РОССИЙСКОЙ ФЕДЕРАЦИИ
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

по дисциплине: Дискретная математика
тема: «Отношения и их свойства»

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Рязанов Юрий Дмитриевич

Белгород 2022 г.

Цель работы: изучить способы задания отношений, операции над от ношениями и свойства отношений, научиться программно реализовывать операции и определять свойства отношений.

Содержание отчета:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Тексты заданий и их решения.
- Выводы.

Задания:
Вариант 10
Часть 1. Операции над отношениями

1.1. Представить отношения (см."Варианты заданий п.а) графиком, графом и матрицей.

а) $A = \{(x,y) \mid x \in N$ и $y \in N$ и $x < 11$ и $y < 11$ и $x < y < (9 - x)$ или $(9 - x) < y < x\}$

x/y	1	2	3	4	5	6	7	8	9	10
1		1	1	1	1	1	1			
2			1	1	1	1				
3				1	1					
4										
5										
6				1	1					
7			1	1	1	1				
8		1	1	1	1	1	1			
9	1	1	1	1	1	1	1	1		
10	1	1	1	1	1	1	1	1	1	

таблица А

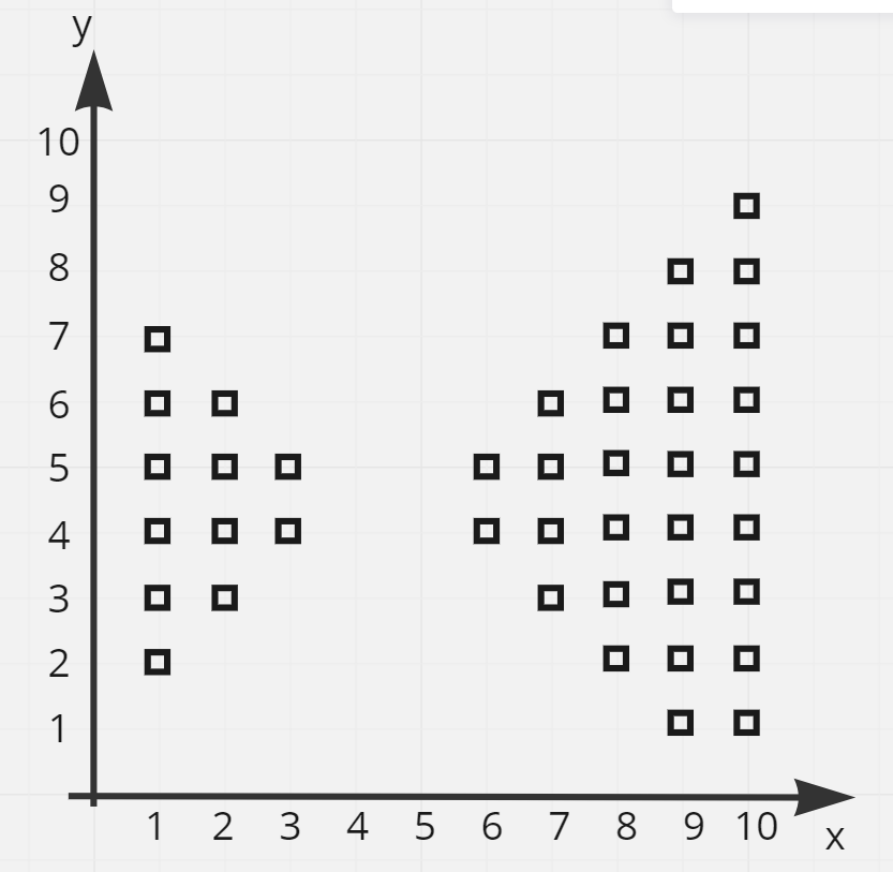
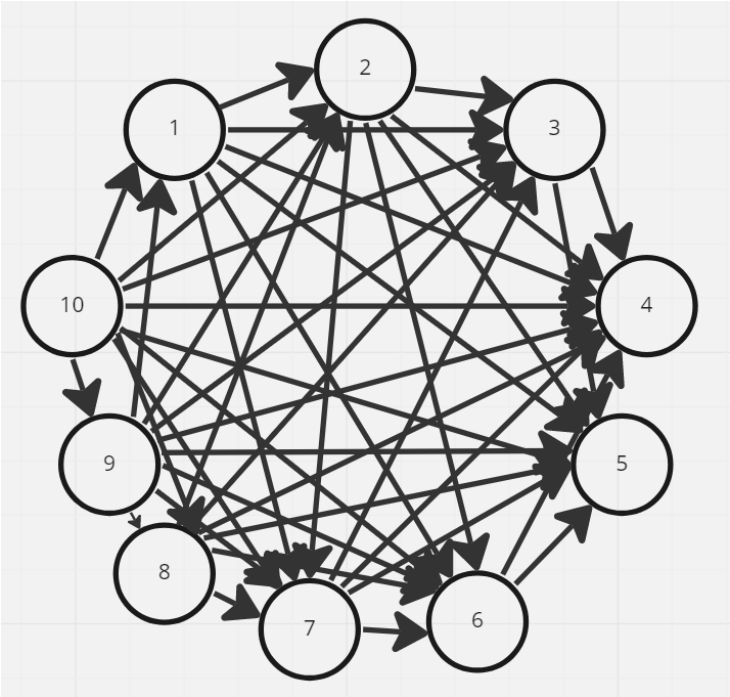


График А



Граф А

б) A = {(x,y) | x ∈ N и y ∈ N и x < 11 и y < 11 и x *четно* и y *нечетно*}

x/y	1	2	3	4	5	6	7	8	9	10
1										
2	1		1		1		1		1	
3										
4	1		1		1		1		1	
5										
6	1		1		1		1		1	
7										
8	1		1		1		1		1	
9										
10	1		1		1		1		1	

Таблица Б

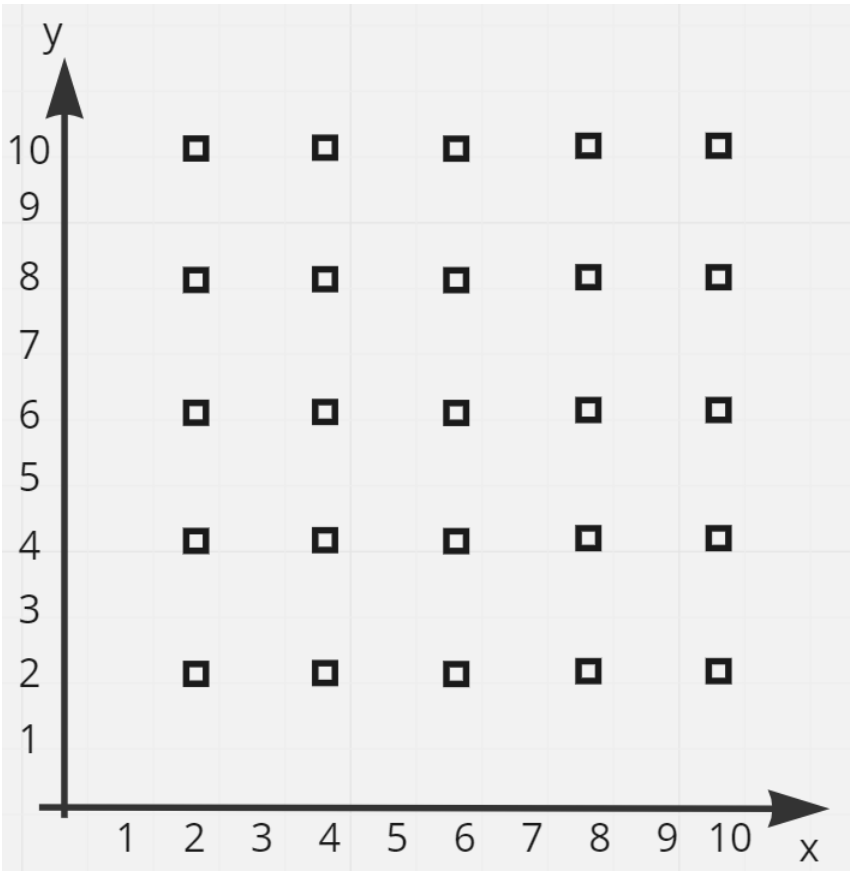
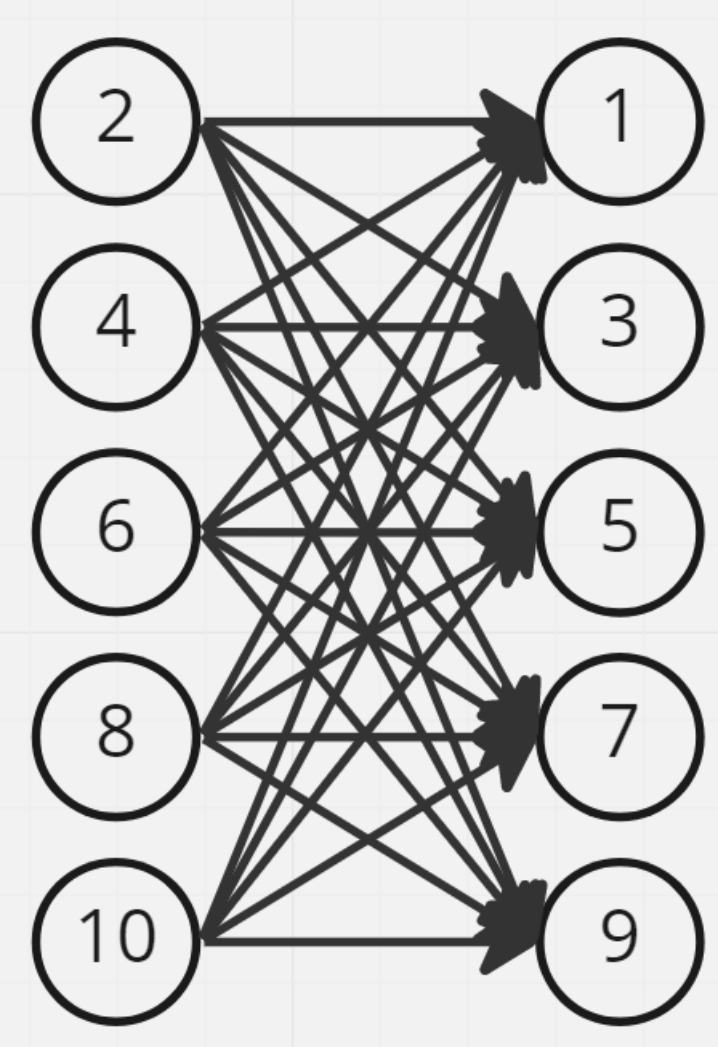


График Б



Граф Б

в) A = {(x,y) | x ∈ N и y ∈ N и x < 11 и y < 11 и x*y *кратно трем*}

x/y	1	2	3	4	5	6	7	8	9	10
1			1			1			1	
2			1			1			1	
3	1	1	1	1	1	1	1	1	1	1
4			1			1			1	
5			1			1			1	
6	1	1	1	1	1	1	1	1	1	1
7			1			1			1	
8			1			1			1	
9	1	1	1	1	1	1	1	1	1	1
10			1			1			1	

Таблица В

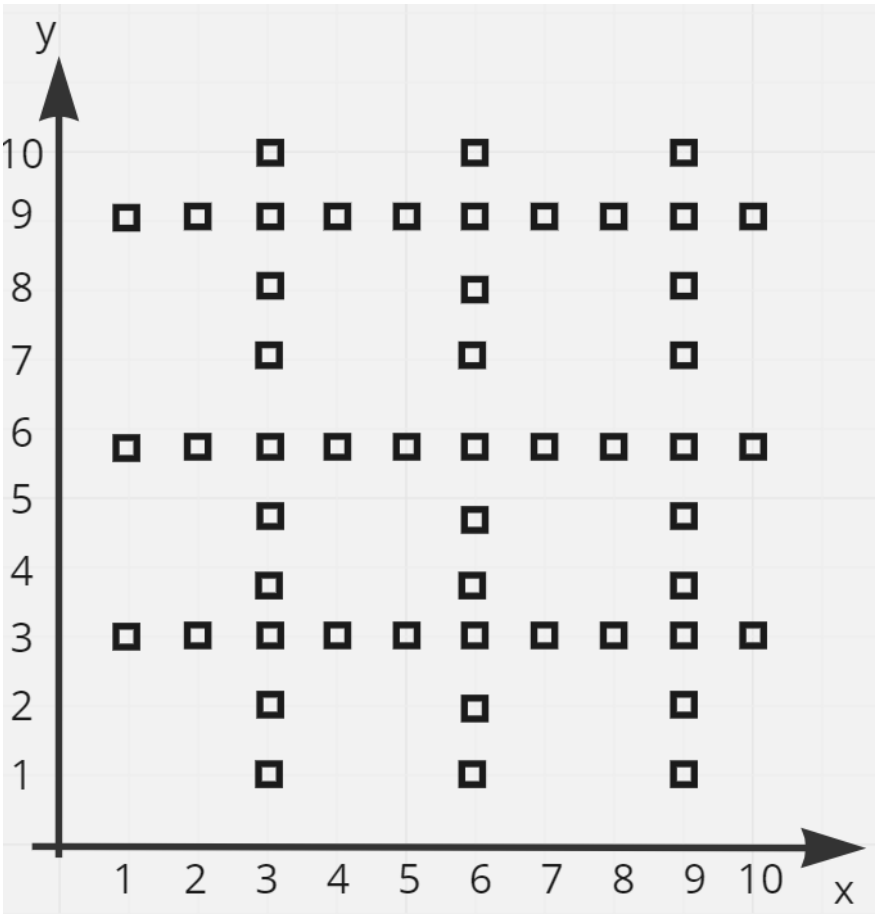
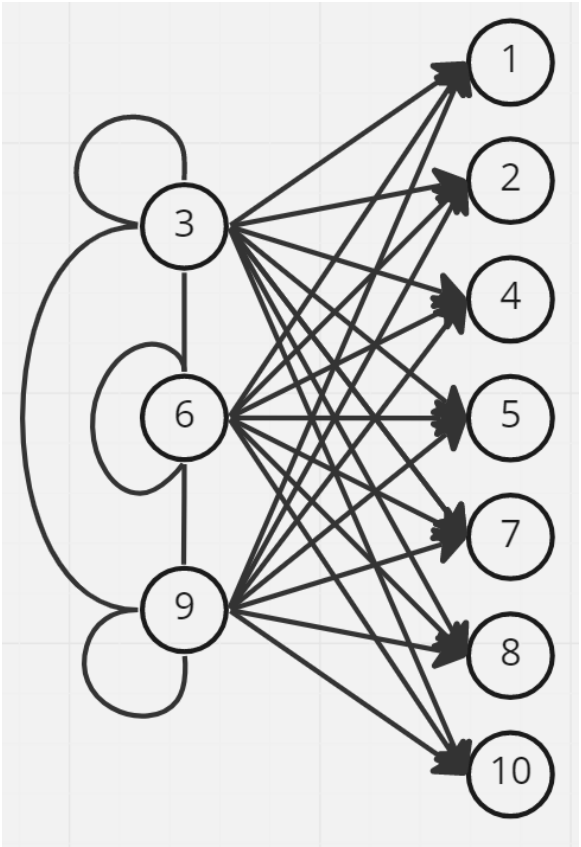


График В



Граф B

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

$D = A \circ B - \bigstar C \cup C^{-1}$

1) $I = A \circ B$

x/y	1	2	3	4	5	6	7	8	9	10
1	1		1		1		1		1	
2	1		1		1		1		1	
3	1		1		1		1		1	
4										
5										
6	1		1		1		1		1	
7	1		1		1		1		1	
8	1		1		1		1		1	
9	1		1		1		1		1	
10	1		1		1		1		1	

2) $\bigstar C$

x/y	1	2	3	4	5	6	7	8	9	10
1	1	1	0	1	1	0	1	1	0	1
2	1	1	0	1	1	0	1	1	0	1
3	0	0	0	0	0	0	0	0	0	0
4	1	1	0	1	1	0	1	1	0	1
5	1	1	0	1	1	0	1	1	0	1
6	0	0	0	0	0	0	0	0	0	0
7	1	1	0	1	1	0	1	1	0	1
8	1	1	0	1	1	0	1	1	0	1
9	0	0	0	0	0	0	0	0	0	0
10	1	1	0	1	1	0	1	1	0	1

3) $II = I - \bigstar C$

x/y	1	2	3	4	5	6	7	8	9	10
1	0	0	1	0	0	0	0	0	1	0
2	0	0	1	0	0	0	0	0	1	0
3	1	0	1	0	1	0	1	0	1	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	1	0	1	0	1	0	1	0	1	0
7	0	0	1	0	0	0	0	0	1	0
8	0	0	1	0	0	0	0	0	1	0
9	1	0	1	0	1	0	1	0	1	0
10	0	0	1	0	0	0	0	0	1	0

4) IV = C^{-1}

x/y	1	2	3	4	5	6	7	8	9	10
1	0	0	1	0	0	1	0	0	1	0
2	0	0	1	0	0	1	0	0	1	0
3	1	1	1	1	1	1	1	1	1	1
4	0	0	1	0	0	1	0	0	1	0
5	0	0	1	0	0	1	0	0	1	0
6	1	1	1	1	1	1	1	1	1	1
7	0	0	1	0	0	1	0	0	1	0
8	0	0	1	0	0	1	0	0	1	0
9	1	1	1	1	1	1	1	1	1	1
10	0	0	1	0	0	1	0	0	1	0

5) III = II – IV

x/y	1	2	3	4	5	6	7	8	9	10
1	0	0	1	0	0	1	0	0	1	0
2	0	0	1	0	0	1	0	0	1	0
3	1	1	1	1	1	1	1	1	1	1
4	0	0	1	0	0	1	0	0	1	0
5	0	0	1	0	0	1	0	0	1	0
6	1	1	1	1	1	1	1	1	1	1
7	0	0	1	0	0	1	0	0	1	0
8	0	0	1	0	0	1	0	0	1	0
9	1	1	1	1	1	1	1	1	1	1
10	0	0	1	0	0	1	0	0	1	0

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).

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <malloc.h>

static bool  **I;

bool  IsVoid(int N, bool  **A) {
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      if (A[i][j])
```

```
      return false;
    }
  }

  return true;
}

bool IsUniversum(int N, bool **A) {
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      if (!A[i][j])
        return false;
    }
  }

  return true;
}

bool **CreateRelation(int sizeX, int sizeY) {
  bool **A = (bool **) malloc(sizeX * sizeof(bool *));
  for (int i = 0; i < sizeX; ++i) {
    A[i] = (bool *) calloc(sizeY, sizeof(bool));
  }

  return A;
}

void FillRelation(int N, bool **A, bool a[N][N]) {
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      A[i][j] = a[i][j];
    }
  }
}

bool AreRelationsEquals(int N, bool **A, bool **B) {
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      if (A[i][j] != B[i][j])
        return false;
    }
  }
  return true;
}

bool IsIncluding(int N, bool **A, bool **B) {
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      if (A[i][j] == 1 && A[i][j] != B[i][j])
        return false;
    }
  }
  return true;
}

bool IsStrictIncluding(int N, bool **A, bool **B) {
  return (!AreRelationsEquals(N, A, B) && IsIncluding(N, A, B));
}
```

```
bool  **NegativeRelation(int N, bool  **A) {
 bool  **B = CreateRelation(N, N);
 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   B[i][j] = !A[i][j];
  }
 }
 return B;
}

bool  **UnionRelations(int N, bool  **A, bool  **B) {
 bool  **C = CreateRelation(N, N);
 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   if (A[i][j] || B[i][j])
    C[i][j] = true;
  }
 }

 return C;
}

bool  **IntersectionRelations(int N, bool  **A, bool  **B) {
 bool  **C = CreateRelation(N, N);
 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   if (A[i][j] && B[i][j])
    C[i][j] = true;
  }
 }

 return C;
}

bool  **DifferenceRelations(int N, bool  **A, bool  **B) {
 bool  **C = CreateRelation(N, N);
 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   if (A[i][j] > B[i][j])
    C[i][j] = true;
  }
 }

 return C;
}

bool  **SymmetricalDifferenceRelations(int N, bool  **A, bool  **B) {
 return UnionRelations(N, DifferenceRelations(N, A, B),
                       DifferenceRelations(N, B, A));
}

bool  **ComplementRelations(int N, bool  **A) {
 bool  **C = CreateRelation(N, N);
 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   C[i][j] = !A[i][j];
  }
```

```
 }
 return C;
}

bool **OverturnRelation(int N, bool **A) {
 bool **C = CreateRelation(N, N);
 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   if (A[i][j])
    C[j][i] = true;
  }
 }

 return C;
}

bool **CompositionRelation(int N, bool **A, bool **B) {
 bool **C = CreateRelation(N, N);

 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   for (int k = 0; k < N; ++k) {
    if (A[i][k] && B[k][j])
     C[i][j] = true;
   }
  }
 }

 return C;
}

bool **PowRelation(int N, bool **A, int P) {
 if (P > 1)
  return CompositionRelation(N, A, PowRelation(N, A, P - 1));
 else
  return A;
}

int main() {
 I = CreateRelation(10, 10);

 for (int i = 0; i < 10; ++i) {
  I[i][i] = true;
 }

 bool **A = CreateRelation(10, 10);

 for (int i = 1; i < 11; ++i) {
  for (int j = 1; j < 11; ++j) {
   A[i - 1][j - 1] = (i < j && j < 9 - i) || (9 - i < j && j < i);
  }
 }



 bool **B = CreateRelation(10, 10);

 for (int i = 1; i < 11; ++i) {
```

```
  for (int j = 1; j < 10; ++j) {
    B[i - 1][j - 1] = i % 2 == 0 && j % 2 == 1;
  }
 }



 bool  **C = CreateRelation(10, 10);
 for (int i = 1; i < 11; ++i) {
  for (int j = 1; j < 11; ++j) {
    C[i - 1][j - 1] = (i * j) % 3 == 0;
  }
 }



 bool  **First = CompositionRelation(10, A, B);
 bool  **Second = NegativeRelation(10, C);
 bool  **Third = DifferenceRelations(10, First, Second);
 bool  **Fourth = OverturnRelation(10, C);
 bool  **D = UnionRelations(10, Third, Fourth);

 for (int i = 0; i < 10; ++i) {
  for (int j = 0; j < 10; ++j) {
    printf("%d ", D[i][j]);
  }
  printf("\n");
 }
}
```

```
0 0 1 0 0 1 0 0 1 0
0 0 1 0 0 1 0 0 1 0
1 1 1 1 1 1 1 1 1 1
0 0 1 0 0 1 0 0 1 0
0 0 1 0 0 1 0 0 1 0
1 1 1 1 1 1 1 1 1 1
0 0 1 0 0 1 0 0 1 0
0 0 1 0 0 1 0 0 1 0
1 1 1 1 1 1 1 1 1 1
0 0 1 0 0 1 0 0 1 0
```

Результат работы программы

Часть 2. Свойства отношений

2.1.Определить основные свойства отношений (см. "Варианты заданий п.а).

Основыне свойства	A	B	C
Рефлексивно	нет	нет	нет
Антирефлексивно	да	да	нет
Симметрично	нет	нет	да
Антисимметрично	нет	нет	нет
Транзитивно	нет	нет	да
Антитранзитивно	нет	да	нет
Полно	нет	нет	нет

A:

Отношение не является *рефлексивным*, так как на главной диагонали нет единиц (A[0][0] != 1);

Отношение является *антирефлексивным*, так как на главной диагонали стоят все нули;

Отношение не является *симметричным*, так как противоположные элементы относительно главной диагонали не равны (A[0][1] != A[1][0]);

Отношение не является *антисимметричным*, так как противоположные элементы относительно главной диагонали равны (A[0][2] == A[2][0]);

Отношение не является *транзитивным*, так как на графе присутствуют неполные “треугольники” (пути из x в z, z в y и x в y не соединены друг с другом каким-либо образом);

Отношение не является *антитранзитивным*, так как на графе присутствуют “треугольники” (пути из x в z, z в y и x в y соединены друг с другом);

Отношение не является *полным*, так как в матрице существуют такие элементы x, y (x ≠ y), что Ax,y = Ay,x = 0 (A[0][0] == 0);

B:

Отношение не является *рефлексивным*, так как на главной диагонали не все элементы единицы (A[0][0] != 1);

Отношение является *антирефлексивным*, так как на главной диагонали стоят все нули;

Отношение не является *симметричным*, так как противоположные элементы относительно главной диагонали не равны (A[0][1] != A[1][0]);

Отношение не является *антисимметричным*, так как противоположные элементы относительно главной диагонали равны (A[0][2] == A[2][0]);

Отношение не является *транзитивным*, так как на графе присутствуют неполные “треугольники” (пути из x в z, z в y и x в y не соединены друг с другом каким-либо образом);

Отношение не является *антитранзитивным*, так как на графе не присутствуют “треугольники” (пути из x в z, z в y и x в y соединены друг с другом);

Отношение не является *полным*, так как в матрице существуют такие элементы x, y (x ≠ y), что Ax,y = Ay,x = 0 (A[0][0] == 0);

C:

Отношение не является *рефлексивным*, так как на главной диагонали не все элементы единицы (A[0][0] != 1);

Отношение не является *антирефлексивным*, так как на главной диагонали не все элементы нули (A[2][2] != 1);

Отношение является *симметричным*, так как противоположные элементы относительно главной диагонали равны;

Отношение не является *антисимметричным*, так как противоположные элементы относительно главной диагонали равны (A[0][1] == A[1][0]);

Отношение является *транзитивным*, так как на графе для каждых 3х элементов существует “треугольник” (пути из x в z, z в y и x в y соединены друг с другом);

Отношение не является *антитранзитивным*, так как на графе присутствуют “треугольники” (пути из x в z, z в y и x в y соединены друг с другом);

Отношение не является *полным*, так как в матрице существуют такие элементы x, y (x ≠ y), что Ax,y = Ay,x = 0 (A[0][0] == 0);

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

A, B, C:

Отношения не являются отношениями толерантности, эквивалентности и порядка, так как не обладают необходимыми основными свойствами отношений.

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

Код программы:

```
bool  IsReflex(int N, bool  **A) {
  for (int i = 0; i < N; ++i)
    if (!A[i][i]) {
      printf("A[%d][%d] = 0 matrix is not reflex\n", i + 1, i + 1);
      return false;
    }
  return true;
}
```

```
bool IsAntiReflex(int N, bool **A) {
 for (int i = 0; i < N; ++i)
  if (A[i][i]) {
   printf("A[%d][%d] = 1 matrix is not anti reflex\n", i + 1, i + 1);
   return false;
  }
 return true;
}

bool IsSymmetric(int N, bool **A) {
 for (int i = 0; i < N; ++i)
  for (int j = i + 1; j < N; ++j)
   if (A[i][j] != A[j][i]) {
    printf("A[%d][%d] = %d A[%d][%d] = %d Matrix not symmetric\n", i + 1, j + 1, A[i][j], j + 1, i + 1,
        A[j][i]);
    return false;
   }

 return true;
}

bool IsAntiSymmetric(int N, bool **A) {
 for (int i = 0; i < N; ++i)
  for (int j = i + 1; j < N; ++j)
   if (A[i][j] && A[j][i]) {
    printf("A[%d][%d] = A[%d][%d] Matrix not antisymmetric\n", i + 1, j + 1, j + 1, i + 1);
    return false;
   }
 return true;
}

bool IsTranzitive(int N, bool **A) {
 for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
   for (int k = 0; k < N; ++k)
    if (A[i][k] && A[k][j] && !A[i][j]) {
     printf("A[%d][%d] = 1 A[%d][%d] = 1 A[%d][%d] = 0  Matrix not tranzitive\n", i + 1, k + 1, k + 1,
         j + 1, i + 1, j + 1);
     return false;
    }

 return true;
}

bool IsAntiTranzitive(int N, bool **A) {
 for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
   for (int k = 0; k < N; ++k)
    if (A[i][k] && A[k][j] && A[i][j]) {
     printf("A[%d][%d] = 1 A[%d][%d] = 1 A[%d][%d] = 1  Matrix not anti tranzitive\n", i + 1, k + 1,
         k + 1,
         j + 1, i + 1, j + 1);
     return false;
    }

 return true;
}
```

```
bool IsFull(int N, bool **A) {
 for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
   if (i != j) {
    if (!A[i][j])
     printf("A[%d][%d] = 0 Matrix not full\n", i + 1, j + 1);
    else if (!A[j][i])
     printf("A[%d][%d] = 0 Matrix not full\n", j + 1, i + 1);
    return false;
   }

 return true;
}

bool IsTolerant(int N, bool **A) {
 return IsReflex(N, A) && IsSymmetric(N, A);
}

bool IsEcvivalent(int N, bool **A) {
 return IsTolerant(N, A) && IsTranzitive(N, A);
}

bool IsOrder(int N, bool **A) {
 return IsAntiSymmetric(N, A) && IsTranzitive(N, A);
}

int main() {
 I = CreateRelation(10, 10);

 for (int i = 0; i < 10; ++i) {
  I[i][i] = true;
 }

 bool **A = CreateRelation(10, 10);
 for (int i = 1; i < 11; ++i) {
  for (int j = 1; j < 11; ++j) {
   A[i - 1][j - 1] = (i < j && j < 9 - i) || (9 - i < j && j < i);
  }
 }



 bool **B = CreateRelation(10, 10);
 for (int i = 1; i < 11; ++i) {
  for (int j = 1; j < 10; ++j) {
   B[i - 1][j - 1] = i % 2 == 0 && j % 2 == 1;
  }
 }



 bool **C = CreateRelation(10, 10);
 for (int i = 1; i < 11; ++i) {
  for (int j = 1; j < 11; ++j) {
   C[i - 1][j - 1] = (i * j) % 3 == 0;
  }
 }
```

```
 printf("is reflexive: %d, %d, %d\n", IsReflex(10, A), IsReflex(10, B), IsReflex(10, C));
 printf("is antireflexive: %d, %d, %d\n", IsAntiReflex(10, A), IsAntiReflex(10, B), IsAntiReflex(10, C));
 printf("is symmetric: %d, %d, %d\n", IsSymmetric(10, A), IsSymmetric(10, B), IsSymmetric(10, C));
 printf("is antisymmetric: %d, %d, %d\n", IsAntiSymmetric(10, A), IsAntiSymmetric(10, B), IsAntiSymmetric(10, C));
 printf("is transitive: %d, %d, %d\n", IsTranzitive(10, A), IsTranzitive(10, B), IsTranzitive(10, C));
 printf("is antitranzitive: %d, %d, %d\n", IsAntiTranzitive(10, A), IsAntiTranzitive(10, B),
       IsAntiTranzitive(10, C));
 printf("is full: %d, %d, %d\n", IsFull(10, A), IsFull(10, B), IsFull(10, C));
 printf("is tolerant: %d, %d, %d\n", IsTolerant(10, A), IsTolerant(10, B), IsTolerant(10, C));
 printf("is ecvivalent: %d, %d, %d\n", IsEcvivalent(10, A), IsEcvivalent(10, B), IsEcvivalent(10, C));
 printf("is order: %d, %d, %d\n", IsOrder(10, A), IsOrder(10, B), IsOrder(10, C));
}
```

Результат работы программы:

```
A[1][1] = 0 matrix is not reflex
A[1][1] = 0 matrix is not reflex
A[1][1] = 0 matrix is not reflex
is reflexive: 0, 0, 0
A[3][3] = 1 matrix is not anti reflex
is antireflexive: 1, 1, 0
A[1][2] = 0 A[2][1] = 1 Matrix not symmetric
A[1][2] = 1 A[2][1] = 0 Matrix not symmetric
is symmetric: 0, 0, 1
A[1][3] = A[3][1] Matrix not antisymmetric
is antisymmetric: 1, 1, 0
A[1][3] = 1 A[3][1] = 1 A[1][1] = 0  Matrix not tranzitive
is transitive: 1, 1, 0
A[1][3] = 1 A[3][3] = 1 A[1][3] = 1  Matrix not anti tranzitive
A[1][2] = 1 A[2][3] = 1 A[1][3] = 1  Matrix not anti tranzitive
is antitranzitive: 0, 1, 0
A[1][2] = 0 Matrix not full
A[1][2] = 0 Matrix not full
A[2][1] = 0 Matrix not full
is full: 0, 0, 0
A[1][1] = 0 matrix is not reflex
A[1][1] = 0 matrix is not reflex
A[1][1] = 0 matrix is not reflex
is tolerant: 0, 0, 0
A[1][1] = 0 matrix is not reflex
A[1][1] = 0 matrix is not reflex
A[1][1] = 0 matrix is not reflex
is ecvivalent: 0, 0, 0
A[1][3] = A[3][1] Matrix not antisymmetric
is order: 1, 1, 0
```

Вывод: в ходе работы были изучены способы задания отношений, операции над отношениями и свойства отношений, освоена программная реализация операций и определение свойств отношений.