

Міністерство освіти і науки України
Національний університет "Львівська політехніка"
Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення



Звіт

Про виконання лабораторної роботи №2

На тему:

«Основні поняття та застосування теорії графів»

з дисципліни

«Комп'ютерна дискретна математика»

Лектор:

Журавчак Л.М.

Виконав:

ст. гр. ПЗ-18

Юшкевич А.І.

Прийняв:

Курапов П.Р.

« -- » -- 2022 р.

Σ = _____

Тема: Основні поняття та застосування теорії графів

Мета: Ознайомитись на практиці із основними поняттями теорії неорієнтованих та орієнтованих графів, навчитись виконувати операції над графами, будувати матриці досяжності, знаходити у зваженому графі субоптимальний гамільтоновий цикл, засвоїти алгоритми пошуку вглиб і вшир та алгоритм Дейкстри.

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)

```
namespace KDM_Lab01
{
    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 3:
                        Task03.ThirdTask();
                        break;
                    case 4:
                        Task04.FourthTask();
                        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\n\"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 № 2

Завдання 1

Граф $G=(V,E)$ задано матрицею суміжності (таблиця 2.1).

Вказати кількість вершин і ребер у графі-доповненні до графа G , у графі-перетині $G \cap H$ та у графі-об'єднанні $G \cup H$. Граф H – варіант $k+1$.

Написати програму (на будь-якій відомій студентів мові програмування), яка реалізує обчислення матриць суміжності графа-доповнення, графа-перетину та графа-об'єднання.

Програма має передбачати такі можливості:

- введення вхідних даних вручну (матриці суміжності);
- реалізацію обчислення та виведення матриць суміжності графа-доповнення, графа-перетину та графа-об'єднання;
- виведення кількості вершин і ребер вказаних графів;
- перевірку на некоректне введення даних.

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

Таблица 2.1

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

Handwritten notes on graph theory, showing two graphs G and H with their adjacency matrices and vertex/edge counts.

Graph G:

$$G = \begin{pmatrix}
 a & b & c & d & e & f & g & h \\
 \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{matrix} & \begin{pmatrix}
 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0
 \end{pmatrix}
 \end{pmatrix}$$

$|V| = 8$
 $|E|_{2m} = 13$

Graph H:

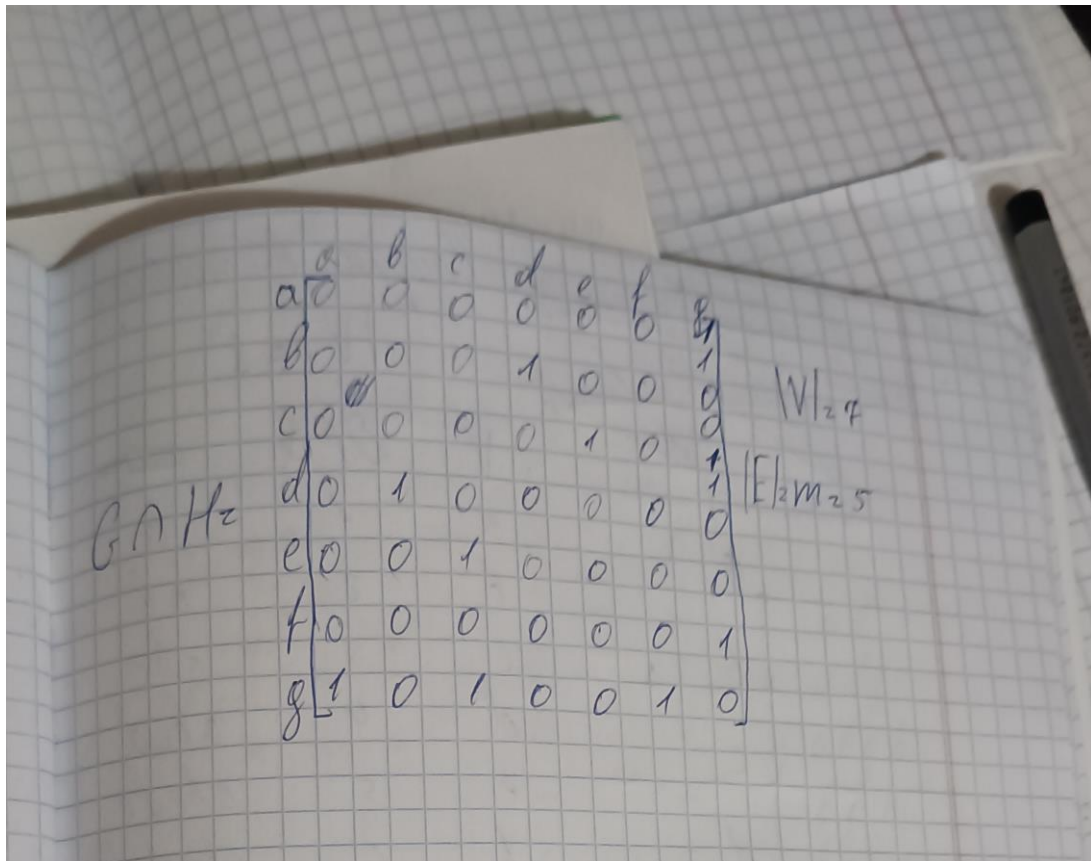
$$H = \begin{pmatrix}
 a & b & c & d & e & f & g \\
 \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{matrix} & \begin{pmatrix}
 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
 1 & 1 & 1 & 1 & 1 & 1 & 0
 \end{pmatrix}
 \end{pmatrix}$$

$|V| = 7$ (seven)
 $|E|_{2m} = 12$ (twelve)

$$\bar{G} = \begin{bmatrix} a & b & c & d & e & f & g & h \\ a & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ c & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ d & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ e & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ f & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ g & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ h & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \quad |V| = 8 \quad |E|_{zm} = 15$$

$$G \cup H = \begin{bmatrix} a & b & c & d & e & f & g & h \\ a & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ b & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ c & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ d & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ e & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ f & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ g & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ h & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$|V| = 8; |E|_{zm} = 20$$



Реалізація у коді

```
namespace KDM_Lab02
{
    static class Task01
    {
        public static void FirstTask()
        {
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("\nEnter first matrix: \n");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] myMatrix = GetMatrix();
            int[,] myMatrix =
            {
                {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 }
            };
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("Enter second matrix: \n");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] secMatrix = GetMatrix();

            int[,] secMatrix =
            {
                {0, 0, 1, 0, 1, 0, 1 },
            }
        }
    }
}
```

```

        {0, 0, 0, 1, 0, 1, 1 },
        {1, 0, 0, 0, 1, 0, 1 },
        {0, 1, 0, 0, 0, 1, 1 },
        {1, 0, 1, 0, 0, 0, 1 },
        {0, 1, 0, 1, 0, 0, 1 },
        {1, 1, 1, 1, 1, 1, 0 }
    };

    int[,] myMatrixComplence = FindComplence(myMatrix);
    int[,] matrixesAssociation = FindGraphsUnion(myMatrix, secMatrix);
    int[,] matrixIntersection = FindGraphsIntersection(myMatrix, secMatrix);

    Console.ForegroundColor = ConsoleColor.Blue;
    Console.WriteLine("First matrix: ");
    ShowMatrix(myMatrix);

    Console.ForegroundColor = ConsoleColor.Blue;
    Console.WriteLine("Second matrix: ");
    ShowMatrix(secMatrix);

    Console.ForegroundColor = ConsoleColor.Blue;
    Console.WriteLine("First matrix Complence: ");
    ShowMatrix(myMatrixComplence);

    Console.ForegroundColor = ConsoleColor.Blue;
    Console.WriteLine("Association of first and second matrixes: ");
    ShowMatrix(matrixesAssociation);

    Console.ForegroundColor = ConsoleColor.Blue;
    Console.WriteLine("Intersection of first and second matrixes: ");
    ShowMatrix(matrixIntersection);

    Console.ForegroundColor = ConsoleColor.Gray;
}
static int[,] FindGraphsIntersection(int[,] myMatrix, int[,] secMatrix)
{
    int myMtrxLength = myMatrix.GetLength(0);
    int secMtrxLength = secMatrix.GetLength(0);
    int[,] result;

    if (myMtrxLength > secMtrxLength)
    {
        result = new int[secMtrxLength, secMtrxLength];
    }
    else
    {
        result = new int[myMtrxLength, myMtrxLength];
    }

    for (int i = 0; i < result.GetLength(0); i++)
    {
        for (int j = 0; j < result.GetLength(0); j++)
        {
            if (j <= i) continue;

            if (myMatrix[i, j] == 1 && secMatrix[i, j] == 1)
            {
                result[i, j] = 1;
                result[j, i] = 1;
            }
        }
    }

    return result;
}

```



```

static int[,] FindGraphsUnion(int[,] myMatrix, int[,] secMatrix)
{
    int myMtrxLength = myMatrix.GetLength(0);
    int secMtrxLength = secMatrix.GetLength(0);
    int[,] result, smaller, bigger;

    if (myMtrxLength > secMtrxLength)
    {
        result = new int[myMtrxLength, myMtrxLength];
        smaller = secMatrix;
        bigger = myMatrix;
    }
    else
    {
        result = new int[secMtrxLength, secMtrxLength];
        smaller = myMatrix;
        bigger = secMatrix;
    }

    for (int i = 0; i < result.GetLength(0); i++)
    {
        for (int j = 0; j < result.GetLength(0); j++)
        {
            if (j <= i) continue;

            if (j >= smaller.GetLength(0) || i >= smaller.GetLength(0))
            {
                result[i, j] = bigger[i, j];
                result[j, i] = result[i, j];
                continue;
            }

            if (myMatrix[i, j] == 1 || secMatrix[i, j] == 1)
            {
                result[i, j] = 1;
                result[j, i] = 1;
            }
        }
    }

    return result;
}

static int[,] FindComplement(int[,] matrix)
{
    int len = matrix.GetLength(0);
    int[,] result = new int[len, len];

    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            if (j <= i) continue;

            result[i, j] = matrix[i, j] == 0 ? 1 : 0;
            result[j, i] = result[i, j];
        }
    }

    return result;
}

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 void ShowMatrix(int[,] matrix)
{
    int edges = 0, vertex = 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 (j >= i && matrix[i, j] != 0)
                edges++;

            Console.Write(matrix[i, j] + "    ");
        }
        Console.WriteLine("\n");
    }
    Console.WriteLine("Number of edges is: {0}\nNumber of vertex is: {1}\n",
edges, vertex);
    Console.ForegroundColor = ConsoleColor.Gray;
}
}

```

Вивід у консоль

First matrix Compliment:

0	0	1	1	1	1	0	0
0	0	0	0	1	1	1	1
1	0	0	0	0	1	0	1
1	0	0	0	0	1	1	1
1	1	0	0	0	0	1	1
1	1	1	1	0	0	0	0
0	1	0	1	1	0	0	0
0	1	1	1	1	0	0	0

Number of edges is: 15

Number of vertex is: 8

Association of first and second matrixes:

0	1	1	0	1	0	1	1
1	0	1	1	0	1	1	0
1	1	0	1	1	0	1	0
0	1	1	0	1	1	1	0
1	0	1	1	0	1	1	0
0	1	0	1	1	0	1	1
1	1	1	1	1	1	0	1
1	0	0	0	0	1	1	0

Number of edges is: 20

Number of vertex is: 8

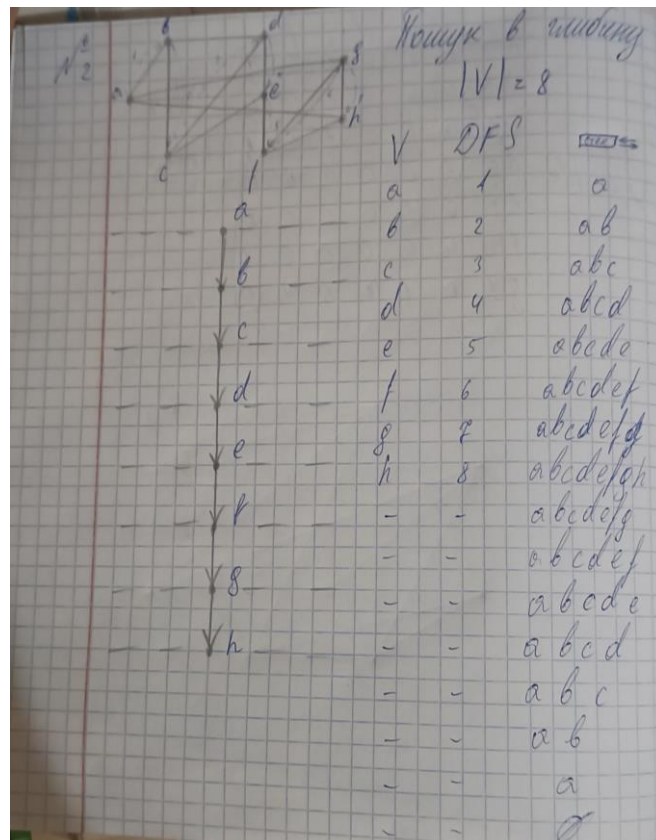
0	0	0	0	0	0	1
0	0	0	1	0	0	0
0	0	0	0	1	0	1
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	1
1	0	1	0	0	1	0

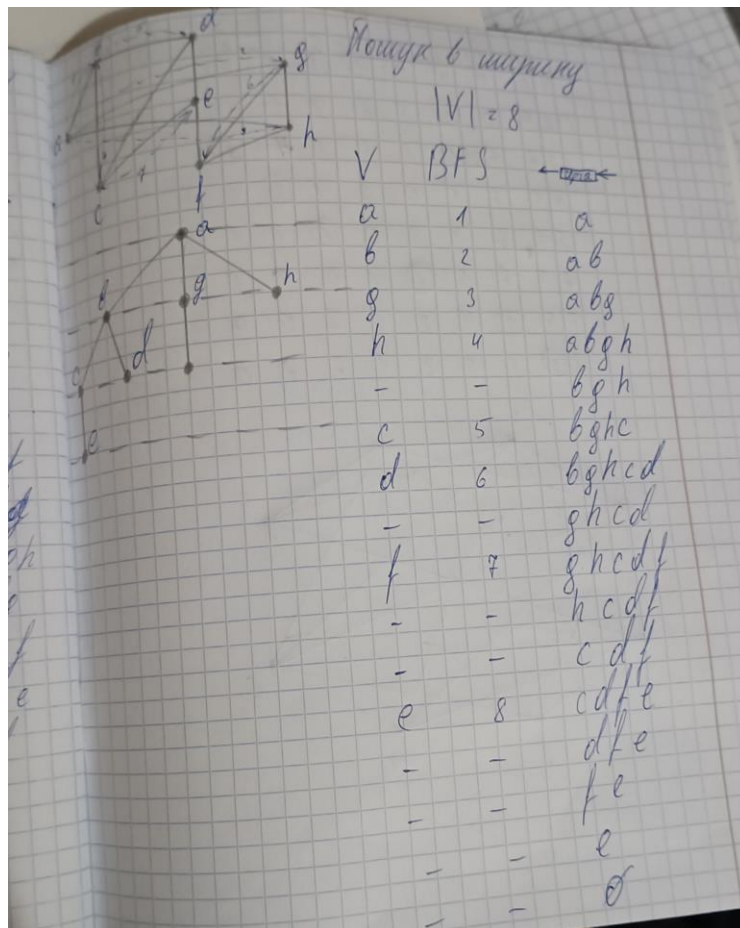
```
Number of edges is: 5
Number of vertex is: 7
```

Завдання 2

Граф $G=(V,E)$ задано матрицею суміжності (таблиця 2.1). Здійснити пошук углиб та пошук ушир у графі G , починаючи з першої вершини. Записати протоколи обходу. Під час обходу при виборі наступної вершини обов'язково брати найменшу відповідно до лексикографічного порядку.

Зошит





Завдання 3

Знайти гамільтоновий цикл (ГЦ) у зваженому графі, який заданий ваговою матрицею (таблиця 2.2), взявши за вихідну вершину 1. Використайте алгоритм найближчого сусіда.

Написати програму (на будь-якій відомій студентів мові програмування), яка реалізує алгоритм найближчого сусіда для пошуку ГЦ у зваженому графі (таблиця 2.2), взявши за вихідну вершину 1.

Програма має передбачати такі можливості:

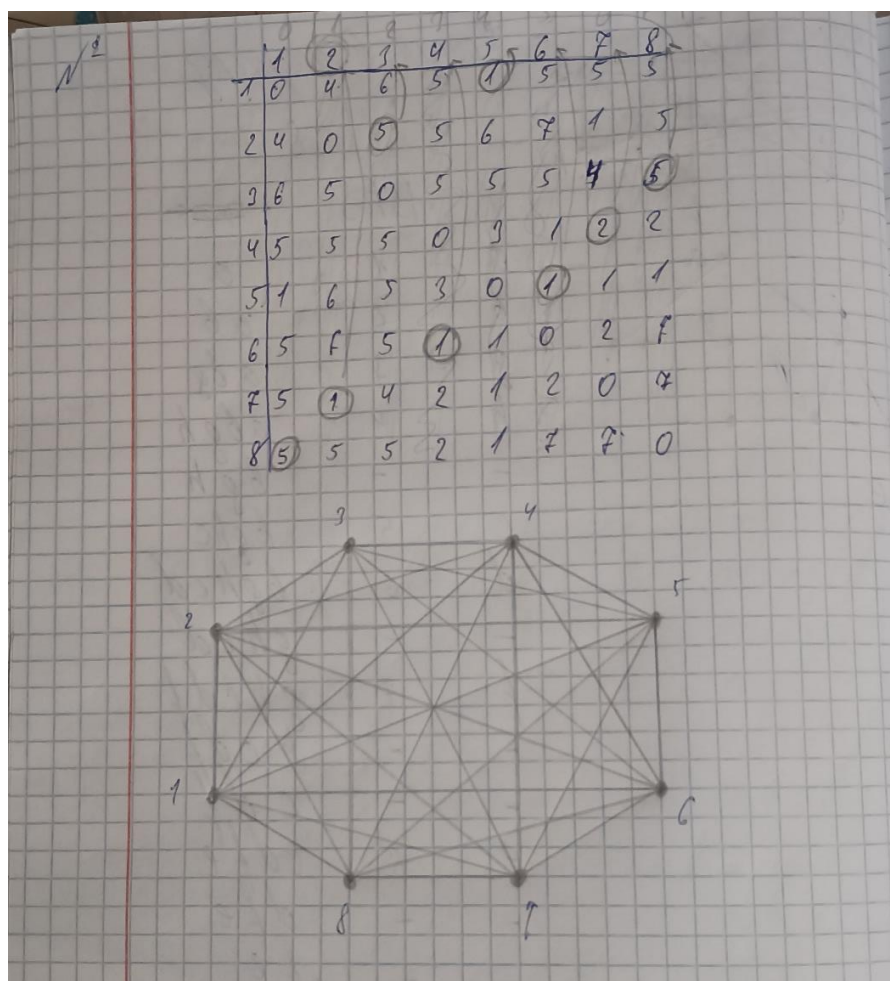
- введення вхідних даних вручну (вагова матриця);
- реалізацію алгоритму найближчого сусіда для пошуку ГЦ; □ виведення на екран вершин побудованого ГЦ та його довжини;
- перевірку на некоректне введення даних.

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

Таблиця 2.2

25		1	2	3	4	5	6	7	8
1	0	4	6	5	1	5	5	5	
2	4	0	5	5	6	7	1	5	
3	6	5	0	5	5	5	4	5	
4	5	5	5	0	3	1	2	2	
5	1	6	5	3	0	1	1	1	
6	5	7	5	1	1	0	2	7	
7	5	1	4	2	1	2	0	7	
8	5	5	5	2	1	7	7	0	

Зошит



V	W	Шлях
1	0	1
5	1	1-5
6	2	1-5-6
4	3	1-5-6-4
7	5	1-5-6-4-7
2	6	1-5-6-4-7-2
3	11	1-5-6-4-7-2-3
8	16	1-5-6-4-7-2-3-8
1	21	1-5-6-4-7-2-3-8-1

Реалізація у коді

```
namespace KDM_Lab02
{
    static class Task03
    {
        public static void ThirdTask()
        {
            int value;
            //int[,] myMatrix = GetMatrix();
            int[,] myMatrix =
            {
                {0, 4, 6, 5, 1, 5, 5, 5 },
                {4, 0, 5, 5, 6, 7, 1, 5 },
                {6, 5, 0, 5, 5, 5, 4, 5 },
                {5, 5, 5, 0, 3, 1, 2, 2 },
                {1, 6, 5, 3, 0, 1, 1, 1 },
                {5, 7, 5, 1, 1, 0, 2, 7 },
                {5, 1, 4, 2, 1, 2, 0, 7 },
                {5, 5, 5, 2, 1, 7, 7, 0 }
            };

            int[] walk = new int[myMatrix.GetLength(0) + 1];

            value = NearestNeighbour(myMatrix, walk);

            Show(walk, value);
        }
        static int NearestNeighbour(int[,] myMatrix, int[] walk)
        {
            int minIndex = 0, realMinIndex = 0, min = 100, counter = 0;
            int[] indexHolder = new int[myMatrix.GetLength(0)];
            walk[0] = 1;

            for (int i = 0; i < indexHolder.GetLength(0); i++)
                indexHolder[i] = i + 1;

            for (int f = 0; f < myMatrix.GetLength(0); f++)
            {
```



```

        if (f == myMatrix.GetLength(0) - 1)
        {
            counter += myMatrix[realMinIndex, 0];
            break;
        }

        for (int i = indexHolder[0]; i < indexHolder.GetLength(0); i++)
        {
            if (indexHolder[i] != 0 && myMatrix[realMinIndex, i] < min && i !=
realMinIndex)
            {
                min = myMatrix[realMinIndex, i];
                minIndex = i;
            }

            }
            indexHolder[minIndex] = 0;
            realMinIndex = minIndex;
            walk[f + 1] = realMinIndex + 1;
            counter += min;

            min = 100;
        }
        walk[myMatrix.GetLength(0)] = 1;

        return counter;
    }
    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];

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

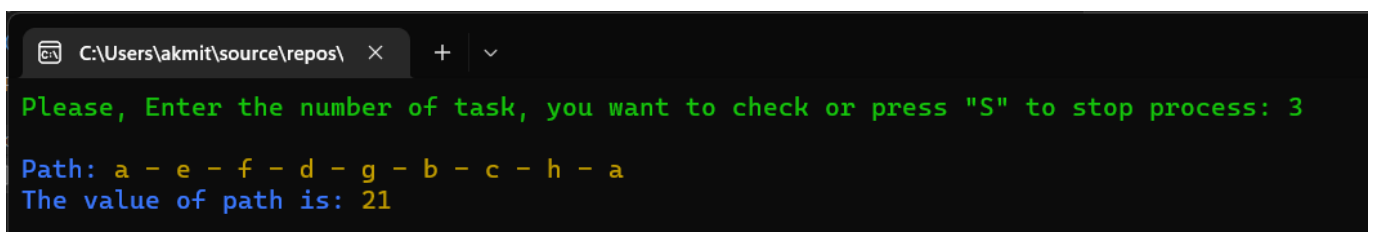
    return myMatrix;
}

static void Show(int[] walk, int value)
{
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Write("\nPath: ");
    Console.ForegroundColor = ConsoleColor.DarkYellow;
    for (int i = 0; i < walk.Length; i++)
    {
        Console.Write((char)(96 + walk[i]));
        if (i != walk.Length - 1)
            Console.Write(" - ");
    }
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Write("\nThe value of path is: ");
    Console.ForegroundColor = ConsoleColor.DarkYellow;
    Console.Write(value);
    Console.ForegroundColor = ConsoleColor.Gray;

    Console.WriteLine("\n");
}
}
}

```

Вивід у консоль



```

C:\Users\akmit\source\repos  X  +  v
Please, Enter the number of task, you want to check or press "S" to stop process: 3
Path: a - e - f - d - g - b - c - h - a
The value of path is: 21

```

Завдання 4

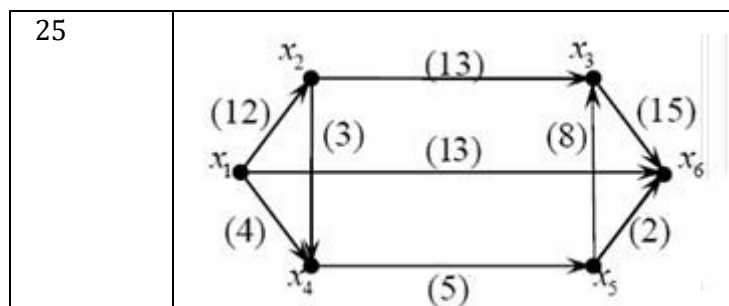
Орграф G задано геометрично (таблиця 2.3). Обчисліть матрицю досяжності орграфа G (таблиця 2.3, ваги дуг не брати до уваги) двома способами: за допомогою множення матриці суміжності та алгоритму Воршелла.

Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує обчислення матриці досяжності орграфа G (таблиця 2.3, ваги дуг не брати до уваги) двома способами: за допомогою піднесення до степеня матриці суміжності та алгоритму Воршелла згідно зі своїм варіантом.

Програма має передбачати такі можливості:

- реалізацію введення структури графа за допомогою побудови матриці суміжності;
- реалізацію обчислення та виведення різних степенів матриці суміжності графа та матриці досяжності;
- відповідно до свого варіанту реалізацію алгоритму Воршелла для обчислення та виведення матриці досяжності;
- перевірку на некоректне введення даних.

Лабораторна робота вважається зарахованою, якщо програма протестована в присутності викладача та отримано правильний результат.



Таблиця 2.3

Зошит

№4

G
 $|V|=6$

1) Найдите элементы матрицы.
Пусть M^* - матрица достижимости графа G , тогда:

$$M^* = M \vee M^2 \vee M^3 \vee M^4 \vee M^5 \vee M^6, \text{ где}$$

M - матрица смежности. Найдем:

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	0	1	0	0
x_2	0	0	1	0	0	0
x_3	0	0	0	0	0	1
x_4	0	0	0	0	1	0
x_5	0	0	1	0	0	1
x_6	0	0	0	0	0	0

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0	1	1	1	0
x_2	0	0	0	0	1	1
x_3	0	0	0	0	0	0
x_4	0	0	1	0	0	1
x_5	0	0	0	0	0	1
x_6	0	0	0	0	0	0

M^2

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0	1	0	0	2
x_2	0	0	0	0	0	1
x_3	0	0	0	0	0	0
x_4	0	0	0	0	0	0
x_5	0	0	0	0	0	0
x_6	0	0	0	0	0	0

M^3

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0	1	0	1	1
x_2	0	0	0	0	0	1
x_3	0	0	0	0	0	0
x_4	0	0	0	0	0	1
x_5	0	0	0	0	0	0
x_6	0	0	0	0	0	0

M^4

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0	1	0	0	2
x_2	0	0	0	0	0	1
x_3	0	0	0	0	0	0
x_4	0	0	0	0	0	0
x_5	0	0	0	0	0	0
x_6	0	0	0	0	0	0

M^5

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0	0	0	0	1
x_2	0	0	0	0	0	0
x_3	0	0	0	0	0	0
x_4	0	0	0	0	0	0
x_5	0	0	0	0	0	0
x_6	0	0	0	0	0	0

M^6

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0	0	0	0	0
x_2	0	0	0	0	0	0
x_3	0	0	0	0	0	0
x_4	0	0	0	0	0	0
x_5	0	0	0	0	0	0
x_6	0	0	0	0	0	0

M^*

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	1	1	1	1
x_2	0	0	1	1	1	1
x_3	0	0	0	0	0	1
x_4	0	0	1	0	1	1
x_5	0	0	1	0	0	1
x_6	0	0	0	0	0	0

2) Алгоритм Веруленса

$$W_0 = M_z$$

x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	0	1	0
x_2	0	0	1	0	0
x_3	0	0	0	0	1
x_4	0	0	0	1	0
x_5	0	1	0	0	1
x_6	0	0	0	0	0

де M — матриця суміжності.

Шляхи:

$$W_{1z}$$

x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	0	1	0
x_2	0	0	1	1	0
x_3	0	0	0	0	1
x_4	0	0	0	1	0
x_5	0	1	0	0	1
x_6	0	0	0	0	0

$$W_{2z}$$

x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	1	0	1
x_2	0	0	1	1	0
x_3	0	0	0	0	1
x_4	0	0	0	0	1
x_5	0	1	0	0	1
x_6	0	0	0	0	0

$$W_{3z}$$

x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	1	0	1
x_2	0	0	1	1	0
x_3	0	0	0	0	1
x_4	0	0	0	1	0
x_5	0	1	0	0	1
x_6	0	0	0	0	0

$$W_{4z}$$

x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	1	1	1
x_2	0	0	1	1	0
x_3	0	0	0	0	1
x_4	0	0	0	1	0
x_5	0	1	0	0	1
x_6	0	0	0	0	0

$$W_{5z}$$

x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	1	1	1
x_2	0	0	1	1	1
x_3	0	0	0	0	1
x_4	0	1	0	1	1
x_5	0	1	0	0	1
x_6	0	0	0	0	0

$$W_{6z}$$

x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	1	1	1	1
x_2	0	0	1	1	1
x_3	0	0	0	0	1
x_4	0	1	0	1	1
x_5	0	1	0	0	1
x_6	0	0	0	0	0

Реалізація у коді

```
namespace KDM_Lab02
{
    static class Task04
    {
        public static void FourthTask()
        {
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("\nEnter first matrix: \n");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] myMatrix = GetMatrix();
            int[,] myMatrix =
            {
                {0, 1, 0, 1, 0, 1},
                {0, 0, 1, 1, 0, 0},
                {0, 0, 0, 0, 0, 1},
                {0, 0, 0, 0, 1, 0},
                {0, 0, 1, 0, 0, 1},
                {0, 0, 0, 0, 0, 0}
            };

            Console.ForegroundColor = ConsoleColor.Blue;
            Console.Write("\nPlease, choose method (Multiplication or Warshall): \n");

            while (true)
            {
                Console.ForegroundColor = ConsoleColor.Cyan;
                string variant = Console.ReadLine();
                Console.ForegroundColor = ConsoleColor.Gray;

                if (variant.ToLower() == "m" || variant.ToLower() == "multiplication")
                {
                    ViaMultiplication(myMatrix);
                    break;
                }
                else if (variant.ToLower() == "w" || variant.ToLower() == "Warshall")
                {

```

```

        ViaWarshall(myMatrix);
        break;
    }
else
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.Write("\nPlease, enter proper value: ");
    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();
}

```



```

int[] l = new int[size];
for (int i = 0; i < l.Length; i++)
{
    l[i] = i + 1;
}

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 \"x{0}\" and \"x{1}\": ", l[i],
l[j]);

            try
            {
                Console.ForegroundColor = ConsoleColor.Cyan;
                myMatrix[i, j] = Convert.ToInt32(Console.ReadLine());
                Console.ForegroundColor = ConsoleColor.Gray;

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

                    Console.ForegroundColor = ConsoleColor.Gray;
                }
            }
            catch (FormatException)
            {
                Console.ForegroundColor = ConsoleColor.Red;

```

```

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

    }

    }

    }

    return myMatrix;
}

static void ViaMultiplication(int[,] myMatrix)
{
    int index;
    int[,] adjacencyMatrix;

    while (true)
    {
        try
        {
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.Write("\nPlease, enter power of matrix multiplication: ");
            Console.ForegroundColor = ConsoleColor.Cyan;
            index = Convert.ToInt32(Console.ReadLine());
            Console.ForegroundColor = ConsoleColor.Gray;

            if (index > 0 && index <= myMatrix.GetLength(0))
                break;
            else
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("\nPlease, enter proper value (From 1 to the
number of vertex)\n");
                Console.ForegroundColor = ConsoleColor.Gray;
            }
        }

        catch (FormatException)

```

```

        {

            Console.ForegroundColor = ConsoleColor.Red;

            Console.WriteLine("\nPlease, enter numbers only!\n");

            Console.ForegroundColor = ConsoleColor.Gray;

        }

    }

    Console.WriteLine();

    adjacencyMatrix = new int[myMatrix.GetLength(0), myMatrix.GetLength(0)];

    int[,] reachabilityMatrix = MatrixMultiplication(index, myMatrix,
adjacencyMatrix);


    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Write("\nGraph's reachability in ONLY {0} steps", index);
    Console.ForegroundColor = ConsoleColor.Gray;
    ShowMatrix(adjacencyMatrix);


    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Write("\nGraph's reachability in MAXIMUM {0} steps", index);
    Console.ForegroundColor = ConsoleColor.Gray;
    ShowMatrix(reachabilityMatrix);

}

static int[,] MatrixMultiplication(int index, int[,] matrix, int[,]
adjacencyMatrix)
{
    int powerNumber = 1;

    int[,] tempAdjacencyMatrix = new int[matrix.GetLength(0),
matrix.GetLength(0)];

    int[,] reachabilityMatrix = new int[matrix.GetLength(0),
matrix.GetLength(0)];

    int path = 0;

    for(int i = 0; i < matrix.GetLength(0); i++)
    {
        for(int j = 0; j < matrix.GetLength(0); j++)

```

```

        {
            adjacencyMatrix[i, j] = matrix[i, j];
            reachabilityMatrix[i, j] = matrix[i, j];
        }
    }

while(powerNumber < index)
{
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        for (int j = 0; j < matrix.GetLength(0); j++)
        {
            for (int k = 0; k < matrix.GetLength(0); k++)
                path += adjacencyMatrix[i,k] * matrix[k, j];

            tempAdjacencyMatrix[i, j] = path;
            path = 0;
        }
    }

    for (int i = 0; i < matrix.GetLength(0); i++)
        for (int j = 0; j < matrix.GetLength(0); j++)
        {
            if(adjacencyMatrix[i, j] != 0 || tempAdjacencyMatrix[i, j] != 0)
                reachabilityMatrix[i, j] = 1;
            adjacencyMatrix[i, j] = tempAdjacencyMatrix[i, j];
        }

    powerNumber++;
}

return reachabilityMatrix;
}

static void ViaWarshall(int[,] myMatrix)
{

```

```

        int[,] reachabilityMatrix = new int[myMatrix.GetLength(0),
myMatrix.GetLength(0)];

        int[,] tempReachabilityMatrix = new int[myMatrix.GetLength(0),
myMatrix.GetLength(0)];

        for (int i = 0; i < myMatrix.GetLength(0); i++)
        {
            for (int j = 0; j < myMatrix.GetLength(0); j++)
            {
                reachabilityMatrix[i, j] = myMatrix[i, j];
                tempReachabilityMatrix[i, j] = myMatrix[i, j];
            }
        }

        for (int i = 0; i < myMatrix.GetLength(0); i++)
        {
            for (int j = 0; j < myMatrix.GetLength(0); j++)
            {
                if (reachabilityMatrix[j, i] == 1)
                {
                    for (int k = 0; k < myMatrix.GetLength(0); k++)
                    {
                        if (reachabilityMatrix[j, k] == 0 &&
tempReachabilityMatrix[i, k] == 1)
                            reachabilityMatrix[j, k] = 1;
                    }
                }
            }
        }

        //Console.WriteLine("\nW{0}:\n\n", i+1);
        //ShowMatrix(reachabilityMatrix);
        //Console.WriteLine();

        for (int j = 0; j < myMatrix.GetLength(0); j++)
            for (int k = 0; k < myMatrix.GetLength(0); k++)
                tempReachabilityMatrix[j, k] = reachabilityMatrix[j, k];
    }

```

```

        Console.ForegroundColor = ConsoleColor.Blue;
        Console.WriteLine("\nGraph's reachability matrix: ");
        Console.ForegroundColor = ConsoleColor.Gray;
        ShowMatrix(reachabilityMatrix);
    }
    static void ShowMatrix(int[,] matrix)
    {
        int edges = 0, vertex = 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 vertex is: {1}\n",
edges, vertex);

        Console.ForegroundColor = ConsoleColor.Gray;
    }
}

```

Вивід у консоль

```
C:\Users\akmit\source\repos\ × + v

Please, Enter the number of task, you want to check or press "S" to stop process: 4
Please, choose method (Multiplication or Warshall):
m
Please, enter power of matrix multiplication: 3

Graph's reachability in ONLY 3 steps
0  0  1  0  1  2
0  0  1  0  0  1
0  0  0  0  0  0
0  0  0  0  0  1
0  0  0  0  0  0
0  0  0  0  0  0

Number of edges is: 6
Number of vertex is: 6

Graph's reachability in MAXIMUM 3 steps
0  1  1  1  1  1
0  0  1  1  1  1
0  0  0  0  0  1
0  0  1  0  1  1
0  0  1  0  0  1
0  0  0  0  0  0

Number of edges is: 15
Number of vertex is: 6
```



```

Please, Enter the number of task, you want to check or press "S" to stop process: 4
Please, choose method (Multiplication or Warshall):
w
Graph's reachability matrix:
0  1  1  1  1  1
0  0  1  1  1  1
0  0  0  0  0  1
0  0  1  0  1  1
0  0  1  0  0  1
0  0  0  0  0  0

Number of edges is: 15
Number of vertex is: 6

```

Завдання 5

Орграф G задано геометрично (таблиця 2.3). Вкажіть кількість вузлів і дуг: у орграфі-доповненні до орграфа G , у орграфі-перетині $G \cap H$ та у орграфі-об'єднанні $G \cup H$ (таблиця 2.3, ваги дуг не брати до уваги). Орграф H – варіант $k+1$.

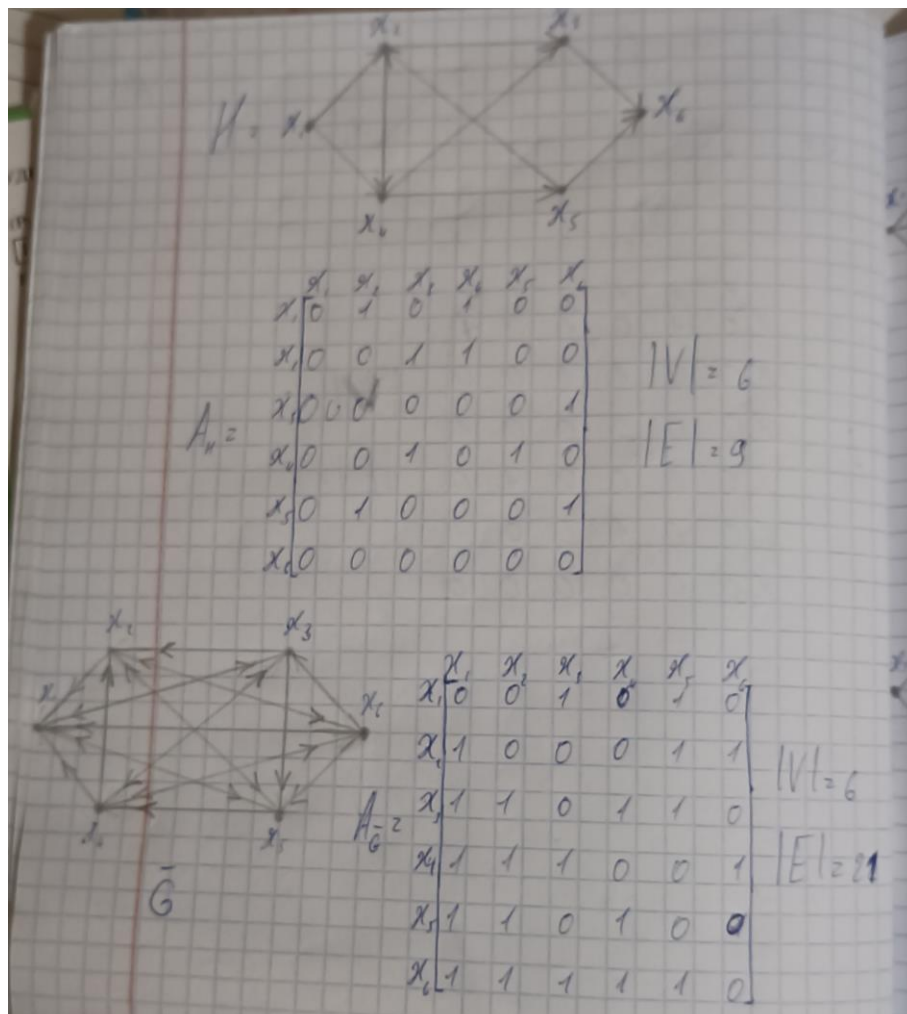
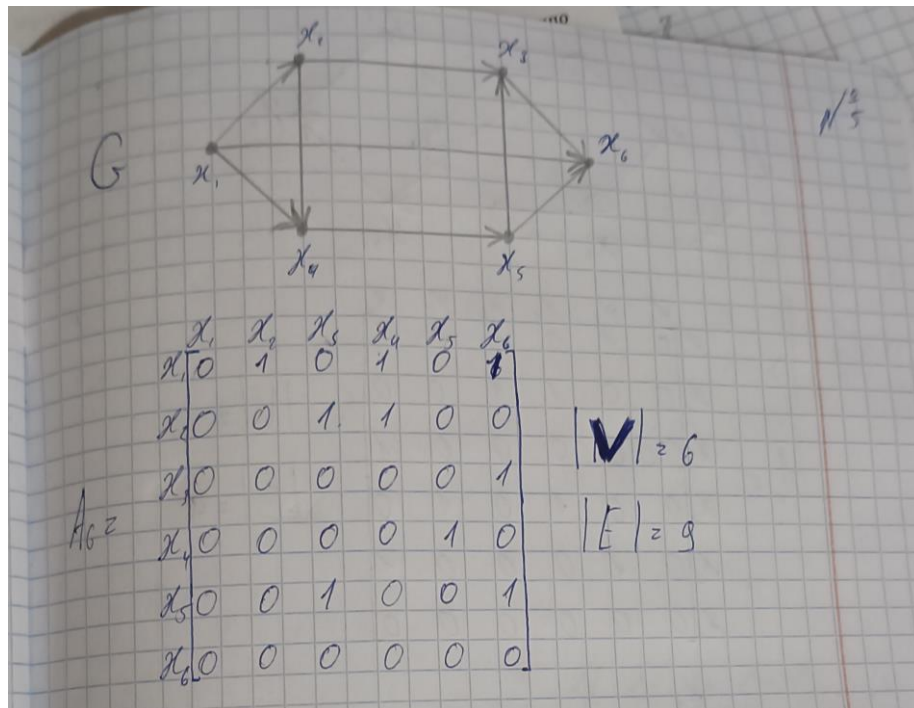
Написати програму (на будь-якій відомій студентові мові програмування), яка реалізує обчислення матриць суміжності орграфа-доповнення, орграфа-перетину $G \cap H$ та орграфа-об'єднання $G \cup H$ (таблиця 2.3, ваги дуг не брати до уваги, орграф H – варіант $k+1$).

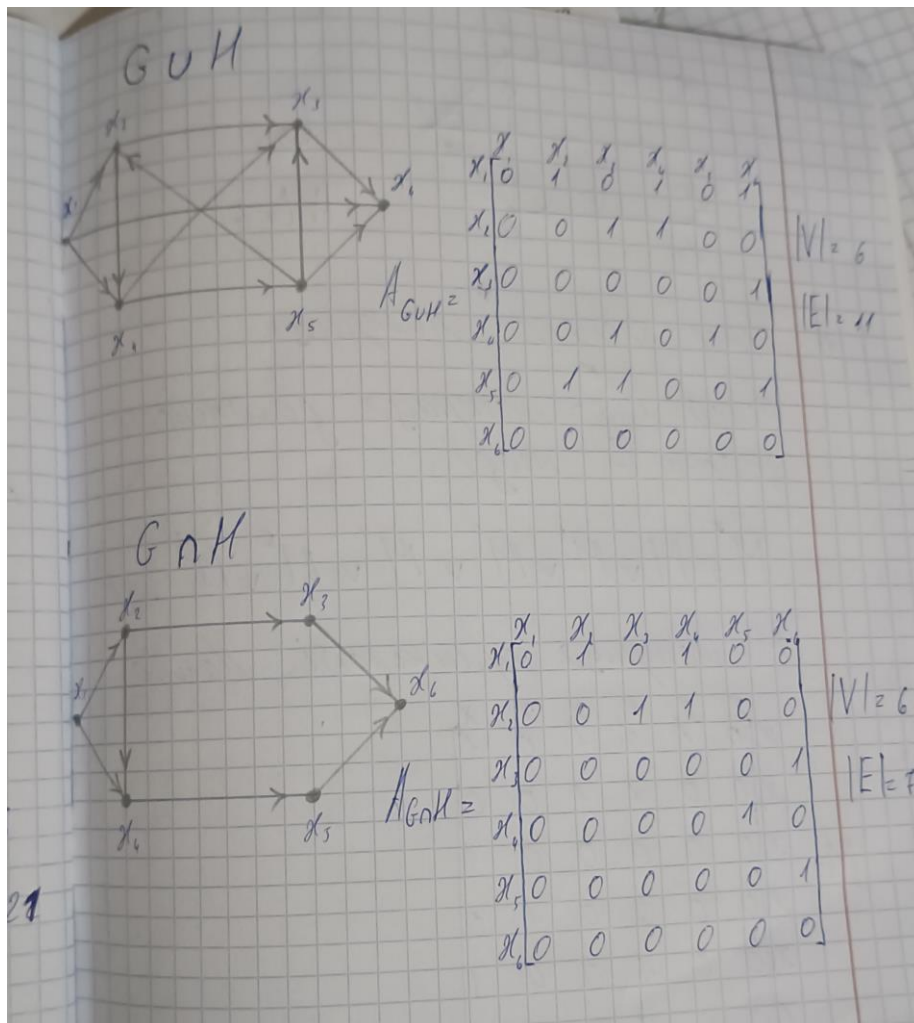
Програма має передбачати такі можливості:

- реалізацію введення структури орграфа за рахунок побудови матриці суміжності;
- реалізацію обчислення та виведення матриць суміжності орграфа-доповнення, орграфа-перетину та орграфа-об'єднання;
- виведення кількості вузлів та дуг вказаних графів;
- перевірку на некоректне введення даних.

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

Зошит





Реалізація у коді

```
namespace KDM_Lab02
{
    static class Task05
    {
        public static void FifthTask()
        {
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("\nEnter first matrix: \n");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] myMatrix = GetMatrix();
            int[,] myMatrix =
            {
                {0, 1, 0, 1, 0, 1},
                {0, 0, 1, 1, 0, 0},
                {0, 0, 0, 0, 0, 1},
                {0, 0, 0, 0, 1, 0},
                {0, 0, 1, 0, 0, 1},
                {0, 0, 0, 0, 0, 0}
            };
            //Console.ForegroundColor = ConsoleColor.Blue;
            //Console.WriteLine("Enter second matrix: \n");
            //Console.ForegroundColor = ConsoleColor.Gray;
            //int[,] secMatrix = GetMatrix();
        }
    }
}
```

```

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

int[,] myMatrixComplence = FindComplence(myMatrix);
int[,] matrixesAssociation = FindGraphsUnion(myMatrix, secMatrix);
int[,] matrixIntersection = FindGraphsIntersection(myMatrix, secMatrix);

Console.ForegroundColor = ConsoleColor.Blue;
Console.WriteLine("First matrix: ");
ShowMatrix(myMatrix);

Console.ForegroundColor = ConsoleColor.Blue;
Console.WriteLine("Second matrix: ");
ShowMatrix(secMatrix);

Console.ForegroundColor = ConsoleColor.Blue;
Console.WriteLine("First matrix Complence: ");
ShowMatrix(myMatrixComplence);

Console.ForegroundColor = ConsoleColor.Blue;
Console.WriteLine("Association of first and second matrixes: ");
ShowMatrix(matrixesAssociation);

Console.ForegroundColor = ConsoleColor.Blue;
Console.WriteLine("Intersection of first and second matrixes: ");
ShowMatrix(matrixIntersection);

Console.ForegroundColor = ConsoleColor.Gray;
}
static int[,] FindGraphsIntersection(int[,] myMatrix, int[,] secMatrix)
{
    int myMtrxLength = myMatrix.GetLength(0);
    int secMtrxLength = secMatrix.GetLength(0);
    int[,] result;

    if (myMtrxLength > secMtrxLength)
    {
        result = new int[secMtrxLength, secMtrxLength];
    }
    else
    {
        result = new int[myMtrxLength, myMtrxLength];
    }

    for (int i = 0; i < result.GetLength(0); i++)
    {
        for (int j = 0; j < result.GetLength(0); j++)
        {
            if (j == i) continue;

            if (myMatrix[i, j] == 1 && secMatrix[i, j] == 1)
            {
                result[i, j] = 1;
            }
        }
    }

    return result;
}

```

```

}
static int[,] FindGraphsUnion(int[,] myMatrix, int[,] secMatrix)
{
    int myMtrxLength = myMatrix.GetLength(0);
    int secMtrxLength = secMatrix.GetLength(0);
    int[,] result, smaller, bigger;

    if (myMtrxLength > secMtrxLength)
    {
        result = new int[myMtrxLength, myMtrxLength];
        smaller = secMatrix;
        bigger = myMatrix;
    }
    else
    {
        result = new int[secMtrxLength, secMtrxLength];
        smaller = myMatrix;
        bigger = secMatrix;
    }

    for (int i = 0; i < result.GetLength(0); i++)
    {
        for (int j = 0; j < result.GetLength(0); j++)
        {
            if (j == i) continue;

            if (j >= smaller.GetLength(0) || i >= smaller.GetLength(0))
            {
                result[i, j] = bigger[i, j];
                continue;
            }

            if (myMatrix[i, j] == 1 || secMatrix[i, j] == 1)
            {
                result[i, j] = 1;
            }
        }
    }

    return result;
}
static int[,] FindComplement(int[,] matrix)
{
    int len = matrix.GetLength(0);
    int[,] result = new int[len, len];

    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            if (j == i) continue;

            result[i, j] = matrix[i, j] == 0 ? 1 : 0;
        }
    }

    return result;
}
static int[,] GetMatrix()
{
    int size;
    while (true)
    {
        try
        {
            Console.ForegroundColor = ConsoleColor.Blue;

```

```

        Console.WriteLine("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();

int[] l = new int[size];
for (int i = 0; i < l.Length; i++)
{
    l[i] = i + 1;
}

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.WriteLine("Edge between \"x{0}\" and \"x{1}\": ", l[i],
l[j]);

            try
            {
                Console.ForegroundColor = ConsoleColor.Cyan;
                myMatrix[i, j] = Convert.ToInt32(Console.ReadLine());
                Console.ForegroundColor = ConsoleColor.Gray;

                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, vertix = 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 vertex is: {1}\n",
edges, vertex);
Console.ForegroundColor = ConsoleColor.Gray;
    }
}
}

```

Вивід у консоль

```

First matrix:

0    1    0    1    0    1
0    0    1    1    0    0
0    0    0    0    0    1
0    0    0    0    1    0
0    0    1    0    0    1
0    0    0    0    0    0

Number of edges is: 9
Number of vertex is: 6

Second matrix:

0    1    0    1    0    0
0    0    1    1    0    0
0    0    0    0    0    1
0    0    1    0    1    0
0    1    0    0    0    1
0    0    0    0    0    0

Number of edges is: 9
Number of vertex is: 6

```


First matrix Compliment:

0	0	1	0	1	0
1	0	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	0	1	0	0
1	1	1	1	1	0

Number of edges is: 21
Number of vertex is: 6

Association of first and second matrixes:

0	1	0	1	0	1
0	0	1	1	0	0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	1	0	0	1
0	0	0	0	0	0

Number of edges is: 11
Number of vertex is: 6

Intersection of first and second matrixes:

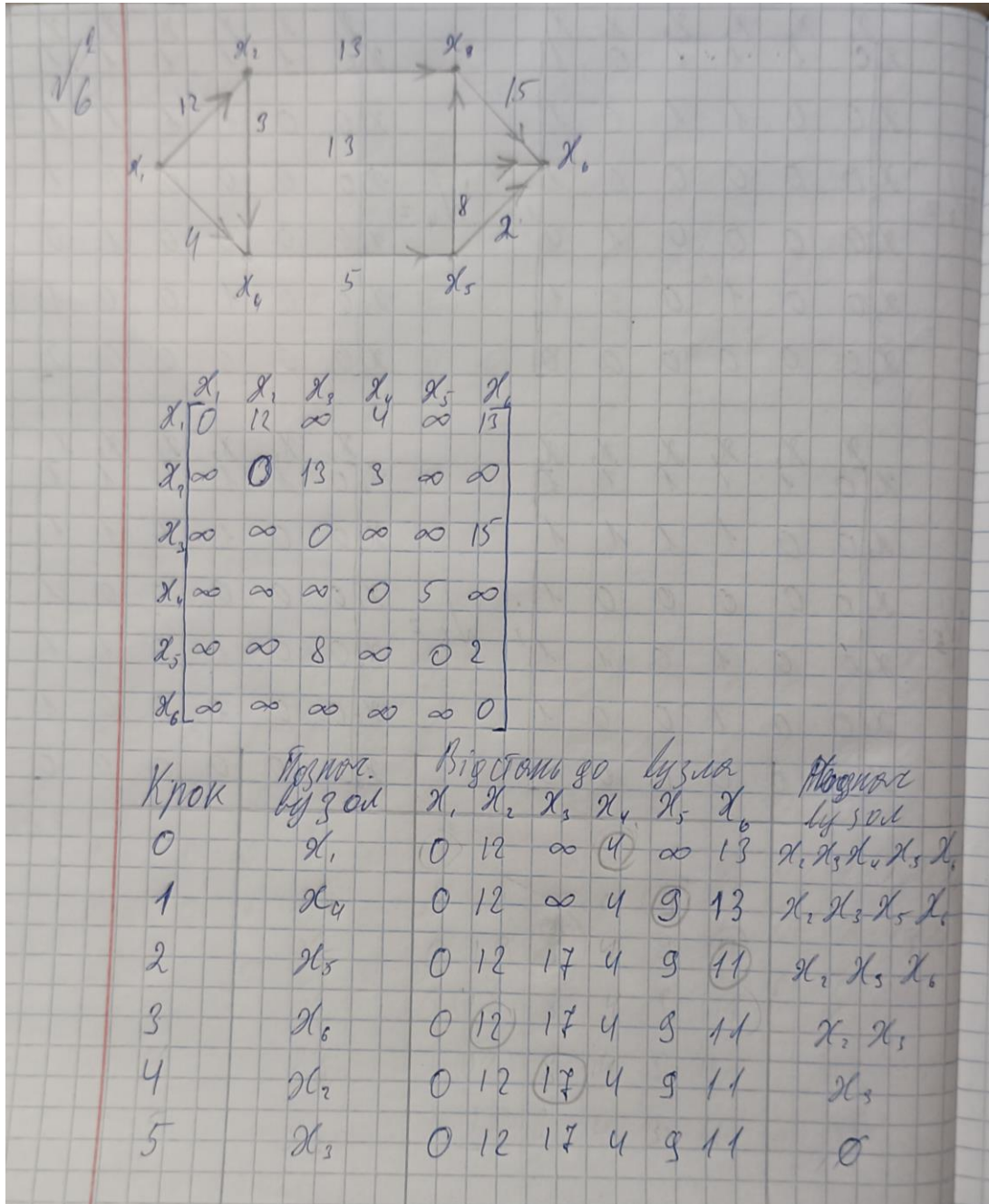
0	1	0	1	0	0
0	0	1	1	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	0	0	0

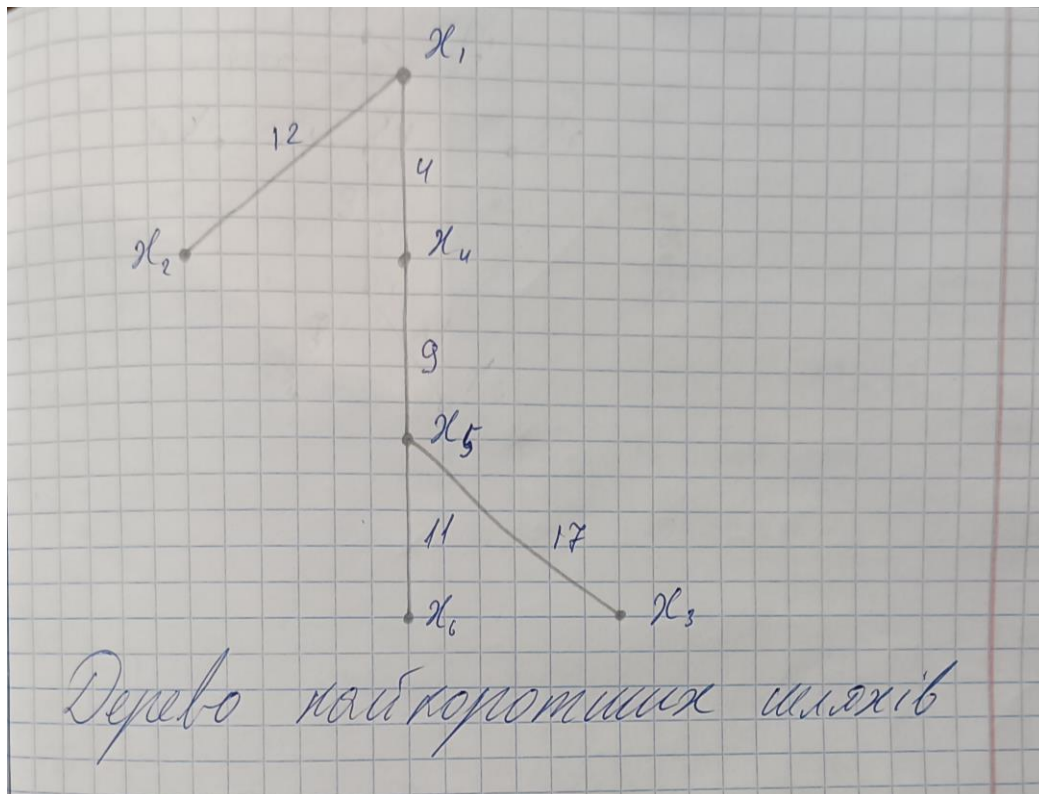
Number of edges is: 7
Number of vertex is: 6

Завдання 6

За допомогою алгоритму Дейкстри знайдіть найкоротші шляхи від вершини x_1 зваженого орграфу G до будь-якої іншої (таблиця 2.3).

Зошит





Висновок

Під час виконання лабораторної роботи, я на практиці ознайомився із основними поняттями теорії неорієнтованих та орієнтованих графів, навчився виконувати операції над графами, будувати матриці досяжності, знаходити у зваженому графі субоптимальний гамільтоновий цикл, засвоїв алгоритми пошуку вглиб і вшир та алгоритм Дейкстри.