Министерство образования и науки Российской Федерации

ФГБОУ ВО «Пензенский государственный университет»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Пенза 2025

**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 24ВВВ1

Егорова К.А.

Винникова Е.С

Марушкин Д.И.

Проверил: к.т.н., доцент

Юрова О.В.

Деев М.В.

### **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

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.

**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*1X *G*2.

Результат выполнения операции выведите на экран.

## **Пояснительный текст к программе**

### **1.1 Цель программы**

Цель программы — изучить работу с графами в **матрицах смежности и списках смежности**, а также освоить основные операции над графами:

* **отождествление вершин** (слияние двух вершин в одну);
* **стягивание ребра** (сведение двух связанных вершин в одну);
* **расщепление вершины** (создание новой вершины с частью связей исходной вершины).

Программа также демонстрирует работу с несколькими графами: объединение, пересечение, кольцевую сумму и декартово произведение.

### **1.2 Структура программы**

Программа реализована на **C++** в виде класса Graph.

**Основные методы класса Graph:**

* genAdjMatrix(int n, double probability) — генерация случайной матрицы смежности;
* print() — вывод матрицы смежности и соответствующего списка смежности;
* otojdestvenie(int v1, int v2) — отождествление двух вершин;
* stjagivanie(int v1, int v2) — стягивание ребра между вершинами;
* rassheplenie(int v) — расщепление вершины;
* unite, intersection, xorSum, dekart — операции над двумя графами.

**Функция main():**

* Организует меню программы для работы с графами.
* Позволяет создавать два графа, преобразовывать их, выполнять операции и выводить результаты на экран.

### **1.3 Используемые средства**

* **Язык:** C++
* **Библиотеки:** <iostream>, <vector>, <random>, <iomanip>
* **Методы реализации:**
  + работа с двумерными динамическими массивами;
  + генерация случайных чисел;
  + реализация основных операций с графами в матрицах и списках смежности;
  + принципы ООП (инкапсуляция, методы класса).

## **2. Описание методов класса Graph**

### **2.1 genAdjMatrix(int n, double probability)**

Генерирует **матрицу смежности** графа с n вершинами. Для каждой пары вершин (i, j) случайным образом решает, будет ли ребро, и создает **неориентированное ребро** (matrix[i][j] = matrix[j][i] = 1). После генерации обновляет число рёбер и создаёт список смежности.

### **2.2 print()**

Выводит граф в двух формах:

* **Матрица смежности** — связи между вершинами;
* **Список смежности** — для каждой вершины перечислены смежные вершины.

Также выводится количество вершин и рёбер.

### **2.3 genAdjList()**

Преобразует **матрицу смежности в список смежности**: для каждой вершины добавляются все смежные вершины в отдельный список.

### **2.4 countEdges()**

Подсчитывает общее количество рёбер, проходя по верхней треугольной части матрицы (так как граф неориентированный).

### **2.5 otojdestvenie(int v1, int v2)**

Отождествляет вершины v1 и v2:

* Объединяет их связи в одну вершину;
* Пересчитывает матрицу смежности и число рёбер;
* Обновляет список смежности.

### **2.6 stjagivanie(int v1, int v2)**

Стягивает ребро между вершинами v1 и v2:

* Проверяет наличие ребра;
* Вызывает **otojdestvenie(v1, v2),** чтобы объединить вершины.

### **2.7 rassheplenie(int v)**

Создаёт новую вершину, частично копируя связи вершины v:

* Генерируется новая вершина;
* Случайным образом половина связей переносится на новую вершину;
* Обновляется матрица и список смежности.

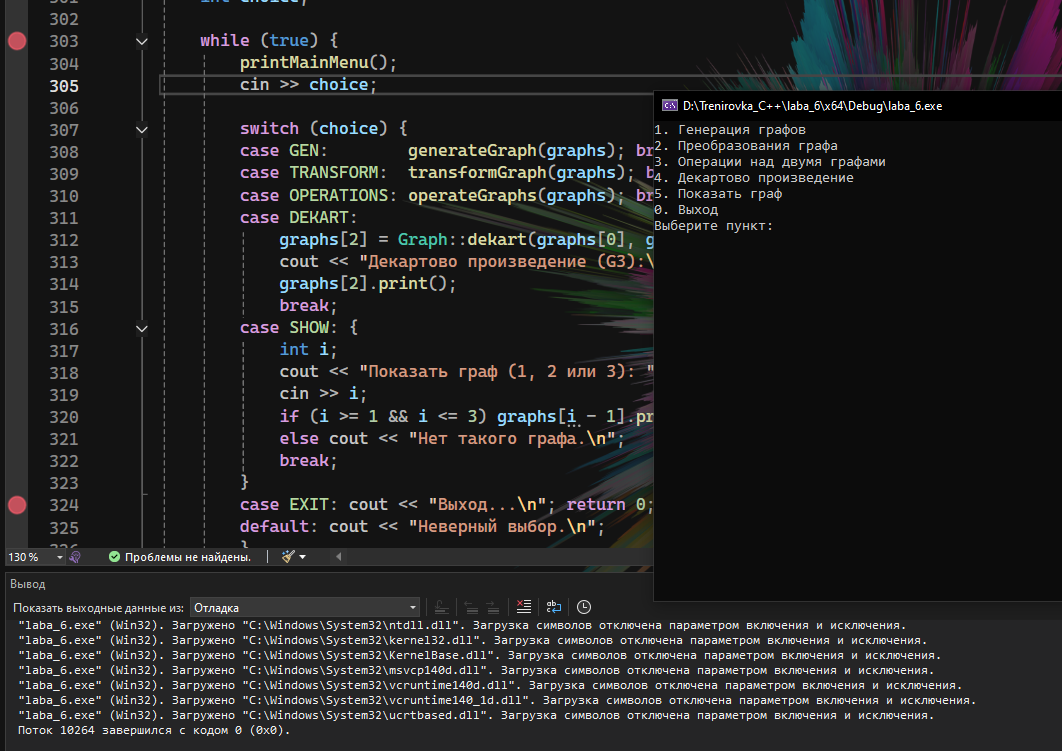
### **2.8 Методы для работы с двумя графами**

* **unite(g1, g2)** — объединение графов;
* **intersection(g1, g2)** — пересечение;
* **xorSum(g1, g2)** — кольцевая сумма (XOR);

**2.9 Декартово произведение графов**

* **dekart(g1, g2)** — декартово произведение: создаётся граф с вершинами, соответствующими парам (v1, v2) из обоих графов, а ребро соединяет пары в соответствии с правилами произведения.

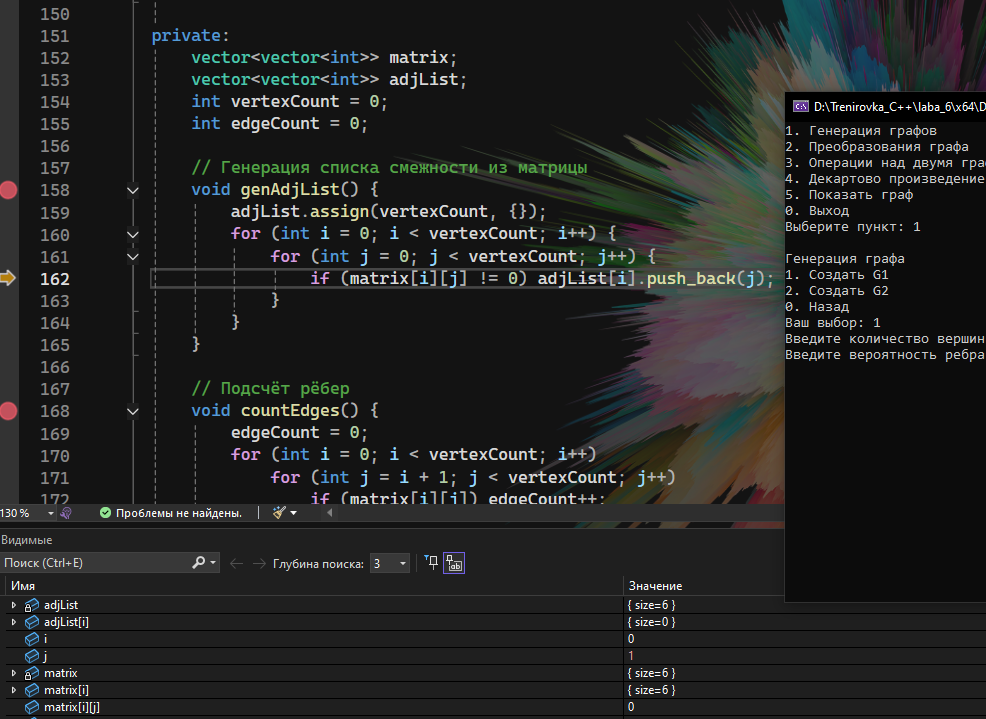
## **Трассировка программы**



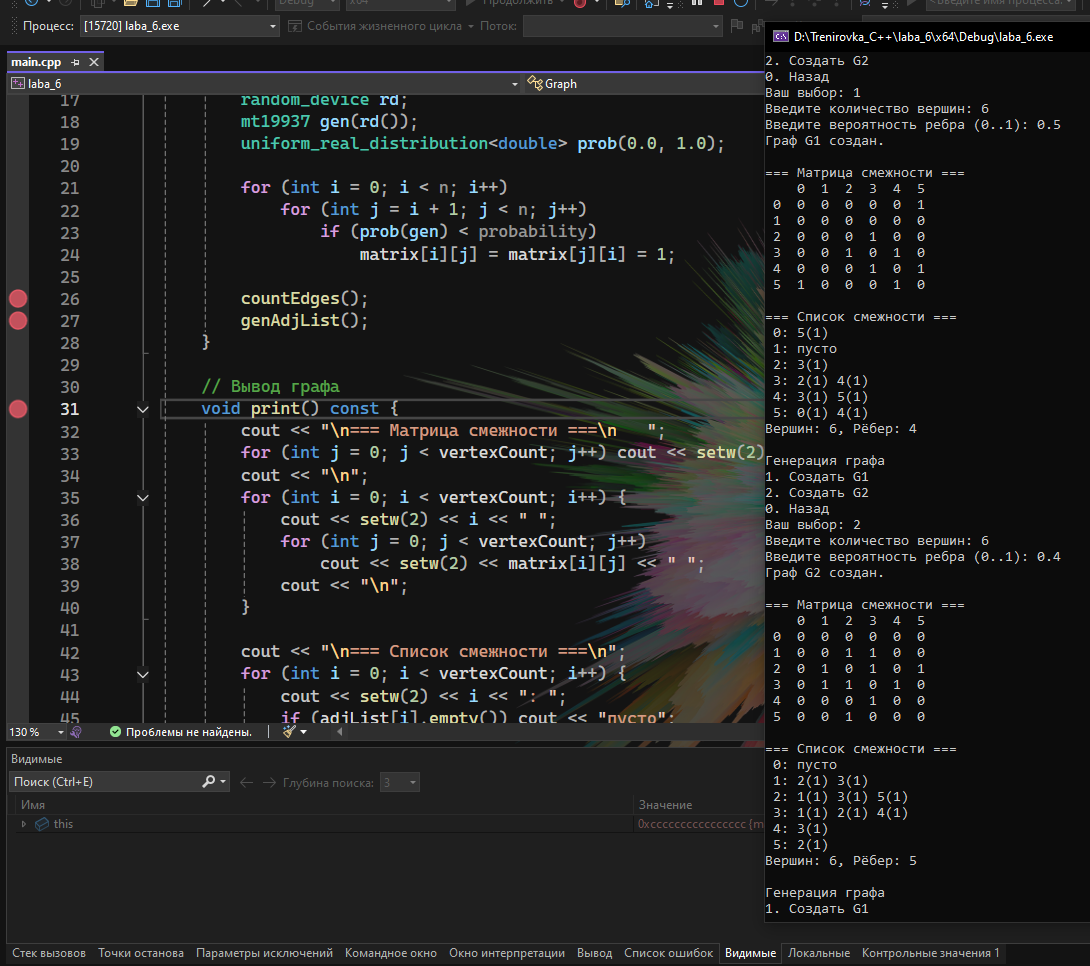
**Рисунок 1** – запуск программы и отображение главного меню.



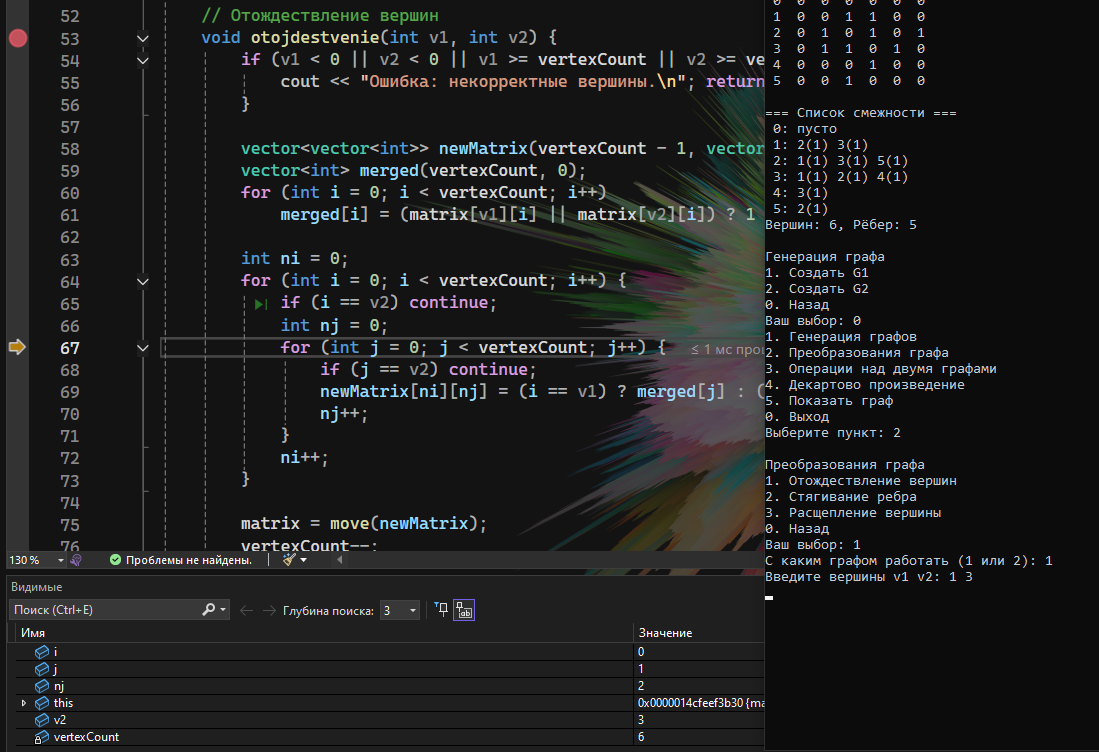
**Рисунок 2** – генерация графов G1 и G2.



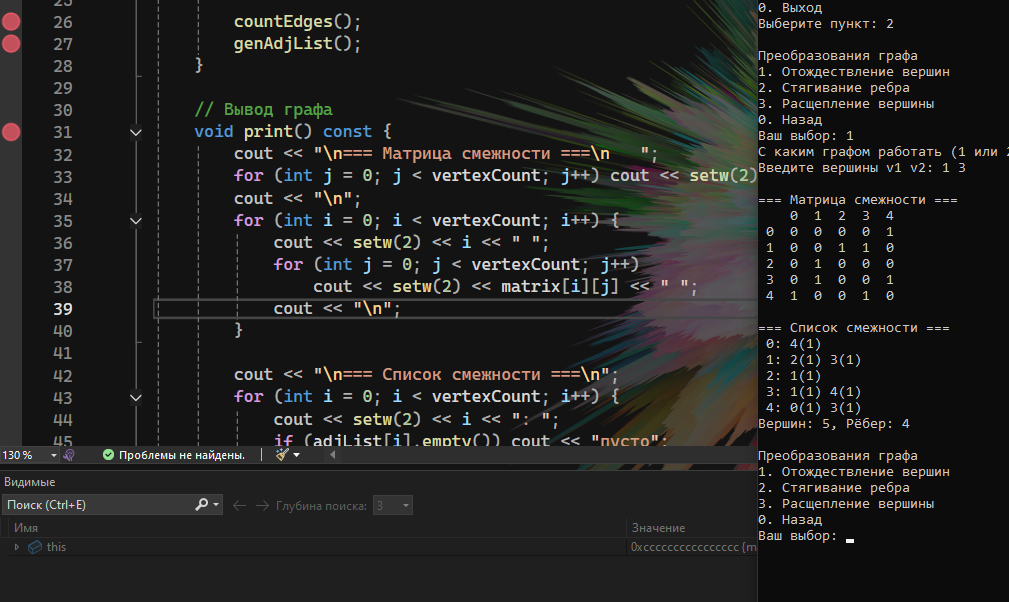
**Рисунок 3** – преобразование матриц смежности в списки смежности.



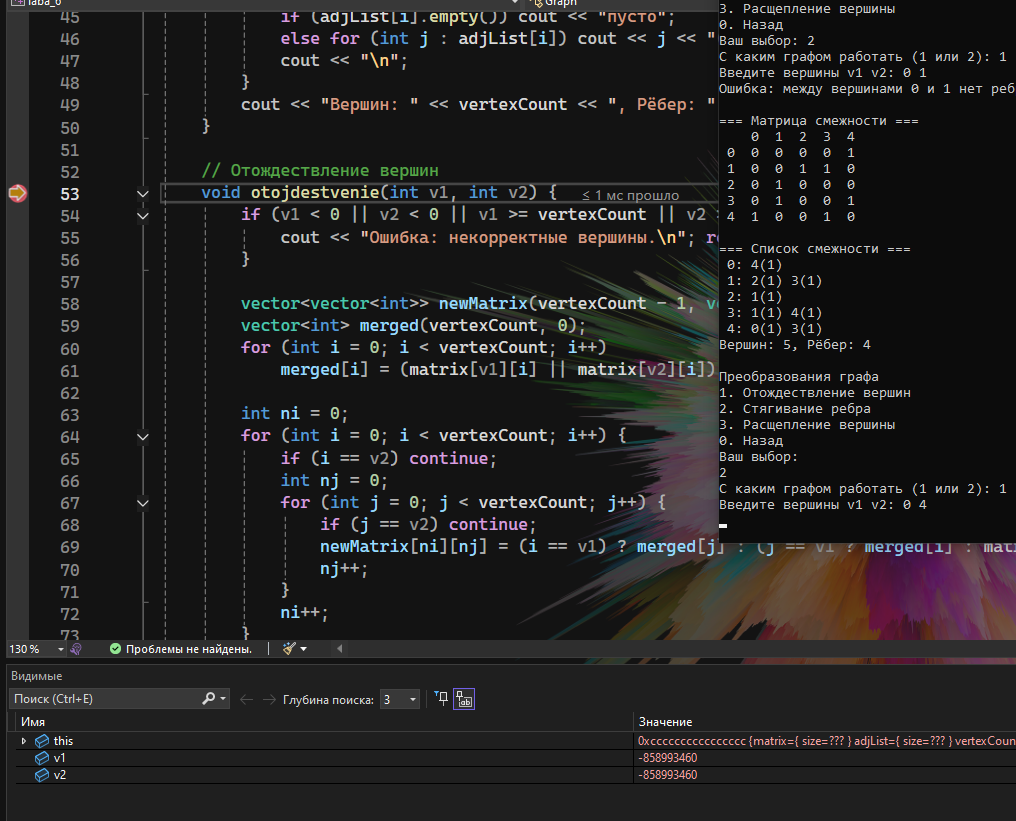
**Рисунок 4** – вывод матрицы и списка смежности для графов G1 и G2.



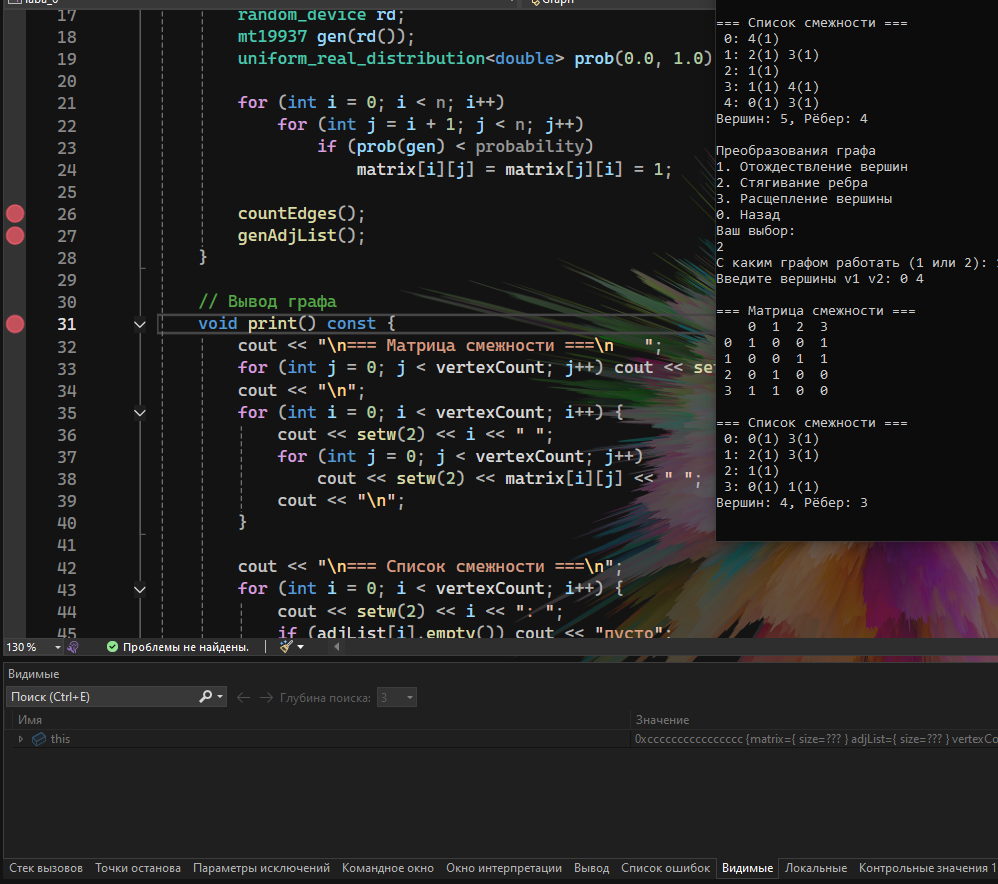
**Рисунок 5** – выполнение операции **отождествления вершин**.



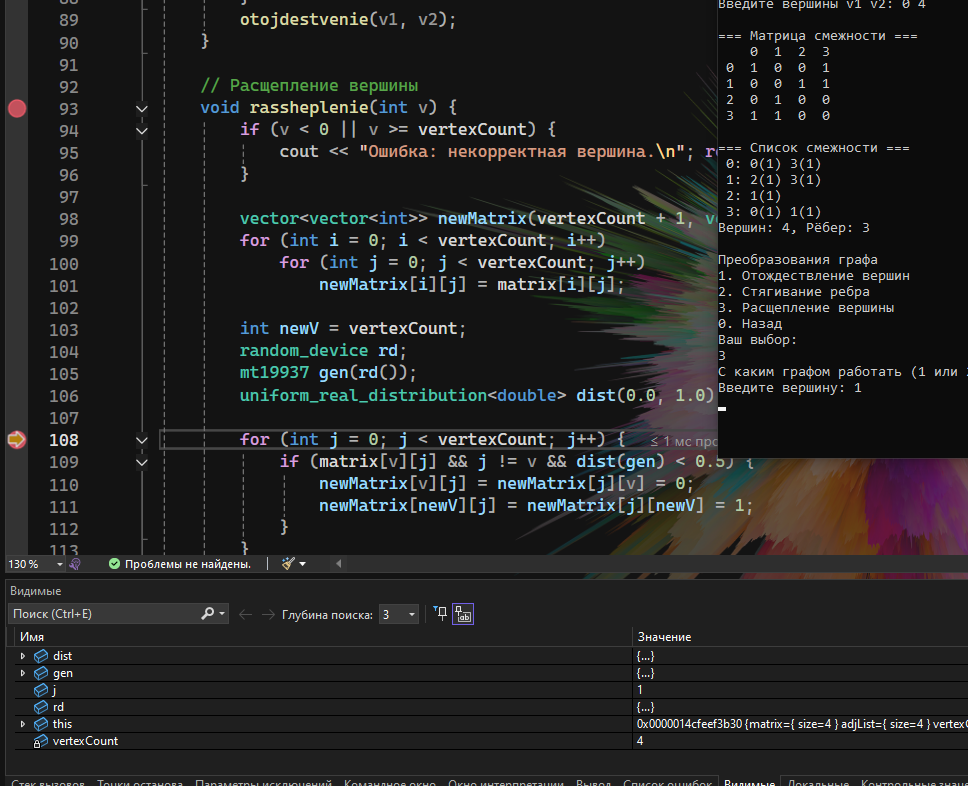
**Рисунок 6** – вывод результата.



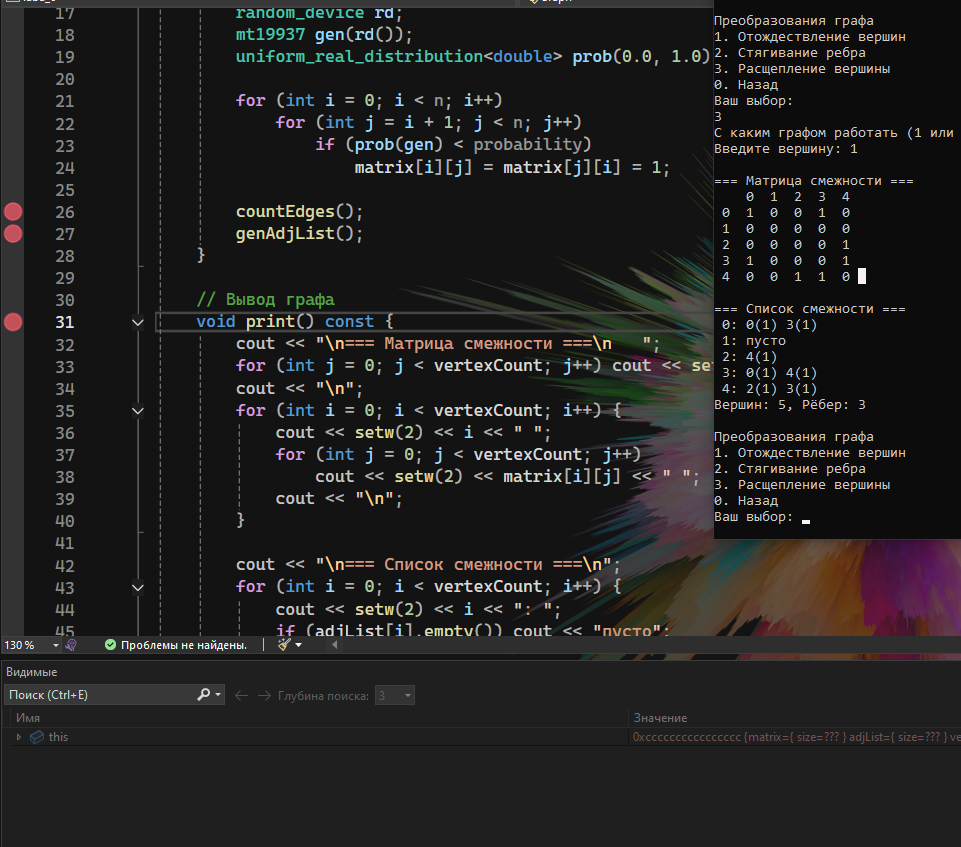
**Рисунок 7** – выполнение операции **стягивания ребра**.



**Рисунок 8** – вывод результата.



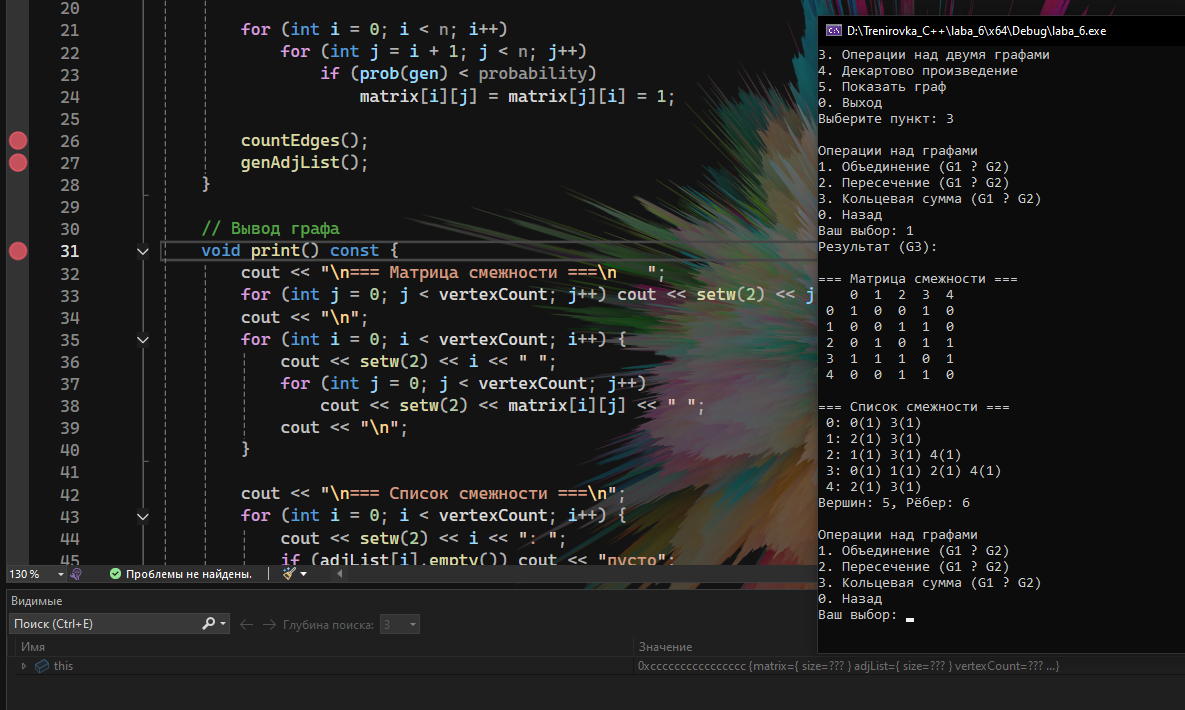
**Рисунок 9** – выполнение операции **расщепления вершины**.



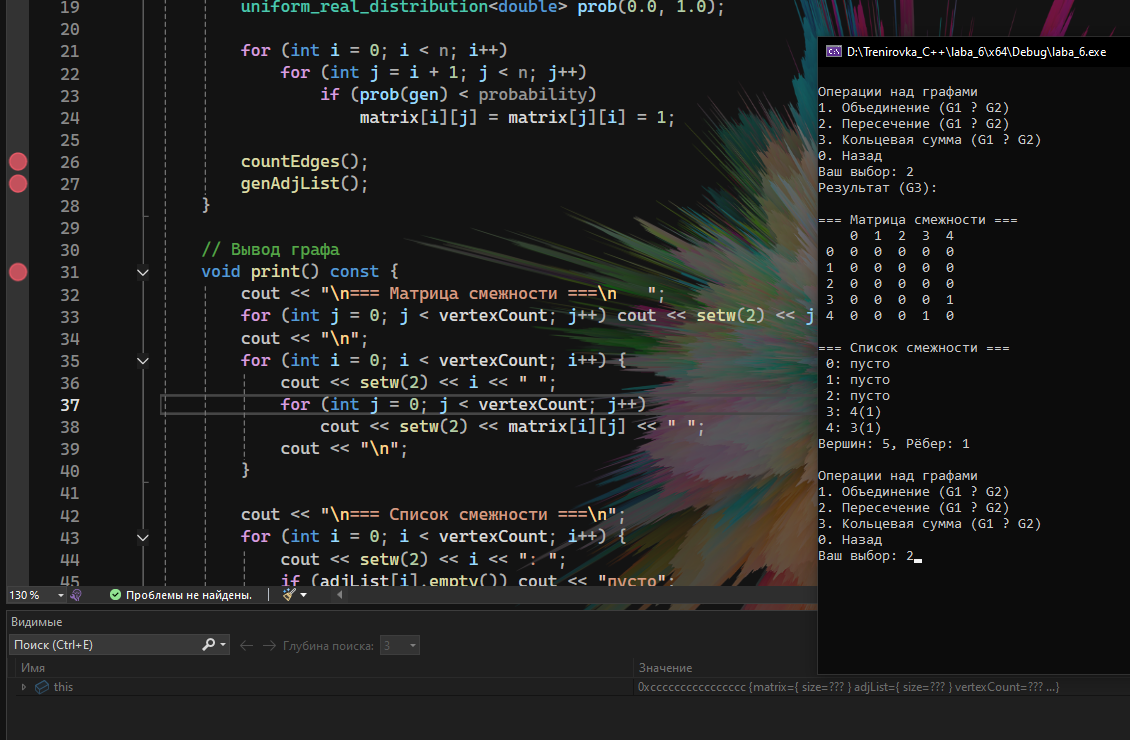
**Рисунок 10** – вывод результата.



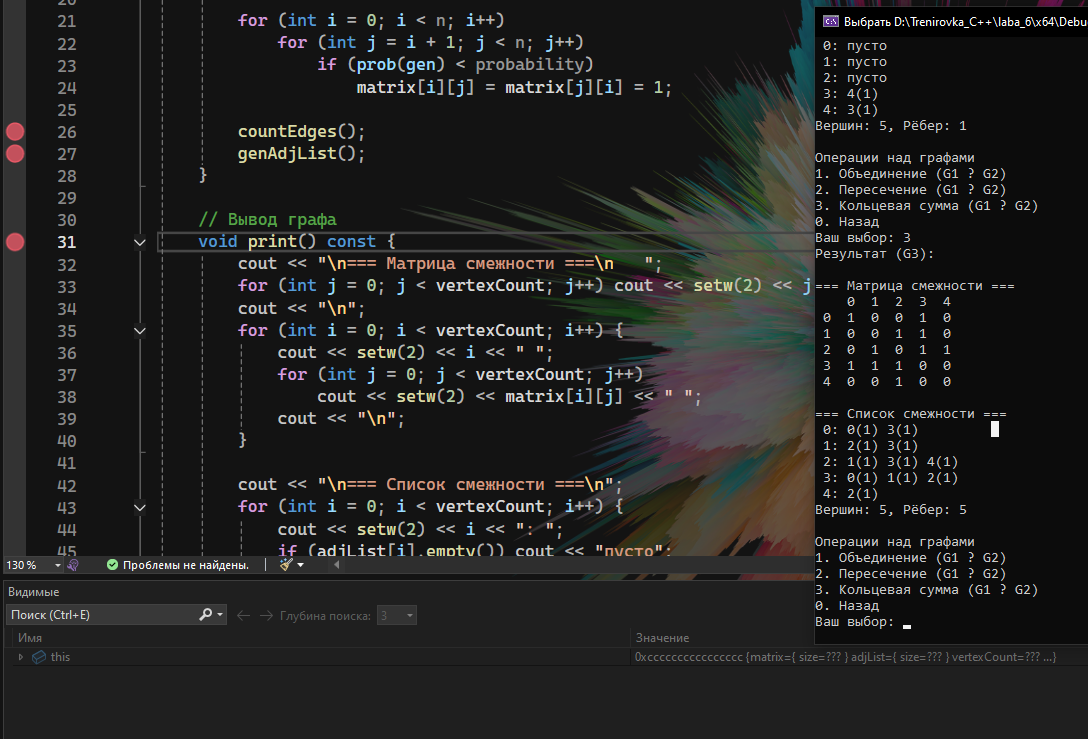
**Рисунок 11** – выполнение операций над двумя графами: объединение, пересечение, кольцевая сумма.

****

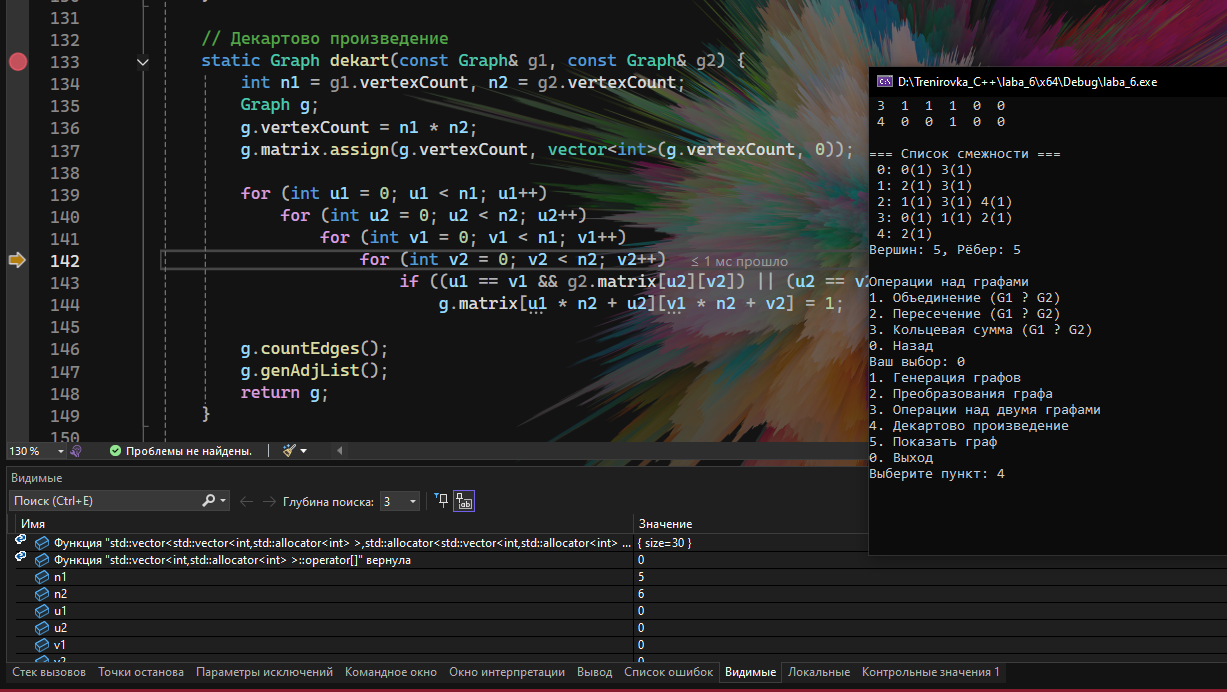
**Рисунок 12** – результат объединения.



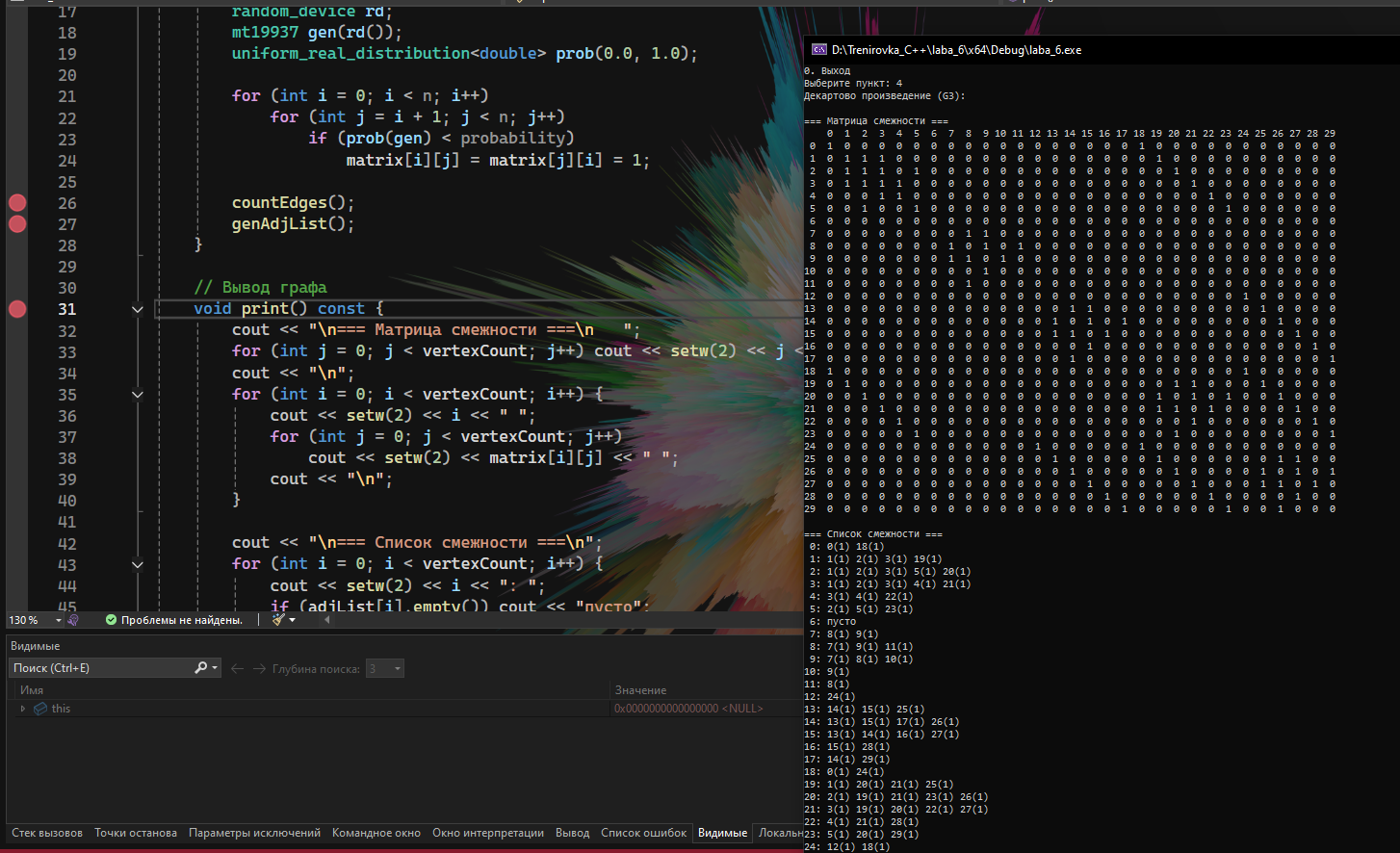
**Рисунок 13** – результат пересечения.



**Рисунок 14** – результат кольцевой суммы.

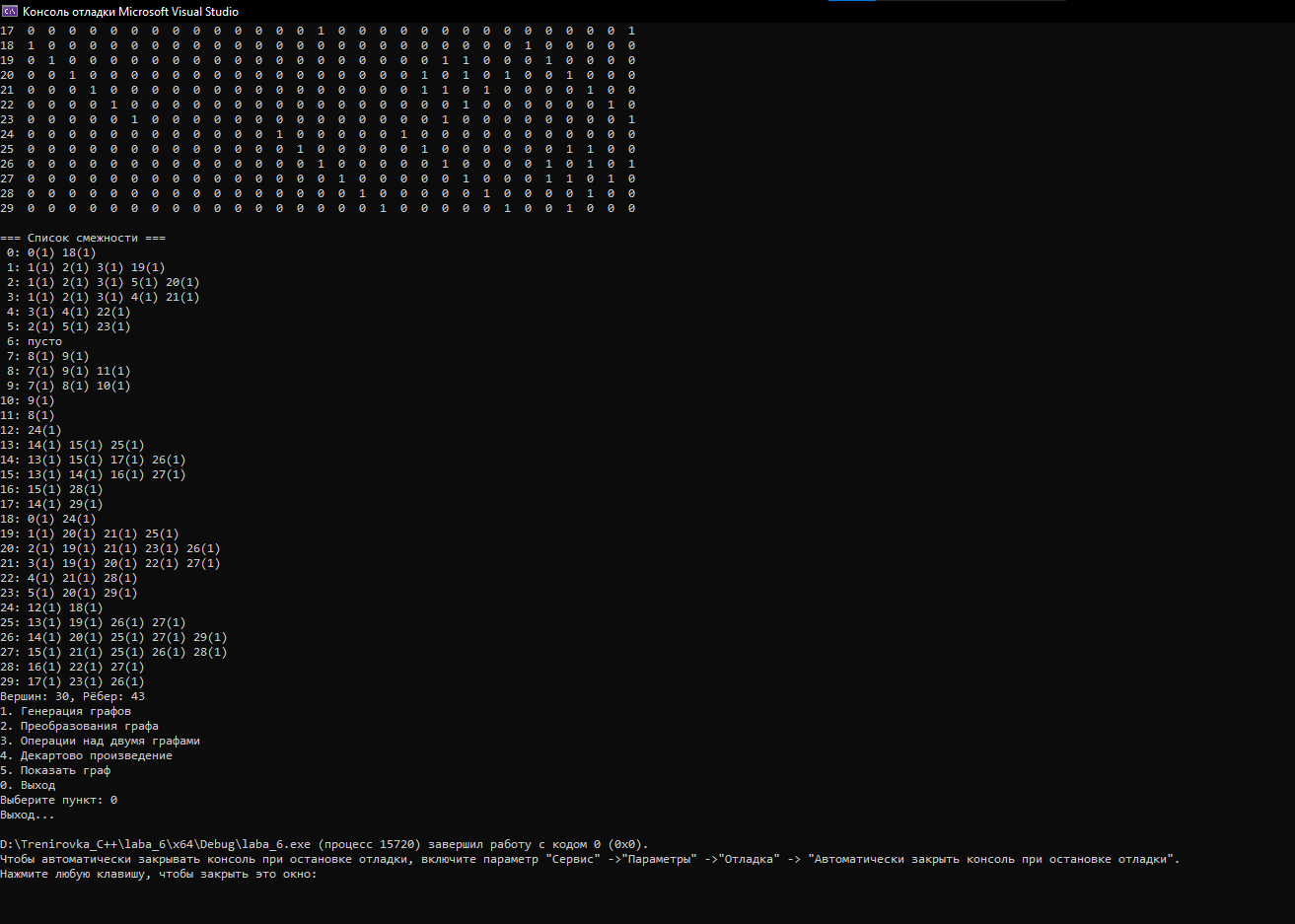


**Рисунок 15** – декартово произведение графов G1 и G2.



**Рисунок 15** – результат декартово произведения.

**Результат работы программы:**



**Рисунок 16** – результат работы программы.

**Вывод:**

В ходе лабораторной работы я изучил различные способы представления графов матрицей смежности и списком смежности а так же освоил основные операции над ними: отождествление вершин, стягивание ребра и расщепление вершины. Также я научился выполнять объединение, пересечение, кольцевую сумму и декартово произведение графов.

**Трассировка программы:**

#include <iostream>

#include <vector>

#include <random>

#include <iomanip>

using namespace std;

class Graph {

public:

Graph() = default;

Graph(int n, double probability) { genAdjMatrix(n, probability); }

// Генерация матрицы смежности

void genAdjMatrix(int n, double probability) {

vertexCount = n;

matrix.assign(n, vector<int>(n, 0));

random\_device rd;

mt19937 gen(rd());

uniform\_real\_distribution<double> prob(0.0, 1.0);

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

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

if (prob(gen) < probability)

matrix[i][j] = matrix[j][i] = 1;

countEdges();

genAdjList();

}

// Вывод графа

void print() const {

cout << "\n=== Матрица смежности ===\n ";

for (int j = 0; j < vertexCount; j++) cout << setw(2) << j << " ";

cout << "\n";

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

cout << setw(2) << i << " ";

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

cout << setw(2) << matrix[i][j] << " ";

cout << "\n";

}

cout << "\n=== Список смежности ===\n";

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

cout << setw(2) << i << ": ";

if (adjList[i].empty()) cout << "пусто";

else for (int j : adjList[i]) cout << j << "(" << matrix[i][j] << ") ";

cout << "\n";

}

cout << "Вершин: " << vertexCount << ", Рёбер: " << edgeCount << "\n";

}

// Отождествление вершин

void otojdestvenie(int v1, int v2) {

if (v1 < 0 || v2 < 0 || v1 >= vertexCount || v2 >= vertexCount || v1 == v2) {

cout << "Ошибка: некорректные вершины.\n"; return;

}

vector<vector<int>> newMatrix(vertexCount - 1, vector<int>(vertexCount - 1, 0));

vector<int> merged(vertexCount, 0);

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

merged[i] = (matrix[v1][i] || matrix[v2][i]) ? 1 : 0;

int ni = 0;

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

if (i == v2) continue;

int nj = 0;

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

if (j == v2) continue;

newMatrix[ni][nj] = (i == v1) ? merged[j] : (j == v1 ? merged[i] : matrix[i][j]);

nj++;

}

ni++;

}

matrix = move(newMatrix);

vertexCount--;

countEdges();

genAdjList();

}

// Стягивание ребра

void stjagivanie(int v1, int v2) {

if (v1 < 0 || v2 < 0 || v1 >= vertexCount || v2 >= vertexCount || v1 == v2) {

cout << "Ошибка: некорректные вершины.\n"; return;

}

if (!matrix[v1][v2]) {

cout << "Ошибка: между вершинами " << v1 << " и " << v2 << " нет ребра.\n"; return;

}

otojdestvenie(v1, v2);

}

// Расщепление вершины

void rassheplenie(int v) {

if (v < 0 || v >= vertexCount) {

cout << "Ошибка: некорректная вершина.\n"; return;

}

vector<vector<int>> newMatrix(vertexCount + 1, vector<int>(vertexCount + 1, 0));

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

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

newMatrix[i][j] = matrix[i][j];

int newV = vertexCount;

random\_device rd;

mt19937 gen(rd());

uniform\_real\_distribution<double> dist(0.0, 1.0);

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

if (matrix[v][j] && j != v && dist(gen) < 0.5) {

newMatrix[v][j] = newMatrix[j][v] = 0;

newMatrix[newV][j] = newMatrix[j][newV] = 1;

}

}

matrix = move(newMatrix);

vertexCount++;

countEdges();

genAdjList();

}

// Объединение, пересечение, XOR

static Graph unite(const Graph& g1, const Graph& g2) {

return combine(g1, g2, [](int a, int b) { return a || b; });

}

static Graph intersection(const Graph& g1, const Graph& g2) {

return combine(g1, g2, [](int a, int b) { return a && b; });

}

static Graph xorSum(const Graph& g1, const Graph& g2) {

return combine(g1, g2, [](int a, int b) { return a != b; });

}

// Декартово произведение

static Graph dekart(const Graph& g1, const Graph& g2) {

int n1 = g1.vertexCount, n2 = g2.vertexCount;

Graph g;

g.vertexCount = n1 \* n2;

g.matrix.assign(g.vertexCount, vector<int>(g.vertexCount, 0));

for (int u1 = 0; u1 < n1; u1++)

for (int u2 = 0; u2 < n2; u2++)

for (int v1 = 0; v1 < n1; v1++)

for (int v2 = 0; v2 < n2; v2++)

if ((u1 == v1 && g2.matrix[u2][v2]) || (u2 == v2 && g1.matrix[u1][v1]))

g.matrix[u1 \* n2 + u2][v1 \* n2 + v2] = 1;

g.countEdges();

g.genAdjList();

return g;

}

private:

vector<vector<int>> matrix;

vector<vector<int>> adjList;

int vertexCount = 0;

int edgeCount = 0;

// Генерация списка смежности из матрицы

void genAdjList() {

adjList.assign(vertexCount, {});

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

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

if (matrix[i][j] != 0) adjList[i].push\_back(j);

}

}

}

// Подсчёт рёбер

void countEdges() {

edgeCount = 0;

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

for (int j = i + 1; j < vertexCount; j++)

if (matrix[i][j]) edgeCount++;

}

// Универсальная операция над двумя графами

static Graph combine(const Graph& g1, const Graph& g2, auto op) {

int n = min(g1.vertexCount, g2.vertexCount);

Graph g;

g.vertexCount = n;

g.matrix.assign(n, vector<int>(n, 0));

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

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

g.matrix[i][j] = op(g1.matrix[i][j], g2.matrix[i][j]);

g.countEdges();

g.genAdjList();

return g;

}

};

enum MainMenu {

GEN = 1, TRANSFORM, OPERATIONS, DEKART, SHOW, EXIT = 0

};

void printMainMenu() {

cout << "1. Генерация графов\n"

<< "2. Преобразования графа\n"

<< "3. Операции над двумя графами\n"

<< "4. Декартово произведение\n"

<< "5. Показать граф\n"

<< "0. Выход\n"

<< "Выберите пункт: ";

}

int selectGraph() {

int g;

cout << "С каким графом работать (1 или 2): ";

cin >> g;

if (g < 1 || g > 2) {

cout << "Ошибка: нет такого графа.\n";

return -1;

}

return g - 1;

}

void generateGraph(vector<Graph>& graphs) {

while (true) {

cout << "\nГенерация графа\n"

<< "1. Создать G1\n"

<< "2. Создать G2\n"

<< "0. Назад\n"

<< "Ваш выбор: ";

int a; cin >> a;

if (a == 0) break;

int n; double p;

cout << "Введите количество вершин: "; cin >> n;

cout << "Введите вероятность ребра (0..1): "; cin >> p;

int index = (a == 1 ? 0 : 1);

graphs[index].genAdjMatrix(n, p);

cout << "Граф G" << (index + 1) << " создан.\n";

graphs[index].print();

}

}

void transformGraph(vector<Graph>& graphs) {

while (true) {

cout << "\nПреобразования графа\n"

<< "1. Отождествление вершин\n"

<< "2. Стягивание ребра\n"

<< "3. Расщепление вершины\n"

<< "0. Назад\n"

<< "Ваш выбор: ";

int a; cin >> a;

if (a == 0) break;

int idx = selectGraph();

if (idx == -1) continue;

int v1, v2, v;

switch (a) {

case 1:

cout << "Введите вершины v1 v2: ";

cin >> v1 >> v2;

graphs[idx].otojdestvenie(v1, v2);

break;

case 2:

cout << "Введите вершины v1 v2: ";

cin >> v1 >> v2;

graphs[idx].stjagivanie(v1, v2);

break;

case 3:

cout << "Введите вершину: ";

cin >> v;

graphs[idx].rassheplenie(v);

break;

default: cout << "Неверный выбор.\n";

}

graphs[idx].print();

}

}

void operateGraphs(vector<Graph>& graphs) {

while (true) {

cout << "\nОперации над графами\n"

<< "1. Объединение (G1 ∪ G2)\n"

<< "2. Пересечение (G1 ∩ G2)\n"

<< "3. Кольцевая сумма (G1 ⊕ G2)\n"

<< "0. Назад\n"

<< "Ваш выбор: ";

int a; cin >> a;

if (a == 0) break;

switch (a) {

case 1: graphs[2] = Graph::unite(graphs[0], graphs[1]); break;

case 2: graphs[2] = Graph::intersection(graphs[0], graphs[1]); break;

case 3: graphs[2] = Graph::xorSum(graphs[0], graphs[1]); break;

default: cout << "Неверный выбор.\n"; continue;

}

cout << "Результат (G3):\n";

graphs[2].print();

}

}

int main() {

setlocale(LC\_ALL, "Russian");

vector<Graph> graphs(3);

int choice;

while (true) {

printMainMenu();

cin >> choice;

switch (choice) {

case GEN: generateGraph(graphs); break;

case TRANSFORM: transformGraph(graphs); break;

case OPERATIONS: operateGraphs(graphs); break;

case DEKART:

graphs[2] = Graph::dekart(graphs[0], graphs[1]);

cout << "Декартово произведение (G3):\n";

graphs[2].print();

break;

case SHOW: {

int i;

cout << "Показать граф (1, 2 или 3): ";

cin >> i;

if (i >= 1 && i <= 3) graphs[i - 1].print();

else cout << "Нет такого графа.\n";

break;

}

case EXIT: cout << "Выход...\n"; return 0;

default: cout << "Неверный выбор.\n";

}

}

return 0;

}