***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №2**

**По дисциплине Дискретная математика за II семестр**

**Тема: «Отношения»**

**Выполнил:**

Студент группы ИИ-15 (1)

1-го курса

Волк И. А.

**Проверил:**

Глущенко Т.А.

Брест 2018

Цель: познакомиться с отношениями.

На множестве *A = {1,2,3,4,5,6,7,8,9,10}* заданы отношения  и  согласно вашему варианту. Варианты заданий указаны в *таблице 1*.

1. Для заданных отношений составить матрицы отношений. Построить орграфы отношений.
2. Определить область определения и множество значений отношений.
3. Найти обратные отношения и дополнения отношений.
4. Указать свойства отношений.
5. Для отношения, не обладающего свойством транзитивности, построить транзитивное замыкание алгоритмом *Флойда-Уоршолла*.
6. Найти композицию  и  отношений, определить обладает ли операция композиции отношений свойством коммутативности.
7. Задание *4* выполнить вручную, для остальных – программная реализация.

|  |  |  |  |
| --- | --- | --- | --- |
| *7.* | *- четное число* |  | *S = (1,8),(2,6),(3,8),(4,6)* |

Код программы:

#include "stdafx.h"

#include <Windows.h>

#include <iostream>

using namespace std;

const int notName = system("color f0");

/\*\*\*\* Additional functions \*\*\*\*/

bool R1(int, int); // Relation #1

bool R2(int, int); // Relation #2

bool R3(int, int); // Full

bool composition(int, int, int, bool(\*)(int, int), bool(\*)(int, int)); // Composition of relations

bool(\*R[])(int, int) = {R1, R2, R3}; // Array of relations

int \* domainChVec(int\*, int, int\*\*); // Characteristic vector of domain

int \* setOfValuesChVec(int\*, int, int\*\*); // Characteristic vector of set of values

void matrixTranspose(int\*\*, int); // Transposing the matrix

int \*\* relationSet(int\*, int, int\*\*, int\*); // Outputting of relation through matrix

int sizeOfRelationSet(int\*\*, int); // size of relationset

bool isTransitive(int\*\*, int); // is set transitive

bool matrixComparison(int\*\*, int\*\*, int, int); // matrix comparison

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\* Main functions \*\*\*\*/

// Matrix of relation

int \*\* matrixOfRelation(int\*, int, bool(\*)(int, int));

// For outputing of matrix

void matrixOut(int\*\*, int, int);

/\* Domain and set of values\*/

void domainAndSetOfValues(int\*, int, int\*\*); // array, size, matrix

void domainAndSetOfValues(int\*,int,int\*\*,int\*,int\*,int\*,int\*); // array, size of array, matrix,

// domain, size of domain,

// set of values, size of set of values

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Outputting inver relation

int \*\* inverseRelationSet(int\*, int, int\*\*, int\*);

void relationSetOutput(int\*\*, int, int);

// Addition of the set

int \*\* matrixOfAddition(int\*\*, int\*\*, int);

// Transitive closure

int \*\* transitiveClosure(int\*\*, int);

// Composition

int \*\* matrixOfComposition(int\*, int, bool(\*)(int, int), bool(\*)(int, int));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

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

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

/\* Matrixes of relations \*/

int \*\*\* matrix = new int\*\*[3];

for (int i = 0; i < 2; i++)

{

cout << "R" << i + 1 << ": \n";

matrix[i] = matrixOfRelation(A, sizeA, R[i]);

matrixOut(matrix[i], sizeA, sizeA);

cout << endl;

}

matrix[2] = matrixOfRelation(A, sizeA, R[2]); // Full

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Domain and set of values \*/

cout << "Version 1" << endl;

int \*\*domain = new int\*[2], \*\*setOfValues = new int\*[2], \*sizeOfDomain = new int[2], \*sizeOfSetOfValues = new int[2];

for (int i = 0; i < 2; i++)

{

setOfValues[i] = new int[sizeA];

domain[i] = new int[sizeA];

domainAndSetOfValues(A, sizeA, matrix[i], domain[i], &sizeOfDomain[i], setOfValues[i], &sizeOfSetOfValues[i]);

cout << "R" << i + 1 << " domain: ";

for (int j = 0; j < sizeOfDomain[i]; j++)

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

cout << "\nR" << i + 1 << " set of values: ";

for (int j = 0; j < sizeOfSetOfValues[i]; j++)

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

cout << endl;

}

cout << "\nVersion 2" << endl;

for (int i = 0; i < 2; i++)

domainAndSetOfValues(A, sizeA, matrix[i]);

cout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Outputting of inverse relations \*/

int \*sizeOfRelationSet = new int[2];

int \*\*\* inverseRelSet = new int\*\*[2];

for (int i = 0; i < 2; i++)

{

cout << "Inverse relation " << i + 1 << ": ";

inverseRelSet[i] = inverseRelationSet(A, sizeA, matrix[i], &sizeOfRelationSet[i]);

relationSetOutput(inverseRelSet[i], sizeOfRelationSet[i], 2);

cout << endl;

}

cout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Addition of the set \*/

int \*\*\* matrixOfAdd = new int\*\*[2];

int \*\*\* additionSet = new int\*\*[2], \*sizeOfAdditionSet = new int[2];

for (int i = 0; i < 2; i++)

{

cout << "Addition of the R" << i + 1 << ": ";

matrixOfAdd[i] = matrixOfAddition(matrix[i], matrix[2], sizeA);

additionSet[i] = relationSet(A, sizeA, matrixOfAdd[i], &sizeOfAdditionSet[i]);

relationSetOutput(additionSet[i], sizeOfAdditionSet[i], 2);

cout << endl;

}

cout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Transitive closure \*/

for (int i = 0; i < 2; i++)

if (!isTransitive(matrix[i], sizeA))

{

int \*\* tMatrix = transitiveClosure(matrix[i], sizeA);

cout << "Transitive closure of R" << i + 1 << endl;

matrixOut(tMatrix, sizeA, sizeA);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Composition \*/

int \*\* matrixOfComp[2] = { matrixOfComposition(A, sizeA, R1, R2), matrixOfComposition(A, sizeA, R2, R1) };

for (int i = 0; i < 2; i++)

{

cout << "R" << ((i) ? 2 : 1) << "\*" << "R" <<((i) ? 1 : 2) << ": " << endl;

matrixOut(matrixOfComp[i], sizeA, sizeA);

}

cout << endl;

bool isMatrixEqual = matrixComparison(matrixOfComp[0], matrixOfComp[1], sizeA, sizeA);

if (isMatrixEqual) cout << "The operation of composition is commutative." << endl;

else cout << "The operation of composition is not commutative." << endl << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

system("pause");

return 0;

}

/\*\* Matrix \*\*/

int \*\* matrixOfRelation(int\* arr, int size, bool (\*func)(int, int))

{

int \*\* resultArr = new int\*[size]; // array (matrix) of relation

// first consists of 0

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

{

resultArr[i] = new int[size];

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

resultArr[i][j] = 0;

}

// 1 if true

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

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

if ((\*func)(arr[i], arr[j]))

resultArr[i][j] = 1;

return resultArr;

}

void matrixOut(int \*\* arr, int size\_i, int size\_j)

{

for (int i = 0; i < size\_i; i++)

{

for (int j = 0; j < size\_j; j++)

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

cout << endl;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* Domain and set of values \*\*/

//output

void domainAndSetOfValues(int\* arr, int size, int\*\*matrix)

{

cout << "Domain: ";

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

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

if (matrix[i][j])

{

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

break;

}

cout << endl;

cout << "Set of values: ";

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

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

if (matrix[i][j])

{

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

break;

}

cout << endl;

}

// without output; find domain and set of values

void domainAndSetOfValues (int\* arr, int size, int\*\*matrix,

int\* domain, int\* sizeOfDomain,

int\* setOfValues, int\* sizeOfSetOfValues)

{

int \* domChVec = domainChVec(arr, size, matrix); // characteristic vector of domain

int \* setOfValChVec = setOfValuesChVec(arr, size, matrix); // characteristic vector of set of values

int sizeOfDom = 0, sizeOfSetOfVal = 0;

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

{

if (domChVec[i])

{

domain[sizeOfDom] = arr[i];

sizeOfDom++;

}

if (setOfValChVec[i])

{

setOfValues[sizeOfSetOfVal] = arr[i];

sizeOfSetOfVal++;

}

}

\*sizeOfDomain = sizeOfDom; \*sizeOfSetOfValues = sizeOfSetOfVal;

}

int \* domainChVec(int\* arr, int size, int\*\* matrix)

{

int \* chVec = new int[size]; // characteristic vector of domain

// first consists of 0

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

chVec[i] = 0;

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

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

if (matrix[i][j])

{

chVec[i] = 1;

break;

}

return chVec;

}

int \* setOfValuesChVec(int\* arr, int size, int\*\* matrix)

{

int \* chVec = new int[size]; // characteristic vector of domain

// first consists of 0

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

chVec[i] = 0;

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

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

if (matrix[i][j])

{

chVec[j] = 1;

break;

}

return chVec;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* Outputting of inverse relations \*\*/

int \*\* inverseRelationSet(int\* arr, int size, int\*\*matrix, int \*sizeOfRelSet)

{

matrixTranspose(matrix, size);

int \*\* rel = relationSet(arr, size, matrix, sizeOfRelSet);

matrixTranspose(matrix, size);

return rel;

}

void matrixTranspose(int\*\* matrix, int size)

{

int \*\* transposedMatrix = new int\*[size];

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

{

transposedMatrix[i] = new int[size];

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

transposedMatrix[i][j] = matrix[j][i];

}

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

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

matrix[i][j] = transposedMatrix[i][j];

}

int \*\* relationSet(int\* arr, int size, int\*\*matrix, int \*sizeOfRelSet)

{

int sizeOfRelSetLocal = \*sizeOfRelSet = sizeOfRelationSet(matrix, size);

int \*\* relSet = new int\*[sizeOfRelSetLocal];

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

relSet[i] = new int[2];

for(int i = 0, r = 0; i < size; i++)

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

if (matrix[i][j])

{

relSet[r][0] = arr[i];

relSet[r][1] = arr[j];

r++;

}

return relSet;

}

int sizeOfRelationSet(int\*\* matrix, int size)

{

int sizeOfRel = 0;

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

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

if (matrix[i][j])

sizeOfRel++;

return sizeOfRel;

}

void relationSetOutput(int\*\*relation, int sizeOfRelSet, int numOfEl)

{

cout << "{";

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

{

if (i) cout << ", ";

cout << "(";

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

{

if (j) cout << ", ";

cout << relation[i][j];

}

cout << ")";

}

cout << "}";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* Addition of the set \*\*/

int \*\* matrixOfAddition(int\*\*matrix\_1, int\*\*matrix\_2, int size)

{

int \*\* matrixOfAdd = new int\*[size];

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

{

matrixOfAdd[i] = new int[size];

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

{

if (matrix\_1[i][j] == matrix\_2[i][j])

matrixOfAdd[i][j] = 0;

else

matrixOfAdd[i][j] = matrix\_2[i][j];

}

}

return matrixOfAdd;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* Transitive closure \*\*/

int \*\* transitiveClosure(int \*\* matrix, int size)

{

int \*\* matrixRes = new int\*[size];

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

matrixRes[i] = new int[size];

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

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

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

matrixRes[i][j] = matrix[i][j] || matrix[i][k] && matrix[k][j];

return matrixRes;

}

bool isTransitive(int \*\* matrix, int size)

{

int \*\* tMatrix = transitiveClosure(matrix, size);

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

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

if (tMatrix[i][j] != matrix[i][j])

return false;

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* Composition \*\*/

int \*\* matrixOfComposition(int\* arr, int size, bool(\*r1)(int, int), bool(\*r2)(int, int))

{

int \*\* resultArr = new int\*[size]; // array (matrix) of relation // first consists of 0

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

{

resultArr[i] = new int[size];

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

resultArr[i][j] = 0;

}

// 1 if true

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

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

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

if (composition(arr[i], arr[k], arr[j], r1, r2))

resultArr[i][j] = 1;

return resultArr;

}

bool matrixComparison(int\*\* arr1, int\*\* arr2, int size\_i, int size\_j)

{

for (int i = 0; i < size\_i; i++)

for (int j = 0; j < size\_j; j++)

if (!(arr1[i][j] && arr2[i][j]))

return false;

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Composition \*/

bool composition(int a, int b, int c, bool(\*r1)(int, int), bool(\*r2)(int, int))

{

if((r2)(a, b))

return (\*r1)(b, c);

return false;

}

/\* Relation #1 \*/

bool R1(int a, int b)

{

return ((a - b) % 2 == 0);

}

/\* Relation #2 \*/

bool R2(int a, int b)

{

return (b == (a \* a));

}

/\* Full \*/

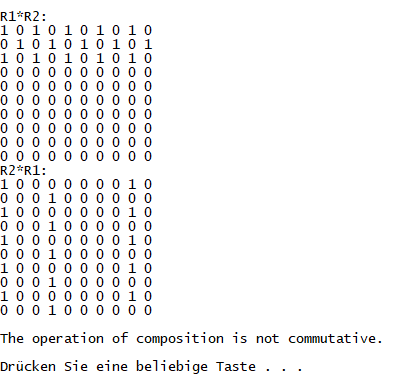
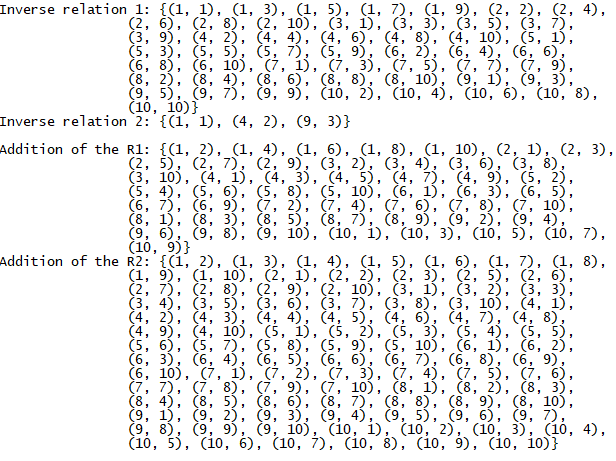
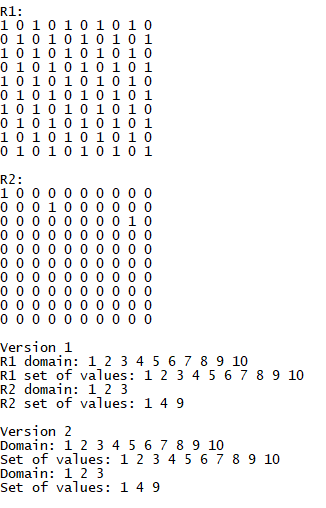
bool R3(int a, int b)

{

return true;

}

Output:



Вывод: познакомился с отношениями.