Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Выполнил:**

Студенты группы 19ВВ2:

Юдин Д.А.

**Приняли:**

Митрохин М.А.

Юрова О.В.

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Пенза 2020

### Название:

### Унарные и бинарные операции над графами

### 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.

### Цель работы:

Разработать программы по данным заданиям.

**Общие сведения:**

Все унарные операции над графами можно объединить в две группы. Первую группу составляют операции, с помощью которых из исходного графа *G*1*,*можно построить граф *G*2 с меньшим числом элементов. В группу входят операции удаления ребра или вершины, отождествления вершин, стягивание ребра. Вторую группу составляют операции, позволяющие строить графы с большим числом элементов. В группу входят операции расщепления вершин, добавления ребра.

*Отождествление вершин.* В графе *G*1 выделяются вершины *и,v.* Определяют окружение *Q*1 вершины *u*,и окружение *Q*2 вершины *v,* вычисляют их объединение *Q* = *Q1Q2.* Затем над графом *G*1 выполняются следующие преобразования:

* из графа *G*1 удаляют вершины *u,v (H*1 *= G*1 *- u - v);*
* к графу *Н*1присоединяют новую вершину *z (H*1 *= H*1 *+z);*
* вершину *z*соединяют ребром с каждой из вершин *w*1*Q*

*(G*2 *= H*1 *+ zwi*, *i =* 1,2,3*,…).*

*Стягивание ребра.* Данная операция является операцией отождествления смежных вершин *и, v* в графе *G*1.

Наиболее важными бинарными операциями являются: объединение, пересечение, декартово произведение и кольцевая сумма.

*Объединение.* Граф *G* называется объединением или наложением графов *G*1 и *G*2, если *VG = V*1*V*2*; UG = U*1*U*2 (рис. 1).

**U**

*v*1

*v*2

*v*3

*v*4

*v*3

*v*4

*v*5

*v*2

*v*1

*v*3

*v*4

*v*5

Рис. 1. Объединение графов *G*1, *G*2

Объединение графов *G*1 и *G*2 называется дизъюнктным, если *V*1*V*2 *= *. При дизъюнктном объединении никакие два из объединяемых графов не должны иметь общих вершин.

*Пересечение.* Граф *G* называется пересечением графов *G*1, *G*2,если *VG = V*1*V*2и*UG = U*1*U*2 (риc.2). Операция "пересечения" записывается следующим образом: *G = G*1*G*2*.*

**∩**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*6

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

*v*3

*v*2

Рис.2. Пересечение графов *G*1,*G*2*.*

*Декартово произведение.* Граф *G* называется декартовым произведением графов *G*1 и *G*2 если *VG* = *V*1*V*2 —декартово произведение множеств вершин графов *G*1, *G*2, а множество ребер *U*c задается следующим образом: вершины (*zi*, *vk*) и (*zj*, *vl*) смежны в графе *G* тогда и только тогда, когда *zi* = *zj*(*i* = *j*), a*v*k и *vl*смежны в *G*2 или *vk* = *vl*(*k* = *l*), смежны в графе *G*1 (см. рис.3).

**X**

*z*1

*z*2

*v*1

*v*3

*v*2

*z*1*v*1

*z*1*v*2

*z*1*v*3

*z*2*v*1

*z*2*v*2

*z*2*v*3

Рис. 3. Декартово произведение графов *G*1,*G*2

*Кольцевая сумма* графов представляет граф, который не имеет изолированных вершин и состоит из ребер, присутствующих либо в первом исходном графе, либо во втором. Кольцевая сумма определяется следующим соотношением: *G* = *G*1*G*2 (рис.4).

**⊕**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

*v*3

*v*2

Рис.4. Кольцевая сумма графов *G1, G2*

### Описание метода решения задачи:

**Задача 1:**

### Листинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <locale.h>

int\*\* mas1, \*\* mas2,\*\* m\_longed;

int N;

typedef struct Node {

char vertex;

struct Node\* next;

} Node;

void add\_Node\_to\_beginning(Node\*\* head, int data) //функция добавления узла в начало

{

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = (\*head); //присваиваем указателю tmp адрес след. узла

(\*head) = tmp\_node; //Присваиваем указателю head адрес tmp

} //после выхода из функции tmp уничтожается

int delete\_node(Node\*\* head) { //удаляем узел на который указывает адрес

Node\* prev = NULL;

int val;

prev = (\*head); //передаем во временную переменную prev адрес

val = prev->vertex;

(\*head) = (\*head)->next; //в head кладем адрес следующего узла

free(prev); //очищаем временную переменную prev

return val;

}

Node\* findLastHead(Node\* head) { //поиск адреса последнего элемента

if (head == NULL) {

return NULL;

}

while (head->next) {

head = head->next;

}

return head;

}

void add\_Node\_to\_end(Node\* head, int data) { //функция добавления нового узла в конец

Node\* last = findLastHead(head); //получаем указатель на последний элемент списка

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = NULL;

last->next = tmp\_node; //записываем в последний элемент списка указатель на новый узел

}

void createLinkedList(int\*\* mas, Node\*\* head, int N) //создаем связный список

{

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

{

add\_Node\_to\_beginning(&head[i], i); //передаем адрес вершины и ее номер

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

{

if (mas[i][j] == 1)

{

add\_Node\_to\_end(head[i], j); //добавляем к вершине связные с ней вершины

}

}

}

}

void cleanlist(Node\*\* head, int n)

{

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

head[i] = NULL;

}

void cleanlistfield(Node\*\* head, int del, int n)

{

Node\* tmp;

int back;

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

{

tmp = head[i];

while (tmp)

{

if (tmp->next)

if (tmp->next->vertex == del)

{

tmp->next = tmp->next->next;

}

if (tmp->vertex >= del)

{

back = tmp->vertex;

back--;

tmp->vertex = back;

}

tmp = tmp->next;

}

}

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

{

head[i] = head[i + 1];

}

head[n - 1] = NULL;

}

void paste(Node\* head, int m)

{

Node\* tmp;

tmp = (Node\*)malloc(sizeof(Node));

tmp->vertex = m;

tmp->next = head->next;

head->next = tmp;

}

void addver(Node\*\* head, int field, int n)

{

Node\* ptr, \* tmp;

int base = head[field]->vertex, search;

tmp = head[field];

tmp = tmp->next;

while (tmp)

{

search = tmp->vertex;

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

{

if (head[i]->vertex == search)

{

if (!head[i]->next)

{

add\_Node\_to\_beginning(&head[i], base);

}

ptr = head[i];

while (ptr->next)

{

if (ptr->next->vertex == base)

{

break;

}

if (ptr->next->vertex > base)

{

paste(ptr, base);

}

}

break;

}

}

tmp = tmp->next;

}

}

void printLinkedList(Node\* head) {

printf("\n");

while (head) {

printf("%d -->", head->vertex + 1);

head = head->next;

}

printf("\n");

}

void identificationlist(int v1, int v2, Node\*\* head, int n)

{

int del, field;

Node\* ptr = NULL, \* tmp;

if (v1 > v2)

{

del = v1 - 1;

field = v2 - 1;

}

else

{

del = v2 - 1;

field = v1 - 1;

}

tmp = head[del]->next;

ptr = (Node\*)malloc(sizeof(Node));

ptr->vertex = head[field]->vertex;

ptr->next = NULL;

head[field] = head[field]->next;

while (head[field] || tmp)

{

if (head[field] && tmp)

{

if (head[field]->vertex > tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

continue;

}

if (head[field]->vertex < tmp->vertex)

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

continue;

}

if (head[field]->vertex == tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

head[field] = head[field]->next;

continue;

}

}

if (head[field])

{

while (head[field])

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

}

}

else

{

while (tmp)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

}

}

}

head[field] = ptr;

fflush(stdin);

cleanlistfield(head, del, n);

addver(head, field, n);

}

void identificationlist\_2(int v1, int v2, Node\*\* head, int n)

{

int del, field;

Node\* ptr = NULL, \* tmp;

if (v1 > v2)

{

del = v1 - 1;

field = v2 - 1;

}

else

{

del = v2 - 1;

field = v1 - 1;

}

tmp = head[del]->next;

ptr = (Node\*)malloc(sizeof(Node));

ptr->vertex = head[field]->vertex;

ptr->next = NULL;

head[field] = head[field]->next;

while (head[field] || tmp)

{

if (head[field] && tmp)

{

if (head[field]->vertex > tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

continue;

}

if (head[field]->vertex < tmp->vertex)

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

continue;

}

if (head[field]->vertex == tmp->vertex)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

head[field] = head[field]->next;

continue;

}

}

if (head[field])

{

while (head[field])

{

add\_Node\_to\_end(ptr, head[field]->vertex);

head[field] = head[field]->next;

}

}

else

{

while (tmp)

{

add\_Node\_to\_end(ptr, tmp->vertex);

tmp = tmp->next;

}

}

}

head[field] = ptr;

fflush(stdin);

cleanlistfield(head, del, n);

}

void splitinglist(Node\*\* head, int v, int n)

{

add\_Node\_to\_beginning(&head[n-1], n-1);

Node\* ptr = head[v];

if (ptr->next)

{

if (ptr->vertex < ptr->next->vertex)

{

add\_Node\_to\_end(head[n-1], ptr->vertex);

add\_Node\_to\_end(head[n-1], ptr->next->vertex);

ptr->next = ptr->next->next;

}

else

{

add\_Node\_to\_end(head[n-1], ptr->next->vertex);

add\_Node\_to\_end(head[n-1], ptr->vertex);

ptr->next = ptr->next->next;

}

}

else

{

add\_Node\_to\_end(head[n], ptr->vertex);

}

}

void main() {

setlocale(LC\_ALL, "russian");

printf("\n Укажите размер матрицы N\*N: ");

scanf("%d", &N);

Node\*\* head = (Node\*\*)malloc(N \* sizeof(Node\*));

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

head[i] = NULL;

mas1 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

mas1[i] = (int\*)malloc(N \* sizeof(int));

}

mas2 = (int\*\*)malloc(N \* sizeof(int\*));

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

{

mas2[i] = (int\*)malloc(N \* sizeof(int));

}

srand(time(NULL));

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

{

mas1[i][i] = 0;

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

{

mas1[i][j] = rand() % 2;

mas1[j][i] = mas1[i][j];

}

}

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

{

mas2[i][i] = 0;

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

{

mas2[i][j] = rand() % 2;

mas2[j][i] = mas2[i][j];

}

}

printf("\n 1. Вывод сгенерированных матриц смежности неореентированных графов \n");

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

printf("\n ");

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

printf("%d", mas1[i][j]);

}

}

printf("\n ");

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

printf("\n ");

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

printf("%d", mas2[i][j]);

}

}

createLinkedList(mas1, head, N);

printf("\n\nСписок смежности графа А:\n");

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

printLinkedList(head[i]);

cleanlist(head, N);

printf("\n\n");

createLinkedList(mas2, head, N);

printf("\n\nСписок смежности графа B:\n");

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

printLinkedList(head[i]);

cleanlist(head, N);

printf("\n\n");

m\_longed = (int\*\*)malloc((N + 1) \* sizeof(int\*)); printf("\t");

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

{

m\_longed[i] = (int\*)malloc((N + 1) \* sizeof(int));

}

int v1, v2;

createLinkedList(mas2, head, N);

printf("\n\n 2. Какие вершины второй матрицы отождествить?\n");

printf(" ");

scanf("%d %d", &v1, &v2);

int del, not\_del;

int i1 = 0, j = 0, j1 = 1, sch = 0;

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

{

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

{

m\_longed[i][j] = mas2[i][j];

}

}

if (v1 > v2)

{

del = v1 - 1;

not\_del = v2 - 1;

}

else

{

del = v2 - 1;

not\_del = v1 - 1;

}

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

{

if (m\_longed[del][i] == 1)

m\_longed[not\_del][i] = 1;

if (m\_longed[i][del] == 1)

m\_longed[i][not\_del] = 1;

}

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

{

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

{

m\_longed[i][j] = m\_longed[i][j + 1];

}

}

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

{

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

{

m\_longed[i][j] = m\_longed[i + 1][j];

}

}

if (mas2[del][not\_del]){

m\_longed[not\_del][not\_del] = 1;

}

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

{

printf("\n ");

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

{

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

}

}

identificationlist(v1, v2, head, N);

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

printLinkedList(head[i]);

cleanlist(head, N);

printf("\n\n");

createLinkedList(mas2, head, N);

printf("\n\n 2. Какое ребро второй матрицы стянуть?\n");

printf(" ");

scanf("%d %d", &v1, &v2);

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

{

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

{

m\_longed[i][j] = mas2[i][j];

}

}

if (v1 > v2)

{

del = v1 - 1;

not\_del = v2 - 1;

}

else

{

del = v2 - 1;

not\_del = v1 - 1;

}

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

{

if (m\_longed[del][i] == 1)

m\_longed[not\_del][i] = 1;

if (m\_longed[i][del] == 1)

m\_longed[i][not\_del] = 1;

}

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

{

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

{

m\_longed[i][j] = m\_longed[i][j + 1];

}

}

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

{

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

{

m\_longed[i][j] = m\_longed[i + 1][j];

}

}

m\_longed[not\_del][not\_del] = 0;

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

{

printf("\n ");

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

{

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

}

}

int N2 = N + 1;

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

{

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

{

m\_longed[i][j] = mas2[i][j];

}

}

for (int i = 0; i < N2; i++)

{

for (int j = 0; j < N2; j++)

{

if (m\_longed[i][j] < 0 || m\_longed[i][j] > 1)

m\_longed[i][j] = 0;

}

}

identificationlist\_2(v1, v2, head, N);

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

printLinkedList(head[i]);

cleanlist(head, N);

printf("\n\n 2. Какую вершину второй матрицы расщепить?\n");

printf(" ");

scanf("%d", &v1);

v1 = v1 - 1;

int i = v1;

for (; i < N2; i++)

{

if (m\_longed[v1][i] == 1)

{

m\_longed[N2 - 1][i] = m\_longed[i][N2 - 1] = 1;

}

else

m\_longed[N2 - 1][i] = m\_longed[i][N2 - 1] = 0;

}

m\_longed[N2 - 1][v1] = m\_longed[v1][N2 - 1] = 1;

for (int i = 0; i < N2; i++)

{

printf("\n ");

for (int j = 0; j < N2; j++)

{

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

}

}

cleanlist(head, N+1);

createLinkedList(m\_longed, head, N+1);

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

printLinkedList(head[i]);

cleanlist(head, N);

printf("\n\n 3. Объединение графов 1 и 2\n");

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

{

printf("\n ");

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

{

m\_longed[i][j] = mas1[i][j] || mas2[i][j];

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

}

}

printf("\n\n 3. Пересечение графов 1 и 2\n");

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

{

printf("\n ");

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

{

m\_longed[i][j] = mas1[i][j] & mas2[i][j];

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

}

}

printf("\n\n 3. Кольцевая сумма графов 1 и 2\n");

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

{

printf("\n ");

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

{

m\_longed[i][j] = mas1[i][j] ^ mas2[i][j];

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

}

}

printf("\n\nДекартово произведение A и B");

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

{

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

{

printf("\n\n");

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

{

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

{

if (i == j) {

printf("%5d", mas2[m][k]);

}

if (m == k)

{

printf("%5d", mas1[i][j]);

}

if ((i != j) && (m != k))

{

printf("%5d", 0);

}

}

}

}

}

printf("\n\n");

\_getch();

}

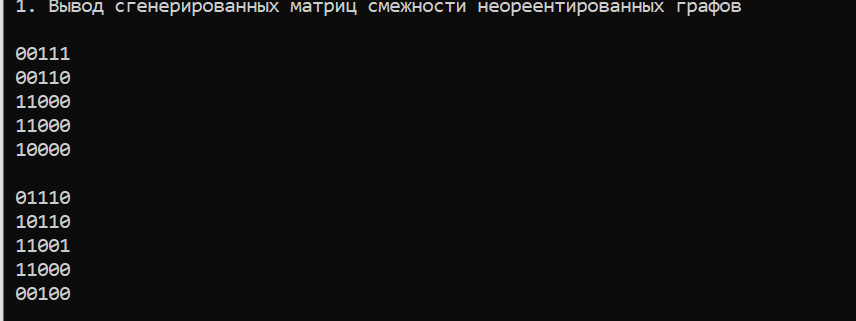
### Результаты работы программы:

### 

Рисунок 1 - Вывод матриц и списков смежности

### 

Рисунок 2 - Отождествление вершин (между вершинами создается связь, тк она была в исходной матрице)



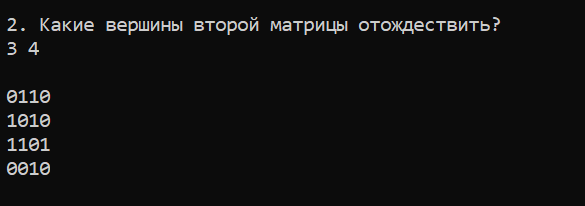


Рисунок 3 – пример матрицы и вершин, где не создается петля (тк в исходной матрице вершины не были смежны)

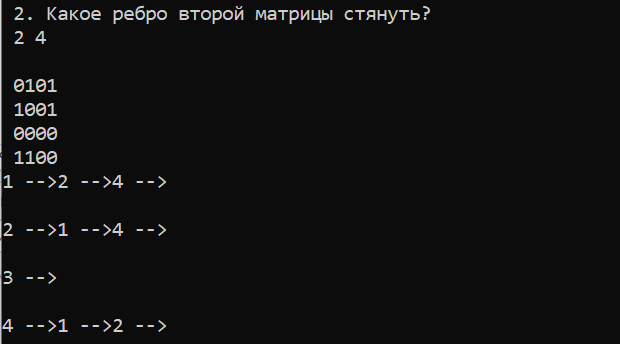


Рисунок 4 - Стягивание рёбра

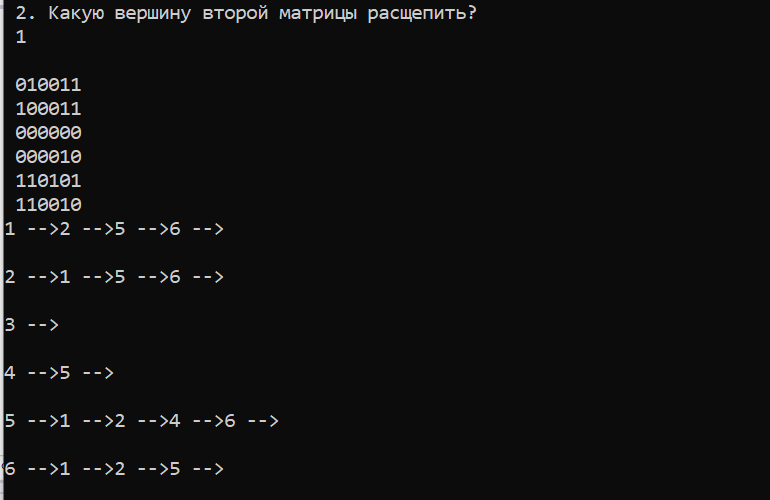


Рисунок 5 – расщепление вершины (связи старой вершины сохраняются)

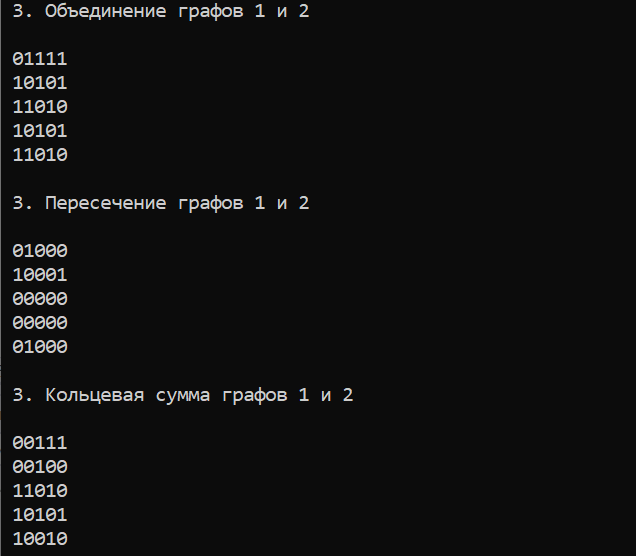


Рисунок 4 Объединение, пересечение и кольцевая сумма

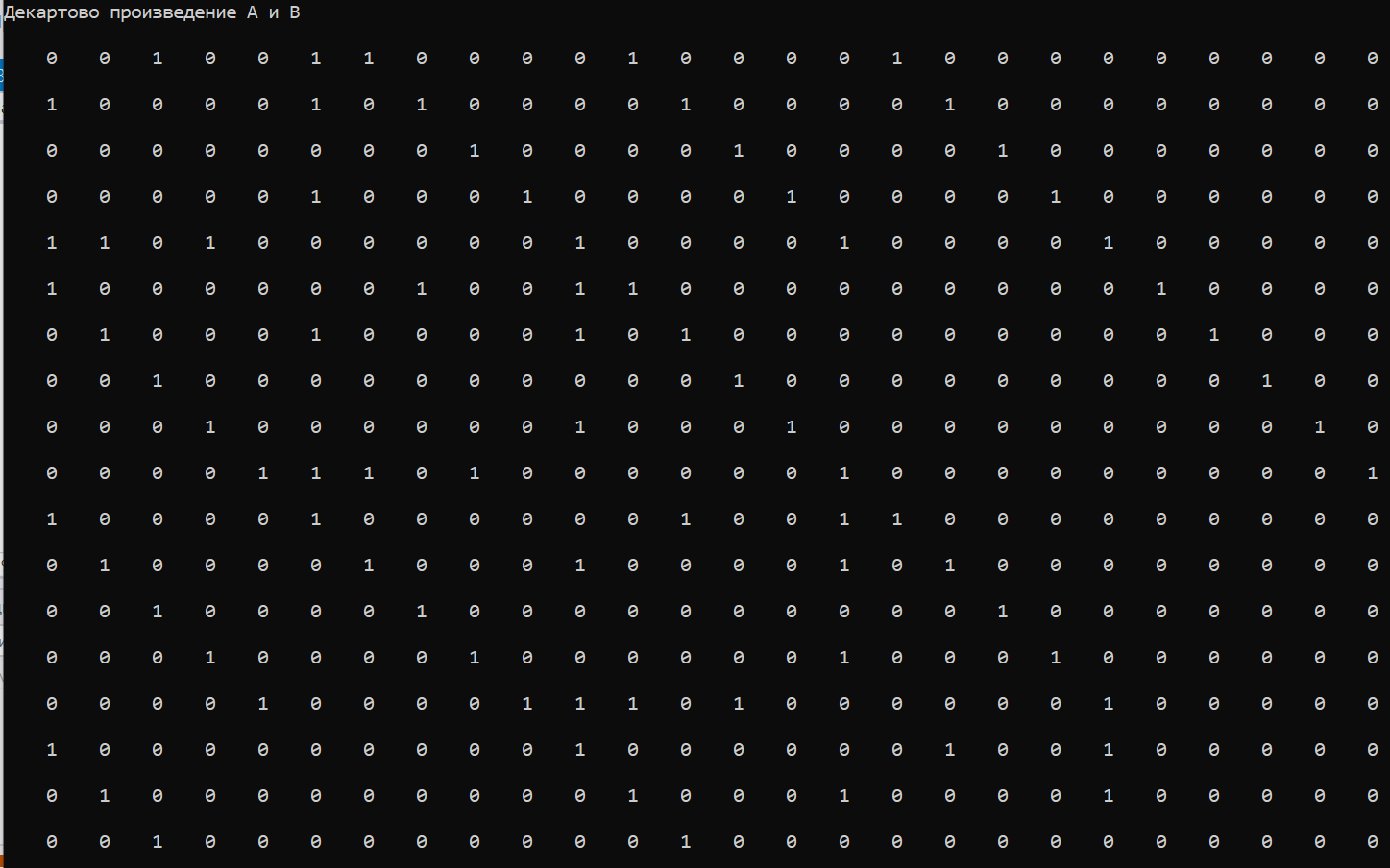


Рисунок 5 Декартово произведение

### Вывод:

В ходе выполнения лабораторной работы мы научились производить унарные операции над графами, заданными списками и матрицами смежности.