Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»  
  
  
  
  
  
  
**ОТЧЁТ**

по лабораторной работе № 6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнил:

Студент групп 21ВВ1.3

Морозов К.В.

Приняли:

к.т.н., доцент Юрова О.А.

д.т.н., профессор, зав. каф. ВТ Митрохин М. А.

Пенза 2022

**Название:** Унарные и бинарные операции над графами

**Цель работы:** Выполнить унарные и бинарные операции, используя матрицу смежности.

**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.

**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 https://lh5.googleusercontent.com/Z2tMU1_zvmHJXGqOCTinKcXEGHvklWBKsqsD9Ab_JkbWQZ3XE2nsoA6LeueCxdgkBilXF6PWVOxP8WSASBncLa8fE6zOQQspekehp8i1gcSxEgX5iFnl3MLih5-nXYtMzZrwgj7zOS4ZkF_vfZIQEM6DzS_8m68hkF1h7fqoR9hQHjFWvyQ1i3D_b45EGn5ZbQ_5TFf0XQ *G*2

б) пересечения  *G* = *G*1 https://lh3.googleusercontent.com/jug0mrcPqopSpbpigzXkpBCQr5O4S5-WiOYReKMIo1CRaj4JwE1rylvgltxVXZuYZb5Rx8N-RvVHLJANvmrQnUe1C6apGOnUnQxtkZFGZrWogCzSlhysNVCnX-ATTzYI7aY9akL1hQ41wsLBEOUNN3b66ZlGWz-5_btI5MqmjbhDlxhzdaRU78HbdtfgP5IJvpr3Jjzp9g *G*2

в) кольцевой суммы *G* = *G*1 https://lh3.googleusercontent.com/Y51piDz5wat-7MopCSACAFrNz7IJ0yNytMFVrqMGScJSMrRnPnnn2zcASpZAD2T1kDAZ2NJqHZty_-GpCleDCxBO7IdZArEnUU5UwydI_uOXTNGAcp3qrdpyPw11cE49WdRIugCGuA0qVV0xWQhxqGYRMpNYTcpcvZiFmEud1qalgmHucc-Y0aQZSvZUCKfMboxMBnRdzw *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.

**Листинг:**

#include <iostream>

#include <set>

#include <vector>

#include <algorithm>

#include <stack>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <time.h>

#include <locale.h>

using namespace std;

int \_stateMenu;

void Menu()

{

printf("Выберите действие:\n1 - Вывести матрицы M1 и M2\n2 - Вывести матрицу M\n3 - Выполнение операции отождествления графов\n4 - Выполнение операции стягивания ребра\n5 - Выполнение операции расщепления вершины\n6 - Выполнение операции объединения\n7 - Выполнение операции пересечения\n8 - Выполнения операции кольцевой суммы\n9 - Выполнение операции декартова произведения\n0 - Выйти\n");

\_stateMenu = \_getch() - 48;

}

void PrintGraph(int N, int\*\* m)

{

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

{

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

{

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

}

printf("|\n");

}

}

int\*\* DekGr(int N1, int N2, int\*\* m, int\*\* n)

{

int\*\* h = new int\* [N1 \* N2];

for (int x1 = 0; x1 != N1; x1++) {

for (int y1 = 0; y1 != N2; y1++) {

h[x1 \* N2 + y1] = new int[N1 \* N2];

for (int x2 = 0; x2 != N1; x2++) {

for (int y2 = 0; y2 != N2; y2++) {

if (x1 == x2) {

h[x1 \* N2 + y1][x2 \* N2 + y2] = n[y1][y2];

}

else if (y1 == y2) {

h[x1 \* N2 + y1][x2 \* N2 + y2] = m[x1][x2];

}

else h[x1 \* N2 + y1][x2 \* N2 + y2] = 0;

}

}

}

}

return h;

}

int\*\* KolGr(int N1, int N2, int\*\* m, int\*\* n)

{

int min = 1;

int max = 1;

if (N1 > N2) {

min = N2;

max = N1;

}

else {

min = N1;

max = N2;

}

stack<int> S1;

stack<int> S2;

S1.push(0);

S2.push(0);

int\* edges1 = new int[N1] { 0 };

int\* edges2 = new int[N2] { 0 };

while (!S1.empty() && !S2.empty()) {

int edge1 = S1.top();

int edge2 = S2.top();

S1.pop();

S2.pop();

if (edges1[edge1] == 2 && edges2[edge2]) continue;

edges1[edge1] = 2;

edges2[edge2] = 2;

for (int i = 0; i != N1; i++) {

if (m[edge1][i] && edges1[i] != 2) {

S1.push(i);

edges1[i] = 1;

}

}

for (int i = 0; i != N2; i++) {

if (n[edge2][i] && edges2[i] != 2) {

S2.push(i);

edges2[i] = 1;

}

}

}

int k = 0;

for (int i = 0; i != N1; i++) {

if (!edges1[i]) k++;

}

for (int i = 0; i != N2; i++) {

if (!edges2[i]) k++;

}

if (k > 0) {

printf("Один из графов имеет изолированн%s вершин%c \n", k == 1 ? "ую" : "ые", k == 1 ? 'у' : 'ы');

int M[1][1]{ 0 };

return NULL;

}

int\*\* h = new int\* [max];

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

h[i] = new int[max];

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

{

h[i][j] = 0;

}

}

if (min != max)

{

for (int i = 0; i != min; i++) {

for (int j = i + 1; j != min; j++) {

if (m[i][j] != n[i][j]) {

h[i][j] = 1;

h[j][i] = 1;

}

}

}

for (int i = min; i != max; i++) {

for (int j = 0; j != max; j++) {

if (N1 == max) {

h[i][j] = m[i][j];

h[j][i] = m[i][j];

}

else {

h[i][j] = n[i][j];

h[j][i] = n[i][j];

}

}

}

}

else

{

for (int i = 0; i != min; i++) {

for (int j = i; j != min; j++) {

if (m[i][j] != n[i][j]) {

h[i][j] = 1;

h[j][i] = 1;

}

}

}

}

return h;

}

int\*\* PerGr(int N1, int N2, int\*\* m, int\*\* n)

{

int min = 1;

int max = 1;

if (N1 > N2) {

min = N2;

max = N1;

}

else {

min = N1;

max = N2;

}

int\*\* h = new int\* [min];

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

h[i] = new int[min];

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

{

h[i][j] = 0;

}

}

for (int i = 0; i != min; i++) {

for (int j = i + 1; j != min; j++) {

if (m[i][j] && n[i][j]) {

h[i][j] = 1;

h[j][i] = 1;

}

}

}

return h;

}

int\*\* ObGr(int N1, int N2, int\*\* m, int\*\* n)

{

int min = 1;

int max = 1;

if (N1 > N2) {

min = N2;

max = N1;

}

else {

min = N1;

max = N2;

}

int\*\* h = new int\* [max];

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

h[i] = new int[max];

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

{

h[i][j] = 0;

}

}

for (int i = 0; i != min; i++) {

for (int j = 0; j != min; j++) {

if (m[i][j] || n[i][j]) h[i][j] = 1;

}

}

for (int i = min; i != max; i++) {

for (int j = 0; j != max; j++) {

if (N1 == max) {

h[i][j] = m[i][j];

h[j][i] = m[i][j];

}

else {

h[i][j] = n[i][j];

h[j][i] = n[i][j];

}

}

}

return h;

}

int\*\* RasVer(int N, int\*\* m, int ver)

{

int\*\* h = new int\* [N + 1];

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

h[i] = new int[N + 1];

for (int j = 0; j < N + 1; j++)

{

h[i][j] = 0;

}

}

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

for (int j = 0; j != N; j++) {

if (!i) {

h[0][j + 1] = m[ver][j];

h[j + 1][0] = m[ver][j];

}

if (m[i][j]) h[i + 1][j + 1] = 1;

}

}

h[0][ver + 1] = 1;

h[ver + 1][0] = 1;

if (h[ver + 1][ver + 1]) h[0][0] = 1;

return h;

}

int\*\* StyagReb(int N, int\*\* m, int u, int v)

{

int\* ver = new int[2];

if (u > v) {

ver[0] = v;

ver[1] = u;

}

else

{

ver[1] = v;

ver[0] = u;

}

int\* h = new int[N - 2]{ 0 };

for (int i = 0; i < N - 2; i++) h[i] = 0;

for (int i = 0, j = 0; i != N; i++, j++) {

if (i != ver[0] && i != ver[1]) h[j] = i;

else j--;

}

int\*\* h1 = new int\* [N - 1];

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

h1[i] = new int[N - 1];

for (int j = 0; j < N - 1; j++)

{

h1[i][j] = 0;

}

}

for (int i = 0; i != 2; i++) {

for (int j = 0, x = 0; j != N; j++, x++) {

if (ver[0] == j) continue;

if (ver[1] == j && j != N - 1) j++;

if (ver[1] == N - 1 && j == N - 1) x--;

if (m[ver[i]][j]) {

h1[ver[0]][x] = 1;

h1[x][ver[0]] = 1;

}

}

}

h1[ver[0]][ver[0]] = 0;

for (int i = 0; i < N - 2; i++) {

for (int j = 0, y = 0; j < N - 2; j++, y++) {

if (m[h[i]][h[j]]) {

if (h[j] > ver[1] && h[i] > ver[1]) {

h1[h[i] - 1][h[j] - 1] = 1;

h1[h[j] - 1][h[i] - 1] = 1;

}

else if (h[j] > ver[1]) {

h1[h[i]][h[j] - 1] = 1;

h1[h[j] - 1][h[i]] = 1;

}

else if (h[i] > ver[1]) {

h1[h[i] - 1][h[j]] = 1;

h1[h[j]][h[i] - 1] = 1;

}

else {

h1[h[i]][h[j]] = 1;

h1[h[j]][h[i]] = 1;

}

}

}

}

return h1;

}

int\*\* OtogVer(int N, int\*\* m, int u, int v)

{

int\* ver = new int[2];

if (u > v) {

ver[0] = v;

ver[1] = u;

}

else

{

ver[1] = v;

ver[0] = u;

}

int\* h = new int[N - 2]{ 0 };

for (int i = 0; i < N - 2; i++) h[i] = 0;

for (int i = 0, j = 0; i != N; i++, j++) {

if (i != ver[0] && i != ver[1]) h[j] = i;

else j--;

}

int\*\* h1 = new int\* [N - 1];

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

h1[i] = new int[N - 1];

for (int j = 0; j < N - 1; j++)

{

h1[i][j] = 0;

}

}

for (int i = 0; i != 2; i++) {

for (int j = 0, x = 0; j != N; j++, x++) {

if (u == j || v == i) {

if (m[ver[0]][ver[1]] == 1 || m[ver[1]][ver[0]] == 1) {

h1[ver[0]][ver[0]] = 1;

}

}

if (ver[1] == j && j != N - 1) j++;

if (ver[1] == N - 1 && j == N - 1) x--;

if (m[ver[i]][j] == 1) {

h1[ver[0]][x] = 1;

h1[x][ver[0]] = 1;

}

}

}

for (int i = 0; i != N - 2; i++) {

for (int j = 0, y = 0; j != N - 2; j++, y++) {

if (m[h[i]][h[j]]) {

if (h[j] > ver[1] && h[i] > ver[1]) {

h1[h[i] - 1][h[j] - 1] = 1;

h1[h[j] - 1][h[i] - 1] = 1;

}

else if (h[j] > ver[1]) {

h1[h[i]][h[j] - 1] = 1;

h1[h[j] - 1][h[i]] = 1;

}

else if (h[i] > ver[1]) {

h1[h[i] - 1][h[j]] = 1;

h1[h[j]][h[i] - 1] = 1;

}

else {

h1[h[i]][h[j]] = 1;

h1[h[j]][h[i]] = 1;

}

}

}

}

return h1;

}

int main()

{

setlocale(0, "rus");

srand(time(NULL));

bool M1T = true, M2T = true;

int n1, n2, n, u, v, ver, min, max;

int\*\* M1;

int\*\* M2;

int\*\* M;

n = 2;

printf("Введите количество вершин в графе G1 (> 2): ");

scanf\_s("%d", &n1);

printf("Введите количество вершин в графе G2 (> 2): ");

scanf\_s("%d", &n2);

M1 = new int\* [n1];

M2 = new int\* [n2];

M = new int\* [n];

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

{

M[i] = new int[n];

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

{

M[i][j] = 0;

}

}

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

{

M1[i] = new int[n1];

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

{

M1[i][j] = rand() % 2;

}

}

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

{

M2[i] = new int[n2];

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

{

M2[i][j] = rand() % 2;

}

}

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

{

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

{

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

else M1[i][j] = 0;

}

}

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

{

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

{

if (M2[j][i] == 1) M2[i][j] = 1;

else M2[i][j] = 0;

}

}

system("pause");

system("cls");

Menu();

while (\_stateMenu != 0)

{

switch (\_stateMenu)

{

case 1:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

printf("\nГраф G2:\n");

PrintGraph(n2, M2);

system("pause");

system("cls");

Menu();

break;

case 2:

system("cls");

printf("Граф G:\n");

PrintGraph(n, M);

system("pause");

system("cls");

Menu();

break;

case 3:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

for (;;) {

printf("Введите 2 вершины для их отождествления в графе G1: ");

scanf\_s("%d %d", &u, &v);

if (u == v) {

printf("Вы ввели неправильное значение\n");

continue;

}

if ((u >= n1 || u < 0) || (v >= n1 || v < 0)) {

printf("Вы ввели неправильное значение\n");

continue;

}

break;

}

printf("Граф G1:\n");

M1 = OtogVer(n1, M1, u, v);

n1 = n1 - 1;

PrintGraph(n1, M1);

printf("Граф G2:\n");

PrintGraph(n2, M2);

for (;;) {

printf("Введите 2 вершины для их отождествления в графе G2: ");

scanf\_s("%d %d", &u, &v);

if (u == v) {

printf("Вы ввели неправильное значение\n");

continue;

}

if ((u >= n2 || u < 0) || (v >= n2 || v < 0)) {

printf("Вы ввели неправильное значение\n");

continue;

}

break;

}

printf("\nГраф G2:\n");

M2 = OtogVer(n2, M2, u, v);

n2 = n2 - 1;

PrintGraph(n2, M2);

system("pause");

system("cls");

Menu();

break;

case 4:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

for (;;)

{

printf("Введите 2 вершины для стягивания ребра между ними в графе G1: ");

scanf\_s("%d %d", &u, &v);

if (u > v) {

int tmp = u;

u = v;

v = tmp;

}

if ((u > n1 - 1 || u < 0 || v > n1 - 1 || v < 0)) {

printf("Вы ввели неправильное значение\n");

continue;

}

if (u == v) {

printf("Вы ввели неправильное значение\n");

continue;

}

if (!(M1[u][v] && M1[v][u])) {

printf("Данные вершины не образуют ребро в графе G1\n");

M1T = false;

}

break;

}

if (M1T)

{

printf("Граф G1:\n");

M1 = StyagReb(n1, M1, u, v);

n1 = n1 - 1;

PrintGraph(n1, M1);

}

printf("Граф G2:\n");

PrintGraph(n2, M2);

for (;;)

{

printf("Введите 2 вершины для стягивания ребра между ними в графе G2: ");

scanf\_s("%d %d", &u, &v);

if (u > v) {

int tmp = u;

u = v;

v = tmp;

}

if ((u > n2 - 1 || u < 0) || (v > n2 - 1 || v < 0)) {

printf("Вы ввели неправильное значение\n");

continue;

}

if (u == v) {

printf("Вы ввели неправильное значение\n");

continue;

}

if (!(M2[u][v] && M2[v][u])) {

printf("Данные вершины не образуют ребро в графе G2\n");

M2T = false;

}

break;

}

if (M2T)

{

printf("\nГраф G2:\n");

M2 = StyagReb(n2, M2, u, v);

n2 = n2 - 1;

PrintGraph(n2, M2);

}

M1T = true;

M2T = true;

system("pause");

system("cls");

Menu();

break;

case 5:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

for (;;) {

printf("Введите вершину для расщепления в графе G1: ");

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

if (ver > n1 - 1 || ver < 0) {

printf("Вы ввели неправильное значение\n");

continue;

}

break;

}

printf("Граф G1:\n");

M1 = RasVer(n1, M1, ver);

n1 = n1 + 1;

PrintGraph(n1, M1);

printf("Граф G2:\n");

PrintGraph(n2, M2);

for (;;) {

printf("Введите вершину для расщепления в графе G2: ");

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

if (ver > n2 - 1 || ver < 0) {

printf("Вы ввели неправильное значение\n");

continue;

}

break;

}

printf("Граф G2:\n");

M2 = RasVer(n2, M2, ver);

n2 = n2 + 1;

PrintGraph(n2, M2);

system("pause");

system("cls");

Menu();

break;

case 6:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

printf("Граф G2:\n");

PrintGraph(n2, M2);

if (n1 > n2)

{

max = n1;

min = n2;

}

else

{

max = n2;

min = n1;

}

printf("Объединение графов G1 и G2:\n");

M = ObGr(n1, n2, M1, M2);

n = max;

PrintGraph(n, M);

system("pause");

system("cls");

Menu();

break;

case 7:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

printf("Граф G2:\n");

PrintGraph(n2, M2);

if (n1 > n2)

{

max = n1;

min = n2;

}

else

{

max = n2;

min = n1;

}

printf("Пересечение графов G1 и G2:\n");

M = PerGr(n1, n2, M1, M2);

n = min;

PrintGraph(n, M);

system("pause");

system("cls");

Menu();

break;

case 8:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

printf("Граф G2:\n");

PrintGraph(n2, M2);

if (n1 > n2)

{

max = n1;

min = n2;

}

else

{

max = n2;

min = n1;

}

printf("Кольцевая сумма графов G1 и G2:\n");

M = KolGr(n1, n2, M1, M2);

n = max;

if (M != NULL) PrintGraph(n, M);

system("pause");

system("cls");

Menu();

break;

case 9:

system("cls");

printf("Граф G1:\n");

PrintGraph(n1, M1);

printf("Граф G2:\n");

PrintGraph(n2, M2);

printf("Декартово произведение графов G1 и G2:\n");

M = DekGr(n1, n2, M1, M2);

n = n1 \* n2;

PrintGraph(n, M);

system("pause");

system("cls");

Menu();

break;

case 0:

exit(EXIT\_SUCCESS);

default:

system("cls");

printf("Неправильная цифра\n");

system("pause");

system("cls");

Menu();

break;

}

}

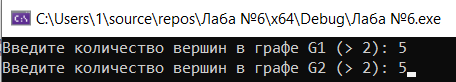
}

**Вывод результатов:**

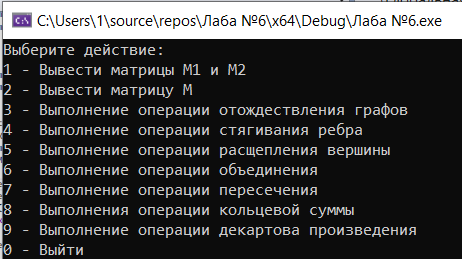
**Задание 1**

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.

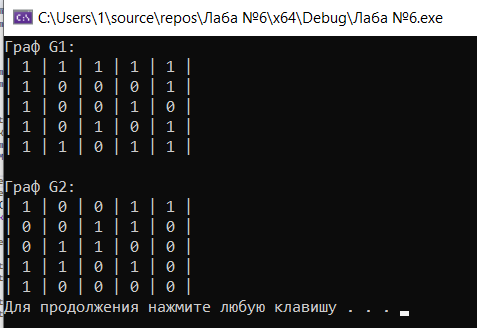
Вывод значений в начале программы:

****

Вывод меню:

****

Первый пункт меню как раз выводит на экран полученные матрицы G1 и G2:



**Задание 2**

1. Для матричной формы представления графов выполните операцию:

а) отождествления вершин

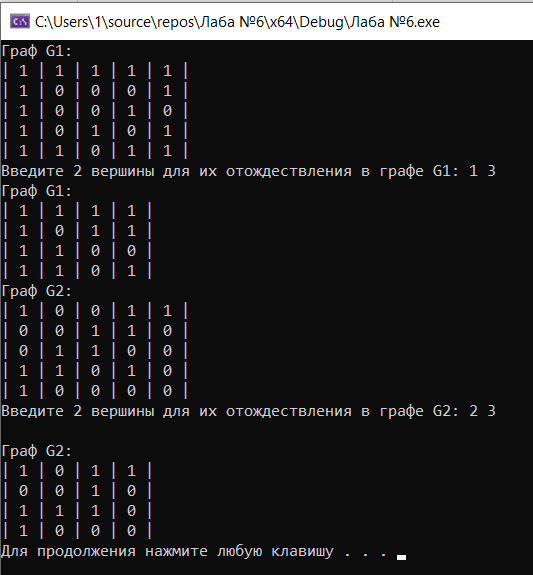
б) стягивания ребра

в) расщепления вершины

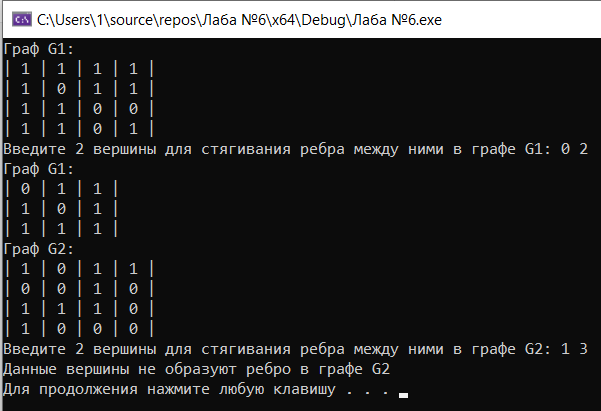
Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

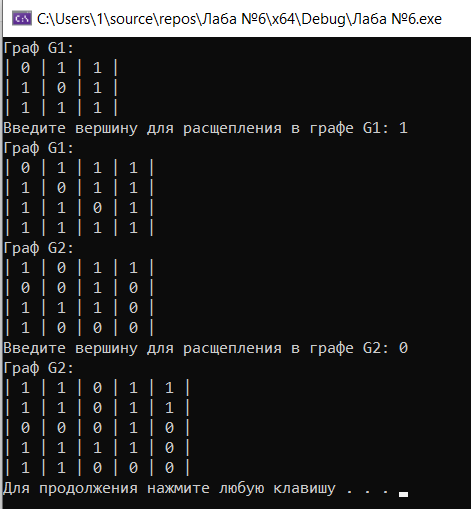
Третий пункт меню – отождествление графов. (отсчёт начинается с 0):



Четвёртый пункт меню – стягивание ребра:



Пятый пункт меню – расщепление вершины:



**Задание 3**

1. Для матричной формы представления графов выполните операцию:

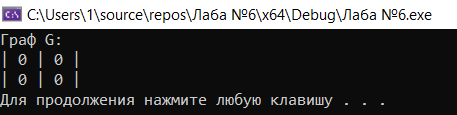
а) объединения *G* = *G*1 https://lh5.googleusercontent.com/Z2tMU1_zvmHJXGqOCTinKcXEGHvklWBKsqsD9Ab_JkbWQZ3XE2nsoA6LeueCxdgkBilXF6PWVOxP8WSASBncLa8fE6zOQQspekehp8i1gcSxEgX5iFnl3MLih5-nXYtMzZrwgj7zOS4ZkF_vfZIQEM6DzS_8m68hkF1h7fqoR9hQHjFWvyQ1i3D_b45EGn5ZbQ_5TFf0XQ *G*2

б) пересечения  *G* = *G*1 https://lh3.googleusercontent.com/jug0mrcPqopSpbpigzXkpBCQr5O4S5-WiOYReKMIo1CRaj4JwE1rylvgltxVXZuYZb5Rx8N-RvVHLJANvmrQnUe1C6apGOnUnQxtkZFGZrWogCzSlhysNVCnX-ATTzYI7aY9akL1hQ41wsLBEOUNN3b66ZlGWz-5_btI5MqmjbhDlxhzdaRU78HbdtfgP5IJvpr3Jjzp9g *G*2

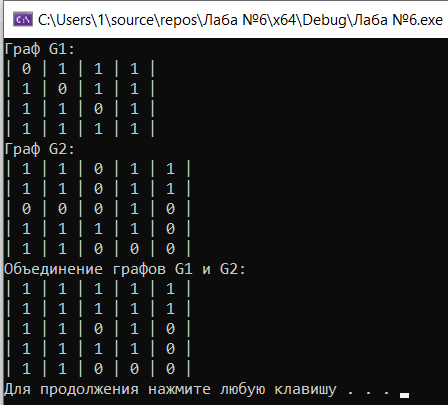
в) кольцевой суммы *G* = *G*1 https://lh3.googleusercontent.com/Y51piDz5wat-7MopCSACAFrNz7IJ0yNytMFVrqMGScJSMrRnPnnn2zcASpZAD2T1kDAZ2NJqHZty_-GpCleDCxBO7IdZArEnUU5UwydI_uOXTNGAcp3qrdpyPw11cE49WdRIugCGuA0qVV0xWQhxqGYRMpNYTcpcvZiFmEud1qalgmHucc-Y0aQZSvZUCKfMboxMBnRdzw *G*2

Результат выполнения операции выведите на экран.

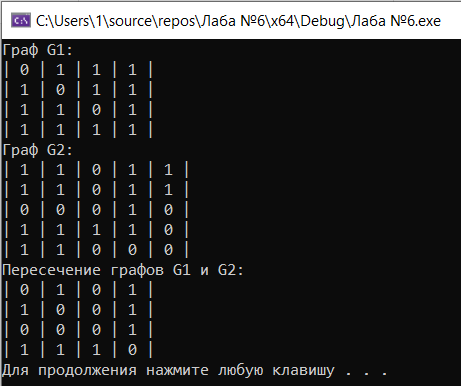
Второй пункт меню – вывод результата последней операции среди 6-9 пунктов.



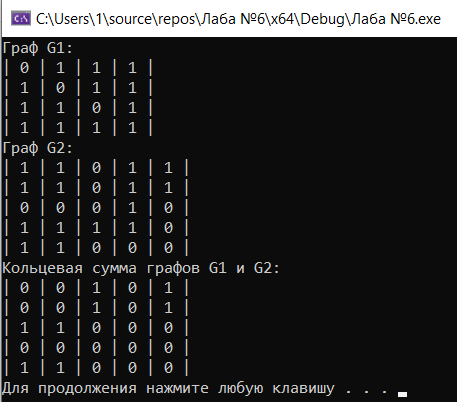
Шестой пункт меню – объединение графов:



Седьмой пункт меню – пересечение графов:



Восьмой пункт меню – кольцевая сумма графов:

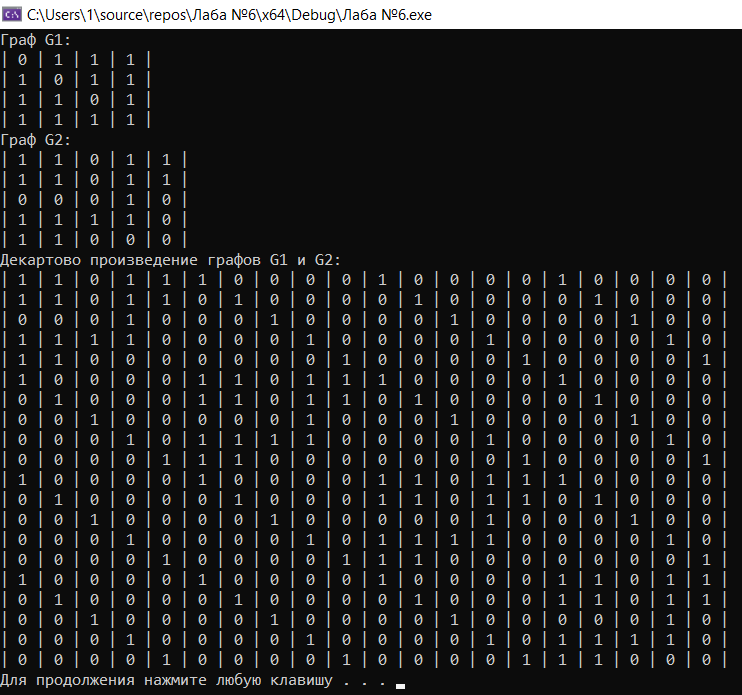


**Задание 4 \***

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

Результат выполнения операции выведите на экран.

Девятый пункт меню – декартовое произведение графов:



**Вывод:** мы создали программу, которая выполняет бинарные и унарные операции над графами, и она корректно работает.