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 № 2.3

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 у програмі напрямлений і ненапрямлений графи із заданими параметрами:

- число вершин ***n***;
- розміщення вершин;
- матриця суміжності ***A***.

Параметри задаються на основі номера групи, представленого десятковими цифрами n_1, n_2 та номера студента у списку групи – десяткового числа n_3, n_4.

Число вершин n дорівнює $10 + n_3$.

Розміщення вершин:

- колом при $n_4 = 0, 1$;
- прямокутником (квадратом) при $n_4 = 2, 3$;
- трикутником при $n_4 = 4, 5$;
- колом з вершиною в центрі при $n_4 = 6, 7$;
- прямокутником (квадратом) з вершиною в центрі при $n_4 = 8, 9$.

Наприклад, при $n_4 = 10$ розміщення вершин прямокутником з вершиною в центрі повинно виглядати так, як на прикладі графа *рис.4*.

Матриця ***A*** напрямленого графа за варіантом формується за функціями:

```
srand(n1 n2 n3 n4);
T = randm(n, n);
A = mulmr((1.0 – n3 * 0.02 – n4 * 0.005 - 0.25), T);
```

де `randm(n, n)` – розроблена функція, яка формує матрицю розміром $n \times n$, що складається з випадкових чисел у діапазоні (0, 2.0);

`mulmr()` – розроблена функція множення матриці на коефіцієнт та округлення результату до 0 чи 1 (0, якщо результат *менший за 1.0* і 1 – якщо *більший за 1.0*).

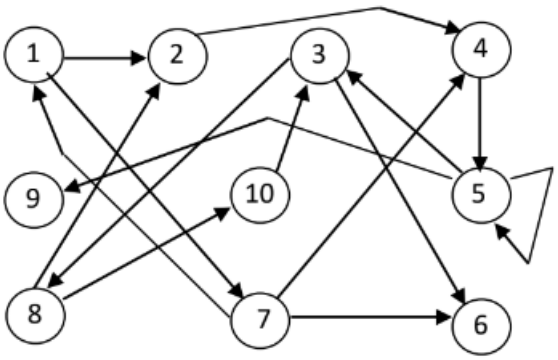


Рис.4 — Приклад зображення графа

2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

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 * 0.02 - 3.0 * 0.005 - 0.25), T);
```

Посилання на репозиторій з лабораторною роботою

Текст програми

Вміст файлу Configurations.h

```
/* 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, які часто 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) /* Відстань між двома
вершинами графа */
#define WINDOW_BORDER_OFFSET    (2 * LOOP_RADIUS + 10)

const wchar_t *vertices_names[11] =
{
    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 graph_full_size[2] =
{
    (2 * WINDOW_BORDER_OFFSET + GRAPH_WIDTH),
    (2 * WINDOW_BORDER_OFFSET + GRAPH_HEIGHT + 40),
};

#define PRINT_DISTANCE 20
const int vertex_print_offset[2] = {5, 8};

/* Визначають зміщення центру ребер для огинання вершин або вже намальованих
ребер */
const double edge_center_offset_dividers[5] = {1, 4.1, 4.8, 7, 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;

/* 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 **vertices_coords = CreateInt2dArr(n, n);
    int min_coords[2] =
    {
        WINDOW_BORDER_OFFSET,
        WINDOW_BORDER_OFFSET
    };
    int max_coords[2] =
    {
        min_coords[x] + GRAPH_WIDTH,
        min_coords[y] + GRAPH_HEIGHT
    };
    int current_pos[2] =
    {
        min_coords[x],
        min_coords[y]
    };
    for (int i = 0; i < n; i++)
    {
        vertices_coords[i][x] = current_pos[x];
        vertices_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);
    }
    return vertices_coords;
}

draw_data SetEdgeDrawData(int v1, int v2, int is_directed, int **coords, int
drawn_lines[N][N])
{
    const int max_one_step_len = coords[N - 1][y] - coords[0][y];
    int min_coords[2] =
    {
        WINDOW_BORDER_OFFSET,
        WINDOW_BORDER_OFFSET
    };
    int max_coords[2] =
    {
        min_coords[x] + GRAPH_WIDTH,
        min_coords[y] + GRAPH_HEIGHT
    };
    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];
    double distance = GetDistance(data.start, data.end);
    int center_offset[2] =
    {
        abs((int)(dy / edge_center_offset_dividers[(int)(distance /
ONE_STEP_LENGTH)])),
        abs((int)(dx / edge_center_offset_dividers[(int)(distance /
ONE_STEP_LENGTH)]))
    };
    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;
    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 (dy == 0 || dx == 0)
    {
        if (abs(dx) > max_one_step_len || abs(dy) > max_one_step_len)
        {
            if (!is_drawn)
            {
                data.center[x] += center_offset[x];
                data.center[y] += center_offset[y];
            }
            else
            {
                data.center[x] -= 2 * center_offset[x];
                data.center[y] -= 2 * center_offset[y];
            }
        }
    }
    else if (dx == dy && is_drawn)
        data.center[x] -= center_offset[x];
    else
    {
        if (is_drawn)
        {
            int graph_center[2] =
            {
                ((max_coords[x] + min_coords[x]) / 2),
                ((max_coords[y] + min_coords[y]) / 2)
            };
            int alternative_center[2] =
            {
                (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 DrawingFunctions.h

```
#include "DrawingDataSetter.h"

void ShowInt2dArr(int rows, int cols, int **matrix, const int
print_start_pos[2], HDC hdc)
{
    int current_output_pos[2] =
    {
        print_start_pos[x],
        print_start_pos[y]
    };
    wchar_t *output;
    int i, j;
```

```
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
        {
            if (matrix[i][j] == 1)
                output = L"1";
            else
                output = L"0";
            TextOut(hdc,
                    current_output_pos[x],
                    current_output_pos[y],
                    output,
                    1);
            current_output_pos[x] += PRINT_DISTANCE;
        }
        current_output_pos[x] = print_start_pos[x];
        current_output_pos[y] += PRINT_DISTANCE;
    }
}

void DrawGraph(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]);
            /* 9-й та 10-й 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 WorkWithMatrices.h

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

/* Допоміжні функції */
double RandInRange(double min, double max);
int **CreateInt2dArr(int rows, int cols);
void PrintDouble2dArr(int rows, int cols, double **arr);
void FreeInt2dArr(int rows, int **arr);
void FreeDouble2dArr(int rows, double **arr);

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

```
}

int **mulmr(int rows, int cols, double **matrix_T, double coefficient)
{
    int **matrix_A = CreateInt2dArr(rows, cols);
    int i, j;
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < cols; j++)
            matrix_A[i][j] = (int) (matrix_T[i][j] * coefficient);
    }
    return matrix_A;
}

int **SymmetrizeMatrix(int n, int **matrix_A)
{
    int **symmetric_matrix = CreateInt2dArr(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)
                symmetric_matrix[i][j] = symmetric_matrix[j][i] = 1;
    }
    return symmetric_matrix;
}

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

int **CreateInt2dArr(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 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");
    }
    printf("\n");
}

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

void FreeInt2dArr(int rows, int **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 "DrawingFunctions.h"

#define DRAW_OTHER_GRAPH 0

int **A;
int **undirected_A;
int **current_matrix;
int **vertices_coords;

HWND hButton;
HWND hLabel1;
HWND hLabel2;

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

void CreateButton(HWND);
void CreateTips(HWND);

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
    WNDCLASS wndClass;
    wndClass.lpszClassName = L"Лабораторна робота 2.3";
    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 2.3",
                        L"Лабораторна робота 2.3, виконав М.М.Кушнір",
                        WS_OVERLAPPEDWINDOW,
                        10, 10, (graph_full_size[x] + 340), graph_full_size[y],
                        (HWND)NULL,
                        (HMENU)NULL,
                        (HINSTANCE)hInstance,
                        (HINSTANCE)NULL);
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    int GetMessageRes;
    while ((GetMessageRes = GetMessage(&lpMsg, hWnd, 0, 0)) != 0)
    {
        if (GetMessageRes == -1)
```

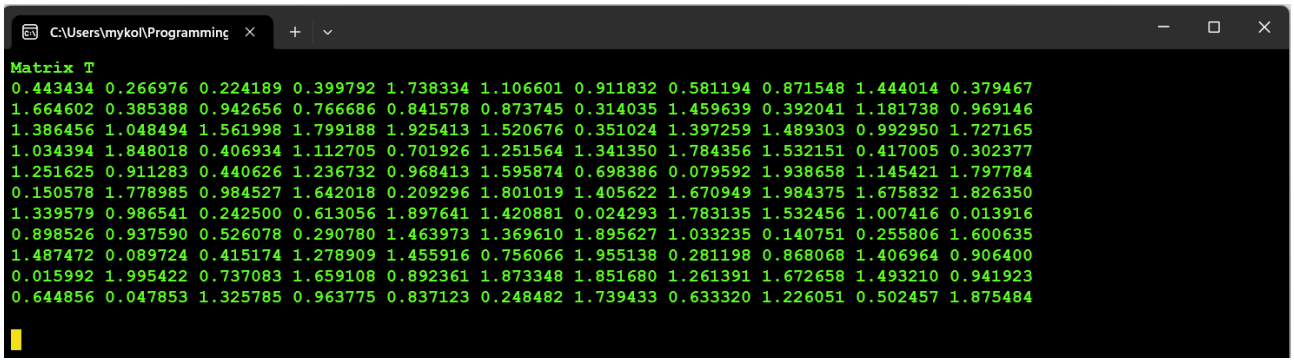
```
            return lpMsg.wParam;
        else
        {
            TranslateMessage(&lpMsg);
            DispatchMessage(&lpMsg);
        }
    }
}

LRESULT CALLBACK WndProc(HWND hWnd,
                         UINT message,
                         WPARAM wParam,
                         LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    switch (message)
    {
    case WM_COMMAND:
    {
        if (wParam == DRAW_OTHER_GRAPH)
        {
            if (current_matrix == undirected_A)
                current_matrix = A;
            else if (current_matrix == A)
                current_matrix = undirected_A;
            RedrawWindow(hWnd, NULL, NULL,
                         RDW_ERASE | RDW_INVALIDATE | RDW_UPDATENOW);
            DestroyWindow(hButton); // Видалення старих та створення
            DestroyWindow(hLabel1); // нових контролерів зі
            DestroyWindow(hLabel2); // зміненими відповідно до
            CreateButton(hWnd);            // поточного стану програми
            CreateTips(hWnd);              // текстовими назвами
            break;
        }
    }
    case WM_PAINT:
    {
        hdc = BeginPaint(hWnd, &ps);
        SetBkMode(hdc, TRANSPARENT);
        HPEN ePen = CreatePen(PS_SOLID, 1, RGB(0, 38, 0));
        HBRUSH vFillBrush = CreateSolidBrush(RGB(37, 255, 127));
        HPEN vOutlinePen = CreatePen(PS_SOLID, 3, RGB(3, 104, 65));
        DrawGraph(current_matrix,
                  vertices_coords,
                  ePen,
                  vFillBrush,
                  vOutlinePen,
                  hdc);
        int matrix_print_pos[2] = {(graph_full_size[x] + 55), 250};
        ShowInt2dArr(N, N, current_matrix, matrix_print_pos, hdc);
        DeleteObject(ePen);
        DeleteObject(vOutlinePen);
        DeleteObject(vFillBrush);
        EndPaint(hWnd, &ps);
        break;
    }
    case WM_CREATE:
    {
        vertices_coords = SetVerticesCoords(N);
        srand(N1 * 1000 + 100 * N2 + 10 * N3 + N4);
        double **T = randm(N, N);
        A = mulmr(N, N, T,
                  (1.0 - N3 * 0.02 - N4 * 0.005 - 0.25));
```

```
        undirected_A = SymmetrizeMatrix(N, A);
        /* Логування матриці T до консолі для перевірки коректності роботи
функції randm() */
        printf("Matrix T\n");
        PrintDouble2dArr(N, N, T);
        /* На початку роботи програми завжди буде виводитися ненапрямлений граф
*/
        current_matrix = undirected_A;
        FreeDouble2dArr(N, T);
        CreateButton(hWnd);
        CreateTips(hWnd);
        break;
    }
    case WM_DESTROY:
    {
        FreeInt2dArr(N, A);
        FreeInt2dArr(N, undirected_A);
        FreeInt2dArr(N, vertices_coords);
        PostQuitMessage(0);
        break;
    }
    default :
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
}

void CreateButton(HWND hWnd)
{
    wchar_t button_directed_graph[] = L"Намалювати напрямлений граф";
    wchar_t button_undirected_graph[] = L"Намалювати ненапрямлений граф";
    wchar_t *buttonText;
    if (current_matrix == A)
        buttonText = button_undirected_graph;
    else
        buttonText = button_directed_graph;
    hButton = CreateWindowW(L"Button", buttonText,
                            WS_VISIBLE | WS_CHILD | WS_BORDER |
                            BS_FLAT | BS_MULTILINE | BS_CENTER,
                            (graph_full_size[x] + 55), 550, 210, 50,
                            hWnd,
                            (HMENU)DRAW_OTHER_GRAPH,
                            NULL, NULL);
}

void CreateTips(HWND hWnd)
{
    wchar_t label1_directed_graph[] = L"Матриця A";
    wchar_t label1_undirected_graph[] = L"Матриця undirected_A";
    wchar_t *label1_text;
    wchar_t label2_directed_graph[] = L"( напрямлений граф )";
    wchar_t label2_undirected_graph[] = L"( ненапрямлений граф )";
    wchar_t *label2_text;
    if (current_matrix == A)
    {
        label1_text = label1_directed_graph;
        label2_text = label2_directed_graph;
    }
    else
    {
        label1_text = label1_undirected_graph;
        label2_text = label2_undirected_graph;
    }
    hLabel1 = CreateWindowW(L"Static", label1_text,
                            WS_VISIBLE | WS_CHILD | WS_BORDER | SS_CENTER,
```

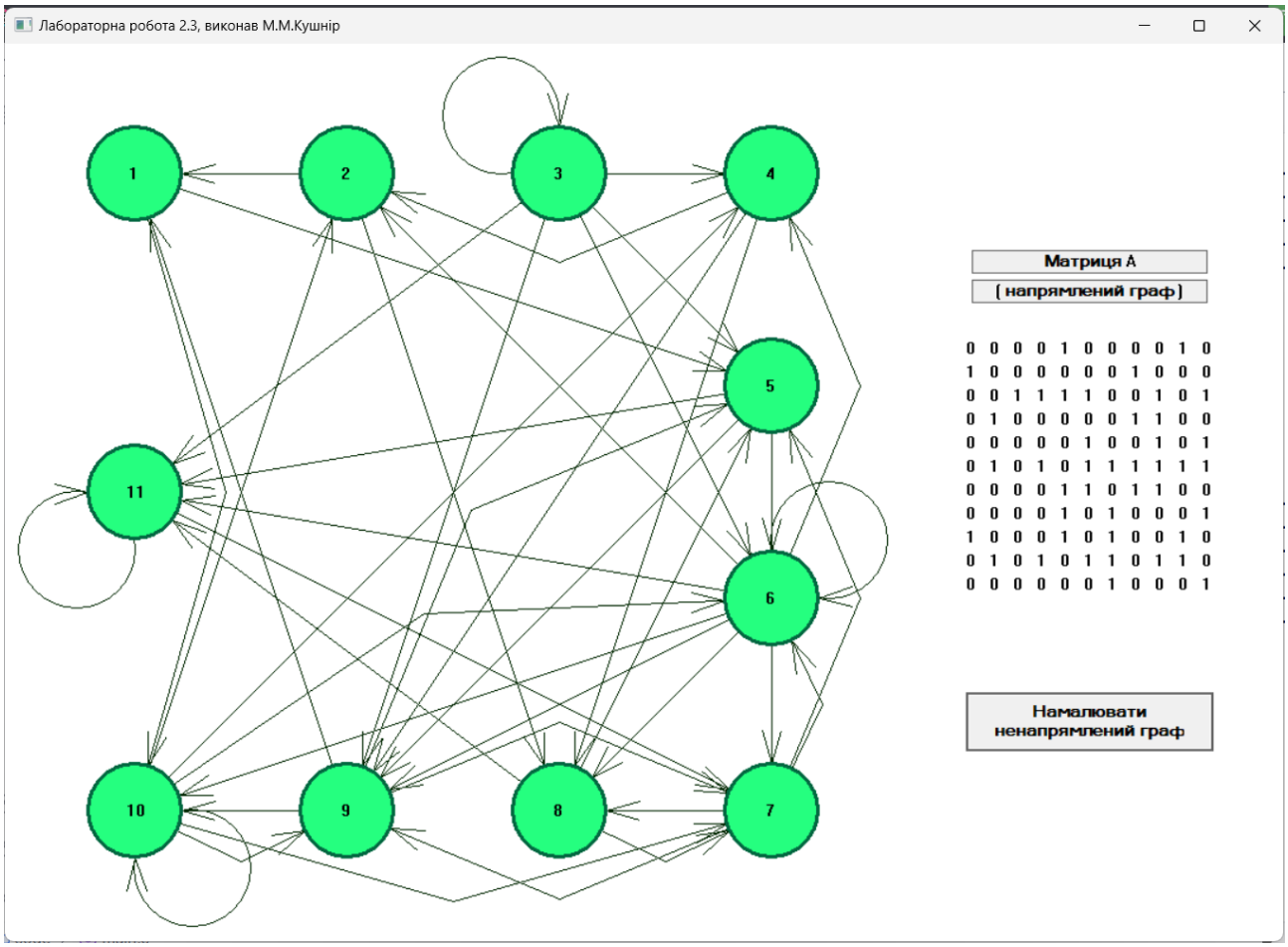
```
                                (graph_full_size[x] + 60), 175, 200, 20,
                                hWnd,
                                NULL, NULL, NULL);
    hLabel2 = CreateWindowW(L"Static", label2_text,
                                WS_VISIBLE | WS_CHILD | WS_BORDER | SS_CENTER,
                                (graph_full_size[x] + 60), 200, 200, 20,
                                hWnd,
                                NULL, NULL, NULL);
}
```

Результати тестування програми

Матриця T



Матриця A (матриця суміжності напрямленого графа) та напрямлений граф



Матриця undirected_A (матриця суміжності ненапрямленого графа) та ненапрямлений граф