Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему: «Унарные и бинарные операции над графами.»

Выполнил**:**

студент группы 21ВВ4

Савкин В.В

Проверили:

Юрова О.В,

Акифьев И.В.

Пенза 2022

**Цель работы****:** Изучение унарных и бинарных операций над графами.

**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.

### **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.

**Задание 3**

1. Для матричной формы представления графов выполните операцию:

а) объединения *G* = *G*1  *G*2

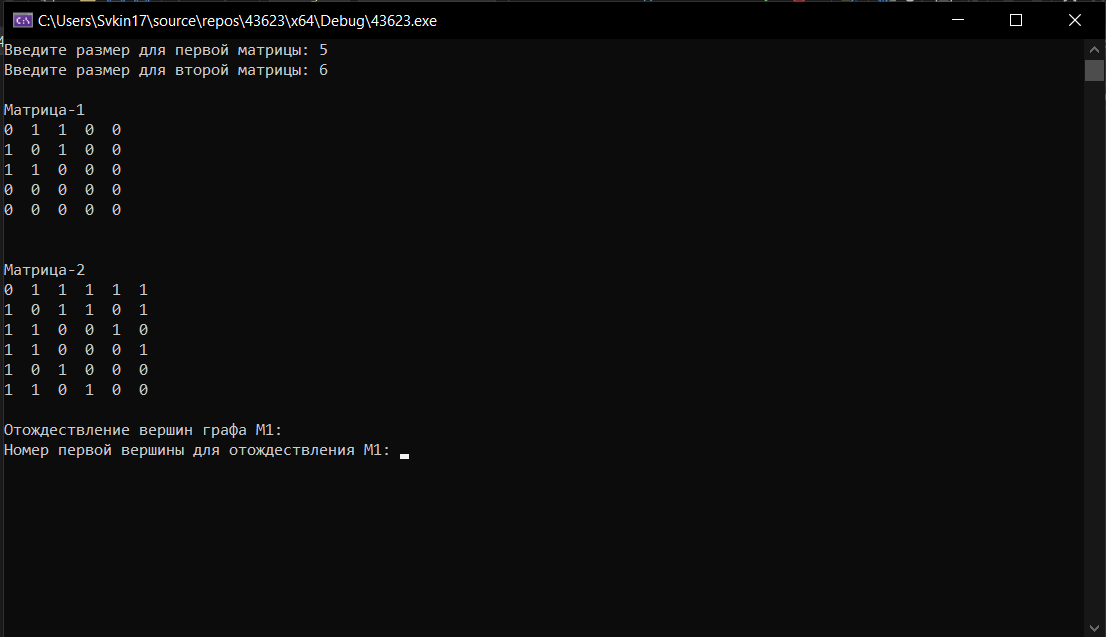
б) пересечения *G* = *G*1  *G*2

в) кольцевой суммы *G* = *G*1  *G*2

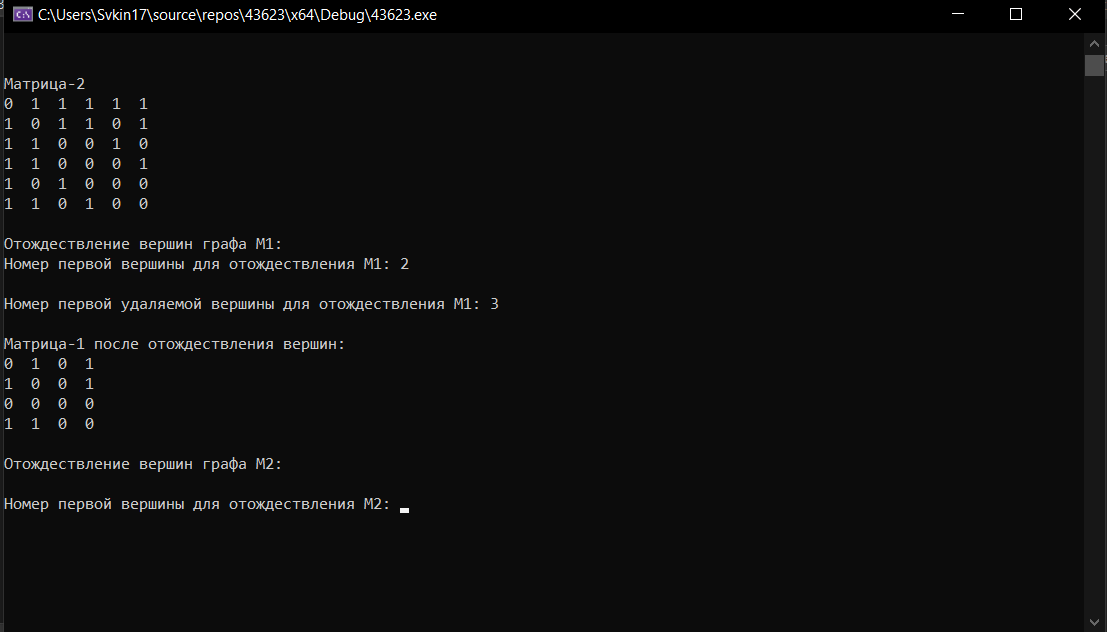
Результат выполнения операции выведите на экран.

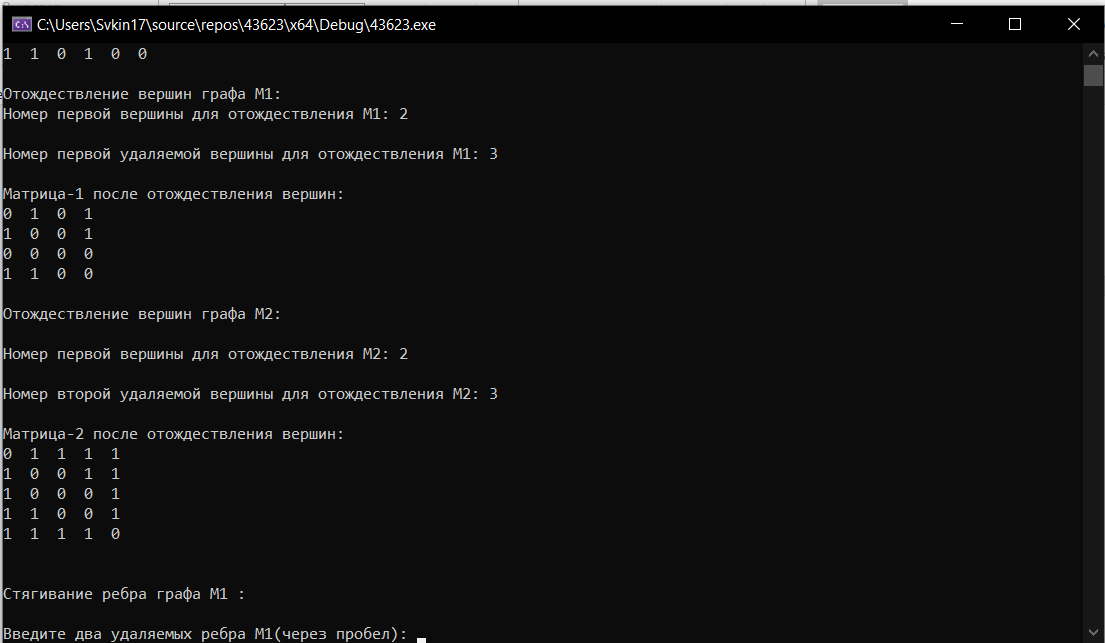
**Ход работы:**

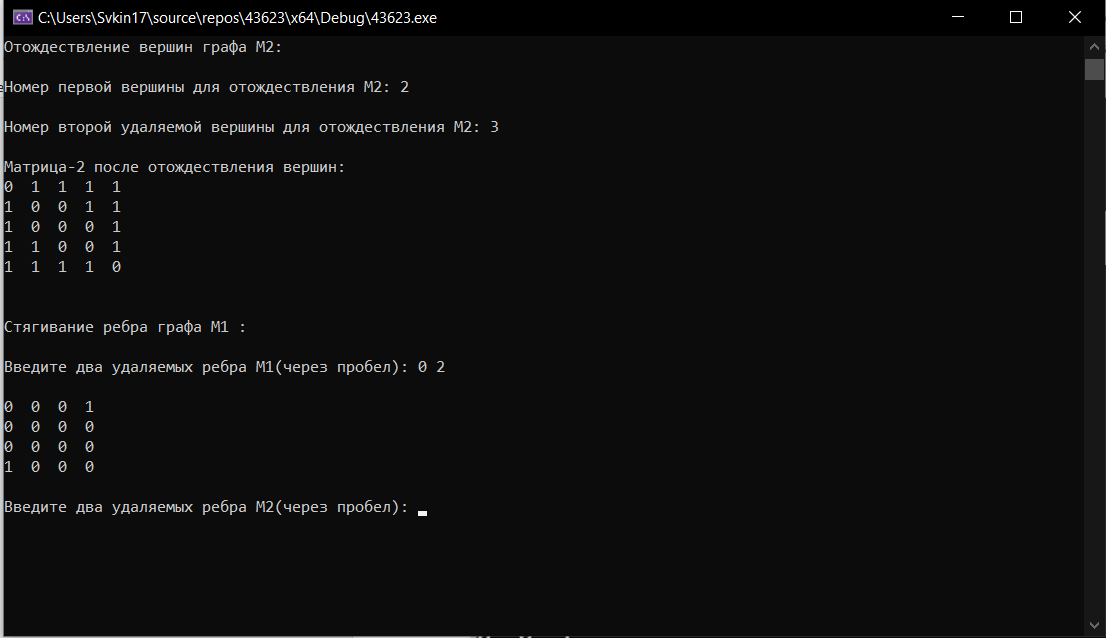
**Задание-1**

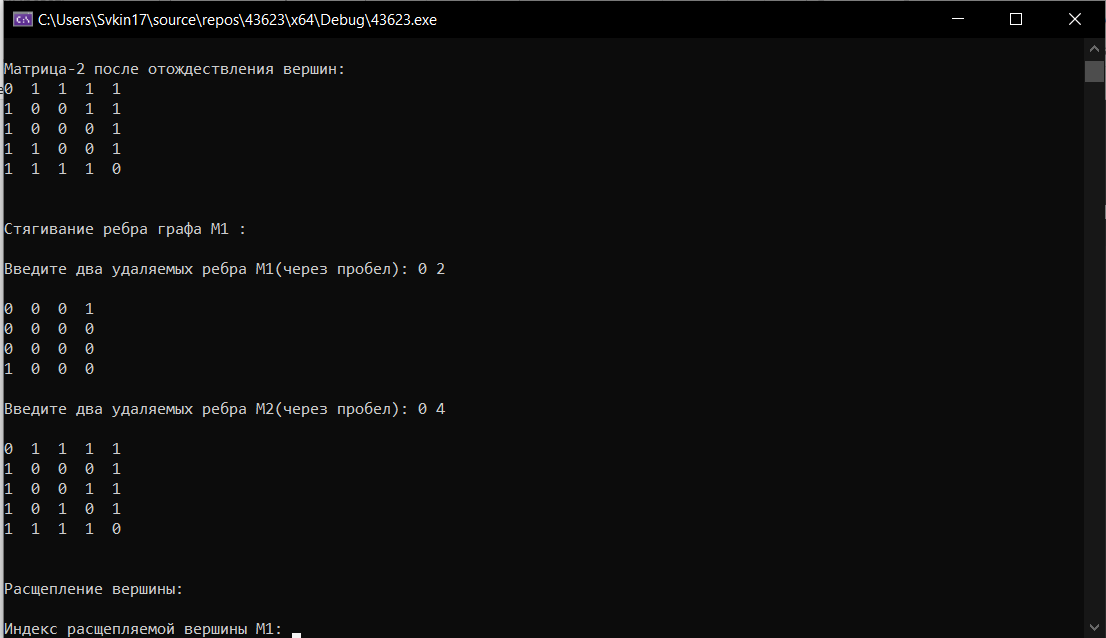
****

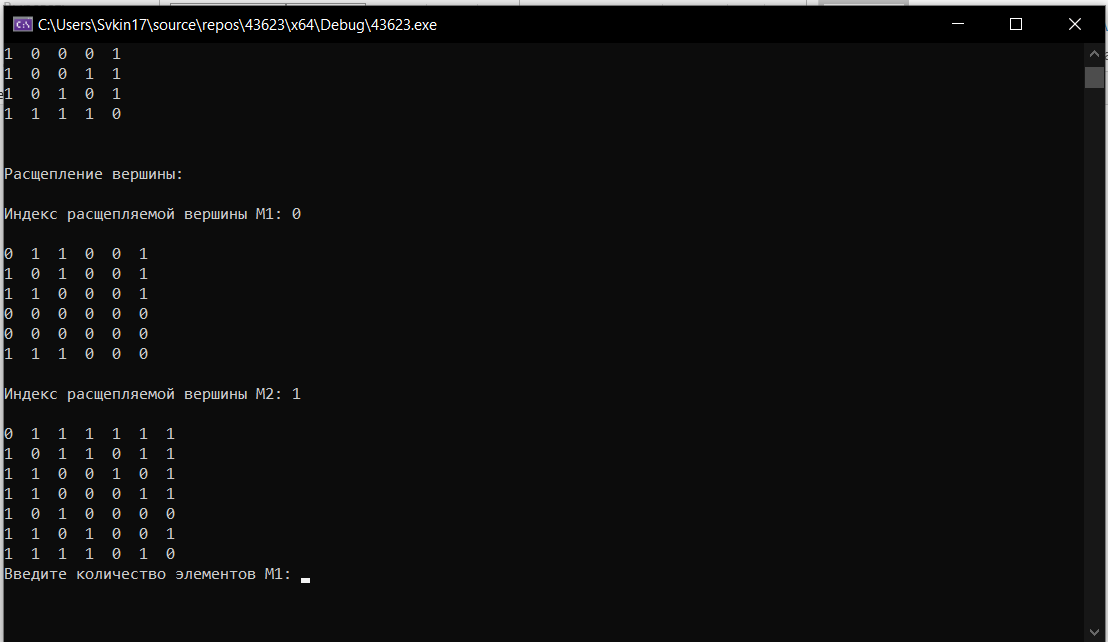
**Задание 2**

****

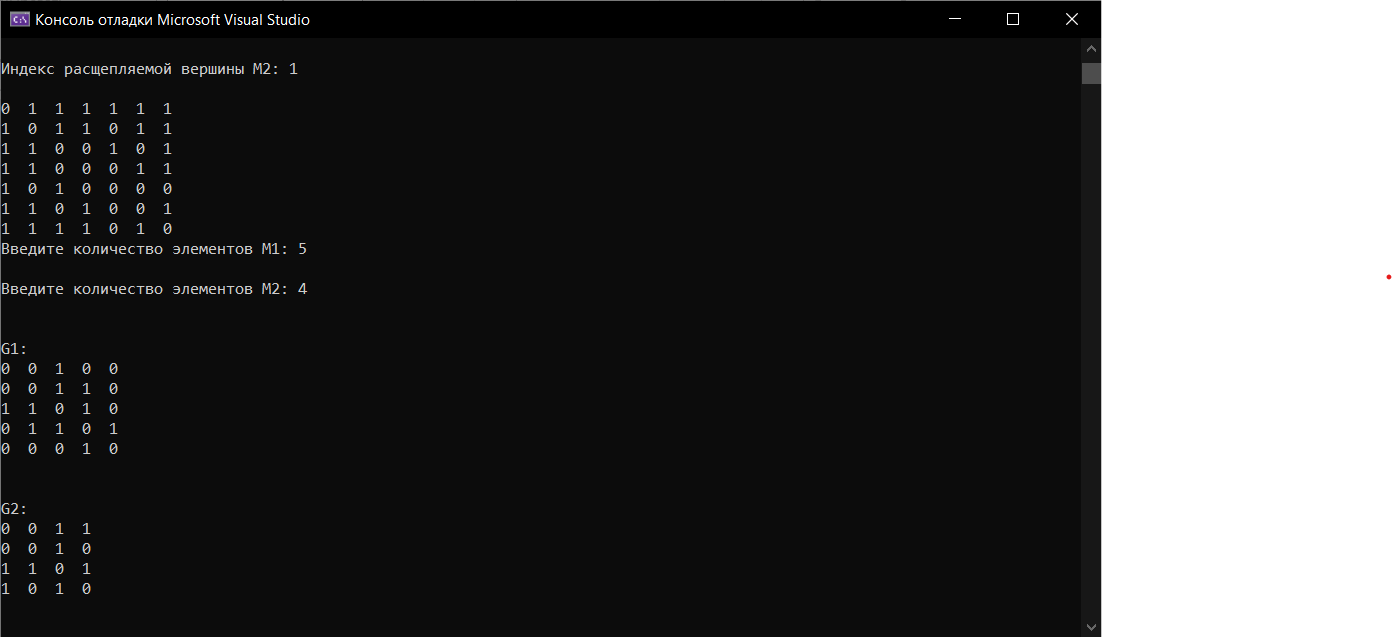
****

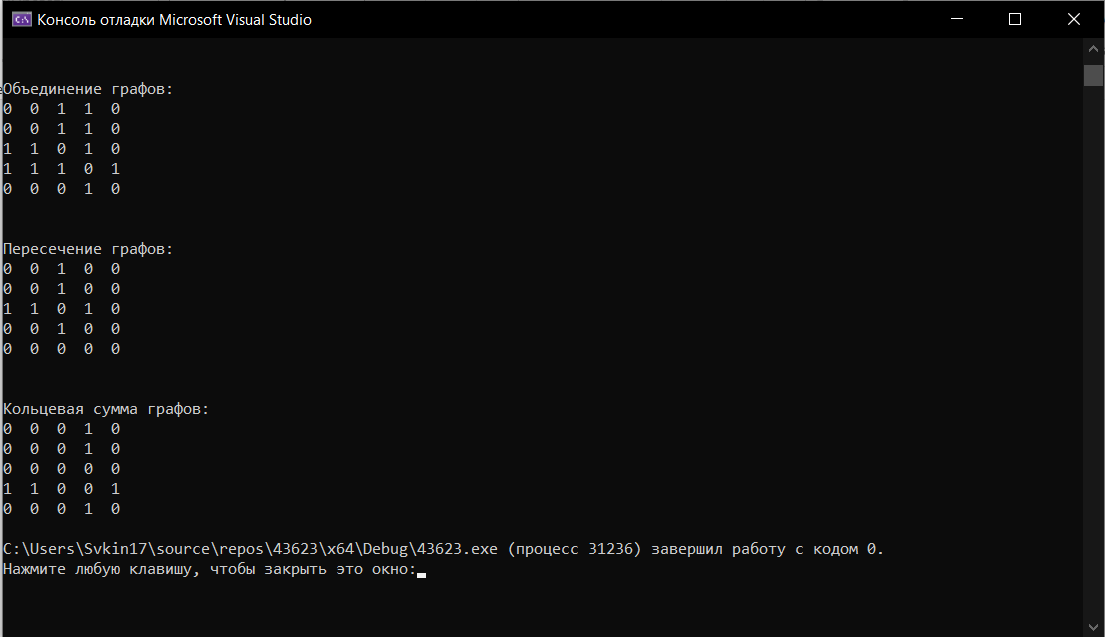
****

****

****

**Задание 3**

****

****

**Листинг:**

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

bool CheckAdj(int\*\* matrix, int del1, int del2) {

if (matrix[del1][del2] == 1) {

return true;

}

return false;

}

void create(int\*\*\* matrix, int size) {

\*matrix = (int\*\*)malloc(size \* sizeof(int\*));

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

{

(\*matrix)[i] = (int\*)calloc(size, sizeof(int));

}

}

void fill(int\*\* matrix, int size) {

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

{

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

{

if (i == j) {

matrix[i][j] = 0;

}

else

{

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

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

}

}

}

}

void print(int\*\* matrix, int size) {

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

{

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

{

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

}

printf("\n");

}

}

int\*\* identification(int\*\* matrix, int size, int del1, int del2, bool compres) {

int\*\* newMatrix = NULL;

int\* save1 = NULL;

int\* save2 = NULL;

int newsize = size - 1;

create(&newMatrix, newsize);

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

{

if (i == del1)

{

save1 = matrix[i];

}

if (i == del2) {

save2 = matrix[i];

}

}

int index1 = 0;

int index2 = 0;

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

if (i == del1 || i == del2) {

continue;

}

index2 = 0;

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

{

if (j == del2 || j == del1) {

continue;

}

newMatrix[index2][index1] = matrix[i][j];

newMatrix[index1][index2] = matrix[i][j];

index2++;

}

index1++;

}

index1 = 0;

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

{

if (i == del1 || i == del2)

{

continue;

}

newMatrix[newsize - 1][index1] = save1[i];

newMatrix[index1][newsize - 1] = save1[i];

index1++;

}

index1 = 0;

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

{

if (i == del1 || i == del2) {

continue;

}

if (newMatrix[index1][newsize - 1] == 1 || save2[i] == 0) {

index1++;

continue;

}

newMatrix[newsize - 1][index1] = save2[i];

newMatrix[index1][newsize - 1] = save2[i];

index1++;

}

if (!compres && CheckAdj(matrix, del1, del2)) {

newMatrix[newsize - 1][newsize - 1] = 1;

}

return newMatrix;

}

int\*\* split(int\*\* matrix, int size, int split) {

int\*\* splitMatrix = NULL;

create(&splitMatrix, size+1);

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

{

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

{

splitMatrix[i][j] = matrix[i][j];

}

}

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

{

splitMatrix[i][size] = matrix[i][split];

splitMatrix[size][i] = matrix[split][i];

splitMatrix[size][size] = 0;

splitMatrix[split][size] = 1;

splitMatrix[size][split] = 1;

}

return splitMatrix;

}

int\*\* disconnect(int\*\* matrix1, int\*\* matrix2, int maxSize) {

int\*\* resultMatrix = NULL;

create(&resultMatrix, maxSize);

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

{

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

if (matrix1[i][j] == 1 || matrix2[i][j] == 1)

{

resultMatrix[i][j] = 1;

}

}

}

return resultMatrix;

}

int\*\* connect(int\*\* matrix1, int\*\* matrix2, int maxSize) {

int\*\* resultMatrix = NULL;

create(&resultMatrix, maxSize);

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

{

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

if (matrix1[i][j] == 1 && matrix2[i][j] == 1)

{

resultMatrix[i][j] = 1;

}

}

}

return resultMatrix;

}

int\*\* ringsum(int\*\* firstMatrix, int\*\* secondMatrix, int maxSize) {

int\*\* resultMatrix = NULL;

create(&resultMatrix, maxSize);

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

{

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

if ((firstMatrix[i][j] == 1) ^ (secondMatrix[i][j] == 1))

{

resultMatrix[i][j] = 1;

}

else

{

resultMatrix[i][j] = 0;

}

}

}

return resultMatrix;

}

int main()

{

//Часть 1//

setlocale(LC\_ALL, "Rus");

int\*\* matrix1 = NULL;

int\*\* matrix2 = NULL;

int\*\* matrixRes = NULL;

int size1 = 0;

int size2 = 0;

int del1 = 0;

int del2 = 0;

int userInputFirstMatrix = 0;

int userInputSecondMatrix = 0;

printf("Введите размер для первой матрицы: ");

scanf\_s("%d", &size1);

printf("Введите размер для второй матрицы: ");

scanf\_s("%d", &size2);

printf("\nМатрица-1\n");

create(&matrix1, size1);

fill(matrix1, size1);

print(matrix1, size1);

printf("\n\nМатрица-2\n");

create(&matrix2, size2);

fill(matrix2, size2);

print(matrix2, size2);

printf("\nОтождествление вершин графа M1:\n");

printf("Номер первой вершины для отождествления М1: ");

scanf\_s("%d", &del1);

printf("\nНомер первой удаляемой вершины для отождествления М1: ");

scanf\_s("%d", &del2);

int\*\* identificationFirstMatrix = identification(matrix1, size1, del1, del2, false);

printf("\nМатрица-1 после отождествления вершин: \n");

print(identificationFirstMatrix, size1 - 1);

printf("\nОтождествление вершин графа M2:\n");

printf("\nНомер первой вершины для отождествления М2: ");

scanf\_s("%d", &del1);

printf("\nНомер второй удаляемой вершины для отождествления М2: ");

scanf\_s("%d", &del2);

int\*\* identificationSecondMatrix = identification(matrix2, size2, del1, del2, false);

printf("\nМатрица-2 после отождествления вершин: \n");

print(identificationSecondMatrix, size2 - 1);

bool InputCheck = false;

while (!InputCheck)

{

printf("\n\nСтягивание ребра графа М1 :\n");

printf("\nВведите два удаляемых ребра М1(через пробел): ");

scanf\_s("%d %d", &del1, &del2);

printf("\n");

if (CheckAdj(matrix1, del1, del2)) {

int\*\* EdgeContractionFirstMatrix = identification(matrix1, size1, del1, del2, true);

print(EdgeContractionFirstMatrix, size1 - 1);

InputCheck = true;

free(EdgeContractionFirstMatrix);

}

else {

printf("\nВершины не смежны\n");

InputCheck = false;

}

}

InputCheck = false;

while (!InputCheck) {

printf("\nВведите два удаляемых ребра М2(через пробел): ");

scanf\_s("%d %d", &del1, &del2);

printf("\n");

if (CheckAdj(matrix2, del1, del2)) {

int\*\* EdgeContractionSecondMatrix = identification(matrix2, size2, del1, del2, true);

print(EdgeContractionSecondMatrix, size2 - 1);

InputCheck = true;

free(EdgeContractionSecondMatrix);

}

else {

printf("\nВершины не смежны\n");

InputCheck = false;

}

}

//Часть 2//

printf("\n\nРасщепление вершины:\n");

printf("\nИндекс расщепляемой вершины М1: ");

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

printf("\n");

int\*\* splitFirstMatrix = split(matrix1, size1, userInputFirstMatrix);

print(splitFirstMatrix, size1 + 1);

printf("\nИндекс расщепляемой вершины М2: ");

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

printf("\n");

int\*\* splitSecondMatrix = split(matrix2, size2, userInputSecondMatrix);

print(splitSecondMatrix, size2 + 1);

int\*\* firstMatrix = NULL;

int\*\* secondMatrix = NULL;

int\*\* resultMatrix = NULL;

size1 = 0;

size2 = 0;

int countElements = 0;

printf("Введите количество элементов М1: ");

scanf\_s("%d", &size1);

printf("\nВведите количество элементов М2: ");

scanf\_s("%d", &size2);

printf("\n");

int maxMatrixSize = 0;

if (size1 > size2) {

maxMatrixSize = size1;

}

else {

maxMatrixSize = size2;

}

printf("\nG1:\n");

create(&firstMatrix, maxMatrixSize);

fill(firstMatrix, size1);

print(firstMatrix, size1);

printf("\n\nG2:\n");

create(&secondMatrix, maxMatrixSize);

fill(secondMatrix, size2);

print(secondMatrix, size2);

printf("\n\nОбъединение графов: \n");

int\*\* resultunionOfMatrix = disconnect(firstMatrix, secondMatrix, maxMatrixSize);

print(resultunionOfMatrix, maxMatrixSize);

printf("\n\nПересечение графов: \n");

int\*\* resultIntersectionMatrix = connect(firstMatrix, secondMatrix, maxMatrixSize);

print(resultIntersectionMatrix, maxMatrixSize);

printf("\n\nКольцевая сумма графов: \n");

int\*\* resultringsum = ringsum(firstMatrix, secondMatrix, maxMatrixSize);

print(resultringsum, maxMatrixSize);

}

**Вывод:** в ходе выполнения лабораторной работы была разработана программа, реализующая унарные и бинарные операции над графами.