**Міністерство освіти і науки України**
**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра обчислювальної техніки**

# Лабораторна робота № 2.4

з дисципліни
«Алгоритми і структури даних»

Виконав:

студент групи ІМ-22
Кушнір Микола Миколайович
номер у списку групи: 13

Перевірила:

Молчанова А. А.

Київ 2023

## *Постановка задачі*

***1.*** Представити напрямлений граф з заданими параметрами так само, як у лабораторній роботі №3. Відміна: матриця ***A*** напрямленого графа за варіантом формується за функціями:

```
srand(n1 n2 n3 n4);
T = randm(n, n);
A = mulmr(( 1.0 – n3 * 0.01 – n4 * 0.01 – 0.3) * T);
```

Перетворити граф у ненапрямлений.

***2.*** Визначити степені вершин напрямленого і ненапрямленого графів. Програма на екран виводить степені усіх вершин ненапрямленого графу і напівстепені виходу та заходу напрямленого графу. Визначити, чи граф є однорідним та якщо так, то вказати степінь однорідності графу.

***3.*** Визначити всі висячі та ізольовані вершини. Програма на екран виводить перелік усіх висячих та ізольованих вершин графу.

***4.*** Змінити матрицю графу за функцією

```
A = mulmr((1.0 – n3 * 0.005 – n4 * 0.005 – 0.27) * T);
```

Створити програму для обчислення наступних результатів:

*1)* матриця суміжності;

*2)* півстепені вузлів;

*3)* всі шляхи довжини 2 і 3;

*4)* матриця досяжності;

*5)* компоненти сильної зв’язності;

*6)* матриця зв’язності;

*7)* граф конденсації.

Шляхи довжиною **2** і **3** слід шукати за матрицями $A^2$ і $A^3$, відповідно. Матриця досяжності та компоненти сильної зв’язності слід шукати за допомогою операції транзитивного замикання.

## *Завдання для варіанту 13 (групи ІМ-22)*

- $n_1$ = **2**;
- $n_2$ = **2**;
- $n_3$ = **1**;
- $n_4$ = **3**;

Число вершин $n$: 10 + **1** = **11**.

Розміщення вершин: **прямокутником (квадратом)**.

Формування матриці ***A***:

```
srand(2 2 1 3);
T = randm(11, 11);
A = mulmr(( 1.0 - 1.0 * 0.01 - 3.0 * 0.01 - 0.3) * T);
```

Формування зміненої матриці:

```
A = mulmr((1.0 - 1.0 * 0.005 - 3.0 * 0.005 - 0.27) * T).
```

## ***Посилання на репозиторій з лабораторною роботою***

## *Текст програми*

### *Вміст файлу Configurations.h*

```
/*
 * Список скорочень, застосованих для найменування деяких ідентифікаторів
 * d або D => directed   (напрямлений)
 * u або U => undirected (ненапрямлений)
 * m або M => modified   (для позначення модифікованої матриці)
 * l або L => leaf       (для висячих вершин)
 * i або I => isolated   (для ізольованих вершин)
 */

/* N1N2 - номер групи, N3N4 - порядковий номер у списку групи */
#define N1  2
#define N2  2
#define N3  1
#define N4  3
/* Кількість рядків і стовпців матриць суміжності графів */
#define N (10 + N3)
/* Для позначення осей координат (позиції елементів записані у векторах) */
#define x   0
#define y   1
/* Значення, що використовуються в обчисленнях */
#define PI      3.1415926536
#define SQRT_2  1.4142135624

#define VERTEX_RADIUS           40
#define LOOP_RADIUS             (5 * VERTEX_RADIUS / 4)
```

```
#define ONE_STEP_LENGTH          (9 * VERTEX_RADIUS / 2) /* Найменша відстань між
вершинами графа */
#define MAX_ONE_STEP_LENGTH      (3 * ONE_STEP_LENGTH / 2)
#define WINDOW_BORDER_OFFSET     (2 * LOOP_RADIUS + 10)

const wchar_t *vertices_names[] =
{
    L"1", L"2", L"3",
    L"4", L"5", L"6",
    L"7", L"8", L"9",
    L"10", L"11",
};

#define GRAPH_WIDTH      ((int) ((double) (N - 5) * 0.5) * ONE_STEP_LENGTH)
#define GRAPH_HEIGHT     (3 * ONE_STEP_LENGTH)
/* Щоб граф коректно відобразився у вікні, його висота має бути сталою */

const int min_coords[] =
{
    WINDOW_BORDER_OFFSET,
    WINDOW_BORDER_OFFSET
};
const int max_coords[] =
{
    WINDOW_BORDER_OFFSET + GRAPH_WIDTH,
    WINDOW_BORDER_OFFSET + GRAPH_HEIGHT
};

const int vertex_print_offset[] = { 5, 8 };

/* Визначають зміщення центру ребер для огинання вершин або вже намальованих
ребер */
const double edge_center_offset_dividers[] = { 1, 3.6, 4.5, 6.5, 7 };
```

## *Вміст файлу DrawingDataSetter.h*

```
#include <math.h>
#include "Configurations.h"
#include "WorkWithMatrices.h"

typedef struct DrawingData
{
    int edge_type;
    int start[2];
    int center[2];
    int end[2];
    double angle;
    int arrow_end[2];
} draw_data;

/********************************************** ПЕРЕЛІК ФУНКЦІЙ
*****************************************************/
/****** Функції, що виконують основні обчислення
*************************************************************************/
int      **SetVerticesCoords    (int n);
draw_data SetEdgeDrawData       (int v1, int v2, int is_directed, int **coords,
int drawn_lines[N][N]);
/****** Допоміжні функції
*************************************************************************
*************/
int       CheckGraphType        (int n, int **graph_matrix);
double    Pow2                  (int value);
```

```
double    GetDistance          (const int *v1_pos, const int *v2_pos);
double    ConvertDegreeToRad   (double degree_value);
/*****************************************************************************
************************************/

int **SetVerticesCoords(int n)
{
    int **coords = Create2dIntArr(n, n);
    int current_pos[2] = { min_coords[x], min_coords[y] };
    for (int i = 0; i < n; i++)
    {
        coords[i][x] = current_pos[x];
        coords[i][y] = current_pos[y];
        if (current_pos[x] < max_coords[x] && current_pos[y] == min_coords[y])
            current_pos[x] += ONE_STEP_LENGTH;
        else if (current_pos[y] < max_coords[y] && current_pos[x] ==
max_coords[x])
            current_pos[y] += ONE_STEP_LENGTH;
        else if (current_pos[x] > min_coords[x] && current_pos[y] ==
max_coords[y])
            current_pos[x] -= ONE_STEP_LENGTH;
        else if (current_pos[y] > min_coords[y] && current_pos[x] ==
min_coords[x])
            current_pos[y] -= GRAPH_HEIGHT / (3 - n % 2);
        /*
         * Якщо к-сть вершин парна та більша ніж 10, то з
         * лівого боку буде розміщено дві вершини,
         * а якщо непарна - 1
         */
    }
    return coords;
}

draw_data SetEdgeDrawData(int v1, int v2, int is_directed, int **coords, int
drawn_lines[N][N])
{
    draw_data data;
    data.edge_type = 1;
    data.start[x] = coords[v1][x];
    data.start[y] = coords[v1][y];
    data.end[x] = coords[v2][x];
    data.end[y] = coords[v2][y];
    data.center[x] = (data.start[x] + data.end[x]) / 2;
    data.center[y] = (data.start[y] + data.end[y]) / 2;
    int dx = data.end[x] - data.start[x];
    int dy = data.end[y] - data.start[y];
    int index = (int) (GetDistance(data.start, data.end) / ONE_STEP_LENGTH);
    int center_offset[2] =
    {
        abs((int) (dy / edge_center_offset_dividers[index])),
        abs((int) (dx / edge_center_offset_dividers[index]))
    };
    int is_drawn = 0;
    if (drawn_lines[v1][v2] == 1 && drawn_lines[v2][v1] == 1)
    {
        is_drawn = 1;
        if (!is_directed)
        {
            data.edge_type = 0;
            return data;
        }
        else
        {
            data.center[x] += center_offset[x];
```

```
            data.center[y] += center_offset[y];
        }
    }
    else
        drawn_lines[v1][v2] = drawn_lines[v2][v1] = 1;
    int variable_delta, static_coord;
    if (v1 == v2)
    {
        data.edge_type = 2;
        int loop_offset_direction[2] = { 0 };
        int arrow_direction[2] = { 0 };
        if (data.center[x] > min_coords[x] && data.center[y] == min_coords[y])
        {
            --loop_offset_direction[x];
            --loop_offset_direction[y];
            --arrow_direction[y];
            data.angle = ConvertDegreeToRad(-87);
        }
        else if (data.center[x] == max_coords[x] && data.center[y] >
min_coords[y])
        {
            ++loop_offset_direction[x];
            --loop_offset_direction[y];
            ++arrow_direction[x];
            data.angle = ConvertDegreeToRad(183);
        }
        else if (data.center[x] < max_coords[x] && data.center[y] ==
max_coords[y])
        {
            ++loop_offset_direction[x];
            ++loop_offset_direction[y];
            ++arrow_direction[y];
            data.angle = ConvertDegreeToRad(93);
        }
        else if (data.center[x] == min_coords[x] && data.center[y] <
max_coords[y])
        {
            --loop_offset_direction[x];
            ++loop_offset_direction[y];
            --arrow_direction[x];
            data.angle = ConvertDegreeToRad(3);
        }
        int loop_center_offset = (int)round(
                (VERTEX_RADIUS * SQRT_2 / 2 +
                 sqrt(Pow2(LOOP_RADIUS) - Pow2(VERTEX_RADIUS) / 2))
                / SQRT_2);
        data.center[x] += loop_center_offset * loop_offset_direction[x];
        data.center[y] += loop_center_offset * loop_offset_direction[y];
        data.arrow_end[x] = data.end[x] + VERTEX_RADIUS * arrow_direction[x];
        data.arrow_end[y] = data.end[y] + VERTEX_RADIUS * arrow_direction[y];
        return data;
    }
    else if ((dx == 0 ? (variable_delta = dy, static_coord = x, 1) : 0) ||
             (dy == 0 ? (variable_delta = dx, static_coord = y, 1) : 0 ))
    {
        if (abs(variable_delta) > MAX_ONE_STEP_LENGTH)
        {
            if (data.start[static_coord] == min_coords[static_coord])
            {
                if (!is_drawn)
                    data.center[static_coord] -= center_offset[static_coord];
            }
            else if (data.start[static_coord] == max_coords[static_coord])
            {
```

```
                if (!is_drawn)
                    data.center[static_coord] += center_offset[static_coord];
                else
                    data.center[static_coord] -= 2 *
center_offset[static_coord];
            }
        }
    }
    else if (dx == dy && is_drawn)
        data.center[x] -= center_offset[x];
    else
    {
        if (is_drawn)
        {
            int graph_center[] =
            {
                ((min_coords[x] + max_coords[x]) / 2),
                ((min_coords[y] + max_coords[y]) / 2)
            };
            int alternative_center[] =
            {
                (data.center[x] - 2 * center_offset[x]),
                (data.center[y] - 2 * center_offset[y])
            };
            if (GetDistance(data.center, graph_center) >
                GetDistance(alternative_center, graph_center))
            {
                data.center[x] = alternative_center[x];
                data.center[y] = alternative_center[y];
            }
        }
    }
    if (is_directed)
    {
        int new_dx = data.end[x] - data.center[x];
        int new_dy = data.end[y] - data.center[y];
        double hypotenuse = GetDistance(data.center, data.end);
        if (new_dx >= 0 && new_dy >= 0)
            data.angle = acos(abs(new_dx) / hypotenuse) * -1;
        else if (new_dx >= 0 && new_dy < 0)
            data.angle = acos(abs(new_dx) / hypotenuse);
        else if (new_dx < 0 && new_dy >= 0)
            data.angle = (PI - acos(abs(new_dx) / hypotenuse)) * -1;
        else if (new_dx < 0 && new_dy < 0)
            data.angle = PI - acos(abs(new_dx) / hypotenuse);
        data.arrow_end[x] = data.end[x] - (int) round((double) VERTEX_RADIUS *
cos(data.angle));
        data.arrow_end[y] = data.end[y] + (int) round((double) VERTEX_RADIUS *
sin(data.angle));
    }
    return data;
}

int CheckGraphType(int n, int **graph_matrix)
{
    int is_directed = 0;
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (graph_matrix[i][j] != graph_matrix[j][i])
            {
                is_directed = 1;
```

```
                break;
            }
        }
    }
    return is_directed;
}

double Pow2(int value)
{
    return (double)(value * value);
}

double GetDistance(const int *v1_pos, const int *v2_pos)
{
    int a = v2_pos[x] - v1_pos[x];
    int b = v2_pos[y] - v1_pos[y];
    return sqrt(Pow2(a) + Pow2(b));
}

double ConvertDegreeToRad(double degree_value)
{
    return PI * degree_value / 180.0;
}
```

## *Вміст файлу GraphPainter.h*

```
#include "DrawingDataSetter.h"

void DrawGraph(int n,
               int **graph_matrix,
               int **coords,
               HPEN e_pen,
               HBRUSH v_brush,
               HPEN v_pen,
               HDC hdc)
{
    /* Зображаємо ребра */
    int i, j;
    int is_directed = CheckGraphType(n, graph_matrix);
    int drawn_lines[N][N] = { 0 };
    SelectObject(hdc, e_pen);
    SelectObject(hdc, GetStockObject(NULL_BRUSH));
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (graph_matrix[i][j] == 1)
            {
                draw_data data = SetEdgeDrawData(i, j, is_directed, coords,
drawn_lines);
                switch (data.edge_type)
                {
                case 0:
                    break;
                case 1:
                    MoveToEx(hdc, data.start[x], data.start[y], NULL);
                    LineTo(hdc, data.center[x], data.center[y]);
                    MoveToEx(hdc, data.center[x], data.center[y], NULL);
                    LineTo(hdc, data.end[x], data.end[y]);
                    break;
                case 2:
                    Ellipse(hdc,
```

```
                                (data.center[x] - LOOP_RADIUS),
                                (data.center[y] - LOOP_RADIUS),
                                (data.center[x] + LOOP_RADIUS),
                                (data.center[y] + LOOP_RADIUS));
                        break;
                    }
                    if (is_directed)
                    {
                        /* Якщо граф напрямлений, малюємо стрілку */
                        double fi = PI - data.angle;
                        int leftLineEnd[2], rightLineEnd[2];
                        rightLineEnd[x] = data.arrow_end[x] + (int) (30 * cos(fi +
0.3));
                        rightLineEnd[y] = data.arrow_end[y] + (int) (30 * sin(fi +
0.3));
                        leftLineEnd[x] = data.arrow_end[x] + (int) (30 * cos(fi -
0.3));
                        leftLineEnd[y] = data.arrow_end[y] + (int) (30 * sin(fi -
0.3));
                        MoveToEx(hdc, leftLineEnd[x], leftLineEnd[y], NULL);
                        LineTo(hdc, data.arrow_end[x], data.arrow_end[y]);
                        LineTo(hdc, rightLineEnd[x], rightLineEnd[y]);
                    }
                }
            }
        }
        /* Зображаємо вершини */
        int left, top, right, bottom;
        int print_pos[2];
        for (i = 0; i < n; i++)
        {
            left =      (coords[i][x] - VERTEX_RADIUS);
            top =       (coords[i][y] - VERTEX_RADIUS);
            right =     (coords[i][x] + VERTEX_RADIUS);
            bottom =    (coords[i][y] + VERTEX_RADIUS);
            if (i > 8)
                print_pos[x] = coords[i][x] - (int)(1.5 * vertex_print_offset[x]);
                /* Ці елементи складаються з двох цифр, тому зміщення має бути
більшим */
            else
                print_pos[x] = coords[i][x] - vertex_print_offset[x];
            print_pos[y] = coords[i][y] - vertex_print_offset[y];
            SelectObject(hdc, v_brush);
            Ellipse(hdc, left, top, right, bottom);
            SelectObject(hdc, v_pen);
            Ellipse(hdc, left, top, right, bottom);
            TextOut(hdc, print_pos[x], print_pos[y], vertices_names[i], 2);
        }
    }
```

## *Вміст файлу PrimitiveTableOutput.h*

```
/* Типи лінії таблиці */
#define FIRST_LINE      1
#define MIDDLE_LINE     2
#define LAST_LINE       3
/* ASCII символи для зображення кожного типу лінії таблиці */
const int first_line_chars[]    = {218, 196, 194, 191};
const int middle_line_chars[]   = {195, 196, 197, 180};
const int last_line_chars[]     = {192, 196, 193, 217};

void PrintTableLine(int cols_quantity, const int cols_lengths[cols_quantity],
```

```
int line_type)
{
    const int *pointer;
    switch (line_type)
    {
    case FIRST_LINE :
        pointer = first_line_chars;
        break;
    case MIDDLE_LINE :
        pointer = middle_line_chars;
        break;
    case LAST_LINE :
        pointer = last_line_chars;
    }
    int symbols[4];
    int i, j;
    for (i = 0; i < 4; i++)
        symbols[i] = pointer[i];
    printf("%c", symbols[0]);
    for (i = 0; i < cols_quantity; i++)
    {
        for (j = 0; j < cols_lengths[i]; j++)
            printf("%c", symbols[1]);
        if ((i + 1) != cols_quantity)
            printf("%c", symbols[2]);
    }
    printf("%c\n", symbols[3]);
}
```

## *Вміст файлу WorkWithMatrices.h*

```
#include <stdio.h>
#include <stdlib.h>
#include "PrimitiveTableOutput.h"

/************************************************ ПЕРЕЛІК ФУНКЦІЙ
*****************************************************/
/***** Функції, що задані в умові лабораторної роботи
*****************************************************************/
double **randm                              (int n1, int n2);
int    **mulmr                              (int n1, int n2, double **matrix_T,
double coefficient);
/***** Функції, що виконують основні обчислення
*************************************************************************/
int    **SymmetrizeMatrix                   (int n, int **matrix_A);
int    **GetVerticesDegsOfDGraph            (int n, int **matrix_A);
void     PrintDGraphHomogeneityDeg          (int n, int **vertices_degs);
void     PrintLIVerticesOfDGraph            (int n, int **vertices_degs);
int     *GetVerticesDegsOfUGraph            (int n, int **graph_matrix);
void     PrintUGraphHomogeneityDeg          (int n, int *vertices_degs);
void     PrintLIVerticesOfUGraph            (int n, int *vertices_degs);
int    **MultSquareMatrices                 (int n, int **matrix_1, int
**matrix_2);
void     PrintPathsWithLen2                 (int n, int **matrix_pow1, int
**matrix_pow2);
void     PrintPathsWithLen3                 (int n, int **matrix_pow1, int
**matrix_pow3);
int    **MultSquareMatricesElemByElem       (int n, int **matrix_1, int
**matrix_2);
int    **TransposeSquareMatrix              (int n, int **matrix);
int    **SumSquareMatrices                  (int n, int **matrix_1, int
**matrix_2);
```

```
int    **GetConnectComponents              (int n, int **matrix_S);
int    **GetReachabilityMatrix             (int n, int **matrix_A);
int    **GetCondensationGraphMatrix        (int n, int **components, int
**matrix_mA);
/***** Допоміжні функції
*****************************************************************************
**************/
double   RandInRange                       (double min, double max);
void     PrintDouble2dArr                  (int rows, int cols, double **arr);
int    **Create2dIntArr                    (int rows, int cols);
void     PrintInt2dArr                     (int rows, int cols, int *lengths,
char **headings, int **arr);
void     PrintIntArr                       (int arr_length, int *lengths, char
*heading, int *arr);
void     PrintBooleanMatrix                (int n, int **matrix);
void     PrintConnectComponents            (int n, int **components);
void     FreeInt2dArr                      (int rows, int **arr);
void     FreeDouble2dArr                   (int rows, double **arr);
/*****************************************************************************
************************************/

double **randm(int n1, int n2)
{
    double **matrix_T = (double **) malloc(sizeof(double *) * n1);
    int i, j;
    for (i = 0; i < n1; i++)
    {
        matrix_T[i] = (double *) malloc(sizeof(double) * n2);
        for (j = 0; j < n2; j++)
            matrix_T[i][j] = RandInRange(0.0, 2.0);
    }
    return matrix_T;
}

int **mulmr(int n1, int n2, double **matrix_T, double coefficient)
{
    int **matrix_A = Create2dIntArr(n1, n2);
    int i, j;
    for (i = 0; i < n1; i++)
    {
        for (j = 0; j < n2; j++)
            matrix_A[i][j] = (int) (matrix_T[i][j] * coefficient);
    }
    return matrix_A;
}

int **SymmetrizeMatrix(int n, int **matrix_A)
{
    int **matrix = Create2dIntArr(n, n);
    int i, j;
    for (i = 0; i < n; i++)
        for (j = i; j < n; j++)
            if (matrix_A[i][j] == 1 || matrix_A[j][i] == 1)
                matrix[i][j] = matrix[j][i] = 1;
    return matrix;
}

int **GetVerticesDegsOfDGraph(int n, int **matrix_A)
{
    int **degs = Create2dIntArr(n, 2);
    /*
     * degs[0-10][1] - напівстепінь виходу вершини;
     * degs[0-10][0] - напівстепінь заходу вершини;
     */
```

```
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
        {
            if (matrix_A[i][j])
                degs[i][0]++;
            if (matrix_A[j][i])
                degs[i][1]++;
        }
    return degs;
}

void PrintDGraphHomogeneityDeg(int n, int **vertices_degs)
{
    int r = 0, i;
    for (i = 1; i < n; i++)
        if (vertices_degs[0][0] != vertices_degs[i][0])
        {
            r = -1;
            break;
        }
    if (r == -1)
        printf("Graph isn't homogeneous\n");
    else
    {
        r = vertices_degs[0][0];
        printf("Graph homogeneity degree: %d\n", r);
    }
}

void PrintLIVerticesOfDGraph(int n, int **vertices_degs)
{
    int i;
    int leaf_ct = 0, isolated_ct = 0;
    printf("Leaf vertices    : ");
    for (i = 0; i < n; i++)
        if ((vertices_degs[i][0] == 1 && vertices_degs[i][1] == 0) ||
            (vertices_degs[i][0] == 0 && vertices_degs[i][1] == 1))
        {
            printf("%d ", (i + 1));
            leaf_ct++;
        }
    if (!leaf_ct)
        printf("graph hasn't leaf vertices");
    printf("\nIsolated vertices: ");
    for (i = 0; i < n; i++)
        if (vertices_degs[i][0] == 0 &&
            vertices_degs[i][1] == 0)
        {
            printf("%d ", (i + 1));
            isolated_ct++;
        }
    if (!isolated_ct)
        printf("graph hasn't isolated vertices");
    printf("\n");
}

int *GetVerticesDegsOfUGraph(int n, int **graph_matrix)
{
    int *degs = (int *) calloc(n, sizeof(int));
    /* degs[0-10] - степінь вершини */
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
```

```
            if (graph_matrix[i][j])
                (i == j) ? (degs[i] += 2) : ++degs[i];
    return degs;
}

void PrintUGraphHomogeneityDeg(int n, int *vertices_degs)
{
    int r = 0, i;
    for (i = 1; i < n; i++)
        if (vertices_degs[0] != vertices_degs[i])
        {
            r = -1;
            break;
        }
    if (r == -1)
        printf("Graph isn't homogeneous\n");
    else
    {
        r = vertices_degs[0];
        printf("Graph homogeneity degree: %d\n", r);
    }
}

void PrintLIVerticesOfUGraph(int n, int *vertices_degs)
{
    int i;
    int leaf_ct = 0, isolated_ct = 0;
    printf("Leaf vertices    : ");
    for (i = 0; i < n; i++)
        if (vertices_degs[i] == 1)
        {
            printf("%d ", (i + 1));
            leaf_ct++;
        }
    if (!leaf_ct)
        printf("graph hasn't leaf vertices");
    printf("\nIsolated vertices: ");
    for (i = 0; i < n; i++)
        if (vertices_degs[i] == 0)
        {
            printf("%d ", (i + 1));
            isolated_ct++;
        }
    if (!isolated_ct)
        printf("graph hasn't isolated vertices");
    printf("\n");
}

int **MultSquareMatrices(int n, int **matrix_1, int **matrix_2)
{
    int **result = Create2dIntArr(n, n);
    int i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            for (k = 0; k < n; k++)
                result[i][j] += matrix_1[i][k] * matrix_2[k][j];
    return result;
}

int **MultSquareMatricesElemByElem(int n, int **matrix_1, int **matrix_2)
{
    int **result = Create2dIntArr(n, n);
    int i, j;
    for (i = 0; i < n; i++)
```

```
        for (j = 0; j < n; j++)
            result[i][j] = matrix_1[i][j] * matrix_2[i][j];
    return result;
}

int **TransposeSquareMatrix(int n, int **matrix)
{
    int **result = Create2dIntArr(n, n);
    int i, j;
    for (i = 0; i < n; i++)
        for (j = i; j < n; j++)
        {
            result[i][j] = matrix[j][i];
            result[j][i] = matrix[i][j];
        }
    return result;
}

int **SumSquareMatrices(int n, int** matrix_1, int **matrix_2)
{
    int **result = Create2dIntArr(n, n);
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            result[i][j] = matrix_1[i][j] + matrix_2[i][j];
    return result;
}

void PrintPathsWithLen2(int n, int **matrix_pow1, int **matrix_pow2)
{
    int cols_quantity = 4;
    int cols_lengths[] = { 11, 7, 7, 7 };
    PrintTableLine(cols_quantity, cols_lengths, 1);
    printf("%c    No.    %c  v-1  %c  v-2  %c  v-3  %c\n",
           179, 179, 179, 179, 179);
    int path_counter = 1;
    int path[3];
    int i, j, k;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (matrix_pow2[i][j])
            {
                path[0] = i;
                path[2] = j;
                for (k = 0; k < n; k++)
                {
                    if (matrix_pow1[i][k] && matrix_pow1[k][j])
                    {
                        path[1] = k;
                        PrintTableLine(cols_quantity, cols_lengths, 2);
                        printf("%c   %5d   %c  %2d   %c  %2d   %c  %2d   %c\n",
                               179, path_counter,
                               179, (path[0] + 1),
                               179, (path[1] + 1),
                               179, (path[2] + 1), 179);
                        path_counter++;
                    }
                }
            }
        }
    }
    PrintTableLine(cols_quantity, cols_lengths, 3);
```

```
}

void PrintPathsWithLen3(int n, int **matrix_pow1, int **matrix_pow3)
{
    int cols_quantity = 5;
    int cols_lengths[] = { 11, 7, 7, 7, 7 };
    PrintTableLine(cols_quantity, cols_lengths, 1);
    printf("%c    No.    %c  v-1  %c  v-2  %c  v-3  %c  v-4  %c\n",
           179, 179, 179, 179, 179, 179);
    int path_counter = 1;
    int path[4];
    int i, j, k, l;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (matrix_pow3[i][j])
            {
                path[0] = i;
                path[3] = j;
                for (k = 0; k < n; k++)
                {
                    if (matrix_pow1[i][k])
                    {
                        for (l = 0; l < n; l++)
                        {
                            if (matrix_pow1[k][l] && matrix_pow1[l][j])
                            {
                                path[1] = k;
                                path[2] = l;
                                PrintTableLine(cols_quantity, cols_lengths, 2);
                                printf("%c   %5d   %c  %2d   %c  %2d   %c  %2d
%c  %2d   %c\n",
                                       179, path_counter,
                                       179, (path[0] + 1),
                                       179, (path[1] + 1),
                                       179, (path[2] + 1),
                                       179, (path[3] + 1), 179);
                                path_counter++;
                            }
                        }
                    }
                }
            }
        }
    }
    PrintTableLine(cols_quantity, cols_lengths, 3);
}

int **GetConnectComponents(int n, int **matrix_S)
{
    int **components = Create2dIntArr(n, (n + 1));
    /*
     * Максимальне число компонент зв'язності = к-сті вершин = n;
     * максимальне число вершин у компоненті зв'язності = к-сті вершин = n;
     * останній елемент у рядку вказуватиме на першу вільну позицію у
components[0-10][]
     */
    int counter = 0; /* лічильник вказуватиме на першу вільну позицію у
components[] */
    int *included_vertices = (int *) calloc(n, sizeof(int));
    int i, j;
    for (i = 0; i < n; i++)
    {
```

```
        if (!included_vertices[i])
        {
            for (j = 0; j < n; j++)
            {
                if (matrix_S[i][j] != 0)
                {
                    included_vertices[j]++;
                    components[counter][components[counter][n]] = j + 1;
                    components[counter][n]++;
                }
            }
            counter++;
        }
    }
    free(included_vertices);
    return components;
}

int **GetReachabilityMatrix(int n, int **matrix_A)
{
    int **matrix = Create2dIntArr(n, n);
    int **t_closing = Create2dIntArr(n, n);
    int i, j;
    for (i = 0; i < n; i++)
    {
        matrix[i][i] = 1; /* додавання одиничної матриці */
        for (j = 0; j < n; j++)
            t_closing[i][j] = matrix_A[i][j];
    }
    for (i = 0; i < n; i++)
    {
        matrix = SumSquareMatrices(n, matrix, t_closing);
        t_closing = MultSquareMatrices(n, t_closing, matrix_A);
    }
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            if (matrix[i][j] != 0)
                matrix[i][j] = 1; /* застосування булевого відображення */
    }
    FreeInt2dArr(n, t_closing);
    return matrix;
}

int **GetCondensationGraphMatrix(int n, int **components, int **matrix_mA)
{
    int index = 0;
    while (index < n && components[index][n] != 0)
        index++;
    int **graph_matrix = Create2dIntArr((index + 1), (index + 1));
    int i, j, k, l;
    for (i = 0; i < index; i++)
    {
        for (j = 0; j < index; j++)
        {
            if (i != j)
            {
                for (k = 0; k < components[i][n]; k++)
                {
                    for (l = 0; l < components[j][n]; l++)
                    {
                        if (matrix_mA[components[i][k] - 1][components[j][l] -
1])
                        {
```

```
                                /* Пошук ребра, що з'єднує компоненти */
                                graph_matrix[i][j] = 1;
                                break;
                            }
                        }
                        if (graph_matrix[i][j] == 1)
                            break;
                    }
                }
            }
        }
        return graph_matrix;
}

double RandInRange(double min, double max)
{
    double random = (double) rand() / RAND_MAX;
    double range = max - min;
    return min + range * random;
}

void PrintDouble2dArr(int rows, int cols, double **arr)
{
    int i, j;
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
            printf("%lf ", arr[i][j]);
        printf("\n");
    }
}

int **Create2dIntArr(int rows, int cols)
{
    int **arr = (int **) malloc(sizeof(int *) * rows);
    for (int i = 0; i < rows; i++)
        arr[i] = (int *) calloc(cols, sizeof(int));
    return arr;
}

void PrintInt2dArr(int rows, int cols,
                   int *lengths, char **headings,
                   int **arr)
{
    PrintTableLine((cols + 1), lengths, 1);
    int i, j, k;
    printf("%c No. %c", 179, 179);
    for (i = 0; i < cols; i++)
        printf(" %s %c", headings[i], 179);
    printf("\n");
    int space_before, space_after;
    for (i = 0; i < rows; i++)
    {
        PrintTableLine((cols + 1), lengths, 2);
        printf("%c %2d  %c", 179, (i + 1), 179);
        for (j = 0; j < cols; j++)
        {
            space_before = (int) (lengths[j + 1] / 2) - 1;
            space_after = space_before + (lengths[j + 1] % 2);
            for (k = 0; k < space_before; k++)
                printf(" ");
            printf("%2d", arr[i][j]);
            for (k = 0; k < space_after; k++)
                printf(" ");
```

```
            printf("%c", 179);
        }
        printf("\n");
    }
    PrintTableLine((cols + 1), lengths, 3);
}

void PrintIntArr(int arr_length,
                 int *lengths, char *heading,
                 int *arr)
{
    PrintTableLine(2, lengths, 1);
    int i, j;
    printf("%c No. %c %s %c\n", 179, 179, heading, 179);
    int space_before = (int) (lengths[1] / 2) - 1;
    int space_after = space_before + (lengths[1] % 2);
    for (i = 0; i < arr_length; i++)
    {
        PrintTableLine(2, lengths, 2);
        printf("%c %2d  %c", 179, (i + 1), 179);
        for (j = 0; j < space_before; j++)
            printf(" ");
        printf("%2d", arr[i]);
        for (j = 0; j < space_after; j++)
            printf(" ");
        printf("%c\n", 179);
    }
    PrintTableLine(2, lengths, 3);
}

void PrintBooleanMatrix(int n, int **matrix)
{
    int i, j;
    int cols_quantity = n;
    int *cols_lengths = (int *) calloc(cols_quantity, sizeof(int));
    printf("      %c", 179);
    for (i = 0; i < n; i++)
    {
        cols_lengths[i] = 3;
        printf("%2d %c", (i + 1), 179);
    }
    printf("\n");
    for (i = 0; i < n; i++)
    {
        printf("   %c%c%c", 196, 196, 196);
        PrintTableLine(cols_quantity, cols_lengths, 2);
        printf("   %2d %c", (i + 1), 179);
        for (j = 0; j < n; j++)
            printf("%2d %c", matrix[i][j], 179);
        printf("\n");
    }
    printf("   %c%c%c", 196, 196, 196);
    PrintTableLine(cols_quantity, cols_lengths, 3);
    free(cols_lengths);
}
void PrintConnectComponents(int n, int **components)
{
    int cols_lengths[] = { 5, 44 };
    PrintTableLine(2, cols_lengths, 1);
    printf("%c No. %c           Vertices in component            %c\n",
           179, 179, 179);
    int components_ct = 0;
    int i;
    while (components_ct < n && components[components_ct][n] != 0)
```

```
    {
        PrintTableLine(2, cols_lengths, 2);
        printf("%c %2d  %c", 179, (components_ct + 1), 179);
        for (i = 0; i < n; i++)
        {
            if (components[components_ct][i] != 0)
                printf(" %2d ", components[components_ct][i]);
            else
                printf("    ");
        }
        printf("%c\n", 179);
        components_ct++;
    }
    PrintTableLine(2, cols_lengths, 3);
}

void FreeInt2dArr(int rows, int **arr)
{
    for (int i = 0; i < rows; i++)
        free(arr[i]);
    free(arr);
}

void FreeDouble2dArr(int rows, double **arr)
{
    for (int i = 0; i < rows; i++)
        free(arr[i]);
    free(arr);
}
```

## *Вміст файлу main.c*

```
#ifndef UNICODE
#define UNICODE
#endif

#include <windows.h>
#include "GraphPainter.h"

#define TASK_1_DIRECTED_GRAPH       1
#define TASK_1_UNDIRECTED_GRAPH     2
#define TASK_2_DIRECTED_GRAPH       3
#define TASK_2_UNDIRECTED_GRAPH     4
#define TASK_3_DIRECTED_GRAPH       5
#define TASK_3_UNDIRECTED_GRAPH     6
#define TASK_4_1                    7
#define TASK_4_2                    8
#define TASK_4_3_1                  9
#define TASK_4_3_2                  10
#define TASK_4_4                    11
#define TASK_4_5                    12
#define TASK_4_6                    13
#define TASK_4_7                    14

int current_task = 0;
int **A;
int **uA;
int **mA;
int **current_matrix;
int **vertices_coords;
int **components_coords;
int **current_coords;
```

```
int current_n;

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

void AddMenu(HWND hwnd);

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
    WNDCLASS wndClass;
    wndClass.lpszClassName = L"Лабораторна робота 2.4";
    wndClass.hInstance = hInstance;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.hIcon = 0;
    wndClass.lpszMenuName = 0;
    wndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.cbClsExtra = 0;
    wndClass.cbWndExtra = 0;
    if (!RegisterClass(&wndClass)) return 0;
    HWND hWnd;
    MSG lpMsg;
    hWnd = CreateWindow(L"Лабораторна робота 2.4",
                        L"Лабораторна робота 2.4, виконав М.М.Кушнір",
                        WS_OVERLAPPEDWINDOW,
                        0, 0,
                        (WINDOW_BORDER_OFFSET * 2 + GRAPH_WIDTH + 40),
                        (WINDOW_BORDER_OFFSET * 2 + GRAPH_HEIGHT + 60),
                        (HWND)NULL,
                        (HMENU)NULL,
                        (HINSTANCE)hInstance,
                        (HINSTANCE)NULL);
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    int GetMessage_res;
    while ((GetMessage_res = GetMessage(&lpMsg, hWnd, 0, 0)) != 0)
    {
        if (GetMessage_res == -1)
            return lpMsg.wParam;
        else
        {
            TranslateMessage(&lpMsg);
            DispatchMessage(&lpMsg);
        }
    }
}

LRESULT CALLBACK WndProc(HWND hWnd,
                         UINT message,
                         WPARAM wParam,
                         LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    switch (message)
    {
    case WM_COMMAND:
    {
        system("cls");
        RedrawWindow(hWnd, NULL, NULL,RDW_ERASE | RDW_INVALIDATE);
        switch (wParam)
```

```
{
case TASK_1_DIRECTED_GRAPH:
{
    current_task = TASK_1_DIRECTED_GRAPH;
    current_matrix = A;
    current_coords = vertices_coords;
    current_n = N;

    printf("Task 1 (directed graph)\n\n");
    printf("Adjacency matrix of directed graph (matrix 'A'):\n\n");
    PrintBooleanMatrix(N, A);
    break;
}
case TASK_1_UNDIRECTED_GRAPH:
{
    current_task = TASK_1_UNDIRECTED_GRAPH;
    current_matrix = uA;
    current_coords = vertices_coords;
    current_n = N;

    printf("Task 1 (undirected graph)\n\n");
    printf("Adjacency matrix of undirected graph (matrix 'uA'):\n\n");
    PrintBooleanMatrix(N, uA);
    break;
}
case TASK_2_DIRECTED_GRAPH:
{
    current_matrix = A;
    current_coords = vertices_coords;
    current_task = TASK_2_DIRECTED_GRAPH;
    current_n = N;

    printf("Task 2 (directed graph)\n\n");
    int **vertices_degs = GetVerticesDegsOfDGraph(N, A);
    int cols_lengths[] = { 5,12, 11 };
    char *headings[] = { "Outdegrees", "Indegrees" };
    PrintInt2dArr(N, 2,
                  cols_lengths, headings,
                  vertices_degs);
    printf("\n");
    PrintDGraphHomogeneityDeg(N, vertices_degs);
    FreeInt2dArr(N, vertices_degs);
    break;
}
case TASK_2_UNDIRECTED_GRAPH:
{
    current_task = TASK_2_UNDIRECTED_GRAPH;
    current_matrix = uA;
    current_coords = vertices_coords;
    current_n = N;

    printf("Task 2 (undirected graph)\n\n");
    int *vertices_degs = GetVerticesDegsOfUGraph(N, uA);
    int cols_lengths[] = { 5,9 };
    PrintIntArr(N, cols_lengths, "Degrees", vertices_degs);
    printf("\n");
    PrintUGraphHomogeneityDeg(N, vertices_degs);
    free(vertices_degs);
    break;
}
case TASK_3_DIRECTED_GRAPH:
{
    current_task = TASK_3_DIRECTED_GRAPH;
    current_matrix = A;
```

```
            current_coords = vertices_coords;
            current_n = N;

            printf("Task 3 (directed graph)\n\n");
            int **vertices_degs = GetVerticesDegsOfDGraph(N, A);
            PrintLIVerticesOfDGraph(N, vertices_degs);
            FreeInt2dArr(N, vertices_degs);
            break;
        }
        case TASK_3_UNDIRECTED_GRAPH:
        {
            current_task = TASK_3_UNDIRECTED_GRAPH;
            current_matrix = uA;
            current_coords = vertices_coords;
            current_n = N;

            printf("Task 3 (undirected graph)\n\n");
            int *vertices_degs = GetVerticesDegsOfUGraph(N, uA);
            PrintLIVerticesOfUGraph(N, vertices_degs);
            free(vertices_degs);
            break;
        }
        case TASK_4_1:
        {
            current_task = TASK_4_1;
            current_matrix = mA;
            current_coords = vertices_coords;
            current_n = N;

            printf("Task 4.1\n\n");
            printf("Modified adjacency matrix of graph (matrix 'mA'):\n\n");
            PrintBooleanMatrix(N, mA);
            break;
        }
        case TASK_4_2:
        {
            current_task = TASK_4_2;
            current_matrix = mA;
            current_coords = vertices_coords;
            current_n = N;

            printf("Task 4.2\n\n");
            int **vertices_degs = GetVerticesDegsOfDGraph(N, mA);
            int cols_lengths[] = { 5,12, 11 };
            /* Перший стовпчик буде використаний для нумерації рядків */
            char *headings[] = { "Outdegrees", "Indegrees" };
            PrintInt2dArr(N, 2,
                          cols_lengths, headings,
                          vertices_degs);
            FreeInt2dArr(N, vertices_degs);
            break;
        }
        case TASK_4_3_1:
        {
            current_task = TASK_4_3_1;
            current_matrix = mA;
            current_coords = vertices_coords;
            current_n = N;

            printf("Task 4.3.1\n\n");
            printf("Paths of length 2:\n");
            int **mA2 =MultSquareMatrices(N, mA, mA);
            PrintPathsWithLen2(N, mA, mA2);
            FreeInt2dArr(N, mA2);
```

```
        break;
    }
    case TASK_4_3_2:
    {
        current_task = TASK_4_3_2;
        current_matrix = mA;
        current_coords = vertices_coords;
        current_n = N;

        printf("Task 4.3.2\n\n");
        printf("Paths of length 3:\n");
        int **mA2 = MultSquareMatrices(N, mA, mA);
        int **mA3 = MultSquareMatrices(N,mA2,mA);
        PrintPathsWithLen3(N, mA, mA3);
        FreeInt2dArr(N, mA2);
        FreeInt2dArr(N, mA3);
        break;
    }
    case TASK_4_4:
    {
        current_task = TASK_4_4;
        current_matrix = mA;
        current_coords = vertices_coords;
        current_n = N;

        printf("Task 4.4\n\n");
        printf("Reachability matrix:\n\n");
        int **R = GetReachabilityMatrix(N, mA);
        PrintBooleanMatrix(N, R);
        FreeInt2dArr(N, R);
        break;
    }
    case TASK_4_5:
    {
        current_task = TASK_4_5;
        current_matrix = mA;
        current_coords = vertices_coords;
        current_n = N;

        printf("Task 4.5\n\n");
        printf("Components of strong connectivity:\n");
        int **R = GetReachabilityMatrix(N, mA);
        int **Rt = TransposeSquareMatrix(N, R);
        int **S = MultSquareMatricesElemByElem(N, R, Rt);
        int **components = GetConnectComponents(N, S);
        PrintConnectComponents(N, components);
        FreeInt2dArr(N, R);
        FreeInt2dArr(N, Rt);
        FreeInt2dArr(N, S);
        FreeInt2dArr(N, components);
        break;
    }
    case TASK_4_6:
    {
        current_task = TASK_4_6;
        current_matrix = mA;
        current_coords = vertices_coords;
        current_n = N;

        printf("Task 4.6\n\n");
        printf("Connectivity matrix:\n\n");
        int **R = GetReachabilityMatrix(N, mA);
        int **Rt = TransposeSquareMatrix(N, R);
        int **S = MultSquareMatricesElemByElem(N, R, Rt);
```

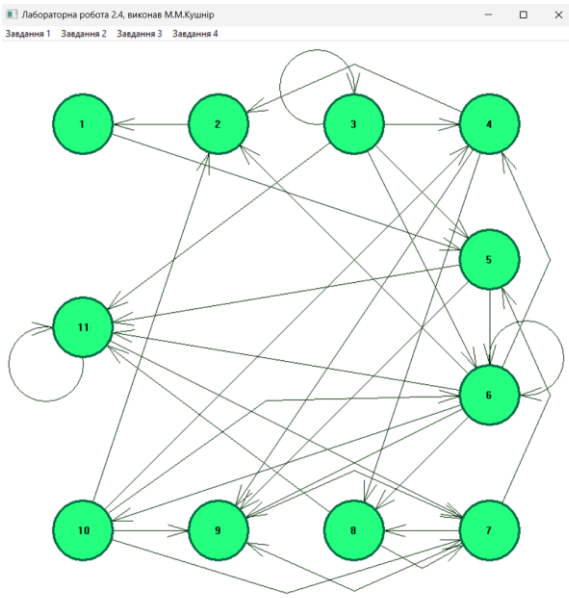
```
            PrintBooleanMatrix(N, S);
            FreeInt2dArr(N, R);
            FreeInt2dArr(N, Rt);
            FreeInt2dArr(N, S);
            break;
        }
        case TASK_4_7:
        {
            current_task = TASK_4_7;
            printf("Task 4.7\n\n");
            int **R = GetReachabilityMatrix(N, mA);
            int **Rt = TransposeSquareMatrix(N, R);
            int **S = MultSquareMatricesElemByElem(N, R, Rt);
            int **components = GetConnectComponents(N, S);
            int **graph_matrix = GetCondensationGraphMatrix(N, components, mA);
            int components_ct = 0;
            while (components_ct < N && components[components_ct][N] != 0)
                components_ct++;
            current_n = components_ct;
            printf("Adjacency matrix of the condensation graph:\n\n");
            PrintBooleanMatrix(current_n, graph_matrix);
            components_coords = SetVerticesCoords(current_n);
            current_matrix = graph_matrix;
            current_coords = components_coords;
            FreeInt2dArr(N, R);
            FreeInt2dArr(N, Rt);
            FreeInt2dArr(N, S);
            FreeInt2dArr(N, components);
            break;
        }
        }
    }
    case WM_PAINT:
    {
        hdc = BeginPaint(hWnd, &ps);
        SetBkMode(hdc, TRANSPARENT);
        HPEN ePen = CreatePen(PS_SOLID, 1, RGB(0, 38, 0));
        HPEN vOutlinePen = CreatePen(PS_SOLID, 3, RGB(3, 104, 65));
        HBRUSH vFillBrush = CreateSolidBrush(RGB(37, 255, 127));
        if (current_task)
            DrawGraph(current_n,
                      current_matrix,
                      current_coords,
                      ePen,
                      vFillBrush,
                      vOutlinePen,
                      hdc);
        if (current_n != N)
            FreeInt2dArr(current_n, components_coords);
        DeleteObject(ePen);
        DeleteObject(vOutlinePen);
        DeleteObject(vFillBrush);
        EndPaint(hWnd, &ps);
        break;
    }
    case WM_CREATE:
        AddMenu(hWnd);
        vertices_coords = SetVerticesCoords(N);
        srand(N1 * 1000 + N2 * 100 + N3 * 10 + N4);
        double **T = randm(N, N);
        A = mulmr(N, N, T,
                  (1.0 - N3 * 0.01 - N4 * 0.01 - 0.3));
        uA = SymmetrizeMatrix(N, A);
        mA = mulmr(N, N, T,
```

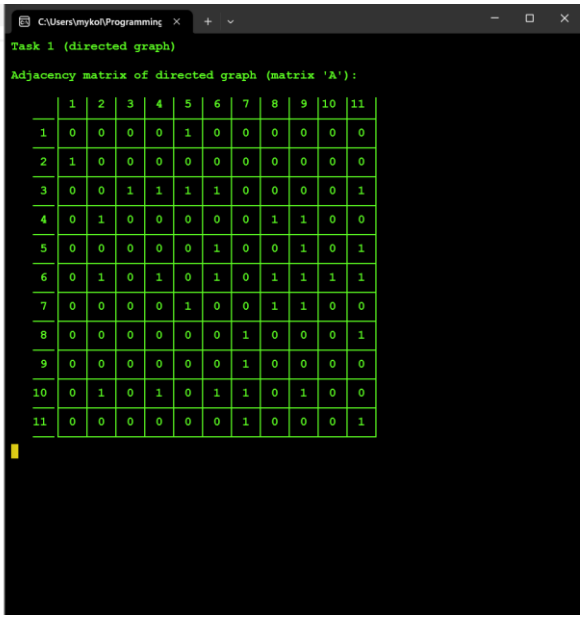
```
                    (1.0 - N3 * 0.005 - N4 * 0.005 - 0.27));
            printf("Matrix T: \n\n");
            PrintDouble2dArr(N, N, T);
            printf("\n*You can select an option from the window pop-up menu\n");
            FreeDouble2dArr(N, T);
            break;
        case WM_DESTROY:
            FreeInt2dArr(N, vertices_coords);
            FreeInt2dArr(N, A);
            FreeInt2dArr(N, uA);
            FreeInt2dArr(N, mA);
            PostQuitMessage(0);
            break;
        default :
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
}

void AddMenu(HWND hwnd)
{
    HMENU hMenu = CreateMenu();
    /* Підменю для завдання 1 */
    HMENU hTask1 = CreateMenu();
    AppendMenu(hTask1, MF_STRING, TASK_1_DIRECTED_GRAPH,
               L"1) намалювати напрямлений граф");
    AppendMenu(hTask1, MF_STRING, TASK_1_UNDIRECTED_GRAPH,
               L"2) намалювати ненапрямлений граф");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hTask1,
               L"Завдання 1");
    /* Підменю для завдання 2 */
    HMENU hTask2 = CreateMenu();
    AppendMenu(hTask2, MF_STRING, TASK_2_DIRECTED_GRAPH,
               L"1) визначити напівстепені вершин напрямленого графа");
    AppendMenu(hTask2, MF_STRING, TASK_2_UNDIRECTED_GRAPH,
               L"2) визначити степені вершин ненапрямленого графа");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hTask2,
               L"Завдання 2");
    /* Підменю для завдання 3 */
    HMENU hTask3 = CreateMenu();
    AppendMenu(hTask3, MF_STRING, TASK_3_DIRECTED_GRAPH,
               L"1) визначити висячі та ізольовані вершини напрямленого графа");
    AppendMenu(hTask3, MF_STRING, TASK_3_UNDIRECTED_GRAPH,
               L"2) визначити висячі та ізольовані вершини ненапрямленого
графа");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hTask3,
               L"Завдання 3");
    /* Підменю для завдання 4 */
    HMENU hTask4 = CreateMenu();
    AppendMenu(hTask4, MF_STRING, TASK_4_1,
               L"1) обчислити змінену матрицю суміжності");
    AppendMenu(hTask4, MF_STRING, TASK_4_2,
               L"2) визначити напівстепені вузлів");
    /* Підменю для завдання 4-3) */
    HMENU hTask4_3 = CreateMenu();
    AppendMenu(hTask4_3, MF_STRING, TASK_4_3_1,
               L" 2 ");
    AppendMenu(hTask4_3, MF_STRING, TASK_4_3_2,
               L" 3 ");
    AppendMenu(hTask4, MF_POPUP, (UINT_PTR)hTask4_3,
               L"3) знайти всі шляхи довжиною ...");
    /*****************************/
    AppendMenu(hTask4, MF_STRING, TASK_4_4,
               L"4) обчислити матрицю досяжності");
    AppendMenu(hTask4, MF_STRING, TASK_4_5,
```

```
            L"5) знайти компоненти сильної зв'язності");
    AppendMenu(hTask4, MF_STRING, TASK_4_6,
            L"6) обчислити матрицю зв'язності");
    AppendMenu(hTask4, MF_STRING, TASK_4_7,
            L"7) намалювати граф конденсації");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hTask4,
            L"Завдання 4");
    SetMenu(hwnd, hMenu);
}
```

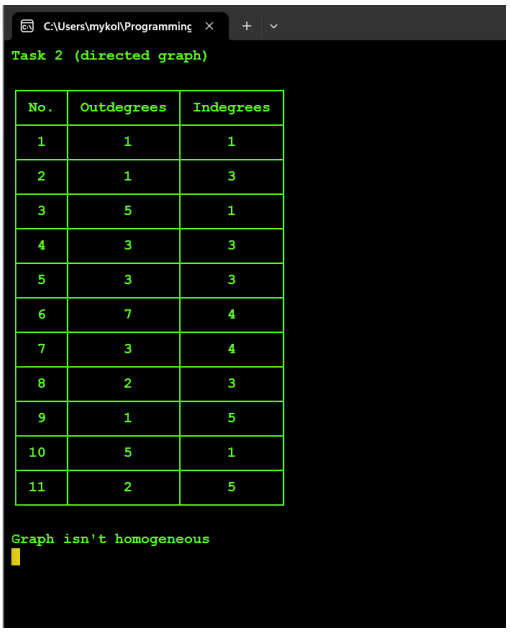
# *Результати тестування програми*

## *Напрямлений (не модифікований) граф, його матриця суміжності, таблиця напівстепенів вузлів, перелік висячих та ізольованих вершин*



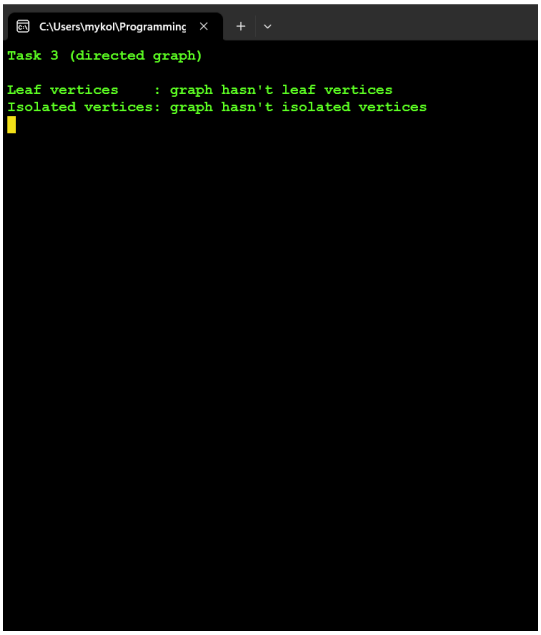




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

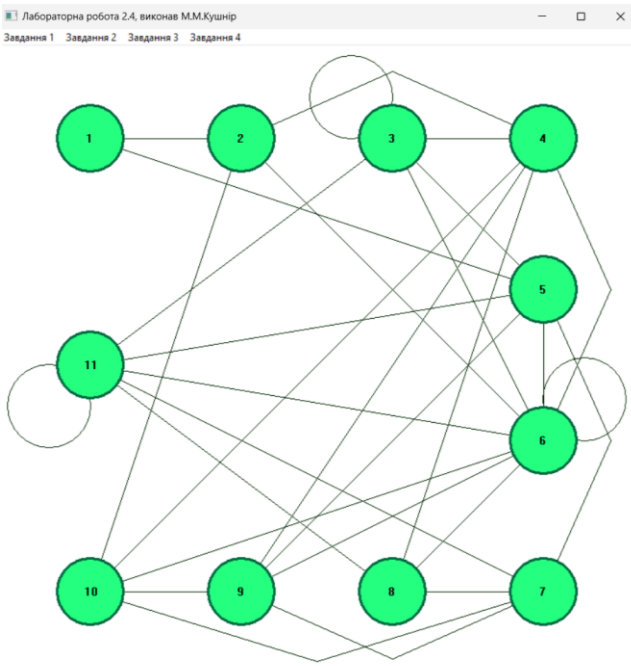



| No. | Outdegrees | Indegrees |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 3 |
| 3 | 5 | 1 |
| 4 | 3 | 3 |
| 5 | 3 | 3 |
| 6 | 7 | 4 |
| 7 | 3 | 4 |
| 8 | 2 | 3 |
| 9 | 1 | 5 |
| 10 | 5 | 1 |
| 11 | 2 | 5 |





## *Ненапрямлений граф, його матриця суміжності, таблиця степенів вузлів, перелік висячих та ізольованих вершин*




```
C:\Users\mykol\Programming
Task 1 (undirected graph)

Adjacency matrix of undirected graph (matrix 'uA'):
```

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 11 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

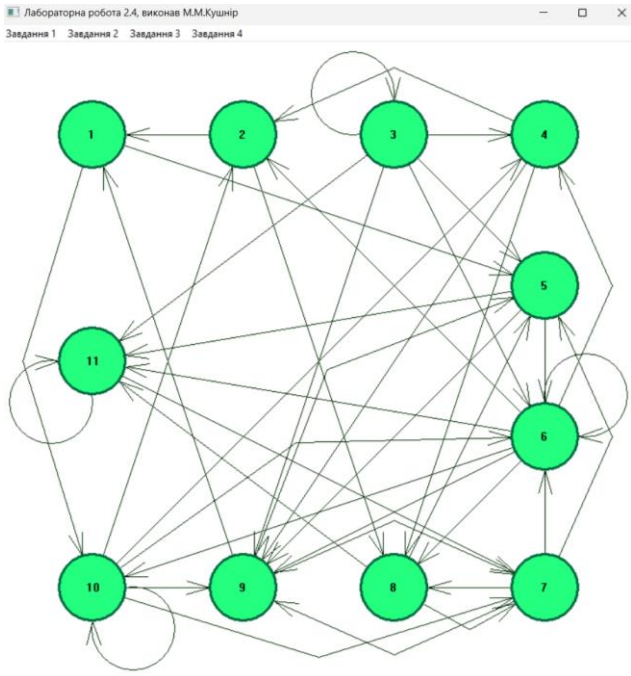
```
C:\Users\mykol\Programming
Task 2 (undirected graph)
```

| No. | Degrees |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 6 |
| 5 | 6 |
| 6 | 10 |
| 7 | 5 |
| 8 | 4 |
| 9 | 5 |
| 10 | 5 |
| 11 | 7 |

```
Graph isn't homogeneous
```

```
C:\Users\mykol\Programming
Task 3 (undirected graph)

Leaf vertices     : graph hasn't leaf vertices
Isolated vertices: graph hasn't isolated vertices
```

## *Модифікований напрямлений граф та його матриця суміжності*




```
C:\Users\mykol\Programming
Task 4.1

Modified adjacency matrix of graph (matrix 'mA'):
```

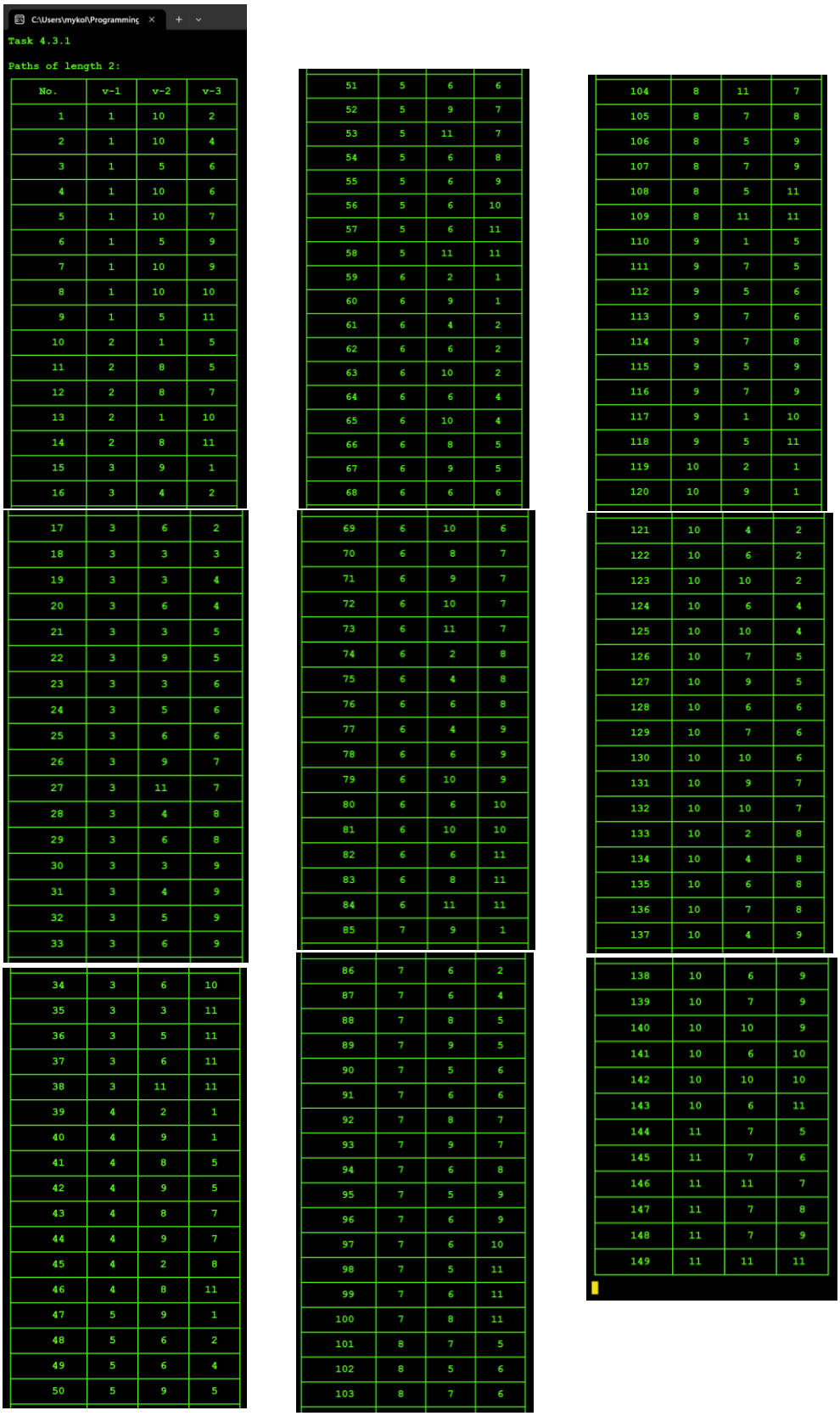
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

## *Таблиця степенів вузлів модифікованого графа*

```
C:\Users\mykol\Programming
Task 4.2
```

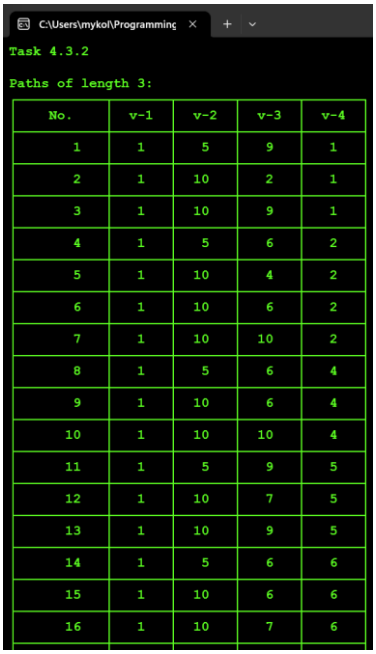
| No. | Outdegrees | Indegrees |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 2 | 3 |
| 3 | 6 | 1 |
| 4 | 3 | 3 |
| 5 | 3 | 5 |
| 6 | 7 | 5 |
| 7 | 4 | 4 |
| 8 | 3 | 4 |
| 9 | 3 | 6 |
| 10 | 6 | 3 |
| 11 | 2 | 5 |

# *Шляхи довжиною 2*




| No. | v-1 | v-2 | v-3 |
|---|---|---|---|
| 1 | 1 | 10 | 2 |
| 2 | 1 | 10 | 4 |
| 3 | 1 | 5 | 6 |
| 4 | 1 | 10 | 6 |
| 5 | 1 | 10 | 7 |
| 6 | 1 | 5 | 9 |
| 7 | 1 | 10 | 9 |
| 8 | 1 | 10 | 10 |
| 9 | 1 | 5 | 11 |
| 10 | 2 | 1 | 5 |
| 11 | 2 | 8 | 5 |
| 12 | 2 | 8 | 7 |
| 13 | 2 | 1 | 10 |
| 14 | 2 | 8 | 11 |
| 15 | 3 | 9 | 1 |
| 16 | 3 | 4 | 2 |
| 17 | 3 | 6 | 2 |
| 18 | 3 | 3 | 3 |
| 19 | 3 | 3 | 4 |
| 20 | 3 | 6 | 4 |
| 21 | 3 | 3 | 5 |
| 22 | 3 | 9 | 5 |
| 23 | 3 | 3 | 6 |
| 24 | 3 | 5 | 6 |
| 25 | 3 | 6 | 6 |
| 26 | 3 | 9 | 7 |
| 27 | 3 | 11 | 7 |
| 28 | 3 | 4 | 8 |
| 29 | 3 | 6 | 8 |
| 30 | 3 | 3 | 9 |
| 31 | 3 | 4 | 9 |
| 32 | 3 | 5 | 9 |
| 33 | 3 | 6 | 9 |
| 34 | 3 | 6 | 10 |
| 35 | 3 | 3 | 11 |
| 36 | 3 | 5 | 11 |
| 37 | 3 | 6 | 11 |
| 38 | 3 | 11 | 11 |
| 39 | 4 | 2 | 1 |
| 40 | 4 | 9 | 1 |
| 41 | 4 | 8 | 5 |
| 42 | 4 | 9 | 5 |
| 43 | 4 | 8 | 7 |
| 44 | 4 | 9 | 7 |
| 45 | 4 | 2 | 8 |
| 46 | 4 | 8 | 11 |
| 47 | 5 | 9 | 1 |
| 48 | 5 | 6 | 2 |
| 49 | 5 | 6 | 4 |
| 50 | 5 | 9 | 5 |
| 51 | 5 | 6 | 6 |
| 52 | 5 | 9 | 7 |
| 53 | 5 | 11 | 7 |
| 54 | 5 | 6 | 8 |
| 55 | 5 | 6 | 9 |
| 56 | 5 | 6 | 10 |
| 57 | 5 | 6 | 11 |
| 58 | 5 | 11 | 11 |
| 59 | 6 | 2 | 1 |
| 60 | 6 | 9 | 1 |
| 61 | 6 | 4 | 2 |
| 62 | 6 | 6 | 2 |
| 63 | 6 | 10 | 2 |
| 64 | 6 | 6 | 4 |
| 65 | 6 | 10 | 4 |
| 66 | 6 | 8 | 5 |
| 67 | 6 | 9 | 5 |
| 68 | 6 | 6 | 6 |
| 69 | 6 | 10 | 6 |
| 70 | 6 | 8 | 7 |
| 71 | 6 | 9 | 7 |
| 72 | 6 | 10 | 7 |
| 73 | 6 | 11 | 7 |
| 74 | 6 | 2 | 8 |
| 75 | 6 | 4 | 8 |
| 76 | 6 | 6 | 8 |
| 77 | 6 | 4 | 9 |
| 78 | 6 | 6 | 9 |
| 79 | 6 | 10 | 9 |
| 80 | 6 | 6 | 10 |
| 81 | 6 | 10 | 10 |
| 82 | 6 | 6 | 11 |
| 83 | 6 | 8 | 11 |
| 84 | 6 | 11 | 11 |
| 85 | 7 | 9 | 1 |
| 86 | 7 | 6 | 2 |
| 87 | 7 | 6 | 4 |
| 88 | 7 | 8 | 5 |
| 89 | 7 | 9 | 5 |
| 90 | 7 | 5 | 6 |
| 91 | 7 | 6 | 6 |
| 92 | 7 | 8 | 7 |
| 93 | 7 | 9 | 7 |
| 94 | 7 | 6 | 8 |
| 95 | 7 | 5 | 9 |
| 96 | 7 | 6 | 9 |
| 97 | 7 | 6 | 10 |
| 98 | 7 | 5 | 11 |
| 99 | 7 | 6 | 11 |
| 100 | 7 | 8 | 11 |
| 101 | 8 | 7 | 5 |
| 102 | 8 | 5 | 6 |
| 103 | 8 | 7 | 6 |
| 104 | 8 | 11 | 7 |
| 105 | 8 | 7 | 8 |
| 106 | 8 | 5 | 9 |
| 107 | 8 | 7 | 9 |
| 108 | 8 | 5 | 11 |
| 109 | 8 | 11 | 11 |
| 110 | 9 | 1 | 5 |
| 111 | 9 | 7 | 5 |
| 112 | 9 | 5 | 6 |
| 113 | 9 | 7 | 6 |
| 114 | 9 | 7 | 8 |
| 115 | 9 | 5 | 9 |
| 116 | 9 | 7 | 9 |
| 117 | 9 | 1 | 10 |
| 118 | 9 | 5 | 11 |
| 119 | 10 | 2 | 1 |
| 120 | 10 | 9 | 1 |
| 121 | 10 | 4 | 2 |
| 122 | 10 | 6 | 2 |
| 123 | 10 | 10 | 2 |
| 124 | 10 | 6 | 4 |
| 125 | 10 | 10 | 4 |
| 126 | 10 | 7 | 5 |
| 127 | 10 | 9 | 5 |
| 128 | 10 | 6 | 6 |
| 129 | 10 | 7 | 6 |
| 130 | 10 | 10 | 6 |
| 131 | 10 | 9 | 7 |
| 132 | 10 | 10 | 7 |
| 133 | 10 | 2 | 8 |
| 134 | 10 | 4 | 8 |
| 135 | 10 | 6 | 8 |
| 136 | 10 | 7 | 8 |
| 137 | 10 | 4 | 9 |
| 138 | 10 | 6 | 9 |
| 139 | 10 | 7 | 9 |
| 140 | 10 | 10 | 9 |
| 141 | 10 | 6 | 10 |
| 142 | 10 | 10 | 10 |
| 143 | 10 | 6 | 11 |
| 144 | 11 | 7 | 5 |
| 145 | 11 | 7 | 6 |
| 146 | 11 | 11 | 7 |
| 147 | 11 | 7 | 8 |
| 148 | 11 | 7 | 9 |
| 149 | 11 | 11 | 11 |

# *Шляхи довжиною 3*




| No. | v-1 | v-2 | v-3 | v-4 |
|---|---|---|---|---|
| 1 | 1 | 5 | 9 | 1 |
| 2 | 1 | 10 | 2 | 1 |
| 3 | 1 | 10 | 9 | 1 |
| 4 | 1 | 5 | 6 | 2 |
| 5 | 1 | 10 | 4 | 2 |
| 6 | 1 | 10 | 6 | 2 |
| 7 | 1 | 10 | 10 | 2 |
| 8 | 1 | 5 | 6 | 4 |
| 9 | 1 | 10 | 6 | 4 |
| 10 | 1 | 10 | 10 | 4 |
| 11 | 1 | 5 | 9 | 5 |
| 12 | 1 | 10 | 7 | 5 |
| 13 | 1 | 10 | 9 | 5 |
| 14 | 1 | 5 | 6 | 6 |
| 15 | 1 | 10 | 6 | 6 |
| 16 | 1 | 10 | 7 | 6 |
| 17 | 1 | 10 | 10 | 6 |
| 18 | 1 | 5 | 9 | 7 |
| 19 | 1 | 5 | 11 | 7 |
| 20 | 1 | 10 | 9 | 7 |
| 21 | 1 | 10 | 10 | 7 |
| 22 | 1 | 5 | 6 | 8 |
| 23 | 1 | 10 | 2 | 8 |
| 24 | 1 | 10 | 4 | 8 |
| 25 | 1 | 10 | 6 | 8 |
| 26 | 1 | 10 | 7 | 8 |
| 27 | 1 | 5 | 6 | 9 |
| 28 | 1 | 10 | 4 | 9 |
| 29 | 1 | 10 | 6 | 9 |
| 30 | 1 | 10 | 7 | 9 |
| 31 | 1 | 10 | 10 | 9 |
| 32 | 1 | 5 | 6 | 10 |
| 33 | 1 | 10 | 6 | 10 |
| 34 | 1 | 10 | 10 | 10 |
| 35 | 1 | 5 | 6 | 11 |
| 36 | 1 | 5 | 11 | 11 |
| 37 | 1 | 10 | 6 | 11 |
| 38 | 2 | 1 | 10 | 2 |
| 39 | 2 | 1 | 10 | 4 |
| 40 | 2 | 8 | 7 | 5 |
| 41 | 2 | 1 | 5 | 6 |
| 42 | 2 | 1 | 10 | 6 |
| 43 | 2 | 8 | 5 | 6 |
| 44 | 2 | 8 | 7 | 6 |
| 45 | 2 | 1 | 10 | 7 |
| 46 | 2 | 8 | 11 | 7 |
| 47 | 2 | 8 | 7 | 8 |
| 48 | 2 | 1 | 5 | 9 |
| 49 | 2 | 1 | 10 | 9 |
| 50 | 2 | 8 | 5 | 9 |
| 51 | 2 | 8 | 7 | 9 |
| 52 | 2 | 1 | 10 | 10 |
| 53 | 2 | 1 | 5 | 11 |
| 54 | 2 | 8 | 5 | 11 |
| 55 | 2 | 8 | 11 | 11 |
| 56 | 3 | 3 | 9 | 1 |
| 57 | 3 | 4 | 2 | 1 |
| 58 | 3 | 4 | 9 | 1 |
| 59 | 3 | 5 | 9 | 1 |
| 60 | 3 | 6 | 2 | 1 |
| 61 | 3 | 6 | 9 | 1 |
| 62 | 3 | 3 | 4 | 2 |
| 63 | 3 | 3 | 6 | 2 |
| 64 | 3 | 5 | 6 | 2 |
| 65 | 3 | 6 | 4 | 2 |
| 66 | 3 | 6 | 6 | 2 |
| 67 | 3 | 6 | 10 | 2 |
| 68 | 3 | 3 | 3 | 3 |
| 69 | 3 | 3 | 3 | 4 |
| 70 | 3 | 3 | 6 | 4 |
| 71 | 3 | 5 | 6 | 4 |
| 72 | 3 | 6 | 6 | 4 |
| 73 | 3 | 6 | 10 | 4 |
| 74 | 3 | 3 | 3 | 5 |
| 75 | 3 | 3 | 9 | 5 |
| 76 | 3 | 4 | 8 | 5 |
| 77 | 3 | 4 | 9 | 5 |
| 78 | 3 | 5 | 9 | 5 |
| 79 | 3 | 6 | 8 | 5 |
| 80 | 3 | 6 | 9 | 5 |
| 81 | 3 | 9 | 1 | 5 |
| 82 | 3 | 9 | 7 | 5 |
| 83 | 3 | 11 | 7 | 5 |
| 84 | 3 | 3 | 3 | 6 |
| 85 | 3 | 3 | 5 | 6 |
| 86 | 3 | 3 | 6 | 6 |
| 87 | 3 | 5 | 6 | 6 |
| 88 | 3 | 6 | 6 | 6 |
| 89 | 3 | 6 | 10 | 6 |
| 90 | 3 | 9 | 5 | 6 |
| 91 | 3 | 9 | 7 | 6 |
| 92 | 3 | 11 | 7 | 6 |
| 93 | 3 | 3 | 9 | 7 |
| 94 | 3 | 3 | 11 | 7 |
| 95 | 3 | 4 | 8 | 7 |
| 96 | 3 | 4 | 9 | 7 |
| 97 | 3 | 5 | 9 | 7 |
| 98 | 3 | 5 | 11 | 7 |
| 99 | 3 | 6 | 8 | 7 |
| 100 | 3 | 6 | 9 | 7 |
| 101 | 3 | 6 | 10 | 7 |
| 102 | 3 | 6 | 11 | 7 |
| 103 | 3 | 11 | 11 | 7 |
| 104 | 3 | 3 | 4 | 8 |
| 105 | 3 | 3 | 6 | 8 |
| 106 | 3 | 4 | 2 | 8 |
| 107 | 3 | 5 | 6 | 8 |
| 108 | 3 | 6 | 2 | 8 |
| 109 | 3 | 6 | 4 | 8 |
| 110 | 3 | 6 | 6 | 8 |
| 111 | 3 | 9 | 7 | 8 |
| 112 | 3 | 11 | 7 | 8 |
| 113 | 3 | 3 | 3 | 9 |
| 114 | 3 | 3 | 4 | 9 |
| 115 | 3 | 3 | 5 | 9 |
| 116 | 3 | 3 | 6 | 9 |
| 117 | 3 | 5 | 6 | 9 |
| 118 | 3 | 6 | 4 | 9 |
| 119 | 3 | 6 | 6 | 9 |
| 120 | 3 | 6 | 10 | 9 |
| 121 | 3 | 9 | 5 | 9 |
| 122 | 3 | 9 | 7 | 9 |
| 123 | 3 | 11 | 7 | 9 |
| 124 | 3 | 3 | 6 | 10 |
| 125 | 3 | 5 | 6 | 10 |
| 126 | 3 | 6 | 6 | 10 |
| 127 | 3 | 6 | 10 | 10 |
| 128 | 3 | 9 | 1 | 10 |
| 129 | 3 | 3 | 3 | 11 |
| 130 | 3 | 3 | 5 | 11 |
| 131 | 3 | 3 | 6 | 11 |
| 132 | 3 | 3 | 11 | 11 |
| 133 | 3 | 4 | 8 | 11 |
| 134 | 3 | 5 | 6 | 11 |
| 135 | 3 | 5 | 11 | 11 |
| 136 | 3 | 6 | 6 | 11 |
| 137 | 3 | 6 | 8 | 11 |
| 138 | 3 | 6 | 11 | 11 |
| 139 | 3 | 9 | 5 | 11 |
| 140 | 3 | 11 | 11 | 11 |
| 141 | 4 | 2 | 1 | 5 |
| 142 | 4 | 2 | 8 | 5 |
| 143 | 4 | 8 | 7 | 5 |
| 144 | 4 | 9 | 1 | 5 |
| 145 | 4 | 9 | 7 | 5 |
| 146 | 4 | 8 | 5 | 6 |
| 147 | 4 | 8 | 7 | 6 |
| 148 | 4 | 9 | 5 | 6 |
| 149 | 4 | 9 | 7 | 6 |
| 150 | 4 | 2 | 8 | 7 |
| 151 | 4 | 8 | 11 | 7 |
| 152 | 4 | 8 | 7 | 8 |
| 153 | 4 | 9 | 7 | 8 |
| 154 | 4 | 8 | 5 | 9 |
| 155 | 4 | 8 | 7 | 9 |
| 156 | 4 | 9 | 5 | 9 |
| 157 | 4 | 9 | 7 | 9 |
| 158 | 4 | 2 | 1 | 10 |

| | | | | |
|---|---|---|---|---|
| 159 | 4 | 9 | 1 | 10 |
| 160 | 4 | 2 | 8 | 11 |
| 161 | 4 | 8 | 5 | 11 |
| 162 | 4 | 8 | 11 | 11 |
| 163 | 4 | 9 | 5 | 11 |
| 164 | 5 | 6 | 2 | 1 |
| 165 | 5 | 6 | 9 | 1 |
| 166 | 5 | 6 | 4 | 2 |
| 167 | 5 | 6 | 6 | 2 |
| 168 | 5 | 6 | 10 | 2 |
| 169 | 5 | 6 | 6 | 4 |
| 170 | 5 | 6 | 10 | 4 |
| 171 | 5 | 6 | 8 | 5 |
| 172 | 5 | 6 | 9 | 5 |
| 173 | 5 | 9 | 1 | 5 |
| 174 | 5 | 9 | 7 | 5 |
| 175 | 5 | 11 | 7 | 5 |

| | | | | |
|---|---|---|---|---|
| 176 | 5 | 6 | 6 | 6 |
| 177 | 5 | 6 | 10 | 6 |
| 178 | 5 | 9 | 5 | 6 |
| 179 | 5 | 9 | 7 | 6 |
| 180 | 5 | 11 | 7 | 6 |
| 181 | 5 | 6 | 8 | 7 |
| 182 | 5 | 6 | 9 | 7 |
| 183 | 5 | 6 | 10 | 7 |
| 184 | 5 | 6 | 11 | 7 |
| 185 | 5 | 11 | 11 | 7 |
| 186 | 5 | 6 | 2 | 8 |
| 187 | 5 | 6 | 4 | 8 |
| 188 | 5 | 6 | 6 | 8 |
| 189 | 5 | 9 | 7 | 8 |
| 190 | 5 | 11 | 7 | 8 |
| 191 | 5 | 6 | 4 | 9 |
| 192 | 5 | 6 | 6 | 9 |
| 193 | 5 | 6 | 10 | 9 |

| | | | | |
|---|---|---|---|---|
| 194 | 5 | 9 | 5 | 9 |
| 195 | 5 | 9 | 7 | 9 |
| 196 | 5 | 11 | 7 | 9 |
| 197 | 5 | 6 | 6 | 10 |
| 198 | 5 | 6 | 10 | 10 |
| 199 | 5 | 9 | 1 | 10 |
| 200 | 5 | 6 | 6 | 11 |
| 201 | 5 | 6 | 8 | 11 |
| 202 | 5 | 6 | 11 | 11 |
| 203 | 5 | 9 | 5 | 11 |
| 204 | 5 | 11 | 11 | 11 |
| 205 | 6 | 4 | 2 | 1 |
| 206 | 6 | 4 | 9 | 1 |
| 207 | 6 | 6 | 2 | 1 |
| 208 | 6 | 6 | 9 | 1 |
| 209 | 6 | 10 | 2 | 1 |
| 210 | 6 | 10 | 9 | 1 |

| | | | | |
|---|---|---|---|---|
| 211 | 6 | 6 | 4 | 2 |
| 212 | 6 | 6 | 6 | 2 |
| 213 | 6 | 6 | 10 | 2 |
| 214 | 6 | 10 | 4 | 2 |
| 215 | 6 | 10 | 6 | 2 |
| 216 | 6 | 10 | 10 | 2 |
| 217 | 6 | 6 | 6 | 4 |
| 218 | 6 | 6 | 10 | 4 |
| 219 | 6 | 10 | 6 | 4 |
| 220 | 6 | 10 | 10 | 4 |
| 221 | 6 | 2 | 1 | 5 |
| 222 | 6 | 2 | 8 | 5 |
| 223 | 6 | 4 | 8 | 5 |
| 224 | 6 | 4 | 9 | 5 |
| 225 | 6 | 6 | 8 | 5 |
| 226 | 6 | 6 | 9 | 5 |
| 227 | 6 | 8 | 7 | 5 |
| 228 | 6 | 9 | 1 | 5 |

| | | | | |
|---|---|---|---|---|
| 229 | 6 | 9 | 7 | 5 |
| 230 | 6 | 10 | 7 | 5 |
| 231 | 6 | 10 | 9 | 5 |
| 232 | 6 | 11 | 7 | 5 |
| 233 | 6 | 6 | 6 | 6 |
| 234 | 6 | 6 | 10 | 6 |
| 235 | 6 | 8 | 5 | 6 |
| 236 | 6 | 8 | 7 | 6 |
| 237 | 6 | 9 | 5 | 6 |
| 238 | 6 | 9 | 7 | 6 |
| 239 | 6 | 10 | 6 | 6 |
| 240 | 6 | 10 | 7 | 6 |
| 241 | 6 | 10 | 10 | 6 |
| 242 | 6 | 11 | 7 | 6 |
| 243 | 6 | 2 | 8 | 7 |
| 244 | 6 | 4 | 8 | 7 |
| 245 | 6 | 4 | 9 | 7 |

| | | | | |
|---|---|---|---|---|
| 246 | 6 | 6 | 8 | 7 |
| 247 | 6 | 6 | 9 | 7 |
| 248 | 6 | 6 | 10 | 7 |
| 249 | 6 | 6 | 11 | 7 |
| 250 | 6 | 8 | 11 | 7 |
| 251 | 6 | 10 | 9 | 7 |
| 252 | 6 | 10 | 10 | 7 |
| 253 | 6 | 11 | 11 | 7 |
| 254 | 6 | 4 | 2 | 8 |
| 255 | 6 | 6 | 2 | 8 |
| 256 | 6 | 6 | 4 | 8 |
| 257 | 6 | 6 | 6 | 8 |
| 258 | 6 | 8 | 7 | 8 |
| 259 | 6 | 9 | 7 | 8 |
| 260 | 6 | 10 | 2 | 8 |
| 261 | 6 | 10 | 4 | 8 |
| 262 | 6 | 10 | 6 | 8 |
| 263 | 6 | 10 | 7 | 8 |

| | | | | |
|---|---|---|---|---|
| 264 | 6 | 11 | 7 | 8 |
| 265 | 6 | 6 | 4 | 9 |
| 266 | 6 | 6 | 6 | 9 |
| 267 | 6 | 6 | 10 | 9 |
| 268 | 6 | 8 | 5 | 9 |
| 269 | 6 | 8 | 7 | 9 |
| 270 | 6 | 9 | 5 | 9 |
| 271 | 6 | 9 | 7 | 9 |
| 272 | 6 | 10 | 4 | 9 |
| 273 | 6 | 10 | 6 | 9 |
| 274 | 6 | 10 | 7 | 9 |
| 275 | 6 | 10 | 10 | 9 |
| 276 | 6 | 11 | 7 | 9 |
| 277 | 6 | 2 | 1 | 10 |
| 278 | 6 | 6 | 6 | 10 |
| 279 | 6 | 6 | 10 | 10 |
| 280 | 6 | 9 | 1 | 10 |

| | | | | |
|---|---|---|---|---|
| 281 | 6 | 10 | 6 | 10 |
| 282 | 6 | 10 | 10 | 10 |
| 283 | 6 | 2 | 8 | 11 |
| 284 | 6 | 4 | 8 | 11 |
| 285 | 6 | 6 | 6 | 11 |
| 286 | 6 | 6 | 8 | 11 |
| 287 | 6 | 6 | 11 | 11 |
| 288 | 6 | 8 | 5 | 11 |
| 289 | 6 | 8 | 11 | 11 |
| 290 | 6 | 9 | 5 | 11 |
| 291 | 6 | 10 | 6 | 11 |
| 292 | 6 | 11 | 11 | 11 |
| 293 | 7 | 5 | 9 | 1 |
| 294 | 7 | 6 | 2 | 1 |
| 295 | 7 | 6 | 9 | 1 |
| 296 | 7 | 5 | 6 | 2 |
| 297 | 7 | 6 | 4 | 2 |
| 298 | 7 | 6 | 6 | 2 |

| | | | | |
|---|---|---|---|---|
| 299 | 7 | 6 | 10 | 2 |
| 300 | 7 | 5 | 6 | 4 |
| 301 | 7 | 6 | 6 | 4 |
| 302 | 7 | 6 | 10 | 4 |
| 303 | 7 | 5 | 9 | 5 |
| 304 | 7 | 6 | 8 | 5 |
| 305 | 7 | 6 | 9 | 5 |
| 306 | 7 | 8 | 7 | 5 |
| 307 | 7 | 9 | 1 | 5 |
| 308 | 7 | 9 | 7 | 5 |
| 309 | 7 | 5 | 6 | 6 |
| 310 | 7 | 6 | 6 | 6 |
| 311 | 7 | 6 | 10 | 6 |
| 312 | 7 | 8 | 5 | 6 |
| 313 | 7 | 8 | 7 | 6 |
| 314 | 7 | 9 | 5 | 6 |
| 315 | 7 | 9 | 7 | 6 |

| | | | | |
|---|---|---|---|---|
| 316 | 7 | 5 | 9 | 7 |
| 317 | 7 | 5 | 11 | 7 |
| 318 | 7 | 6 | 8 | 7 |
| 319 | 7 | 6 | 9 | 7 |
| 320 | 7 | 6 | 10 | 7 |
| 321 | 7 | 6 | 11 | 7 |
| 322 | 7 | 8 | 11 | 7 |
| 323 | 7 | 5 | 6 | 8 |
| 324 | 7 | 6 | 2 | 8 |
| 325 | 7 | 6 | 4 | 8 |
| 326 | 7 | 6 | 6 | 8 |
| 327 | 7 | 8 | 7 | 8 |
| 328 | 7 | 9 | 7 | 8 |
| 329 | 7 | 5 | 6 | 9 |
| 330 | 7 | 6 | 4 | 9 |
| 331 | 7 | 6 | 6 | 9 |
| 332 | 7 | 6 | 10 | 9 |
| 333 | 7 | 8 | 5 | 9 |

| | | | | |
|---|---|---|---|---|
| 334 | 7 | 8 | 7 | 9 |
| 335 | 7 | 9 | 5 | 9 |
| 336 | 7 | 9 | 7 | 9 |
| 337 | 7 | 5 | 6 | 10 |
| 338 | 7 | 6 | 6 | 10 |
| 339 | 7 | 6 | 10 | 10 |
| 340 | 7 | 9 | 1 | 10 |
| 341 | 7 | 5 | 6 | 11 |
| 342 | 7 | 5 | 11 | 11 |
| 343 | 7 | 6 | 6 | 11 |
| 344 | 7 | 6 | 8 | 11 |
| 345 | 7 | 6 | 11 | 11 |
| 346 | 7 | 8 | 5 | 11 |
| 347 | 7 | 8 | 11 | 11 |
| 348 | 7 | 9 | 5 | 11 |
| 349 | 8 | 5 | 9 | 1 |
| 350 | 8 | 7 | 9 | 1 |

| | | | | |
|---|---|---|---|---|
| 351 | 8 | 5 | 6 | 2 |
| 352 | 8 | 7 | 6 | 2 |
| 353 | 8 | 5 | 6 | 4 |
| 354 | 8 | 7 | 6 | 4 |
| 355 | 8 | 5 | 9 | 5 |
| 356 | 8 | 7 | 8 | 5 |
| 357 | 8 | 7 | 9 | 5 |
| 358 | 8 | 11 | 7 | 5 |
| 359 | 8 | 5 | 6 | 6 |
| 360 | 8 | 7 | 5 | 6 |
| 361 | 8 | 7 | 6 | 6 |
| 362 | 8 | 11 | 7 | 6 |
| 363 | 8 | 5 | 9 | 7 |
| 364 | 8 | 5 | 11 | 7 |
| 365 | 8 | 7 | 8 | 7 |
| 366 | 8 | 7 | 9 | 7 |
| 367 | 8 | 11 | 11 | 7 |
| 368 | 8 | 5 | 6 | 8 |

| | | | | |
|---|---|---|---|---|
| 369 | 8 | 7 | 6 | 8 |
| 370 | 8 | 11 | 7 | 8 |
| 371 | 8 | 5 | 6 | 9 |
| 372 | 8 | 7 | 5 | 9 |
| 373 | 8 | 7 | 6 | 9 |
| 374 | 8 | 11 | 7 | 9 |
| 375 | 8 | 5 | 6 | 10 |
| 376 | 8 | 7 | 6 | 10 |
| 377 | 8 | 5 | 6 | 11 |
| 378 | 8 | 5 | 11 | 11 |
| 379 | 8 | 7 | 5 | 11 |
| 380 | 8 | 7 | 6 | 11 |
| 381 | 8 | 7 | 8 | 11 |
| 382 | 8 | 11 | 11 | 11 |
| 383 | 9 | 5 | 9 | 1 |
| 384 | 9 | 7 | 9 | 1 |
| 385 | 9 | 1 | 10 | 2 |

| | | | | |
|---|---|---|---|---|
| 386 | 9 | 5 | 6 | 2 |
| 387 | 9 | 7 | 6 | 2 |
| 388 | 9 | 1 | 10 | 4 |
| 389 | 9 | 5 | 6 | 4 |
| 390 | 9 | 7 | 6 | 4 |
| 391 | 9 | 5 | 9 | 5 |
| 392 | 9 | 7 | 8 | 5 |
| 393 | 9 | 7 | 9 | 5 |
| 394 | 9 | 1 | 5 | 6 |
| 395 | 9 | 1 | 10 | 6 |
| 396 | 9 | 5 | 6 | 6 |
| 397 | 9 | 7 | 5 | 6 |
| 398 | 9 | 7 | 6 | 6 |
| 399 | 9 | 1 | 10 | 7 |
| 400 | 9 | 5 | 9 | 7 |
| 401 | 9 | 5 | 11 | 7 |
| 402 | 9 | 7 | 8 | 7 |
| 403 | 9 | 7 | 9 | 7 |

| | | | | |
|---|---|---|---|---|
| 404 | 9 | 5 | 6 | 8 |
| 405 | 9 | 7 | 6 | 8 |
| 406 | 9 | 1 | 5 | 9 |
| 407 | 9 | 1 | 10 | 9 |
| 408 | 9 | 5 | 6 | 9 |
| 409 | 9 | 7 | 5 | 9 |
| 410 | 9 | 7 | 6 | 9 |
| 411 | 9 | 1 | 10 | 10 |
| 412 | 9 | 5 | 6 | 10 |
| 413 | 9 | 7 | 6 | 10 |
| 414 | 9 | 1 | 5 | 11 |
| 415 | 9 | 5 | 6 | 11 |
| 416 | 9 | 5 | 11 | 11 |
| 417 | 9 | 7 | 5 | 11 |
| 418 | 9 | 7 | 6 | 11 |
| 419 | 9 | 7 | 8 | 11 |
| 420 | 10 | 4 | 2 | 1 |

| | | | | |
|---|---|---|---|---|
| 421 | 10 | 4 | 9 | 1 |
| 422 | 10 | 6 | 2 | 1 |
| 423 | 10 | 6 | 9 | 1 |
| 424 | 10 | 7 | 9 | 1 |
| 425 | 10 | 10 | 2 | 1 |
| 426 | 10 | 10 | 9 | 1 |
| 427 | 10 | 6 | 4 | 2 |
| 428 | 10 | 6 | 6 | 2 |
| 429 | 10 | 6 | 10 | 2 |
| 430 | 10 | 7 | 6 | 2 |
| 431 | 10 | 10 | 4 | 2 |
| 432 | 10 | 10 | 6 | 2 |
| 433 | 10 | 10 | 10 | 2 |
| 434 | 10 | 6 | 6 | 4 |
| 435 | 10 | 6 | 10 | 4 |
| 436 | 10 | 7 | 6 | 4 |
| 437 | 10 | 10 | 6 | 4 |
| 438 | 10 | 10 | 10 | 4 |

| | | | | |
|---|---|---|---|---|
| 439 | 10 | 2 | 1 | 5 |
| 440 | 10 | 2 | 8 | 5 |
| 441 | 10 | 4 | 8 | 5 |
| 442 | 10 | 4 | 9 | 5 |
| 443 | 10 | 6 | 8 | 5 |
| 444 | 10 | 6 | 9 | 5 |
| 445 | 10 | 7 | 8 | 5 |
| 446 | 10 | 7 | 9 | 5 |
| 447 | 10 | 9 | 1 | 5 |
| 448 | 10 | 9 | 7 | 5 |
| 449 | 10 | 10 | 7 | 5 |
| 450 | 10 | 10 | 9 | 5 |
| 451 | 10 | 6 | 6 | 6 |
| 452 | 10 | 6 | 10 | 6 |
| 453 | 10 | 7 | 5 | 6 |
| 454 | 10 | 7 | 6 | 6 |
| 455 | 10 | 9 | 5 | 6 |

| | | | | |
|---|---|---|---|---|
| 456 | 10 | 9 | 7 | 6 |
| 457 | 10 | 10 | 6 | 6 |
| 458 | 10 | 10 | 7 | 6 |
| 459 | 10 | 10 | 10 | 6 |
| 460 | 10 | 2 | 8 | 7 |
| 461 | 10 | 4 | 8 | 7 |
| 462 | 10 | 4 | 9 | 7 |
| 463 | 10 | 6 | 8 | 7 |
| 464 | 10 | 6 | 9 | 7 |
| 465 | 10 | 6 | 10 | 7 |
| 466 | 10 | 6 | 11 | 7 |
| 467 | 10 | 7 | 8 | 7 |
| 468 | 10 | 7 | 9 | 7 |
| 469 | 10 | 10 | 9 | 7 |
| 470 | 10 | 10 | 10 | 7 |
| 471 | 10 | 4 | 2 | 8 |
| 472 | 10 | 6 | 2 | 8 |
| 473 | 10 | 6 | 4 | 8 |

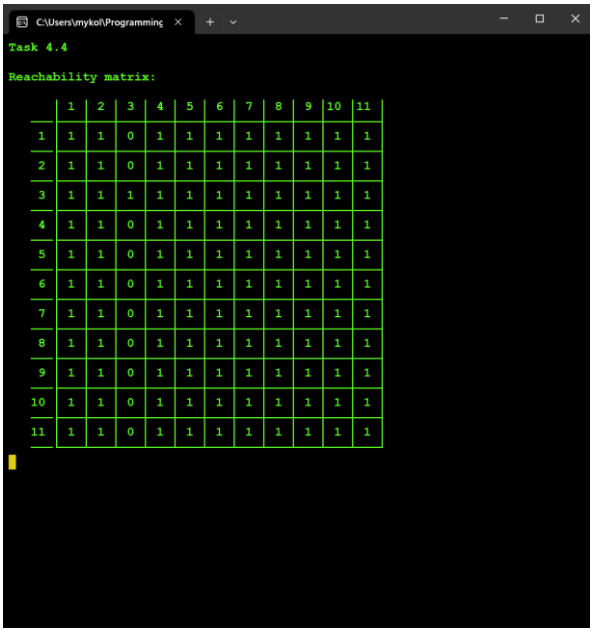
| | | | | |
|---|---|---|---|---|
| 474 | 10 | 6 | 6 | 8 |
| 475 | 10 | 7 | 6 | 8 |
| 476 | 10 | 9 | 7 | 8 |
| 477 | 10 | 10 | 2 | 8 |
| 478 | 10 | 10 | 4 | 8 |
| 479 | 10 | 10 | 6 | 8 |
| 480 | 10 | 10 | 7 | 8 |
| 481 | 10 | 6 | 4 | 9 |
| 482 | 10 | 6 | 6 | 9 |
| 483 | 10 | 6 | 10 | 9 |
| 484 | 10 | 7 | 5 | 9 |
| 485 | 10 | 7 | 6 | 9 |
| 486 | 10 | 9 | 5 | 9 |
| 487 | 10 | 9 | 7 | 9 |
| 488 | 10 | 10 | 4 | 9 |
| 489 | 10 | 10 | 6 | 9 |
| 490 | 10 | 10 | 7 | 9 |

| | | | | |
|---|---|---|---|---|
| 491 | 10 | 10 | 10 | 9 |
| 492 | 10 | 2 | 1 | 10 |
| 493 | 10 | 6 | 6 | 10 |
| 494 | 10 | 6 | 10 | 10 |
| 495 | 10 | 7 | 6 | 10 |
| 496 | 10 | 9 | 1 | 10 |
| 497 | 10 | 10 | 6 | 10 |
| 498 | 10 | 10 | 10 | 10 |
| 499 | 10 | 2 | 8 | 11 |
| 500 | 10 | 4 | 8 | 11 |
| 501 | 10 | 6 | 6 | 11 |
| 502 | 10 | 6 | 8 | 11 |
| 503 | 10 | 6 | 11 | 11 |
| 504 | 10 | 7 | 5 | 11 |
| 505 | 10 | 7 | 6 | 11 |
| 506 | 10 | 7 | 8 | 11 |
| 507 | 10 | 9 | 5 | 11 |

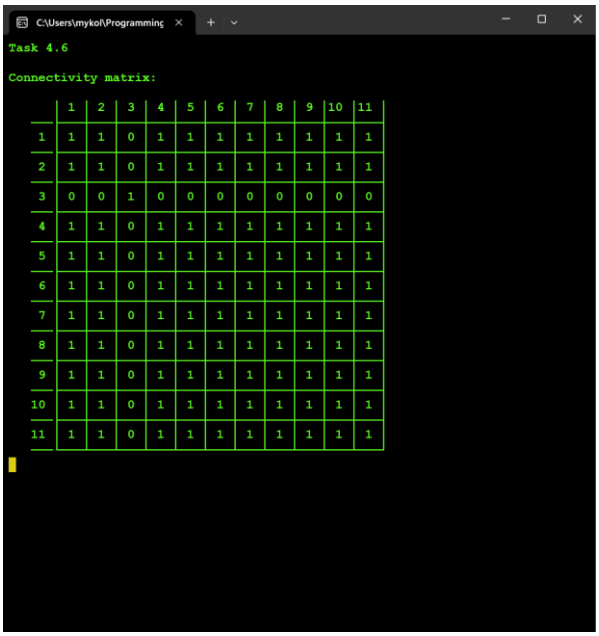
| | | | | |
|---|---|---|---|---|
| 508 | 10 | 10 | 6 | 11 |
| 509 | 11 | 7 | 9 | 1 |
| 510 | 11 | 7 | 6 | 2 |
| 511 | 11 | 7 | 6 | 4 |
| 512 | 11 | 7 | 8 | 5 |
| 513 | 11 | 7 | 9 | 5 |
| 514 | 11 | 11 | 7 | 5 |
| 515 | 11 | 7 | 5 | 6 |
| 516 | 11 | 7 | 6 | 6 |
| 517 | 11 | 11 | 7 | 6 |
| 518 | 11 | 7 | 8 | 7 |
| 519 | 11 | 7 | 9 | 7 |
| 520 | 11 | 11 | 11 | 7 |
| 521 | 11 | 7 | 6 | 8 |
| 522 | 11 | 11 | 7 | 8 |
| 523 | 11 | 7 | 5 | 9 |
| 524 | 11 | 7 | 6 | 9 |
| 525 | 11 | 11 | 7 | 9 |

| | | | | |
|---|---|---|---|---|
| 526 | 11 | 7 | 6 | 10 |
| 527 | 11 | 7 | 5 | 11 |
| 528 | 11 | 7 | 6 | 11 |
| 529 | 11 | 7 | 8 | 11 |
| 530 | 11 | 11 | 11 | 11 |

# *Матриці досяжності та зв'язності модифікованого графа*

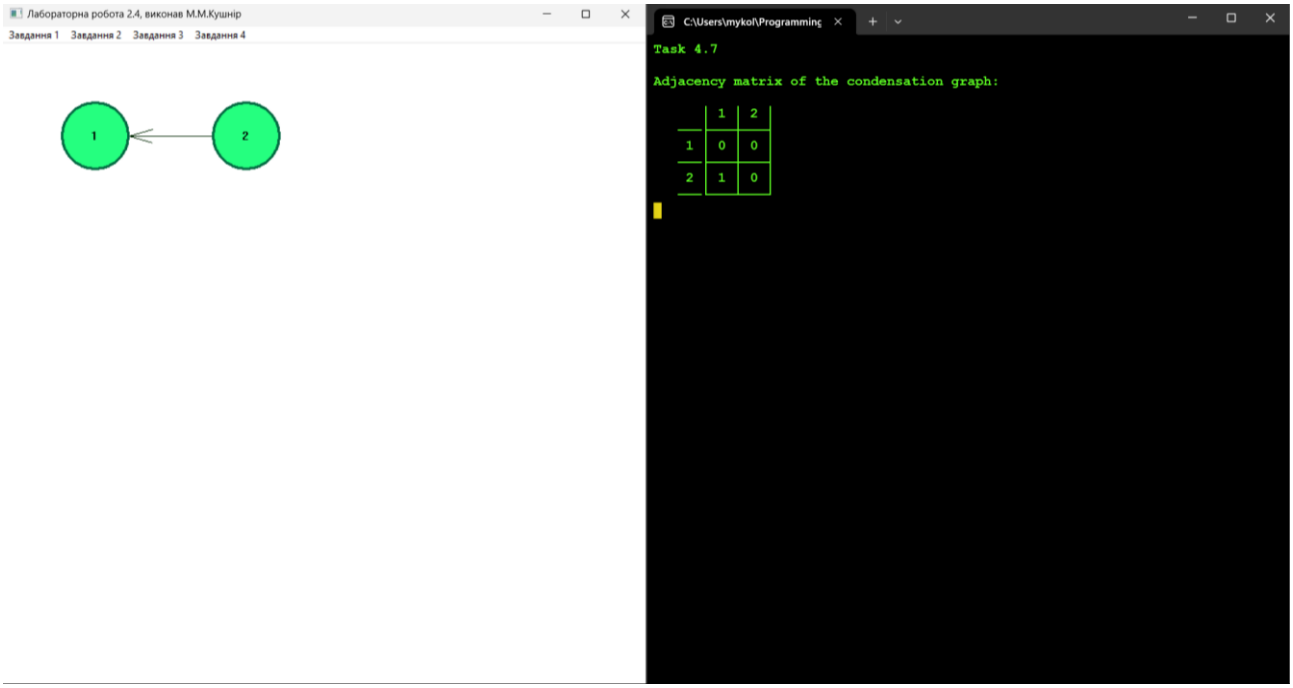



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 8 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 8 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### *Компоненти сильної зв'язності, граф конденсації та його матриця суміжності*


### *Висновки*

Графи – це дуже корисна структура даних, яка має багато прикладних застосувань у сучасному програмуванні. У ході виконання цієї лабораторної роботи, я досліджував графи, задані матрицями суміжності. Я мав змогу попрактикуватися у знаходженні степенів ненапрямленого та напівстепенів напрямленого графа, утворенні матриць досяжності і зв'язності, побудові графа конденсації, утвореного з компонент сильної зв'язності. Найцікавішим завданням для мене було написання алгоритму для пошуку проміжних вузлів, на шляху від вершини А до вершини Б. Також виведення інформації, закодованої матрицею, у графічне вікно, дозволило наочно переконатися у істинності обчислень та отриманого на лекціях теоретичного матеріалу. Впевнений, що ці навички стануть у нагоді при роботі з графовими структурами даних.