Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнили:

студенты группы 22ВВВ2:

Зубриянова А.А.

Кондратьева В.И.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Унарные и бинарные операции над графами.

**Цель работы**

Научиться выполнять унарные и бинарные операции над графами.

**Лабораторное задание**

Задание 1

1. Сгенерируйте (используя генератор случайных чисел) две матрицы M1, М2 смежности неориентированных помеченных графов G1, G2. 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

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

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 = G1  G2

б) пересечения G = G1  G2

в) кольцевой суммы G = G1  G2

Результат выполнения операции выведите на экран.

Задание 4 \*

1. Для матричной формы представления графов выполните операцию декартова произведения графов G = G1 X G2.

Результат выполнения операции выведите на экран.

**Описание метода решения задачи**

Задание 1:

1. Сгенерировали (используя генератор случайных чисел) 2 матрицы смежности для неориентированного графа. Главную диагональ матрицы заполнили нулями. Так как граф неориентированный – матрица симметрична относительно главной диагонали.
2. Преобразовали матрицы в списки смежности, используя массив структур.

Задание 2:

1. Для матричной формы представления графов выполнили операции:

а) отождествления вершин

Для выбранных элементов матрицы выполнили операцию «или». Выполнили запись в новый массив.

б) стягивания ребра

Выполнили операцию «отождествление» и удалили петлю.

в) расщепления вершины

Выполнили запись в массив элементов, которые находились до заданной вершины, записали выбранную вершину дважды, а затем заполнили матрицу после указанной вершины. Создали ребро между расщепленными вершинами.

Номера выбираемых для выполнения операции вершин введены с клавиатуры.

Результат выполнения операции выводится на экран.

Выполнили аналогичные операции со списками смежности:

а) отождествления вершин

Создаем новую вершину, записываем в неё информацию (связи) из выбранных, удаляем выбранные.

б) стягивания ребра

Создаем новую вершину, записываем в неё информацию из выбранных, удаляем выбранные. Удаляем петлю.

в) расщепления вершины

Создаем новую вершину, переписываем в неё информацию из выбранной. Создали ребро между расщепленными вершинами.

Номера выбираемых для выполнения операции вершин введены с клавиатуры.

Результат выполнения операции выводится на экран.

Задание 3:

1. Для матричной формы представления графов выполнили операции:

а) объединения

Создаем новый массив, заполняем его нулями. Делаем операцию «или» по размеру наименьшей матрицы. Записываем в массив оставшиеся элементы из второй матрицы.

б) пересечения

Осуществляем конъюнкцию элементов двух матриц в новый массив по размеру большей матрицы.

в) кольцевой суммы

Создаем новый массив, заполняем его нулями. Осуществляем операцию «исключающее или» среди элементов наименьшей матрицы и элементов из наибольшей матрицы (в ней берется столько элементов, сколько было в наименьшей матрице).

Задание 4:

Создаем новый массив размером size1 \* size2. Создаем 4 цикла, чтобы обратиться к индексам каждой из первоначальных матриц. В новом графе вершины смежны, если первый индекс первой матрицы равен первому индексу второй матрицы и между вторыми индексами есть ребро. Аналогично, если вторые индексы матриц равны, а между первыми есть ребро. Иначе записываем 0.

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

#include <conio.h>

struct Node

{

int n;

Node\* next;

};

Node\* LA;

Node\* make\_LA(int nv, int\*\* v);

void merge\_LA(int nv, Node\* la, int del);

void split\_LA(int nv, Node\* la);

int find\_LA(Node\* p, int n);

void add\_LA(Node\* p, int n);

int del\_LA(Node\* p, int n);

void print\_LA(int nv, Node\* la);

Node\* make\_LA(int nv, int\*\* v)

{

Node\* la, \* p;

la = (Node\*)malloc(sizeof(Node) \* nv);

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

{

la[i].n = i;

la[i].next = NULL;

p = &la[i];

if (i < nv - 1)

{

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

{

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

{

p->next = (Node\*)malloc(sizeof(Node));

p = p->next;

p->n = j;

p->next = NULL;

}

}

}

}

return la;

}

void merge\_LA(int nv, Node\* la, int del)

{

int n1, n2;

if (del)

printf("Стягивание ребра\n");

else

printf("Отождествление вершин\n");

//Ввод номеров вершин 1 и 2

while (1)

{

printf("Введите номер вершины 1: ");

scanf("%d", &n1);

if ((n1 >= 0) && (n1 < nv - 1)) break;

}

while (1)

{

printf("Введите номер вершины 2: ");

scanf("%d", &n2);

if ((n2 >= 0) && (n2 < nv - 1) && (n2 != n1)) break;

}

//Удаление ребра

if (del)

{

del\_LA(&la[n1], n2);

del\_LA(&la[n2], n1);

}

//Создание списка для новой вершины

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

{

if (find\_LA(&la[n1], i) || find\_LA(&la[n2], i)) //Есть связь вершины 1 или 2 с вершиной i

{

add\_LA(&la[nv - 1], i); //Добавляем вершину i в список новой вершины

}

}

//Удаление вершины 1

la[n1].n = -1;

la[n1].next = NULL;

//Удаление вершины 2

la[n2].n = -1;

la[n2].next = NULL;

//Удаление вершин 1 и 2 в списках

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

{

int f = del\_LA(&la[i], n1) + del\_LA(&la[i], n2); //Удаление вершин 1 и 2

if (f > 0) //При удалении вершин 1 или 2

add\_LA(&la[i], nv - 1); //добавление новой вершины

}

}

void split\_LA(int nv, Node\* la)

{

int n1;

printf("Расщепление вершины\n");

//Ввод номера вершины

while (1)

{

printf("Введите номер вершины: ");

scanf("%d", &n1);

if ((n1 >= 0) && (n1 < nv - 1)) break;

}

//Создание списка для новой вершины

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

{

if (find\_LA(&la[n1], i)) //Есть связь вершины с вершиной i

{

if (n1 != i) //Не петля

add\_LA(&la[nv - 1], i); //Добавляем вершину i в список новой вершины

else //Петля

add\_LA(&la[nv - 1], nv - 1); //Добавляем петлю в список новой вершины

}

}

//Добавление ребра

add\_LA(&la[n1], nv - 1);

add\_LA(&la[nv - 1], n1);

}

int find\_LA(Node\* p, int n)

{

while (p->next != NULL)

{

if (p->next->n == n)

return 1;

p = p->next;

}

return 0;

}

void add\_LA(Node\* p, int n)

{

while (p->next != NULL)

{

p = p->next;

}

p->next = (Node\*)malloc(sizeof(Node));

p = p->next;

p->n = n;

p->next = NULL;

}

int del\_LA(Node\* p, int n)

{

while (p->next != NULL)

{

if (p->next->n == n)

{

p->next = p->next->next;

return 1;

}

p = p->next;

}

return 0;

}

void print\_LA(int nv, Node\* la)

{

Node\* p;

printf("Списки смежности:\n");

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

{

p = &la[i];

if (p->n != -1)

{

while (p != NULL)

{

printf("%d > ", p->n);

p = p->next;

}

printf("\n");

}

}

}

void print\_matrix(int size, int\*\* arr) {

printf(" ");

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

printf("%4d ", i);

}

printf("\n");

printf("-----------------------------------------------------------\n");

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

printf("%4d|", i);

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

printf("%4d ", arr[i][j]);

}

printf("\n");

}

printf("\n");

}

void peres(int\*\* a1, int\*\* a2, int\*\* a3, int si1, int si2) {

for (int i = 0; i < si1; i++) {

for (int j = 0; j < si1; j++) {

a3[i][j] = a1[i][j] && a2[i][j];

}

}

print\_matrix(si1, a3);

for (int i = 0; i < si1; ++i) free(a3[i]);

free(a3);

}

void obed(int\*\* a1, int\*\* a2, int\*\* a3, int si1, int si2) {

for (int i = 0; i < si2; i++) {

for (int j = 0; j < si2; j++) {

a3[i][j] = 0;

}

}

for (int i = 0; i < si1; i++) {

for (int j = 0; j < si1; j++) {

a3[i][j] = a1[i][j] | a2[i][j];

}

}

for (int i = 0; i < si2; i++) {

for (int j = 0; j < si2; j++) {

if ((i >= si1) && (j >= si1)) {

a3[i][j] = a2[i][j];

}

else if (i >= si1) {

a3[i][j] = a2[i][j];

}

else if (j >= si1) {

a3[i][j] = a2[i][j];

}

}

}

print\_matrix(si2, a3);

for (int i = 0; i < si2; ++i) free(a3[i]);

free(a3);

}

void otoz(int\*\* a, int\*\* arr, int size, int b1, int b2) {

for (int i = 0; i < size; i++) { //склеиваем введенные первые и вторые строку и столбец с записью в первый

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

arr[i][b1] = arr[i][b1] | arr[i][b2];

//printf("arr[%d][%d] = %d\n", i, b1, arr[i][b1]);

arr[b1][j] = arr[b1][j] | arr[b2][j];

//printf("arr[%d][%d] = %d\n", b1, j, arr[b1][j]);

}

}

for (int i = 0; i < size - 1; i++) { //заполняем новый массив значениями

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

if ((i < b2) && (j < b2)) {

a[i][j] = arr[i][j];

}

else if ((i >= b2) && (j >= b2)) {

a[i][j] = arr[i + 1][j + 1];

}

else if (i >= b2) {

a[i][j] = arr[i + 1][j];

}

else if (j >= b2) {

a[i][j] = arr[i][j + 1];

}

}

}

}

void kol\_sum(int\*\* a1, int\*\* a2, int\*\* a3, int si1, int si2) {

int g = 0;

for (int i = 0; i < si2; i++) {

for (int j = 0; j < si2; j++) {

a3[i][j] = 0;

}

}

for (int i = 0; i < si1; i++) {

for (int j = 0; j < si1; j++) {

a3[i][j] = a1[i][j] xor a2[i][j];

}

}

for (int i = 0; i < si2; i++) {

for (int j = 0; j < si2; j++) {

if ((i >= si1) && (j >= si1)) {

a3[i][j] = a2[i][j];

}

else if (i >= si1) {

a3[i][j] = a2[i][j];

}

else if (j >= si1) {

a3[i][j] = a2[i][j];

}

}

}

print\_matrix(si2, a3);

int k = 0;

int\* mass;

mass = (int\*)malloc(sizeof(int) \* si2);

for (int i = 0; i < si2; i++) {

mass[i] = 0;

}

for (int i = 0; i < si2; i++) {

for (int j = 0; j < si2; j++) {

if (a3[i][j] == 1) {

g++;

}

}

if (g == 0) {

mass[i] = 1;

k++;

printf("Вершина %d - изолирована\n", i);

}

g = 0;

}

printf("Массив изолированных вершин: \n");

for (int i = 0; i < si2; i++) {

printf("%5d", mass[i]);

}

printf("\n\n");

//printf("k = %d\n", k);

for (int i = 0; i < si2; i++)

{

if (mass[i] == 0)

{

for (int j = 0; j < si2; j++)

{

if (mass[j] == 0)

{

printf("%5d", a3[i][j]);

}

}

printf("\n");

}

}

if (k == si2)

{

printf("Самоуничтожение в результате кольцевой суммы...");

}

}

void create\_matrix(int size, int\*\* arr) {

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

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

if (i == j) { //главная диагональ == 0

arr[i][j] = 0;

}

else { //ниже главной диагонали отразить то, что выше

arr[i][j] = rand() % 2;

arr[j][i] = arr[i][j];

}

}

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "Russian");

char menu;

do {

system("cls");

printf("МЕНЮ:\n");

printf("1 - Задание 1: генерация двух матриц смежности и вывод на экран\n матрица смежности => список смежности\n\n");

printf("2 - Задание 2.1: операции отождествления вершин, стягивания ребра, расщепления вершины для матричной формы\n\n");

printf("3 - Задание 3: операции объединения, пересечeния, кольцевой суммы для матричной формы\n\n");

printf("4 - Задание 4: операция декартова произведения для матричной формы\n\n");

printf("5 - Задание 2.2: операции отождествления вершин, стягивания ребра, расщепления вершины для списков смежности\n\n");

printf("ecs - выход\n");

menu = \_getch();

switch (menu) { //1

case '1':

system("cls");

int size1, size2;

int\*\* arr1;

int\*\* arr2;

printf("Введите размер матрицы смежности M1: ");

scanf("%d", &size1);

printf("\n");

printf("Введите размер матрицы смежности M2: ");

scanf("%d", &size2);

printf("\n");

arr1 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr1[i] = (int\*)malloc(sizeof(int) \* size1);

}

arr2 = (int\*\*)malloc(sizeof(int\*) \* size2);

for (int i = 0; i < size2; i++) {

arr2[i] = (int\*)malloc(sizeof(int) \* size2);

}

printf("Матрица M1:\n");

create\_matrix(size1, arr1);

print\_matrix(size1, arr1);

printf("Матрица M2:\n");

create\_matrix(size2, arr2);

print\_matrix(size2, arr2);

printf("Список смежности S1:\n");

LA = make\_LA(size1+1, arr1);

print\_LA(size1, LA);

printf("\n");

printf("Список смежности S2:\n");

LA = make\_LA(size2+1, arr2);

print\_LA(size2, LA);

for (int i = 0; i < size1; ++i) free(arr1[i]);

free(arr1);

for (int i = 0; i < size2; ++i) free(arr2[i]);

free(arr2);

\_getch();

break;

}

switch (menu) { //2

case '2':

system("cls");

//printf("2");

int size1;

int\*\* arr1;

int\*\* arr2;

int\*\* arr3;

//int number;

printf("Введите размер матрицы смежности M1: ");

scanf("%d", &size1);

printf("\n");

arr1 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr1[i] = (int\*)malloc(sizeof(int) \* size1);

}

arr2 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr2[i] = (int\*)malloc(sizeof(int) \* size1);

}

arr3 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size1);

}

printf("Матрица M1:\n");

create\_matrix(size1, arr1);

print\_matrix(size1, arr1);

memcpy(arr2, arr1, sizeof(int) \* size1 \* size1);

memcpy(arr3, arr1, sizeof(int) \* size1 \* size1);

while (1)

{

printf("Введите Q(q), W(w), E(e) для отождествления, стягивания ребра и расщепления соответственно\n");

int k = \_getch();

if ((k == 'Q') || (k == 'q'))

{

printf("===================================================================\n");

//отождествление

int s1, s2;

printf("Отождествление вершин графа M1\n\n");

printf("Введите первую вершину:");

scanf("%d", &s1);

printf("Введите вторую вершину:");

scanf("%d", &s2);

printf("\n");

int\*\* a1;

a1 = (int\*\*)malloc(sizeof(int\*) \* size1 - 1);

for (int i = 0; i < size1 - 1; i++) {

a1[i] = (int\*)malloc(sizeof(int) \* size1 - 1);

}

otoz(a1, arr1, size1, s1, s2);

printf("Отождествление вершин %d и %d для M1:\n", s1, s2);

print\_matrix(size1 - 1, a1);

for (int i = 0; i < size1 - 1; ++i) free(a1[i]);

free(a1);

}

else if ((k == 'W') || (k == 'w'))

{

printf("===================================================================\n");

//стягивание

int s3, s4;

printf("Стягивание графа M1\n\n");

printf("Введите первую вершину:");

scanf("%d", &s3);

printf("Введите вторую вершину:");

scanf("%d", &s4);

printf("\n");

int\*\* a2;

a2 = (int\*\*)malloc(sizeof(int\*) \* size1 - 1);

for (int i = 0; i < size1 - 1; i++) {

a2[i] = (int\*)malloc(sizeof(int) \* size1 - 1);

}

if (arr2[s3][s4] == 0) {

printf("Ребра не существует\n\n");

}

else {

otoz(a2, arr2, size1, s3, s4);

for (int i = 0; i < size1 - 1; i++) {

for (int j = 0; j < size1 - 1; j++) {

if (i == j) {

a2[i][j] = 0;

}

}

}

printf("Стягивание ребра между вершинами %d и %d для M1:\n", s3, s4);

print\_matrix(size1 - 1, a2);

}

for (int i = 0; i < size1 - 1; ++i) free(a2[i]);

free(a2);

}

else if ((k == 'E') || (k == 'e'))

{

printf("===================================================================\n");

//расщепление

int s5;

printf("Расщепление вершины графа M1\n\n");

printf("Введите вершину для расщепления:");

scanf("%d", &s5);

printf("\n");

int sss = size1 + 1;

int\*\* a3;

a3 = (int\*\*)malloc(sizeof(int\*) \* sss);

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

a3[i] = (int\*)malloc(sizeof(int) \* sss);

}

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

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

a3[i][j] = 0;

}

}

for (int i = 0; i < size1; i++) {

for (int j = 0; j < size1; j++) {

a3[i][j] = arr3[i][j];

}

}

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

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

a3[i][size1] = a3[i][s5]; //size1 == sss - 1

a3[size1][j] = a3[s5][j];

}

}

a3[s5][size1] = 1;

a3[size1][s5] = 1;

printf("Расщепление вершины %d для графа M1:\n", s5);

print\_matrix(sss, a3);

for (int i = 0; i < sss; ++i) free(a3[i]);

free(a3);

}

else if (k == 27)

{

break;

}

}

for (int i = 0; i < size1; ++i) free(arr1[i]);

free(arr1);

for (int i = 0; i < size1; ++i) free(arr2[i]);

free(arr2);

for (int i = 0; i < size1; ++i) free(arr3[i]);

free(arr3);

\_getch();

break;

}

switch (menu) { //5

case '5':

system("cls");

int N;

printf("Введите количество вершин графа: ");

scanf("%d", &N);

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

for (int i = 0; i < N; i++) A[i] = (int\*)malloc(sizeof(int) \* N);

create\_matrix(N, A);

printf("Матрица смежности:\n");

print\_matrix(N, A);

while (1)

{

printf("Введите Q(q), W(w), E(e) для отождествления, стягивания ребра и расщепления соответственно\n");

int k = \_getch();

if ((k == 'Q') || (k == 'q'))

{

//Отождествление

LA = make\_LA(N + 1, A);

print\_LA(N, LA);

merge\_LA(N + 1, LA, 0);

print\_LA(N + 1, LA);

}

else if ((k == 'W') || (k == 'w'))

{

//Стягивание

LA = make\_LA(N + 1, A);

print\_LA(N, LA);

merge\_LA(N + 1, LA, 1);

print\_LA(N + 1, LA);

}

else if ((k == 'E') || (k == 'e'))

{

//Расщепление

LA = make\_LA(N + 1, A);

print\_LA(N, LA);

split\_LA(N + 1, LA);

print\_LA(N + 1, LA);

}

// esc

else if (k == 27)

{

break;

}

}

\_getch();

break;

}

switch (menu) { //3

case '3':

system("cls");

int size1, size2;

int\*\* arr1;

int\*\* arr2;

int\*\* arr3;

printf("Введите размер матрицы смежности M1: ");

scanf("%d", &size1);

printf("\n");

printf("Введите размер матрицы смежности M1: ");

scanf("%d", &size2);

printf("\n");

arr1 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr1[i] = (int\*)malloc(sizeof(int) \* size1);

}

arr2 = (int\*\*)malloc(sizeof(int\*) \* size2);

for (int i = 0; i < size2; i++) {

arr2[i] = (int\*)malloc(sizeof(int) \* size2);

}

printf("Матрица M1:\n");

create\_matrix(size1, arr1);

print\_matrix(size1, arr1);

printf("Матрица M2:\n");

create\_matrix(size2, arr2);

print\_matrix(size2, arr2);

printf("===================================================================\n");

printf("Объединение графов M1 и M2:\n\n");

if (size1 < size2) {

arr3 = (int\*\*)malloc(sizeof(int\*) \* size2);

for (int i = 0; i < size2; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size2);

}

obed(arr1, arr2, arr3, size1, size2);

}

else {

arr3 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size1);

}

obed(arr2, arr1, arr3, size2, size1);

}

printf("===================================================================\n");

printf("Пересечение графов M1 и M2:\n\n");

if (size1 < size2) {

arr3 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size1);

}

peres(arr1, arr2, arr3, size1, size2);

}

else {

arr3 = (int\*\*)malloc(sizeof(int\*) \* size2);

for (int i = 0; i < size2; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size2);

}

peres(arr2, arr1, arr3, size2, size1);

}

printf("===================================================================\n");

printf("Кольцевая сумма графов M1 и M2:\n\n");

if (size1 < size2) {

arr3 = (int\*\*)malloc(sizeof(int\*) \* size2);

for (int i = 0; i < size2; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size2);

}

kol\_sum(arr1, arr2, arr3, size1, size2);

}

else {

arr3 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size1);

}

kol\_sum(arr2, arr1, arr3, size2, size1);

}

for (int i = 0; i < size1; ++i) free(arr1[i]);

free(arr1);

for (int i = 0; i < size2; ++i) free(arr2[i]);

free(arr2);

\_getch();

break;

}

switch (menu) { //4

case '4':

system("cls");

int size1, size2;

int\*\* arr1;

int\*\* arr2;

int\*\* arr3;

printf("Введите размер матрицы смежности M1: ");

scanf("%d", &size1);

printf("\n");

printf("Введите размер матрицы смежности M1: ");

scanf("%d", &size2);

printf("\n");

arr1 = (int\*\*)malloc(sizeof(int\*) \* size1);

for (int i = 0; i < size1; i++) {

arr1[i] = (int\*)malloc(sizeof(int) \* size1);

}

arr2 = (int\*\*)malloc(sizeof(int\*) \* size2);

for (int i = 0; i < size2; i++) {

arr2[i] = (int\*)malloc(sizeof(int) \* size2);

}

printf("Матрица M1:\n");

create\_matrix(size1, arr1);

print\_matrix(size1, arr1);

printf("Матрица M2:\n");

create\_matrix(size2, arr2);

print\_matrix(size2, arr2);

printf("===================================================================\n");

printf("Декартово произведение графов M1 и M2:\n\n");

int size3 = size1 \* size2;

arr3 = (int\*\*)malloc(sizeof(int\*) \* size3);

for (int i = 0; i < size3; i++) {

arr3[i] = (int\*)malloc(sizeof(int) \* size3);

}

int m = -1, n = -1;

for (int i = 0; i < size1; i++)

{

for (int k = 0; k < size2; k++)

{

n++;

m = -1;

for (int j = 0; j < size1; j++)

{

for (int l = 0; l < size2; l++)

{

m++;

if (i == j)

{

arr3[m][n] = arr2[k][l];

}

if (k == l)

{

arr3[m][n] = arr1[i][j];

}

if ((i != j) && (k != l)) {

arr3[m][n] = 0;

}

}

}

}

}

printf("Результат \n");

print\_matrix(size3, arr3);

for (int i = 0; i < size1; ++i) free(arr1[i]);

free(arr1);

for (int i = 0; i < size2; ++i) free(arr2[i]);

free(arr2);

for (int i = 0; i < size3; ++i) free(arr3[i]);

free(arr3);

\_getch();

break;

}

} while (menu != 27);

return 0;

}

**Результаты работы программы**

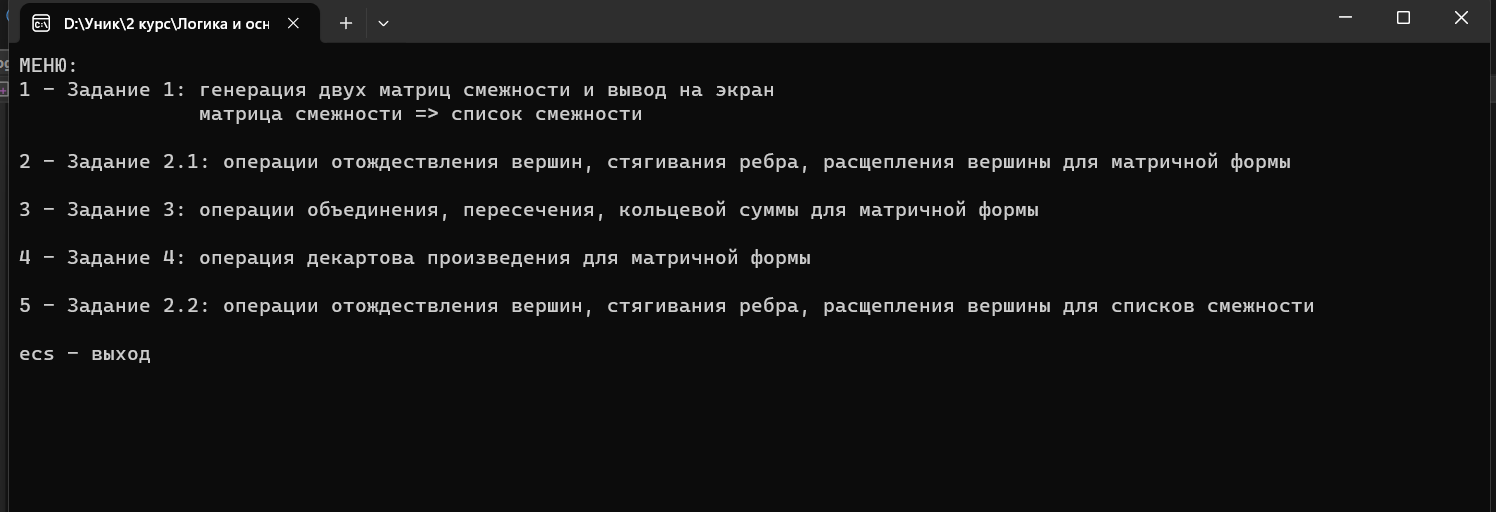


Рисунок 1 – Меню программы

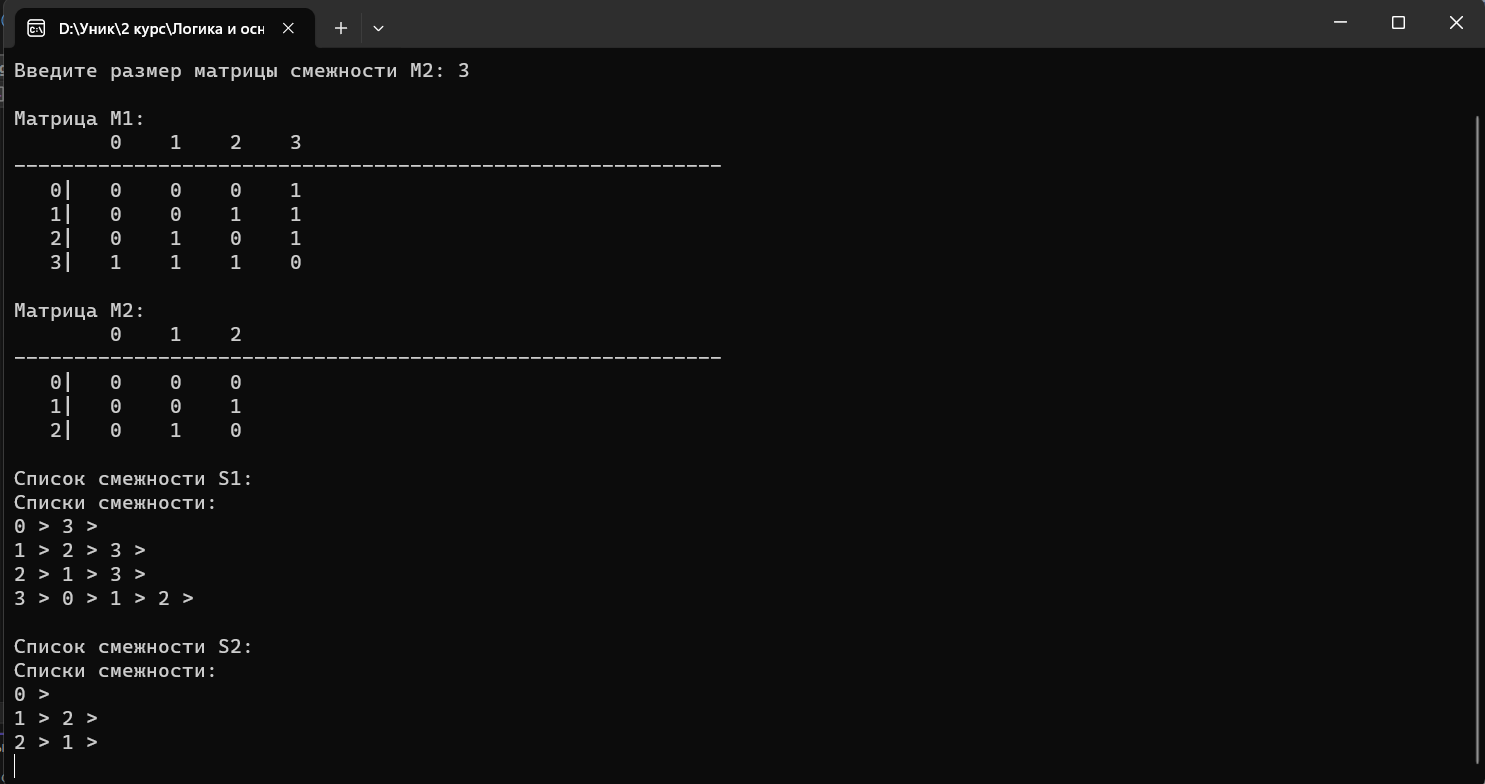


Рисунок 2 – Задание 1

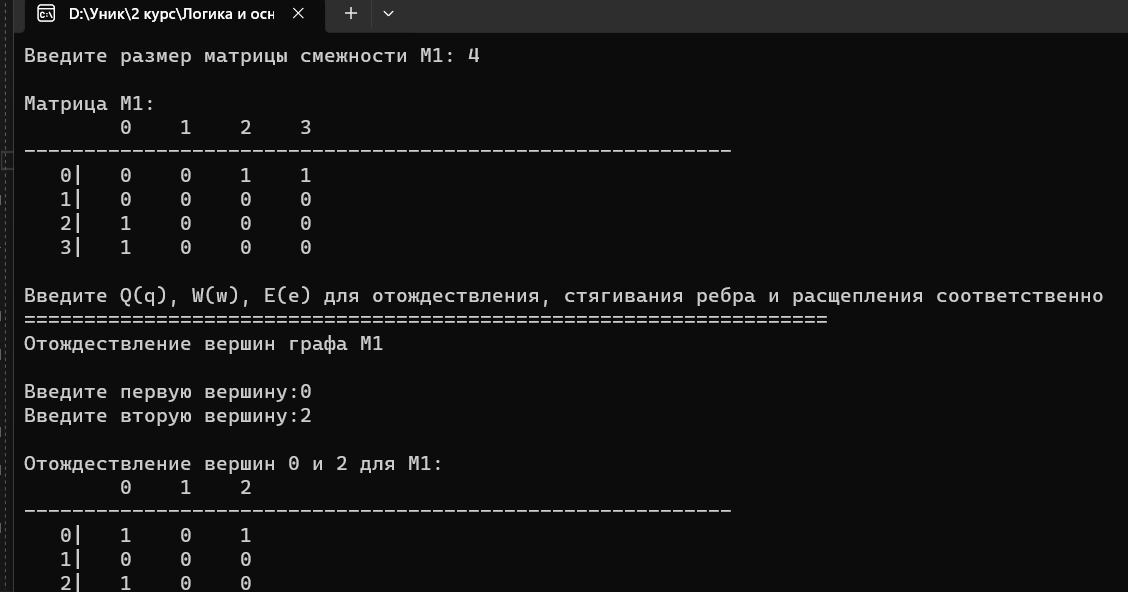


Рисунок 3 – Задание 2.1 (отождествление вершин в матричной форме)

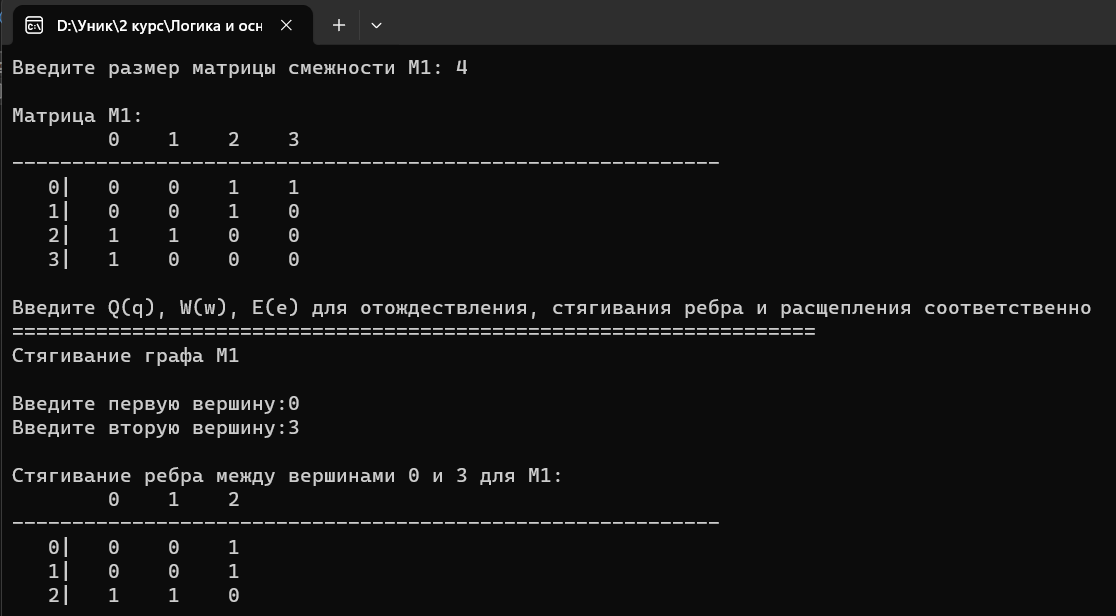


Рисунок 4 – Задание 2.1 (стягивание ребра в матричной форме)

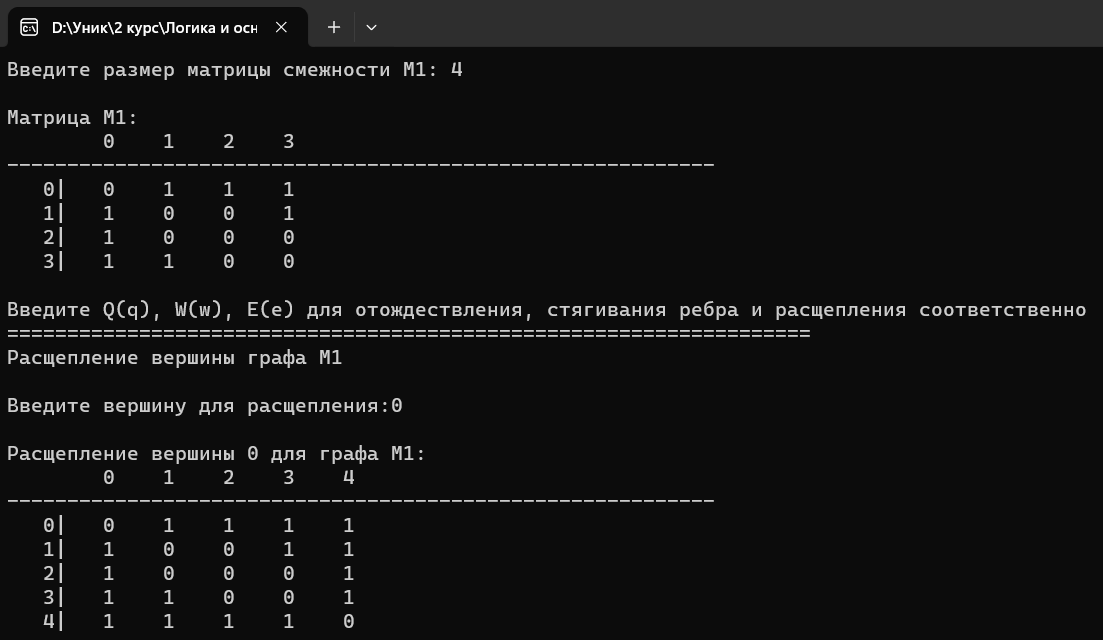


Рисунок 5 – Задание 2.1 (расщепление вершин в матричной форме)

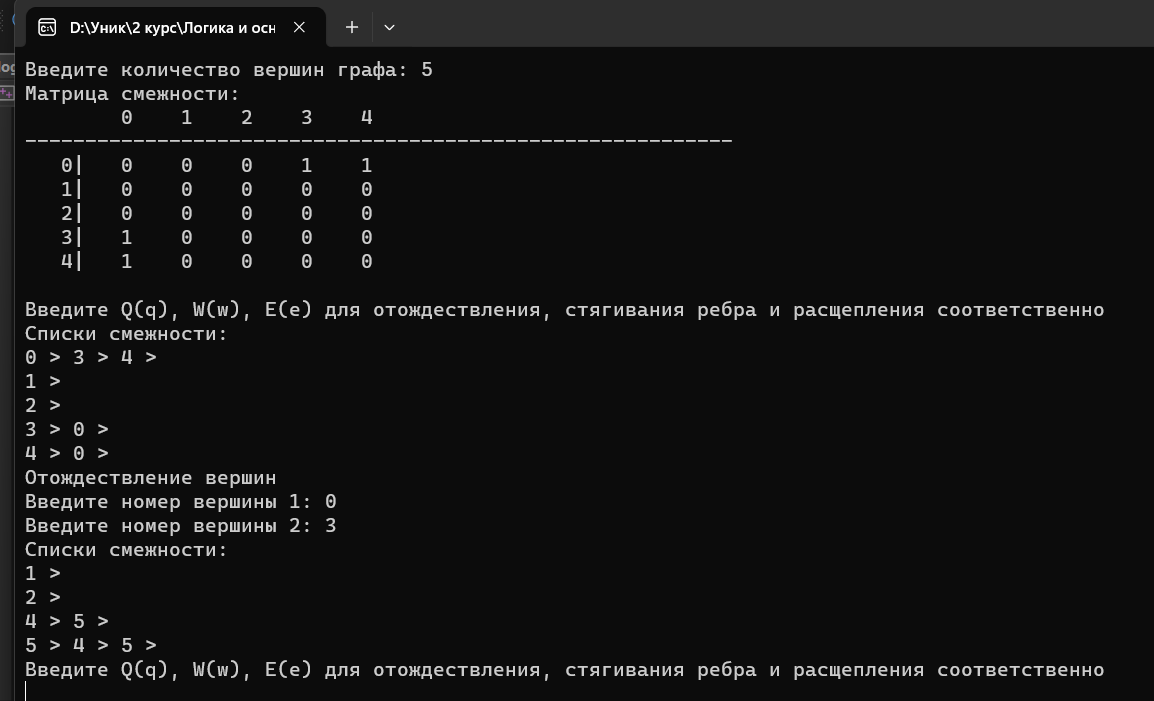


Рисунок 6 – Задание 2.2 (отождествление вершин в списках)

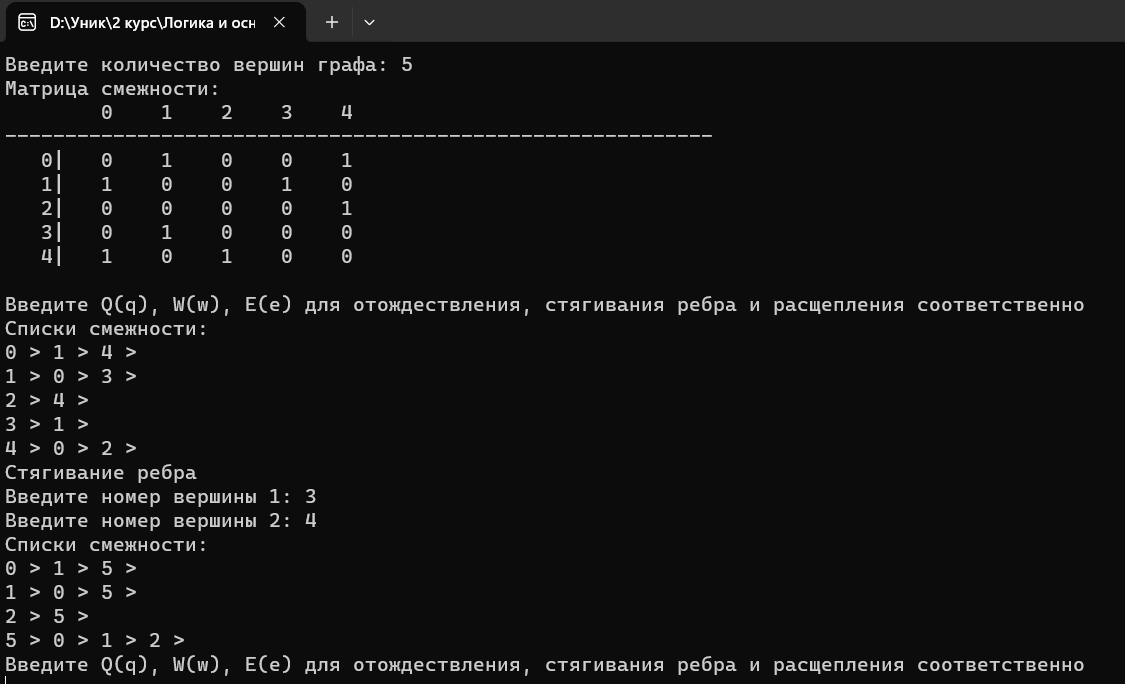


Рисунок 7 – Задание 2.2 (стягивание вершин в списках)

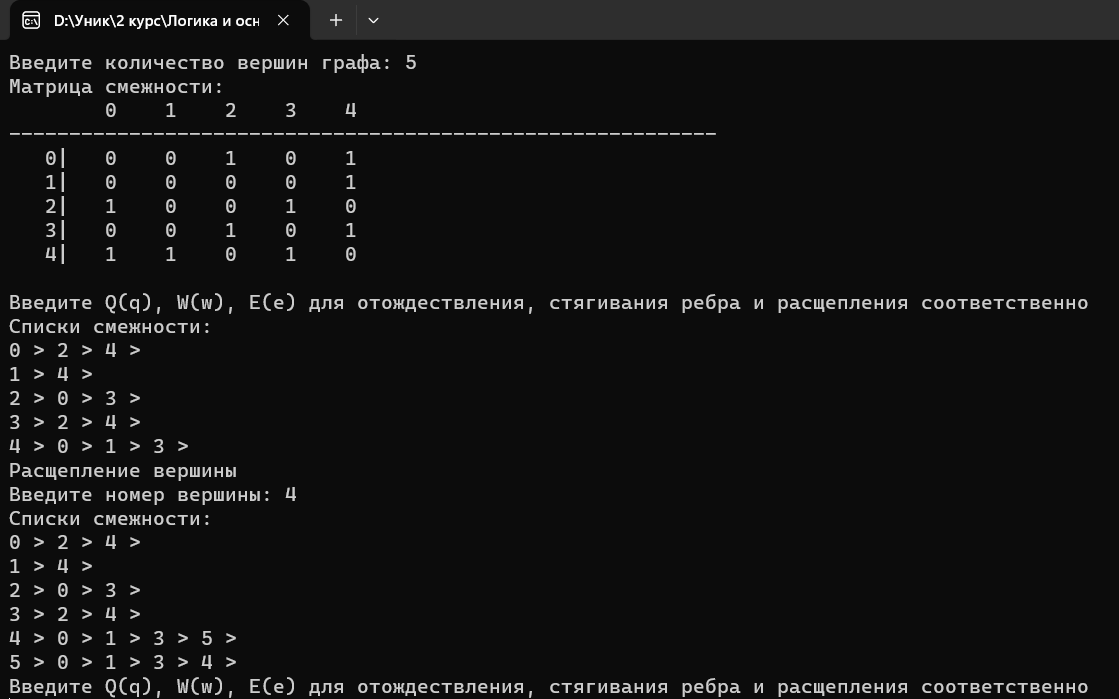


Рисунок 8 – Задание 2.3 (расщепление вершин в списках)

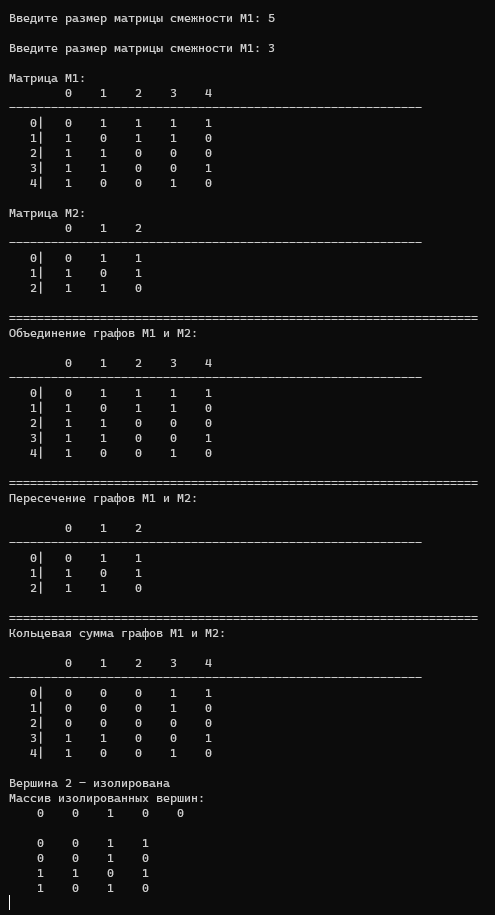


Рисунок 9 – Задание 3

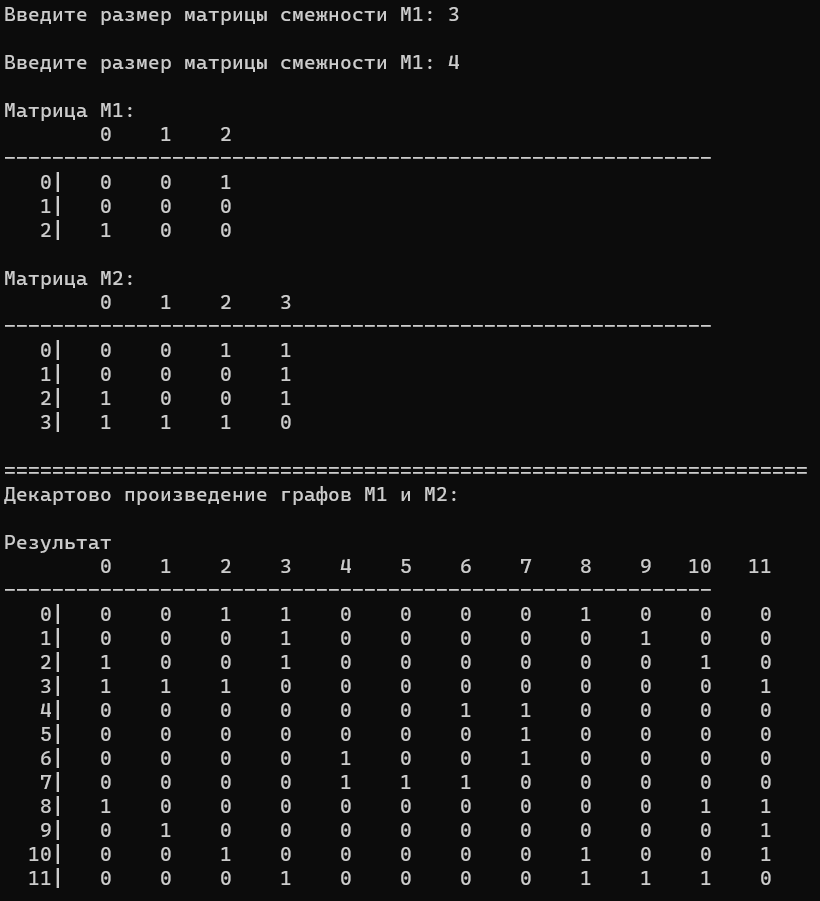


Рисунок 10 – Задание 4

**Вывод**

В ходе выполнения лабораторной работы была разработана программа, выполняющая унарные и бинарные операции над графами. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок.

Получили опыт в создании проектов в среде Microsoft Visual Studio, приобрели навыки программирования алгоритмов.