**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: Перевірила:

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

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 *n1*, *n2* та номера студента у списку групи – десяткового

числа *n3*, *n4*.

Число вершин *n* дорівнює 10+ *n3*.

Розміщення вершин:

* колом при *n4* = 0, 1;
* прямокутником (квадратом) при *n4* = 2, 3;
* трикутником при *n4* = 4, 5;
* колом з вершиною в центрі при *n4* = 6, 7;
* прямокутником (квадратом) з вершиною в центрі при *n4* = 8, 9.

Наприклад, при *n4* = 10 розміщення вершин прямокутником з вершиною

в центрі повинно виглядати так, як на прикладі графа *рис.4*.

Матриця ***А*** 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);

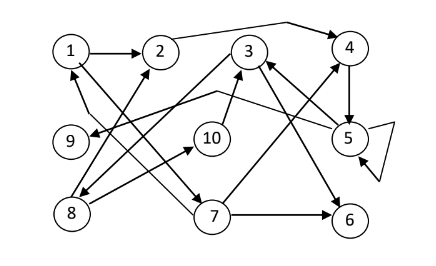
де randm(n, n) – розроблена функція, яка формує матрицю розміром *n×n*,

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)***

* *n1* = **2**;
* *n2* = **2**;
* *n3* = **1**;
* *n4* = **3**;

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

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

Формування матриці ***А***:

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***](https://github.com/kushnirko/asd-labs/tree/main/semester-2/lab-2.3)

***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

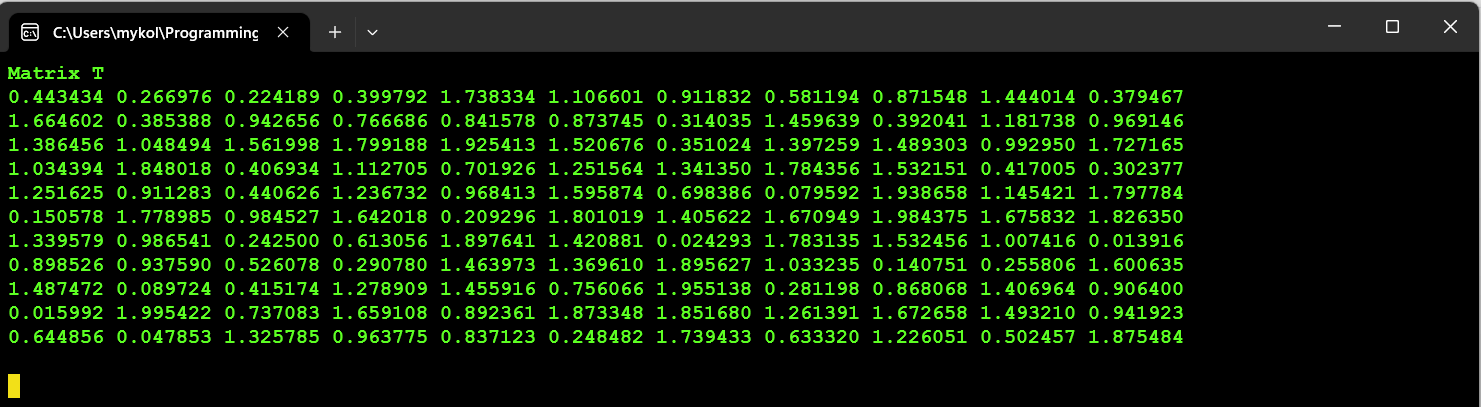
{  
 TranslateMessage(&lpMsg);  
 **DispatchMessage**(&lpMsg);  
 }  
 }  
}  
  
LRESULT **CALLBACK** WndProc(HWND hWnd,  
 UINT message,  
 WPARAM wParam,  
 LPARAM lParam)  
{  
 HDC hdc;  
 PAINTSTRUCT ps;  
 switch (message)  
 {  
 case **WM\_COMMAND**:  
 {  
 if (wParam == **DRAW\_OTHER\_GRAPH**)  
 {  
 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 Т 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);  
}

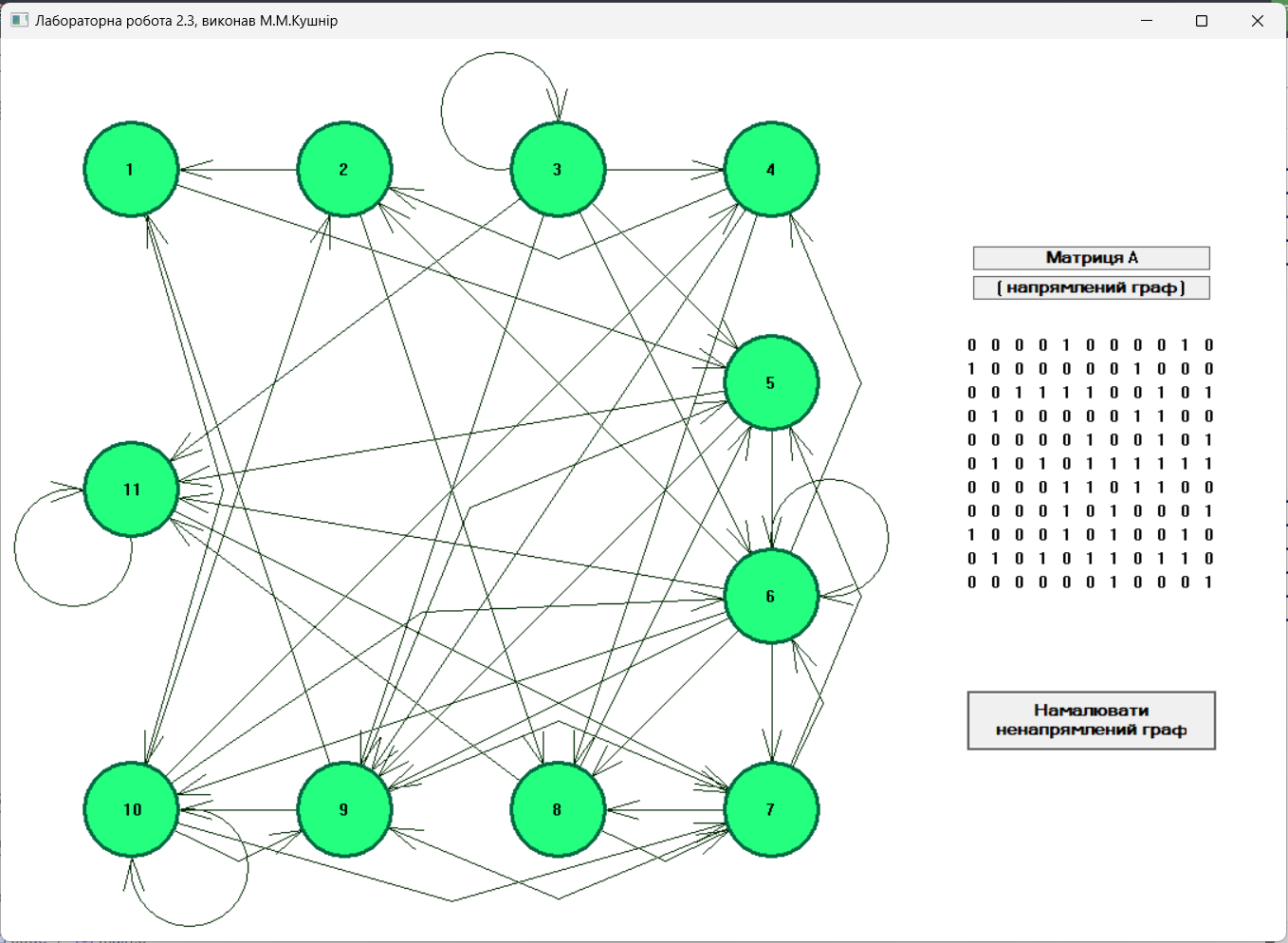
***Результати тестування програми***

***Матриця Т***

******

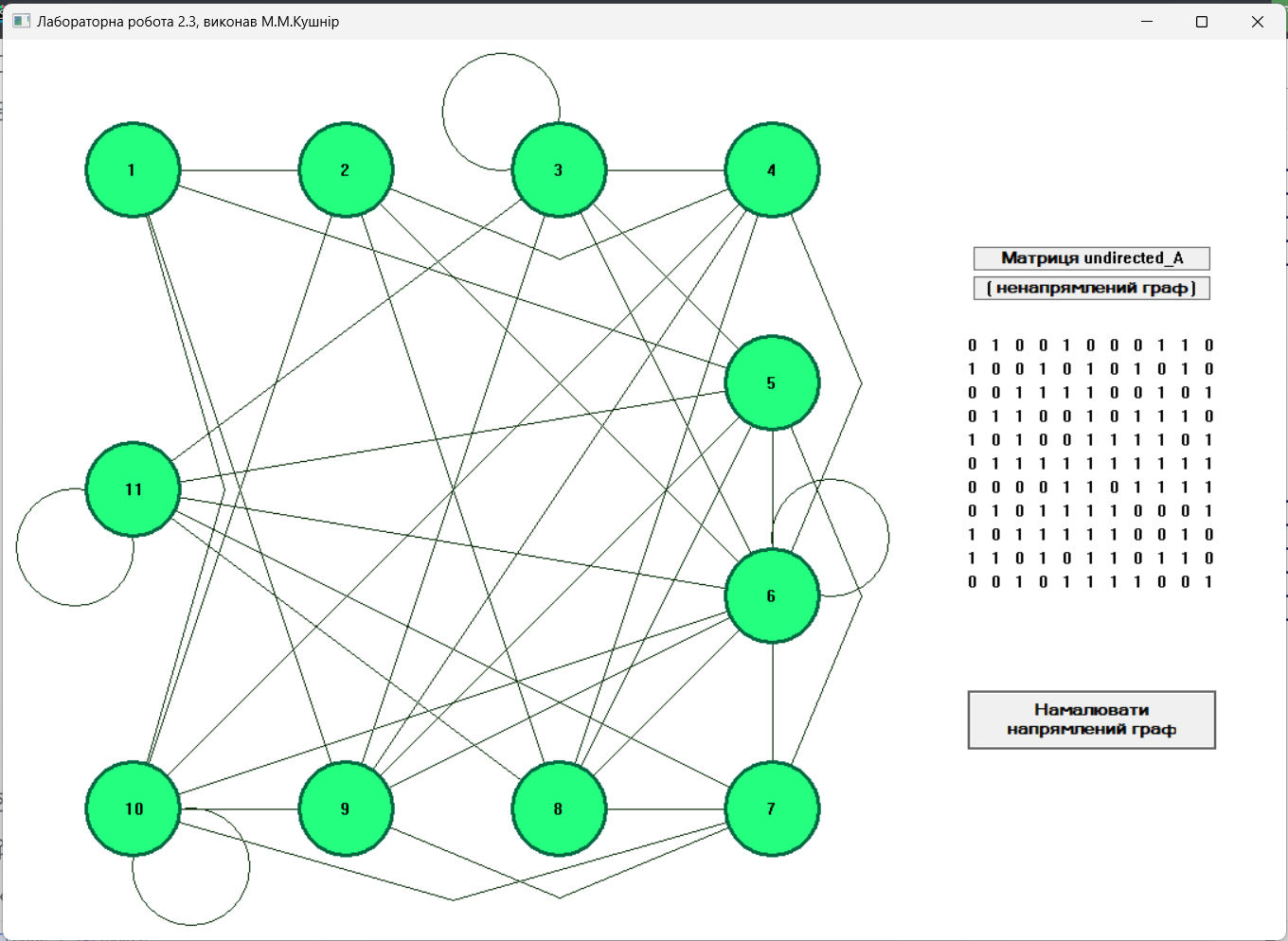
***Матриця А (матриця суміжності напрямленого графа)***

***та напрямлений граф***

******

***Матриця undirected\_А (матриця суміжності ненапрямленого графа)***

***та ненапрямлений граф***

******