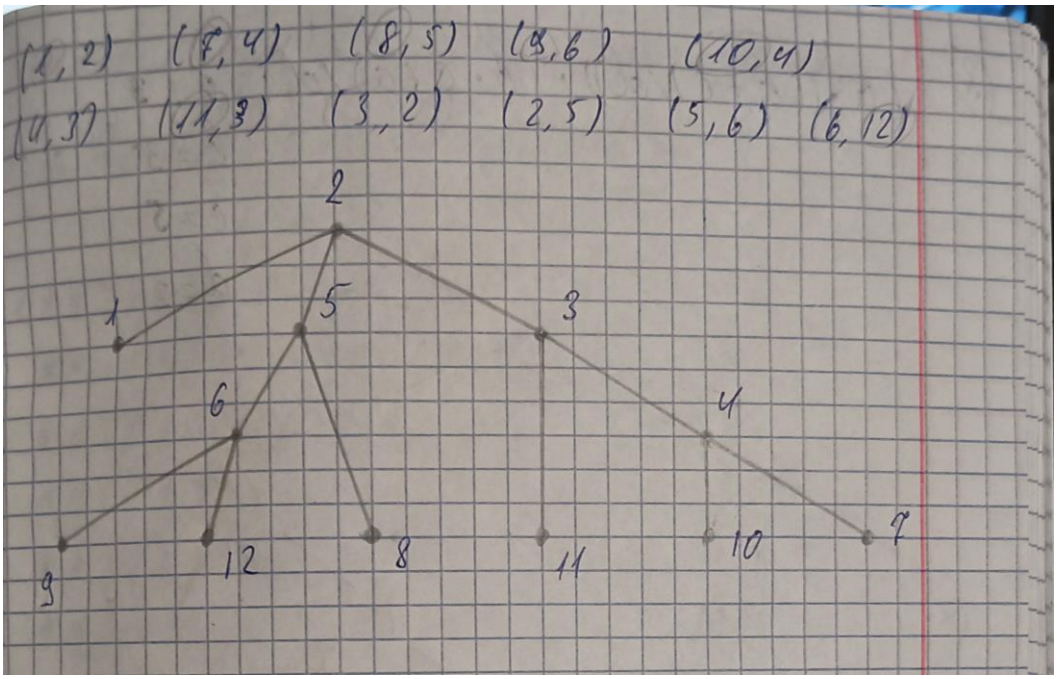
Завдання вважається зарахованим, якщо при тестуванні програми в присутності викладача отримано правильний результат.

Таблиця 3.2

25	2,4,5,6,4,3,3,2,5,6

Зошит


②
V
P
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
2, 4, 5, 6, 4, 3, 3, 2, 5, 6
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
4, 5, 6, 4, 3, 3, 2, 5, 6
2, 3, 4, 5, 6, 8, 9, 10, 11, 12
5, 6, 4, 3, 3, 2, 5, 6
2, 3, 4, 5, 6, 9, 10, 11, 12
6, 4, 3, 3, 2, 5, 6
2, 3, 4, 5, 6, 10, 11, 12
4, 3, 3, 2, 5, 6
2, 3, 4, 5, 6, 11, 12
3, 3, 2, 5, 6
2, 3, 5, 6, 12
3, 2, 5, 6
2, 3, 5, 6, 12
2, 5, 6
2, 5, 6, 12
5, 6
5, 6, 12
6
6, 12
0



Реалізація у коді

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KDM_Lab03
{
    static class Task02
    {
        public static void SecondTask()
        {
            int vNum;
            int badCounter = 0;
            int EdgesIndex = 0;
            while (true)
            {
                try
                {
                    Console.ForegroundColor = ConsoleColor.Blue;
                    Console.Write("\nPlease, enter number of vertices: ");
                    Console.ForegroundColor = ConsoleColor.Cyan;
                    vNum = Convert.ToInt32(Console.ReadLine());
                    Console.ForegroundColor = ConsoleColor.Gray;

                    break;
                }
                catch (FormatException)
                {
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("\nPlease, enter numbers only!\n");
                    Console.ForegroundColor = ConsoleColor.Gray;
                }
            }

            int[] v = new int[vNum];
            int[] p = new int[vNum - 2];
            //int[] p = { 2, 4, 5, 6, 4, 3, 3, 2, 5, 6 };
```

```
            int[,] edges = new int[vNum - 1, 2];

            Console.WriteLine();
            for(int i = 0; i < vNum; i++)
            {
                v[i] = i + 1;

                if (i < vNum - 2)
                {

                    while (true)
                    {
                        try
                        {
                            Console.ForegroundColor = ConsoleColor.Blue;
                            Console.Write("\nPlease, enter {0} element of sequence: ", i
+ 1);
                            Console.ForegroundColor = ConsoleColor.Cyan;
                            p[i] = Convert.ToInt32(Console.ReadLine());
                            Console.ForegroundColor = ConsoleColor.Gray;

                            break;
                        }
                        catch (FormatException)
                        {
                            Console.ForegroundColor = ConsoleColor.Red;
                            Console.WriteLine("\nPlease, enter numbers only!\n");
                            Console.ForegroundColor = ConsoleColor.Gray;
                        }
                    }
                }
            }

            while (vNum - 2 != 0)
            {
                for (int i = 0; i < v.Length; i++)
                {
                    for (int j = 0; j < p.Length; j++)
                    {
                        if (v[i] == p[j])
                        {
                            badCounter++;
                            break;
                        }
                    }
                    if (badCounter == 0)
                    {
                        edges[EdgesIndex, 0] = v[i];
                        edges[EdgesIndex, 1] = p[0];
                        EdgesIndex++;

                        v = MatchArray(v, i);
                        p = MatchArray(p, 0);
                        vNum--;
                        break;
                    }
                    badCounter = 0;
                }
            }
            edges[EdgesIndex, 0] = v[0];
            edges[EdgesIndex, 1] = v[1];
            ShowMatrix(edges);
        }

        static int[] MatchArray(int[] arr, int index)
        {
            int[] tempArr = new int [arr.Length - 1];
```

```
            for (int i = 0; i < index; i++)
                tempArr[i] = arr[i];

            for(int i = index; i < tempArr.Length; i++)
                tempArr [i] = arr [i + 1];

            return tempArr;
        }
        static void ShowMatrix(int[,] matrix)
        {
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.DarkYellow;

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                if(i % 5 == 0)
                    Console.WriteLine();
                Console.Write("{");
                for (int j = 0; j < 2; j++)
                {
                    Console.Write((char)(96 + matrix[i, j]));
                    if (j + 1 < 2)
                        Console.Write("    ");
                }
                Console.Write("}\t");
            }
            Console.WriteLine("\n\nNumber of edges is: {0}\nNumber of vertix is: {1}\n",
matrix.GetLength(0), matrix.GetLength(0) + 1);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Вивід у консоль

```
C:\Users\akmit\source\repos\
Please, Enter the number of task, you want to check or press "S" to stop process: 2

Please, enter number of vertices: 12


Please, enter 1 element of sequence: 2

Please, enter 2 element of sequence: 4

Please, enter 3 element of sequence: 5

Please, enter 4 element of sequence: 6

Please, enter 5 element of sequence: 4

Please, enter 6 element of sequence: 3

Please, enter 7 element of sequence: 3

Please, enter 8 element of sequence: 2

Please, enter 9 element of sequence: 5

Please, enter 10 element of sequence: 6


{a    b}        {g    d}        {h    e}        {i    f}        {j    d}
{d    c}        {k    c}        {c    b}        {b    e}        {e    f}
{f    l}

Number of edges is: 11
Number of vertix is: 12
```

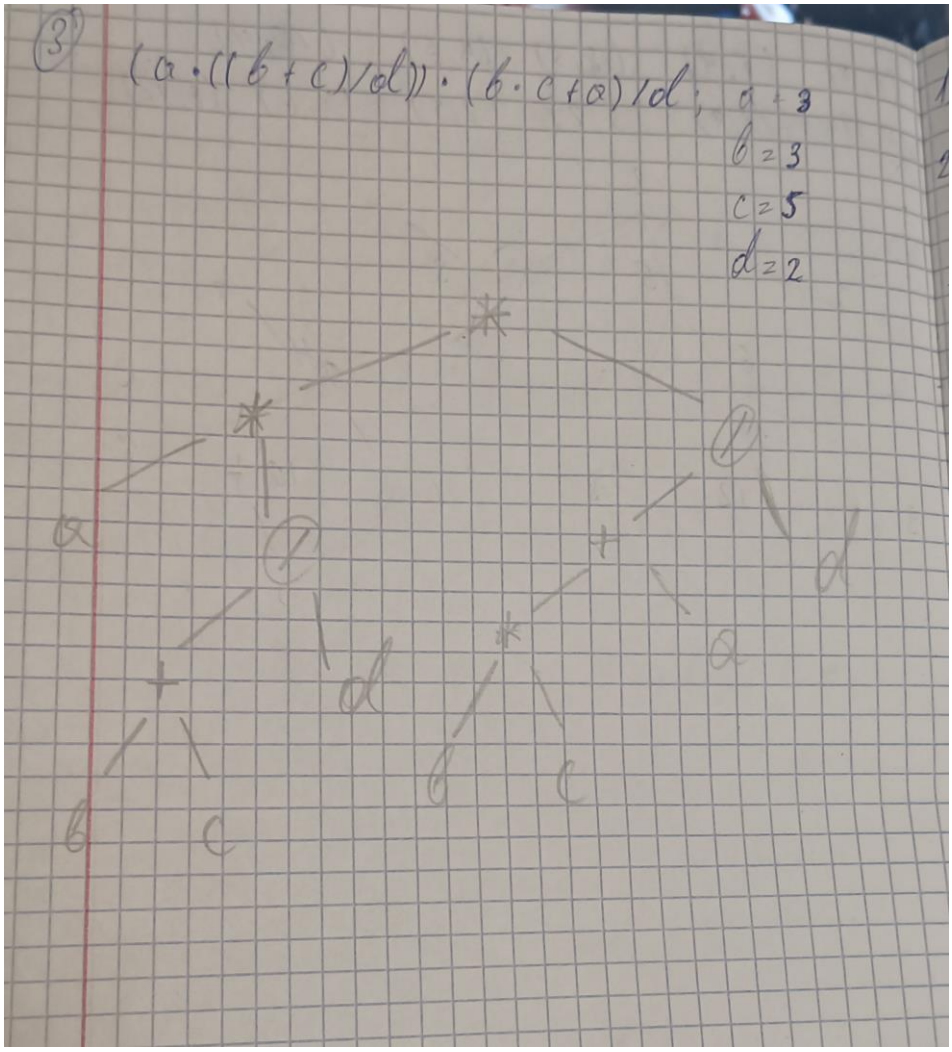
Завдання 3

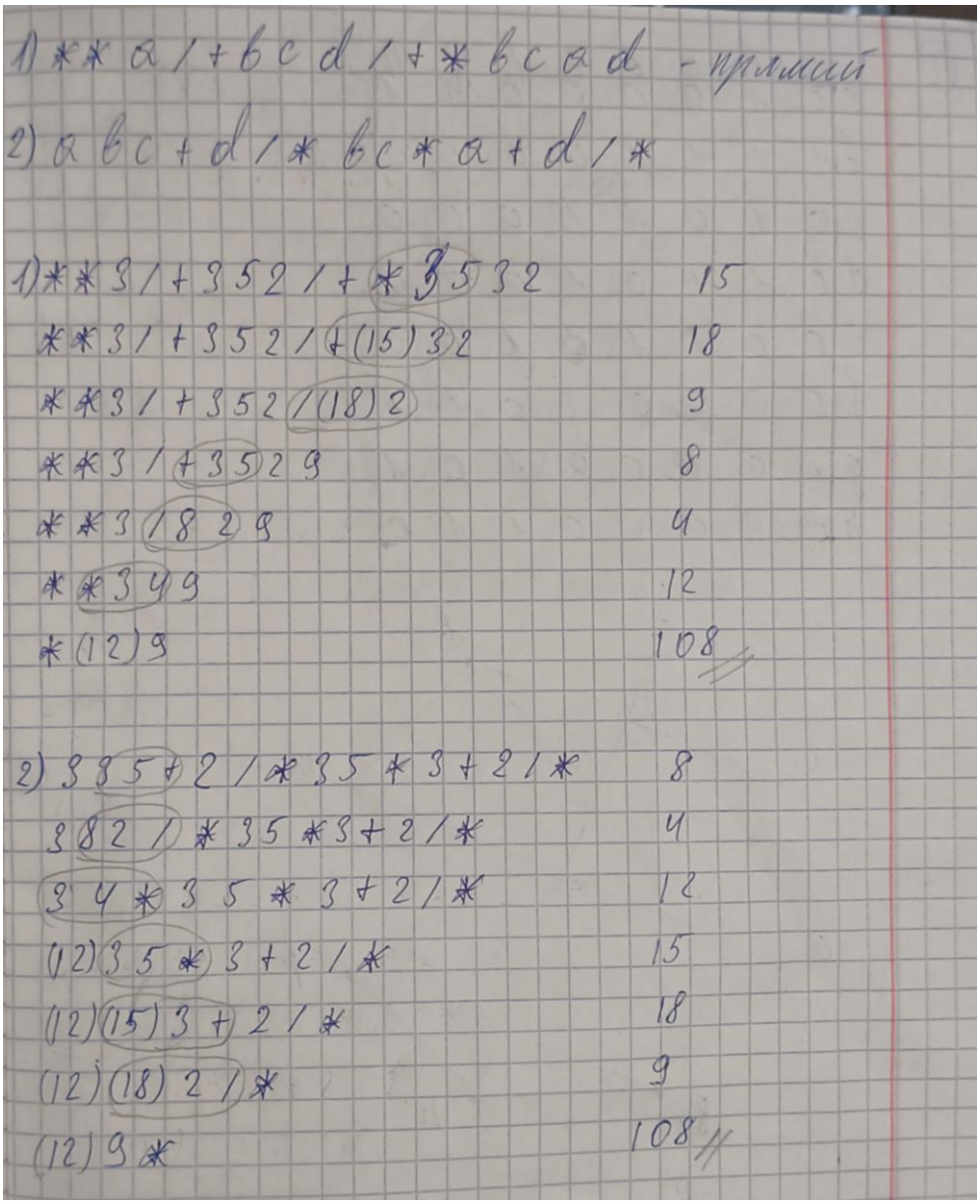
Подати у вигляді математичного (бінарного) дерева арифметичний вираз (таблиця 3.3). Отримати дві послідовності його обходу: префіксний (польський) та постфіксний (зворотний польський) записи. Відобразити динаміку обчислень та знайти значення виразів в обох цих записах.

Таблиця 3.3

25	(a*((b+c)/d))*(b*c+a)/d; a=3, b=3, c=5, d=2

Зошит





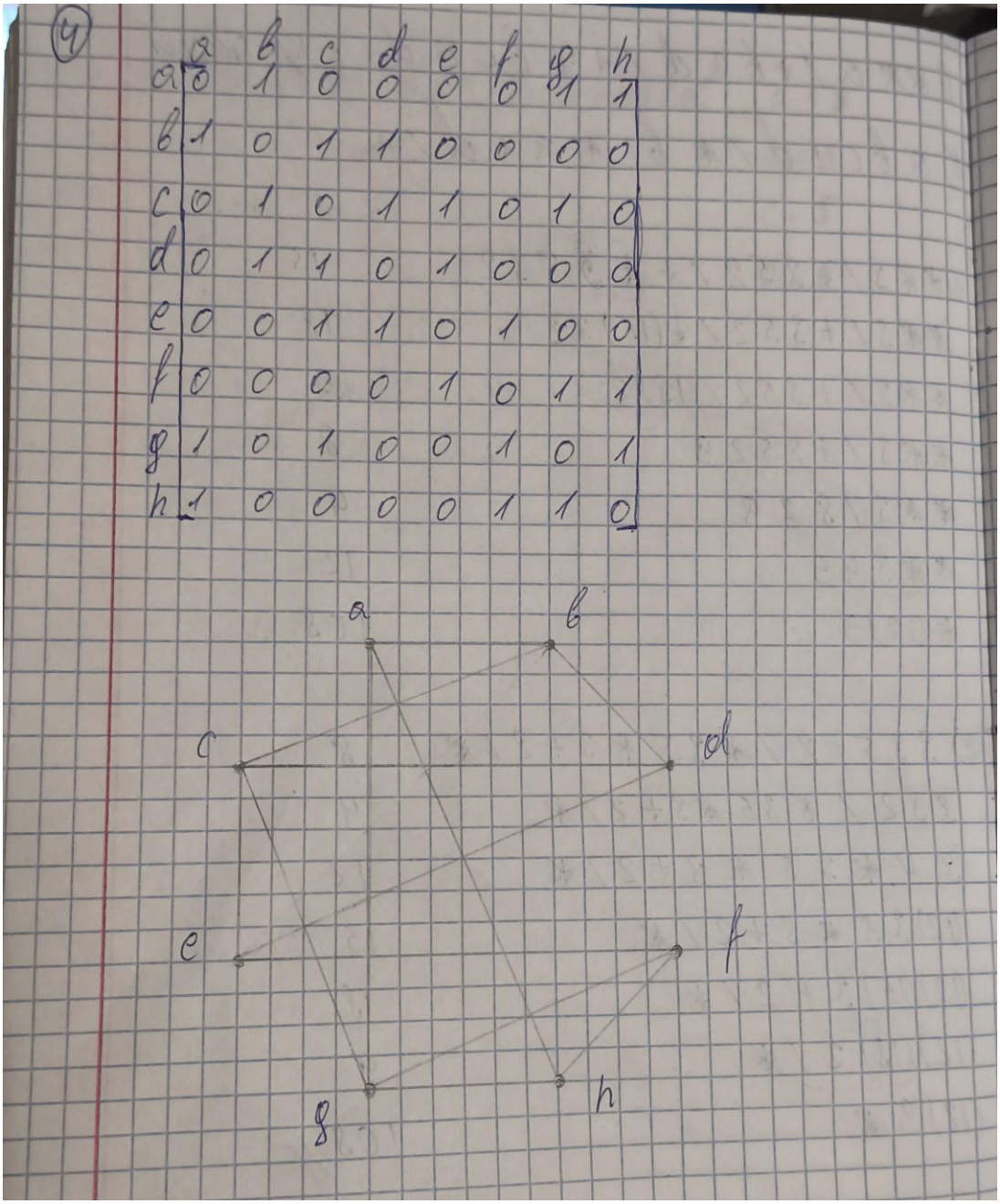
Завдання 4

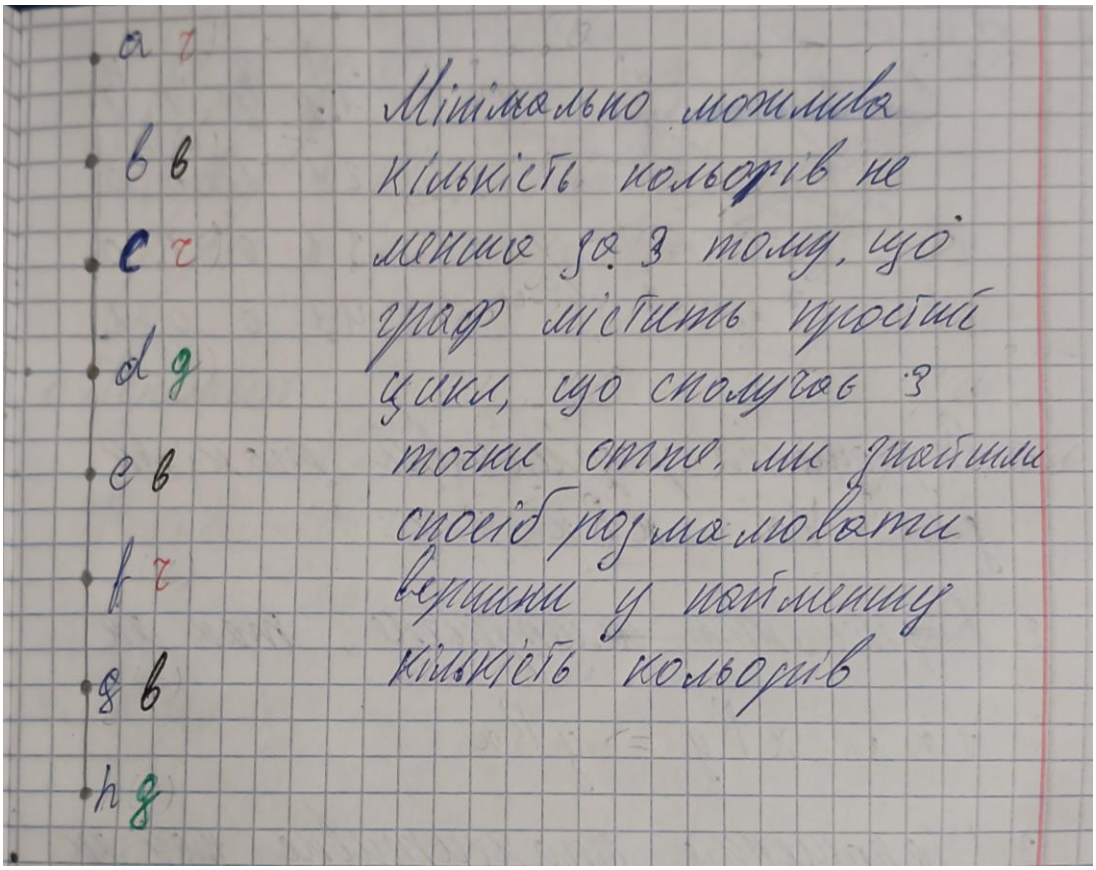
Розфарбувати граф, заданий матрицею суміжності (таблиця 3.4), мінімально можливою кількістю кольорів. Використати алгоритм бектрекінгу, результат подати у вигляді дерева.

Таблиця 3.4

25	$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$

Зошит


4
a b c d e f g h
a 0 1 0 0 0 0 1 1
b 1 0 1 1 0 0 0 0
c 0 1 0 1 1 0 1 0
d 0 1 1 0 1 0 0 0
e 0 0 1 1 0 1 0 0
f 0 0 0 0 1 0 1 1
g 1 0 1 0 0 1 0 1
h 1 0 0 0 0 1 1 0
a
b
c
d
e
f
g
h



Завдання 5

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує визначення властивостей бінарного відношення $R1$, заданого переліком елементів на множині A={1,2,3,4}, $R1 \subseteq A^2$ (таблиця 3.5). Вказати, чи є дане відношення відношенням еквівалентності, толерантності, порядку.

Програма має передбачати такі можливості:

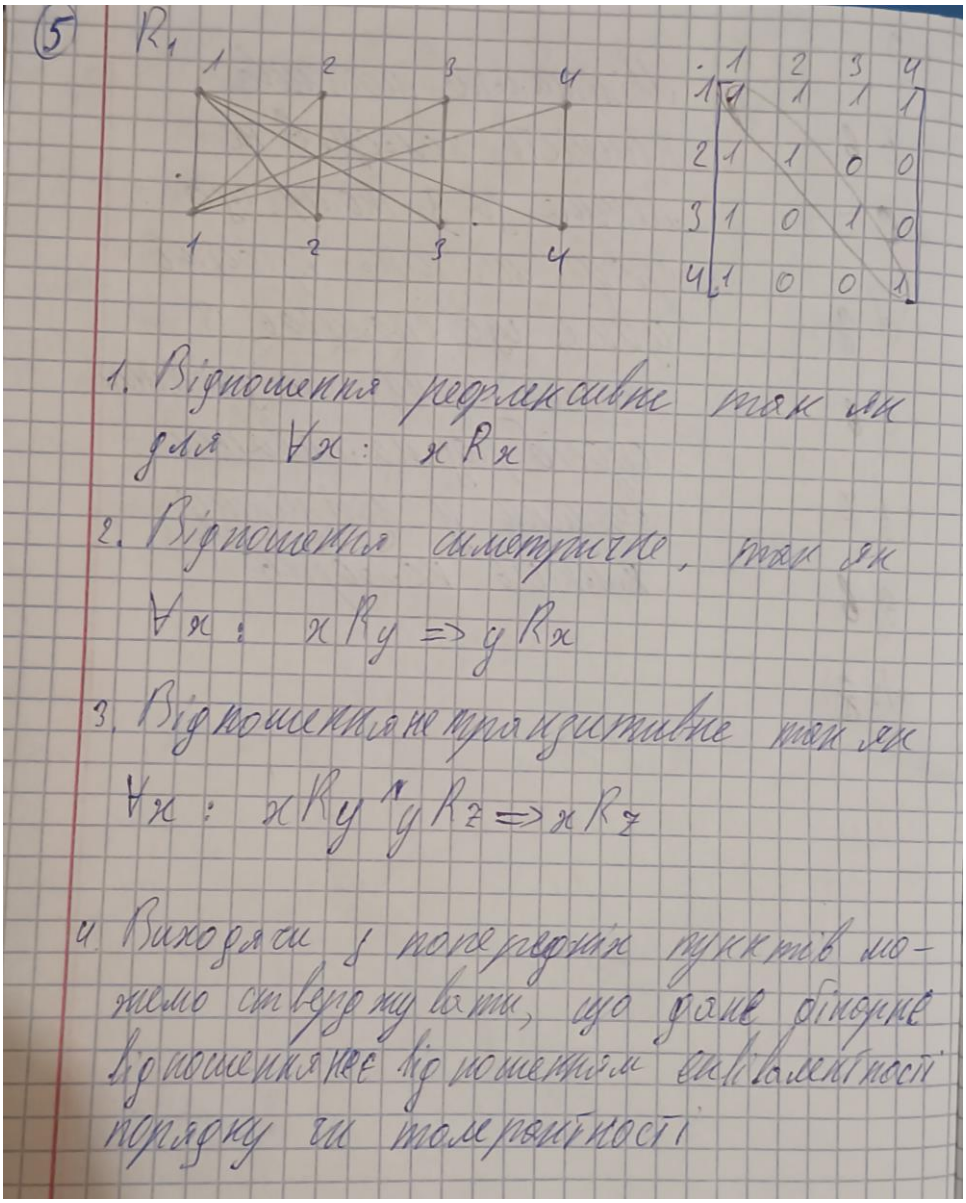
- введення вхідних даних вручну (задання відношення матрицею);
- реалізацію дослідження та визначення властивостей бінарного відношення;
- виведення результату досліджень властивостей заданого відношення та повідомлення, чи є дане відношення відношенням еквівалентності, толерантності, порядку;
- перевірку на некоректне введення даних.

Завдання вважається зарахованим, якщо при тестуванні програми в присутності викладача отримано правильний результат.

Таблиця 3.5

□25	R1={(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(3,1),(3,3),(4,1),(4,4)}

Зошит



Реалізація у коді

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.ExceptionServices;
using System.Text;
using System.Threading.Tasks;

namespace KDM_Lab03
{
    static class Task05
    {
        public static void FifthTask()
        {
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("Please, Enter matrix: ");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] R1 = GetMatrix();
```

```
            int[,] R1 =
            {
                {1,  1,  1,  1},
                {1,  1,  0,  0},
                {1,  0,  1,  0},
                {1,  0,  0,  1}
            };

            Console.WriteLine("Marix:\n");
            ShowMatrix(R1);

            int isReflective = IsReflective(R1);
            int isSymmetrical = IsSymmetrical(R1);
            int isTransitive = IsTransitive(R1);

            Console.ForegroundColor = ConsoleColor.Yellow;
            switch (isReflective)
            {
                case 0:
                    Console.WriteLine("Relation is neither reflective nor irreflective");
                    break;

                case -1:
                    Console.WriteLine("Relation is irreflective");
                    break;

                case 1:
                    Console.WriteLine("Relation is reflective");
                    break;

                default:

                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Something went completely wrong :(");
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    break;
            }

            switch (isSymmetrical)
            {
                case 0:
                    Console.WriteLine("Relation is neither symmetrical nor
antisymmetrical");
                    break;

                case -1:
                    Console.WriteLine("Relation is antisymmetrical");
                    break;

                case 1:
                    Console.WriteLine("Relation is symmetrical");
                    break;
                case -2:
                    Console.WriteLine("Relation is asymmetrical");
                    break;

                default:
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Something went completely wrong :(");
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    break;
            }

            switch (isTransitive)
            {
                case 0:
```

```
                    Console.WriteLine("Relation is neither transitive nor
antitransitive");
                    break;

                case -1:
                    Console.WriteLine("Relation is antitransitive");
                    break;

                case 1:
                    Console.WriteLine("Relation is transitive");
                    break;

                default:
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Something went completely wrong :(");
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    break;
                    Console.ForegroundColor = ConsoleColor.Gray;

            }

            Console.ForegroundColor = ConsoleColor.Yellow;

            if (isReflective == 1 && isSymmetrical == 1 && isTransitive == 1)
            {
                Console.WriteLine("\nThis is equivalence relation");
            }
            else if (isSymmetrical < -1 && isTransitive == 1)
            {
                Console.WriteLine("\nThis is order relation");
            }
            else if (isReflective == 1 && isSymmetrical == 1 && isTransitive == -1)
            {
                Console.WriteLine("\nThis is tolerance relation");
            }
            Console.ForegroundColor = ConsoleColor.Gray;
        }
        static int[,] GetMatrix()
        {
            int size;

            while (true)
            {
                try
                {
                    Console.ForegroundColor = ConsoleColor.Blue;
                    Console.Write("Please, enter matrix size NxN:\nN = ");
                    Console.ForegroundColor = ConsoleColor.Cyan;
                    size = Convert.ToInt32(Console.ReadLine());
                    Console.ForegroundColor = ConsoleColor.Gray;

                    break;
                }
                catch (FormatException)
                {

                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("\nPlease, enter numbers only!\n");
                    Console.ForegroundColor = ConsoleColor.Gray;
                }
            }
            Console.WriteLine();

            char[] l = new char[size];
            for (int i = 0; i < l.Length; i++)
            {
                l[i] = (char)(97 + i);
```

```
            }

            int[,] myMatrix = new int[size, size];
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    while (true)
                    {
                        Console.ForegroundColor = ConsoleColor.Blue;
                        Console.Write("Edge between \"{0}\" and \"{1}\": ", l[i], l[j]);
                        try
                        {
                            Console.ForegroundColor = ConsoleColor.Cyan;
                            myMatrix[i, j] = Convert.ToInt32(Console.ReadLine());
                            Console.ForegroundColor = ConsoleColor.Gray;

                            myMatrix[j, i] = myMatrix[i, j];
                            if (myMatrix[i, j] == 0 || myMatrix[i, j] == 1)
                                break;
                            else
                            {
                                Console.ForegroundColor = ConsoleColor.Red;
                                Console.WriteLine("\nPlease, enter only \"1\" or
\"0\"\n");

                                Console.ForegroundColor = ConsoleColor.Gray;
                            }
                        }
                        catch (FormatException)
                        {
                            Console.ForegroundColor = ConsoleColor.Red;
                            Console.WriteLine("\nPlease, only digits\n");
                            Console.ForegroundColor = ConsoleColor.Gray;
                        }

                    }
                }
            }

            return myMatrix;
        }
        static void ShowMatrix(int[,] matrix)
        {
            int edges = 0, vertices = matrix.GetLength(0);
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.DarkYellow;

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(0); j++)
                {
                    if(matrix[i, j] != 0)
                        edges++;
                    Console.Write(matrix[i, j] + "    ");
                }
                Console.WriteLine("\n");
            }
            Console.WriteLine("Number of edges is: {0}\nNumber of vertix is: {1}\n",
edges, vertices);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
        static int IsReflective(int[,] matrix)
        {
            int zeroCounter = 0, oneCounter = 0, result;
            for(int i = 0; i < matrix.GetLength(0); i++)
            {
                if (matrix[i,i] == 0)
```

```
                zeroCounter++;
            else if(matrix[i,i] == 1)
                oneCounter++;
        }

        if(zeroCounter == matrix.GetLength(0))
        {
            result = -1;
        }
        else if(oneCounter == matrix.GetLength(0))
        {
            result = 1;
        }
        else
        {
            result = 0;
        }

        return result;
    }
    static int IsSymmetrical(int[,] matrix)
    {
        int symmetry = 0, antisymmetry = 0, result;
        for (int i = 0; i < matrix.GetLength(0); i++)
            for (int j = 0; j < matrix.GetLength(0); j++)
            {
                if (i >= j)
                    continue;

                if (matrix[i, j] == matrix[j, i])
                    symmetry++;
                else
                    antisymmetry++;
            }

        if (symmetry == ((matrix.GetLength(0) * (matrix.GetLength(0) - 1)) / 2))
            result = 1;
        else if (antisymmetry == ((matrix.GetLength(0) * (matrix.GetLength(0) - 1)) /
2) && IsReflective(matrix) == -1)
            result = -2;
        else if (antisymmetry == ((matrix.GetLength(0) * (matrix.GetLength(0) - 1)) /
2))
            result = -1;
        else
            result = 0;
        return result;
    }
    static int IsTransitive(int[,] matrix)
    {
        int transitivity = 1, ones = 1, result;

        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            for (int j = 0; j < matrix.GetLength(0); j++)
            {
                if (matrix[i, j] != 0)
                {
                    for (int k = 0; k < matrix.GetLength(0); k++)
                    {
                        if (matrix[j, k] != 0)
                        {
                            ones--;

                            if(matrix[i, k] == 0)
                                transitivity--;
                        }
                    }
```

```
                    }
                }
            }

            if (transitivity == 1)
                result = 1;
            else if (transitivity == ones)
                result = -1;
            else
                result = 0;

            return result;
        }
    }
}
```

Вивід у консоль

```
Please, Enter the number of task, you want to check or press "S" to stop process: 5
Marix:


1    1    1    1

1    1    0    0

1    0    1    0

1    0    0    1

Number of edges is: 10
Number of vertix is: 4

Relation is reflective
Relation is symmetrical
Relation is neither transitive nor antitransitive
```

Завдання 6

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує обчислення матриць перетину, об'єднання, різниці, симетричної різниці, доповнення, обернених, композицій $R_1 \circ R_2$, $R_2 \circ R_1$ двох бінарних відношень, заданих на множині $A = \{a, b, c\}$ (таблиця 3.6). У випадку подання відношень переліком елементів спочатку записати їх матрицями.

Програма має передбачати такі можливості:

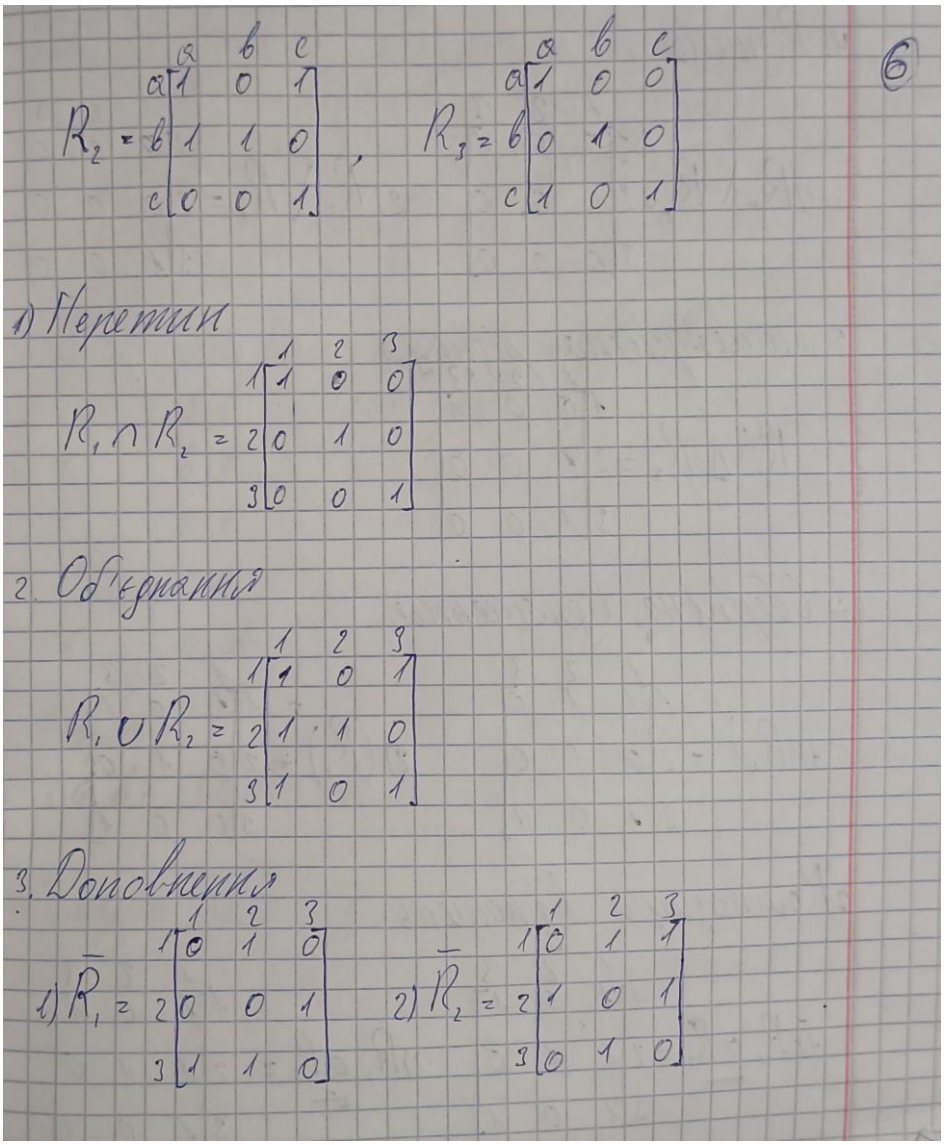
- введення вхідних даних вручну (задання відношень матрицями);
- реалізацію обчислення та виведення матриць перетину, об'єднання, різниці, симетричної різниці, доповнення, обернених, композицій двох бінарних відношень; перевірку на некоректне введення даних.

Завдання вважається зарахованим, якщо при тестуванні програми в присутності викладача отримано правильний результат.

Таблиця 3.6

25	$R_2=\begin{pmatrix}1&0&1\\1&1&0\\0&0&1\end{pmatrix}$, $R_3=\begin{pmatrix}1&0&0\\0&1&0\\1&0&1\end{pmatrix}$

Зошит



4. Різниця

1) $R_1 \setminus R_2 =$

	1	2	3
1	0	0	1
2	1	0	0
3	0	0	0

2) $R_2 \setminus R_1 =$

	1	2	3
1	0	0	0
2	0	0	0
3	1	0	0

5. Симетрична різниця

$R_1 \Delta R_2 =$

	1	2	3
1	0	0	1
2	1	0	0
3	1	0	0

6. Обернене відношення

1) $(R_1)^{-1} =$

	1	2	3
1	1	1	0
2	0	1	0
3	1	0	1

2) $(R_2)^{-1} =$

	1	2	3
1	1	0	1
2	0	1	0
3	0	0	1

7) Композиція відношень

1) $R_1 \underset{\leftarrow}{\circ} R_2 =$

	1	2	3
1	1	0	1
2	1	1	0
3	1	0	1

2) $R_2 \underset{\leftarrow}{\circ} R_1 =$

	1	2	3
1	1	0	1
2	1	1	0
3	1	0	1

Реалізація у коді

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.ExceptionServices;
using System.Text;
using System.Threading.Tasks;

namespace KDM_Lab03
{
    static class Task05
    {
        public static void FifthTask()
        {
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("Please, Enter matrix: ");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] R1 = GetMatrix();

            int[,] R1 =
            {
                {1,  1,  1,  1},
                {1,  1,  0,  0},
                {1,  0,  1,  0},
                {1,  0,  0,  1}
            };

            Console.WriteLine("Marix:\n");
            ShowMatrix(R1);

            int isReflective = IsReflective(R1);
            int isSymmetrical = IsSymmetrical(R1);
            int isTransitive = IsTransitive(R1);

            Console.ForegroundColor = ConsoleColor.Yellow;
            switch (isReflective)
            {
                case 0:
                    Console.WriteLine("Relation is neither reflective nor irreflective");
                    break;

                case -1:
                    Console.WriteLine("Relation is irreflective");
                    break;

                case 1:
                    Console.WriteLine("Relation is reflective");
                    break;

                default:

                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Something went completely wrong :(");
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    break;
            }

            switch (isSymmetrical)
            {
                case 0:
                    Console.WriteLine("Relation is neither symmetrical nor
antisymmetrical");
                    break;
```

```
                case -1:
                    Console.WriteLine("Relation is antisymmetrical");
                    break;

                case 1:
                    Console.WriteLine("Relation is symmetrical");
                    break;
                case -2:
                    Console.WriteLine("Relation is asymmetrical");
                    break;

                default:
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Something went completely wrong :(");
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    break;
            }

            switch (isTransitive)
            {
                case 0:
                    Console.WriteLine("Relation is neither transitive nor
antitransitive");
                    break;

                case -1:
                    Console.WriteLine("Relation is antitransitive");
                    break;

                case 1:
                    Console.WriteLine("Relation is transitive");
                    break;

                default:
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Something went completely wrong :(");
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    break;
                    Console.ForegroundColor = ConsoleColor.Gray;

            }

            Console.ForegroundColor = ConsoleColor.Yellow;

            if (isReflective == 1 && isSymmetrical == 1 && isTransitive == 1)
            {
                Console.WriteLine("\nThis is equivalence relation");
            }
            else if (isSymmetrical < -1 && isTransitive == 1)
            {
                Console.WriteLine("\nThis is order relation");
            }
            else if (isReflective == 1 && isSymmetrical == 1 && isTransitive == -1)
            {
                Console.WriteLine("\nThis is tolerance relation");
            }
            Console.ForegroundColor = ConsoleColor.Gray;
        }
        static int[,] GetMatrix()
        {
            int size;

            while (true)
            {
                try
                {
                    Console.ForegroundColor = ConsoleColor.Blue;
```

```
                Console.Write("Please, enter matrix size NxN:\nN = ");
                Console.ForegroundColor = ConsoleColor.Cyan;
                size = Convert.ToInt32(Console.ReadLine());
                Console.ForegroundColor = ConsoleColor.Gray;

                break;
            }
            catch (FormatException)
            {

                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("\nPlease, enter numbers only!\n");
                Console.ForegroundColor = ConsoleColor.Gray;
            }
        }
        Console.WriteLine();

        char[] l = new char[size];
        for (int i = 0; i < l.Length; i++)
        {
            l[i] = (char)(97 + i);
        }

        int[,] myMatrix = new int[size, size];
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                while (true)
                {
                    Console.ForegroundColor = ConsoleColor.Blue;
                    Console.Write("Edge between \"{0}\" and \"{1}\": ", l[i], l[j]);
                    try
                    {
                        Console.ForegroundColor = ConsoleColor.Cyan;
                        myMatrix[i, j] = Convert.ToInt32(Console.ReadLine());
                        Console.ForegroundColor = ConsoleColor.Gray;

                        myMatrix[j, i] = myMatrix[i, j];
                        if (myMatrix[i, j] == 0 || myMatrix[i, j] == 1)
                            break;
                        else
                        {
                            Console.ForegroundColor = ConsoleColor.Red;
                            Console.WriteLine("\nPlease, enter only \"1\" or
\"0\"\n");
                            Console.ForegroundColor = ConsoleColor.Gray;
                        }
                    }
                    catch (FormatException)
                    {
                        Console.ForegroundColor = ConsoleColor.Red;
                        Console.WriteLine("\nPlease, only digits\n");
                        Console.ForegroundColor = ConsoleColor.Gray;
                    }

                }
            }
        }

        return myMatrix;
    }
    static void ShowMatrix(int[,] matrix)
    {
        int edges = 0, vertices = matrix.GetLength(0);
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkYellow;
```

```
            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(0); j++)
                {
                    if(matrix[i, j] != 0)
                        edges++;
                    Console.Write(matrix[i, j] + "    ");
                }
                Console.WriteLine("\n");
            }
            Console.WriteLine("Number of edges is: {0}\nNumber of vertix is: {1}\n",
edges, vertices);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
        static int IsReflective(int[,] matrix)
        {
            int zeroCounter = 0, oneCounter = 0, result;
            for(int i = 0; i < matrix.GetLength(0); i++)
            {
                if (matrix[i,i] == 0)
                    zeroCounter++;
                else if(matrix[i,i] == 1)
                    oneCounter++;
            }

            if(zeroCounter == matrix.GetLength(0))
            {
                result = -1;
            }
            else if(oneCounter == matrix.GetLength(0))
            {
                result = 1;
            }
            else
            {
                result = 0;
            }

            return result;
        }
        static int IsSymmetrical(int[,] matrix)
        {
            int symmetry = 0, antisymmetry = 0, result;
            for (int i = 0; i < matrix.GetLength(0); i++)
                for (int j = 0; j < matrix.GetLength(0); j++)
                {
                    if (i >= j)
                        continue;

                    if (matrix[i, j] == matrix[j, i])
                        symmetry++;
                    else
                        antisymmetry++;
                }

            if (symmetry == ((matrix.GetLength(0) * (matrix.GetLength(0) - 1)) / 2))
                result = 1;
            else if (antisymmetry == ((matrix.GetLength(0) * (matrix.GetLength(0) - 1)) /
2) && IsReflective(matrix) == -1)
                result = -2;
            else if (antisymmetry == ((matrix.GetLength(0) * (matrix.GetLength(0) - 1)) /
2))
                result = -1;
            else
                result = 0;
            return result;
```

```
        }
        static int IsTransitive(int[,] matrix)
        {
            int transitivity = 1, ones = 1, result;

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(0); j++)
                {
                    if (matrix[i, j] != 0)
                    {
                        for (int k = 0; k < matrix.GetLength(0); k++)
                        {
                            if (matrix[j, k] != 0)
                            {
                                ones--;

                                if(matrix[i, k] == 0)
                                    transitivity--;
                            }
                        }
                    }
                }
            }

            if (transitivity == 1)
                result = 1;
            else if (transitivity == ones)
                result = -1;
            else
                result = 0;

            return result;
        }
    }
}
```

Вивід у консоль

```
Please, Enter the number of task, you want to check or press "S" to stop process: 6


Intersection:

1    0    0

0    1    0

0    0    1

Number of edges is: 3
Number of vertix is: 3



Union:

1    0    1

1    1    0

1    0    1

Number of edges is: 6
Number of vertix is: 3



Complemence to R1:

0    1    0

0    0    1

1    1    0

Number of edges is: 4
Number of vertix is: 3
```

```
Complemence to R2:

0    1    1

1    0    1

0    1    0

Number of edges is: 5
Number of vertix is: 3



Difference R1 \ R2:

0    0    1

1    0    0

0    0    0

Number of edges is: 2
Number of vertix is: 3



Difference R2 \ R1:

0    0    0

0    0    0

1    0    0

Number of edges is: 1
Number of vertix is: 3
```

```
Difference in both ways (R1 \ R2) \/ (R2 \ R1):

0    0    1

1    0    0

1    0    0

Number of edges is: 3
Number of vertix is: 3


Transposed R1:

1    1    0

0    1    0

1    0    1

Number of edges is: 5
Number of vertix is: 3


Transposed R2:

1    0    1

0    1    0

0    0    1

Number of edges is: 4
Number of vertix is: 3
```

```
Composition R1 o R2:

1    0    1

1    1    0

1    0    1

Number of edges is: 6
Number of vertix is: 3


Composition R2 o R1:

1    0    1

1    1    0

1    0    1

Number of edges is: 6
Number of vertix is: 3
```

Висновок

Під час виконання лабораторної роботи я ознайомився на практиці з алгоритмом Краскала побудовою мінімального остовного дерева, з алгоритмом побудови дерева за заданим кодом Прюфера, з поданням арифметичних виразів у вигляді математичного (бінарного) дерева та польськими записами, навчився обчислювати значення виразів в обох цих записах, освоїв алгоритм бектрекінгу для розфарбування графа мінімально можливою кількістю кольорів. Зміг реалізувати обчислення матриць операцій над двома бінарними відношеннями, а також навчився визначати властивості бінарного відношення.