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 №6

з дисципліни

«Алгоритми і структури даних»

Виконала:

студентка групи ІП-05

Лавринович Марія Юріївна

номер у списку групи: **15**

Перевірила:

Сергієнко А. А.

Київ 2021

Лабораторна робота 6.

Мінімальний кістяк графа Мета лабораторної роботи Метою лабораторної роботи №6. «Мінімальний кістяк графа» є вивчення методів розв'язання задачі знаходження мінімального кістяка графа.

```
#include <windows.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#define N 10
#define PI 3.14159265


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

wchar_t  ProgName[] = L"Lab 6";

struct stack {
    int array[N];
    int top;
};

struct stack* initStack() {
    struct stack* stack = malloc(sizeof(struct stack));
    stack->top = 0;
    return stack;
}

void pushStack(struct stack* stack, int value) {
    if (stack->top < N) {
        stack->array[stack->top] = value;
        stack->top++;
    }
}

void flush(struct stack* stack) {
```

```
    stack->top--;
}

int top(struct stack* stack) {
    if (stack->top > 0)
        return stack->array[stack->top - 1];
    else return -1;
}

int isEmptyStack(struct stack* stack) {
    if (stack->top == 0)
        return 1;
    else
        return 0;
}

int DFS(int A[N][N], int start, int end) {//depth
    int treeMatrix[N][N] = { 0 };
    struct stack* s = initStack();
    int visited[N] = { 0 };
    int curVertex;
    pushStack(s, start);
    visited[start] = 1;
    while (!isEmptyStack(s)) {
        curVertex = top(s);
        for (int i = 0; i < N; i++) {
            if (A[curVertex][i]) {
                if (i == end) return 1;
                if (visited[i] == 0) {
                    visited[i] = 1;
                    treeMatrix[curVertex][i] = 1;
                    pushStack(s, i);
                    break;
                }
            }
            if (i == N - 1) {
                flush(s);
```

```
            }
        }
    }
    return 0;
}


int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpszCmdLine, int nCmdShow)
{
    HWND hWnd;
    MSG lpMsg;
    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 = LTGRAY_BRUSH;
    w.style = CS_HREDRAW | CS_VREDRAW;
    w.cbClsExtra = 0;
    w.cbWndExtra = 0;


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


    hWnd = CreateWindow(ProgName,
        L"LAB 6. Lavrynovych M.",
        WS_OVERLAPPEDWINDOW,
        100,
        100,
        1200,
        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);
}


void drawEdge(HDC hdc, int B[N][N], int xPos[N], int yPos[N], int
start, int end) {
    int xDif = xPos[start] - xPos[end];
    int yDif = yPos[start] - yPos[end];
    char text[5];
    sprintf_s(text, sizeof(text), "%d", B[start][end]);
    if (B[start][end]) {
        if (start == end) {//LOOP check
            MoveToEx(hdc, xPos[end], yPos[end], NULL);
            LineTo(hdc, xPos[end] + 40, yPos[end] + 10);
            LineTo(hdc, xPos[end] + 40, yPos[end] + 40);
            LineTo(hdc, xPos[end] + 10, yPos[end] + 40);
            LineTo(hdc, xPos[end], yPos[end]);
            TextOutA(hdc, xPos[end] + 50, yPos[end] + 50, text,
B[start][end] < 100 ? 2 : 3);
        }
        else {
            MoveToEx(hdc, xPos[start], yPos[start], NULL);
            if (yDif == 0 && abs(xDif) > 300 && end <= 3) {//check on
horizntal obstacle
                LineTo(hdc, xPos[end] + xDif / 2, yPos[end] - 35);
                LineTo(hdc, xPos[end], yPos[end]);
                TextOutA(hdc, xPos[end] + xDif / 2, yPos[end] - 30,
text, B[start][end] < 100 ? 2 : 3);
            }
            else if (abs(xDif) == 300 && abs(yDif) == 300 && (start ==
0 || start == 3)) {//check on diagonal obstacle
                LineTo(hdc, xPos[end] + xDif / 2, yPos[end]);
```

```
                LineTo(hdc, xPos[end], yPos[end]);
                TextOutA(hdc, xPos[end] + xDif / 2 + 20, yPos[end],
text, B[start][end] < 100 ? 2 : 3);
            }
            else if (abs(xDif) == 300 && abs(yDif) == 300 && ((start
== 6 && end == 0) || (start == 7 && end == 3))) {//check on diagonal
obstacle
                LineTo(hdc, xPos[end] + xDif / 2, yPos[start]);
                LineTo(hdc, xPos[end], yPos[end]);
                TextOutA(hdc, xPos[end] + xDif / 2 + 20, yPos[start],
text, B[start][end] < 100 ? 2 : 3);
            }
            else if (abs(xDif) == 300 && abs(yDif) == 300 && ((start
== 6 && end != 2) || (start == 7 && end != 1))) {//check on diagonal
obstacle
            }
            else {
                LineTo(hdc, xPos[end], yPos[end]);
                TextOutA(hdc, xPos[end] + xDif / 2 - 20, yPos[end] +
yDif / 2, text, B[start][end] < 100 ? 2 : 3);
            }
        }
    }
}


void drawVertex(HDC hdc, int xPos[N], int yPos[N], char*
ellipseName[N], int i) {
    int dtx = 5, radius = 16;
    Ellipse(hdc, xPos[i] - radius, yPos[i] - radius, xPos[i] + radius,
yPos[i] + radius);
    TextOut(hdc, xPos[i] - dtx, yPos[i] - 8, ellipseName[i], 1);
}


void printMatrix(HDC hdc, int A[N][N]) {
    char text[11];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            sprintf_s(text, sizeof(text), "%d", A[i][j]);
            TextOutA(hdc, 5 + 30 * j, 400 + 20 * i, text, A[i][j] <
100 ? 2 : 3);
```

```
        }
    }
}

//MATRIX SIMETRICAL
void simMatrix(int A[N][N], int* B[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (A[i][j]) {
                B[j][i] = A[i][j];
                B[i][j] = A[i][j];
            }
        }
    }
}

//GENERATE MATRIX
void mulmr(int* matrix[N][N], float k) {
    int element;
    float num;
    srand(0516);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            num = (rand() / (float)RAND_MAX * 2) * k;
            if (num < 1) element = 0;
            else element = 1;
            matrix[i][j] = element;
        }
    }
}

//GENERATE MATRIX
void weightMatrix(HDC hdc, int A[N][N], int* W[N][N]) {
    int num;
    int Wt[N][N];
    int B[N][N];
    int C[N][N];
```

```
    int D[N][N];
    //Wt + B
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            num = roundf((rand() / (float)RAND_MAX * 2) * 100) *
A[i][j];
            Wt[i][j] = num;
            if (num == 0) B[i][j] = 0;
            else B[i][j] = 1;
        }
    }
    //C+D
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (B[i][j] != B[j][i]) C[i][j] = 1;
            else C[i][j] = 0;

            if (B[i][j] == B[j][i] && B[i][j] == 1 && j <= i) D[i][j]
= 1;
            else D[i][j] = 0;
        }
    }
    //ResultMatrix
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            Wt[i][j] = (C[i][j] + D[i][j]) * Wt[i][j];
        }
    }
    simMatrix(Wt, W);
}

int cycle(int start, int end, int visited[N], int A[N][N])
{
    if (start == end) return 1;
    if (visited[start] && visited[end]) return 1;
    return 0;
}
```

```
void kruskal(HDC hdc, int A[N][N], int xPos[N], int yPos[N], char*
ellipseName[N]) {
    int visitedEdges = 0, totalWeight = 0, start, end;
    char text[5];
    int treeMatrix[N][N] = { 0 };
    int visitedVertex[N] = { 0 };
    while (visitedEdges < N - 1) {
        int num = 200;
        for (int i = 0; i < N; i++) {
            for (int j = i; j < N; j++) {
                if (A[i][j] && A[i][j] < num) {
                    num = A[i][j];
                    start = i;
                    end = j;
                }
            }
        }
        if (cycle(start, end, visitedVertex, treeMatrix) &&
DFS(treeMatrix, start, end)) {
            A[start][end] = 0;
        }
        else {
            sprintf_s(text, sizeof(text), "%d", num);
            TextOutA(hdc, 5, 40, text, 2);
            drawEdge(hdc, A, xPos, yPos, start, end);
            drawVertex(hdc, xPos, yPos, ellipseName, start);
            drawVertex(hdc, xPos, yPos, ellipseName, end);
            totalWeight += num;
            treeMatrix[start][end] = num;
            treeMatrix[end][start] = num;
            A[start][end] = 0;
            visitedVertex[start] = 1;
            visitedVertex[end] = 1;
            visitedEdges++;
            system("pause");
            system("cls");
```

```
        }
    }
    sprintf_s(text, sizeof(text), "%d", totalWeight);
    TextOutA(hdc, 5, 40, text, 3);
    printMatrix(hdc, treeMatrix);
}


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


    switch (messg) {
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        int A[N][N];
        int W[N][N] = { 0 };
        int B[N][N] = {
            {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},
            {0,0,0,0,0,0,0,0,0,0},
            {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}
        };
        char* ellipseName[10] = { "0", "1", "2", "3", "4", "5", "6",
"7", "8", "9" };
        int xPos[10];
        int yPos[10];
        int dtx = 5, radius = 16, startX = 100, divine = -1, xDif,
yDif, koef, dx, dy;
        HPEN BluePen = CreatePen(PS_SOLID, 2, RGB(50, 0, 255));
        HPEN BlackPen = CreatePen(PS_SOLID, 1, RGB(20, 20, 5));
        HPEN RedPen = CreatePen(PS_SOLID, 2, RGB(250, 0, 0));
```

```
SelectObject(hdc, BlackPen);


mulmr(A, (1.0 - 1 * 0.01 - 5 * 0.005 - 0.05));
simMatrix(A, B);
weightMatrix(hdc, A, W);
printMatrix(hdc, W);

//vertex coords
int R = 250;
int step = 0;
int Centre1_X = 300, Centre1_Y = 300;
xPos[9] = Centre1_X;
yPos[9] = Centre1_Y;
for (int vertex = 0; vertex < N - 1; vertex++) {
    xPos[vertex] = Centre1_X + R * cos(step * PI / 180);
    yPos[vertex] = Centre1_Y + R * sin(step * PI / 180);
    step += 360 / N + 4;
}

//draw graph edges
for (int start = 0; start < N; start++) {
    for (int end = start; end < N; end++) {
        drawEdge(hdc, W, xPos, yPos, start, end);
    }
}
//draw vertex
SelectObject(hdc, BluePen);
for (int i = 0; i < N; i++) {
    drawVertex(hdc, xPos, yPos, ellipseName, i);
}


SelectObject(hdc, RedPen);
//algorithm of kruskal
kruskal(hdc, W, xPos, yPos, ellipseName);
```

```
        EndPaint(hWnd, &ps);
        break;

    case WM_DESTROY:
        PostQuitMessage(0);
        break;

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