

**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 №5
з дисципліни
«Алгоритми і структури даних»

Виконав

Студент групи ІМ-22
Тимофеев Даниїл Костянтинівич
номер у списку групи: 20

Перевірила:

Молчанова А. А.

Київ 2023

Постановка задачі :

1. Представити напрямлений граф з заданими параметрами так само, як у лабораторній роботі No3. Відміна: матриця A за варіантом формується за функцією:

$$A = \text{mulmr}((1.0 - n3*0.01 - n4*0.005 - 0.15)*T);$$

2. Створити програми для обходу в глибину та в ширину. Обхід починати з вершини, яка має вихідні дуги. При цьому у програмі:

- встановити зупинку у точці призначення номеру черговій вершині за допомогою повідомлення про натискання кнопки,
- виводити зображення графа у графічному вікні перед кожною зупинкою.

3. Під час обходу графа побудувати дерево обходу. Вивести побудоване дерево у графічному вікні.

Завдання варіанту № 20 :

Номер – 2220

Число вершин = $10 + 2 = 12$.

Розміщення вершин колом , бо $n4 = 0$;

Srand(2220)

Текст програми мовою C.

Файл *main.c* :

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <math.h>
#include "props.h"
#define vertices 12
#define IDC_BUTTON 1
#define IDC_BUTTON2 2

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

char ProgName[] = "Lab 5";

struct coordinates {
```

```

    double nx[vertices];
    double ny[vertices];
    double loopX[vertices];
    double loopY[vertices];
};

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);
}

void depictArch(int startX, int startY, int finalX, int finalY, int
archInterval, HDC hdc) {
    XFORM transformMatrix;
    XFORM initialMatrix;
    GetWorldTransform(hdc, &initialMatrix);

    double angle = atan2(finalY - startY, finalX - 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 archWidthRatio = 0.75;
    double archLength = sqrt((finalX - startX) * (finalX - startX) + (finalY -
startY) * (finalY - startY));
    double radiusOfVertex = 15.0;
    double semiMinorAxis = archWidthRatio * archLength;
    double semiMajorAxis = archLength / 2;

    double ellipseStartY = semiMajorAxis;

    double vertexAreaSquared = semiMajorAxis * semiMajorAxis * radiusOfVertex *
radiusOfVertex;
    double semiAxesSquared = semiMinorAxis * semiMinorAxis * semiMajorAxis *
semiMajorAxis;
    double distanceFromCenter = semiMinorAxis * semiMinorAxis * ellipseStartY *
ellipseStartY;
    double distanceFromVertex = semiMinorAxis * semiMinorAxis * radiusOfVertex
* radiusOfVertex;
    double semiMinorAxisPow = pow(semiMinorAxis, 4);

    double intersection = semiMajorAxis *
sqrt(vertexAreaSquared - semiAxesSquared +

```

```

distanceFromCenter - distanceFromVertex +
                    semiMinorAxisPow);
    double semiMinorAxisSquaredEllipseStartY = semiMinorAxis * semiMinorAxis *
ellipseStartY;
    double denominator = -semiMajorAxis * semiMajorAxis + semiMinorAxis *
semiMinorAxis;

    double contactYRightTop = (semiMinorAxisSquaredEllipseStartY -
intersection) / denominator;
    double contactXRightTop = sqrt(radiusOfVertex * radiusOfVertex -
contactYRightTop * contactYRightTop);
    double contactYBottom = archLength - contactYRightTop;
    double contactXLeftBottom = -contactXRightTop;

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

    SetWorldTransform(hdc, &initialMatrix);
}

void depictDirectedGraph(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 < vertices; i++) {
        for (int j = 0; j < vertices; 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 radiusOfContact = radiusOfGraph + radiusOfLoop / 2.;
                    double triangleHeight = sqrt(3) * radiusOfVertex / 2.;
                    double loopAngle = atan2(triangleHeight, radiusOfContact);
                    double contactDistance = sqrt(radiusOfContact * radiusOfContact +
triangleHeight * triangleHeight);
                    double angleToContactVertex = atan2(coordinates.ny[i] - centerY,
coordinates.nx[i] - centerX);

                    double contactPointX = centerX + contactDistance *
cos(angleToContactVertex + loopAngle);
                    double contactPointY = centerY + contactDistance *
sin(angleToContactVertex + loopAngle);

```

```

        double curvatureAngle = angleToContactVertex + 0.3 / 2.;
        arrow(curvatureAngle, contactPointX, contactPointY, 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) {
        depictArch(coordinates.nx[i], coordinates.ny[i], coordinates.nx[j],
coordinates.ny[j], fabs(i - j), hdc);
    }
}
}
}

int dfsIterationAmount = 0;
int bfsIterationAmount = 0;

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) "Lab 5. by Daniil Timofeev from IM-22",
WS_OVERLAPPEDWINDOW,
100,
100,
1500,
750,
(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 ButtonForDFS;
HWND ButtonForBFS;
const int numVertices = vertices;
int state = 0;

double **T = randm(vertices);
double coefficient = 1.0 - 0.01 - 0.005 - 0.15;
double **A = mulmr(coefficient, T, vertices);

int* visitedVertex = malloc(vertices * sizeof(int));
int* depthVertices = malloc(vertices * sizeof(int));
int* queue = malloc(vertices * sizeof(int));
int birthVertex = findFirstArch(A, vertices);
double** dfsTree = createMatrix(vertices);
double** bfsTree = createMatrix(vertices);
chargeWithZero(dfsTree, vertices);
chargeWithZero(bfsTree, vertices);

runDfsForNotVisitedVertices(birthVertex, A, visitedVertex, 0,
depthVertices, dfsTree);
breadthFirstSearch(A, birthVertex, queue, bfsTree);

switch (messg) {
case WM_CREATE: {
    ButtonForDFS = CreateWindow(
        (LPCSTR) "BUTTON",
        (LPCSTR) "Next Step: DFS",
        WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
        700,
        30,
        160,
        50,
        hWnd,
        (HMENU) IDC_BUTTON,
        (HINSTANCE) GetWindowLongPtr(hWnd, GWLP_HINSTANCE),
        NULL);
    ButtonForBFS = CreateWindow(
        (LPCSTR) "BUTTON",
        (LPCSTR) "Next Step: BFS",
        WS_TABSTOP | WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
        700,
        600,
        160,
        50,
        hWnd,
        (HMENU) IDC_BUTTON2,
        (HINSTANCE) GetWindowLongPtr(hWnd, GWLP_HINSTANCE),
        NULL);
    return 0;
}
case WM_COMMAND: {
    switch (LOWORD(wParam)) {

        case IDC_BUTTON:
            state = 0;
            if (dfsIterationAmount < numVertices) dfsIterationAmount++;
            InvalidateRect(hWnd, NULL, FALSE);
            break;

        case IDC_BUTTON2:
            state = 1;
            if (bfsIterationAmount < numVertices) bfsIterationAmount++;
    }
}
}

```

```

        InvalidateRect(hWnd, NULL, FALSE);
        break;
    }
}
case WM_PAINT :
    hdc = BeginPaint(hWnd, &ps);
    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 OPen = CreatePen(PS_SOLID, 2, RGB(255, 165, 0));
    HPEN PPen = CreatePen(PS_SOLID, 2, RGB(255, 0, 255));
    HPEN NoPen = CreatePen(PS_NULL, 0, RGB(0, 0, 0));
    SelectObject(hdc, NoPen);
    Rectangle(hdc, 0, 0, 670, 700);

    char *nn[vertices] = {"1", "2", "3", "4", "5", "6", "7", "8", "9",
"10\0", "11\0", "12\0"};

    struct coordinates coordinates;

    double circleRadius = 200;
    double vertexRadius = circleRadius / 12;

    double loopRadius = vertexRadius;
    double dtx = vertexRadius / 2.5;

    double circleCenterX = 370;
    double circleCenterY = 360;

    double angleAlpha = 2.0 * M_PI / (double) vertices;
    for (int i = 0; i < vertices; 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;

    }

    int initialXOofRandMatrix = 750;
    int initialYofRandMatrix = 150;
    double** dfsDetour = createTraversalMatrix(depthVertices);
    double** bfsDetour = createTraversalMatrix(queue);
    TextOut(hdc, initialXOofRandMatrix, initialYofRandMatrix, (LPCSTR)
L"Initial Matrix", 28);
    printMatrix(A, vertices, initialXOofRandMatrix, initialYofRandMatrix,
hdc);

    TextOut(hdc, initialXOofRandMatrix+200, initialYofRandMatrix, (LPCSTR)
L"DFS Relativity", 28);
    printMatrix(dfsDetour, vertices, initialXOofRandMatrix + 200,
initialYofRandMatrix, hdc);
    TextOut(hdc, initialXOofRandMatrix+400, initialYofRandMatrix, (LPCSTR)
L"DFS Tree", 15);
    printMatrix(dfsTree, vertices, initialXOofRandMatrix + 400,
initialYofRandMatrix, hdc);

```

```

        TextOut(hdc, initialXOfRandMatrix+200, initialYofRandMatrix+250,
(LPCSTR) L"BFS Relativity", 28);
        printMatrix(bfsDetour, vertices, initialXOfRandMatrix + 200,
initialYofRandMatrix + 250, hdc);
        TextOut(hdc, initialXOfRandMatrix+400, initialYofRandMatrix+250,
(LPCSTR) L"BFS Tree", 15);
        printMatrix(bfsTree, vertices, initialXOfRandMatrix + 400,
initialYofRandMatrix + 250, hdc);

        SelectObject(hdc, GetStockObject(HOLLOW_BRUSH));
        SelectObject(hdc, KPen);
        depictDirectedGraph(circleCenterX, circleCenterY, circleRadius,
vertexRadius, loopRadius, angleAlpha,
                        coordinates, A, 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 < vertices; ++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);
        }

        SelectObject(hdc, OPen);
        SetDCBrushColor(hdc, RGB(255, 165, 0));

        if (state == 0) {
            double** modified = createMatrix(numVertices);
            chargeWithZero(modified, numVertices);
            for (int i = 0; i < dfsIterationAmount; ++i) {
                constructSearchMatrix(dfsTree, depthVertices[i], modified);
                depictDirectedGraph(circleCenterX, circleCenterY, circleRadius,
vertexRadius, loopRadius, angleAlpha,
                        coordinates, modified, OPen, GPen, hdc);
                Ellipse(hdc, coordinates.nx[depthVertices[i]] - vertexRadius,
coordinates.ny[depthVertices[i]] - vertexRadius,
                        coordinates.nx[depthVertices[i]] + vertexRadius,
coordinates.ny[depthVertices[i]] + vertexRadius);
            }

            for (int i = 0; i < dfsIterationAmount; i++)
            {
                wchar_t buffer[5];
                swprintf(buffer, 5, L"%d", depthVertices[i] + 1);
                TextOut(hdc, coordinates.nx[depthVertices[i]] - dtx,
coordinates.ny[depthVertices[i]] - vertexRadius / 2, buffer, 3);
            }

            freeMatrix(modified, numVertices);

```



```

    }

    SelectObject(hdc, PPen);
    SetDCBrushColor(hdc, RGB(255, 0, 255));

    if (state == 1) {
        double** modified = createMatrix(numVertices);
        chargeWithZero(modified, numVertices);

        for (int i = 0; i < bfsIterationAmount; ++i) {
            constructSearchMatrix(bfsTree, queue[i], modified);
            depictDirectedGraph(circleCenterX, circleCenterY, circleRadius,
vertexRadius, loopRadius, angleAlpha,
                                coordinates, modified, BPen, GPen, hdc);

            Ellipse(hdc, coordinates.nx[queue[i]] - vertexRadius,
coordinates.ny[queue[i]] - vertexRadius,
                                coordinates.nx[queue[i]] + vertexRadius,
coordinates.ny[queue[i]] + vertexRadius);
        }

        for (int i = 0; i < bfsIterationAmount; i++)
        {
            wchar_t buffer[5];
            swprintf(buffer, 5, L"%d", queue[i] + 1);
            TextOut(hdc, coordinates.nx[queue[i]] - dtx,
coordinates.ny[queue[i]] - vertexRadius / 2, buffer, 3);
        }
        freeMatrix(modified, numVertices);
    }

    EndPaint(hWnd, &ps);

    freeMatrix(A, vertices);
    free(queue);
    free(depthVertices);
    free(visitedVertex);
    freeMatrix(dfsDetour, vertices);
    freeMatrix(bfsDetour, vertices);
    freeMatrix(dfsTree, vertices);
    freeMatrix(bfsTree, vertices);

    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return (DefWindowProc(hWnd, messg, wParam, lParam));
}
return 0;
}

```

Файл *components.c* :

```

#include <stdio.h>
#include <stdlib.h>
#define vertices 12

void typeMatrix(double **matrix) {

```

```

const int number = vertices;
for (int i = 0; i < number; i++) {
    for (int j = 0; j < number; j++) {
        printf("%.01f ", matrix[i][j]);
    }
    printf("\n");
}
}

double **randm(int n) {
    srand(2220);
    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** createMatrix(int n) {
    double **matrix = (double **) malloc(n * sizeof(double *));
    for (int i = 0; i < n; i++) {
        matrix[i] = (double *) malloc(n * sizeof(double));
    }
    return matrix;
}

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

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 depthFirstSearch(double** adjacencyMatrix, int currentVertex, int*
visited, int* depthVertices, double** tree, int* numVisited) {
    const int number = vertices;
    visited[currentVertex] = 1;
    depthVertices[(*numVisited)] = currentVertex;
    (*numVisited)++;

    for (int neighborVertex = 0; neighborVertex < number; ++neighborVertex) {
        if (adjacencyMatrix[currentVertex][neighborVertex] == 1 &&
!visited[neighborVertex]) {

            tree[currentVertex][neighborVertex] = 1;
            depthFirstSearch(adjacencyMatrix, neighborVertex, visited,
depthVertices, tree, numVisited);

        }
    }
}

void breadthFirstSearch(double** adjacencyMatrix, int startVertex, int*
queue, double** tree) {
    const int number = vertices;
    int visited[number];
    for (int i = 0; i < number; i++)
    {
        visited[i] = 0;
    }
    int queueStart = 0;
    int queueFinish = -1;

    queue[++queueFinish] = startVertex;
    visited[startVertex] = 1;

    while (queueStart <= queueFinish) {
        int currentVertex = queue[queueStart++];
        for (int neighborVertex = 0; neighborVertex < number; neighborVertex++) {
            if (adjacencyMatrix[currentVertex][neighborVertex] == 1 &&
visited[neighborVertex] == 0)
            {
                tree[currentVertex][neighborVertex] = 1;
                queue[++queueFinish] = neighborVertex;
                visited[neighborVertex] = 1;
            }
        }
    }
}

int findFirstArch(double** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            double forwardArch = matrix[i][j];
            double backwardArch = matrix[j][i];
            if (forwardArch == 1 && backwardArch == 1) {
                return i;
            }
        }
    }
    return -1;
}

void constructSearchMatrix(double** graph, int sourceVertex, double**
searchMatrix) {
    const int number = vertices;

```

```

    for (int i = 0; i < number; ++i) {
        if (graph[i][sourceVertex]) searchMatrix[i][sourceVertex] = 1;
    }
}

double** createTraversalMatrix(const int* arr) {

    double** traversalMatrix = createMatrix(vertices);
    chargeWithZero(traversalMatrix, vertices);

    for (int i = 0; i < vertices; i++)
    {
        traversalMatrix[arr[i]][i] = 1.0;
    }

    return traversalMatrix;
}

void runDfsForNotVisitedVertices(int currentVertex, double** adjacencyMatrix,
                                int* visited, int amount, int*
                                depthVertices, double** graph ) {

    const int number = vertices;

    for (int i = 0; i < number; ++i) {
        visited[i] = 0;
    }

    for (int i = 0; i < number; ++i) {
        if (visited[i] == 0) {
            depthFirstSearch(adjacencyMatrix, currentVertex, visited,
                            depthVertices, graph, &amount);
        }
    }
}

```

Матриця суміжності

I n i t i a l M a t r i x

```

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

```

Пошук у глибину

Матриця дерева обходу, матриця відповідності, зображення дерева :

D F S T r e e

```

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

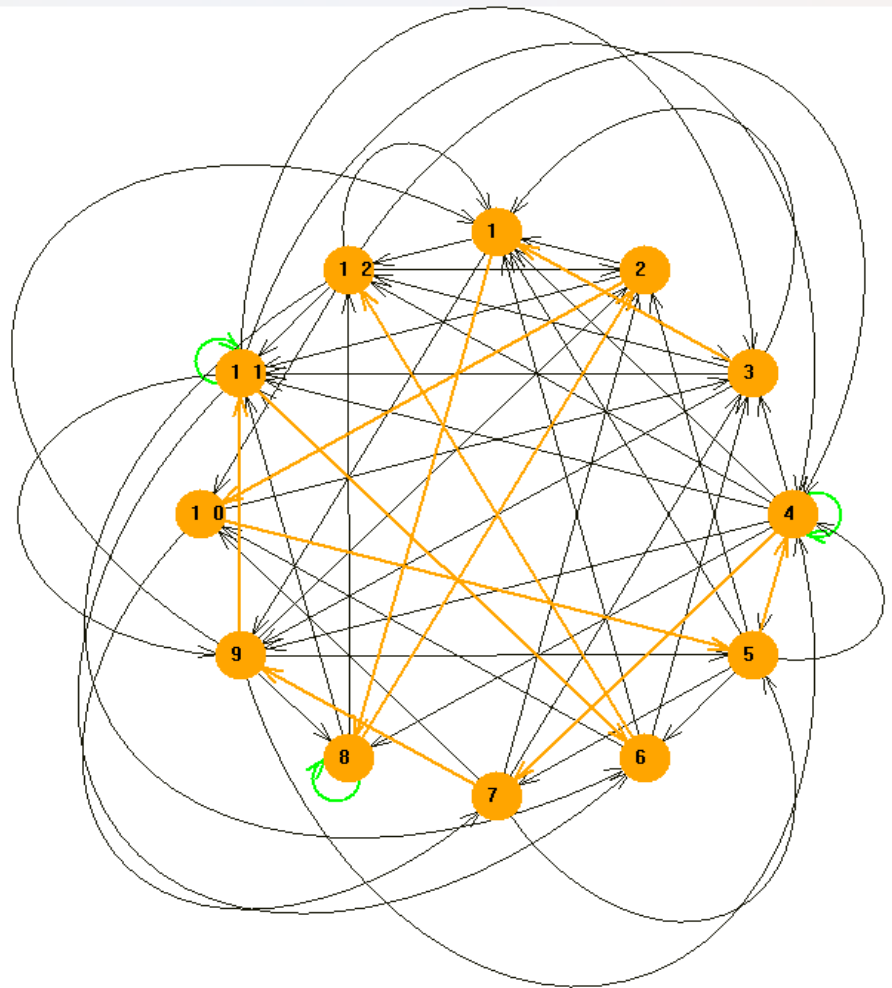
```

D F S R e l a t i v i t y

```

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

```

*Пошук у ширину*

Матриця дерева обходу, матриця відповідності, зображення дерева :

B F S T r e e

```

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

```

B F S R e l a t i v i t y

```

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

```

