Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

за 1 семестр

По дисциплине: «Дискретная математика»

Тема: «Отношения. Функции»

Выполнил:

Студент 2 курса

Группы ПО-4(2)

Кречко К. А.

Проверила:

Глущенко Т.А.

2020

**ЛАБОРАТОРНАЯ РАБОТА No 2.**

**Отношения.Функции.**

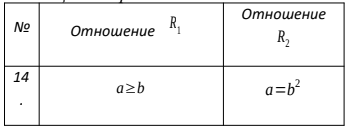
**Цель**: изучить теорию по теме «Отношения», ознакомиться с понятиями, данными в теории. По данному варианту программно реализовать предложенные задачи.

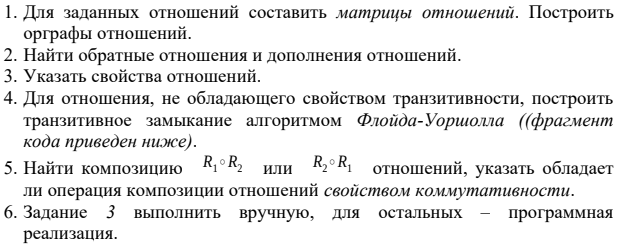
**Отношения.**

**Задание 1.**

На множестве A = {1,2,3,4,5,6,7,8,9,10} заданы отношения R1 и R2 согласно вашему варианту.

**Вариант 14**

****

****

**1.** Для заданных отношений составить матрицы отношений. Построить орграфы отношений.

**Код создания матрицы:**

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int A[] = { 1,2,3,4,5,6,7,8,9,10 };

int sizeA = sizeof(A) / sizeof(int);

const int ROWS = 11;

const int COLS = 11;

int arr[ROWS][COLS];

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

{

arr[i][0] = i;

arr[0][i] = i;

}

for (int i = 1; i < ROWS; i++)

{

for (int j = 1; j < COLS; j++)

{

if (arr[i][0] >= arr[0][j]) //Для отношения R2 (arr[i][0] == pow(arr[0][j],2))

{

arr[i][j] = 1;

}

else

{

arr[i][j] = 0;

}

}

}

for (int i = 1; i < ROWS; i++)

{

for (int j = 1; j < COLS; j++)

{

cout << arr[i][j] << " ";

}

cout << endl;

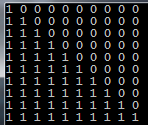
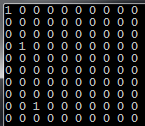
}

return 0;

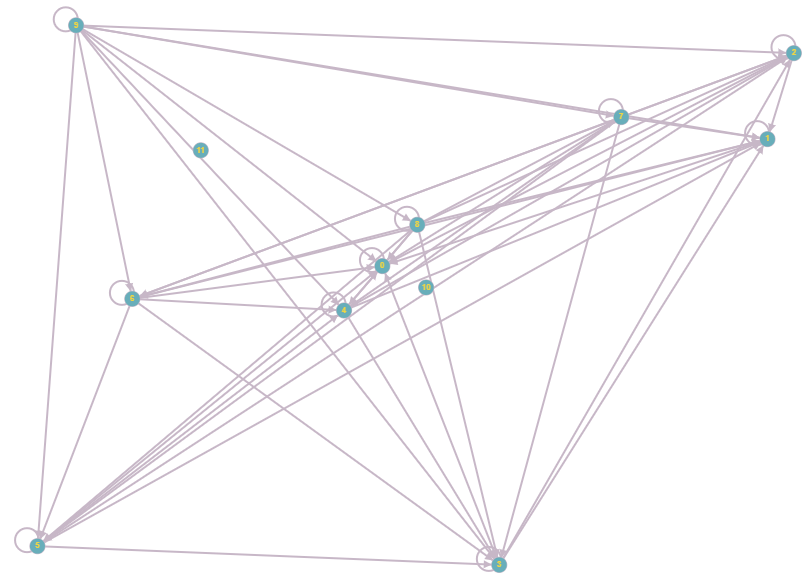
}

**Результат:**

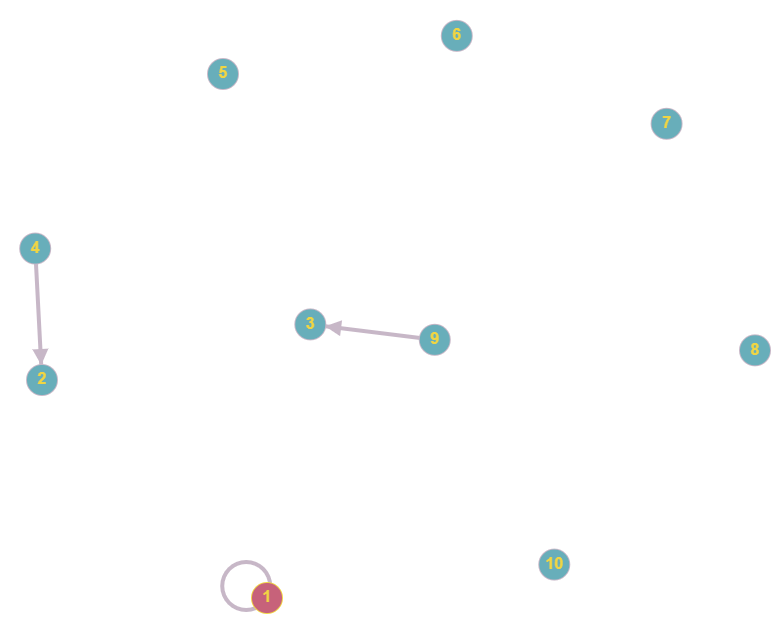
**R1 R2**

** **

**Орграф для R1**

****

**Орграф для R2**

****

**2.** Найти обратные отношения и дополнения отношений.

Формула для обратного отношения (транспонируемая матрица): 

Обратное отношение для R1:

R1-1 = {(b,a)|a,b ɛ А^ a>=b}

Обратное отношение для R2:

R1-1 = {(b,a)|a,b ɛ А^ a=b2}

**Код:**

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int A[] = { 1,2,3,4,5,6,7,8,9,10};

int sizeA = sizeof(A) / sizeof(int);

const int ROWS = 11;

const int COLS = 11;

int arr[ROWS][COLS];

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

{

arr[i][0] = i;

arr[0][i] = i;

}

for (int i = 1; i < ROWS; i++)

{

for (int j = 1; j < COLS; j++)

{

if (arr[i][0] >= arr[0][j]) //Для отношения R2 (arr[i][0] == pow(arr[0][j],2))

{

arr[j][i] = 1;

}

else

{

arr[j][i] = 0;

}

}

}

for (int i = 1; i < ROWS; i++)

{

for (int j = 1; j < COLS; j++)

{

cout << arr[i][j] << " ";

}

cout << endl;

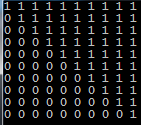
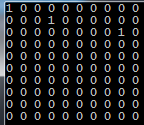
}

return 0;

}

**Результат:**

**R1 R2**

** **

Формула дополнения отношения (инвертируемая матрица):



Тогда для:

!R1 ={(a,b)|a,b ɛ А^ a<b}

!R2 ={(a,b)|a,b ɛ А^ a!=b2}

**Код:**

for (int i = 1; i < ROWS; i++)

{

for (int j = 1; j < COLS; j++)

{

if (arr[i][0] < arr[0][j]) //Для отношения R2 (arr[i][0] != pow(arr[0][j],2))

{

arr[i][j] = 1;

}

else

{

arr[i][j] = 0;

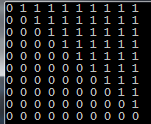
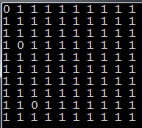
}

}

}

…

**Результат:  
R1 R2**

** **

**3.** Указать свойства отношений.

R1

Рефлексивность + (т.к. отношение содержит знак равенства);

Антирефлексивность - (существует к примеру пара (1,1));

Симметричность - (т.к. 8>=2 обратное неверно);

Антисимметричность + (т.к. отношение содержит знак равенства (21>=22&&22>=21 =>21=22). Другими словами отношение содержит >=);

Транзитивность + (т.к. отношение содержит >= (есть пары(4>=2) и (2>=1) из неё образуется пара (4>=1));

R2

Рефлексивность + (т.к. отношение содержит знак равенства)

Антирефлексивность - (т.к. существует пара (1=1^2)

Симметричность + (т.к. все бинарные отношения в геометрии типа равенства симметричны)

Антисимметричность +

Транзитивность - ( т.к. есть пары (16;4) и (4;2), а пары (16;2) нет).

**4.** Для отношения, не обладающего свойством транзитивности, построить транзитивное замыкание алгоритмом Флойда-Уоршолла

Для отношения R2:

**Код:**

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int A[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

int sizeA = sizeof(A) / sizeof(int);

const int ROWS = 17;

const int COLS = 17;

int arr[ROWS][COLS];

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

{

arr[i][0] = i;

arr[0][i] = i;

}

for (int i = 1; i < ROWS; i++)

{

for (int j = 1; j < COLS; j++)

{

if (arr[i][0] == pow(arr[0][j], 2))

{

arr[i][j] = 1;

}

else

{

arr[i][j] = 0;

}

}

}

int n = 16;

for (int k = 1; k < n; k++)

{

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

{

for (int j = 1; j < n; j++)

{

arr[i][j] = arr[i][j] || arr[i][k] && arr[k][j];

};

};

};

for (int i = 1; i < ROWS; i++)

{

for (int j = 1; j < COLS; j++)

{

cout << arr[i][j] << " ";

}

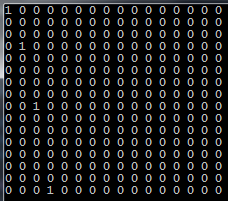
cout << endl;

}

return 0;

}

**Результат:**



**5.** Найти композицию R1∘R2 или R2∘R1 отношений, указать обладает ли операция композиции отношений свойством коммутативности.

**Код:**

#include<iostream>

using namespace std;

void show\_matrix(int arr[11][11], const int size) { //функция для вывода матрицы

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

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

cout << arr[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

void print\_index(int arr[11][11], const int size, int choice) { //функция для вывода индексов где есть 1

for (int i = 1; i < size; i++) {

for (int j = 1; j < size; j++) {

if (arr[i][j] == 1) {

cout << "[" << i << ";" << j << "]" << " ";

}

}

if (choice == 0) {

cout << endl;

}

}

cout << endl;

}

void find\_commonR\_S(int arrS[11][11], int arrR[11][11], int arrRS[11][11], const int size) {

int temp;

int temp1;

for (int i = 1; i < size; i++) {

for (int j = 1; j < size; j++) {

if (arrR[i][j] == 1) {

temp = j;

temp1 = i;

}

if (arrS[temp][j] == 1) {

arrRS[temp1][j] = 1;

}

}

}

}

void find\_commonS\_R(int arrS[11][11], int arrR[11][11], int arrRS[11][11], const int size) {

int temp;

int temp1;

for (int i = 1; i < size; i++) {

for (int j = 1; j < size; j++) {

int count = 0;

if (arrS[i][j] == 1) {

temp = j;

temp1 = i;

while (count < size) {

if (arrR[temp][count] == 1) {

arrRS[temp1][count] = 1;

}

count++;

}

}

}

}

}

int main() {

const int size = 11;

int S[size][size];

int R[size][size];

int RS[size][size];

for (int i = 0; i < size; i++) { //Заполняю первый столбик и строку

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

S[0][j] = j;

S[j][0] = j;

R[0][j] = j;

R[j][0] = j;

RS[i][j] = 0;

RS[0][j] = j;

RS[j][0] = j;

}

}

for (int i = 1; i < size; i++) { //Заполняю S нулями и еденицами

for (int j = 1; j < size; j++) {

if (i >= j) {

S[i][j] = 1;

}

else {

S[i][j] = 0;

}

}

}

for (int i = 1; i < size; i++) { //Заполняю R нулями и еденицами

for (int j = 1; j < size; j++) {

if (j \* j == i) {

R[i][j] = 1;

}

else {

R[i][j] = 0;

}

}

}

int choice = 0;

cout << "R1:" << endl;

show\_matrix(S, size);

print\_index(S, size, choice);

choice = 1;

cout << "R2:" << endl;

show\_matrix(R, size);

print\_index(R, size, choice);

cout << endl;

find\_commonR\_S(S, R, RS, size);

cout << "R1R2:" << endl;

show\_matrix(RS, size);

print\_index(RS, size, choice);

for (int i = 0; i < size; i++) { //Приводим R\_S в начальное положение

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

RS[i][j] = 0;

RS[0][j] = j;

RS[j][0] = j;

}

}

choice = 0;

find\_commonS\_R(S, R, RS, size);

cout << "R2R1:" << endl;

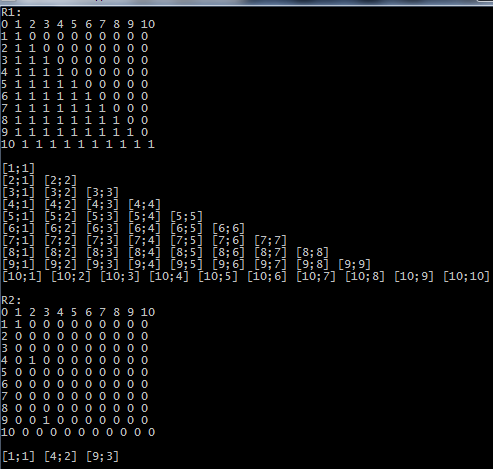
show\_matrix(RS, size);

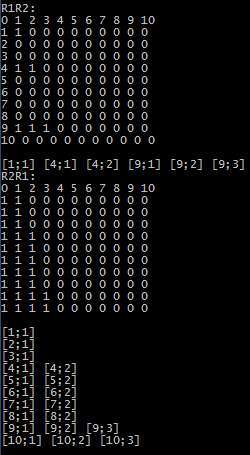
print\_index(RS, size, choice);

return 0;

}

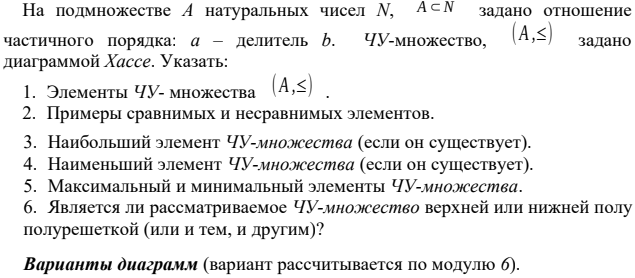
**Результат:**



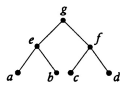


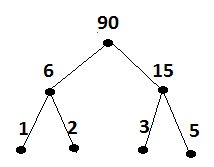
Отсюда видно, что данная операция не обладает свойством коммутативности.

**Задание 2.**



14 mod 6=2





1. Элементы ЧУ-множества: {1,2,3,5,6,15,90}

2. Сравнимые элементы: 1 и 6, 3 и 15, 6 и 90.

Несравнимые элементы: 1 и 2, 6 и 15, 3 и 5.

3.Наибольший элемент g (90)

4.Наименьшего элемента не существует

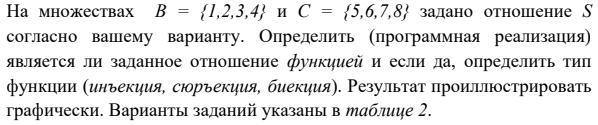
5. Макс. элемент g (90)

Мин. элементы a,b,c,d

6. Данное ЧУ-множество является верхней полурешеткой.

**Функции.**

**Задание 1.**





**Код:**

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

int C[] = { 5,6,7};

int sizeC = 4;

const int ROWS = 5;

const int COLS = 5;

int arr[ROWS][COLS];

bool bee = 0;

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

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

arr[i][j] = 0;

}

}

//заполнение матрицы

for (int i = 1; i < ROWS; i++) {

arr[i][0] = i;

arr[0][i] = i + 4;

}

arr[3][1] = arr[1][2] = arr[4][3] = arr[2][4] = 1;

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

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

cout << arr[i][j] << " ";

}

cout << endl;

}

int k = 0;

for (int x = 1; x < ROWS; x++) {

for (int y = 1; y < COLS; y++) {

if (arr[x][y] == 1) {

k++; //узнаем, функция это или нет

}

}

if (k > 1 || k == 0) {

cout << "Это не функция" << endl;

k = 1;

break;

}

k = 0;

}

if (k == 1) return 0;

cout << "Это функция" << endl;

for (int y = 0, n = 0; y < COLS; y++) {

for (int x = 0; x < ROWS; x++) {

if (arr[x][y] == 1) {

n++;

}

}

if (n > 1) {

cout << "функция не инъективна" << endl; //узнаем, инъективна ли эта функция

bee = 1;

break;

}

n = 0;

bee = 0;

}

for (int y = 1; y < COLS; y++) {

for (int x = 1; x < ROWS; x++) {

if (arr[y][x] == 1) { //узнаем, сюръективна ли эта функция

sizeC--;

break;

}

}

}

if (sizeC == bee) {

cout << "Функция биективна " << endl;

}

else if (sizeC != 0) {

cout << "Функция не cюръективна " << endl;

}

else {

cout << "Функция cюръективна " << endl;

}

if (sizeC != 0 && bee == 0) {

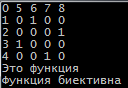
cout << "Функция инъективна" << endl;

}

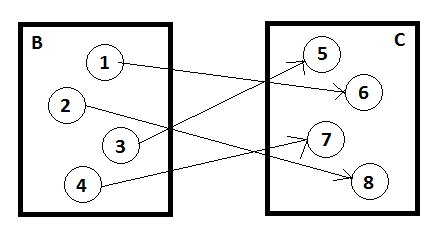
return 0;

}

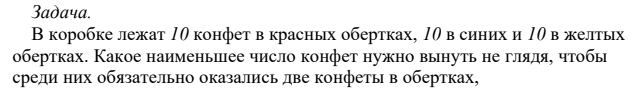
**Результат:**

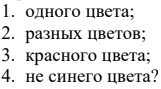
****

**Графически:**

****

**Задание 2.**

****

****

**Ответ:**

1.4 (худший случай, когда выпадает 3 разноцветных, на 4-й обязательно будет одним из трех цветов).

2.11 (худший случай, когда выпадет, к примеру, 10 синих, то на 11 будет уже другой цвет).

3. 22(10 синих и 10 желтых, останутся только красные).

4. 12 (10 синих, потом другие цвета).

**Вывод:**

Я изучил теорию по теме «Отношения», ознакомился с понятиями, данными в теории. По данному варианту программно реализовал предложенные задачи.