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»

**Выполнил:**

**Студент группы 23ВВВ2**

Стрельцов А.П.

Федоров Б.М.

**Приняли:**

Юрова О. В.

Митрохин М. А.

Пенза 2024

**Цель работы:** проработать унарные и бинарные операции с графами, выполнить задания и усвоить материал.

**Лабораторное задание.**

**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.

**ЛИСТИНГ**

**lab6-1.cpp**

﻿#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#define MAX\_VERTICES 10 // Максимальное количество вершин в графе

// Структура для представления списка смежности

typedef struct Node {

int vertex; // Номер вершины

struct Node\* next; // Указатель на следующий элемент списка

} Node;

typedef struct {

Node\* head; // Указатель на голову списка

} AdjList;

void generate\_adjacency\_matrix(int matrix[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

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

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

if (i == j) {

matrix[i][j] = 0; // Нет петель

}

else {

matrix[i][j] = rand() % 2; // Генерация наличия ребра (0 или 1)

matrix[j][i] = matrix[i][j]; // Симметричность для неориентированного графа

}

}

}

}

void print\_adjacency\_matrix(int matrix[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

setlocale(LC\_ALL, "RUS");

printf("Матрица смежности:\n");

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

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

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

}

printf("\n");

}

}

void convert\_to\_adjacency\_list(int matrix[MAX\_VERTICES][MAX\_VERTICES], AdjList list[MAX\_VERTICES], int vertices) {

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

list[i].head = NULL; // Инициализация списка

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

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

// Создание нового узла для списка

Node\* new\_node = (Node\*)malloc(sizeof(Node));

new\_node->vertex = j; // Установка номера вершины

new\_node->next = list[i].head; // Указатель на предыдущую голову списка

list[i].head = new\_node; // Обновление головы списка

}

}

}

}

void print\_adjacency\_list(AdjList list[MAX\_VERTICES], int vertices) {

setlocale(LC\_ALL, "RUS");

printf("Список смежности:\n");

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

printf("Вершина %d: ", i);

Node\* current = list[i].head;

while (current != NULL) {

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

current = current->next;

}

printf("NULL\n");

}

}

void free\_adjacency\_list(AdjList list[MAX\_VERTICES], int vertices) {

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

Node\* current = list[i].head;

while (current != NULL) {

Node\* temp = current;

current = current->next;

free(temp);

}

}

}

int main() {

srand(time(NULL));

int vertices = 5; // Количество вершин в графах

int M1[MAX\_VERTICES][MAX\_VERTICES], M2[MAX\_VERTICES][MAX\_VERTICES];

// Генерация первой матрицы смежности

generate\_adjacency\_matrix(M1, vertices);

print\_adjacency\_matrix(M1, vertices);

// Генерация второй матрицы смежности

generate\_adjacency\_matrix(M2, vertices);

print\_adjacency\_matrix(M2, vertices);

// Преобразование первой матрицы в список смежности

AdjList list1[MAX\_VERTICES];

convert\_to\_adjacency\_list(M1, list1, vertices);

print\_adjacency\_list(list1, vertices);

// Преобразование второй матрицы в список смежности

AdjList list2[MAX\_VERTICES];

convert\_to\_adjacency\_list(M2, list2, vertices);

print\_adjacency\_list(list2, vertices);

// Освобождение памяти

free\_adjacency\_list(list1, vertices);

free\_adjacency\_list(list2, vertices);

return 0;

}

**Результат работы программы**

На рисунке 1 показана реализация задания №1.

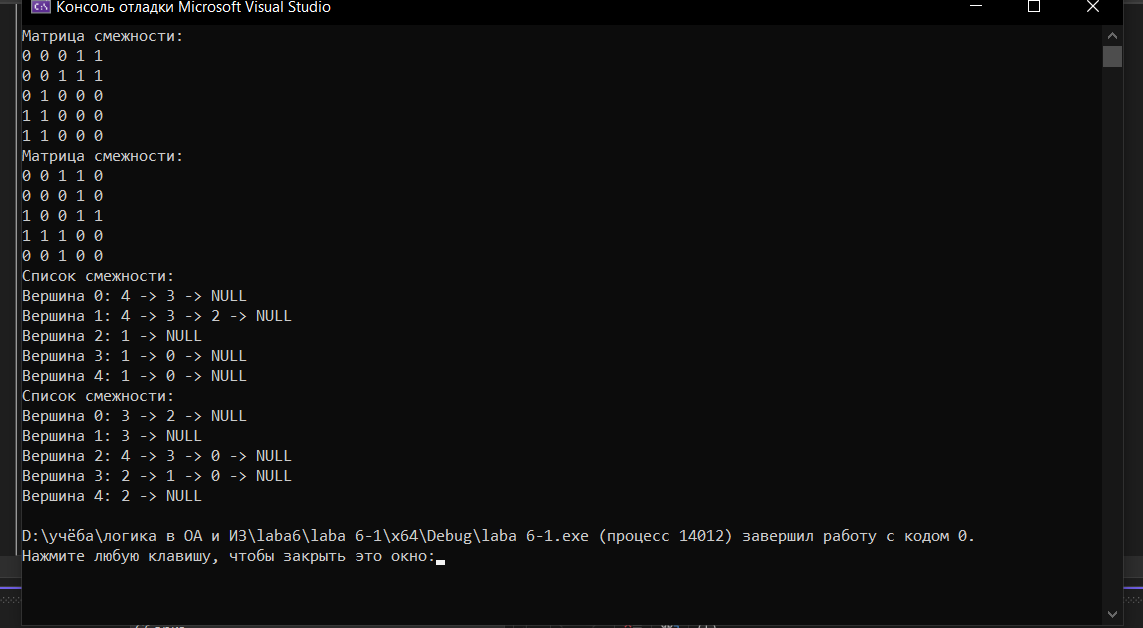


Рисунок 1 - Результат работы программы №1.

**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.

1. \* 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.

**ЛИСТИНГ**

**lab6-2.cpp**

﻿#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#define MAX\_SIZE 10 // Максимальный размер графа

// Функция для генерации матрицы смежности

void generate\_adjacency\_matrix(int matrix[MAX\_SIZE][MAX\_SIZE], int size) {

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

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

if (i == j) {

matrix[i][j] = 0; // Нет петель

}

else {

matrix[i][j] = rand() % 2; // Случайное значение 0 или 1

}

}

}

// Обеспечиваем симметричность для неориентированного графа

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

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

matrix[j][i] = matrix[i][j]; // Копируем значения

}

}

}

// Функция для вывода матрицы смежности

void print\_adjacency\_matrix(int matrix[MAX\_SIZE][MAX\_SIZE], int size) {

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

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

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

}

printf("\n");

}

}

// Функция для отождествления вершин

void merge\_vertices(int matrix[MAX\_SIZE][MAX\_SIZE], int size, int v1, int v2) {

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

matrix[v1][i] = matrix[v1][i] | matrix[v2][i]; // Объединяем рёбра

matrix[i][v1] = matrix[i][v1] | matrix[i][v2]; // Объединяем рёбра

}

// Удаляем вершину v2

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

matrix[v2][i] = 0;

matrix[i][v2] = 0;

}

}

// Функция для стягивания ребра

void collapse\_edge(int matrix[MAX\_SIZE][MAX\_SIZE], int v1, int v2) {

matrix[v1][v2] = 0;

matrix[v2][v1] = 0;

}

// Функция для расщепления вершины

void split\_vertex(int matrix[MAX\_SIZE][MAX\_SIZE], int\* size, int v) {

setlocale(LC\_ALL, "RUS");

if (\*size >= MAX\_SIZE) {

printf("Достигнуто максимальное количество вершин!\n");

return;

}

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

if (matrix[v][i] == 1) {

matrix[\*size][i] = 1; // Новая вершина соединяется с соседями

matrix[i][\*size] = 1; // Обеспечиваем симметричность

}

}

(\*size)++; // Увеличиваем размер графа

}

// Функция для вывода списка смежности

void print\_adjacency\_list(int matrix[MAX\_SIZE][MAX\_SIZE], int size) {

setlocale(LC\_ALL, "RUS");

printf("\nСписок смежности:\n");

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

printf("%d: ", i);

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

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

printf("%d ", j);

}

}

printf("\n");

}

}

int main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL)); // Инициализация генератора случайных чисел

int size;

int M[MAX\_SIZE][MAX\_SIZE] = { 0 };

// Запрос количества вершин у пользователя

printf("Введите количество вершин в графе (макс. %d): ", MAX\_SIZE);

scanf("%d", &size);

if (size > MAX\_SIZE) {

printf("Количество вершин превышает максимальное значение. Установлено на %d.\n", MAX\_SIZE);

size = MAX\_SIZE;

}

// Генерация матрицы смежности

generate\_adjacency\_matrix(M, size);

// Вывод сгенерированной матрицы

printf("Матрица смежности:\n");

print\_adjacency\_matrix(M, size);

int operation, v1, v2;

while (1) {

// Выбор операции

printf("\nВыберите операцию:\n");

printf("1. Отождествление вершин\n");

printf("2. Стягивание ребра\n");

printf("3. Расщепление вершины\n");

printf("4. Выход\n");

scanf("%d", &operation);

if (operation == 4) {

break; // Выход из программы

}

switch (operation) {

case 1:

printf("Введите номера вершин для отождествления (0-%d): ", size - 1);

scanf("%d %d", &v1, &v2);

if (v1 < size && v2 < size) {

merge\_vertices(M, size, v1, v2);

printf("\nМатрица после отождествления вершин %d и %d:\n", v1, v2);

print\_adjacency\_matrix(M, size);

}

else {

printf("Неверные номера вершин!\n");

}

break;

case 2:

printf("Введите номера вершин для стягивания ребра (0-%d): ", size - 1);

scanf("%d %d", &v1, &v2);

if (v1 < size && v2 < size) {

collapse\_edge(M, v1, v2);

printf("\nМатрица после стягивания ребра между вершинами %d и %d:\n", v1, v2);

print\_adjacency\_matrix(M, size);

}

else {

printf("Неверные номера вершин!\n");

}

break;

case 3:

printf("Введите номер вершины для расщепления (0-%d): ", size - 1);

scanf("%d", &v1);

if (v1 < size) {

split\_vertex(M, &size, v1);

printf("\nМатрица после расщепления вершины %d:\n", v1);

print\_adjacency\_matrix(M, size);

}

else {

printf("Неверный номер вершины!\n");

}

break;

default:

printf("Некорректный выбор операции!\n");

}

// Вывод списка смежности

print\_adjacency\_list(M, size);

}

return 0;

}

**Результат работы программы**

На рисунке 2 показана реализация задания №2-5.



Рисунок 2 - Результат работы программы №2.



Рисунок 2 - Результат работы программы №3.

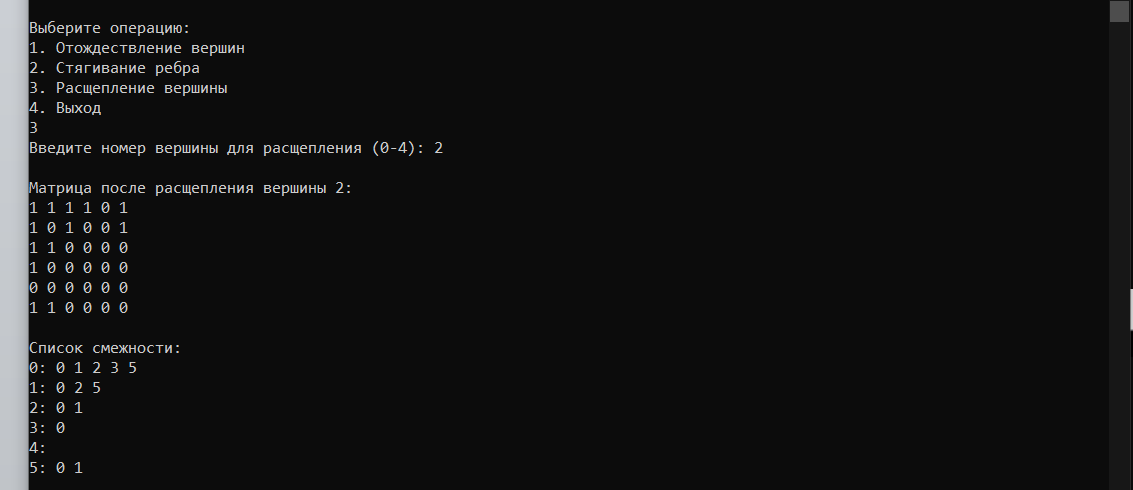


Рисунок 2 - Результат работы программы №4.

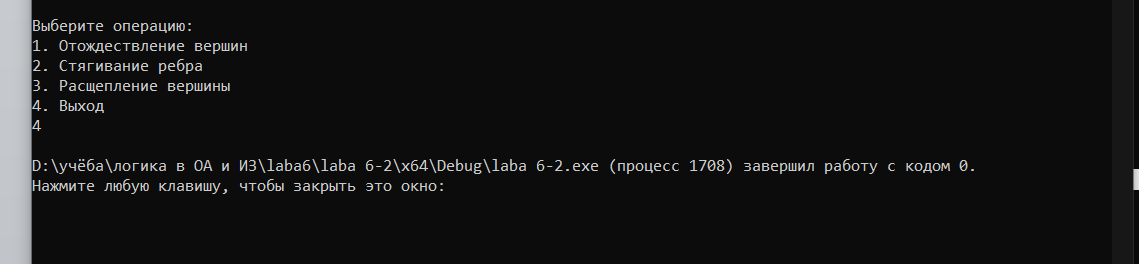


Рисунок 2 - Результат работы программы №5.

**Задание 3**

1. Для матричной формы представления графов выполните операцию:

а) объединения *G* = *G*1 ![Изображение выглядит как черный, темнота

Автоматически созданное описание](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAACcAAAAcCAYAAADiHqZbAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAAIdUAACHVAQSctJ0AAABuSURBVFhH7Y/LCcAwDEO9RPfpJl2th+7ZuJ+AL7nIcilBDwQ5JNKLCSEm4HxTQbpbciiSQ5Fcht/LVQhSOivkqJ1swatrfY55dg9LkP3Rm16aKS4R66CC6DuIODYa3DzxzuL5lDg+yuERQkyOWQORjkmxGiAtDgAAAABJRU5ErkJggg==) *G*2

б) пересечения  *G* = *G*1 ![Изображение выглядит как черный, темнота

Автоматически созданное описание](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAACcAAAAcCAYAAADiHqZbAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAAIdUAACHVAQSctJ0AAABcSURBVFhH7c9BCsAgDETRnL4n8Z5Vt1JUkokI/Q+yK+OvAfiBd+OedkeNAV9Ku9U3cp7H0iMVD6QFqoblgepB2Z46rLs6rpPFZQj/dHhggrgI4ryI88rcBnCQWQVL40PoLLwd7QAAAABJRU5ErkJggg==) *G*2

в) кольцевой суммы *G* = *G*1 ![Изображение выглядит как черный, темнота

Автоматически созданное описание](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAACcAAAAnBAMAAAB+jUwGAAAAAXNSR0IArs4c6QAAAARnQU1BAACxjwv8YQUAAAAYUExURQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFY3HCoAAAAIdFJOUwD/F81D9+chTvvhKwAAAAlwSFlzAAAh1QAAIdUBBJy0nQAAAHVJREFUOMvVkd0NgDAIhLm4QG8ER3EER3AE938xtYVrTWOMT3pJCXz0h4L9TCAZfnVmZqUOgrIhllVp2YnkiSRINxCcUpcpEIKLoO4484jwGQRdb44PX2/qxKh43v9dXVobOOynRz6Pqjyf7cLMds0tpvlxmR1MEQbeirHZ/gAAAABJRU5ErkJggg==) *G*2

Результат выполнения операции выведите на экран.

**ЛИСТИНГ**

**lab6-3.cpp**

﻿#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#define MAX\_VERTICES 10

// Функция для инициализации матрицы смежности

void init\_adjacency\_matrix(int matrix[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

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

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

matrix[i][j] = 0;

}

}

}

// Функция для генерации матрицы смежности

void generate\_adjacency\_matrix(int matrix[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

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

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

// Генерируем случайное значение (0 или 1) для рёбер

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

matrix[j][i] = matrix[i][j]; // Симметричная матрица

}

}

}

// Функция для вывода матрицы смежности

void print\_adjacency\_matrix(int matrix[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

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

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

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

}

printf("\n");

}

}

// Объединение графов G1 и G2

void union\_graphs(int G1[MAX\_VERTICES][MAX\_VERTICES], int G2[MAX\_VERTICES][MAX\_VERTICES], int result[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

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

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

result[i][j] = G1[i][j] | G2[i][j]; // Логическое ИЛИ для объединения

}

}

}

// Пересечение графов G1 и G2

void intersection\_graphs(int G1[MAX\_VERTICES][MAX\_VERTICES], int G2[MAX\_VERTICES][MAX\_VERTICES], int result[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

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

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

result[i][j] = G1[i][j] & G2[i][j]; // Логическое И для пересечения

}

}

}

// Кольцевая сумма графов G1 и G2

void ring\_sum\_graphs(int G1[MAX\_VERTICES][MAX\_VERTICES], int G2[MAX\_VERTICES][MAX\_VERTICES], int result[MAX\_VERTICES][MAX\_VERTICES], int vertices) {

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

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

// Кольцевая сумма: (G1 XOR G2) + (G1 AND G2)

result[i][j] = (G1[i][j] ^ G2[i][j]) | (G1[i][j] && G2[i][j]);

}

}

}

int main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL)); // Инициализация генератора случайных чисел

int vertices;

printf("Введите количество вершин (максимум %d): ", MAX\_VERTICES);

scanf("%d", &vertices);

if (vertices > MAX\_VERTICES) {

printf("Количество вершин превышает максимальное значение.\n");

return 1;

}

int G1[MAX\_VERTICES][MAX\_VERTICES], G2[MAX\_VERTICES][MAX\_VERTICES], result[MAX\_VERTICES][MAX\_VERTICES];

// Инициализация матриц смежности

init\_adjacency\_matrix(G1, vertices);

init\_adjacency\_matrix(G2, vertices);

init\_adjacency\_matrix(result, vertices);

// Генерация матрицы смежности для первого графа

generate\_adjacency\_matrix(G1, vertices);

printf("Сгенерированная матрица смежности для графа G1:\n");

print\_adjacency\_matrix(G1, vertices);

// Генерация матрицы смежности для второго графа

generate\_adjacency\_matrix(G2, vertices);

printf("Сгенерированная матрица смежности для графа G2:\n");

print\_adjacency\_matrix(G2, vertices);

// Объединение графов

union\_graphs(G1, G2, result, vertices);

printf("Результат объединения G1 и G2:\n");

print\_adjacency\_matrix(result, vertices);

// Пересечение графов

intersection\_graphs(G1, G2, result, vertices);

printf("Результат пересечения G1 и G2:\n");

print\_adjacency\_matrix(result, vertices);

// Кольцевая сумма графов

ring\_sum\_graphs(G1, G2, result, vertices);

printf("Результат кольцевой суммы G1 и G2:\n");

print\_adjacency\_matrix(result, vertices);

return 0;

}

**Результат работы программы**

На рисунке 3 показана реализация задания №6

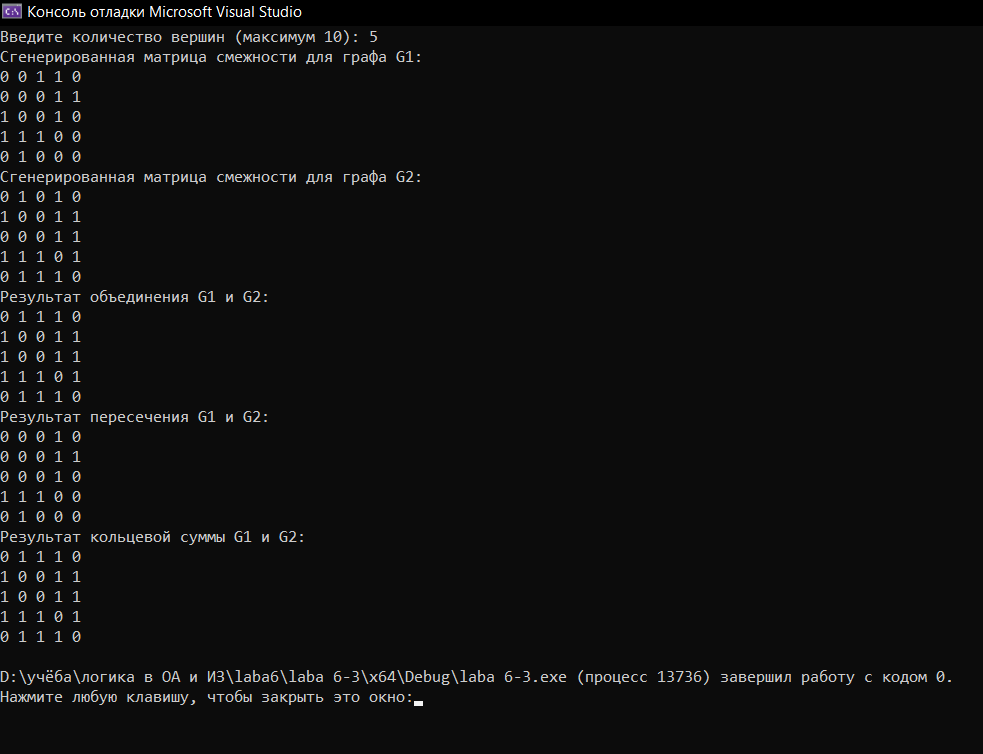


Рисунок 3 - Результат работы программы №6.

**Задание 4 \***

1. Для матричной формы представления графов выполните операцию декартова произведения графов *G = G*1X *G*2.

Результат выполнения операции выведите на экран.

**ЛИСТИНГ**

**lab6-4.cpp**

﻿#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#define MAX\_SIZE 5 // Максимальный размер графа

// Функция для генерации матрицы смежности

void generate\_adjacency\_matrix(int matrix[MAX\_SIZE][MAX\_SIZE], int size) {

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

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

if (i == j) {

matrix[i][j] = 0; // Нет петель

}

else {

matrix[i][j] = rand() % 2; // Случайное значение 0 или 1

}

}

}

// Обеспечиваем симметричность для неориентированного графа

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

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

matrix[j][i] = matrix[i][j]; // Копируем значения

}

}

}

// Функция для вывода матрицы смежности

void print\_adjacency\_matrix(int matrix[MAX\_SIZE][MAX\_SIZE], int size) {

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

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

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

}

printf("\n");

}

}

// Функция для выполнения декартова произведения графов

void cartesian\_product(int M1[MAX\_SIZE][MAX\_SIZE], int size1, int M2[MAX\_SIZE][MAX\_SIZE], int size2, int result[MAX\_SIZE \* MAX\_SIZE][MAX\_SIZE \* MAX\_SIZE]) {

int new\_size = size1 \* size2;

// Инициализация матрицы результата

for (int i = 0; i < new\_size; i++) {

for (int j = 0; j < new\_size; j++) {

result[i][j] = 0;

}

}

// Заполнение матрицы результата

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

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

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

if (M1[i][j] == 1) {

result[i \* size2 + j][i \* size2 + k] = 1; // Связи в первой вершине

}

}

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

if (M2[j][k] == 1) {

result[i \* size2 + j][k \* size2 + j] = 1; // Связи во второй вершине

}

}

}

}

}

// Перегруженная функция для вывода матрицы смежности декартова произведения

void print\_adjacency\_matrix\_large(int matrix[MAX\_SIZE \* MAX\_SIZE][MAX\_SIZE \* MAX\_SIZE], int size1, int size2) {

int new\_size = size1 \* size2;

for (int i = 0; i < new\_size; i++) {

for (int j = 0; j < new\_size; j++) {

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

}

printf("\n");

}

}

int main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL)); // Инициализация генератора случайных чисел

int size1, size2;

int M1[MAX\_SIZE][MAX\_SIZE] = { 0 };

int M2[MAX\_SIZE][MAX\_SIZE] = { 0 };

int result[MAX\_SIZE \* MAX\_SIZE][MAX\_SIZE \* MAX\_SIZE] = { 0 };

// Запрос количества вершин у пользователя для графа G1

printf("Введите количество вершин в графе G1 (макс. %d): ", MAX\_SIZE);

scanf("%d", &size1);

if (size1 > MAX\_SIZE) {

printf("Количество вершин превышает максимальное значение. Установлено на %d.\n", MAX\_SIZE);

size1 = MAX\_SIZE;

}

// Генерация матрицы смежности для графа G1

generate\_adjacency\_matrix(M1, size1);

printf("\nМатрица смежности G1:\n");

print\_adjacency\_matrix(M1, size1);

// Запрос количества вершин у пользователя для графа G2

printf("Введите количество вершин в графе G2 (макс. %d): ", MAX\_SIZE);

scanf("%d", &size2);

if (size2 > MAX\_SIZE) {

printf("Количество вершин превышает максимальное значение. Установлено на %d.\n", MAX\_SIZE);

size2 = MAX\_SIZE;

}

// Генерация матрицы смежности для графа G2

generate\_adjacency\_matrix(M2, size2);

printf("\nМатрица смежности G2:\n");

print\_adjacency\_matrix(M2, size2);

// Выполнение декартова произведения графов

cartesian\_product(M1, size1, M2, size2, result);

printf("\nМатрица смежности декартова произведения G = G1 X G2:\n");

print\_adjacency\_matrix\_large(result, size1, size2);

return 0;

}

**Результат работы программы**

На рисунке 4 показана реализация задания №7



Рисунок 4 - Результат работы программы №7.

**Вывод**

Познакомились с унарными и бинарными операциями над графами, изучили их работу и применили их в лабораторной работе.