Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

**по курсу «Логика и основы алгоритмизации в инженерных задачах»**

на тему «Унарные и бинарные операции над графами»

Выполнили:

студент группы 20ВВ2

Александров В.С.

Кирюткин И.А.

Сафронов Д.В.

Принял:

к.т.н., доцент

Митрохин М.А.

**Пенза 2021**

**Цель работы:** Написать программу, которая выводит матрицы смежности, выполняет операции отождествления, стягивания и расщепления для матричной и в виде списков, а также операции объединения, пересечения и кольцевой суммы.

**Лабораторные задания:**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) две матрицы M 1 ,

М 2 смежности неориентированных помеченных графов G 1 , G 2 . Выведите

сгенерированные матрицы на экран.

2. \* Для указанных графов преобразуйте представление матриц

смежности в списки смежности. Выведите полученные списки на экран.

**Задание 2**

1. Для матричной формы представления графов выполните операцию:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Результат выполнения операции выведите на экран.

2. \* Для представления графов в виде списков смежности выполните

операцию:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

**Задание 3**

1. Для матричной формы представления графов выполните операцию:

а) объединения G = G 1 

G 2

б) пересечения G = G 1 

G 2

в) кольцевой суммы G = G 1  G 2

Результат выполнения операции выведите на экран.

**Задание 4 \***

1. Для матричной формы представления графов выполните операцию

декартова произведения графов G = G 1 X G 2 .

Результат выполнения операции выведите на экран.

**Листинг основной части программы:**

#include <iostream>

#include <cstdlib>

#include <malloc.h>

using namespace std;

void iden(int\*\* M1, int\*\* M3, int N, int n, int m)

{

int MAS[100];

if (n > m) {

int temp;

temp = m;

m = n;

n = temp;

}

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

M3[i][j] = M1[i][j];

}

}

int k = 0;

for (int i = 0; i < N; i++)

{

if (i != m) {

MAS[k] = M1[i][m];

k++;

}

}

for (int i = 0; i < m; i++)

{

for (int j = m + 1; j < N; j++)

{

M3[i][j - 1] = M1[i][j];

}

}

for (int i = m + 1; i < N; i++)

{

for (int j = 0; j < m; j++)

{

M3[i - 1][j] = M1[i][j];

}

}

for (int i = m + 1; i < N; i++)

{

for (int j = m + 1; j < N; j++)

{

M3[i - 1][j - 1] = M1[i][j];

}

}

for (int i = 0; i < N - 1; i++)

{

if (M3[i][n] == 0) {

M3[i][n] = MAS[i];

M3[n][i] = MAS[i];

}

}

for (int i = 0; i < N - 1; i++)

{

int j = i;

M3[i][j] = 0;

}

cout << endl;

cout << " Результат:" << endl;

cout << endl;

for (int i = 0; i < N - 1; i++)

{

for (int j = 0; j < N - 1; j++)

{

cout << " " << M3[i][j];

}cout << endl;

}cout << endl; cout << endl;

}

int main()

{

setlocale(LC\_ALL, "Rus");

int N;

cout << " Введите размер матриц" << endl;

cin >> N;

int\*\* M1; int\*\* M2; int\*\* M3; int\*\* M4;

int n, m;

M1 = (int\*\*)malloc(N \* sizeof(int));

M2 = (int\*\*)malloc(N \* sizeof(int));

M3 = (int\*\*)malloc((N - 1) \* sizeof(int));

M4 = (int\*\*)malloc((N + 1) \* sizeof(int));

for (int i = 0; i < N; i++)

{

M1[i] = (int\*)malloc(N \* sizeof(int));

M2[i] = (int\*)malloc(N \* sizeof(int));

if (i < (N - 1))M3[i] = (int\*)malloc((N - 1) \* sizeof(int));

}

for (int i = 0; i < N + 1; i++)

{

M4[i] = (int\*)malloc((N + 1) \* sizeof(int));

}

for (int i = 0; i < N + 1; i++)

{

for (int j = 0; j < N + 1; j++)

{

M4[i][j] = 0;

}

}

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

if (i != j)

{

int t = rand() % 2;

int t2 = rand() % 2;

M1[i][j] = t;

M1[j][i] = t;

M2[i][j] = t2;

M2[j][i] = t2;

}

else {

M1[i][j] = 0;

M1[j][i] = 0;

M2[i][j] = 0;

M2[j][i] = 0;

}

}

}

cout << " Матрица M1:" << endl;

cout << endl;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

cout << " " << M1[i][j];

}cout << endl;

}

cout << endl;

cout << " 1) Отождествление вершин матрицы M1" << endl;

cout << " Введите номера вершин:" << endl;

cin >> n >> m;

n--; m--;

iden(M1, M3, N, n, m);

cout << " 2) Стягивание ребра матрицы M1:" << endl;

cout << " Введите номера вершин:" << endl;

cin >> n >> m;

n--; m--;

while (M1[n][m] != 1)

{

cout << " Введите корректные номера вершин :" << endl;

cin >> n >> m;

n--;

m--;

}

iden(M1, M3, N, n, m);

cout << " 3) Расщепление вершины матрицы M1:" << endl;

cout << " Введите номер вершины:" << endl;

cin >> n;

n--;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

M4[i][j] = M1[i][j];

}

}

int mas[100][100] = { 0 };

int K = 0;

for (int i = 0; i < N; i++)

{

if (M1[n][i] == 1)

{

mas[K][i] = 1;

K++;

}

M4[n][i] = 0;

M4[i][n] = 0;

}

int T = K;

T = T / 2;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < T; j++)

{

if (mas[j][i] == 1)

{

M4[n][i] = mas[j][i];

M4[i][n] = mas[j][i];

}

}

}

for (int i = 0; i < N; i++)

{

for (int j = T; j < K; j++)

{

if (mas[j][i] == 1)

{

M4[N][i] = mas[j][i];

M4[i][N] = mas[j][i];

}

}

}

M4[n][N] = 1;

M4[N][n] = 1;

for (int i = 0; i < N + 1; i++)

{

for (int j = 0; j < N + 1; j++)

{

cout << " " << M4[i][j];

}cout << endl;

}

int m1[2][100]; int\*\* Mat1; int\*\* Mat2; int\*\* Mat3; int MAX = 0;

cout << " 2) Обьединение/Пересечение матриц Mat1 и Mat2 в матрицу Mat3" << endl;

cout << " Введите размер матрицы Mat1" << endl;

cin >> n;

cout << " Введите номера вершин матрицы Mat1 " << endl;

cout << " Пример 1 2 3 (количество вершин - равно размеру матрицы) " << endl;

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

{

cin >> m1[0][i];

if (m1[0][i] > MAX)MAX = m1[0][i];

}

Mat1 = (int\*\*)malloc(n \* sizeof(int));

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

{

Mat1[i] = (int\*)malloc(n \* sizeof(int));

}

cout << " Введите размер матрицы Mat2" << endl;

cin >> m;

cout << " Введите номера вершин матрицы Mat2 " << endl;

cout << " Пример 4 5 6 (количество вершин - равно размеру матрицы) " << endl;

for (int i = 0; i < m; i++)

{

cin >> m1[1][i];

if (m1[1][i] > MAX)MAX = m1[1][i];

}

Mat2 = (int\*\*)malloc(m \* sizeof(int));

for (int i = 0; i < m; i++)

{

Mat2[i] = (int\*)malloc(m \* sizeof(int));

}

Mat3 = (int\*\*)malloc(MAX \* sizeof(int));

for (int i = 0; i < MAX; i++)

{

Mat3[i] = (int\*)malloc(MAX \* sizeof(int));

}

cout << endl;

for (int i = 0; i < MAX; i++)

{

for (int j = 0; j < MAX; j++)

{

if (i < n && j < n)

{

int t2 = rand() % 2;

Mat1[i][j] = t2;

Mat1[j][i] = t2;

if (i == j)Mat1[j][i] = 0;

}

if (i < m && j < m)

{

int t1 = rand() % 2;

Mat2[i][j] = t1;

Mat2[j][i] = t1;

if (i == j)Mat2[j][i] = 0;

}

Mat3[i][j] = 0;

Mat3[j][i] = 0;

}

}

cout << " Матрица Mat1:" << endl;

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

{

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

{

cout << " " << Mat1[i][j];

}cout << endl;

}

cout << endl;

cout << " Матрица Mat2:" << endl;

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

cout << " " << Mat2[i][j];

}cout << endl;

}cout << endl;

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

{

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

{

int t = m1[0][i] - 1;

Mat3[t][j] = Mat1[i][j];

}

}

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

int t = m1[1][i] - 1;

if (Mat3[t][j] == 0)

{

Mat3[t][j] = Mat2[i][j];

}

}

}

cout << "Объединение матриц\n";

for (int i = 0; i < MAX; i++)

{

for (int j = 0; j < MAX; j++)

{

cout << " " << Mat3[i][j];

}cout << endl;

}

for (int i = 0; i < MAX; i++)

{

for (int j = 0; j < MAX; j++)

{

Mat3[i][j] = 0;

}cout << endl;

}

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

{

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

{

int t = m1[0][i] - 1;

Mat3[t][j] = Mat1[i][j];

}

}

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

int t = m1[1][i] - 1;

if (Mat3[t][j] == 1)

{

Mat3[t][j] = Mat2[i][j];

}

}

}

cout << "Пересечение матриц\n";

for (int i = 0; i < MAX; i++)

{

for (int j = 0; j < MAX; j++)

{

cout << " " << Mat3[i][j];

}cout << endl;

}

for (int i = 0; i < MAX; i++)

{

for (int j = 0; j < MAX; j++)

{

Mat3[i][j] = 0;

}cout << endl;

}

if (n > m) {

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

{

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

{

Mat3[i][j] = Mat1[i][j];

if (j < m && i < m && Mat1[i][j] != Mat2[i][j]) {

Mat3[i][j] = 1;

}

if (j < m && i < m && Mat1[i][j] == Mat2[i][j]) {

Mat3[i][j] = 0;

}

}

}

}

else {

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

Mat3[i][j] = Mat2[i][j];

if (j < n && i < n && Mat1[i][j] != Mat2[i][j]) {

Mat3[i][j] = 1;

}

if (j < n && i < n && Mat1[i][j] == Mat2[i][j]) {

Mat3[i][j] = 0;

}

}

}

}

cout << "Кольцевая сумма\n";

for (int i = 0; i < MAX; i++)

{

for (int j = 0; j < MAX; j++)

{

cout << " " << Mat3[i][j];

}cout << endl;

}

return 0;

}

**Листинг декартового произведения**

#include <cstdlib>

#include <malloc.h>

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

srand(time(0));

int\*\* M1; int\*\* M2; int\*\* M3;

int n, m;

cout << "Введите размер матриц M1 и M2" << endl;

cin >> n >> m;

M1 = (int\*\*)malloc(n \* sizeof(int));

M2 = (int\*\*)malloc(m \* sizeof(int));

M3 = (int\*\*)malloc((m \* n) \* sizeof(int));

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

{

M1[i] = (int\*)malloc(n \* sizeof(int));

}

for (int i = 0; i < m; i++)

{

M2[i] = (int\*)malloc(m \* sizeof(int));

}

for (int i = 0; i < m \* n; i++)

{

M3[i] = (int\*)malloc((m \* n) \* sizeof(int));

}

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

{

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

{

int temp = rand() % 2;

M1[i][j] = temp;

M1[j][i] = temp;

if (i == j)M1[i][j] = 0;

}

}

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

int temp = rand() % 2;

M2[i][j] = temp;

M2[j][i] = temp;

if (i == j)M2[i][j] = 0;

}

}

int mas[100][2];

int k = 0, l = 0;

for (int i = 0; i < n \* m; i++)

{

if (i % n == 0)

{

k++; l = 0;

}

mas[i][0] = k - 1; // n

mas[i][1] = l; // m

l++;

}

for (int i = 0; i < n \* m; i++)

{

for (int j = 0; j < 2; j++)

{

cout << " " << mas[i][j];

}cout << endl;

}

cout << "M1:" << endl;

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

{

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

{

cout << " " << M1[i][j];

}cout << endl;

}

cout << "M2:" << endl;

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

cout << " " << M2[i][j];

}cout << endl;

}

int a1, a2, b1, b2;

for (int i = 0; i < m \* n; i++)

{

for (int j = 0; j < m \* n; j++)

{

a1 = mas[i][1]; // n

b1 = mas[i][0]; // n a1 b1

a2 = mas[j][1]; //

b2 = mas[j][0]; // a2 b2

if (a1 == a2 && b1 != b2)

{

M3[i][j] = M2[b1][b2];

}

else {

if (b2 == b1 && a1 != a2)

{

M3[i][j] = M1[a1][a2];

}

else {

if (b2 == b1 && a1 == a2)

{

M3[i][j] = 0;

}

else {

if (b2 != b1 && a1 != a2)

{

M3[i][j] = 0;

}

}

}

}

}

}

cout << "M3:" << endl;

for (int i = 0; i < m \* n; i++)

{

for (int j = 0; j < m \* n; j++)

{

cout << " " << M3[i][j];

}cout << endl;

}

}

**Листинг дополнительной части программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <ctime>

#include <cmath>

#include <stack>

#include <queue>

#include <vector>

using namespace std;

struct node

{

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph

{

int numVertices;

int\* visited;

struct node\*\* adjLists;

};

struct Graph\* createGraph(int);

void addEdge(struct Graph\*, int, int);

void printGraph(struct Graph\*);

void deleteG(struct Graph\* graph, int ver\_OTJ, int ver\_OTJ1);

void rassh(struct Graph\* graph, int ver\_OTJ);

void sryg(struct Graph\* graph, int ver\_OTJ, int ver\_OTJ1);

int main()

{

setlocale(LC\_ALL, "Rus");

void styg(struct Graph\* graph, int ver\_OTJ, int ver\_OTJ1);

int versh, \* a, i, ver, conect;

int start, j;

int size, \*\* graphM, n;

struct Graph\* graph = createGraph(5);

addEdge(graph, 0, 1);

addEdge(graph, 2, 0);

addEdge(graph, 0, 3);

addEdge(graph, 2, 1);

addEdge(graph, 2, 4);

/\*cout << "Введите размерность: " << endl;

cin >> size;

srand(time(NULL));

graphM = (int\*\*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

{

graphM[i] = (int\*)malloc(size \* sizeof(int));

}

bool\* visited = new bool[size];

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

for (int j = 0; j < i + 1; j++) {

int num = rand() % 2;

graphM[i][j] = num;

graphM[j][i] = num;

}

}

cout << "Матрица смежности графа: " << endl;

for (i = 0; i < size; i++)

{

visited[i] = false;

for (j = 0; j < size; j++)

cout << " " << graphM[i][j];

cout << endl;

}

int tmp;

for (i = 0; i < size; i++)

{

for ( j = 0; j < size; j++)

{

if (graphM[i][j] == 0)

{

addEdge(graph, i, j);

}

}

}\*/

int ras, st1, st2, ot1, ot2;

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nИзначальный граф");

printGraph(graph);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nПосле стягивания ребера\n введите вершины");

scanf\_s("%d %d", &st1, &st2);

sryg(graph, st1, st2);

printGraph(graph);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nПосле отождествления вершин\n введите вершины " );

scanf\_s("%d %d", &ot1, &ot2);

deleteG(graph, ot1, ot2);

printGraph(graph);

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n расщепление вершины\n введите вершину");

scanf\_s("%d", &ras);

rassh(graph, ras);

printGraph(graph);

return 0;

}

struct node\* createNode(int v)

{

struct node\* newNode = (struct node\*)malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

//graph->adjLists = (struct Graph\*)malloc(vertices \* sizeof(struct node\*));

graph->adjLists = (struct node\*\*)malloc(vertices \* sizeof(struct node\*));

graph->visited = (int\*)malloc(vertices \* sizeof(int));

int i;

for (i = 0; i < vertices; i++) {

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph)

{

int v;

for (v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\n Список смежности вершины %d\n ", v);

while (temp)

{

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

void deleteG(struct Graph\* graph, int ver\_OTJ, int ver\_OTJ1)

{

int v = 0;

struct node\* temp = graph->adjLists[ver\_OTJ];

struct node\* tempO = graph->adjLists[ver\_OTJ1];

for (v = 0; v < 5; v++) {

temp = graph->adjLists[v];

while (temp != NULL)

{

if (temp->vertex == ver\_OTJ1 && v!= ver\_OTJ)

{

temp->vertex = ver\_OTJ;

}

if (temp->vertex == ver\_OTJ1 && v == ver\_OTJ)

{

while (temp->next->next != NULL)

{

temp->vertex = temp->next->vertex;

temp = temp->next;

}

temp->vertex = temp->next->vertex;

free(temp->next);

temp->next = NULL;

delete temp->next;

}

temp = temp->next;

}

}

tempO = graph->adjLists[ver\_OTJ1];

temp = graph->adjLists[ver\_OTJ];

while (tempO != NULL)

{

struct node\* newNode = createNode(tempO->vertex);

newNode->next = graph->adjLists[ver\_OTJ];

graph->adjLists[ver\_OTJ] = newNode;

tempO = tempO->next;

}

graph->adjLists[ver\_OTJ1] = NULL;

free(graph->adjLists[ver\_OTJ1]);

delete graph->adjLists[ver\_OTJ1];

}

void sryg(struct Graph\* graph, int ver\_OTJ, int ver\_OTJ1)

{

int v = 0, drop=0;

struct node\* temp = graph->adjLists[ver\_OTJ];

struct node\* tempO = graph->adjLists[ver\_OTJ1];

while (temp->next != NULL)

{

if (temp->vertex == ver\_OTJ1) {

drop = -1;

}

temp = temp->next;

}

if (drop ==-1)

{

for (v = 0; v < 5; v++) {

temp = graph->adjLists[v];

while (temp != NULL)

{

if (temp->vertex == ver\_OTJ1 && v != ver\_OTJ)

{

temp->vertex = ver\_OTJ;

}

if (temp->vertex == ver\_OTJ1 && v == ver\_OTJ)

{

while (temp->next->next != NULL)

{

temp->vertex = temp->next->vertex;

temp = temp->next;

}

temp->vertex = temp->next->vertex;

free(temp->next);

temp->next = NULL;

delete temp->next;

}

temp = temp->next;

}

}

tempO = graph->adjLists[ver\_OTJ1];

temp = graph->adjLists[ver\_OTJ];

while (tempO != NULL)

{

struct node\* newNode = createNode(tempO->vertex);

newNode->next = graph->adjLists[ver\_OTJ];

graph->adjLists[ver\_OTJ] = newNode;

tempO = tempO->next;

}

graph->adjLists[ver\_OTJ1] = NULL;

free(graph->adjLists[ver\_OTJ1]);

delete graph->adjLists[ver\_OTJ1];

struct node\* temp = graph->adjLists[ver\_OTJ];

if (temp->vertex == ver\_OTJ) {

while (temp->next->next != NULL)

{

temp->vertex = temp->next->vertex;

temp = temp->next;

}

temp->vertex = temp->next->vertex;

free(temp->next);

temp->next = NULL;

delete temp->next;

}

temp = temp->next;

}

else printf("\nCтягивание не возможно\n");

}

void rassh(struct Graph\* graph, int ver\_OTJ)

{

int v = 0;

graph->numVertices = 6;

graph->adjLists[5] = NULL;

graph->visited[5] = 0;

struct node\* temp = graph->adjLists[5];

struct node\* tempO = graph->adjLists[ver\_OTJ];

while (tempO != NULL)

{

addEdge(graph, 5, tempO->vertex);

tempO= tempO->next;

}

addEdge(graph, 5, ver\_OTJ);

**Псевдокод m1:**

Вход: Mat1; Mat2; Mat3; MAX; G(v,a) – неориентированный граф

Выход: Mat3

Алгоритм m1:

Для I от 1 до n

Ввод m1[0][i]

Если m1[0][i] > MAX MAX = m1[0][i]

Для I от 1 до n

Ввод m1[1][i]

Если m1[0][i] > MAX MAX = m1[1][i]

Для I от 1 до max

Для J от 1 до max

Если i < n И j < n

t2 = rand() % 2

Mat1[i][j] = t2

Mat1[j][i] = t2

Если i = j

Mat1[j][i] = 0

Если i < m И j < m

t1 = rand() % 2

Mat2[i][j] = t1

Mat2[j][i] = t1

if i = j

Mat2[j][i] = 0

Mat3[i][j] = 0

Mat3[j][i] = 0

Вывод " Матрица Mat1:"

Для i от 0 до n

Для j от 0 до n

Вывод: Mat1[i][j];

Вывод" Матрица Mat2:"

Для i от 0 до m

Для j от 0 до m

Вывод: Mat2[i][j]

Для i от 0 до n

Для j от 0 до n

t = m1[0][i] - 1

Mat3[t][j] = Mat1[i][j]

Для i от 0 до m

Для j от 0 до m

t = m1[1][i] - 1

Если Mat3[t][j] = 0

Mat3[t][j] = Mat2[i][j]

Вывод: "Объединение матриц\n";

Для i от 0 до MAX

Для j от 0 до MAX

Вывод: Mat3[i][j]

Для i от 0 до MAX

Для j от 0 до MAX

Mat3[i][j] = 0

Для i от 0 до n

Для j от 0 до n

t = m1[0][i] - 1

Mat3[t][j] = Mat1[i][j]

Для i от 0 до m

Для j от 0 до m

t = m1[1][i] - 1

Если Mat3[t][j] = 1

Mat3[t][j] = Mat2[i][j]

Вывод "Пересечение матриц\n"

Для i от 0 до MAX

Для j от 0 до MAX

Вывод: Mat3[i][j]

Для i от 0 до MAX

Для j от 0 до MAX

Mat3[i][j] = 0

Если n>m

Для i от 0 до n

Для j от 0 до n

Mat3[i][j] = Mat1[i][j]

Если j < m И i < m И Mat1[i][j] != Mat2[i][j]

Mat3[i][j] = 1

Если j < m И i < m И Mat1[i][j] = Mat2[i][j]

Mat3[i][j] = 0

Иначе

Для i от 0 до m

Для j от 0 до m

{

Mat3[i][j] = Mat2[i][j]

Если j < n И i < n И Mat1[i][j] != Mat2[i][j]

Mat3[i][j] = 1

Если j < n И i < n И Mat1[i][j] = Mat2[i][j]

Mat3[i][j] = 0

Вывод: "Кольцевая сумма\n"

Для i от 0 до MAX

Для о от 0 до MAX

Вывод: " " Mat3[i][j]

**Псевдокод iden:**

Вход: M1, M3, N, n, m, G(v,a) – неориентированный граф

Выход: M3

Алгоритм iden:

Если n > m

temp = m

m = n

n = temp

Для I от 0 до m

Для j от 0 до m

M3[i][j] = M1[i][j]

k = 0

Для I от 0 до N

Если i != m

MAS[k] = M1[i][m]

k++

Для I от 0 до N

Для j от m+1 до N

M3[i][j - 1] = M1[i][j]

Для I от m+1 до N

Для j от 0 до m

M3[i - 1][j] = M1[i][j]

Для I от m+1 до N

Для j от m+1 до N

M3[i - 1][j - 1] = M1[i][j]

Для I от 0 до N - 1

Если M3[i][n] = 0

M3[i][n] = MAS[i]

M3[n][i] = MAS[i]

Для I от 0 до N -1

j = i

M3[i][j] = 0

Для I от 0 до N - 1

Для о от 0 до N - 1

Вывод: M3[i][j]

**Псевдокод основной части:**

Вход: N, M1, M2, M3, M4, n, m,

G(v,a) – неориентированный граф

Выход: m1, m2, m3, m4

Алгоритм main:

Для I от 0 до N

Для I от 0 до N + 1 {

Для j от 0 до N + 1

M4[i][j] = 0

Для I от 0 до N {

Для j от 0 до N

Если i != j

t = rand() % 2

t2 = rand() % 2

M1[i][j] = t

M1[j][i] = t

M2[i][j] = t2

M2[j][i] = t2

Иначе

M1[i][j] = 0

M1[j][i] = 0

M2[i][j] = 0

M2[j][i] = 0

Вывод: " Матрица M1:"

Для I от 0 до N

Для j от 0 до N

Вывод: M1[i][j]

Вывод: " Отождествление вершин матрицы M1"

Вывод: " Введите номера вершин:" << endl;

Ввод: n m

iden(M1, M3, N, n, m)

Пока M1[n][m] != 1

Вывод: " Введите корректные номера вершин :"

iden(M1, M3, N, n, m)

Ввод n

Для I от 0 до N

Для о от 0 до N

M4[i][j] = M1[i][j]

mas[100][100] = { 0 }

K = 0

Для I от 0 до N {

Если M1[n][i] = 1

mas[K][i] = 1

K++

M4[n][i] = 0

M4[i][n] = 0

T = K

T = T / 2

Для I от 0 до N

Для j от 0 до Т

Если mas[j][i] = 1

M4[n][i] = mas[j][i]

M4[i][n] = mas[j][i]

Для I от 0 до N

Для j от T до K

Если mas[j][i] = 1

M4[N][i] = mas[j][i]

M4[i][N] = mas[j][i]

M4[n][N] = 1

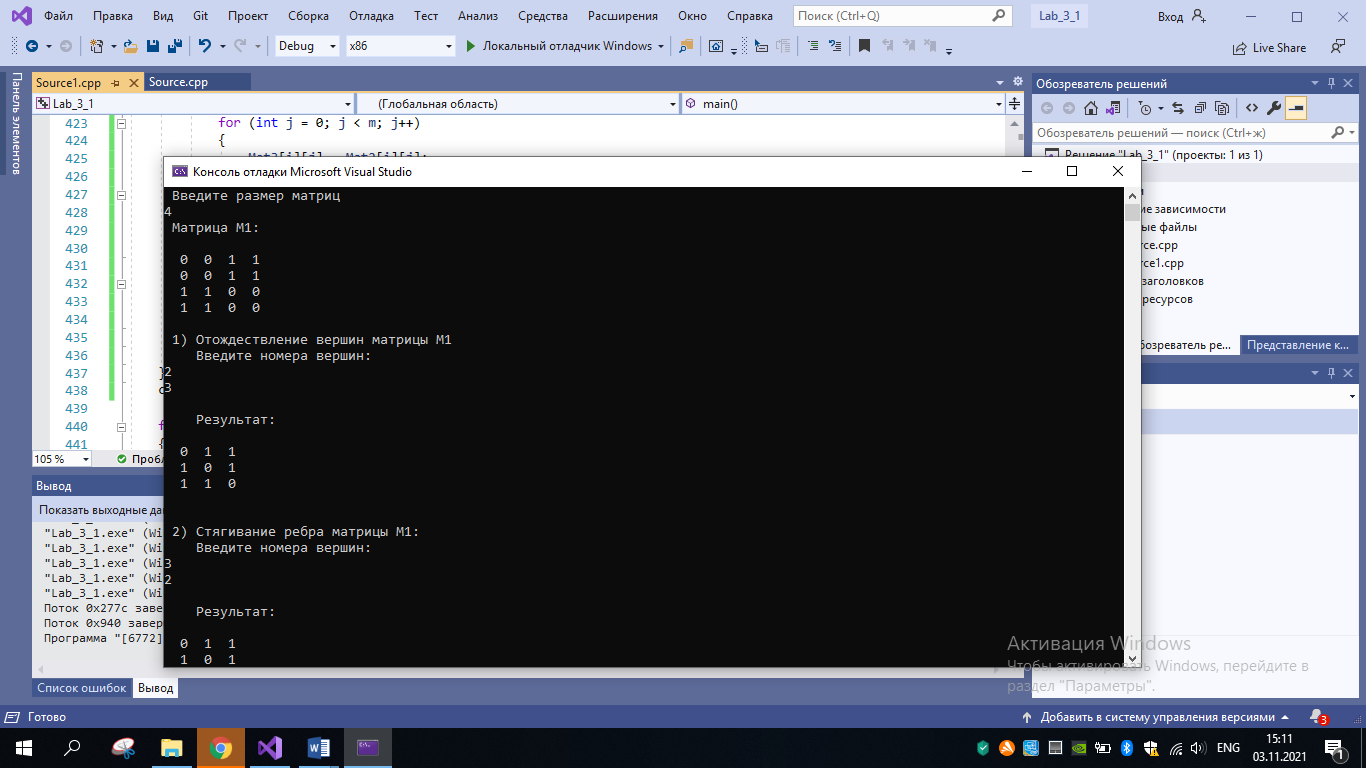
M4[N][n] = 1

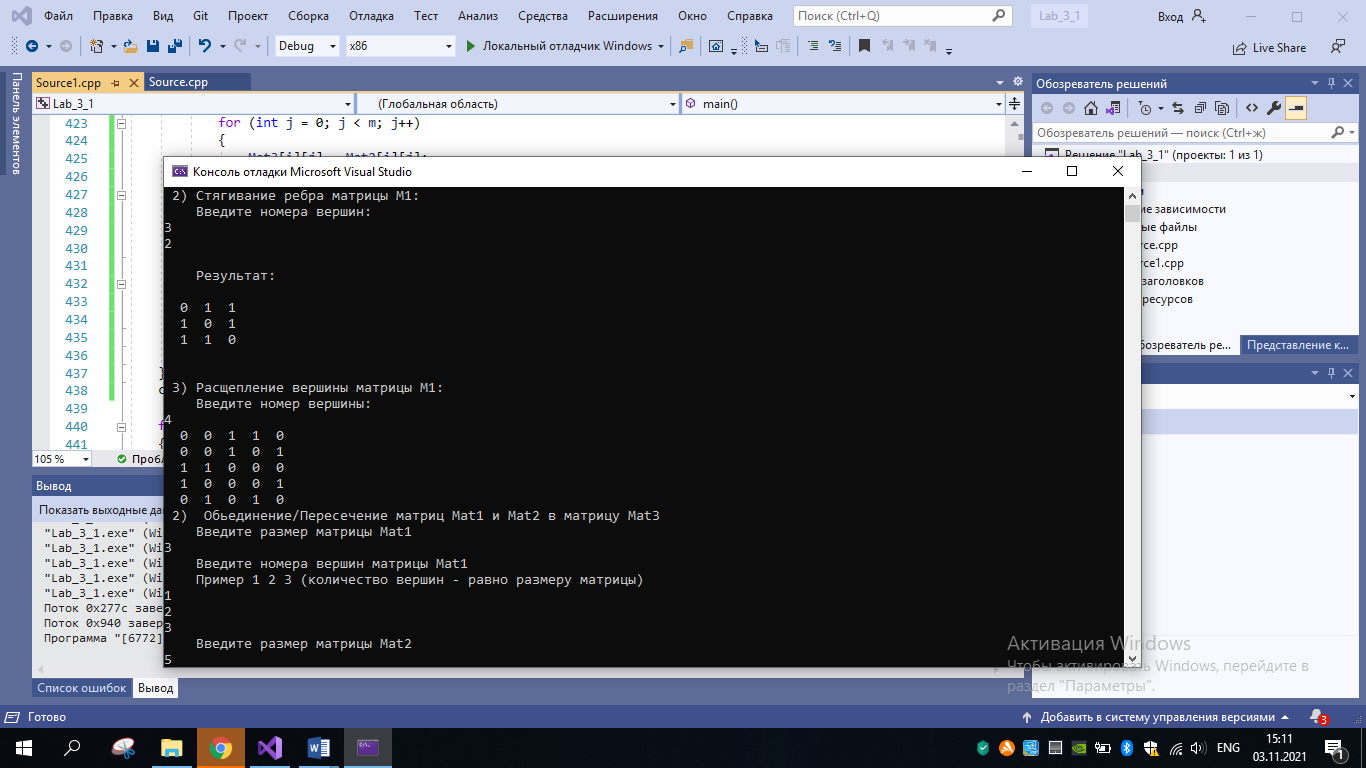
Для I от 0 до N + 1

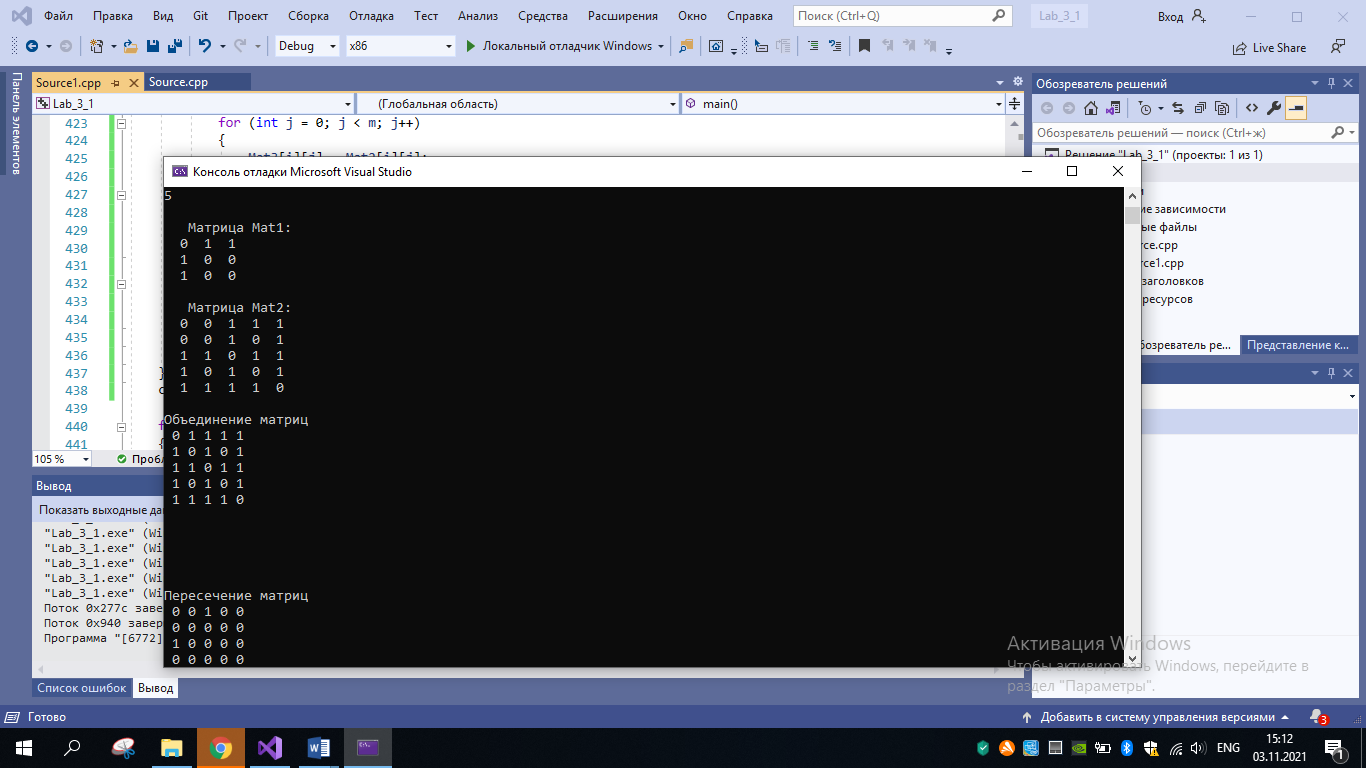
Для j от 0 до N + 1

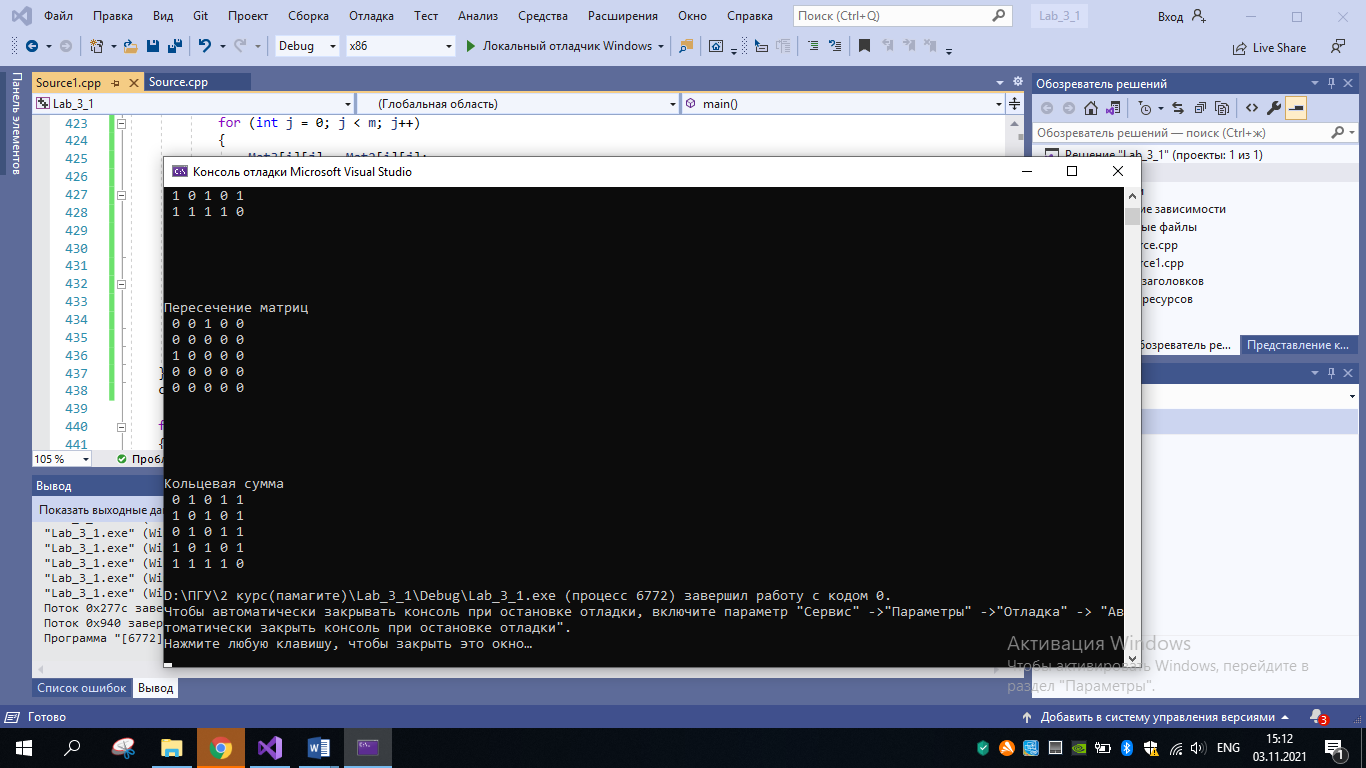
Вывод: M4[i][j]

**Результат программы Основной части :**

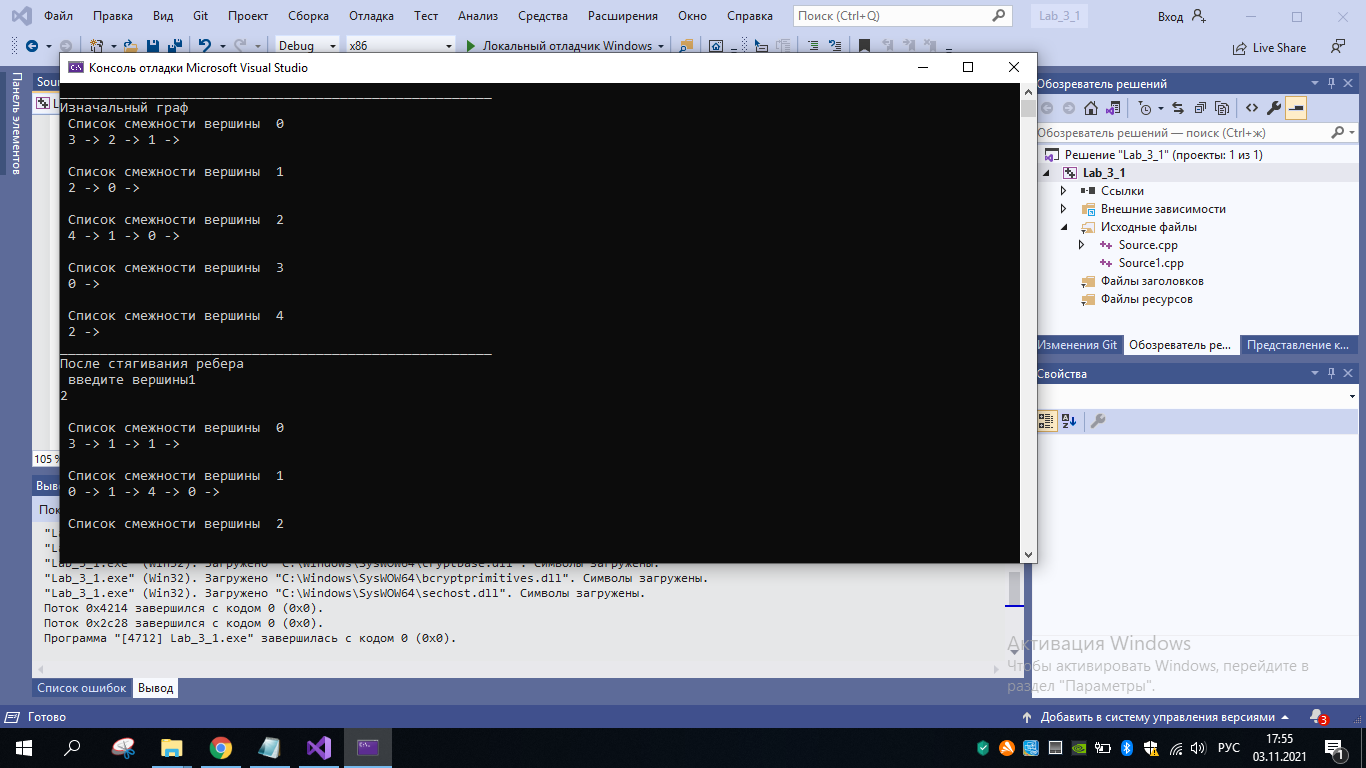


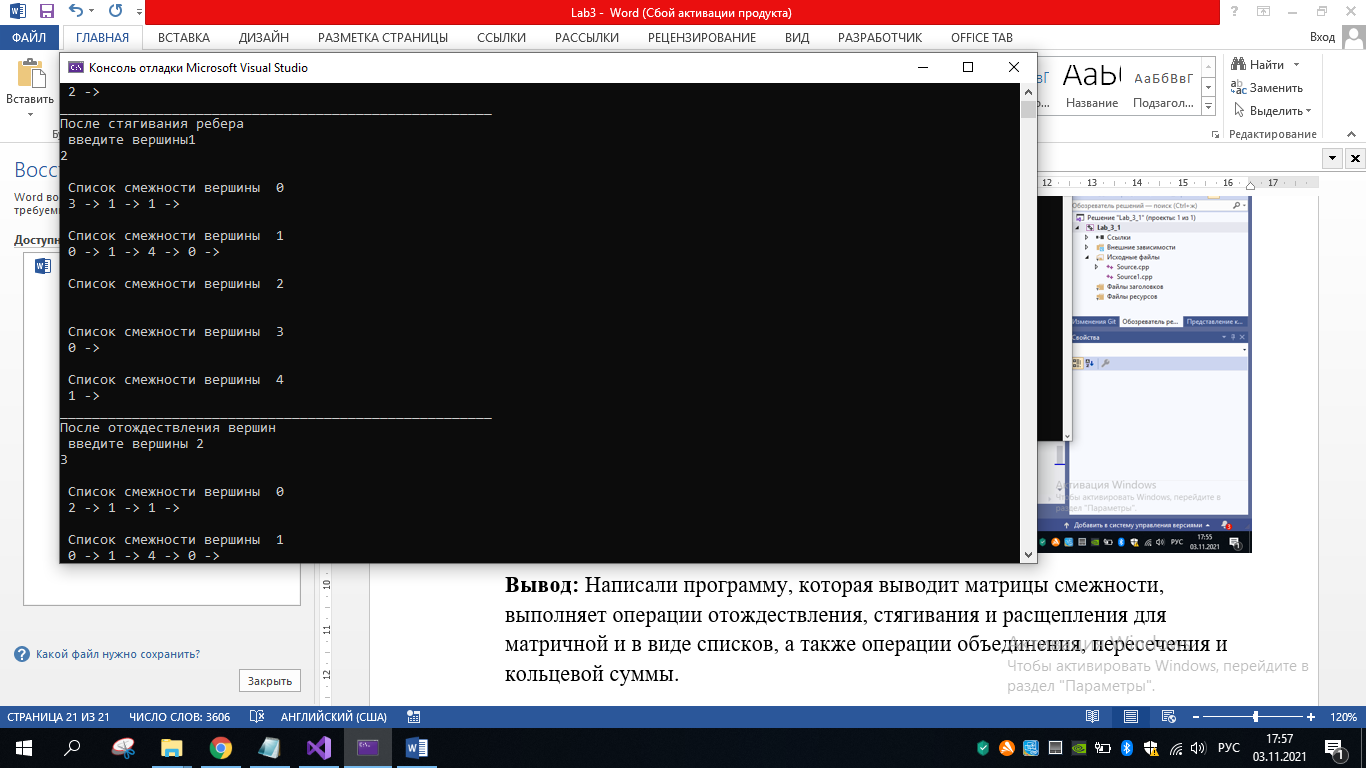


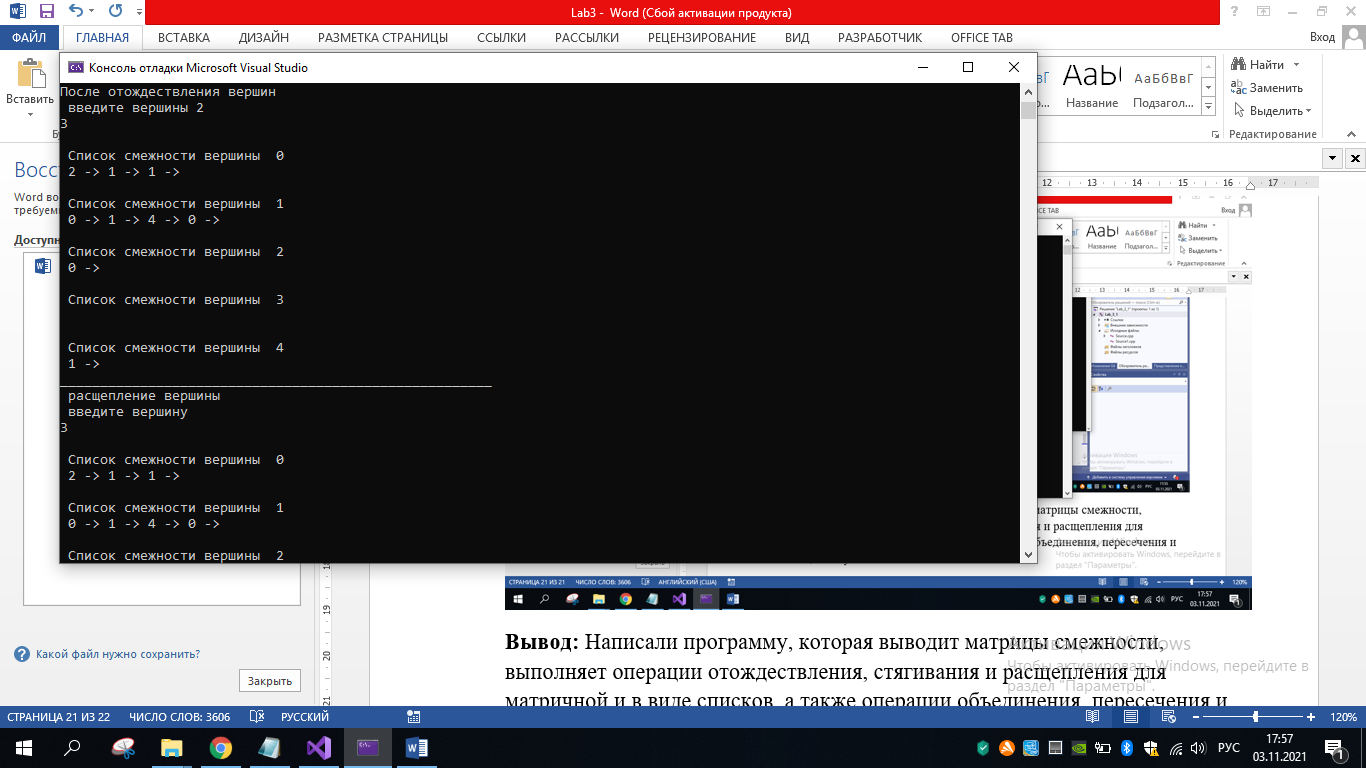


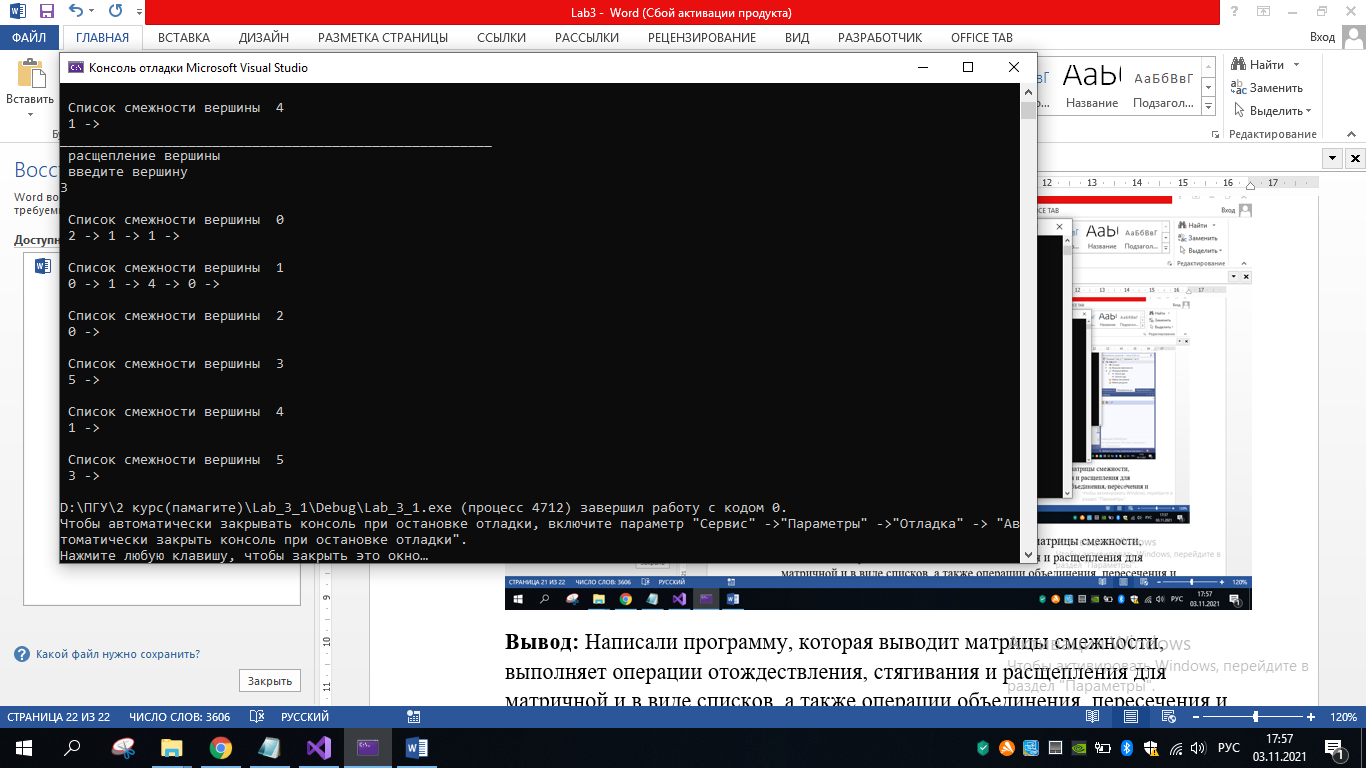


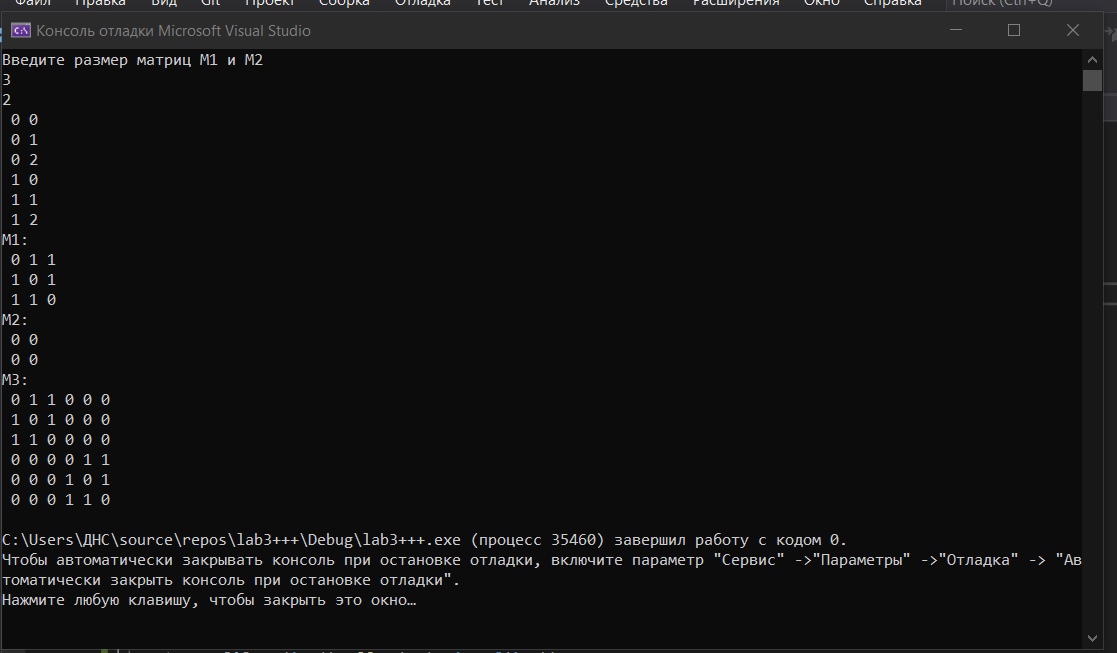
**Результат программы дополнительной части :**











Сложность работы программы:



**Вывод:** Написали программу, которая выводит матрицы смежности, выполняет операции отождествления, стягивания и расщепления для матричной и в виде списков, а также операции объединения, пересечения и кольцевой суммы.