

**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  
Коваленко Михайло Володимирович  
номер у списку групи: 11

Перевірила:

Молчанова А. А.

Київ 2023

## ЛАБОРАТОРНА РОБОТА №2.3.

### ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ ГРАФІВ

#### Мета лабораторної роботи

Метою лабораторної роботи №3. «Графічне представлення графів» є набуття практичних навичок представлення графів у комп'ютері та ознайомлення з принципами роботи ОС.

#### Постановка задачі

1. Представити у програмі напрямлений і ненаправлений граф з заданими параметрами:

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

Параметри задаються на основі номера групи, представленого десятковими цифрами  $p_1$ ,  $p_2$  та номера студента у списку групи — десяткового числа  $p_3$ ,  $p_4$ .

Число вершин  $n$  дорівнює  $10 + p_3$ .

Розміщення вершин:

- колом при  $p_4 = 0,1$ ;
- прямокутником (квадратом) при  $p_4 = 2,3$ ;
- трикутником при  $p_4 = 4,5$ ;
- колом з вершиною в центрі при  $p_4 = 6,7$ ;
- прямокутником (квадратом) з вершиною в центрі при  $p_4 = 8,9$ .

Наприклад, при  $p_4 = 10$  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).

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

### Завдання для конкретного варіанту:

Варіант 11

Число вершин  $n = 10 + 1 = 11$

Вершини розміщені колом ( $n_4 = 1$ )

Seed: 2211

### Текст програми:

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <math.h>
#define verts_amount 11
#define IDC_BUTTON 1
#define IDC_BUTTON2 2

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

char ProgName[] = "Lab 3 Mykhailo Kovalenko";

struct coordinates {
    double nx[verts_amount];
    double ny[verts_amount];
    double loopX[verts_amount];
```

```

    double loopY[verts_amount];
};

double **randm(int n) {
    srand(2211);
    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 coef, double **matrix, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matrix[i][j] *= coef;
            matrix[i][j] = matrix[i][j] < 1 ? 0 : 1;
        }
    }
    return matrix;
}

void freeMatrix(double **matrix, int n) {
    for (int i = 0; i < n; ++i) {
        free(matrix[i]);
    }
    free(matrix);
}

void printMatrix(double **matrix, int n, int initialX, int initialY,
HDC hdc) {
    for (int i = 0, y = initialY + 30; i < n; i++, y += 15) {
        for (int j = 0, x = initialX; j < n; j++, x += 13) {
            wchar_t buffer[2];
            swprintf(buffer, 2, L"%lf", matrix[i][j]);
            TextOut(hdc, x, y, (LPCSTR) buffer, 1);
        }
        MoveToEx(hdc, initialX, y, NULL);
    }
}

void arrow(double fi, double px, double py, HDC hdc) {
    double 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);
}

double **symmetricMatrix(double **matrix, int n) {
    double **symmetrical = (double **) malloc(n * sizeof(double *));
    for (int i = 0; i < n; ++i) {
        symmetrical[i] = (double *) malloc(n * sizeof(double));
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            symmetrical[i][j] = matrix[i][j];
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (symmetrical[i][j] != symmetrical[j][i]) {
                symmetrical[i][j] = 1;
                symmetrical[j][i] = 1;
            }
        }
    }
    return symmetrical;
}

void drawArch(int startX, int startY, int finishX, int finishY, int
archInterval, HDC hdc) {
    XFORM transformMatrix;
    XFORM initialMatrix;
    GetWorldTransform(hdc, &initialMatrix);

    double angle = atan2(finishY - startY, finishX - startX) - M_PI /
2.0;
    transformMatrix.eM11 = (FLOAT) cos(angle);
    transformMatrix.eM12 = (FLOAT) sin(angle);
    transformMatrix.eM21 = (FLOAT) (-sin(angle));
    transformMatrix.eM22 = (FLOAT) cos(angle);
    transformMatrix.eDx = (FLOAT) startX;
    transformMatrix.eDy = (FLOAT) startY;
    SetWorldTransform(hdc, &transformMatrix);

    const double archWidth = 0.75;
    double length = sqrt((finishX - startX) * (finishX - startX) +
(finishY - startY) * (finishY - startY));

```

```

    double radiusOfVertex = 15.0;
    double semiMinorAxis = archWidth * length;
    double semiMajorAxis = length / 2;
    double vertexAreaSquared = semiMajorAxis * semiMajorAxis *
radiusOfVertex * radiusOfVertex;
    double semiAxesSquared = semiMinorAxis * semiMinorAxis *
semiMajorAxis * semiMajorAxis;
    double ellipseStartY = semiMajorAxis;
    double radius = semiMinorAxis * semiMinorAxis * ellipseStartY *
ellipseStartY;
    double distance = semiMinorAxis * semiMinorAxis * radiusOfVertex
* radiusOfVertex;
    double semiMinorAxisPow = pow(semiMinorAxis, 4);
    double crossing = semiMajorAxis * sqrt(vertexAreaSquared -
semiAxesSquared + radius - distance + semiMinorAxisPow);
    double semiMinorAxisSquaredEllipseStartY = semiMinorAxis *
semiMinorAxis * ellipseStartY;
    double denominator = -semiMajorAxis * semiMajorAxis +
semiMinorAxis * semiMinorAxis;
    double contactYRightTop = (semiMinorAxisSquaredEllipseStartY -
crossing) / denominator;
    double contactXRightTop = sqrt(radiusOfVertex * radiusOfVertex -
contactYRightTop * contactYRightTop);
    double contactYBottom = length - contactYRightTop;
    double contactXLeftBottom = -contactXRightTop;

    if (archInterval <= verts_amount / 2) {
        Arc(hdc, -archWidth * length, length, archWidth * length, 0,
0, 0, 0, length);
        double angleOfArrow = -atan2(length - contactYBottom,
contactXLeftBottom) + 0.3 / 3;
        arrow(angleOfArrow, contactXLeftBottom, contactYBottom, hdc);
    } else {
        Arc(hdc, -archWidth * length, length, archWidth * length, 0,
0, length, 0, 0);
        double angleOfArrow = -atan2(length - contactYBottom, -
contactXLeftBottom) - 0.3 / 3;
        arrow(angleOfArrow, -contactXLeftBottom, contactYBottom,
hdc);
    }

    SetWorldTransform(hdc, &initialMatrix);
}

void drawDirectedGraph(int centerX, int centerY, int radiusOfGraph,
int radiusOfVertex, int radiusOfLoop, double angle,
                        struct coordinates coordinates, double
**matrix,

```

```

        HPEN KPen, HPEN GPen, HDC hdc) {
    for (int i = 0; i < verts_amount; i++) {
        for (int j = 0; j < verts_amount; j++) {
            MoveToEx(hdc, coordinates.nx[i], coordinates.ny[i],
NULL);
            if ((j >= i && matrix[i][j] == 1) || (j <= i &&
matrix[i][j] == 1 && matrix[j][i] == 0)) {
                if (i == j) {
                    SelectObject(hdc, GPen);
                    Ellipse(hdc, coordinates.loopX[i] - radiusOfLoop,
coordinates.loopY[i] - radiusOfLoop,
                        coordinates.loopX[i] + radiusOfLoop,
coordinates.loopY[i] + radiusOfLoop);

                    double triangleHeight = sqrt(3) * radiusOfVertex
/ 2.;
                    double radiusOfContact = radiusOfGraph +
radiusOfLoop / 2.;
                    double distance = sqrt(radiusOfContact *
radiusOfContact + triangleHeight * triangleHeight);
                    double angleToContactVertex =
atan2(coordinates.ny[i] - centerY, coordinates.nx[i] - centerX);
                    double loopAngle = atan2(triangleHeight,
radiusOfContact);
                    double contactCoordX = centerX + distance *
cos(angleToContactVertex + loopAngle);
                    double contactCoordY = centerY + distance *
sin(angleToContactVertex + loopAngle);
                    double curveAngle = angleToContactVertex + 0.3 /
2.;
                    arrow(curveAngle, contactCoordX, contactCoordY,
hdc);
                    SelectObject(hdc, KPen);
                } else {
                    LineTo(hdc, coordinates.nx[j],
coordinates.ny[j]);
                    double line_angle = atan2(coordinates.ny[i] -
coordinates.ny[j], coordinates.nx[i] - coordinates.nx[j]);
                    arrow(line_angle, coordinates.nx[j] +
radiusOfVertex * cos(line_angle),
                        coordinates.ny[j] + radiusOfVertex *
sin(line_angle), hdc);
                }
            } else if (j < i && matrix[i][j] == 1 && matrix[j][i] ==
1) {
                drawArch(coordinates.nx[i], coordinates.ny[i],
coordinates.nx[j], coordinates.ny[j], fabs(i - j), hdc);
            }
        }
    }
}

```

```

    }
}

void drawUndirectedGraph(int centerX, int centerY, int radiusOfGraph,
int radiusOfVertex, int radiusOfLoop, double angle,
                        struct coordinates coordinates, double
**matrix,
                        HPEN KPen, HPEN GPen, HDC hdc) {
    for (int i = 0; i < verts_amount; ++i) {
        for (int j = 0; j < verts_amount; ++j) {
            MoveToEx(hdc, coordinates.nx[i], coordinates.ny[i],
NULL);
            if (matrix[i][j] == 1) {
                if (i == j) {
                    SelectObject(hdc, GPen);
                    Ellipse(hdc, coordinates.loopX[i] - radiusOfLoop,
coordinates.loopY[i] - radiusOfLoop,
coordinates.loopX[i] + radiusOfLoop,
coordinates.loopY[i] + radiusOfLoop);
                    SelectObject(hdc, KPen);
                    MoveToEx(hdc, coordinates.loopX[i],
coordinates.loopY[i] + radiusOfLoop, NULL);
                } else {
                    LineTo(hdc, coordinates.nx[j],
coordinates.ny[j]);
                }
            }
        }
    }
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, 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,
                    (LPCSTR) "Lab3 (Mykhailo Kovalenko IM-22)",
                    WS_OVERLAPPEDWINDOW,
                    100,
                    100,
                    1050,
                    800,
                    (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;
    HWND Button_directed;
    HWND Button_undirected;
    int state = 0;
    switch (messg) {
        case WM_CREATE: {
            Button_directed = CreateWindow(
                (LPCSTR) "BUTTON",
                (LPCSTR) "Directed",
                WS_TABSTOP | WS_VISIBLE | WS_CHILD |
BS_DEFPUSHBUTTON,
                700,
                500,
                160,
                50,
                hWnd,
                (HMENU) IDC_BUTTON,
                (HINSTANCE) GetWindowLongPtr(hWnd,
GWLP_HINSTANCE),
                NULL);
            Button_undirected = CreateWindow(
                (LPCSTR) "BUTTON",
                (LPCSTR) "Undirected",
                WS_TABSTOP | WS_VISIBLE | WS_CHILD |
BS_DEFPUSHBUTTON,
                700,

```

```

        580,
        160,
        50,
        hWnd,
        (HMENU) IDC_BUTTON2,
        (HINSTANCE) GetWindowLongPtr(hWnd,
GWL_P_HINSTANCE),
        NULL);
    return 0;
}
case WM_COMMAND: {
    switch (LOWORD(wParam)) {

        case IDC_BUTTON:
            state = 0;
            InvalidateRect(hWnd, NULL, FALSE);
            break;

        case IDC_BUTTON2:
            state = 1;
            InvalidateRect(hWnd, NULL, FALSE);
            break;

    }
}
case WM_PAINT :
    hdc = BeginPaint(hWnd, &ps);
    HFONT hFont = CreateFont(16, 0, 0, 0, FW_NORMAL, FALSE,
FALSE, FALSE,
                                DEFAULT_CHARSET,
OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
                                DEFAULT_QUALITY, DEFAULT_PITCH |
FF_DONTCARE, L"Arial");
    SelectObject(hdc, hFont);
    DeleteObject(hFont);

    SetGraphicsMode(hdc, GM_ADVANCED);
    HPEN BPen = CreatePen(PS_SOLID, 2, RGB(50, 0, 255));
    HPEN KPen = CreatePen(PS_SOLID, 1, RGB(20, 20, 5));
    HPEN GPen = CreatePen(PS_SOLID, 2, RGB(0, 255, 0));
    HPEN NoPen = CreatePen(PS_NULL, 0, RGB(0, 0, 0));
    SelectObject(hdc, NoPen);
    Rectangle(hdc, 0, 0, 670, 700);

    char *nn[verts_amount] = {"1", "2", "3", "4", "5", "6",
"7", "8", "9", "10\0", "11\0"};

    struct coordinates coordinates;
    double circleRadius = 200;
    double vertexRadius = circleRadius / 11;

```

```

        double loopRadius = vertexRadius;
        double dtx = vertexRadius / 2.5;
        double circleCenterX = 370;
        double circleCenterY = 360;

        double angleAlpha = 2.0 * M_PI / (double) verts_amount;
        for (int i = 0; i < verts_amount; i++) {
            double sinAlpha = sin(angleAlpha * (double) i);
            double cosAlpha = cos(angleAlpha * (double) i);
            coordinates.nx[i] = circleCenterX + circleRadius *
sinAlpha;
            coordinates.ny[i] = circleCenterY - circleRadius *
cosAlpha;
            coordinates.loopX[i] = circleCenterX + (circleRadius
+ loopRadius) * sinAlpha;
            coordinates.loopY[i] = circleCenterY - (circleRadius
+ loopRadius) * cosAlpha;
        }

        double **T = randm(verts_amount);
        double coefficient = 1.0 - 0.02 - 0.005 - 0.25;
        double **A = mulmr(coefficient, T, verts_amount);
        int DefaultMatrixX = 720;
        int DefaultMatrixY = 50;
        TextOut(hdc, DefaultMatrixX, DefaultMatrixY, (LPCSTR)
L"Default Matrix", 28);
        printMatrix(A, verts_amount, DefaultMatrixX,
DefaultMatrixY, hdc);
        double **R = randm(verts_amount);
        double **C = symmetricMatrix(mulmr(coefficient, R,
verts_amount), verts_amount);
        int DefaultSymmetricMatrixX = DefaultMatrixX;
        int DefaultSymmetricMatrixY = DefaultMatrixY + 210;
        TextOut(hdc, DefaultSymmetricMatrixX,
DefaultSymmetricMatrixY, (LPCSTR) L"Symmetric Matrix", 31);
        printMatrix(C, verts_amount, DefaultSymmetricMatrixX,
DefaultSymmetricMatrixY, hdc);

        SelectObject(hdc, GetStockObject(HOLLOW_BRUSH));
        SelectObject(hdc, KPen);

        if (state == 0) {
            drawDirectedGraph(circleCenterX, circleCenterY,
circleRadius, vertexRadius, loopRadius, angleAlpha,
coordinates, A, KPen, GPen, hdc);
        } else {
            drawUndirectedGraph(circleCenterX, circleCenterY,
circleRadius, vertexRadius, loopRadius, angleAlpha,
coordinates, C, KPen, GPen, hdc);

```

```

    }

    SelectObject(hdc, BPen);
    SelectObject(hdc, GetStockObject(DC_BRUSH));
    SetDCBrushColor(hdc, RGB(204, 204, 255));
    SetBkMode(hdc, TRANSPARENT);

    for (int i = 0; i < verts_amount; ++i) {
        Ellipse(hdc, coordinates.nx[i] - vertexRadius,
coordinates.ny[i] - vertexRadius,
coordinates.nx[i] + vertexRadius,
coordinates.ny[i] + vertexRadius);
        TextOut(hdc, coordinates.nx[i] - dtx,
coordinates.ny[i] - vertexRadius / 2, nn[i], 2);
    }
    EndPaint(hWnd, &ps);
    freeMatrix(A, verts_amount);
    freeMatrix(C, verts_amount);
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return (DefWindowProc(hWnd, messg, wParam, lParam));
}
return 0;
}

```

**Згенеровані матриці суміжності напруженого і ненапруженого графів:**

### Default Matrix

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

### Symmetric Matrix

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

**Скріншоти напрямленого і ненаправленого графів за варіантом:**

