Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «**Унарные и бинарные операции над графами**»

Выполнили:

студенты группы 23ВВВ4

Брагин А.М.

Зарубин Я.

Герасимов К.

Приняли:

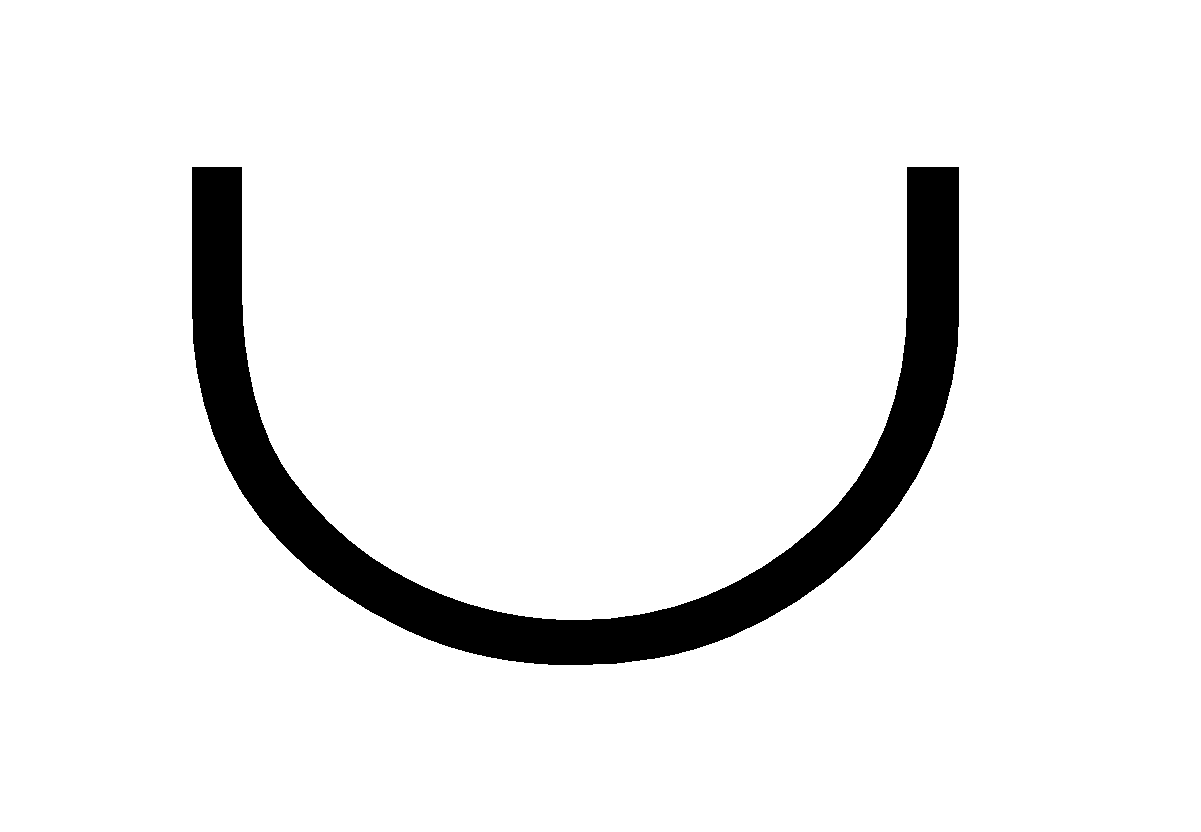
Деев М.В.

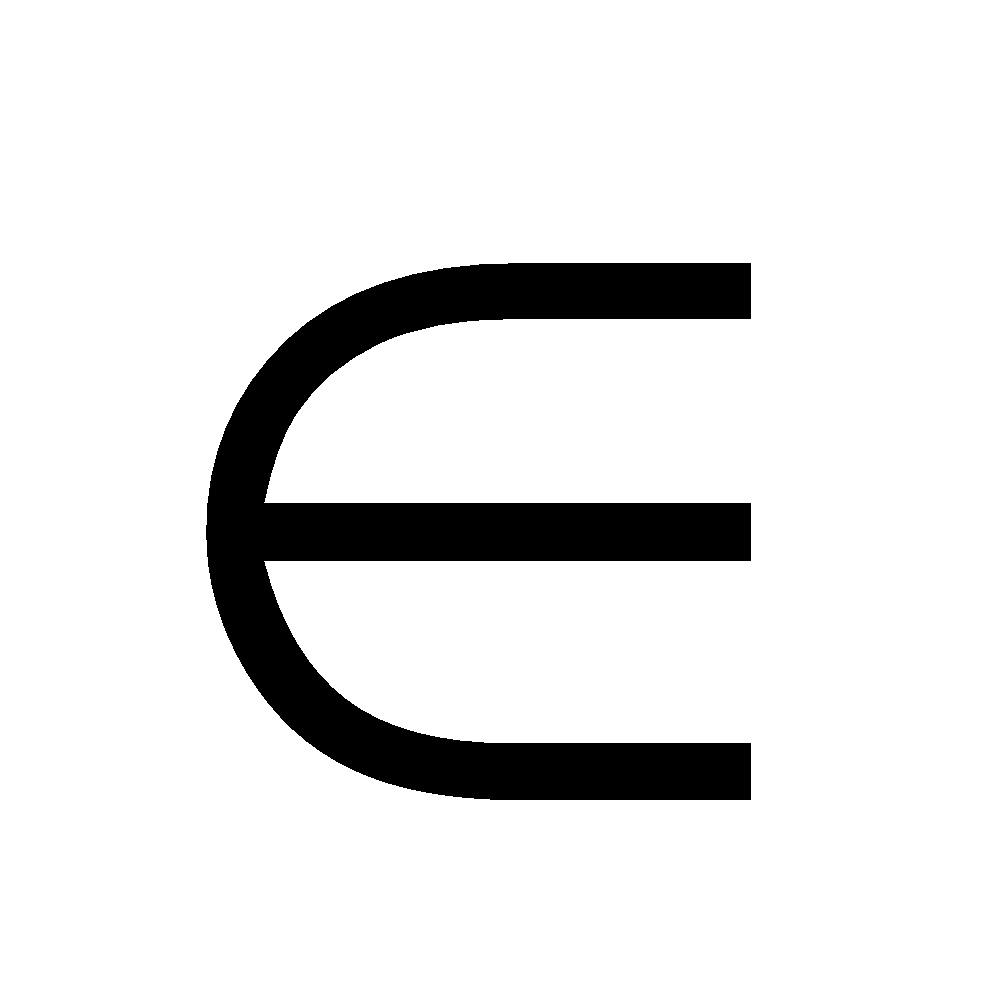
Юрова О.В.

Пенза 2024

**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* = *Q1*  *Q2.* 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*1 присоединяют 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.

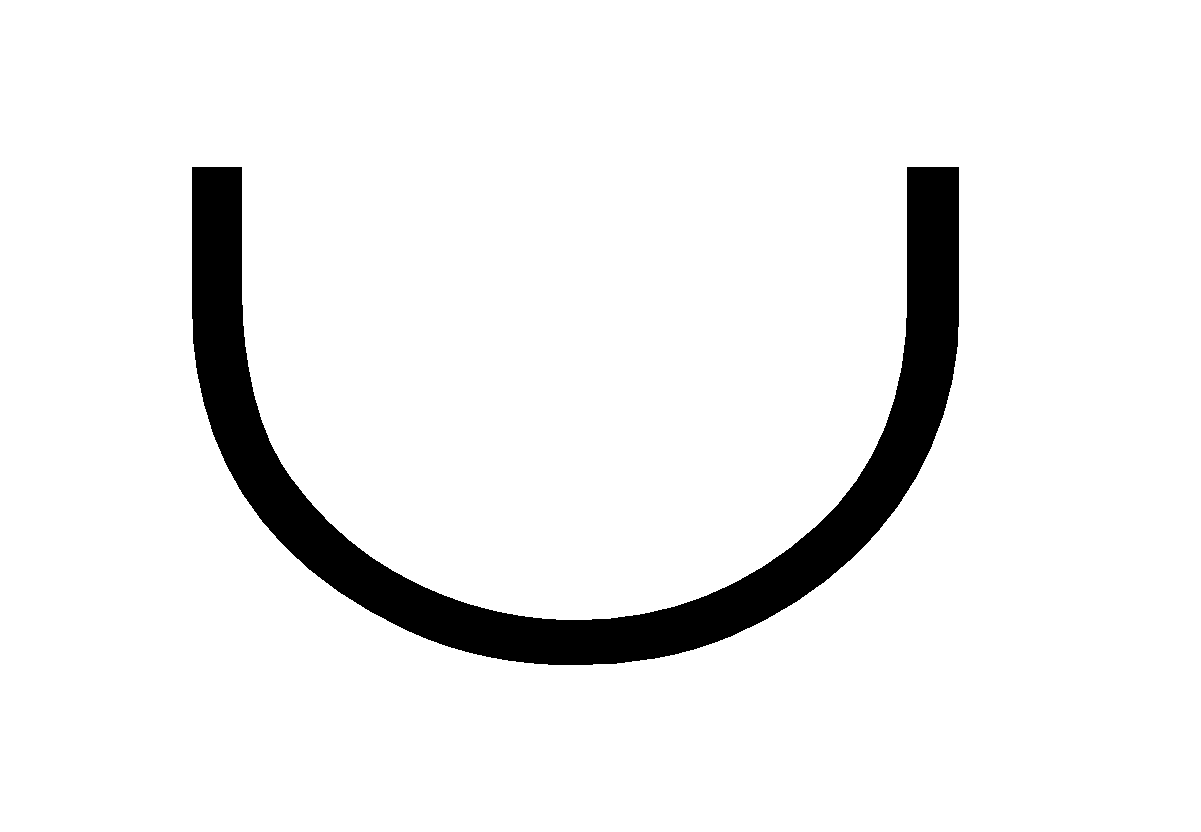
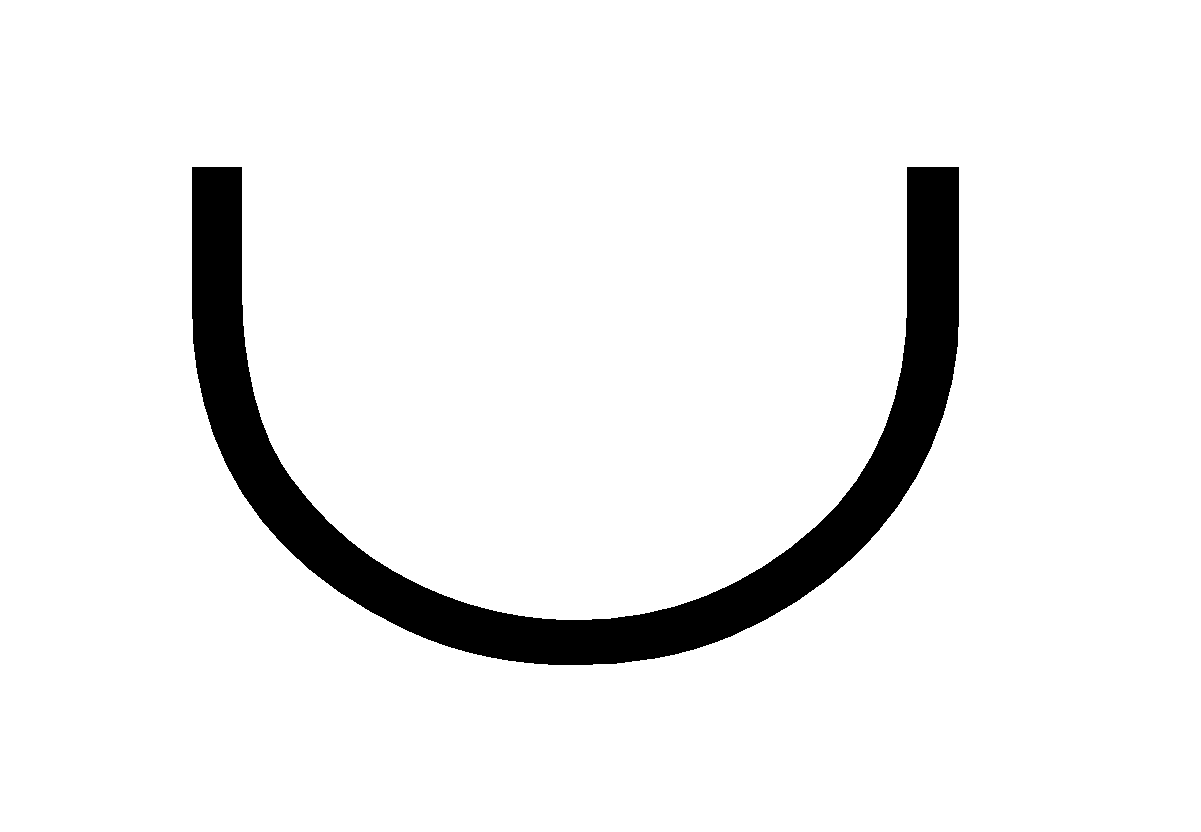
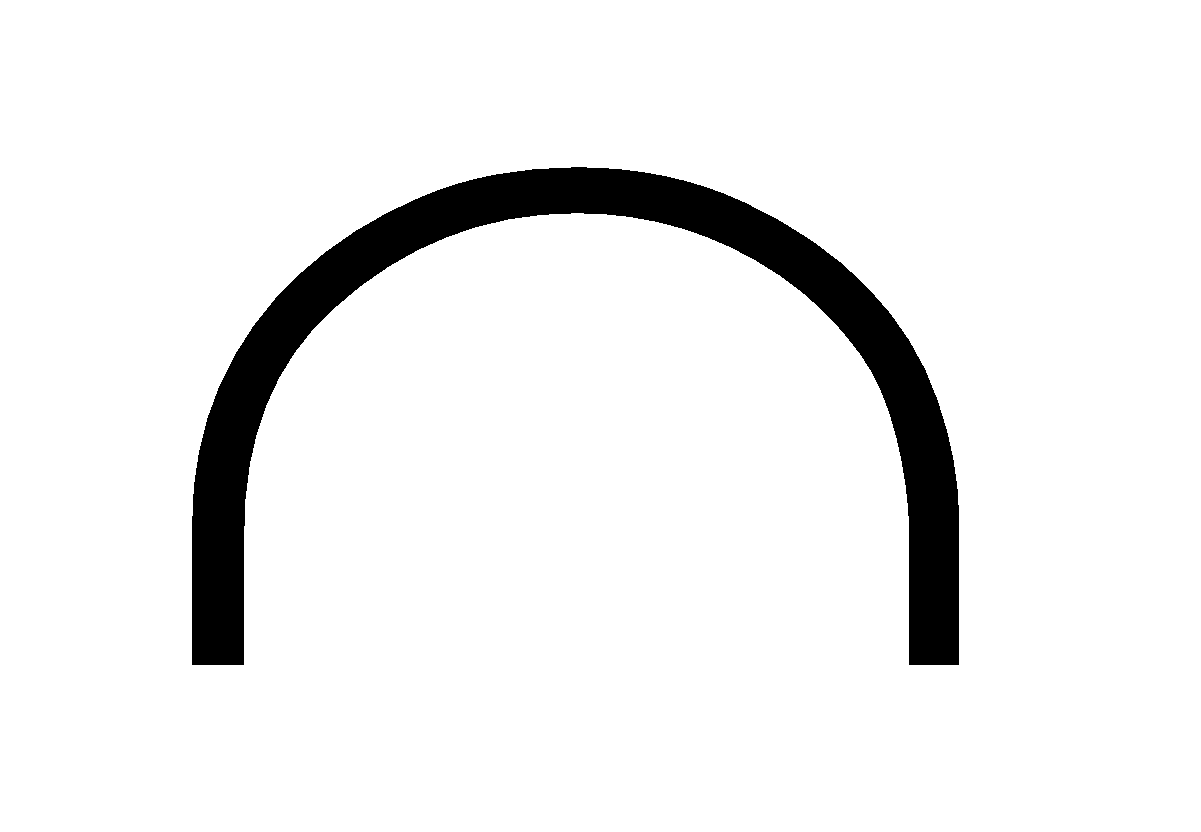
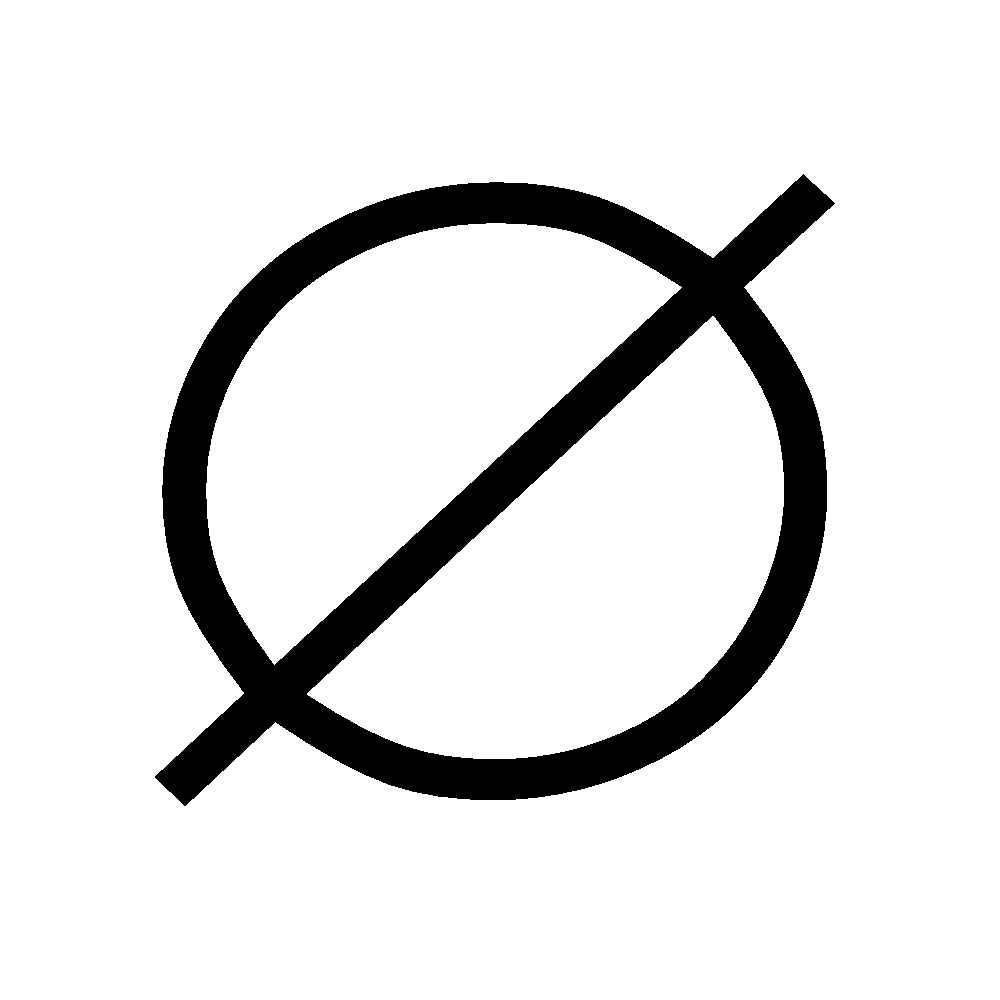
*Объединение.* Граф *G* называется объединением или наложением графов *G*1 и *G*2, если *VG* *= V*1*V*2*; UG* *= U*1 *U*2 (рис. 1).

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.

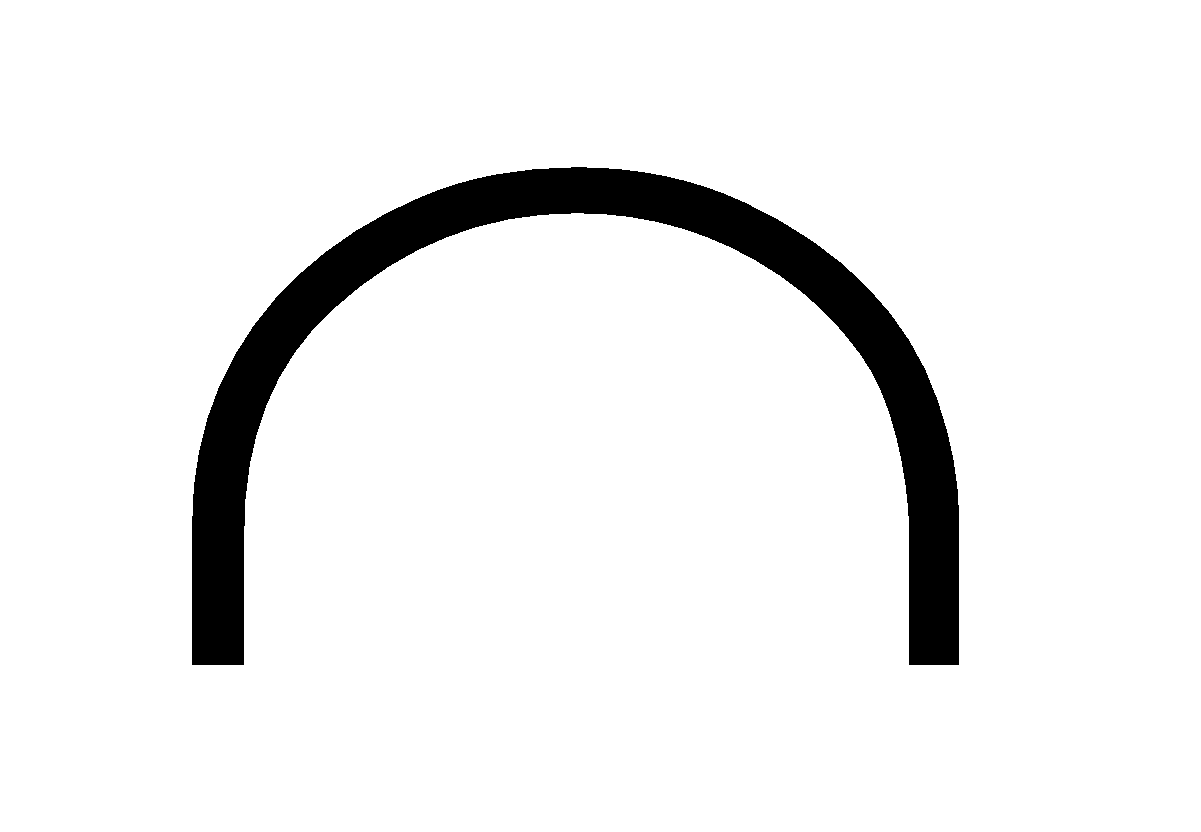
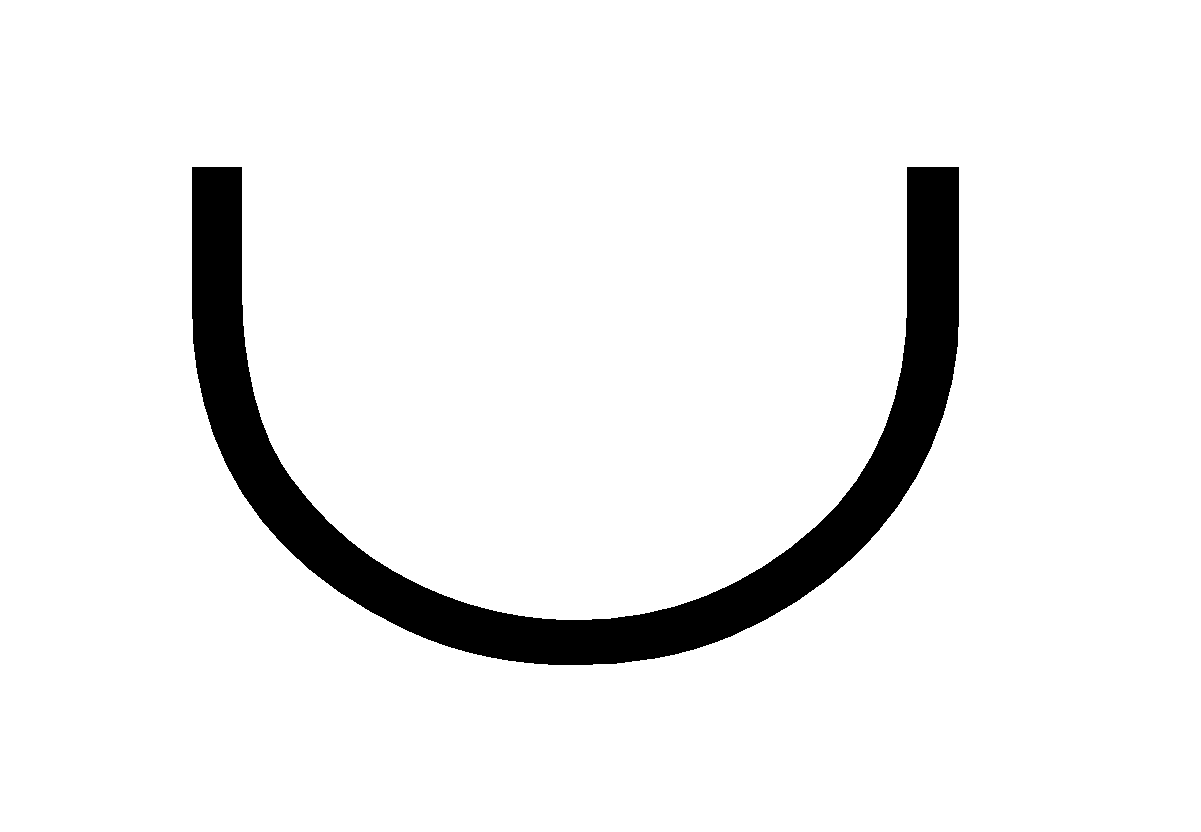
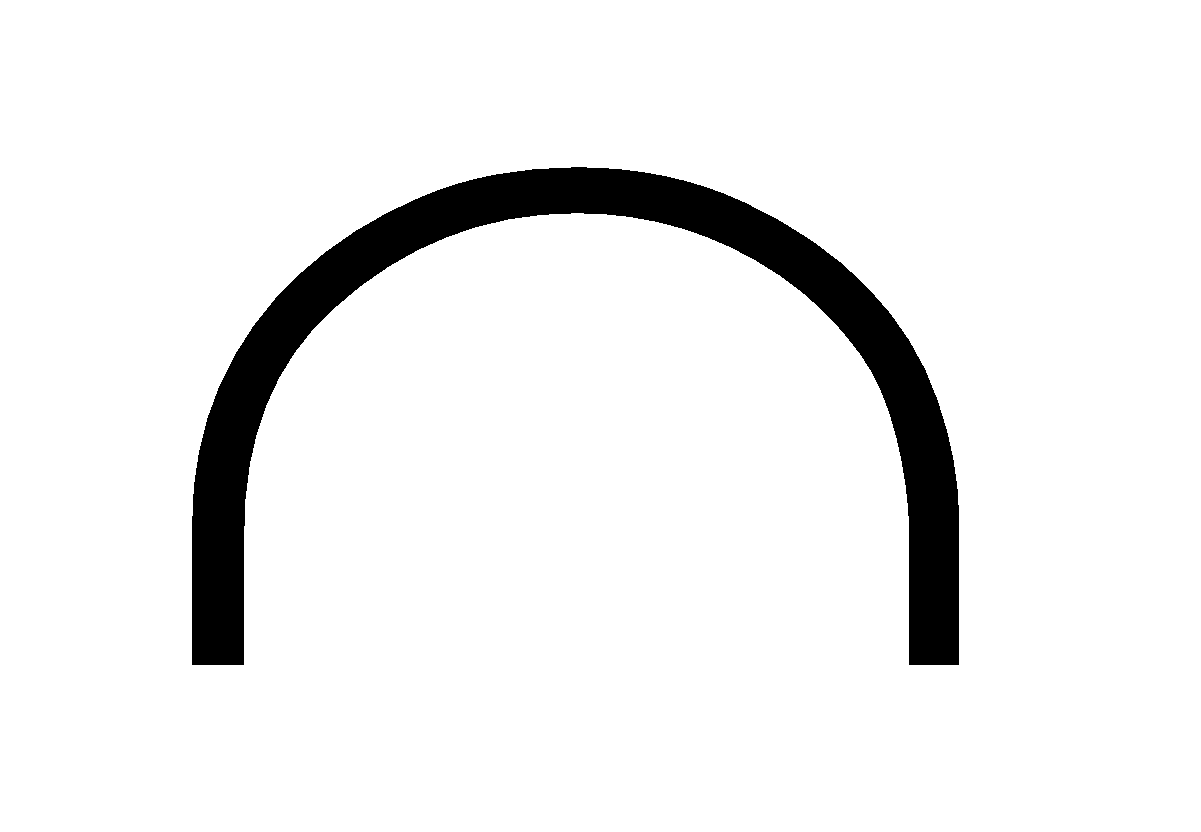
*Пересечение.* Граф *G* называется пересечением графов *G*1, *G*2, если *VG* *= V*1*V*2 и *UG* *= U*1*U*2 (риc.2). Операция "пересечения" записывается следующим образом: *G = G*1*G*2*.*

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

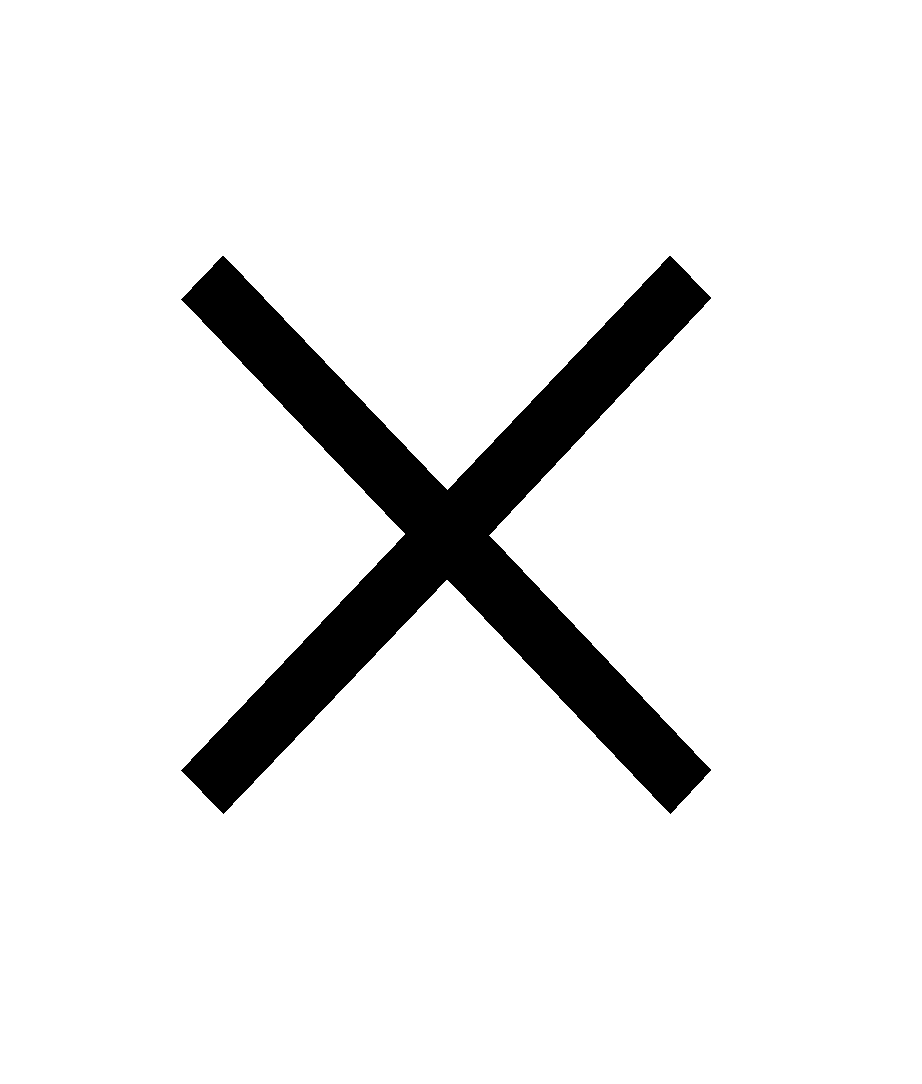
*Декартово произведение.* Граф *G* называется декартовым произведением графов *G*1 и *G*2 если *VG* = *V*1*V*2 —декартово произведение множеств вершин графов *G*1, *G*2, а множество ребер *U*c задается следующим образом: вершины (*zi*, *vk*) и (*zj*, *vl*) смежны в графе *G* тогда и только тогда, когда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

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

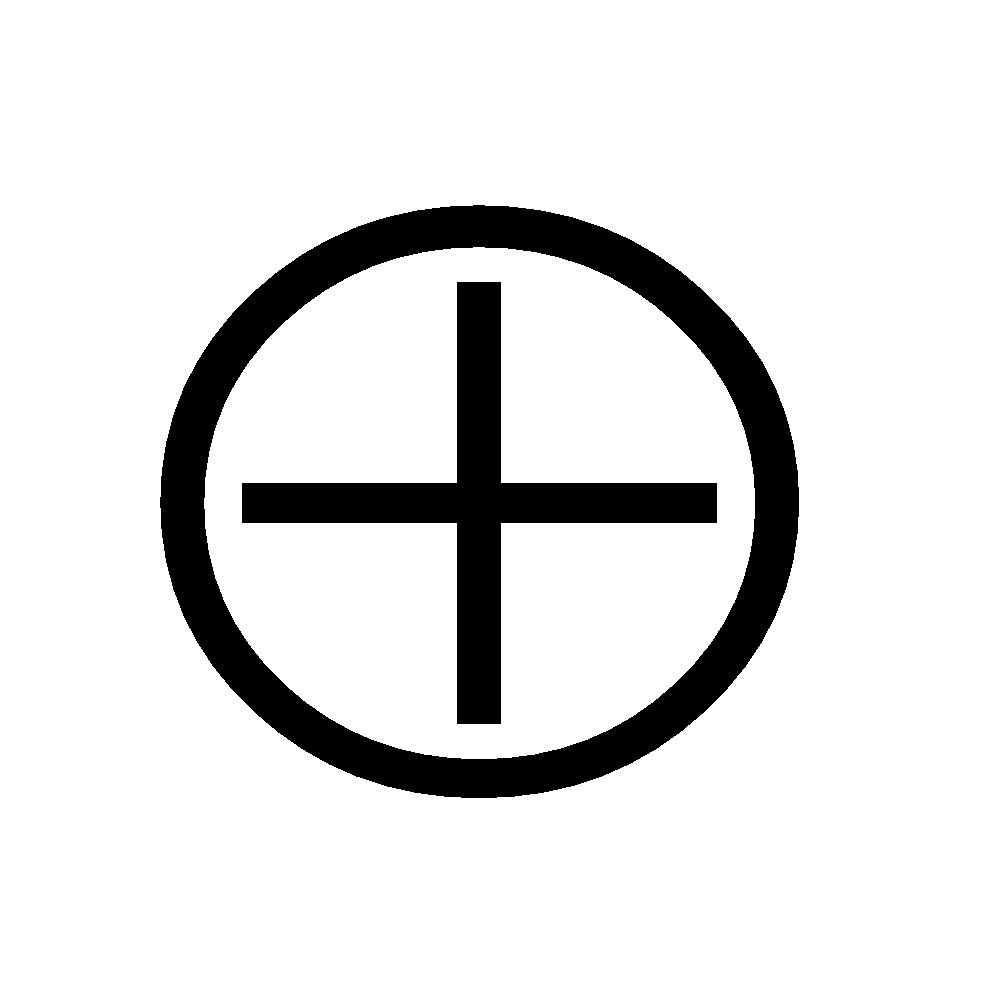
*Кольцевая сумма* графов представляет граф, который не имеет изолированных вершин и состоит из ребер, присутствующих либо в первом исходном графе, либо во втором. Кольцевая сумма определяется следующим соотношением: *G* = *G*1  *G*2 (рис.4).

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

**Практическая часть:**

**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

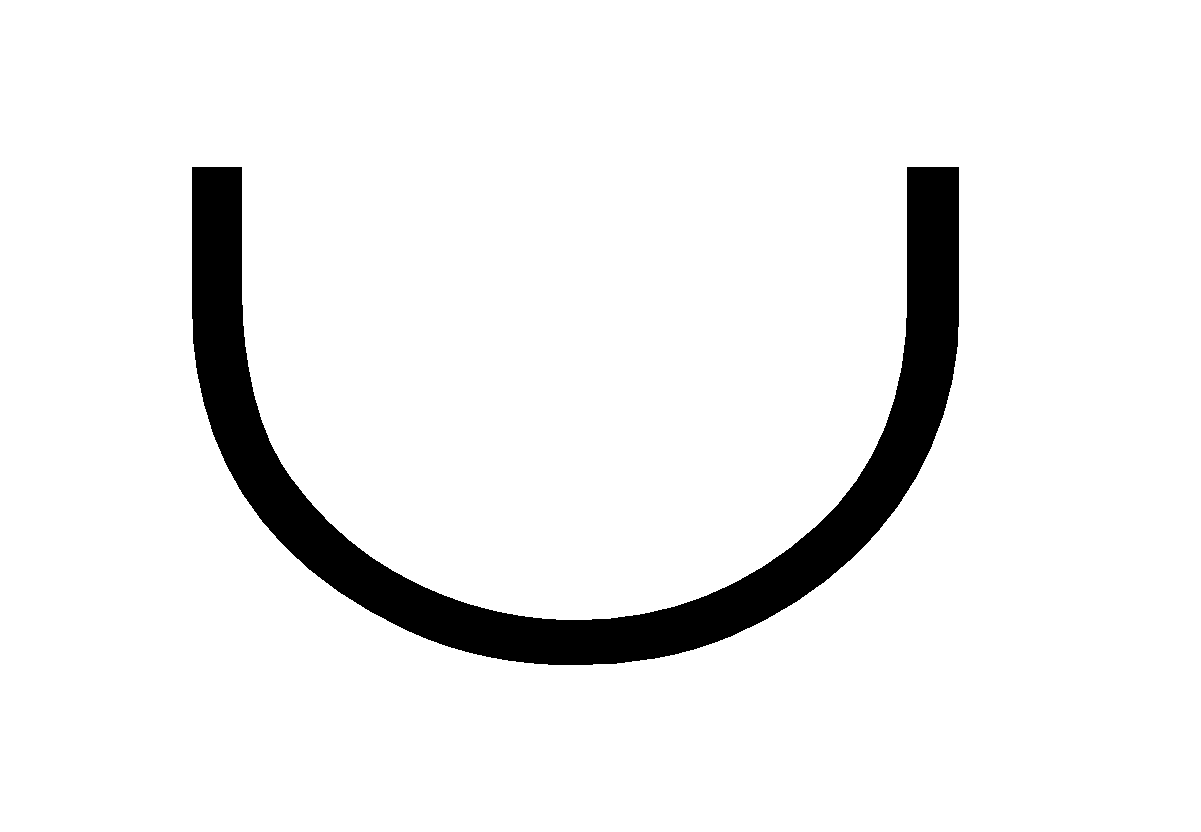
в) расщепления вершины

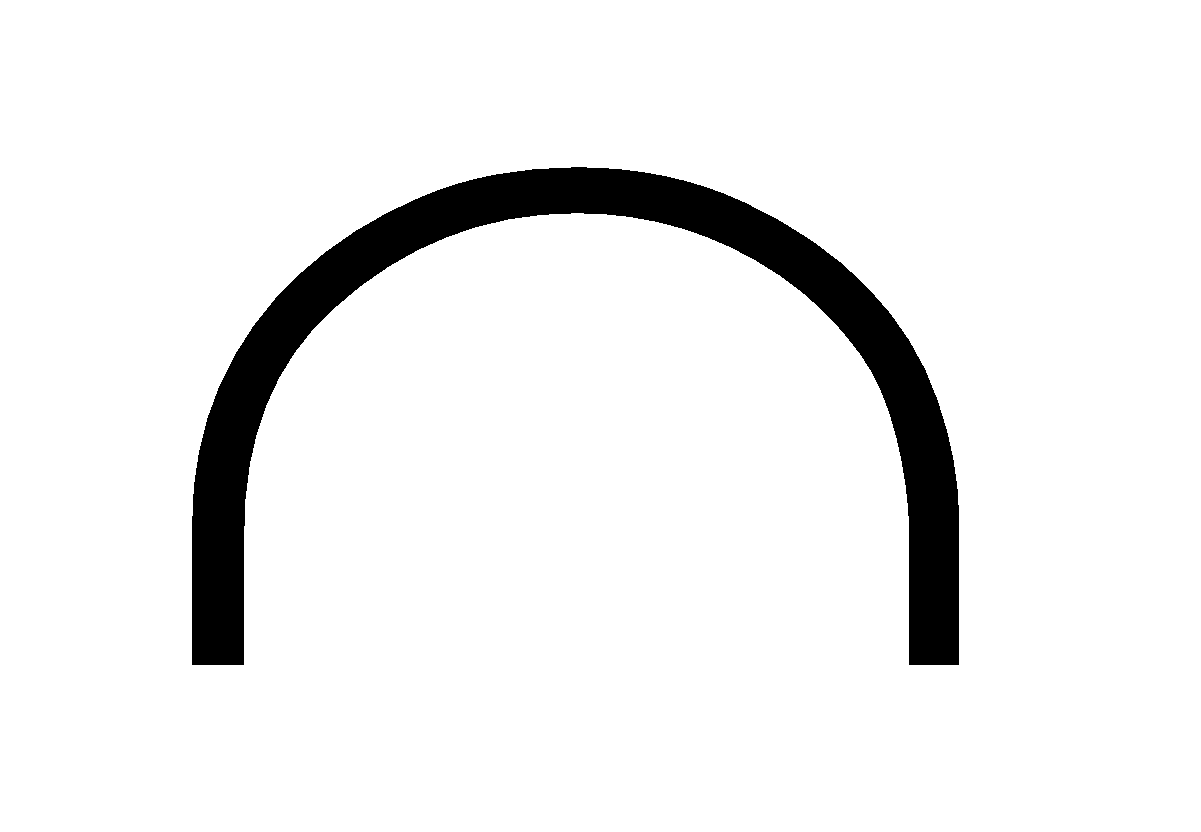
Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

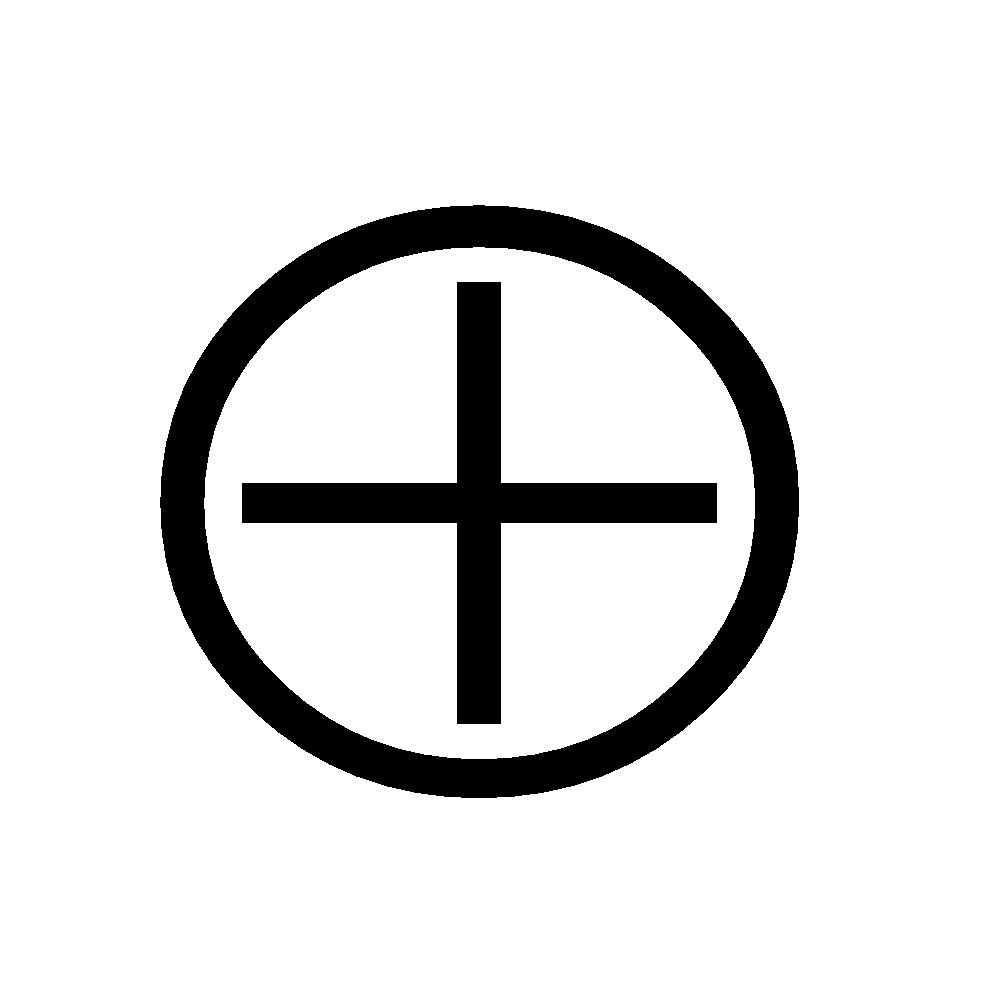
Результат выполнения операции выведите на экран.

**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.

Листинг программы

Задание 1

[#include](https://vk.com/im?sel=297349005&st=%23include) <stdio.h>  
[#include](https://vk.com/im?sel=297349005&st=%23include) <stdlib.h>  
[#include](https://vk.com/im?sel=297349005&st=%23include) <locale.h>  
[#include](https://vk.com/im?sel=297349005&st=%23include) <time.h>  
  
// Функция для генерации случайного графа  
int\*\* generG(int size) {  
int\*\* G = NULL;  
G = (int\*\*)malloc(size \* sizeof(int\*));  
for (int i = 0; i < size; i++) {  
G[i] = (int\*)malloc(size \* sizeof(int));  
}  
for (int i = 0; i < size; i++) {  
for (int j = 0; j < size;j++) {  
G[i][j] = 0;  
}  
}  
for (int i = 0; i < size; i++) {  
for (int j = i + 1; j < size;j++) {  
int edge = rand() % 2;  
G[i][j] = edge;  
G[j][i] = edge;  
}  
}  
return G;  
}  
  
// Функция для вывода матрицы смежности  
void printG(int\*\* G, int size) {  
printf("Матрица смежности:\n");  
for (int i = 0; i < size; i++) {  
for (int j = 0; j < size; j++) {  
printf("%d ", G[i][j]);  
}  
printf("\n");  
}  
}  
  
// Функция для удаления вершины  
int\*\* delv(int\*\* G, int size, int v) {  
int\*\* Gtemp = (int\*\*)malloc((size - 1) \* sizeof(int\*));  
for (int i = 0; i < size - 1; i++) {  
Gtemp[i] = (int\*)malloc((size - 1) \* sizeof(int));  
}  
  
for (int i = 0; i < size - 1; i++) {  
for (int j = 0; j < size - 1; j++) {  
Gtemp[i][j] = (i < v ? G[i][j] : G[i + 1][j]);  
Gtemp[i][j] = (j < v ? Gtemp[i][j] : (i < v ? G[i][j + 1] : G[i + 1][j + 1]));  
}  
}  
for (int i = 0; i < size; i++) {  
free(G[i]);  
}  
free(G);  
return Gtemp;  
}  
  
// Операция отождествления вершин  
int\*\* unionv(int\*\* G, int size, int v1, int v2) {  
if (v1 < 0 || v1 >= size || v2 < 0 || v2 >= size || v1 == v2) {  
printf("Ошибка: некорректные вершины для отождествления.\n");  
return G;  
}  
for (int i = 0; i < size; i++) {  
if (G[v2][i]) {  
G[v1][i] = 1;  
G[i][v1] = 1;  
}  
}  
return delv(G, size, v2);  
}  
  
// Операция стягивания ребра  
int\*\* contrE(int\*\* G, int size, int v1, int v2) {  
if (v1 < 0 || v1 >= size || v2 < 0 || v2 >= size || G[v1][v2] == 0) {  
printf("Ребра между вершинами нет, операция невозможна.\n");  
return G;  
}  
G[v1][v2] = G[v2][v1] = 0;  
for (int i = 0; i < size; i++) {  
if (G[v2][i]) {  
G[v1][i] = 1;  
G[i][v1] = 1;  
}  
}  
return delv(G, size, v2);  
}  
  
// Операция расщепления вершины  
int\*\* splitV(int\*\* G, int size, int v) {  
if (v < 0 || v >= size) {  
printf("Ошибка: номер вершины вне допустимого диапазона (0-%d).\n", size - 1);  
return G;  
}  
int\*\* Gtemp = (int\*\*)malloc((size + 1) \* sizeof(int\*));  
for (int i = 0; i < size + 1; i++) {  
Gtemp[i] = (int\*)calloc(size + 1, sizeof(int));  
}  
  
for (int i = 0; i < size; i++) {  
for (int j = 0; j < size; j++) {  
Gtemp[i][j] = G[i][j];  
}  
}  
  
for (int i = 0; i < size; i++) {  
Gtemp[i][size] = G[i][v];  
Gtemp[size][i] = G[i][v];  
}  
Gtemp[v][size] = 1;  
Gtemp[size][v] = 1;  
  
for (int i = 0; i < size; i++) {  
free(G[i]);  
}  
free(G);  
return Gtemp;  
}  
  
// Главная функция  
int main() {  
setlocale(LC\_ALL, "Russian");  
srand(time(NULL));  
  
int sizeG1, sizeG2;  
  
// Запрос размера графов  
printf("Введите количество вершин для графа 1: ");  
scanf("%d", &sizeG1);  
printf("Введите количество вершин для графа 2: ");  
scanf("%d", &sizeG2);  
  
// Генерация случайных графов  
int\*\* G1 = generG(sizeG1);  
int\*\* G2 = generG(sizeG2);  
  
// Печать графов  
printf("\nГраф 1:\n");  
printG(G1, sizeG1);  
printf("\nГраф 2:\n");  
printG(G2, sizeG2);  
  
printf("\n=== Унарные операции ===\n");  
  
// Отождествление вершин  
printf("\nОтождествление вершин. Введите номера вершин (0-%d): ", sizeG1 - 1);  
int v1, v2;  
scanf("%d %d", &v1, &v2);  
G1 = unionv(G1, sizeG1, v1, v2);  
sizeG1--;  
printG(G1, sizeG1);  
  
// Стягивание ребра  
printf("\nСтягивание ребра. Введите номера вершин (0-%d): ", sizeG1 - 1);  
scanf("%d %d", &v1, &v2);  
G1 = contrE(G1, sizeG1, v1, v2);  
sizeG1--;  
printG(G1, sizeG1);  
  
// Расщепление вершины  
printf("\nРасщепление вершины. Введите номер вершины (0-%d): ", sizeG1 - 1);  
scanf("%d", &v1);  
G1 = splitV(G1, sizeG1, v1);  
sizeG1++;  
printG(G1, sizeG1);  
  
return 0;  
}

Задание 2, 3

[#include](https://vk.com/im?sel=297349005&st=%23include) <stdio.h>  
[#include](https://vk.com/im?sel=297349005&st=%23include) <stdlib.h>  
[#include](https://vk.com/im?sel=297349005&st=%23include) <locale.h>  
[#include](https://vk.com/im?sel=297349005&st=%23include) <time.h>  
  
// Функция для генерации случайного графа  
int\*\* generG(int size) {  
int\*\* G = (int\*\*)malloc(size \* sizeof(int\*));  
for (int i = 0; i < size; i++) {  
G[i] = (int\*)malloc(size \* sizeof(int));  
for (int j = 0; j < size; j++) {  
G[i][j] = 0; // Инициализация нулями  
}  
}  
for (int i = 0; i < size; i++) {  
for (int j = i + 1; j < size; j++) {  
int edge = rand() % 2;  
G[i][j] = edge;  
G[j][i] = edge;  
}  
}  
return G;  
}  
  
// Функция для копирования графа  
int\*\* copyG(int\*\* G, int size) {  
int\*\* newG = (int\*\*)malloc(size \* sizeof(int\*));  
for (int i = 0; i < size; i++) {  
newG[i] = (int\*)malloc(size \* sizeof(int));  
for (int j = 0; j < size; j++) {  
newG[i][j] = G[i][j];  
}  
}  
return newG;  
}  
  
// Функция для объединения графов  
int\*\* unionG(int\*\* G1, int\*\* G2, int size) {  
int\*\* G = (int\*\*)malloc(size \* sizeof(int\*));  
for (int i = 0; i < size; i++) {  
G[i] = (int\*)malloc(size \* sizeof(int));  
for (int j = 0; j < size; j++) {  
G[i][j] = G1[i][j] || G2[i][j];  
}  
}  
return G;  
}  
  
// Функция для пересечения графов  
int\*\* intersectG(int\*\* G1, int\*\* G2, int size) {  
int\*\* G = (int\*\*)malloc(size \* sizeof(int\*));  
for (int i = 0; i < size; i++) {  
G[i] = (int\*)malloc(size \* sizeof(int));  
for (int j = 0; j < size; j++) {  
G[i][j] = G1[i][j] && G2[i][j];  
}  
}  
return G;  
}  
  
// Функция для кольцевой суммы графов  
int\*\* symmetricDiffG(int\*\* G1, int\*\* G2, int size) {  
int\*\* G = (int\*\*)malloc(size \* sizeof(int\*));  
for (int i = 0; i < size; i++) {  
G[i] = (int\*)malloc(size \* sizeof(int));  
for (int j = 0; j < size; j++) {  
G[i][j] = (G1[i][j] + G2[i][j]) % 2;  
}  
}  
return G;  
}  
  
// Функция для вывода графа  
void printG(int\*\* G, int size) {  
printf("Матрица смежности:\n");  
for (int i = 0; i < size; i++) {  
for (int j = 0; j < size; j++) {  
printf("%d ", G[i][j]);  
}  
printf("\n");  
}  
}  
  
// Функция для освобождения памяти графа  
void freeG(int\*\* G, int size) {  
for (int i = 0; i < size; i++) {  
free(G[i]);  
}  
free(G);  
}  
  
// Главная функция  
int main() {  
setlocale(LC\_ALL, "Russian");  
srand(time(NULL));  
  
int sizeG1, sizeG2;  
  
// Запрос размера графов  
printf("Введите количество вершин для графа 1: ");  
scanf("%d", &sizeG1);  
printf("Введите количество вершин для графа 2: ");  
scanf("%d", &sizeG2);  
  
// Генерация случайных графов  
int\*\* G1 = generG(sizeG1);  
int\*\* G2 = generG(sizeG2);  
  
// Печать графов  
printf("\nИзначальный граф 1:\n");  
printG(G1, sizeG1);  
printf("\nГраф 2:\n");  
printG(G2, sizeG2);  
  
printf("\n=== Бинарные операции с изначальным G1 ===\n");  
  
if (sizeG1 == sizeG2) {  
// Объединение  
int\*\* tempG1 = copyG(G1, sizeG1);  
printf("\nОбъединение графов G1 ∪ G2:\n");  
int\*\* unionResult = unionG(tempG1, G2, sizeG1);  
printG(unionResult, sizeG1);  
freeG(tempG1, sizeG1);  
freeG(unionResult, sizeG1);  
  
// Пересечение  
tempG1 = copyG(G1, sizeG1);  
printf("\nПересечение графов G1 ∩ G2:\n");  
int\*\* intersectResult = intersectG(tempG1, G2, sizeG1);  
printG(intersectResult, sizeG1);  
freeG(tempG1, sizeG1);  
freeG(intersectResult, sizeG1);  
  
// Кольцевая сумма  
tempG1 = copyG(G1, sizeG1);  
printf("\nКольцевая сумма графов G1 ⊕ G2:\n");  
int\*\* symmetricDiffResult = symmetricDiffG(tempG1, G2, sizeG1);  
printG(symmetricDiffResult, sizeG1);  
freeG(tempG1, sizeG1);  
freeG(symmetricDiffResult, sizeG1);  
} else {  
printf("Графы должны быть одинакового размера для выполнения бинарных операций.\n");  
}  
  
// Освобождение памяти для исходных графов  
freeG(G1, sizeG1);  
freeG(G2, sizeG2);  
  
return 0;  
}