**МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

**Кафедра программного обеспечения вычислительной техники и автоматизированных систем**

Лабораторная работа №3.1

Дисциплина: Дискретная математика

по теме «**Отношения и их свойства**»

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Рязанов Ю.Д

**Белгород 2018**

**Практическое занятие 3.1  
Отношения и их свойства**

Цель занятия: изучить способы задания отношений, операции над отношениями и свойства отношений, научиться программно реализовывать операции и определять свойства отношений.

**Задания**

**Часть 1. Операции над отношениями**

1.1. Представить отношения на множестве {1,2,3,4,5,6,7,8,9,10}(см.‖Варианты заданий‖, п.а) графиком, графом и матрицей.

1.2. Вычислить значение выражения (см.‖Варианты заданий‖, п.б) при заданных отношениях (см.‖Варианты заданий‖, п.а).

1.3. Написать программы, формирующие матрицы заданных отношений (см.‖Варианты заданий‖, п.а).

1.4. Программно реализовать операции над отношениями.

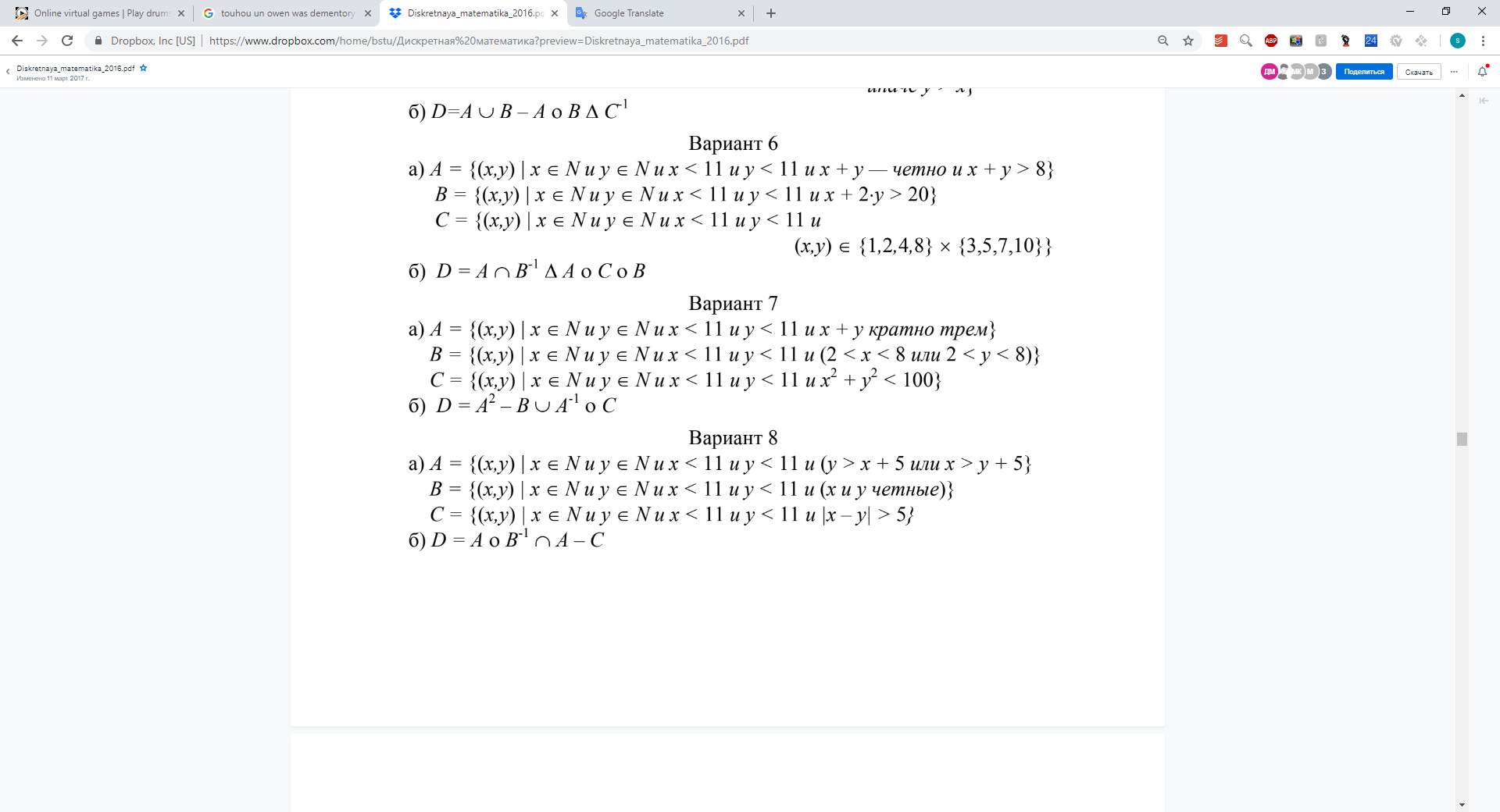
1.5. Написать программу, вычисляющую значение выражения (см.―Варианты заданий‖, п.б) и вычислить его при заданных отношениях (см.‖Варианты заданий‖, п.а).

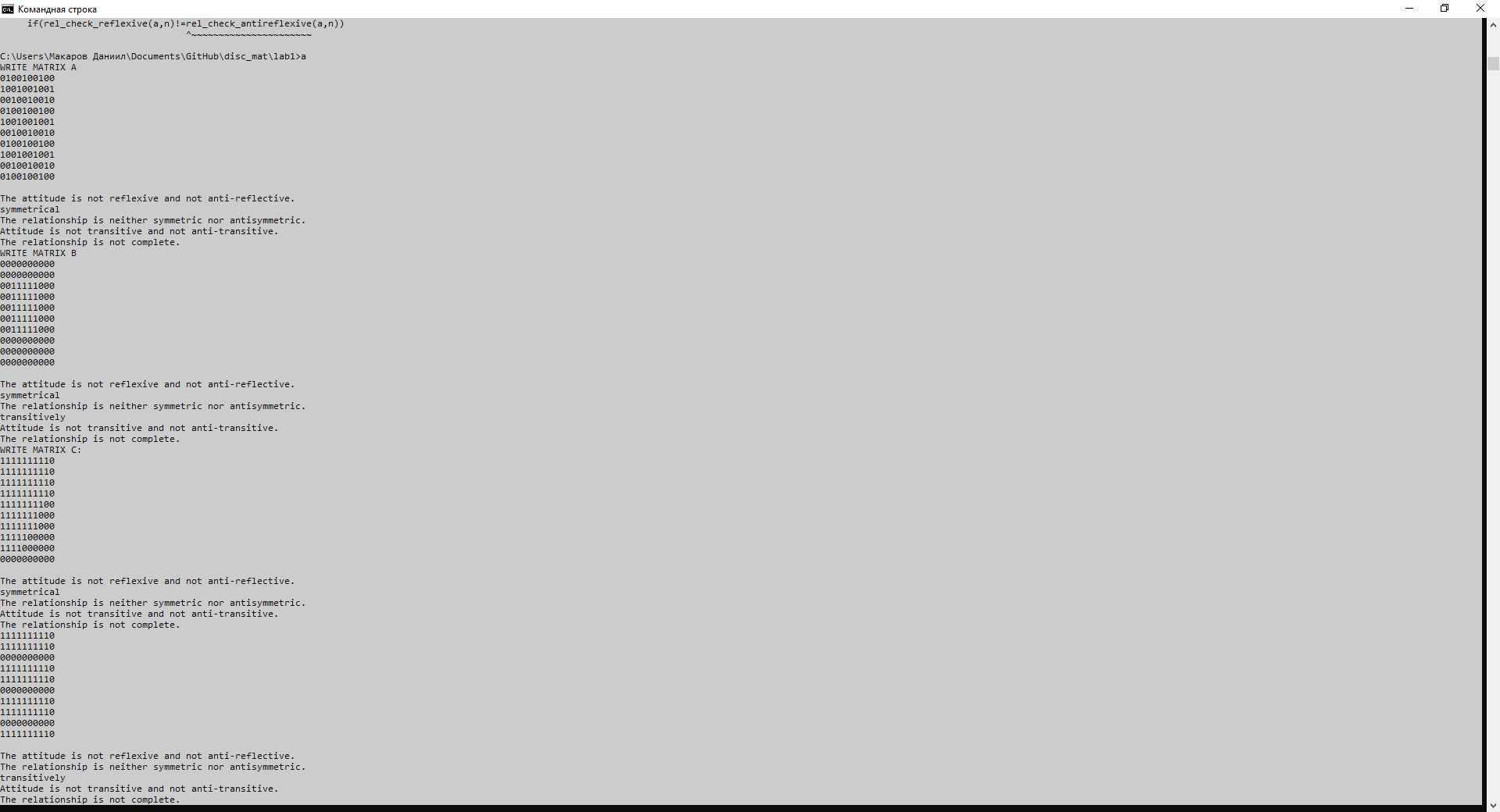
**Часть 2. Свойства отношений**

2.1. Определить основные свойства отношений (см. ‖Варианты зада-ний‖, п.а).

2.2. Определить, являются ли заданные отношения отношениями толерантности, эквивалентности и порядка.

2.3. Написать программу, определяющую свойства отношения, в том числе толерантности, эквивалентности и порядка, и определить свойства отношений (см. ‖Варианты заданий‖, п.а).





**Исходный код**

#include <stdio.h>

#include <stdlib.h>

#define N 10

int rel\_A(int x,int y){

return (((x+y)%3)==0);

}

int rel\_B(int x,int y){

return (((2<x)&&(x<8))&&((2<y)&&(y<8)));

}

int rel\_C(int x,int y){

return ((x\*x+y\*y)<100);

}

void build\_rel\_matr(int matr[N][N], int func(int, int),int n){

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

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

matr[i][j] = func(i+1, j+1);

}

}

}

void write\_matr(int matr[N][N],int n){

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

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

printf("%i", matr[i][j]);

}

printf("\n");

}

}

int rel\_equal(int a[N][N],int b[N][N],int n){

int f=1;

int x=0;

int y;

while(x<=n && f){

y=0;

while(y<=n && f){

f=(a[x][y]==b[x][y]);

y++;

}

x++;

}

return f;

}

int rel\_inclusion(int a[N][N],int b[N][N],int n){

int f=1;

int x=0;

int y;

while(x<=n && f)Х

y=0;

while(y<=n && f){

f=(a[x][y]<=b[x][y]);

y++;

}

x++;

}

return f;

}

void rel\_union(int a[N][N],int b[N][N],int c[N][N],int n){

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

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

c[i][j]=a[i][j] || b[i][j];

}

}

}

void rel\_intersect(int a[N][N],int b[N][N],int c[N][N],int n){

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

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

c[i][j]=a[i][j] && b[i][j];

}

}

}

void rel\_substraction(int a[N][N],int b[N][N],int c[N][N],int n){

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

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

c[i][j]=a[i][j] > b[i][j];

}

}

}

void rel\_sumetric\_substraction(int a[N][N],int b[N][N],int c[N][N],int n){

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

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

c[i][j]=a[i][j] ^ b[i][j];

}

}

}

void rel\_complited(int a[N][N],int c[N][N],int n){

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

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

c[i][j]=!a[i][j];

}

}

}

void rel\_exp(int a[N][N],int c[N][N],int n){

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

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

c[i][j]=a[i][j];

}

}

}

void rel\_composition(int a[N][N], int b[N][N], int c[N][N],int n){

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

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

c[i][j]=0;

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

c[i][j] = c[i][j] || (a[i][z] && b[z][j]);

}

}

}

}

void rel\_check\_reflexive\_antireflexive(int a[N][N],int n){

int f,k;

if(rel\_check\_reflexive(a,n)!=rel\_check\_antireflexive(a,n)){

printf("\nThe attitude is not reflexive and not anti-reflective.\n");

}

else{

if(rel\_check\_reflexive(a,n)==1 && rel\_check\_antireflexive(a,n)==1)

printf("\nReflexive attitude\n");

else

printf("\nAnti-reflexive attitude\n");

}

}

int rel\_check\_reflexive(int a[N][N],int n){

int k=1;

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

if(a[i][i]==1)

k=1;

else

k=0;

}

return k;

}

int rel\_check\_antireflexive(int a[N][N],int n){

int k=0;

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

{

if(a[i][i]==0)

k=0;

else

k=1;

}

return k;

}

int rel\_check\_symmetrical(int a[N][N],int n){

int i,j;

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

for(j=0;j<i;j++)

if(a[i][j]!=a[j][i])

return 0;

return 1;

}

int rel\_check\_antisymmetrical(int a[N][N],int n){

int i,j;

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

for(j=0;j<i;j++)

if(a[i][j]\*a[j][i])

return 0;

return 1;

}

int rel\_check\_transitively(int a[N][N],int n){

int i,j,k;

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

for(k=0;k<n;k++)

if(a[i][k])

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

if(a[k][j]==1&&a[i][j]==0)

return 0;

return 1;

}

int rel\_check\_antitransitively(int a[N][N],int n){

int i,j,k;

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

for(k=0;k<n;k++)

if(a[i][k])

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

if(a[k][j]&&a[i][j])

return 0;

return 1;

}

int rel\_check\_full(int a[N][N],int n){

int i,j;

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

for(j=0;j<i;j++)

if(!(a[i][j]||a[j][i]))

return 0;

return 1;

}

int rel\_check\_tolerance(int a[N][N],int n){

int f,g;

f=rel\_check\_reflexive(a,n);

g=rel\_check\_symmetrical(a,n);

return(f\*g);

}

int rel\_check\_equivalence(int a[N][N],int n){

int f,g,l;

f=rel\_check\_reflexive(a,n);

g=rel\_check\_symmetrical(a,n);

l=rel\_check\_transitively(a,n);

return(f\*g\*l);

}

int rel\_check\_order(int a[N][N],int n){

int f,g;

f=rel\_check\_antisymmetrical(a,n);

g=rel\_check\_transitively(a,n);

return(f\*g);

}

int rel\_check\_characteristic(int a[N][N],int n){

rel\_check\_reflexive\_antireflexive(a,n);

int f,c;

f=rel\_check\_symmetrical(a,n);

c=rel\_check\_antisymmetrical(a,n);

if(f==1)

printf("symmetrical\n");

if(c==1)

printf("antisymmetrical\n");

else

printf("The relationship is neither symmetric nor antisymmetric.\n");

f=rel\_check\_transitively(a,n);

c=rel\_check\_antitransitively(a,n);

if(f==1)

printf("transitively\n");

if(c==1)

printf("antitransitively\n");

else

printf("Attitude is not transitive and not anti-transitive.\n");

f=rel\_check\_full(a,n);

if(f==1)

printf("The attitude is complete.");

else

printf("The relationship is not complete.");

f=rel\_check\_tolerance(a,n);

if(f==1)

printf("Relationship is tolerant.");

f=rel\_check\_equivalence(a,n);

if(f==1)

printf("The ratio is equivalent to.");

f=rel\_check\_order(a,n);

if(f==1)

printf("Order relationship.");

}

int main(){

int a[N][N] = {};

int b[N][N] = {};

int c[N][N] = {};

int d[N][N] = {};

int t1[N][N] = {};

int t2[N][N] = {};

int n=N;

build\_rel\_matr(a, rel\_A,n);

printf("WRITE MATRIX A\n");

write\_matr(a,n);

rel\_check\_characteristic(a,n);

printf("\n");

build\_rel\_matr(b, rel\_B,n);

printf("WRITE MATRIX B\n");

write\_matr(b,n);

rel\_check\_characteristic(b,n);

printf("\n");

printf("WRITE MATRIX C:\n");

build\_rel\_matr(c, rel\_C,n);

write\_matr(c,n);

rel\_check\_characteristic(c,n);

printf("\n");

rel\_exp(a,t1,n);

rel\_union(t1,b,t1,n);

rel\_composition(a,a,t2,n);

rel\_substraction(t2,t1,t1,n);

rel\_composition(t1,c,d,n);

write\_matr(d,n);

rel\_check\_characteristic(d,n);

return 0;

}