

**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 №3**  
з дисципліни  
«Алгоритми і структури даних»

Виконав:

студент групи ІМ-32  
Король Олександр Володимирович  
номер у списку групи: 14

Перевірила:

Молчанова А. А.

Київ 2024

## Постановка задачі

1. Представити у програмі напрямлений і ненаправлений графи з заданими параметрами:
  - кількість вершин  $n$ ;
  - розміщення вершин;
  - матриця суміжності  $A$ .
2. Створити програму для формування зображення напрямленого і ненаправленого графів у графічному вікні.

Згадані вище параметри графа задаються на основі чотиризначного номера варіанту  $n_1n_2n_3n_4$ , де  $n_1n_2$  це десяткові цифри номера групи, а  $n_3n_4$  — десяткові цифри номера варіанту, який був у студента для двох попередніх робіт (див. таблицю з поточними оцінками з АСД, надану викладачем на початку поточного семестру).

*Кількість вершин  $n$  дорівнює  $10 + n_3$ .*

*Розміщення вершин:*

- колом при  $n_4 = 0, 1$ ;
- квадратом (прямокутником) при  $n_4 = 2, 3$ ;
- трикутником при  $n_4 = 4, 5$ ;

- колом з вершиною в центрі при  $n_4 = 6, 7$ ;
- квадратом (прямокутником) з вершиною в центрі при  $n_4 = 8, 9$ .

Наприклад, при  $n_4 = 9$  розміщення вершин прямокутником з вершиною в центрі повинно виглядати так, як на прикладі графа на рис. 5.

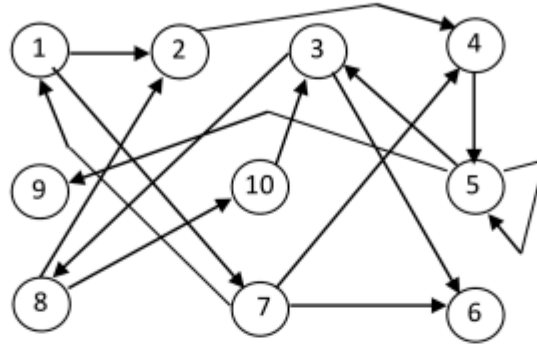


Рис. 5. Приклад зображення графа

**Матриця суміжності**  $A_{dir}$  напрямленого графа за варіантом формується таким чином:

- 1) встановлюється зерно генератора випадкових чисел, рівне номеру варіанту  $n_1 n_2 n_3 n_4$ ;
- 2) матриця розміром  $n \cdot n$  заповнюється згенерованими випадковими числами в діапазоні  $[0, 2.0)$ ;
- 3) обчислюється коефіцієнт  $k = 1.0 - n_3 * 0.02 - n_4 * 0.005 - 0.25$ ;
- 4) кожен елемент матриці множиться на коефіцієнт  $k$ ;
- 5) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

Матриця суміжності  $A_{undir}$  ненапрямленого графа одержується з матриці  $A_{dir}$ :

$$a_{dir_{i,j}} = 1 \Rightarrow a_{undir_{i,j}} = 1, a_{undir_{j,i}} = 1.$$

Наприклад, якщо запрограмувати додаткові функції `randm` та `mulmr`, у програмі на мові С генерація матриці  $A_{dir}$  напрямленого графа може виглядати так:

```
1 srand(n1n2n3n4);  
2 T = randm(n);  
3 k = 1.0 - n3*0.02 - n4*0.005 - 0.25;  
4 A = mulmr(T, k);
```

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

`mulmr(T, k)` — розроблена функція множення матриці на коефіцієнт та округлення результату до 0 чи 1.

При проектуванні програм *слід врахувати наступне*:

- 1) мова програмування обирається студентом самостійно;
- 2) графічне зображення графа має формуватися на основі графічних примітивів з графічної бібліотеки (таких як еліпс, пряма, дуга, текст тощо);
- 3) використання готових бібліотек для роботи з графами не дозволяється;
- 4) вивід графа має бути реалізований універсальним чином: вершини і ребра мають виводитися в циклі, а не окремими командами для кожного графічного елемента;
- 5) типи та структури даних для внутрішнього представлення графа у програмі слід вибрати самостійно;
- 6) матриці суміжності графів можна виводити в графічне вікно або консоль — на розсуд студента;
- 7) матриці суміжності мають виводитися як матриці: в квадратному вигляді, з 1 та 0.

**Отже за варіантом 3215 значення будуть такі**

кількість вершин  $n = 10 + n3 = 10 + 1 = 11$

розміщення вершин за трикутником ( $n4=5$ )

Матриця суміжності має такі параметри:

- зерно генератора випадкового числа = 3215
- коефіцієнт  $k = 1.0 - n_3 * 0.02 - n_4 * 0.005 - 0.25 =$

$$= 1.0 - 1 * 0.02 - 5 * 0.005 - 0.25 = 0.705$$

## Текст програми

header/matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#define NUM_VERTICES 11
#define RADIUS 16

double **directedMatrix();
double **undirectedMatrix();
void freeMatrix(double**);
void freeCoords(int**);
void outputMatrix(double**);
#endif
```

lib/coord.c

```
#include <stdlib.h>
#include "../header/matrix.h"
#include <stdio.h>

int **getVerticesCoords(int start) {
    int n = NUM_VERTICES;

    int vertInSide = (int)(n / 3.0 + 0.5) + 1;

    int **coords;

    coords = malloc(n * 2 * sizeof(size_t));
```

```

for (int i = 0; i < 2; i++) {

    coords[i] = malloc(n * sizeof(int));

}

int distance = 100;

int distanceAtDown;

if (n%3 == 0) {

    distanceAtDown = distance*2;

} else if (n%3 == 1) {

    distanceAtDown = (vertInSide-1)*2*distance / vertInSide;

} else {

    distanceAtDown = (vertInSide-1)*2*distance / (vertInSide-2);

}

int tempX = start;

int tempY = start;

for (int j = 0; j < vertInSide; j++) {

    tempX += distance;

    tempY += distance;

    coords[0][j] = tempX;

    coords[1][j] = tempY;

}

for (int j = vertInSide; j < vertInSide*2-1; j++) {

    tempX += distance;

    tempY -= distance;

    coords[0][j] = tempX;

    coords[1][j] = tempY;

}

```

```

    for (int j = vertInSide*2-1; j < n; j++) {

        tempX -= distanceAtDown;

        coords[0][j] = tempX;

        coords[1][j] = tempY;

    }

    return coords;
}

```

lib/coord.c

```

#include <windows.h>

#include <math.h>

#include "../header/matrix.h"

#include <stdio.h>


void drawArrow(HDC, float, int, int);           //
стрілки

void drawLoops(HDC, double**, int**, int);      //
петлі

void drawMultipleArc(HDC, int, int, int, int);  //
кратні дуги

void drawDoubleArc (HDC, int, int, int, int, int); //
верхні і нижні дуги

void drawLeftArc(HDC, int, int, int, int, int); //
ліві дуги

void drawRightArc(HDC, int, int, int, int, int); //
праві дуги

void drawStraightEdge(HDC, int, int, int, int, int); //
прямі ребра

void drawEdges(HDC, int**, int);

int **getVerticesCoords(int);

```

```

void drawGraph(HWND hWnd, HDC hdc, PAINTSTRUCT ps, int start, int
isDirected) {

    int **coords = getVerticesCoords(start);

    int radius = RADIUS, dtx = 5;

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

    SelectObject(hdc, blackPen);

    drawEdges(hdc, coords, isDirected);

    SelectObject(hdc, bluePen);

    for (int j = 0; j < NUM_VERTICES; j++) {

        char num[2];

        itoa(j + 1, num, 10);

        int x = coords[0][j];

        int y = coords[1][j];

        Ellipse(hdc, x - radius, y - radius, x + radius, y + radius);

        TextOut(hdc, x - dtx, y - radius / 2, num, 2);

    }

    freeCoords(coords);
}

void drawEdges(HDC hdc, int **coords, int isDirected) {

    double **matrix = isDirected ? directedMatrix() : undirectedMatrix();

    int n = NUM_VERTICES;

    int vertInSide = (int)(n / 3.0 + 0.5);

    int k = n;

```



```

int distance = abs(coords[0][0] - coords[0][1]);

drawLoops(hdc, matrix, coords, isDirected);

for (int i = 0; i < n; i++) {

    if (!isDirected) k = i;

    for (int j = 0; j < k; j++) {

        if (matrix[i][j] && i != j) {

            int isFirstInCorner;

            int isSecondInCorner;

            int inWhichRowFirst;

            int inWhichRowSecond;

            isFirstInCorner = i==0 || i==vertInSide || i==vertInSide +
vertInSide;

            isSecondInCorner = j==0 || j==vertInSide || j==vertInSide +
vertInSide;

            inWhichRowFirst = i / vertInSide;

            inWhichRowSecond = j / vertInSide;

            int isInside = isFirstInCorner ? !(inWhichRowFirst ==
inWhichRowSecond || ((inWhichRowFirst+2) % 3) == inWhichRowSecond) :
isSecondInCorner ? !(inWhichRowSecond == inWhichRowFirst ||
((inWhichRowSecond+2) % 3) == inWhichRowFirst) : inWhichRowFirst !=
inWhichRowSecond;

            isInside = isInside || (i+1 == j || i-1 == j || i+10 == j);

            int x1 = coords[0][i];

            int y1 = coords[1][i];

            int x2 = coords[0][j];

            int y2 = coords[1][j];

            if (isInside) {

                if (matrix[j][i] && isDirected) {

                    drawMultipleArc(hdc, x1, y1, x2, y2);

```

```

        } else {

            drawStraightEdge(hdc, x1, y1, x2, y2, isDirected);

        }

    } else {

        if (matrix[j][i] && isDirected) {

            if (i > j) {

                drawDoubleArc(hdc, x1, y1, x2, y2, isDirected);

            }

        } else {

            if (y1 == y2) {

                if (x1 < x2) {

                    drawLeftArc(hdc, x1, y1, x2, y2,
isDirected);

                } else {

                    drawRightArc(hdc, x1, y1, x2, y2,
isDirected);

                }

            } else {

                if (y1 < y2) {

                    if (x1 < x2) {

                        drawLeftArc(hdc, x1, y1, x2, y2,
isDirected);

                    } else {

                        drawRightArc(hdc, x1, y1, x2, y2,
isDirected);

                    }

                } else {

                    if (x1 > x2) {

                        drawLeftArc(hdc, x1, y1, x2, y2,
isDirected);

                    } else {

```



```
    // Draw line from starting point of the arrow to the right point of the  
    arrowhead
```

```
    LineTo(hdc, rx, ry);
```

```
}
```

```
void drawLoops(HDC hdc, double **matrix, int **coords, int isDirected) {
```

```
    int n = NUM_VERTICES;
```

```
    int radius = RADIUS;
```

```
    double indent = radius * 2.5;
```

```
    for (int i = 0; i < n; i++) {
```

```
        if (matrix[i][i]) {
```

```
            int x1 = coords[0][i];
```

```
            int y1 = coords[1][i];
```

```
            Ellipse(hdc, x1 - indent, y1 - indent, x1, y1);
```

```
            if (isDirected) drawArrow(hdc, -10, x1 - radius, y1);
```

```
        }
```

```
    }
```

```
}
```

```
void drawMultipleArc(HDC hdc, int x1, int y1, int x2, int y2) {
```

```
    int radius = RADIUS;
```

```
    double cx, cy;
```

```
    double dx, dy;
```

```
    double fi, tanFi;
```

```
    int ax, ay;
```

```
    if (y1 == y2) {
```

```
        cx = (x1 + x2) * 0.5;
```

```
        cy = (y1 + y2) * 0.5;
```

```
        cy = x2 > x1 ? cy + radius : cy - radius;
```

```
        dx = fabs(cx - x2);
```

```
dy = fabs(cy - y2);

tanFi = dy / dx;

fi = atan(tanFi) * 180 / 3.1416;

ax = radius / sqrt(1 + tanFi * tanFi);

ay = ax * tanFi;

ax = x2 > x1 ? -ax : ax;

ay = x2 > x1 ? ay : -ay;

fi = x2 > x1 ? fi : 180 + fi;

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, cx, cy);

MoveToEx(hdc, cx, cy, NULL);

LineTo(hdc, x2, y2);

drawArrow(hdc, fi, x2 + ax, y2 + ay);

} else if (x1 == x2) {

    cx = (x1 + x2) * 0.5;

    cx = y2 > y1 ? cx - radius : cx + radius;

    cy = (y1 + y2) * 0.5;

    dx = fabs(cx - x2);

    dy = fabs(cy - y2);

    tanFi = dx / dy;

    fi = atan(tanFi) * 180 / 3.1416;

    ay = radius / sqrt(1 + tanFi * tanFi);

    ax = ay * tanFi;
```

```

    ax = y2 > y1 ? -ax : ax;

    ay = y2 > y1 ? -ay : ay;

    fi = y2 > y1 ? -90 + fi : 90 + fi;

    MoveToEx(hdc, x1, y1, NULL);

    LineTo(hdc, cx, cy);

    MoveToEx(hdc, cx, cy, NULL);

    LineTo(hdc, x2, y2);

    drawArrow(hdc, fi, x2 + ax, y2 + ay);

} else if (x2 > x1) {

    cx = (x1 + x2) * 0.5 - radius;

    cy = (y1 + y2) * 0.5 - radius;

    dx = fabs(cx - x2);

    dy = fabs(cy - y2);

    if (dx > dy) {

        tanFi = dx / dy;

        fi = 90 - atan(tanFi) * 180 / 3.1416;

        ay = radius / sqrt(1 + tanFi * tanFi);

        ax = ay * tanFi;

    } else {

        tanFi = dy / dx;

        fi = atan(tanFi) * 180 / 3.1416;

        ax = radius / sqrt(1 + tanFi * tanFi);

        ay = ax * tanFi;

    }

    fi = y2 > y1 ? -fi : fi;

```

```

    ay = y2 > y1 ? -ay : ay;

    MoveToEx(hdc, x1, y1, NULL);

    LineTo(hdc, cx, cy);

    MoveToEx(hdc, cx, cy, NULL);

    LineTo(hdc, x2, y2);

    drawArrow(hdc, fi, x2 - ax, y2 + ay);

} else {

    cx = (x1 + x2) * 0.5 + radius;

    cy = (y1 + y2) * 0.5 + radius;

    dx = abs(cx - x2);

    dy = abs(cy - y2);

    if (dx > dy) {

        tanFi = dx / dy;

        fi = 90 - atan(tanFi) * 180 / 3.1416;

        ay = radius / sqrt(1 + tanFi * tanFi);

        ax = ay * tanFi;

    } else {

        tanFi = dy / dx;

        fi = atan(tanFi) * 180 / 3.1416;

        ax = radius / sqrt(1 + tanFi * tanFi);

        ay = ax * tanFi;

    }

    fi = y1 > y2 ? -fi : fi;

    ay = y2 > y1 ? -ay : ay;

```

```

        MoveToEx(hdc, x1, y1, NULL);

        LineTo(hdc, cx, cy);

        MoveToEx(hdc, cx, cy, NULL);

        LineTo(hdc, x2, y2);

        drawArrow(hdc, fi + 180, x2 + ax, y2 + ay);

    }
}

void drawDoubleArc(HDC hdc, int x1, int y1, int x2, int y2, int isDirected)
{

    int radius = RADIUS;

    double cx1 = (x1 + x2) * 0.60;

    double cy1 = (y1 + y2) * 0.53;

    if (y1 == y2) cy1 = (y1 + y2 - 100) * 0.5;

    double dx1 = fabs(cx1 - x2);

    double dy1 = fabs(cy1 - y2);

    double tanFi1 = dx1 / dy1;

    double fi1 = atan2(dy1, dx1) * 180 / 3.1416;

    int ay1 = radius / sqrt(1 + tanFi1 * tanFi1);

    int ax1 = ay1 * tanFi1;

    double cx2 = (x1 + x2) * 0.52;

    double cy2 = (y1 + y2) * 0.64;

    if (y1 == y2) cy2 = (y1 + y2 - 100) * 0.5;

    double dx2 = fabs(cx2 - x2);

    double dy2 = fabs(cy2 - y2);

    double tanFi2 = dx2 / dy2;

```



```

double fi2 = atan2(dy2, dx2) * 180 / 3.1426;

int ay2 = radius / sqrt(1 + tanFi2 * tanFi2);

int ax2 = ay2 * tanFi2;

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, cx1, cy1);

MoveToEx(hdc, cx1, cy1, NULL);

LineTo(hdc, x2, y2);

if (isDirected) drawArrow(hdc, fi1, x2 + ax1, y2 + ay1);

MoveToEx(hdc, x2, y2, NULL);

LineTo(hdc, cx2, cy2);

MoveToEx(hdc, cx2, cy2, NULL);

LineTo(hdc, x1, y1);

if (isDirected) drawArrow(hdc, fi2, x1 + ax2, y1 + ay2);
}

void drawLeftArc(HDC hdc, int x1, int y1, int x2, int y2, int isDirected )
{
    int radius = RADIUS;

    double cx = (x1 + x2) * 0.5 - radius * 2;

    double cy = (y1 + y2) * 0.55;

    if (y1 == y2) cy = (y1 + y2 - 100) * 0.5;

    double dx = fabs(cx - x2);

    double dy = fabs(cy - y2);

    double tanFi = dx / dy;

    double fi = atan2(dy, dx) * 180 / 3.1416;

    int ay = radius / sqrt(1 + tanFi * tanFi);

    int ax = ay * tanFi;

```

```

    if (y1 == y2) {

        ax = -ax;

        ay = -ay;

        fi = -fi;

    } else {

        if (y1 < y2) {

            ax = -ax;

            ay = -ay;

            fi = -fi;

        } else {

            fi = 180 - fi;

        }

    }

    MoveToEx(hdc, x1, y1, NULL);

    LineTo(hdc, cx, cy);

    MoveToEx(hdc, cx, cy, NULL);

    LineTo(hdc, x2, y2);

    if (isDirected) drawArrow(hdc, fi, x2 + ax, y2 + ay);
}

void drawRightArc(HDC hdc, int x1, int y1, int x2, int y2, int isDirected)
{

    int radius = RADIUS;

    double cx = (x1 + x2) * 0.5 + radius * 2;

    double cy = (y1 + y2) * 0.55;

    if (y1 == y2) cy = (y1 + y2 - 100) * 0.5;

    double dx = abs(cx - x2);

    double dy = abs(cy - y2);

```

```

double tanFi = dy / dx;

double fi = atan(tanFi) * 180 / 3.1416;

int ax = radius / sqrt(1 + tanFi * tanFi);

int ay = ax * tanFi;

if (y1 == y2) {

    ay = -ay;

    fi = 180 + fi;

} else {

    if (y1 < y2) {

        ay = -ay;

        fi = 180 + fi;

    } else {

        ax = -ax;

    }

}

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, cx, cy);

MoveToEx(hdc, cx, cy, NULL);

LineTo(hdc, x2, y2);

if (isDirected) drawArrow(hdc, fi, x2 + ax, y2 + ay);
}

void drawStraightEdge(HDC hdc, int x1, int y1, int x2, int y2, int
isDirected) {

    int radius = RADIUS;

    int absx = abs(x1 - x2);

    int absy = abs(y1 - y2);

    double cx, cy;

    double fi, tanFi;

```

```

int ax, ay;

if (y1 == y2) {

    fi = x2 > x1 ? 0 : 180;

    ax = x2 > x1 ? -radius : radius;

    MoveToEx(hdc, x1, y1, NULL);

    LineTo(hdc, x2, y2);

    if (isDirected) drawArrow(hdc, fi, x2 + ax, y2);

} else if (x1 == x2) {

    fi = y2 > y1 ? -90 : 90;

    ay = y2 > y1 ? -radius : radius;

    MoveToEx(hdc, x1, y1, NULL);

    LineTo(hdc, x2, y2);

    if (isDirected) drawArrow(hdc, fi, x2, y2 + ay);

} else if (x2 > x1) {

    if (absx > absy) {

        tanFi = absx / absy;

        fi = 90 - atan(tanFi) * 180 / 3.1416;

        ay = radius / sqrt(1 + tanFi * tanFi);

        ax = ay * tanFi;

    } else {

        tanFi = absy / absx;

        fi = atan(tanFi) * 180 / 3.1416;

        ax = radius / sqrt(1 + tanFi * tanFi);

        ay = ax * tanFi;
    }
}

```

```

    }

    fi = y2 > y1 ? -fi : fi;

    ay = y2 > y1 ? -ay : ay;

    MoveToEx(hdc, x1, y1, NULL);

    LineTo(hdc, x2, y2);

    if (isDirected) drawArrow(hdc, fi, x2 - ax, y2 + ay);

} else if (x1 > x2) {

    if (absx > absy) {

        tanFi = absx / absy;

        fi = 90 - atan(tanFi) * 180 / 3.1416;

        ay = radius / sqrt(1 + tanFi * tanFi);

        ax = ay * tanFi;

    } else {

        tanFi = absy / absx;

        fi = atan(tanFi) * 180 / 3.1416;

        ax = radius / sqrt(1 + tanFi * tanFi);

        ay = ax * tanFi;

    }

    fi = y1 > y2 ? -fi : fi;

    ay = y2 > y1 ? -ay : ay;

    MoveToEx(hdc, x1, y1, NULL);

    LineTo(hdc, x2, y2);

    if (isDirected) drawArrow(hdc, fi + 180, x2 + ax, y2 + ay);

}

```

```
}
```

lib/matrix.c

```
#include <stdlib.h>

#include <stdio.h>

#define NUM_VERTICES 11

#define SEED 3215

#define K 0.705

double **randm() {

    int n = NUM_VERTICES;

    srand(SEED);

    double **matrix = (double **) malloc(sizeof(double *) * n);

    for (int i = 0; i < n; i++) {

        matrix[i] = (double *) malloc(sizeof(double) * n);

    }

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            matrix[i][j] = (double) (rand() * 2.0) / (double) RAND_MAX;

        }

    }

    return matrix;

}

double **mulmr(double **matrix) {

    int n = NUM_VERTICES;

    for (int i = 0; i < n; i++) {
```

```

        for (int j = 0; j < n; j++) {

            matrix[i][j] *= K;

            matrix[i][j] = matrix[i][j] < 1 ? 0 : 1;

        }

    }

    return matrix;

}

void freeMatrix(double **matrix) { // звільнення пам'яті від матриці

    int n = NUM_VERTICES;

    for (int i = 0; i < n; i++) free(matrix[i]);

    free(matrix);

}

void freeCoords(int **coords) { // звільнення пам'яті від координат

    for (int i = 0; i < 2; i++) free(coords[i]);

    free(coords);

}

double **directedMatrix() { // генерація матриці орієнтованого графа

    double **T = randm();

    double **A = mulmr(T);

    return A;

}

double **undirectedMatrix() { // генерація матриці неорієнтованого графа

    int n = NUM_VERTICES;

    double **A = directedMatrix();

    for (int i = 0; i < n; i++) {

```

```

        for (int j = 0; j < n; j++) {

            if (A[i][j]) A[j][i] = 1;

        }

    }

    return A;
}

void outputMatrix(double **matrix) {

    int n = NUM_VERTICES;

    for (int i = 0; i < n; i++) {

        printf("\n");

        for (int j = 0; j < n; j++) {

            printf("%.01f  ", matrix[i][j]);

        }

    }

    printf("\n");
}

```

app.c

```

#include <windows.h>

#include <stdio.h>

#define WIDTH 1200

#define HEIGHT 1200

#define START 100

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

void drawGraph(HWND, HDC, PAINTSTRUCT, int, int);

```



```
char ProgName[] = "Lab 3 by Korol Oleksandr";

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpzCmdLine, 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,

        "Lab 3 by Korol Oleksandr",

        WS_OVERLAPPEDWINDOW,

        10,

        10,

        WIDTH,

        HEIGHT,
```

```

        (HWND) NULL,

        (HMENU) NULL,

        (HINSTANCE) hInstance,

        (HINSTANCE) NULL

    );

    ShowWindow(hWnd, nCmdShow);

    int flag;

    while((flag = GetMessage(&lpMsg, hWnd, 0, 0)) != 0) {

        if (flag == -1) return lpMsg.wParam;

        TranslateMessage(&lpMsg);

        DispatchMessage(&lpMsg);

    }

    DestroyWindow(hWnd);

    return 0;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM
lParam) {

    HDC hdc;

    PAINTSTRUCT ps;

    RECT rect = {0, 0, WIDTH, HEIGHT};

    static BOOL shiftPressed = TRUE;

    static BOOL ctrlPressed = FALSE;

    switch (msg) {

```

```

    case WM_PAINT:

        hdc = BeginPaint(hWnd, &ps);

        FillRect(hdc, &rect, WHITE_BRUSH);

        TextOut(hdc, WIDTH * 0.5, HEIGHT - START, "press: Shift -
directed, Ctrl - undirected", 42);

        if (shiftPressed) drawGraph(hWnd, hdc, ps, START, 1);

        if (ctrlPressed) drawGraph(hWnd, hdc, ps, START, 0);

        EndPaint(hWnd, &ps);

        break;

    case WM_KEYDOWN:

        if (wParam == VK_SHIFT) {

            shiftPressed = TRUE;

            ctrlPressed = FALSE;

        }

        if (wParam == VK_CONTROL) {

            shiftPressed = FALSE;

            ctrlPressed = TRUE;

        }

        InvalidateRect(hWnd, NULL, TRUE);

        break;

    case WM_DESTROY:

        PostQuitMessage(0);

        break;

    default:

        return(DefWindowProc(hWnd, messg, wParam, lParam));

}
}

```

print.c

```
#include <stdio.h>

#include "../header/matrix.h"

int main() {

    double **directed = directedMatrix();

    double **undirected = undirectedMatrix();

    printf("\nDirected Graph Matrix");

    outputMatrix(directed);

    freeMatrix(directed);

    printf("\nUndirected Graph Matrix");

    outputMatrix(undirected);

    freeMatrix(undirected);

}
```

**Згенеровані за варіантом матриці суміжності напрямленого і  
ненаправленого графів**

### Directed Graph Matrix

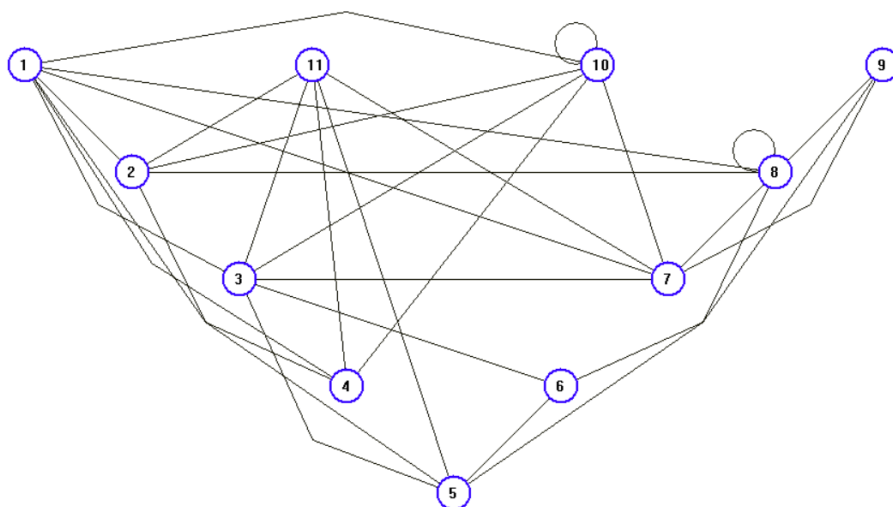
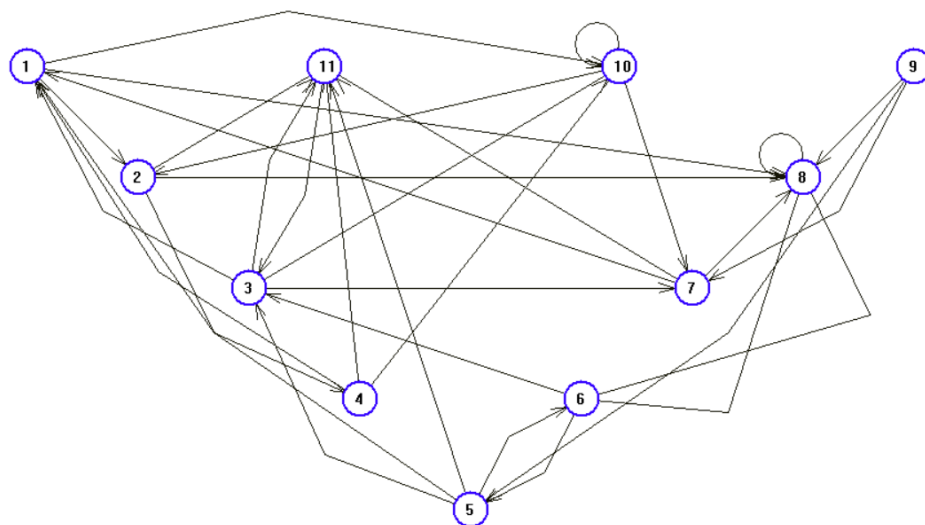
0	1	0	0	0	0	0	1	0	1	0
0	0	0	1	0	0	0	1	0	0	1
1	0	0	0	0	0	1	0	0	1	1
1	0	0	0	0	0	0	0	0	1	1
1	0	1	0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	1	1	0	0	0
0	1	0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0

### Undirected Graph Matrix

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

Process finished with exit code 0

**Результати тестування програми**



## Висновки

Під час виконання цієї лабораторної роботи я отримав цінний практичний досвід з графічного відображення графів. Робота з випадковою матрицею суміжності та графічним представленням графа дозволила мені краще зрозуміти структуру графів та їх взаємозв'язки.

Розділивши граф на вершини та ребра та використовуючи різноманітні методи для їх відображення, я отримав можливість працювати з графами відповідно до їхньої структури. Це виявилось ефективним та корисним способом для візуалізації графічних даних.

У процесі вивчення теоретичного матеріалу я ознайомився з різними аспектами роботи з графами, включаючи їхню математичну модель, роль вершин і ребер, а також алгоритми визначення циклів та шляхів.

Це дозволило мені отримати глибше розуміння принципів функціонування графів та їх використання в різних областях, таких як комп'ютерні мережі, транспортні системи тощо.

Результати дослідження підтвердили, що графічне відображення графів є потужним інструментом для аналізу та візуалізації даних. Воно допомагає зрозуміти взаємозв'язки між об'єктами та виявляти закономірності у складних системах.

Отже, ця лабораторна робота дала мені не лише практичний досвід роботи з графічним відображенням графів, але й поглиблене розуміння їхньої структури та використання, що буде корисним у подальших проектах та дослідженнях в галузі комп'ютерних наук.