Министерство науки и высшего 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:

студенты группы 19ВВ2

Горбунов Д. А

Карамышев А. А

Сидоров Н. Р.

Пенза 2020

# Цель работы

Освоить унарные и бинарные операции над графами

# Лабораторное задание

**Задание 1:**

1. Сгенерировали (используя генератор случайных чисел) две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Вывели сгенерированные матрицы на экран.

**Задание 2**:

1. Для матричной формы представления графов выполнили операцию:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Результат выполнения операции вывели на экран.

**Задание 3**

1. Для матричной формы представления графов выполнили операцию:

а) объединения *G* = *G*1  *G*2

б) пересечения *G* = *G*1  *G*2

в) кольцевой суммы *G* = *G*1  *G*2

**Задание 4**

1. Выполнили декартово произведение двух графов.

Результат выполнения операции вывели на экран.

### **Листинг**

#define CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <windows.h>

#include <iostream>

using namespace std;

void vivod(int n, int\*\* p) {

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

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

if (p[i][j] == 0) {

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

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

}

else {

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

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

}

}

printf("\n");

}

}

int\*\* delit(int\*\* p1, int n, int del)

{ //На вход ссылка на двумерный массив, и количество графов

int\*\* p = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под врменный двумерный массив

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

{

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

}

int k = 0, m = 0; //k~~i,m~~j

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

{

if (i == del)

{

continue;

}

else {

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

if (j == del)

{

continue;

}

else

{

p[k][m] = p1[i][j];

if (p[k][m] == 0) {

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

printf("%d ", p[k][m]);

}

else {

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

printf("%d ", p[k][m]);

}

m++;

}

m = 0;

k++;

}

printf("\n");

}

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

{

free(p1[i]);

}

free(p1);

n = n - 1;

p1 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под двумерный массив p1

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

{

p1[i] = (int\*)malloc(n \* sizeof(int));

}

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

{

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

{

p1[i][j] = p[i][j];

}

}

printf("\n\n");

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

{

free(p[i]);

}

free(p);

return (p1);

} // На выход измененный массив

int\*\* dob(int\*\* p1, int n, int add)

{

n = n + 1;

int\*\* p = (int\*\*)malloc((n) \* sizeof(int\*)); //выделение памяти под временную матрицу порядок которой на 1 больше

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

{

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

}

int m = 0, k = 0;

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

{

if (i == add)

{

i = i + 1;

}

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

{

if (j == add)

{

j = j + 1;

}

p[i][j] = p1[m][k];

k++;

}

k = 0;

m++;

//if (m == add) {

//break;

//}

}

for (int i = 0; i < n; i++) // обнуление центра и заполнение пустых элементов

{

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

{

if (i == add - 1)

{

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

{

p[i + 1][j] = 0;

}

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

{

int temp = rand() % 2;

if (temp == 1)

{

p[i + 1][j] = 1;

p[i][j] = 0;

}

else

{

p[i + 1][j] = 0;

}

}

}

if (j == add - 1)

{

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

{

p[i][j + 1] = 0;

}

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

{

int temp = rand() % 2;

if (temp == 1)

{

p[i][j + 1] = 1;

p[i][j] = 0;

}

else

{

p[i][j + 1] = 0;

}

}

}

if (i == j)

{

p[i][j] = 0;

}

}

}

p[add - 1][add] = 1;

p[add][add - 1] = 1;

for (int i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p[i][j] = p[j][i];

}

}

printf("\n\n");

return (p);

}

int\*\* zamikaniye(int\*\* p1, int n)

{

int i = 0, j = 0;

int x = 0, y = 0;

printf("Введите вершины для отождествления: ");

scanf\_s("%d %d", &x, &y);

x -= 1;

y -= 1;

int del = 0;

for (i = 0; i < n; i++) //сложение строчек

{

if (i == x)

{

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

{

p1[x][j] += p1[y][j];

}

}

else continue;

}

for (i = 0; i < n; i++) // сложение столбцов

{

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

{

if (j == x)

{

p1[i][x] += p1[1][y];

}

else continue;

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p1[j][i] = p1[i][j];

}

}

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

{

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

{

if (p1[i][j] == 2)

{

p1[i][j] = 1;

}

}

}

p1 = delit(p1, n, y);

return p1;

}

int\*\* styagivaniye(int\*\* p1, int n)

{

int i = 0, j = 0;

int x = 0, y = 0;

printf("Введите вершины для стягивания: ");

scanf\_s("%d %d", &x, &y);

x -= 1;

y -= 1;

int del = 0;

for (i = 0; i <

n; i++) //сложение строчек

{

if (i == x)

{

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

{

p1[x][j] += p1[y][j];

}

}

else continue;

}

for (i = 0; i < n; i++) // сложение столбцов

{

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

{

if (j == x)

{

p1[i][x] += p1[1][y];

}

else continue;

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p1[j][i] = p1[i][j];

}

}

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

{

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

{

if (p1[i][j] == 2)

{

p1[i][j] = 1;

}

}

}

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

{

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

{

if (i == j)

{

p1[i][j] = 0;

}

}

}

p1 = delit(p1, n, y);

return p1;

}

int main() {

setlocale(LC\_ALL, "Rus");

int i = 0, j = 0, n;

printf("Введите размер генерируемой матрицы: >> ");

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

int\*\* p1, \*\* p2, \*\* p, \*\* p3;

p1 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под двумерный массив p1

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

{

p1[i] = (int\*)malloc(n \* sizeof(int));

}

for (i = 0; i < n; i++) //заполнение массива p1

{

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

{

if (i == j)

{

p1[i][j] = 0;

}

else

{

p1[i][j] = rand() % 2;

}

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p1[j][i] = p1[i][j];

}

}

while (true) //меню взаимодействия

{

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, 0 | FOREGROUND\_INTENSITY);

int choose;

system("cls");

cout << "1. Вывод матрицы №1" << endl;

cout << "2. Удаление узла " << endl;

cout << "3. Расщепление узла" << endl;

cout << "4. Замыкание узлов" << endl;

cout << "5. Стягивание узлов" << endl;

cout << "6. Обьединение матриц" << endl;

cout << "7. Кольцевая сумма" << endl;

cout << "8. Пересечение" << endl;

cout << "9. Декартово произведение" << endl;

cout << "\nВведи вариант меню: \n>";

cin >> choose;

if (choose == 1) //вывод

{

system("cls");

vivod(n, p1);

printf("\n\n");

//vivod(n, p2);

system("pause");

continue;

}

if (choose == 2) //удаление узла

{

vivod(n, p1);

cout << endl;

int del;

printf("Какой узел необходимо удалить? ");

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

del = del - 1;

p1 = delit(p1, n, del);

n = n - 1;

system("pause");

continue;

}

if (choose == 3) //Добавление узла

{

vivod(n, p1);

cout << endl;

int add;

printf("После какого узла добавить новый? ");

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

add = add;

p = (int\*\*)malloc((n + 1) \* sizeof(int\*)); //выделение памяти под двумерный массив p2

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

{

p[i] = (int\*)malloc((n + 1) \* sizeof(int));

}

p = dob(p1, n, add);

vivod(n + 1, p);

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

{

free(p1[i]);

}

free(p1);

n = n + 1;

p1 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под двумерный массив p2

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

{

p1[i] = (int\*)malloc(n \* sizeof(int));

}

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

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

p1[i][j] = p[i][j];

}

}

system("pause");

continue;

}

if (choose == 4) //Замыкание узла

{

vivod(n, p1);

cout << endl;

p1 = zamikaniye(p1, n);

n -= 1;

system("pause");

continue;

}

if (choose == 5) //Стягивание узла

{

vivod(n, p1);

cout << endl;

p1 = styagivaniye(p1, n);

n = n - 1;

system("pause");

continue;

}

if (choose == 6) { //Обьединение

p2 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под двумерный массив p2

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

{

p2[i] = (int\*)malloc(n \* sizeof(int));

}

for (i = 0; i < n; i++) //заполнение массива p2

{

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

{

if (i == j)

{

p2[i][j] = 0;

}

else

{

p2[i][j] = rand() % 2;

}

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

for (j = 0; j < n;

j++)

{

p2[j][i] = p2[i][j];

}

}

cout << "Матрица #1" << endl << endl;

vivod(n, p1);

cout << endl;

cout << "Матрица #2" << endl << endl;

vivod(n, p2);

cout << endl;

cout << "Результат: " << endl;

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

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

if (p1[i][j] + p2[i][j] != 2)

{

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << p1[i][j] + p2[i][j] << " ";

}

else

{

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

cout << "1 ";

}

}

cout << endl;

}

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

{

free(p2[i]);

}

free(p2);

system("pause");

continue;

}

if (choose == 7) //Кольцевая сумма

{

p2 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под двумерный массив p2

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

{

p2[i] = (int\*)malloc(n \* sizeof(int));

}

for (i = 0; i < n; i++) //заполнение массива p2

{

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

{

if (i == j)

{

p2[i][j] = 0;

}

else

{

p2[i][j] = rand() % 2;

}

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p2[j][i] = p2[i][j];

}

}

cout << "Матрица #1" << endl << endl;

vivod(n, p1);

cout << endl;

cout << "Матрица #2" << endl << endl;

vivod(n, p2);

cout << endl;

cout << "Результат: " << endl;

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

{

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

{

if (p1[i][j] + p2[i][j] == 2) {

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "0 ";

}

if (p1[i][j] + p2[i][j] == 1) {

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

cout << "1 ";

}

if (p1[i][j] + p2[i][j] == 0) {

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << "0 ";

}

}

cout << endl;

}

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

{

free(p2[i]);

}

free(p2);

system("pause");

continue;

}

if (choose == 8) //Пересечение

{

p2 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под двумерный массив p2

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

{

p2[i] = (int\*)malloc(n \* sizeof(int));

}

for (i = 0; i < n; i++) //заполнение массива p2

{

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

{

if (i == j)

{

p2[i][j] = 0;

}

else

{

p2[i][j] = rand() % 2;

}

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p2[j][i] = p2[i][j];

}

}

cout << "Матрица #1" << endl << endl;

vivod(n, p1);

cout << endl;

cout << "Матрица #2" << endl << endl;

vivod(n, p2);

cout << endl;

cout << "Результат: " << endl;

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

{

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

{

if (p1[i][j] \* p2[i][j] == 0) {

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << p1[i][j] \* p2[i][j] << " ";

}

else {

HANDLE hOUTPUT = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hOUTPUT, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

cout << p1[i][j] \* p2[i][j] << " ";

}

}

cout << endl;

}

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

{

free(p2[i]);

}

free(p2);

system("pause");

continue;

}

if (choose == 9)

{

int n2 = n \* n;

p3 = (int\*\*)malloc(n2 \* sizeof(int\*)); //выделение памяти под двумерный массив p2

for (int i = 0; i < n2; i++)

{

p3[i] = (int\*)malloc(n2 \* sizeof(int));

}

for (int i = 0; i < n2; i++) {

for (int j = 0; j < n2; j++) {

p3[i][j] = 0;

}

}

p2 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под двумерный массив p2

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

{

p2[i] = (int\*)malloc(n \* sizeof(int));

}

for (i = 0; i < n; i++) //заполнение массива p2

{

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

{

if (i == j)

{

p2[i][j] = 0;

}

else

{

p2[i][j] = rand() % 2;

}

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p2[j][i] = p2[i][j];

}

}

int x1, x2, y1, y2;

for (int i = 0; i < n2; i++) {

for (int j = 0; j < n2; j++) {

x1 = i / n;

x2 = i % n;

y1 = j / n;

y2 = j % n;

if (x1 == y1) {

if (p2[x2][y2]) {

p3[i][j] = 1;

}

}

else {

if (x2 == y2) {

if (p1[x1][y1]) {

p3[i][j] = 1;

}

}

}

}

}

printf("Декартово произведение: \n\n");

cout << "Матрица #1" << endl << endl;

vivod(n, p1);

cout << endl;

cout << "Матрица #2" << endl << endl;

vivod(n, p2);

cout << endl;

cout << "Матрица #3" << endl << endl;

vivod(n2, p3);

cout << endl;

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

{

free(p2[i]);

free(p3[i]);

}

free(p2);

free(p3);

system("pause");

continue;

}

if (choose == 0) //выход

{

break;

}

}

////////////////////////////////////////////// // очистка памяти

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

{

free(p1[i]);

}

free(p1);

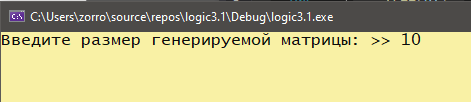
/////////////////////////////////////////////

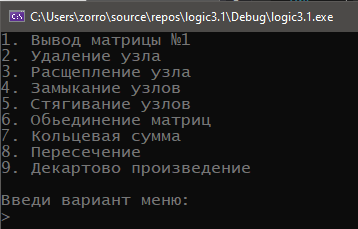
return 0;

}

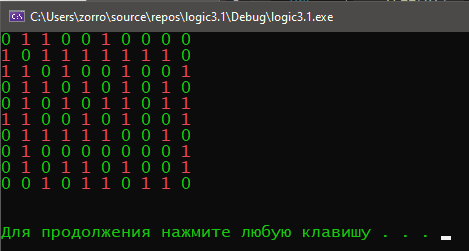
## **Результаты работы программы**

Ввод данных о матрице:

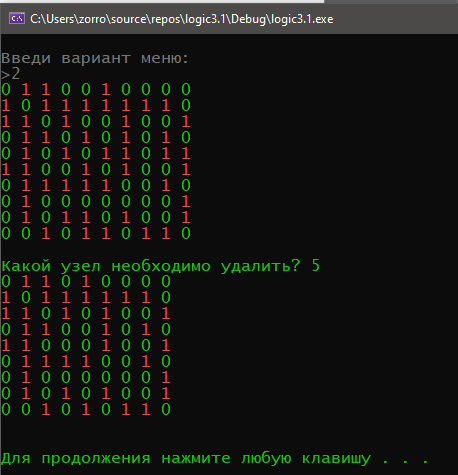




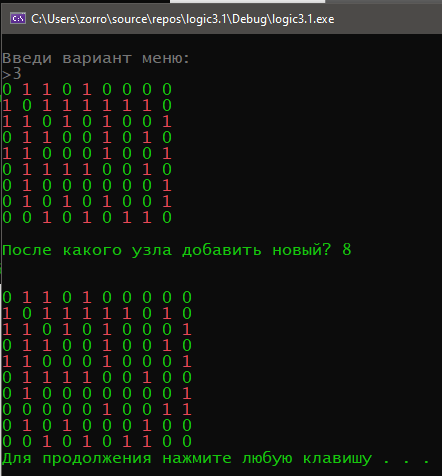
1. Вывод матрицы на экран:



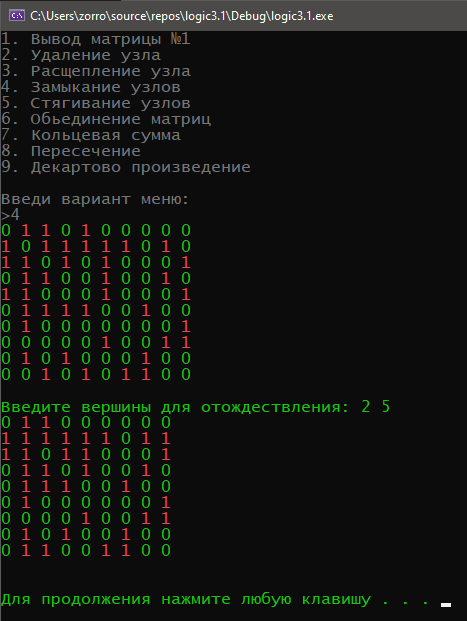
1. Удаление вершины:



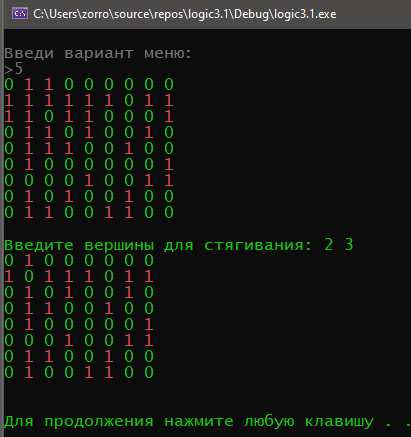
1. Расщепление вершины:



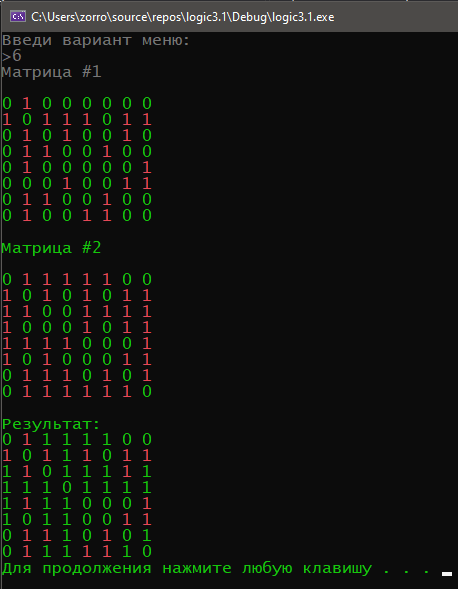
1. Отождествление:



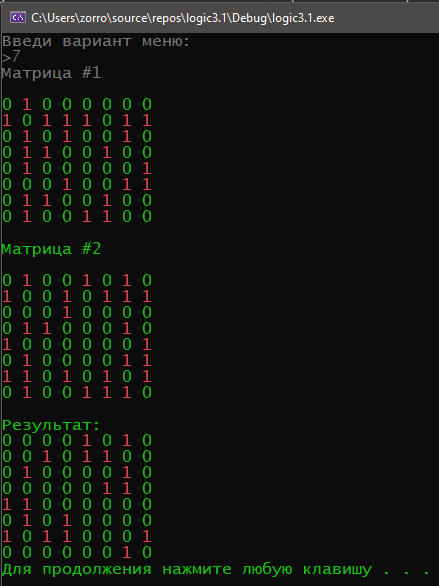
1. Стягивание ребра:



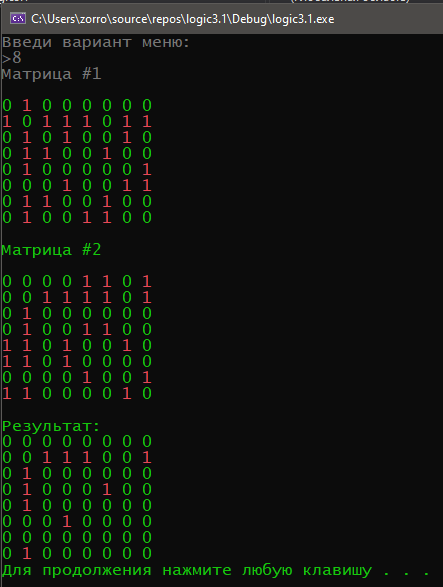
1. Объединение матриц:

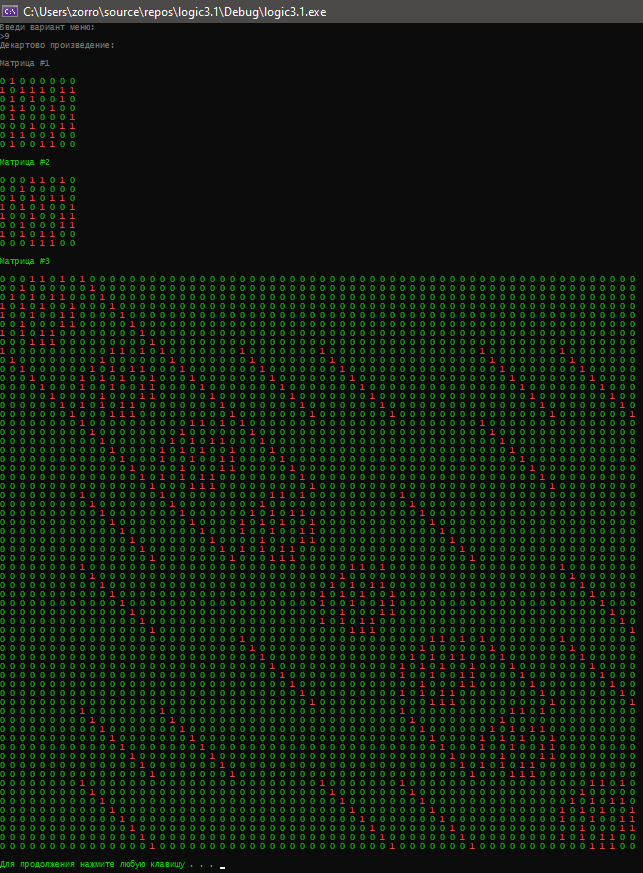


1. Кольцевая сумма:



1. Пересечение:



1. Декартово произведение:

## **Вывод**

Усвоили бинарные и унарные операции при работе с графами и научились применять их на практике: