**Лабораторная работа 3**

**Ход работы:**

**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.

Листинг:

#include <iostream>

#include "time.h"

#include "windows.h"

#pragma warning (disable : 4996)

int\*\* dynamic(int SIZE);

void generate(int\*\* mass, int SIZE);

void print(int\*\* mass, int SIZE);

int otojd(int\*\* mass, int SIZE);

void circle\_sum(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int SIZE);

void dekart\_mul(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int count, int count\_dynamic);

void del(int\*\* mass, int SIZE);

int stag(int\*\* mass, int SIZE);

void unionn(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int SIZE);

void cross(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int SIZE);

void cross(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int SIZE) {

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

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

mass\_res[i][j] = mass\_A[i][j] & mass\_B[i][j];

}

}

}

void unionn(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int SIZE) {

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

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

mass\_res[i][j] = mass\_A[i][j] | mass\_B[i][j];

}

}

}

int stag(int\*\* mass, int SIZE) {

int point\_1, point\_2;

printf("\n Введите номера вершин, которые вы хотели бы стянуть (в порядке возрастания) : ");

scanf("%d%\*c%d", &point\_1, &point\_2);

if (point\_1 > point\_2) return SIZE;

point\_1--;

point\_2--;

if (mass[point\_1][point\_2] == 0) {

printf("\n Введенные вершины невозможно отождествить.\n");

return SIZE;

}

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

if (mass[point\_1][i] == 1 || mass[point\_2][i] == 1) {

mass[point\_1][i] = 1;

mass[i][point\_1] = 1;

}

}

for (int i = point\_2; i < SIZE - 1; i++) {

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

mass[i][j] = mass[i + 1][j];

}

}

for (int i = point\_2; i < SIZE - 1; i++) {

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

mass[j][i] = mass[j][i + 1];

}

}

mass[point\_1][point\_1]=0;

printf(" (в результирующей матрице номера вершин начинная с %d-ой уменьшаются на единицу)\n", point\_2 + 2);

SIZE--;

return SIZE;

}

void del(int\*\* mass, int SIZE) {

bool g = true;

int point\_1;

printf("\n Введите номер вершины, которую вы хотете расщепить: ");

scanf("%d", &point\_1);

int arr[20];

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

arr[i] = mass[i][point\_1 - 1];

}

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

mass[SIZE - 1][i] = 0;

mass[i][SIZE - 1] = 0;

}

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

mass[point\_1-1][i] = 0;

mass[i][point\_1-1] = 0;

}

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

if (arr[i] != 0 && g == true) {

mass[point\_1 - 1][i] = 1;

mass[i][point\_1 - 1] = 1;

g = false;

}

else if (arr[i] != 0) {

mass[SIZE - 1][i] = 1;

mass[i][SIZE - 1] = 1;

g = true;

}

}

mass[SIZE-1][point\_1-1] = 1;

mass[point\_1 - 1][SIZE - 1] = 1;

mass[SIZE - 1][SIZE - 1] = 0;

}

int\*\* dynamic(int SIZE) {

int\*\* mass;

mass = (int\*\*)malloc(SIZE \* sizeof(int\*));

for (int i = 0; i < SIZE; i++) mass[i] = (int\*)malloc(SIZE \* sizeof(int));

return mass;

}

void generate(int\*\* mass, int SIZE) {

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

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

if (j > i) {

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

mass[j][i] = mass[i][j];

}

else if (i == j) mass[i][j] = 0;

}

}

}

void print(int\*\* mass, int SIZE) {

printf(" ");

for (int k = 1; k <= SIZE; k++) printf(" %d", k);

printf("\n -----------\n");

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

printf("%d |", i + 1);

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

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

}

printf("\n");

}

}

int otojd(int\*\* mass, int SIZE) {

int point\_1, point\_2;

printf("\n Введите номера вершин, которые вы хотели бы отождествить (в порядке возрастания) : ");

scanf("%d%\*c%d", &point\_1, &point\_2);

if (point\_1 > point\_2) return SIZE;

point\_1--;

point\_2--;

if (mass[point\_1][point\_2] == 0) {

printf("\n Введенные вершины невозможно отождествить.\n");

return SIZE;

}

//mass[point\_1][point\_2] = 0;

//mass[point\_2][point\_1] = 0;

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

if (mass[point\_1][i] == 1 || mass[point\_2][i] == 1) {

mass[point\_1][i] = 1;

mass[i][point\_1] = 1;

}

}

for (int i = point\_2; i < SIZE - 1; i++) {

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

mass[i][j] = mass[i + 1][j];

}

}

for (int i = point\_2; i < SIZE - 1; i++) {

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

mass[j][i] = mass[j][i + 1];

}

}

printf(" (в результирующей матрице номера вершин начинная с %d-ой уменьшаются на единицу)\n", point\_2 + 2);

SIZE--;

return SIZE;

}

void circle\_sum(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int SIZE) {

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

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

mass\_res[i][j] = mass\_A[i][j] ^ mass\_B[i][j];

}

}

}

void dekart\_mul(int\*\* mass\_A, int\*\* mass\_B, int\*\* mass\_res, int count, int count\_dynamic) {

int poryadok\_low, poryadok\_high;

int mas\_ind1 = 0, mas\_ind2 = 0;

for (int i = 0; i < count \* count\_dynamic; i++) {

for (int j = 0; j < count \* count\_dynamic; j++) {

mass\_res[i][j] = 0;

}

}

if (count > count\_dynamic) {

poryadok\_low = count\_dynamic;

poryadok\_high = count;

}

else {

poryadok\_low = count;

poryadok\_high = count\_dynamic;

}

for (int i = 0; i < poryadok\_high; i++) {

for (int j = 0; j < poryadok\_low; j++) {

mas\_ind2 = 0;

for (int k = 0; k < poryadok\_high; k++) {

for (int l = 0; l < poryadok\_low; l++) {

//if (k == i && l == j) mass\_res[mas\_ind1][mas\_ind2] = 0;

//else

{

if (l == j && mass\_A[k][i] == 1) {

mass\_res[mas\_ind1][mas\_ind2] = 1;

}

if (k == i && mass\_B[l][j] == 1) {

mass\_res[mas\_ind1][mas\_ind2] = 1;

}

//if (k != i && l != j) mass\_res[mas\_ind1][mas\_ind2] = 0;

}

mas\_ind2++;

}

}

mas\_ind1++;

}

}

}

int main()

{

int count, count\_dynamic, count\_m1, countd=6;

int\*\* mass1;

int\*\* mass2;

int\*\* mass3;

int\*\* masss;

int\*\* mass\_res;

int\*\* mass\_res\_mul;

int\*\* mass4;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

printf("Введите размер массива");

scanf("%d", &count);

mass1 = dynamic(count);

mass2 = dynamic(count);

masss = dynamic(count);

mass\_res = dynamic(count);

generate(mass1, count);

generate(mass2, count);

printf(" Массив M1:\n\n");

print(mass1, count);

printf("\n Массив M2:\n\n");

print(mass2, count);

mass4 = dynamic(count+1);

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

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

mass4[i][j] = mass1[i][j];

}

}

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

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

masss[i][j] = mass1[i][j];

}

}

//A

count\_m1 = otojd(mass1, count);

printf("\n Массив M1 после операции отождествления:\n\n");

print(mass1, count\_m1);

//A

//B

count\_m1 = stag(masss, count);

printf("\n Результат стягивания вершин:\n\n");

print(masss, count\_m1);

//B

//C

del(mass4, count+1);

print(mass4, count + 1);

printf("\n Результат расщепления вершины:\n\n");

//rashep

//C

//A2

printf("\n Массив после операции обЪединения M1 и M2:\n\n");

unionn(mass1, mass2, mass\_res, count);

print(mass\_res, count);

//A2

//B2

printf("\n Массив после операции пересечения M1 и M2:\n\n");

cross(mass1, mass2, mass\_res, count);

print(mass\_res, count);

//B2

//C3

printf("\n Массив после операции кольццевой суммы M1 и M2:\n\n");

circle\_sum(mass1, mass2, mass\_res, count);

print(mass\_res, count);

//C3

//dekart\_mul

printf("\n Введите порядок массива M3: ");

scanf("%d", &count\_dynamic);

mass3 = dynamic(count\_dynamic);

generate(mass3, count\_dynamic);

printf("\n Массив M3:\n\n");

print(mass3, count\_dynamic);

mass\_res\_mul = dynamic(count \* count\_dynamic);

dekart\_mul(mass2, mass3, mass\_res\_mul, count, count\_dynamic);

printf("\n Результат декартового произведения массивов M2 и M3:\n\n");

for (int i = 0; i < count \* count\_dynamic; i++) {

for (int j = 0; j < count \* count\_dynamic; j++) {

printf("%d ", mass\_res\_mul[i][j]);

}

printf("\n");

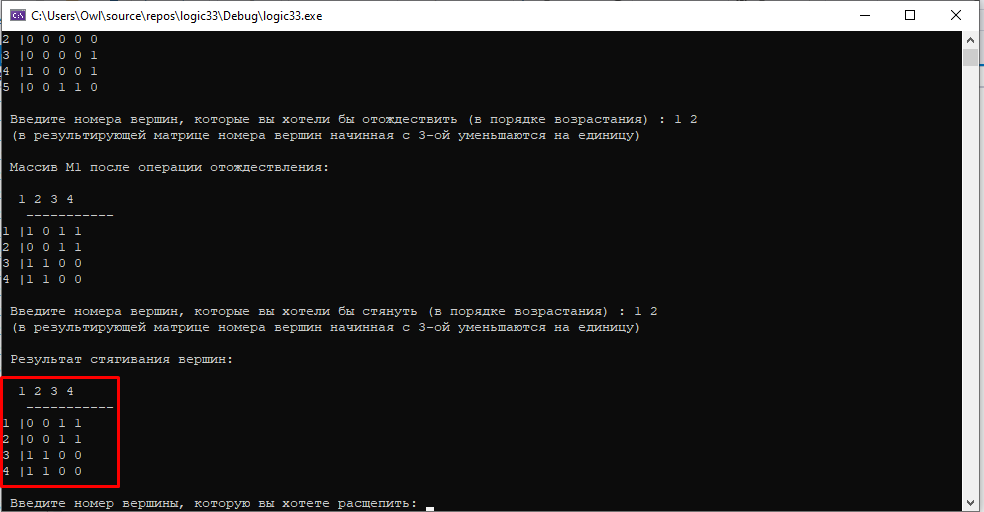
}

//dekart\_mul

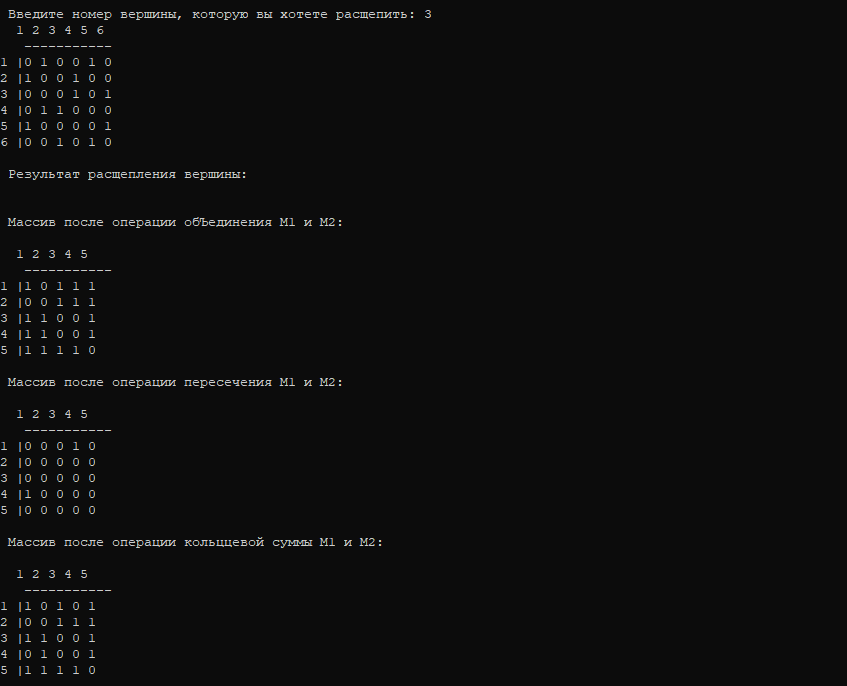
}



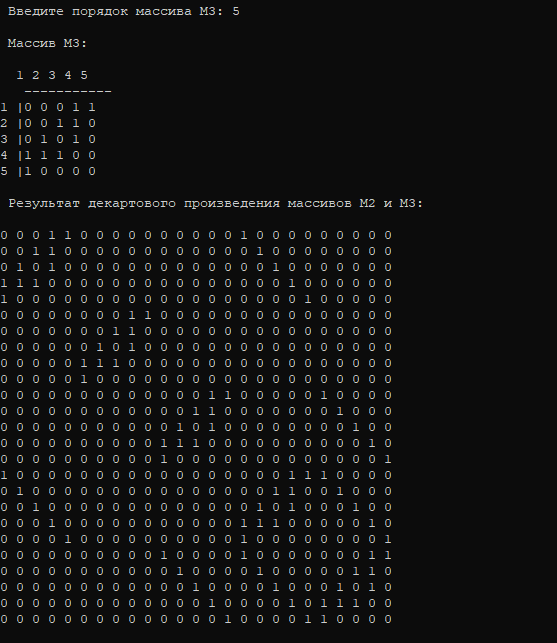
**Рис 1. – Результат отождествления.**



**Рис 2. – Результат стягивания вершин**



**Рис 3. – Результат расщепления, объединения, пересечение, кольцевой суммы**



**Рис 4. – Результат Декартового произведения.**