Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**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 22ВВП1

Воробьева М.М.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Цель работы:** унарные и бинарные операции над графами.

**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 https://lh7-us.googleusercontent.com/q7PwcmMUbz6bmvKuOOfgZgPyXn8L4qdvG8piaHbHMhUM-lnZXQnziwksljvEuB0BMoE52M3Oe0-TRRmr_E-VhfN5pS23_ejESh6AOs6DG0CkxRsB75I_fdIU2ZWBIR9sNHXxnF5nNEs6KxrpsUVzlQ *G*2

б) пересечения  *G* = *G*1 https://lh7-us.googleusercontent.com/8XlNG2AxbrUSQCC9CfCQBFimGbilvD-rKpA0UcRC9TmiwtSEwotbNj1TnqLlJoZCrXSlCrEz8-kSr54cjfzAJVldiKuWrF4BPZGQWNZ7wt6kWcOViDJVXRd8_yF9jRgABk1r-LmnhAk1kaGhAVVogA *G*2

в) кольцевой суммы *G* = *G*1 https://lh7-us.googleusercontent.com/AANvGX39-xh92a0PFhgTRGdskbd6mU6_F_jDP3glx-Xj99sGbnYPDJc6-1IT5VOZEUI7U0ZQiRFYU0qKtEEzZ54MvXIaaMJ9GnfIljmoM7Jus6WREDud9gPQTd2ejPwyLdYPd8aED8Qk8ggmitzhPA *G*2

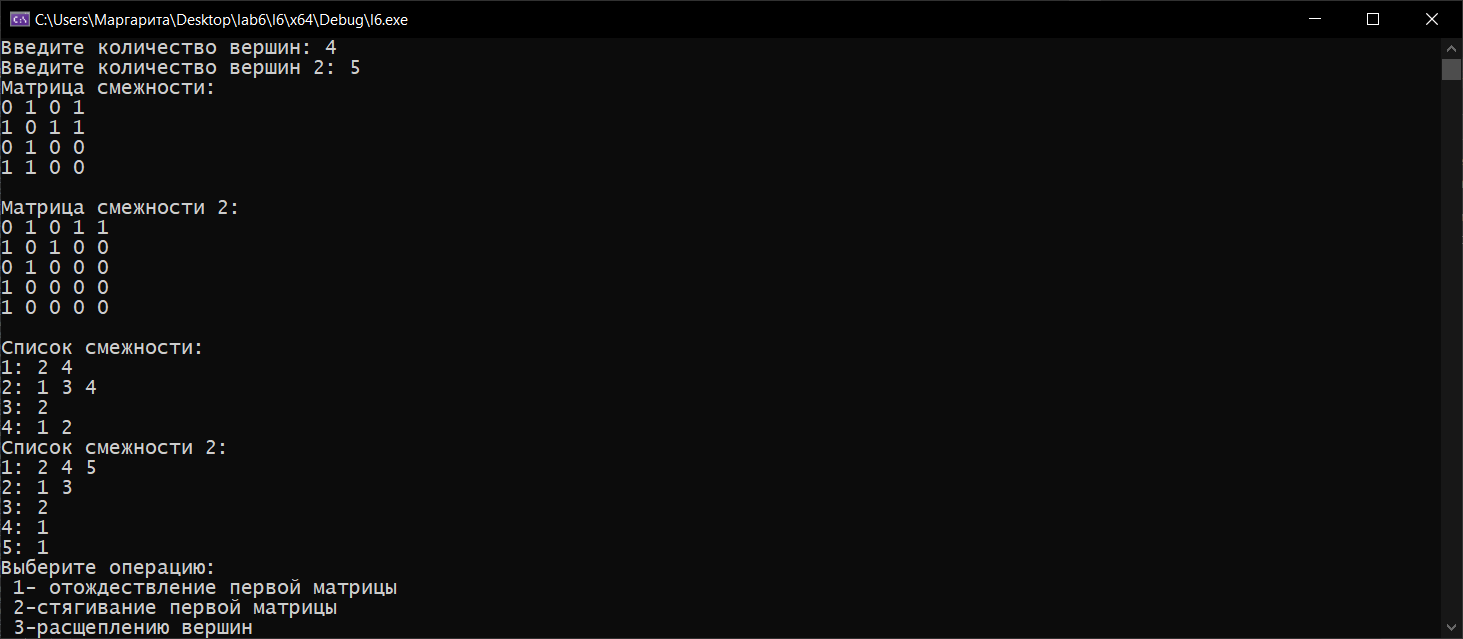
Результат выполнения операции выведите на экран.

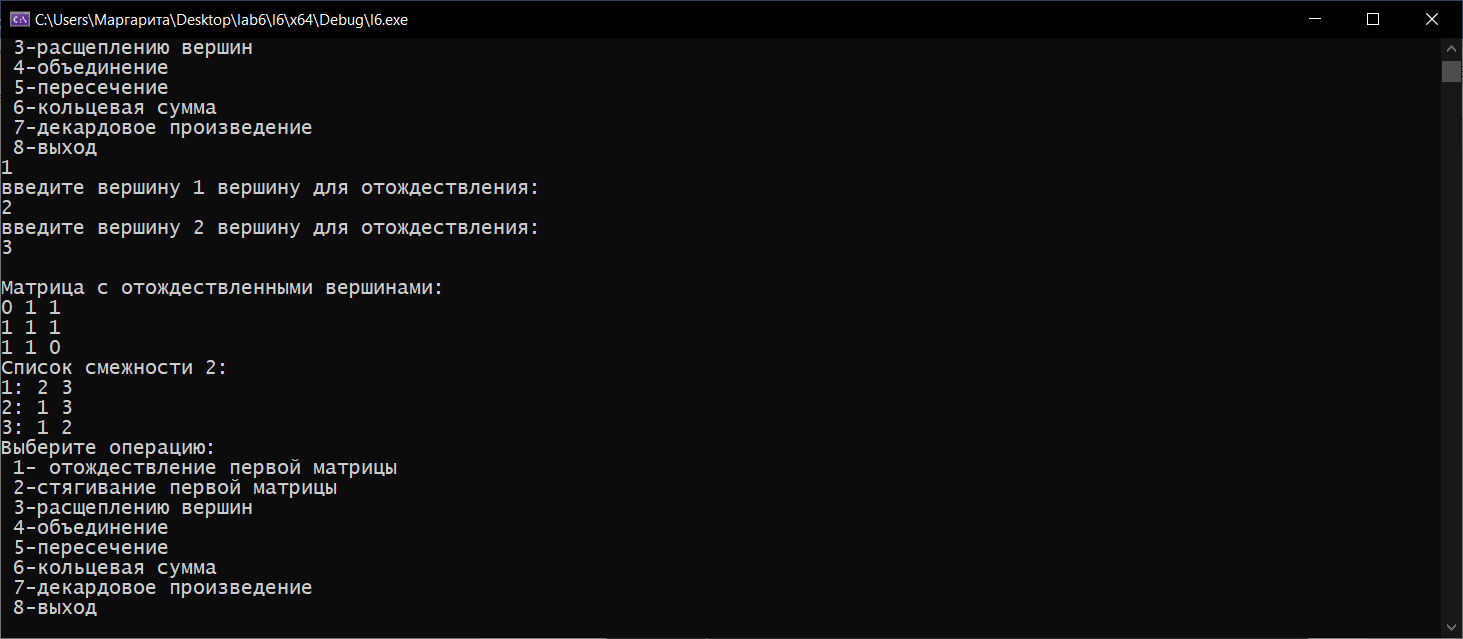
**Задание 4 \***

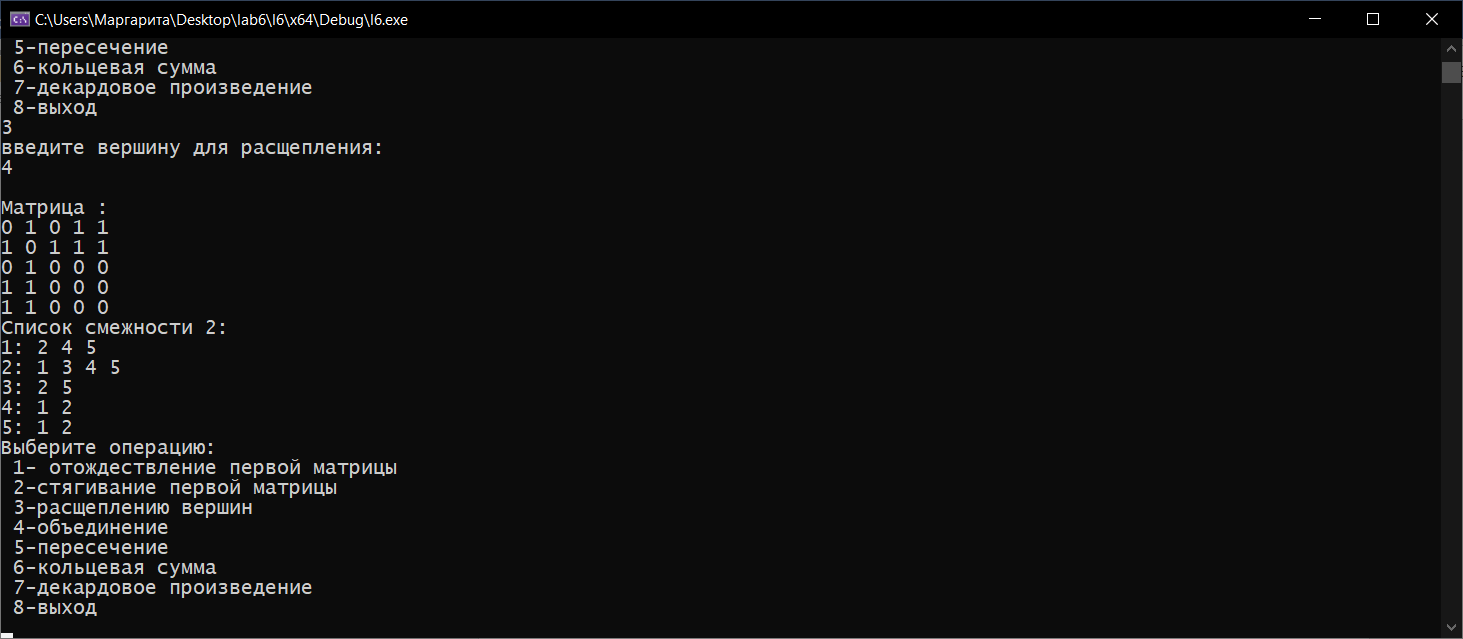
1.Для матричной формы представления графов выполните операцию декартова произведения графов *G = G*1X *G*2.

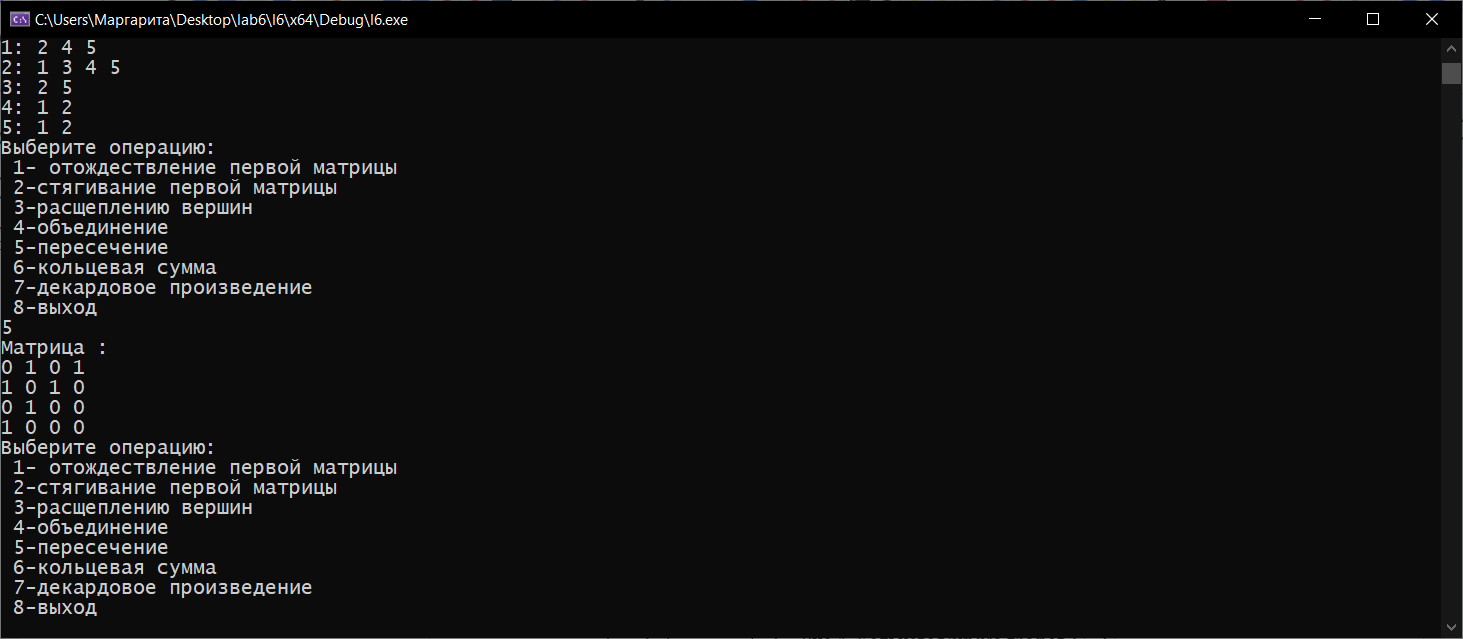
Результат выполнения операции выведите на экран.

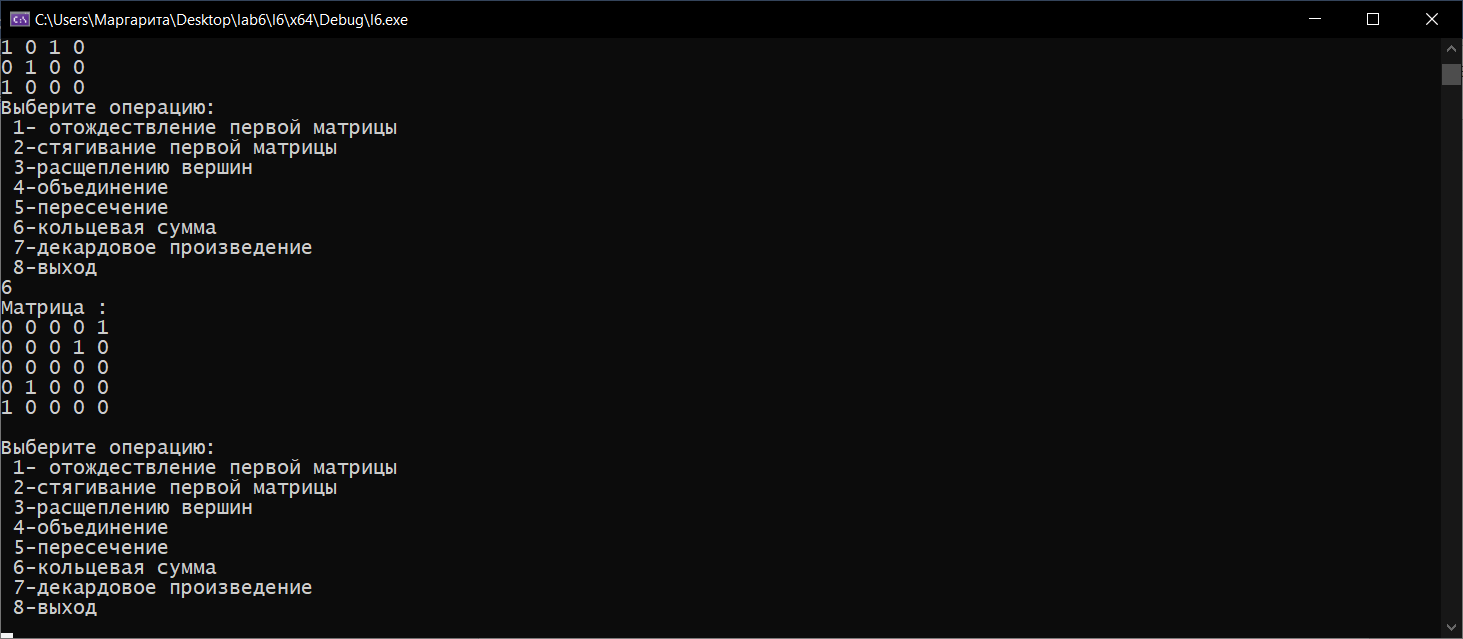
**Результат работы программы:**

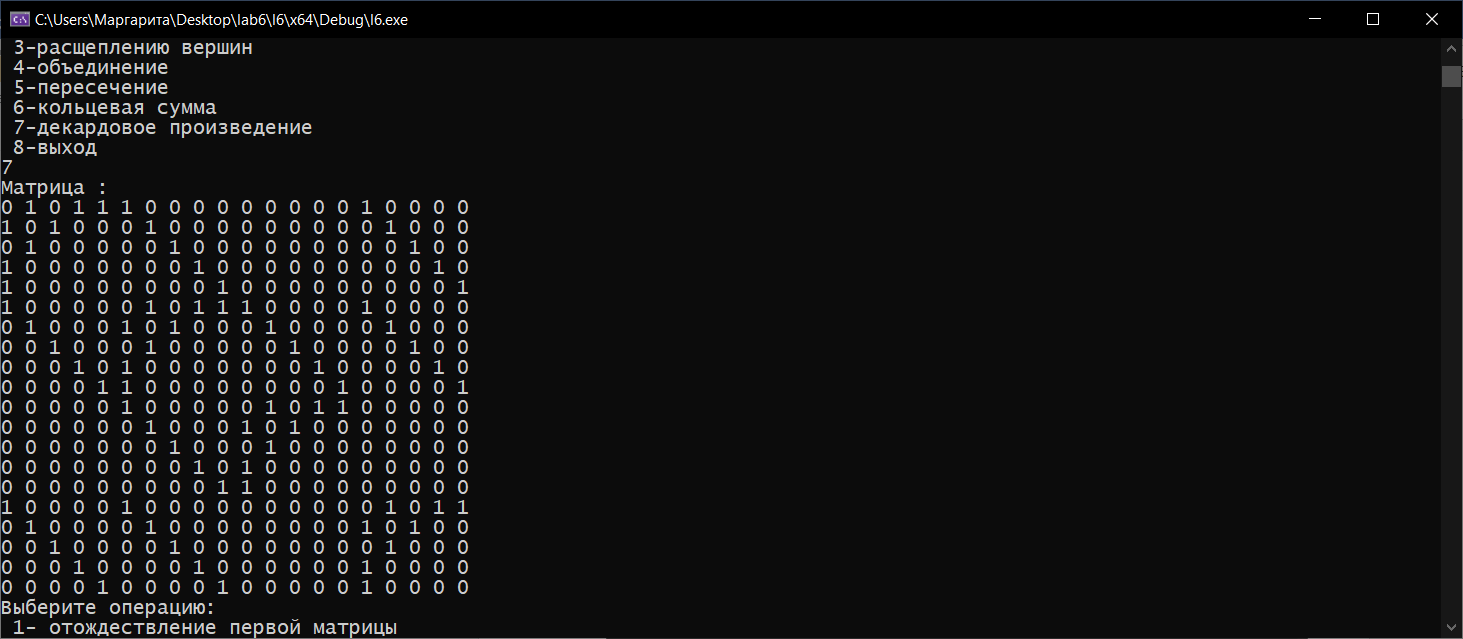












**Вывод:** в ходе выполнения лабораторной работы были освоены бинарные и унарные операции над графами

Листинг программы:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <vector>

#include <ctime>

#include <cstdlib>

#include <locale.h>

#include <algorithm>

using namespace std;

int swt = 0;

//Отождествление (замыкание) вершин

void otojestvlenie(vector < vector < int>> G, vector < vector < int>> List) {

int v1, v2;

vector < vector < int>> r\_G;

vector < vector < int>> r\_L;

printf("введите вершину 1 вершину для отождествления: \n");

scanf\_s("%d", &v1);

printf("введите вершину 2 вершину для отождествления: \n");

scanf\_s("%d", &v2);

printf("\n");

r\_G = G;

r\_L = List;

int n = r\_G.size();

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

if (r\_G[v2][i]) {

r\_G[v1][i] = 1;

r\_G[i][v1] = 1;

}

r\_G[v2][i] = 0;

r\_G[i][v2] = 0;

}

for (int neighbor : r\_L[v2]) {

if (find(r\_L[v1].begin(), r\_L[v1].end(), neighbor) == r\_L[v1].end() && neighbor != v1) {

r\_L[v1].push\_back(neighbor);

}

}

r\_G.erase(r\_G.begin() + v2);

for (auto &row : r\_G) {

row.erase(row.begin() + v2);

}

for (auto& neighbors : r\_L) {

auto it = remove(neighbors.begin(), neighbors.end(), v2);

neighbors.erase(it, neighbors.end());

for (auto& neighbor : neighbors) {

if (neighbor > v2) {

--neighbor;

}

}

}

r\_L.erase(r\_L.begin() + v2);

printf("Матрица с отождествленными вершинами:\n");

for (int i = 1; i < r\_G.size(); ++i) {

for (int j = 1; j < r\_G[i].size(); ++j) {

printf("%d ", r\_G[i][j]);

}

printf("\n");

}

printf("Список смежности 2:\n");

for (int i = 1; i < r\_L.size(); ++i) {

printf("%d: ", i);

for (int j : r\_L[i]) {

printf("%d ", j);

}

printf("\n");

}

}

//стягивание

void stiagivanie(vector<vector<int>>G,vector<vector<int>>List){

int v1, v2;

vector < vector < int>> r\_G;

vector < vector < int>> r\_L;

printf("введите вершину 1 вершину для стягивания: \n");

scanf\_s("%d", &v1);

printf("введите вершину 2 вершину для стягивания: \n");

scanf\_s("%d", &v2);

printf("\n");

r\_G = G;

r\_L = List;

int n = r\_G.size();

if (!r\_G[v1][v2] && !r\_G[v2][v1]) {

printf("Стягивание осуществить нельзя!\n");

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

for (int j = 1; j < G[i].size(); ++j) {

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

}

printf("\n");

}

printf("Список смежности:\n");

for (int i = 1; i < r\_L.size(); ++i) {

printf("%d: ", i);

for (int j : r\_L[i]) {

printf("%d ", j);

}

printf("\n");

}

return;

}

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

if (r\_G[v2][i]) {

r\_G[v1][i] = 1;

r\_G[i][v1] = 1;

}

r\_G[v2][i] = 0;

r\_G[i][v2] = 0;

}

r\_G[v1][v1] = 0;

// Объединение списков смежности

for (int neighbor : r\_L[v2]) {

if (find(r\_L[v1].begin(), r\_L[v1].end(), neighbor) == r\_L[v1].end() && neighbor != v1) {

r\_L[v1].push\_back(neighbor);

}

}

for (auto& neighbors : r\_L) {

auto it = remove(neighbors.begin(), neighbors.end(), v2);

neighbors.erase(it, neighbors.end());

for (auto& neighbor : neighbors) {

if (neighbor > v2) {

--neighbor;

}

}

}

r\_G.erase(r\_G.begin() + v2);

for (auto& row : r\_G) {

row.erase(row.begin() + v2);

}

r\_L.erase(r\_L.begin() + v2);

printf("Матрица со стянутым ребром:\n");

for (int i = 1; i < r\_G.size(); ++i) {

for (int j = 1; j < r\_G[i].size(); ++j) {

printf("%d ", r\_G[i][j]);

}

printf("\n");

}

printf("Список смежности 2:\n");

for (int i = 1; i < r\_L.size(); ++i) {

printf("%d: ", i);

for (int j : r\_L[i]) {

printf("%d ", j);

}

printf("\n");

}

}

vector<vector<int>> list\_cmej(vector<vector<int>> G) {

int num\_vertices = G.size() - 1;

vector<vector<int>> adjacency\_list(num\_vertices + 1);

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

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

adjacency\_list[i].push\_back(j);

}

}

}

return adjacency\_list;

}

void printList(vector<vector<int>>list) {

printf("Список смежности:\n");

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

printf("%d: ", i);

for (int j : list[i]) {

printf("%d ", j);

}

printf("\n");

}

}

vector<vector<int>> list\_cmej2(vector<vector<int>> G2) {

int num\_vertices = G2.size() - 1;

vector<vector<int>> adjacency\_list2(num\_vertices + 1);

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

if (G2[i][j] == 1) {

adjacency\_list2[i].push\_back(j);

}

}

}

return adjacency\_list2;

}

void printList2(vector<vector<int>>list2) {

printf("Список смежности 2:\n");

for (int i = 1; i < list2.size(); ++i) {

printf("%d: ", i);

for (int j : list2[i]) {

printf("%d ", j);

}

printf("\n");

}

}

//int swt = 0;

vector<vector<int>> cmej\_mat(int num\_vertices, double veroj) {

vector<vector<int>> cmej\_mat(num\_vertices + 1, vector<int>(num\_vertices + 1, 0));

int cont = 0;

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = i + 1; j <= num\_vertices; ++j) {

if (static\_cast<double>(rand()) / RAND\_MAX < veroj) {

cmej\_mat[i][j] = 1;

cmej\_mat[j][i] = 1;

cont++;

}

}

}

if (cont == 0) {

swt = 1;

}

printf("Матрица смежности:\n");

for (int i = 1; i <= num\_vertices; ++i) {

for (int j = 1; j <= num\_vertices; ++j) {

printf("%d ", cmej\_mat[i][j]);

}

printf("\n");

}

printf("\n");

return cmej\_mat;

}

//int swt = 0;

vector<vector<int>> cmej\_mat2(int num\_vertices2, double veroj) {

vector<vector<int>> cmej\_matr2(num\_vertices2 + 1, vector<int>(num\_vertices2 + 1, 0));

int cont = 0;

for (int i = 1; i <= num\_vertices2; ++i) {

for (int j = i + 1; j <= num\_vertices2; ++j) {

if (static\_cast<double>(rand()) / RAND\_MAX < veroj) {

cmej\_matr2[i][j] = 1;

cmej\_matr2[j][i] = 1;

cont++;

}

}

}

if (cont == 0) {

swt = 1;

}

printf("Матрица смежности 2:\n");

for (int i = 1; i <= num\_vertices2; ++i) {

for (int j = 1; j <= num\_vertices2; ++j) {

printf("%d ", cmej\_matr2[i][j]);

}

printf("\n");

}

printf("\n");

return cmej\_matr2;

}

void splitVertex(vector < vector < int>> G, vector < vector < int>>List) {

int v1;

vector < vector < int>> r\_G;

vector < vector < int>> r\_L;

printf("введите вершину для расщепления: \n");

scanf\_s("%d", &v1);

printf("\n");

int n = G.size();

r\_G.resize(n + 1, vector<int>(n + 1, 0));

r\_L.resize(n + 1);

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

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

r\_G[i][j] = G[i][j];

}

r\_L[i] = List[i];

}

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

r\_G[i][n] = G[i][v1];

r\_G[n][i] = G[v1][i];

}

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

if (i != v1) {

r\_L[i].push\_back(n);

}

}

for (int i = 0; i < List[v1].size(); ++i) {

r\_L[n].push\_back(List[v1][i]);

}

printf("Матрица :\n");

for (int i = 1; i < r\_G.size(); ++i) {

for (int j = 1; j < r\_G[i].size(); ++j) {

printf("%d ", r\_G[i][j]);

}

printf("\n");

}

printf("Список смежности 2:\n");

for (int i = 1; i < r\_L.size(); ++i) {

printf("%d: ", i);

for (int j : r\_L[i]) {

printf("%d ", j);

}

printf("\n");

}

}

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

int razm = max(G1.size(), G2.size());

vector < vector < int>> r\_G(razm, vector<int>(razm, 0));

for (int i = 0; i < G1.size(); ++i) {

for (int j = 0; j < G1[i].size(); ++j) {

r\_G[i][j] =G1[i][j];

}

}

for (int i = 0; i < G2.size(); ++i) {

for (int j = 0; j < G2[i].size(); ++j) {

if (G2[i][j] > 0) {

r\_G[i][j] = G2[i][j];

}

}

}

printf("Матрица :\n");

for (int i = 1; i < r\_G.size(); ++i) {

for (int j = 1; j < r\_G[i].size(); ++j) {

printf("%d ", r\_G[i][j]);

}

printf("\n");

}

}

void intersectGraphs( vector < vector < int>> G1,vector < vector < int>>G2) {

int size = min(G1.size(), G2.size());

vector < vector < int>> G3(size, vector<int>(size, 0));

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

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

G3[i][j] = (G1[i][j] and G2[i][j]);

}

}

printf("Матрица :\n");

for (int i = 1; i < G3.size(); ++i) {

for (int j = 1; j < G3[i].size(); ++j) {

printf("%d ", G3[i][j]);

}

printf("\n");

}

}

void adaptiveRingSumGraphs(vector < vector < int>> G1, vector < vector < int>>G2) {

int razm = max(G1.size(), G2.size());

vector < vector < int>>G3(razm, vector<int>(razm, 0));

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

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

int val1 = (i < G1.size() && j < G1.size()) ? G1[i][j] : 0;

int val2 = (i < G2.size() && j < G2.size()) ? G2[i][j] : 0;

G3[i][j] = val1 != val2;

}

}

printf("Матрица :\n");

for (int i = 1; i < G3.size(); ++i) {

for (int j = 1; j < G3[i].size(); ++j) {

printf("%d ", G3[i][j]);

}

printf("\n");

}

printf("\n");

}

void cartesianProductGraphs(vector < vector < int>> G1, vector < vector < int>>G2) {

int n1 = G1.size() - 1;

int n2 = G2.size() - 1;

int razm = n1 \* n2;

vector < vector < int>> G3(razm + 1, vector<int>(razm+ 1, 0));

for (int i1 = 1; i1 <= n1; ++i1) {

for (int i2 = 1; i2 <= n2; ++i2) {

for (int j1 = 1; j1 <= n1; ++j1) {

for (int j2 = 1; j2 <= n2; ++j2) {

int v1 = (i1 - 1) \* n2 + i2;

int v2 = (j1 - 1) \* n2 + j2;

if (i1 == j1 && G2[i2][j2]) {

G3[v1][v2] = 1;

}

if (i2 == j2 && G1[i1][j1]) {

G3[v1][v2] = 1;

}

}

}

}

}

printf("Матрица :\n");

for (int i = 1; i < G3.size(); ++i) {

for (int j = 1; j < G3[i].size(); ++j) {

printf("%d ", G3[i][j]);

}

printf("\n");

}

}

int main() {

setlocale(LC\_ALL, "RUS");

int num\_vertices;

int num\_vertices2;

double veroj = 0.5;

int s;

int p;

srand(static\_cast<unsigned int>(time(nullptr)));

printf("Введите количество вершин: ");

scanf("%d", &num\_vertices);

printf("Введите количество вершин 2: ");

scanf("%d", &num\_vertices2);

auto G = cmej\_mat(num\_vertices, veroj);

auto G2 = cmej\_mat2(num\_vertices2, veroj);

auto list = list\_cmej(G);

auto list2 = list\_cmej2(G2);

printList(list);

printList2(list2);

vector < vector < int>> matrix1, matrix2, matrixCopy;

vector < vector < int>> lis1, lis2, lisCopy;

while (true) {

printf("Выберите операцию:\n 1- отождествление первой матрицы\n 2-стягивание первой матрицы\n 3-расщеплению вершин\n 4-объединение\n 5-пересечение\n 6-кольцевая сумма\n 7-декардовое произведение\n 8-выход\n");

if(scanf("%d", &p) != 1) {

printf("Ошибка: ожидался ввод числа.\n");

while (getchar() != '\n'); // Очистка ввода

continue;

}

switch (p) {

case 1: {

otojestvlenie(G, list);

break;

}

case 2: {

stiagivanie(G, list);

break;

}

case 3: {

splitVertex(G, list);

break;

}

case 4: {

mergeGraphs(G, G2);

break;

}

case 5: {

intersectGraphs(G, G2);

break;

}

case 6: {

adaptiveRingSumGraphs(G, G2);

break;

}

case 7: {

cartesianProductGraphs(G, G2);

break;

}

// if (p == 8) {

// break;

//}

return 0;

}

}

}