**Министерство науки и высшего 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 студ. группы 24ВВВ4:

Агуреев А.П.

Невежин И.С.

Приняли:

к.э.н. доцент Акифьев И.В.\_\_\_\_

к.т.н доцент Юрова О.В.\_\_\_\_

Пенза 2025

**Цель работы:**

Изучить и практически освоить унарные и бинарные операции над графами.

**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

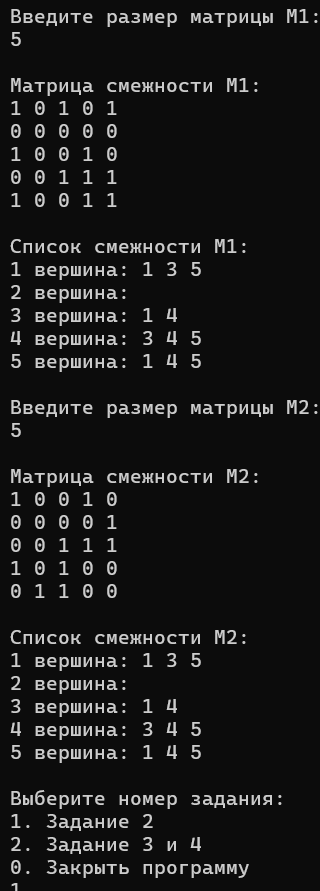
Результат выполнения операции выведите на экран.

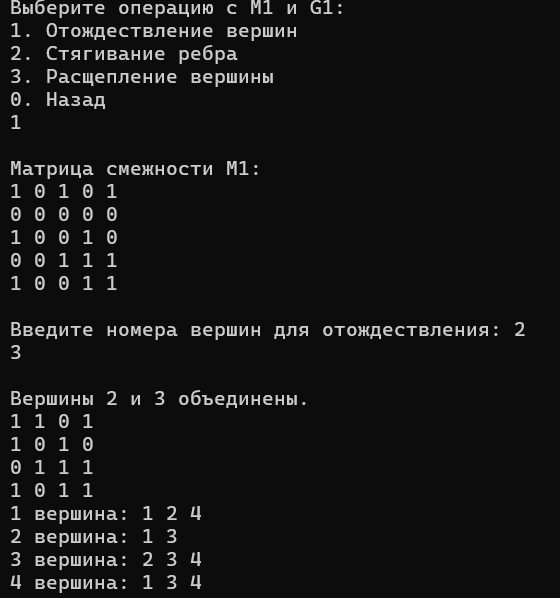
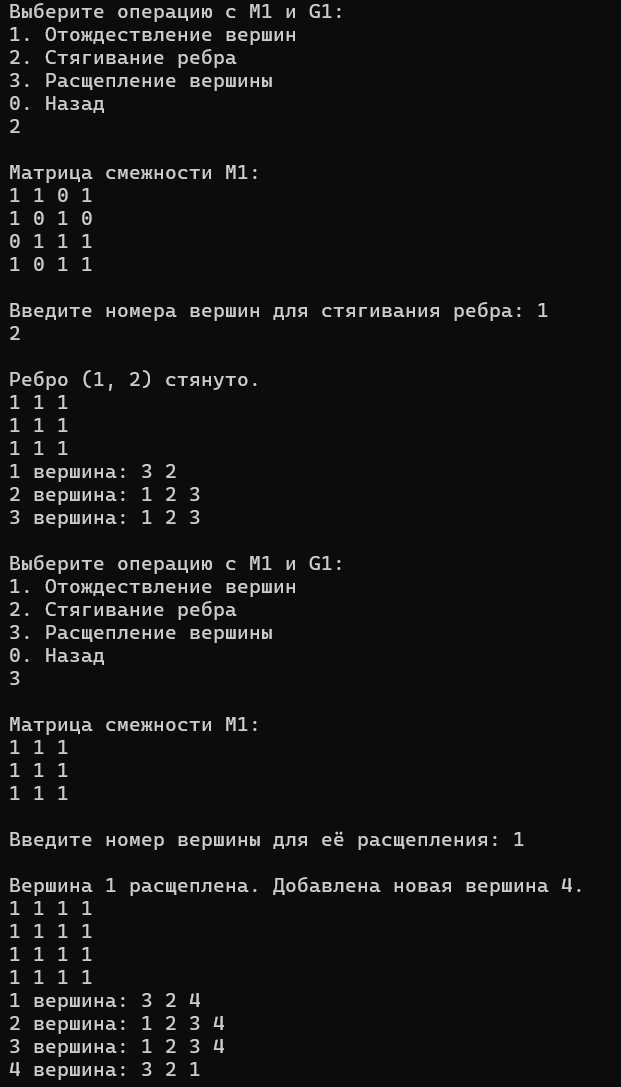
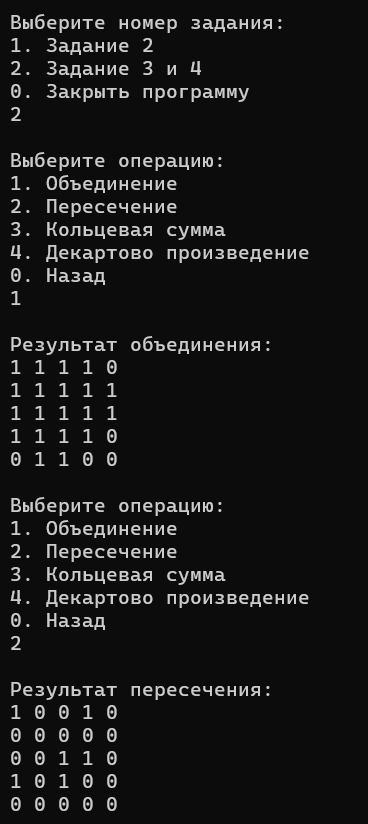
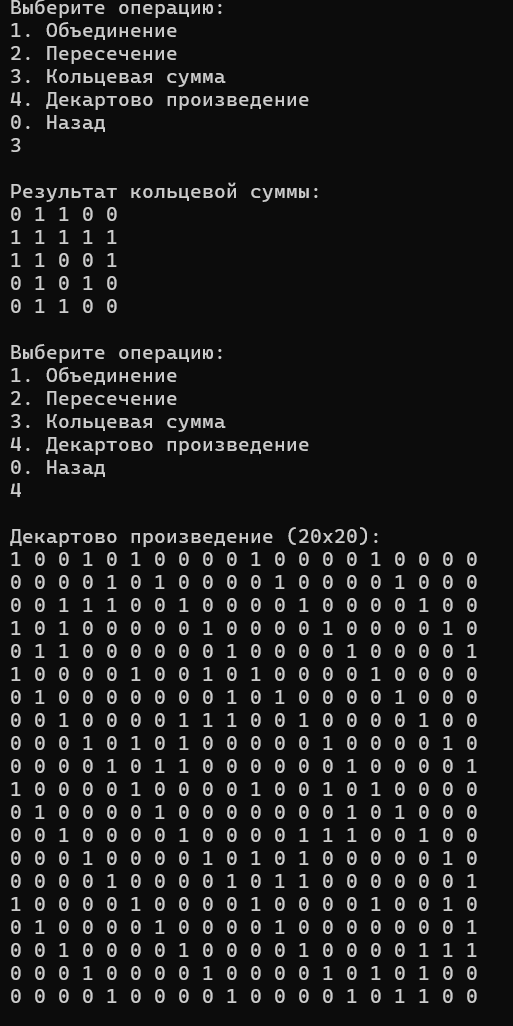
**Задание 4 \***

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

Результат выполнения операции выведите на экран.

**Результат работы программы:**

****

**** ****  

**Вывод:**

Изучили и практически освоили унарные и бинарные операции над графами.

**Листинг:**

#include <iostream>

#include <vector>

#include <set>

using namespace std;

struct node {

int index;

struct node\* next\_node;

};

struct Graph {

int size;

vector<node\*> vertexes;

};

void remove\_node(Graph\* G, int i, int j) {

node\* tmp = G->vertexes[i];

node\* prev = nullptr;

while (tmp != nullptr) {

if (tmp->index == j) {

node\* el\_to\_delete = tmp;

if (prev == nullptr) {

G->vertexes[i] = tmp->next\_node;

tmp = G->vertexes[i];

}

else {

prev->next\_node = tmp->next\_node;

tmp = tmp->next\_node;

}

free(el\_to\_delete);

}

else {

prev = tmp;

tmp = tmp->next\_node;

}

}

}

void add\_el(struct Graph\* G, int i, int j) {

bool repeat = false;

node\* p = (node\*)malloc(sizeof(node));

p->index = j;

if (G->vertexes[i] == nullptr) {

G->vertexes[i] = p;

p->next\_node = nullptr;

}

else {

node\* tmp = G->vertexes[i];

int k = 0;

while (tmp->next\_node != nullptr) {

node\* tmp\_2 = tmp->next\_node;

while (tmp\_2->next\_node != nullptr) {

if (tmp->index == tmp\_2->index) {

return;

}

tmp\_2 = tmp\_2->next\_node;

}

tmp = tmp->next\_node;

}

tmp->next\_node = p;

p->next\_node = nullptr;

}

}

void add\_new\_el(Graph\* G, int i, int j) {

bool has\_note = false;

node\* tmp = G->vertexes[i];

while (tmp != nullptr) {

if (tmp->index == j) {

has\_note = true;

}

tmp = tmp->next\_node;

}

if (has\_note == false) {

add\_el(G, i, j);

}

}

void remove\_copies(Graph\* G) {

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

set<int> seen;

node\* current = G->vertexes[i];

while (current != nullptr) {

int index = current->index;

current = current->next\_node;

if (seen.count(index))

remove\_node(G, i, index);

else

seen.insert(index);

}

}

}

void print\_list(Graph\* G) {

for (int i = 0; i < G->vertexes.size(); i++) {

node\* tmp = G->vertexes[i];

cout << i + 1 << " вершина: ";

while (tmp != nullptr) {

cout << tmp->index + 1 << " ";

tmp = tmp->next\_node;

}

cout << '\n';

}

}

//отождествление

void mergeVerticesMatrix(vector<vector<int>>& M, Graph\* G, int vertex1, int vertex2)

{

//Массив

vertex1--; vertex2--;

int n = M.size();

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

M[vertex1][i] = M[vertex1][i] | M[vertex2][i];

M[i][vertex1] = M[vertex1][i];

}

M.erase(M.begin() + vertex2);

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

M[i].erase(M[i].begin() + vertex2);

}

//Список

node\* tmp = G->vertexes[vertex2];

while (tmp != nullptr) {

if (tmp->index != vertex1) {

add\_new\_el(G, vertex1, tmp->index);

}

tmp = tmp->next\_node;

}

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

node\* current\_el = G->vertexes[i];

while (current\_el != nullptr) {

if (current\_el->index == vertex2)

current\_el->index = vertex1;

if (current\_el->index > vertex2)

current\_el->index--;

current\_el = current\_el->next\_node;

}

}

G->vertexes.erase(G->vertexes.begin() + vertex2);

G->size--;

remove\_copies(G);

cout << "\nВершины " << vertex1 + 1 << " и " << vertex2 + 1 << " объединены.\n";

}

//стягивание

void contractEdgeMatrix(vector<vector<int>>& M, Graph\* G, int vertex1, int vertex2)

{

//Массив

vertex1--; vertex2--;

int n = M.size();

if (vertex1 >= n || vertex2 >= n || vertex1 < 0 || vertex2 < 0) {

cout << "\nОшибка: вершины вне диапазона!\n";

return;

}

if (M[vertex1][vertex2] == 0) {

cout << "Между вершинами " << vertex1 + 1 << " и " << vertex2 + 1 << " нет ребра!\n";

return;

}

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

M[vertex1][i] = M[vertex1][i] | M[vertex2][i];

M[i][vertex1] = M[vertex1][i];

}

M.erase(M.begin() + vertex2);

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

M[i].erase(M[i].begin() + vertex2);

}

//Список

node\* tmp = G->vertexes[vertex2];

while (tmp != nullptr) {

if (tmp->index != vertex1) {

add\_new\_el(G, vertex1, tmp->index);

}

tmp = tmp->next\_node;

}

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

node\* current\_el = G->vertexes[i];

while (current\_el != nullptr) {

if (current\_el->index == vertex2)

current\_el->index = vertex1;

if (current\_el->index > vertex2)

current\_el->index--;

current\_el = current\_el->next\_node;

}

}

G->vertexes.erase(G->vertexes.begin() + vertex2);

G->size--;

remove\_copies(G);

cout << "\nРебро (" << vertex1 + 1 << ", " << vertex2 + 1 << ") стянуто.\n";

}

//расщепление

void splitVertexMatrix(vector<vector<int>>& M, Graph\* G, int vertex1)

{

//Массив

vertex1--;

int n = M.size();

if (vertex1 >= n || vertex1 < 0) {

cout << "\nОшибка: вершина вне диапазона!\n";

return;

}

M.push\_back(vector<int>(n, 0));

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

M[i].push\_back(0);

}

int vertex2 = n;

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

if (M[vertex1][i] == 1) {

if (i == vertex1) {

M[vertex2][vertex2] = 1;

}

else {

M[vertex2][i] = 1;

M[i][vertex2] = 1;

}

}

}

M[vertex1][vertex2] = 1;

M[vertex2][vertex1] = 1;

//Список

G->vertexes.push\_back(nullptr);

G->size++;

node\* tmp = G->vertexes[vertex1];

while (tmp != nullptr) {

if (tmp->index != vertex1) {

add\_new\_el(G, vertex2, tmp->index);

add\_new\_el(G, tmp->index, vertex2);

}

tmp = tmp->next\_node;

}

add\_new\_el(G, vertex1, vertex2);

add\_new\_el(G, vertex2, vertex1);

node\* check\_loop = G->vertexes[vertex1];

while (check\_loop != nullptr) {

if (check\_loop->index == vertex1) {

add\_new\_el(G, vertex2, vertex2);

break;

}

check\_loop = check\_loop->next\_node;

}

remove\_copies(G);

cout << "\nВершина " << vertex1 + 1 << " расщеплена. Добавлена новая вершина " << vertex2 + 1 << ".\n";

}

// объединение

void unionGraphsPrint(const vector<vector<int>>& G1, const vector<vector<int>>& G2)

{

int n = G1.size();

int m = G2.size();

int size;

if (n > m) {

size = n;

}

else {

size = m;

}

vector<vector<int>> result(size, vector<int>(size, 0));

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

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

int val1 = 0;

int val2 = 0;

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

val1 = G1[i][j];

}

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

val2 = G2[i][j];

}

if (val1 == 1 || val2 == 1) {

result[i][j] = 1;

}

}

}

cout << "\nРезультат объединения:\n";

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

for (int j = 0; j < result[i].size(); j++) {

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

}

cout << "\n";

}

}

//пересечение

void intersectionGraphsPrint(const vector<vector<int>>& G1, const vector<vector<int>>& G2)

{

int n = G1.size();

int m = G2.size();

int size;

if (n > m) {

size = n;

}

else {

size = m;

}

vector<vector<int>> result(size, vector<int>(size, 0));

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

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

int val1 = 0;

int val2 = 0;

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

val1 = G1[i][j];

}

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

val2 = G2[i][j];

}

if (val1 == 1 && val2 == 1) {

result[i][j] = 1;

}

}

}

cout << "\nРезультат пересечения:\n";

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

for (int j = 0; j < result[i].size(); j++) {

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

}

cout << "\n";

}

}

//кольцевая сумма

void xorGraphsPrint(const vector<vector<int>>& G1, const vector<vector<int>>& G2)

{

int n = G1.size();

int m = G2.size();

int size;

if (n > m) {

size = n;

}

else {

size = m;

}

vector<vector<int>> result(size, vector<int>(size, 0));

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

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

int val1 = 0;

int val2 = 0;

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

val1 = G1[i][j];

}

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

val2 = G2[i][j];

}

if ((val1 == 1 && val2 == 0) || (val1 == 0 && val2 == 1)) {

result[i][j] = 1;

}

}

}

cout << "\nРезультат кольцевой суммы:\n";

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

for (int j = 0; j < result[i].size(); j++) {

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

}

cout << "\n";

}

}

void matrixPrint(vector<vector<int>>& M) {

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

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

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

}

cout << "\n";

}

}

// декартово произведение

void cartesianProductPrint(const vector<vector<int>>& G1, const vector<vector<int>>& G2) {

int n1 = G1.size();

int n2 = G2.size();

int N = n1 \* n2;

vector<vector<int>> result(N, vector<int>(N, 0));

for (int i = 0; i < n1; ++i) {

for (int j1 = 0; j1 < n2; ++j1) {

for (int j2 = 0; j2 < n2; ++j2) {

if (G2[j1][j2] == 1) {

int idx1 = i \* n2 + j1;

int idx2 = i \* n2 + j2;

result[idx1][idx2] = 1;

}

}

}

}

for (int i1 = 0; i1 < n1; ++i1) {

for (int i2 = 0; i2 < n1; ++i2) {

if (i1 != i2 && G1[i1][i2] == 1) {

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

int idx1 = i1 \* n2 + j;

int idx2 = i2 \* n2 + j;

result[idx1][idx2] = 1;

}

}

}

}

cout << "\nДекартово произведение (" << N << "x" << N << "):\n";

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

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

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

cout << "\n";

}

}

int main()

{

Graph G1, G2;

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

int m1, m2;

cout << "Введите размер матрицы M1: \n";

cin >> m1;

G1.size = m1;

G1.vertexes.resize(G1.size, nullptr);

vector<vector<int>>MatrixM1(m1, vector<int>(m1, 0));

for (int i = 0; i < m1; i++) {

for (int j = 0; j < m1; j++) {

if (j >= i) {

MatrixM1[i][j] = rand() % 2;

MatrixM1[j][i] = MatrixM1[i][j];

}

}

for (int j = 0; j < m1; j++) {

if (MatrixM1[i][j] == 1) {

add\_el(&G1, i, j);

}

}

}

cout << "\nМатрица смежности M1: \n";

matrixPrint(MatrixM1);

cout << "\nСписок смежности M1: \n";

print\_list(&G1);

cout << "\nВведите размер матрицы M2: \n";

cin >> m2;

G2.size = m2;

G2.vertexes.resize(G2.size, nullptr);

vector<vector<int>>MatrixM2(m2, vector<int>(m2, 0));

for (int i = 0; i < m2; i++) {

for (int j = 0; j < m2; j++) {

if (j >= i) {

MatrixM2[i][j] = rand() % 2;

MatrixM2[j][i] = MatrixM2[i][j];

}

}

for (int j = 0; j < m2; j++) {

if (MatrixM2[i][j] == 1) {

add\_el(&G2, i, j);

}

}

}

cout << "\nМатрица смежности M2: \n";

matrixPrint(MatrixM2);

cout << "\nСписок смежности M2: \n";

print\_list(&G1);

int task = -1;

while (task != 0) {

cout << "\nВыберите номер задания: \n1. Задание 2\n2. Задание 3 и 4\n0. Закрыть программу\n";

cin >> task;

switch (task) {

case 1: {

int point = -1;

while (point != 0) {

cout << "\nЗадание 2: \n1. Матрица M1 и список G1\n2. Матрица M2 и список G2\n0. Назад\n";

cin >> point;

cout << "\nМатрица смежности M1: \n";

matrixPrint(MatrixM1);

cout << "\nМатрица смежности M2: \n";

matrixPrint(MatrixM2);

switch (point) {

case 1: {

int number = -1;

while (number != 0) {

cout << "\nВыберите операцию c M1 и G1: \n1. Отождествление вершин\n2. Стягивание ребра\n3. Расщепление вершины\n0. Назад\n";

cin >> number;

cout << "\nМатрица смежности M1: \n";

matrixPrint(MatrixM1);

switch (number) {

case 1: {

int u, v;

cout << "\nВведите номера вершин для отождествления: ";

cin >> u >> v;

mergeVerticesMatrix(MatrixM1, &G1, u, v);

matrixPrint(MatrixM1);

print\_list(&G1);

break;

}

case 2: {

int u, v;

cout << "\nВведите номера вершин для стягивания ребра: ";

cin >> u >> v;

contractEdgeMatrix(MatrixM1, &G1, u, v);

matrixPrint(MatrixM1);

print\_list(&G1);

break;

}

case 3: {

int u;

cout << "\nВведите номер вершины для её расщепления: ";

cin >> u;

splitVertexMatrix(MatrixM1, &G1, u);

matrixPrint(MatrixM1);

print\_list(&G1);

break;

}

case 0: break;

default:

cout << "\nНекорректный ввод!\n";

}

}

break;

}

case 2: {

int number = -1;

while (number != 0) {

cout << "\nВыберите операцию c M2 и G2: \n1. Отождествление вершин\n2. Стягивание ребра\n3. Расщепление вершины\n0. Назад\n";

cin >> number;

cout << "\nМатрица смежности M1: \n";

matrixPrint(MatrixM1);

cout << "\nМатрица смежности M2: \n";

matrixPrint(MatrixM2);

switch (number) {

case 1: {

int u, v;

cout << "\nВведите номера вершин для отождествления: ";

cin >> u >> v;

mergeVerticesMatrix(MatrixM2, &G2, u, v);

matrixPrint(MatrixM2);

print\_list(&G2);

break;

}

case 2: {

int u, v;

cout << "\nВведите номера вершин для стягивания ребра: ";

cin >> u >> v;

contractEdgeMatrix(MatrixM2, &G2, u, v);

matrixPrint(MatrixM2);

print\_list(&G2);

break;

}

case 3: {

int u;

cout << "\nВведите номер вершины для её расщепления: ";

cin >> u;

splitVertexMatrix(MatrixM2, &G2, u);

matrixPrint(MatrixM2);

print\_list(&G2);

break;

}

case 0: break;

default:

cout << "\nНекорректный ввод!\n";

}

}

break;

}

case 0: break;

default:

cout << "\nНекорректный ввод!\n";

}

}

break;

}

case 2: {

int point = -1;

while (point != 0) {

cout << "\nВыберите операцию: \n1. Объединение\n2. Пересечение\n3. Кольцевая сумма\n4. Декартово произведение\n0. Назад\n";

cin >> point;

switch (point)

{

case 1:

unionGraphsPrint(MatrixM1, MatrixM2);

break;

case 2:

intersectionGraphsPrint(MatrixM1, MatrixM2);

break;

case 3:

xorGraphsPrint(MatrixM1, MatrixM2);

break;

case 4:

cartesianProductPrint(MatrixM1, MatrixM2);

break;

case 0: break;

default:

cout << "\nНекорректный ввод!\n";

}

}

break;

}

case 0:

cout << "\nПрограмма завершена.\n";

break;

default:

cout << "\nНекорректный ввод!\n";

}

}

return 0;

}