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 ІП-03
Пашковський Євгеній Сергійович
номер у списку групи: 17

Перевірила:

Сергієнко А. А.

Київ 2021

Завдання

1. Представити напрямлений граф з заданими параметрами так само, як у лабораторній роботі 3.

Відміна: матриця А напрямленого графа за варіантом формується за функціями:

srand(п1 п2 п3 п4);

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 A2 і A3 відповідно. Матриця досяжності та компоненти сильної зв’язності слід шукати за допомогою операції транзитивного замикання.

Варіант 17:

n1 = 3; n2 = 3; n3 = 1; n4 = 7.

Текст програми

```
#include <windows.h>
#include <math.h>
#include <stdio.h>

#define n1 0
#define n2 3
#define n3 1
#define n4 7

#define WIN_WIDTH 1000
#define WIN_HEIGHT 1000
#define GRAPH_MODE 1 // 1 = oriented graph, 0 = non-oriented graph

#define n n3 + 10
// #define n 5
// #define RADIUS 200

double matrix[n][n];

void randm()
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			matrix[i][j] = ((double)rand() / (double)RAND_MAX) * 2.0;
		}
	}
}

void mulmr(double a)
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			matrix[i][j] = floor(matrix[i][j] * a);
		}
	}
}

void modifyMatrix()
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			if (matrix[i][j] == 1)
			{
				matrix[j][i] = 1;
			}
		}
	}
}

COORD GetConsoleCursorPosition(HANDLE hConsoleOutput) {
	CONSOLE_SCREEN_BUFFER_INFO csbi;
	if (GetConsoleScreenBufferInfo(hConsoleOutput, &csbi)) {
		return csbi.dwCursorPosition;
	} else {
		COORD invalid = { 0, 0 };
		return invalid;
	}
}

void gotoX(int x){
	x += GetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE)).X;
```

```
	COORD pos = {x, GetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE)).Y};
	SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}

void gotoY(int y){
	y += GetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE)).Y;
	COORD pos = {GetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE)).X, y};
	SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}

void drawMatrix(int l, double matrix[l][l]) {
	printf("\n\n");
	for (int i = 0; i < l; i++) {
		for (int j = 0; j < l; j++){
			gotoX(2);
			printf("%d", (int)matrix[i][j]);
		}
		printf("\n\n");
	}
	printf("\n");
}

double angleBetween(double Vector1[2], double Vector2[2])
{
	double x1 = Vector1[0], x2 = Vector2[0], y1 = Vector1[1], y2 = Vector2[1];
	return acos((x1 * x2 + y1 * y2) / (sqrt(x1 * x1 + y1 * y1) * sqrt(x2 * x2 + y2 * y2)));
}

int getPowerOfVertex(double m[n][n], int id, int mode /*0 = power, 1 = half power for entrence 2 = half power for exit*/)
{
	int result = 0;

	for (int i = 0; i < n; i++)
	{
		if ((mode == 0 || mode == 2) && m[id][i] == 1)
		{
			if (mode == 0 && id == i)
			{
				result++;
			}
			result++;
		}

		if (((mode == 0 && GRAPH_MODE) || mode == 1) && m[i][id] == 1)
		{
			result++;
		}
	}

	return result;
}

double addMResult[n][n];
void addMatrixes(double m1[n][n], double m2[n][n])
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			addMResult[i][j] = m1[i][j] + m2[i][j];
		}
	}
}


double transponeResult[n][n];
void transponeMatrix(double m[n][n])
{
	for (int i = 0; i < n; i++)
	{
```

```
                for (int j = 0; j < n; j++)
                {
                        transponeResult[i][j] = m[j][i];
                }
        }
}

double mpResult[n][n];
int debug = 0;
void mpMatrixes(double m1[n][n], double m2[n][n])
{
        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {
                        mpResult[i][j] = 0;
                        for (int k = 0; k < n; k++)
                        {
                                mpResult[i][j] += m1[k][j] * m2[i][k];
                        }
                }
        }
}

int isIn(double el, double arr[n])
{
        for (int i = 0; i < n; i++)
        {
                if (el == arr[i])
                {
                        return 1;
                }
        }
        return 0;
}

int isWay(int elFrom, int elTo, double R[n][n])
{
        return (elFrom < n && elTo < n && R[elFrom][elTo] == 1) ? 1 : 0;
}

void drawWays(int length, double A[n][n], double matrix[n][n])
{
        if (length == 2)
        {
                for (int i = 0; i < n; i++)
                {
                        for (int j = 0; j < n; j++)
                        {
                                if (matrix[i][j] == 1)
                                {
                                        for (int k = 0; k < n; k++)
                                        {
                                                if (matrix[j][k] == 1)
                                                {
                                                        printf("%d->%d->%d\n", i + 1, j + 1, k + 1);
                                                }
                                        }
                                }
                        }
                }
        }
        if (length == 3)
        {
                for (int i = 0; i < n; i++)
                {
                        for (int j = 0; j < n; j++)
                        {
                                if (matrix[i][j] == 1)
                                {
```

```
                                                for (int k = 0; k < n; k++)
                                                {
                                                        if (matrix[j][k] == 1)
                                                        {
                                                                for (int m = 0; m < n; m++)
                                                                {
                                                                        if (matrix[k][m] == 1)
                                                                        {
                                                                                printf("%d->%d->%d->%d\n", i + 1, j + 1, k + 1, m +
1);
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }
                        }
                }
}

double strongBlocks[n];
void getCondensationBlocks(double m[n][n])
{
        int k = 0;

        for (int i = 0; i < n; i++)
        {
                if (!isIn(m[i][i], strongBlocks))
                {
                        strongBlocks[k] = m[i][i];
                        k++;
                }
        }
        if (k < n)
        {
                for (int k; k < n; k++)
                {
                        strongBlocks[k] = 0;
                }
        }
}

void drawCharacteristics(double m[n][n])
{
        printf("-------------------------\n");

        printf("Matrix: \n");
        drawMatrix(n, m);

        printf("Half-power (in|out): \n");
        for (int i = 0; i < n; i++)
        {
                printf(" %d:          %d|%d\n", i + 1, getPowerOfVertex(m, i, 1), getPowerOfVertex(m, i, 2));
        }
        printf("\n");

        double A2[n][n], A3[n][n];
        mpMatrixes(m, m);
        memcpy(A2, mpResult, sizeof(mpResult));
        mpMatrixes(m, m);
        memcpy(A3, mpResult, sizeof(mpResult));
        printf("A^2:\n");
        drawMatrix(n, A2);
        drawWays(2, A2, m);
        printf("A^3:\n");
        drawMatrix(n, A3);
        drawWays(3, A3, m);

        double R[n][n];
        memcpy(mpResult, m, sizeof(mpResult));
```

```
	memcpy(addMResult, m, sizeof(addMResult));
	for (int i = 0; i < n - 1; i++)
	{
		double m1[n][n], m2[n][n];
		memcpy(m1, mpResult, sizeof(mpResult));
		memcpy(m2, mpResult, sizeof(mpResult));
		mpMatrixes(m1, m2);
		addMatrixes(addMResult, mpResult);
	}
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			if (i == j)
			{
				mpResult[i][j] = 1;
				addMResult[i][j] = 1;
			}

			if (mpResult[i][j] != 0)
			{
				mpResult[i][j] = 1;
			}

			if (addMResult[i][j] != 0)
			{
				addMResult[i][j] = 1;
			}
		}
	}
	memcpy(R, addMResult, sizeof(addMResult));
	printf("Reach matrix: \n");
	drawMatrix(n, R);

	mpMatrixes(R, R);
	double R2[n][n];
	memcpy(R2, mpResult, sizeof(mpResult));
	printf("R^2: \n");
	drawMatrix(n, R2);

	double strongConnectivityM[n][n];
	transponeMatrix(R);
	mpMatrixes(R, transponeResult);
	memcpy(strongConnectivityM, mpResult, sizeof(mpResult));
	// for (int i = 0; i < n; i++)
	// {
	//	for (int j = 0; j < n; j++)
	//	{
	//		if (strongConnectivityM[i][j] != 0)
	//		{
	//			strongConnectivityM[i][j] = 1;
	//		}
	//	}
	// }
	printf("Strong connectivity matrix: \n");
	drawMatrix(n, strongConnectivityM);

	getCondensationBlocks(strongConnectivityM);
	int l = 0;
	for (l; l < n; l++)
	{
		if (strongBlocks[l] != 0)
		{
			l++;
		} else {
			break;
		}
	}
	double condensationM[l][l];
	for (int i = 0; i < l; i++)
```

```
	{
		double thisBlock = strongBlocks[i];

		for (int j = 0; j < n; j++)
		{
			if (strongConnectivityM[j][j] != thisBlock)
			{
				for (int k = 0; k < n; k++)
				{
					if (strongConnectivityM[k][k] != strongConnectivityM[j][j])
					{
						if (R[k][j] == 1)
						{
							condensationM[i][i + 1] = 1;
						}
						if (R[j][k] == 1)
						{
							condensationM[i + 1][i] = 1;
						}
					}
				}
			}
		}
	}

	printf("Condensation matrix: \n");
	drawMatrix(l, condensationM);

	printf("-------------------------\n");
}

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
char progName[] = "Лабораторна робота 3";

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
{
	WNDCLASS w;

	w.lpszClassName = progName;
	w.hInstance = hInstance;
	w.lpfnWndProc = WndProc;
	w.hCursor = LoadCursor(NULL, IDC_ARROW);
	w.hIcon = 0;
	w.lpszMenuName = 0;
	w.hbrBackground = WHITE_BRUSH;
	w.style = CS_HREDRAW|CS_VREDRAW;
	w.cbClsExtra = 0;
	w.cbWndExtra = 0;

	if (!RegisterClass(&w))
		return 0;

	HWND hWnd;
	MSG lpMsg;

	hWnd = CreateWindow(progName,
		"Лабораторна робота 3. Виконав Пашковський Є.С",
		WS_OVERLAPPEDWINDOW,
		100,
		100,
		WIN_WIDTH,
		WIN_HEIGHT,
		(HWND)NULL,
		(HMENU)NULL,
		(HINSTANCE)hInstance,
		(HINSTANCE)NULL);
	ShowWindow(hWnd, nCmdShow);

	while(GetMessage(&lpMsg, hWnd, 0, 0))
	{
```

```
		TranslateMessage(&lpMsg);
		DispatchMessage(&lpMsg);
	}

	return lpMsg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT messg, WPARAM wParam, LPARAM lParam)
{
	HDC hdc;
	PAINTSTRUCT ps;

	int xc = WIN_WIDTH / 2;
	int yc = WIN_HEIGHT / 2;
	int dx = 16, dy = 16, dtx = 5;
	char *nn[19] = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "19", "20"};
	double R = min(WIN_HEIGHT, WIN_WIDTH) / 3;
	double a = 2 * 3.1416 / (n - 1);

	void arrow(float fi, int px, int py)
	{
		fi = 3.1416 + fi;
		int lx,ly,rx,ry;
		lx = px + 15 * cos(fi + 0.3);
		rx = px + 15 * cos(fi - 0.3);
		ly = py - 15 * sin(fi + 0.3);
		ry = py - 15 * sin(fi - 0.3);
		MoveToEx(hdc, lx, ly, NULL);
		LineTo(hdc, px, py);
		LineTo(hdc, rx, ry);
	}

	void drawVertices(double matrix[n][n], HPEN pen1, HPEN pen2, int id)
	{

		int x = id == 0 ? xc : xc + R * cos(a * (id - 1));
		int y = id == 0 ? yc : yc - R * sin(a * (id - 1));

		SelectObject(hdc, pen1);

		Ellipse(hdc, x - dx, y - dy, x + dx, y + dy);
		TextOut(hdc, x - dtx, y - dy / 2, nn[id], 2);

		for (int j = 0; j < n; j++)
		{
			if (!GRAPH_MODE && id > j)
			{
				continue;
			}

			double b = id == j ? 0 : (abs(id - j) / (id - j)) * (3.1416 - a * abs(id - j)) / 2;

			if (matrix[id][j] == 1)
			{
				unsigned int flagArc = 0, flagDoubledConnection = 0;
				int xStart, yStart, xEnd, yEnd;
				double angle;

				if (GRAPH_MODE && id > j && matrix[j][id] == 1)
				{
					flagDoubledConnection = 1;
				}

				if (id == 0 && id != j)
				{
					xStart = xc + dx * cos(a * (j - 1));
					yStart = yc - dy * sin(a * (j - 1));

					xEnd = xc + (R - dx) * cos(a * (j - 1));
					yEnd = yc - (R - dy) * sin(a * (j - 1));
```

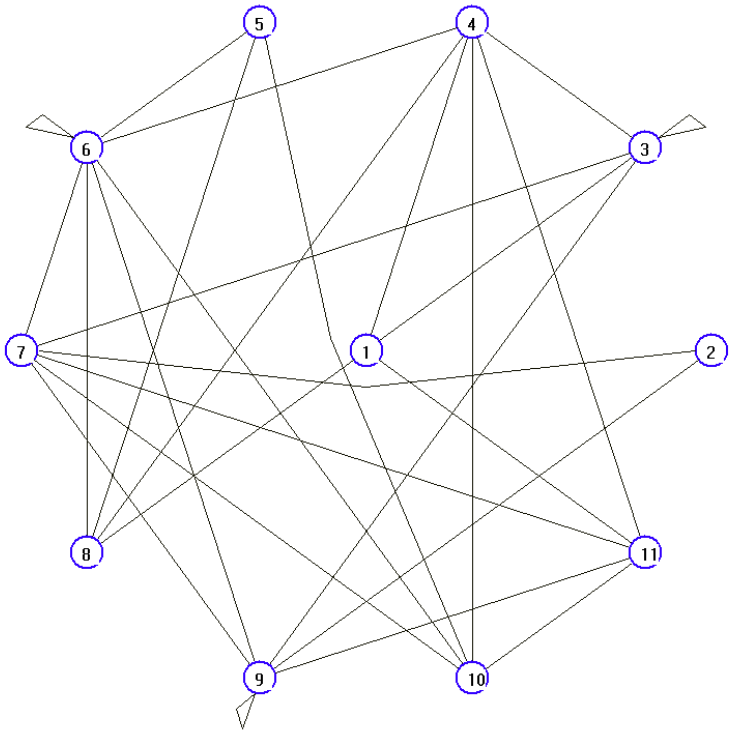
```
                                        angle = a * (j - 1);
                                }
                                else if (j == 0)
                                {
                                        xStart = xc + (R - dx) * cos(a * (id - 1));
                                        yStart = yc - (R - dy) * sin(a * (id - 1));

                                        xEnd = xc + dx * cos(a * (id - 1));
                                        yEnd = yc - dy * sin(a * (id - 1));

                                        angle = a * (id - 1) - 3.1416;
                                }
                                else if (id == j)
                                {
                                        flagArc = 1;
                                        xStart = xc + (R + dx) * cos(a * (id - 1));
                                        yStart = yc - (R + dy) * sin(a * (id - 1));

                                        xEnd = xc + (R + dx) * cos(a * (id - 1));
                                        yEnd = yc - (R + dy) * sin(a * (id - 1));
                                        angle = 3.1416 + a * (id - 1);
                                }
                                else {
                                        if (abs(id - j) == 5)
                                        {
                                                flagArc = 1;
                                        }

                                        xStart = xc + R * cos(a * (id - 1)) - dx * cos(a * (id - 1) + b);
                                        yStart = yc - R * sin(a * (id - 1)) + dy * sin(a * (id - 1) + b);

                                        xEnd = xc + R * cos(a * (j - 1)) - dx * cos(a * (j - 1) - b);
                                        yEnd = yc - R * sin(a * (j - 1)) + dy * sin(a * (j - 1) - b);

                                        angle = a * (j - 1) - b;
                                }

                                SelectObject(hdc, pen2);
                                MoveToEx(hdc, xStart, yStart, NULL);
                                if (!flagDoubledConnection && !flagArc)
                                {
                                        LineTo(hdc, xEnd, yEnd);
                                }
                                else if (flagDoubledConnection)
                                {
                                        if (j != 0)
                                        {
                                                double c = atan(30 / (R * cos(b))) * ((a * (id - 1) <= 3.1416 / 2 || (a * (id - 1) >= 3
* 3.1416 / 2) ? 1 : -1));

                                                angle = angle + c;
                                        }
                                        LineTo(hdc, (xStart + xEnd) / 2 + 30 * cos(a * (j == 0 ? id : id - 1)), (yStart + yEnd) / 2 -
30 * sin(a * (j == 0 ? id : id - 1)));

                                        LineTo(hdc, xEnd, yEnd);
                                }
                                else if (id == j){
                                        LineTo(hdc, xStart + (dx + 20) * cos(a * (id - 1)), yEnd - (dy + 20) * sin(a * (id - 1)));
                                        LineTo(hdc, xEnd + (dx + 40) * cos(a * (id - 1)) , yEnd - (dy) * sin(a * (id - 1)));
                                        LineTo(hdc, xEnd, yEnd);
                                } else {
                                        LineTo(hdc, xc + (dx + 20) * cos(a * (id - 1) + 3.1416 / 2), yc + (dy + 20) * sin(a * (id - 1)
+ 3.1416 / 2));

                                        LineTo(hdc, xEnd, yEnd);
                                }

                                if (GRAPH_MODE)
                                {
                                        arrow(angle, xEnd, yEnd);
                                }
```

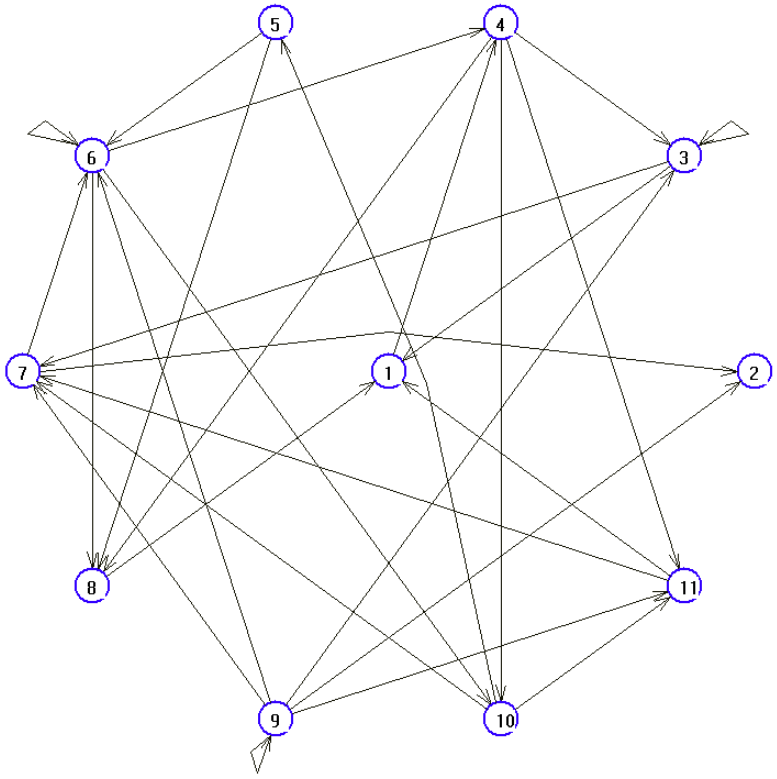
```
                }
        }
}

void drawGraph(double m[n][n], HPEN pen1, HPEN pen2)
{
        for (int i = 0; i < n; i++)
        {
                drawVertices(m, pen1, pen2, i);
        }
}

switch(messg){
        case WM_PAINT:
        system("cls");
        printf("-----\n");
        //1
        srand(n1 * 1000 + n2 * 100 + n3 * 10 + n4);
        randm();
        mulmr(1.0 - n3 * 0.01 - n4 * 0.01 - 0.3);

        double A[n][n]; // oriented graph's matrix
        for (int i = 0; i < n; i++)
        {
                memcpy(A[i], matrix[i], sizeof(matrix[i]));
        }

        if (!GRAPH_MODE)
        {
                modifyMatrix();
        }

        printf("Non-oriented graph's matrix: \n");
        drawMatrix(n, matrix);

        printf("Oriented graph's matrix:\n");
        drawMatrix(n, A);

        printf("-----\n");
        //2
        unsigned int isHomogeneous = 1;
        int defPower = -1;
        for (int i = 0; i < n; i++)
        {
                int power = getPowerOfVertex(matrix, i, 0);
                printf("Power of vertex %d: %d (non-oriented graph)\n", i + 1, power);

                if (defPower == -1)
                {
                        defPower = power;
                } else {
                        if (power != defPower)
                        {
                                isHomogeneous = 0;
                        }
                }
        }

        printf("This graph %s homogeneous\n", isHomogeneous ? "is" : "is NOT");
        if (isHomogeneous)
        {
                printf("Power: %d\n", defPower);
        }
        printf("-----\n");

        printf("\n");

        isHomogeneous = 1;
        defPower = -1;
        for (int i = 0; i < n; i++)
```

```
{
	int powerIn = getPowerOfVertex(A, i, 1);
	int powerOut = getPowerOfVertex(A, i, 2);
	int power = powerIn + powerOut;
	printf("Half-power(entrence) of vertex %d: %d (oriented graph)\n", i + 1, powerIn);
	printf("Half-power(exit) of vertex %d: %d (oriented graph)\n", i + 1, powerOut);
	printf("\n");

	if (defPower == -1)
	{
		defPower = power;
	} else {
		if (power != defPower)
		{
			isHomogeneous = 0;
		}
	}
}

printf("This graph %s homogeneous\n", isHomogeneous ? "is" : "is NOT");
if (isHomogeneous)
{
	printf("Power: %d\n", defPower);
}
printf("-----\n");
//3
printf("%Isolated vertices (non-oriented graph): \n");
for (int i = 0; i < n; i++)
{
	int power = getPowerOfVertex(matrix, i, 0);
	if (power == 0)
	{
		printf("%d ", i + 1);
	}
}
printf("\n");
printf("'Hanged' vertices (non-oriented graph): \n");
for (int i = 0; i < n; i++)
{
	int power = getPowerOfVertex(matrix, i, 0);
	if (power == 1)
	{
		printf("%d ", i + 1);
	}
}
printf("\n");

printf("%Isolated vertices (oriented graph): \n");
for (int i = 0; i < n; i++)
{
	int power = getPowerOfVertex(A, i, 1) + getPowerOfVertex(A, i, 2);
	if (power == 0)
	{
		printf("%d ", i + 1);
	}
}
printf("\n");
printf("'Hanged' vertices (oriented graph): \n");
for (int i = 0; i < n; i++)
{
	int power = getPowerOfVertex(A, i, 1) + getPowerOfVertex(A, i, 2);
	if (power == 1)
	{
		printf("%d ", i + 1);
	}
}
printf("\n");
printf("-----\n");
//...
```

```
                hdc = BeginPaint(hWnd, &ps);

                HPEN BPen = CreatePen(PS_SOLID, 2, RGB(50, 0, 255));
                HPEN KPen = CreatePen(PS_SOLID, 1, RGB(20, 20, 5));

                // drawGraph(matrix, BPen, KPen);


                //4
                randm();
                mulmr(1.0 - n3 * 0.005 - n4 * 0.005 - 0.27);
                drawGraph(matrix, BPen, KPen);
                drawCharacteristics(matrix);
                EndPaint(hWnd, &ps);
                break;

                case WM_DESTROY:
                PostQuitMessage(0);
                break;

                default:
                return DefWindowProc(hWnd, messg, wParam, lParam);
        }
        return 0;
}
                        int power = getPowerOfVertex(A, i, 1) + getPowerOfVertex(A, i, 2);
                        if (power == 0)
                        {
                                printf("%d ", i + 1);
                        }
                }
                printf("\n");
                printf("'Hanged' vertices (oriented graph): \n");
                for (int i = 0; i < n; i++)
                {
                        int power = getPowerOfVertex(A, i, 1) + getPowerOfVertex(A, i, 2);
                        if (power == 1)
                        {
                                printf("%d ", i + 1);
                        }
                }
                printf("\n");
                printf("-----\n");
                //...

                hdc = BeginPaint(hWnd, &ps);

                HPEN BPen = CreatePen(PS_SOLID, 2, RGB(50, 0, 255));
                HPEN KPen = CreatePen(PS_SOLID, 1, RGB(20, 20, 5));

                // drawGraph(matrix, BPen, KPen);


                //4
                randm();
                mulmr(1.0 - n3 * 0.005 - n4 * 0.005 - 0.27);
                drawGraph(matrix, BPen, KPen);
                drawCharacteristics(matrix);
                EndPaint(hWnd, &ps);
                break;

                case WM_DESTROY:
                PostQuitMessage(0);
                break;

                default:
                return DefWindowProc(hWnd, messg, wParam, lParam);
        }
        return 0;
}
```

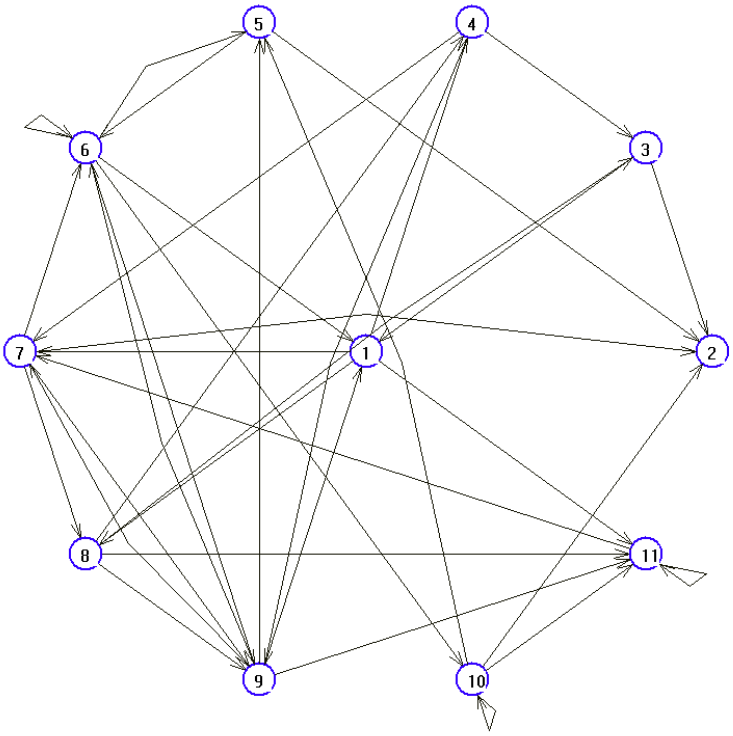
Тестування програми





```
Non-oriented graph's matrix:

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

Oriented graph's matrix:

 0  0  0  1  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0
 1  0  1  0  0  0  1  0  0  0  0
 0  0  1  0  0  0  0  1  0  1  1
 0  0  0  0  0  1  0  1  0  0  0
 0  0  0  1  0  1  0  1  0  1  0
 0  1  0  0  0  1  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0
 0  1  1  0  0  1  1  0  1  0  1
 0  0  0  0  1  0  1  0  0  0  1
```

```
Power of vertex 1: 4 (non-oriented graph)
Power of vertex 2: 2 (non-oriented graph)
Power of vertex 3: 7 (non-oriented graph)
Power of vertex 4: 6 (non-oriented graph)
Power of vertex 5: 3 (non-oriented graph)
Power of vertex 6: 9 (non-oriented graph)
Power of vertex 7: 6 (non-oriented graph)
Power of vertex 8: 4 (non-oriented graph)
Power of vertex 9: 8 (non-oriented graph)
Power of vertex 10: 5 (non-oriented graph)
Power of vertex 11: 5 (non-oriented graph)
This graph is NOT homogeneous
-----

Half-power(entrence) of vertex 1: 3 (oriented graph)
Half-power(exit) of vertex 1: 1 (oriented graph)

Half-power(entrence) of vertex 2: 2 (oriented graph)
Half-power(exit) of vertex 2: 0 (oriented graph)

Half-power(entrence) of vertex 3: 3 (oriented graph)
Half-power(exit) of vertex 3: 3 (oriented graph)

Half-power(entrence) of vertex 4: 2 (oriented graph)
Half-power(exit) of vertex 4: 4 (oriented graph)

Half-power(entrence) of vertex 5: 1 (oriented graph)
Half-power(exit) of vertex 5: 2 (oriented graph)

Half-power(entrence) of vertex 6: 4 (oriented graph)
Half-power(exit) of vertex 6: 4 (oriented graph)

Half-power(entrence) of vertex 7: 4 (oriented graph)
Half-power(exit) of vertex 7: 2 (oriented graph)

Half-power(entrence) of vertex 8: 3 (oriented graph)
Half-power(exit) of vertex 8: 1 (oriented graph)

Half-power(entrence) of vertex 9: 1 (oriented graph)
Half-power(exit) of vertex 9: 6 (oriented graph)

Half-power(entrence) of vertex 10: 2 (oriented graph)
Half-power(exit) of vertex 10: 3 (oriented graph)

Half-power(entrence) of vertex 11: 3 (oriented graph)
Half-power(exit) of vertex 11: 2 (oriented graph)

This graph is NOT homogeneous
-----
Isolated vertices (non-oriented graph):

'Hanged' vertices (non-oriented graph):

Isolated vertices (oriented graph):

'Hanged' vertices (oriented graph):

-----
```




```
Matrix:

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


Half-power (in|out):
 1:     3|4
 2:     4|0
 3:     2|2
 4:     2|3
 5:     3|2
 6:     4|5
 7:     4|4
 8:     2|4
 9:     4|5
 10:    2|4
 11:    5|2
```

A^2:

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

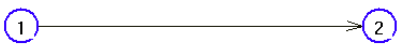
A^3:

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

```
Reach matrix:

  1  1  1  1  1  1  1  1  1  1  1
  0  1  0  0  0  0  0  0  0  0  0
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1
```

```
R^2:

  10   11   10   10   10   10   10   10   10   10   10
  0  1  0  0  0  0  0  0  0  0  0
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
  10   11   10   10   10   10   10   10   10   10   10
```

```
Strong connectivity matrix:

  10  10  10  10  10  10  10  10  10  10  10
  10  11  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10

Condensation matrix:

  0  1
  0  0

--------------------------
```



Таблиці

Степенів і напівстепенів:

1) Неорієнтований граф (вершина | степінь)

1	4
2	2
3	7
4	6
5	3
6	9
7	6
8	4
9	8
10	5
11	5

2) Орієнтований граф (вершина | півстепінь входу | півстепінь виходу)

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

3) Модифікований граф (вершина | півстепінь входу | півстепінь виходу)

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

Висячі та ізольовані вершини відсутні.

Висновки

Під час виконання лабораторної роботи було проведено повне дослідження графа, його властивостей та відношень. У результаті виконання програми були отримані такі дані:

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.