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»

Выполнила:

студентка группы 22ВВВ2

Расторгуева К.В.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Унарные и бинарные операции над графами.

**Цель работы**

Изучение унарных и бинарных операций над графами, их реализация.

**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.
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**

Генерируются матрицы M1 и М2 размеров n1 и n2 (n1 и n2 вводятся пользователем с клавиатуры). Используется генератор случайных чисел. Для формирования списков смежности используется структура, состоящая из вершин, указывающих на связанные с ними вершинами.

Для отождествления вершин 1 и 2 копируются данные их вершины 2 в вершину 1. Вершина 2 удаляется. При этом образуется петля в вершине 1. Для стягивания ребра вершин 1 и 2 копируются данные их вершины 2 в вершину 1. Вершина 2 удаляется. Для расщепления вершины 1 создается новая вершина 1.1, содержащая все связи расщепляемой вершины. При этом устанавливается связь между вершинами 1 и 1.1.

Для объединения графов используется операция «логического или». Для пересечения графов – операция «логического и». Для кольцевой суммы графов используется операция «исключающего или», затем удаляются изолированные вершины.

Декартово произведение находится при помощи алгебраической теории графов. Если граф G1 имеет n1 вершин и n1 × n1 матрицу смежности А1, а граф G2 имеет n2 вершин и n2 × n2 матрицу смежности А2, то матрица смежности задается формулой

А12 = А1 ⊗ En2 + En1 ⊗ А2,

где ⊗ – произведение Кронекера матриц, а En – n × nединичная матрица.

**Результат выполнения программы**

Результат работы программы представлены на рис. 1-4.

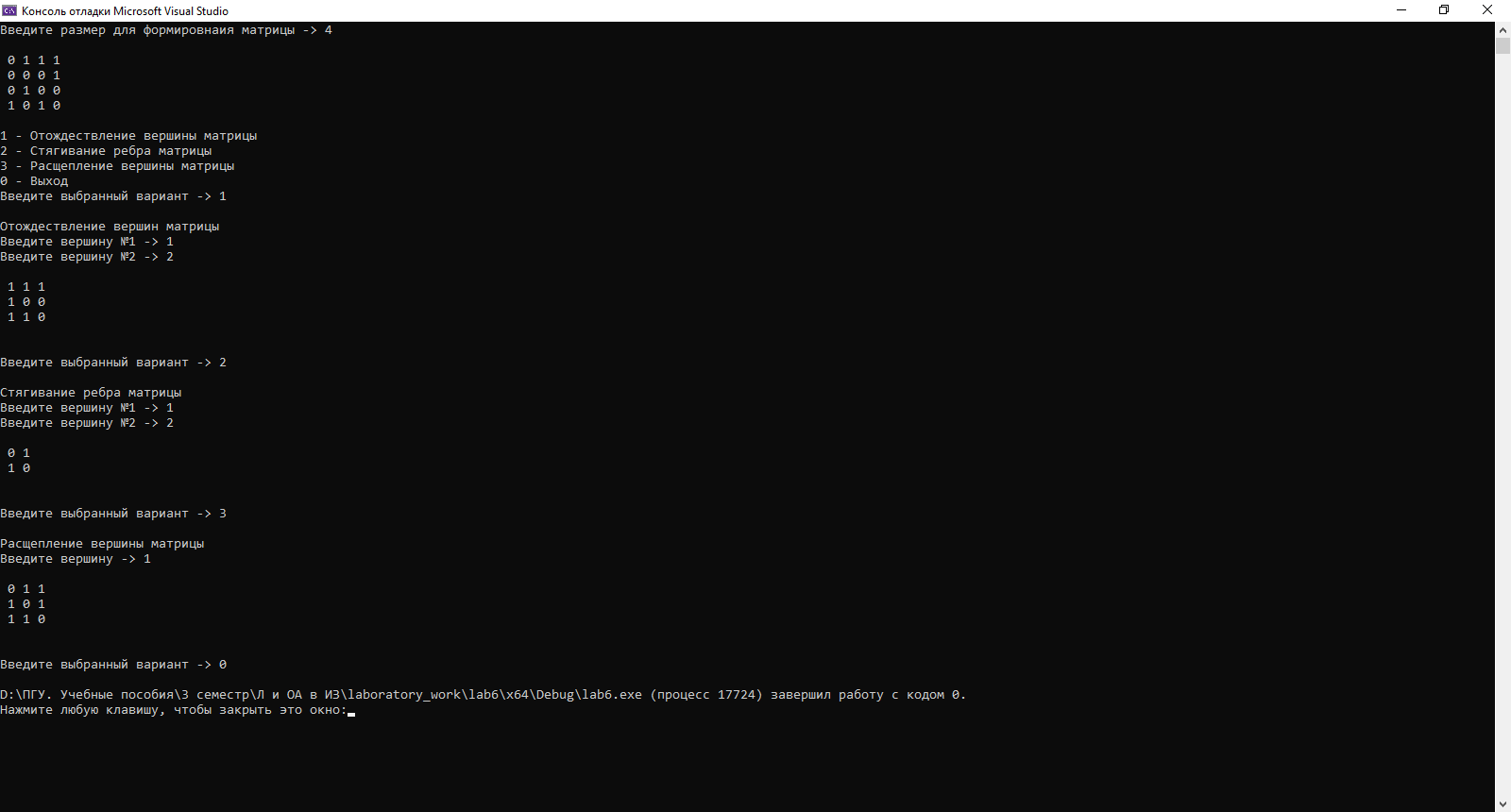


Рисунок 1 – операции стягивания ребра, отождествления и расщепления вершин

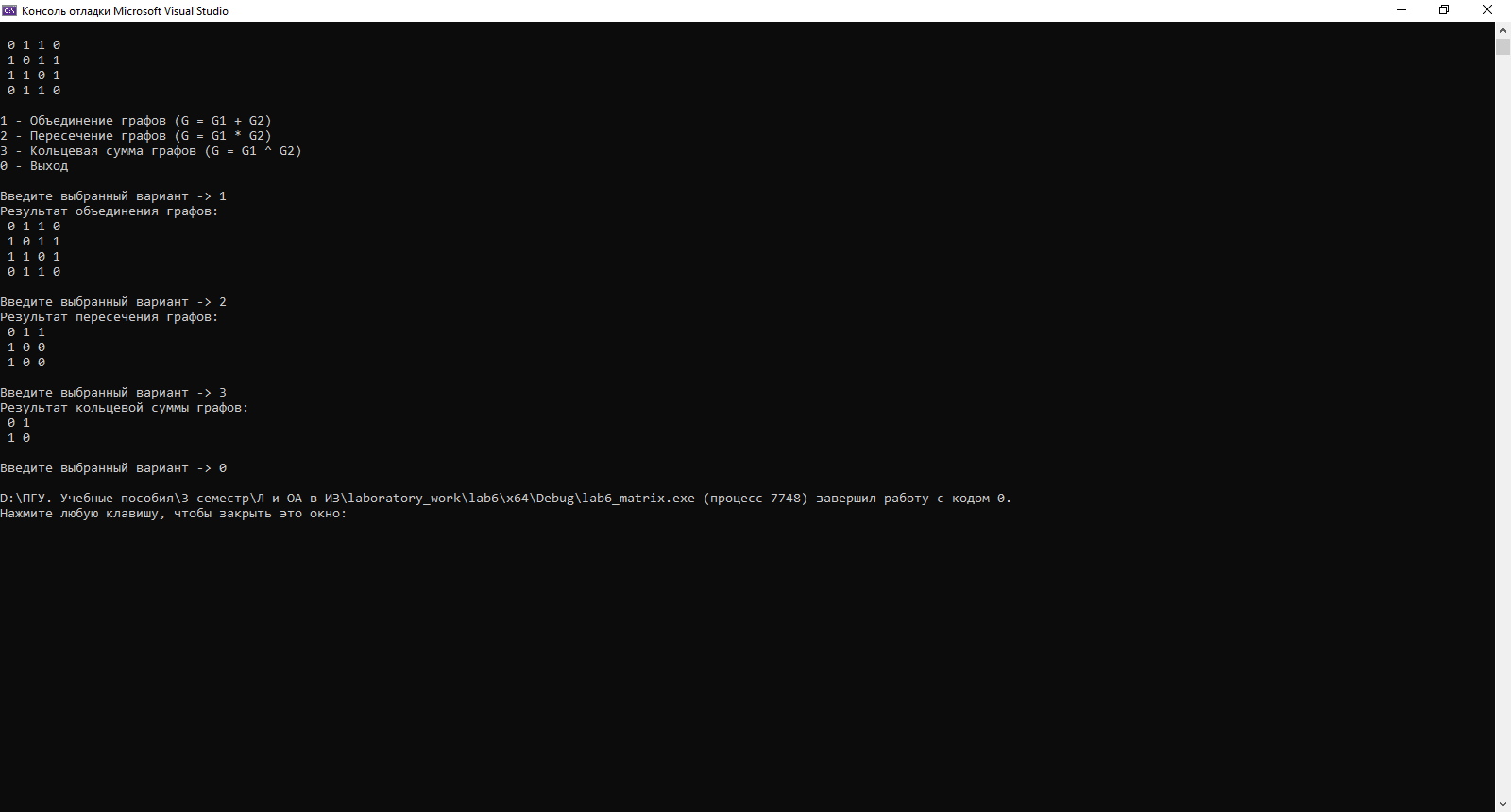


Рисунок 2 – операции над графами

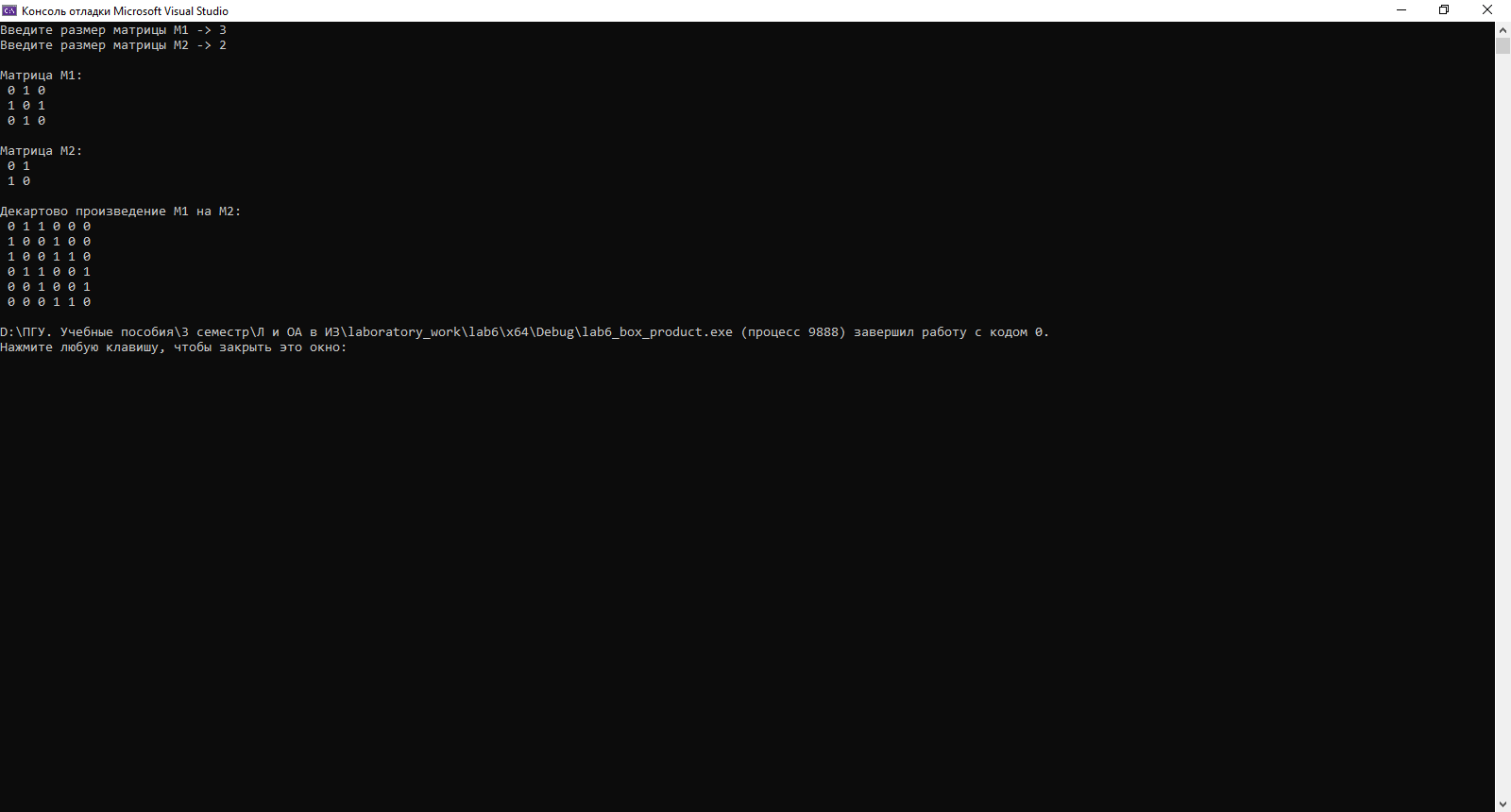


Рисунок 3 – декартово произведение

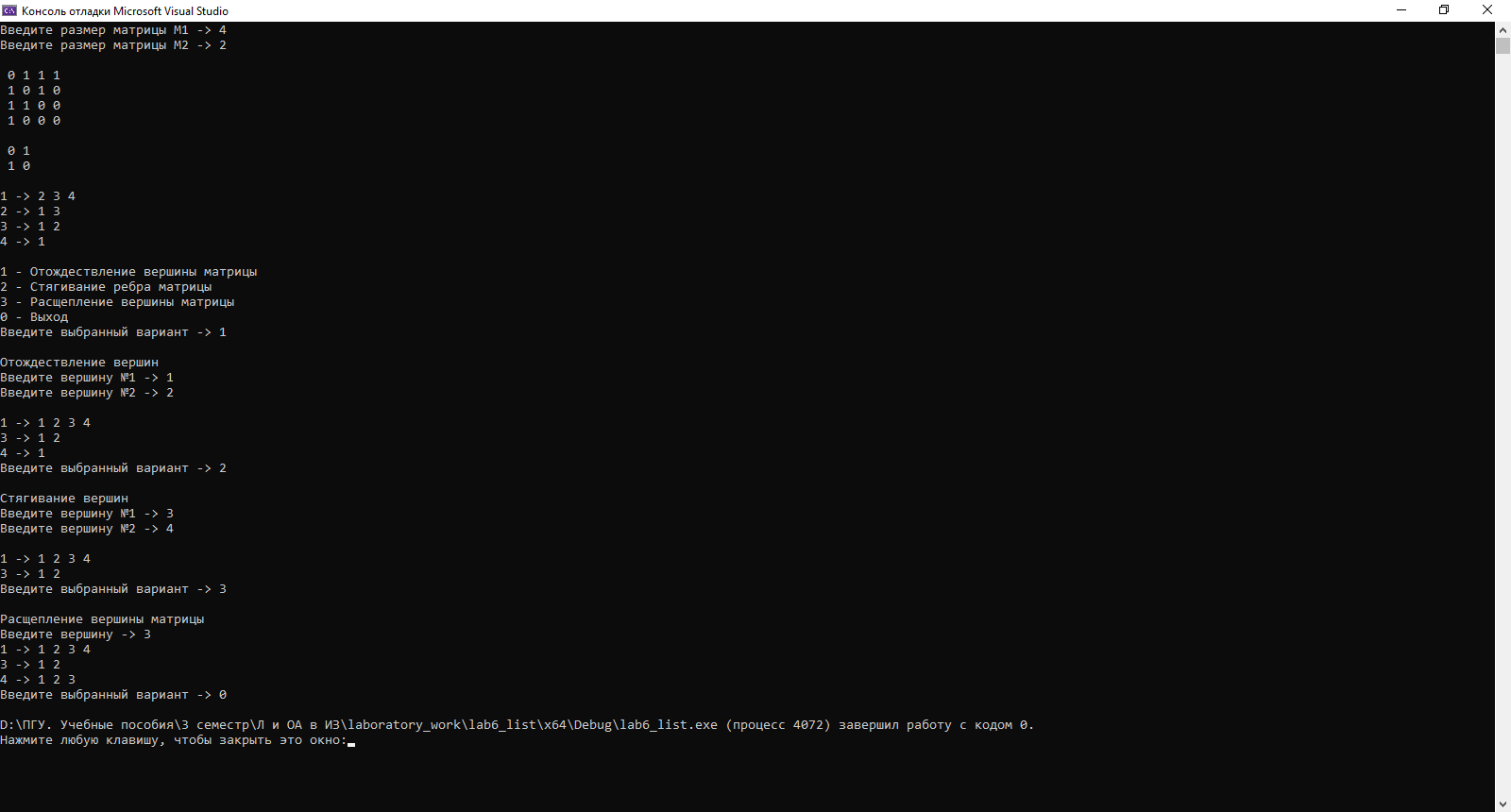


Рисунок 4 – создание списков смежности и операции над ними

**Выводы**

В ходе выполнения лабораторной работы была разработана на языке Си программа, осуществляющая работу с неориентированным графом:

– генерация матрицы смежности и ее преобразование в списки смежности;

– отождествление, расщепление вершин, стягивание ребра графов, представленных в матричной форме и в виде списков смежности;

– объединение, пересечение, кольцевая сумма графов, представленных в матричной форме;

– декартово произведение графов, представленных в матричной форме.

Результаты работы программы совпали с результатами расчета вручную.

**Приложение**

**Программа**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

void Output\_matrix(int\*\* G1, int n) {

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

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

printf(" %d", G1[i][j]);

}

printf("\n");

}

return;

}

int\*\* Otogd\_tops(int \*\*G1, int n) {

int v1\_keyboard, v2\_keyboard, v\_tmp;

printf("\nОтождествление вершин матрицы\n");

printf("Введите вершину №1 -> ");

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

printf("Введите вершину №2 -> ");

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

printf("\n");

int\*\* G1\_otogd = (int\*\*)malloc((n - 1) \* sizeof(int\*));

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

G1\_otogd[i] = (int\*)malloc((n - 1) \* sizeof(int));

}

if (v1\_keyboard > v2\_keyboard) {

v\_tmp = v1\_keyboard;

v1\_keyboard = v2\_keyboard;

v2\_keyboard = v\_tmp;

}

v1\_keyboard--;

v2\_keyboard--;

for (int i = 0; i < n; i++) { //"ИЛИ" для строки

if (i == v1\_keyboard) {

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

G1[v1\_keyboard][j] = G1[v1\_keyboard][j] || G1[v2\_keyboard][j];

G1[v2\_keyboard][j] = 0;

}

}

}

for (int j = 0; j < n; j++) { //"ИЛИ" для столбца

if (j == v1\_keyboard) {

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

G1[i][v1\_keyboard] = G1[i][v1\_keyboard] || G1[i][v2\_keyboard];

G1[i][v2\_keyboard] = 0;

}

}

}

int i\_otogd = 0; //отождествление вершин

int j\_otogd = 0;

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

if (i != v2\_keyboard) {

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

if (j != v2\_keyboard) {

if ((i\_otogd == v1\_keyboard) and (i\_otogd == j\_otogd)) G1\_otogd[i\_otogd][j\_otogd] = 1; //петля

else G1\_otogd[i\_otogd][j\_otogd] = G1[i][j];

//printf(" %d", G1\_otogd[i\_otogd][j\_otogd]);

j\_otogd++;

}

}

i\_otogd++;

j\_otogd = 0;

//printf("\n");

}

}

return G1\_otogd;

}

int \*\* Connection\_rib(int\*\* G1, int n) {

int v1\_keyboard, v2\_keyboard, v\_tmp;

printf("\nСтягивание ребра матрицы\n");

printf("Введите вершину №1 -> ");

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

printf("Введите вершину №2 -> ");

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

printf("\n");

int\*\* G1\_connect = (int\*\*)malloc((n - 1) \* sizeof(int\*));

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

G1\_connect[i] = (int\*)malloc((n - 1) \* sizeof(int));

}

if (v1\_keyboard > v2\_keyboard) {

v\_tmp = v1\_keyboard;

v1\_keyboard = v2\_keyboard;

v2\_keyboard = v\_tmp;

}

v1\_keyboard--;

v2\_keyboard--;

for (int i = 0; i < n; i++) { //"ИЛИ" для строки

if (i == v1\_keyboard) {

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

G1[v1\_keyboard][j] = G1[v1\_keyboard][j] || G1[v2\_keyboard][j];

G1[v2\_keyboard][j] = 0;

}

}

}

for (int j = 0; j < n; j++) { //"ИЛИ" для столбца

if (j == v1\_keyboard) {

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

G1[i][v1\_keyboard] = G1[i][v1\_keyboard] || G1[i][v2\_keyboard];

G1[i][v2\_keyboard] = 0;

}

}

}

int i\_otogd = 0; //стягивание вершин

int j\_otogd = 0;

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

if (i != v2\_keyboard) {

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

if (j != v2\_keyboard) {

if ((i\_otogd == v1\_keyboard) and (i\_otogd == j\_otogd)) G1\_connect[i\_otogd][j\_otogd] = 0; //удаление петли

else G1\_connect[i\_otogd][j\_otogd] = G1[i][j];

//printf(" %d", G1\_otogd[i\_otogd][j\_otogd]);

j\_otogd++;

}

}

i\_otogd++;

j\_otogd = 0;

//printf("\n");

}

}

return G1\_connect;

}

int\*\* Destroy\_tops(int\*\* G1, int n) {

int v\_keyboard, v2\_keyboard, v\_tmp;

printf("\nРасщепление вершины матрицы\n");

printf("Введите вершину -> ");

scanf("%d", &v\_keyboard);

printf("\n");

n++;

int\*\* G1\_destroy = (int\*\*)malloc(n \* sizeof(int\*));

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

G1\_destroy[i] = (int\*)malloc(n \* sizeof(int));

}

v\_keyboard--;

int i\_destroy = 0; //расщепление вершины

int j\_destroy = 0;

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

if (i == v\_keyboard) {

for (int j = 0; j < n - 1; j++) {//повторная запись данных для новой вершины (строка)

G1\_destroy[i\_destroy][j\_destroy] = G1[i][j];

G1\_destroy[i\_destroy + 1][j\_destroy] = G1[i][j];

if (j == v\_keyboard) {

G1\_destroy[i\_destroy][j\_destroy + 1] = G1[i][j];

G1\_destroy[i\_destroy + 1][j\_destroy + 1] = G1[i][j];

j\_destroy++;

}

j\_destroy++;

}

i\_destroy += 2;

j\_destroy = 0;

}

else {

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

if (j == v\_keyboard) {

G1\_destroy[i\_destroy][j\_destroy] = G1[i][j];

G1\_destroy[i\_destroy][j\_destroy + 1] = G1[i][j]; //повторная запись данных для новой вершины (столбец)

j\_destroy += 2;

}

else {

G1\_destroy[i\_destroy][j\_destroy] = G1[i][j];

j\_destroy++;

}

}

i\_destroy++;

j\_destroy = 0;

}

}

G1\_destroy[v\_keyboard][v\_keyboard + 1] = 1;

G1\_destroy[v\_keyboard + 1][v\_keyboard] = 1;

return G1\_destroy;

}

int main(){

int menu\_answer;

int n, v1, v2;

int start = 1;

setlocale(0, "rus");

printf("Введите размер для формировнаия матрицы -> ");

scanf("%d", &n);

printf("\n");

int\*\* G1 = (int\*\*)malloc(n \* sizeof(int\*));

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

G1[i] = (int\*)malloc(n \* sizeof(int));

}

srand(time(0));

for (int i = 0; i < n; i++) { //формирование матрицы смежности

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

if (i == j) G1[i][j] = 0;

else G1[i][j] = rand() % 2;

printf(" %d", G1[i][j]);

}

printf("\n");

}

//меню

printf("\n1 - Отождествление вершины матрицы\n");

printf("2 - Стягивание ребра матрицы\n");

printf("3 - Расщепление вершины матрицы\n");

printf("0 - Выход\n");

while (start){

printf("Введите выбранный вариант -> ");

scanf("%d", &menu\_answer);

if (menu\_answer == 0) {

start = 0;

}

else {

switch (menu\_answer) {

case(1): {

G1 = Otogd\_tops(G1, n);

n--;

Output\_matrix(G1, n);

printf("\n\n");

break;

}

case(2): {

G1 = Connection\_rib(G1, n);

n--;

Output\_matrix(G1, n);

printf("\n\n");

break;

}

case(3): {

G1 = Destroy\_tops(G1, n);

n++;

Output\_matrix(G1, n);

printf("\n\n");

break;

}

case(0): {

break;

}

}

}

}

}

**Операции над графами**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

void Output\_matrix(int\*\* G1, int n) { //вывод кольцевой суммы

int col\_one;

int isolate\_top[10];

int flag = 0;

int flag\_j = 0;

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

isolate\_top[i] = -1;

col\_one = 0;

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

if (G1[i][j] == 1) col\_one++;

}

if (col\_one == 0) {

isolate\_top[i] = i;

}

}

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

flag = 0;

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

if (i == isolate\_top[i\_is]) {

flag = 1;

break;

}

}

if (flag == 1) continue;

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

flag\_j = 0;

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

if (j == isolate\_top[i\_is]) {

flag\_j = 1;

break;

}

}

if (flag\_j == 0)

printf(" %d", G1[i][j]);

}

printf("\n");

}

return;

}

void Print\_matrix(int\*\* G, int n) {

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

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

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

}

printf("\n");

}

}

int main() {

int n1, n2, n\_tmp, menu\_answer, min\_n, max\_n;

int start = 1;

setlocale(0, "rus");

printf("Введите размер матрицы М1 -> ");

scanf("%d", &n1);

printf("Введите размер матрицы М2 -> ");

scanf("%d", &n2);

printf("\n");

if (n2 > n1) {

n\_tmp = n1;

n1 = n2;

n2 = n\_tmp;

min\_n = n1;

max\_n = n2;

}

min\_n = n2;

max\_n = n1;

int\*\* G1 = (int\*\*)malloc(min\_n \* sizeof(int\*));

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

G1[i] = (int\*)malloc(min\_n \* sizeof(int));

}

int\*\* G2 = (int\*\*)malloc(max\_n \* sizeof(int\*));

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

G2[i] = (int\*)malloc(max\_n \* sizeof(int));

}

srand(time(0));

for (int i = 0; i < min\_n; i++) { //формирование матрицы смежности М1

for (int j = 0; j < min\_n; j++) {

if (j >= i) {

if (i == j) G1[i][j] = 0;

else {

G1[i][j] = rand() % 2;

G1[j][i] = G1[i][j];

}

}

printf(" %d", G1[i][j]);

}

printf("\n");

}

printf("\n\n");

for (int i = 0; i < max\_n; i++) { //формирование матрицы смежности М2

for (int j = 0; j < max\_n; j++) {

if (j >= i) {

if (i == j) G2[i][j] = 0;

else {

G2[i][j] = rand() % 2;

G2[j][i] = G2[i][j];

}

}

printf(" %d", G2[i][j]);

}

printf("\n");

}

printf("\n1 - Объединение графов (G = G1 + G2)\n"); //операции над графами

printf("2 - Пересечение графов (G = G1 \* G2)\n");

printf("3 - Кольцевая сумма графов (G = G1 ^ G2)\n");

printf("0 - Выход\n");

while (start) {

printf("\nВведите выбранный вариант -> ");

scanf("%d", &menu\_answer);

if (menu\_answer == 0) {

start = 0;

}

else {

switch (menu\_answer) {

case(1): {

int\*\* G3 = (int\*\*)malloc(max\_n \* sizeof(int\*));

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

G3[i] = (int\*)malloc(max\_n \* sizeof(int));

}

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

for (int j = 0; j < max\_n; j++) {

G3[i][j] = G2[i][j];

}

}

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

for (int j = 0; j < min\_n; j++) {

G3[i][j] = G2[i][j] | G1[i][j];

}

}

printf("Результат объединения графов:\n");

Print\_matrix(G3, max\_n);

break;

}

case(2): {

int\*\* G3 = (int\*\*)malloc(min\_n \* sizeof(int\*));

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

G3[i] = (int\*)malloc(min\_n \* sizeof(int));

}

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

for (int j = 0; j < min\_n; j++) {

G3[i][j] = G2[i][j] & G1[i][j];

}

}

printf("Результат пересечения графов:\n");

Print\_matrix(G3, min\_n);

break;

}

case(3): {

int\*\* G3 = (int\*\*)malloc(min\_n \* sizeof(int\*));

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

G3[i] = (int\*)malloc(min\_n \* sizeof(int));

}

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

for (int j = 0; j < min\_n; j++) {

G3[i][j] = G2[i][j] ^ G1[i][j];

}

}

printf("Результат кольцевой суммы графов:\n");

Output\_matrix(G3, min\_n);

break;

}

case(0): {

break;

}

}

}

}

return 0;

}

**Декартово произведение**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

int main() {

int n1, n2;

int v1\_keyboard, v2\_keyboard, v\_tmp, v1, v2;

int size\_G = 0;

//int col\_one;

setlocale(0, "rus");

printf("Введите размер матрицы М1 -> ");

scanf("%d", &n1);

printf("Введите размер матрицы М2 -> ");

scanf("%d", &n2);

printf("\n");

int\*\* G1 = (int\*\*)malloc(n1 \* sizeof(int\*));

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

G1[i] = (int\*)malloc(n1 \* sizeof(int));

}

int\*\* G2 = (int\*\*)malloc(n2 \* sizeof(int\*));

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

G2[i] = (int\*)malloc(n2 \* sizeof(int));

}

printf("Матрица М1:\n");

srand(time(0));

for (int i = 0; i < n1; i++) { //формирование матрицы смежности G1

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

if (j >= i) { //неориентированный граф

if (i == j) G1[i][j] = 0;

else {

G1[i][j] = rand() % 2;

G1[j][i] = G1[i][j];

}

}

/\*if (i == j) G1[i][j] = 0; //ориентированный граф

else G1[i][j] = rand() % 2;\*/

printf(" %d", G1[i][j]);

}

printf("\n");

}

printf("\nМатрица М2:\n");

for (int i = 0; i < n2; i++) { //формирование матрицы смежности G2

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

if (j >= i) { //неориентированный граф

if (i == j) G2[i][j] = 0;

else {

G2[i][j] = rand() % 2;

G2[j][i] = G2[i][j];

}

}

/\*if (i == j) G1[i][j] = 0; //ориентированный граф

else G1[i][j] = rand() % 2;\*/

printf(" %d", G2[i][j]);

}

printf("\n");

}

int\*\* E1 = (int\*\*)malloc(n1 \* sizeof(int\*));

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

E1[i] = (int\*)malloc(n1 \* sizeof(int));

}

int\*\* E2 = (int\*\*)malloc(n2 \* sizeof(int\*));

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

E2[i] = (int\*)malloc(n2 \* sizeof(int));

}

for (int i = 0; i < n1; i++) { //формирование единичной матрицы E1 для G1

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

if (i == j) E1[i][j] = 1;

else E1[i][j] = 0;

//printf(" %d", E1[i][j]);

}

}

for (int i = 0; i < n2; i++) { //формирование единичной матрицы E2 для G2

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

if (i == j) E2[i][j] = 1;

else E2[i][j] = 0;

//printf(" %d", E2[i][j]);

}

}

int\*\* Rez1 = (int\*\*)malloc((n1\*n2) \* sizeof(int\*));

for (int i = 0; i < (n1 \* n2); i++) {

Rez1[i] = (int\*)malloc((n1 \* n2) \* sizeof(int));

}

int\*\* Rez2 = (int\*\*)malloc((n1 \* n2) \* sizeof(int\*));

for (int i = 0; i < (n1 \* n2); i++) {

Rez2[i] = (int\*)malloc((n1 \* n2) \* sizeof(int));

}

int j\_r = 0;

int i\_r = 0;

int tab = 0;

int flag\_first = 1;

for (int j\_G1 = 0; j\_G1 < n1; j\_G1++) { //транзисторное произведение А1 и Е2

i\_r = 0;

for (int i\_G1 = 0; i\_G1 < n1; i\_G1++) {

for (int i\_E = 0; i\_E < n2; i\_E++) {

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

j\_r = j\_E + tab;

Rez1[i\_r][j\_r] = G1[i\_G1][j\_G1] \* E2[i\_E][j\_E];

//printf("[%d][%d] = %d\n", i\_r, j\_r, Rez[i\_r][j\_r]);

j\_r++;

}

j\_r = 0;

i\_r++;

}

}

flag\_first = 0;

tab = tab + n2;

}

j\_r = 0;

i\_r = 0;

tab = 0;

flag\_first = 1;

for (int j\_E = 0; j\_E < n1; j\_E++) { //транзисторное произведение Е1 и А2

i\_r = 0;

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

for (int i\_G2 = 0; i\_G2 < n2; i\_G2++) {

for (int j\_G2 = 0; j\_G2 < n2; j\_G2++) {

j\_r = j\_G2 + tab;

Rez2[i\_r][j\_r] = G2[i\_G2][j\_G2] \* E1[i\_E][j\_E];

//printf("[%d][%d] = %d\n", i\_r, j\_r, Rez2[i\_r][j\_r]);

j\_r++;

}

j\_r = 0;

i\_r++;

}

}

flag\_first = 0;

tab = tab + n2;

}

int\*\* Box\_product = (int\*\*)malloc((n1 \* n2) \* sizeof(int\*)); //А1\*E2 + E1\*A2

for (int i = 0; i < (n1 \* n2); i++) {

Box\_product[i] = (int\*)malloc((n1 \* n2) \* sizeof(int));

}

printf("\nДекартово произведение М1 на М2:\n");

for (int i = 0; i < (n1 \* n2); i++) {

for (int j = 0; j < (n1 \* n2); j++) {

Box\_product[i][j] = Rez1[i][j] + Rez2[i][j];

printf(" %d", Box\_product[i][j]);

}

printf("\n");

}

return 0;

}