Министерство образования Республики Беларусь

Учреждение образования

Брестский государственный технический университет

Кафедра ИИТ

Лабораторная работа №2

За 3 семестр

По дисциплине «Дискретная математика»

Тема: «Отношения. Функции»

Выполнил: студент 2 курса

Группы ПО-4(2)

Коташевич С.Н.

Проверил: Глущенко Т.А.

Брест 2020

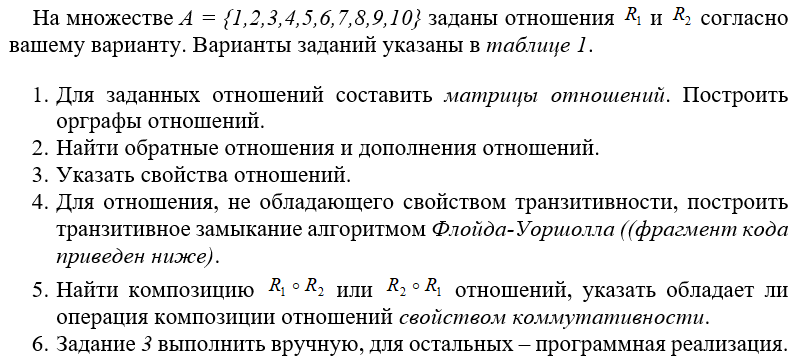
Лабораторная работа №2

Отношения. Функция

Вариант 13

Отношения.

**Задание 1.**

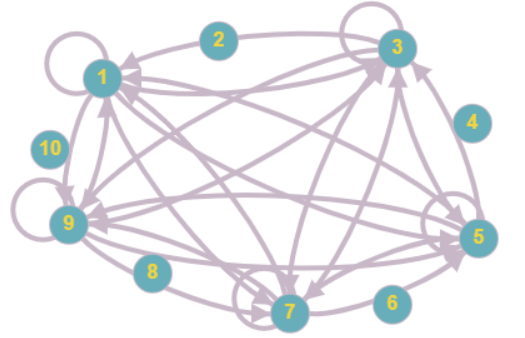
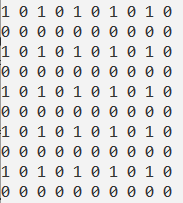




1.

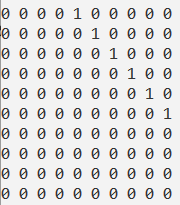
R1

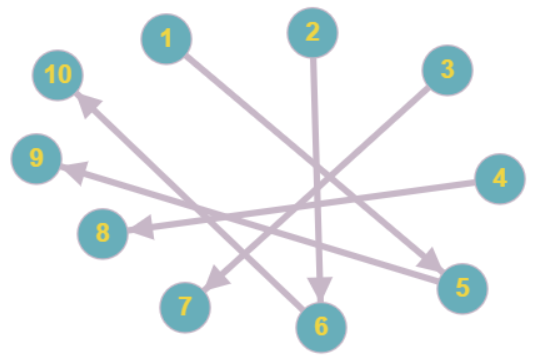




R2



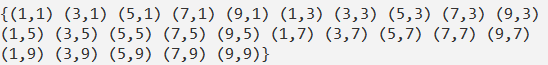


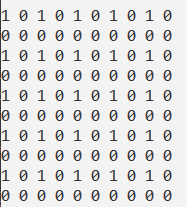


2.

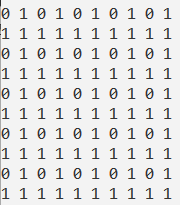
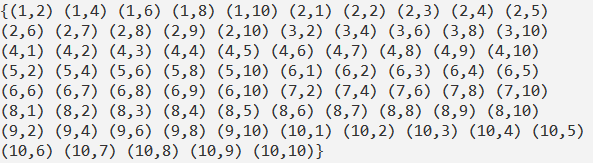
R1

Обратное отношение:





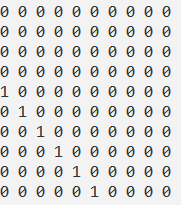
Дополнение отношения:



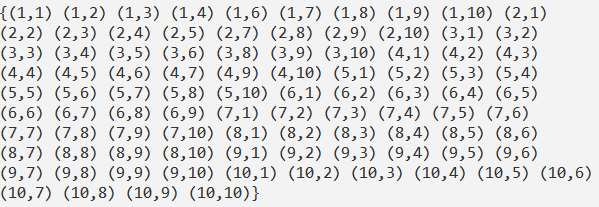
R2

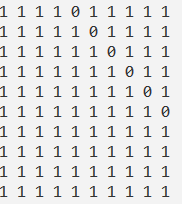
Обратное отношение:





Дополнение отношения:





3.

R1:

1) рефлексивность -

2) антисимметричность -

3) симметричность +

4) антисимметричность -

5) транзитивность +

R2:

1) рефлексивность -

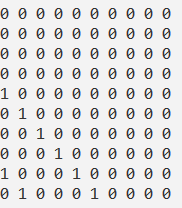
2) антисимметричность +

3) симметричность -

4) антисимметричность -

5) транзитивность -

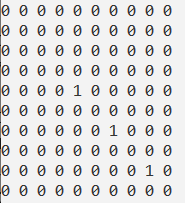
4.



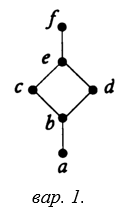
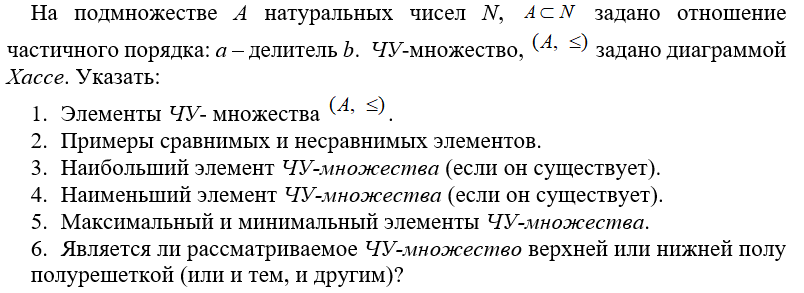
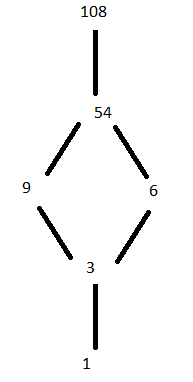
5.

R1 o R2





Задание 2.

1. 1, 3, 6, 9, 54, 108

2. сравнимые: 9 и 3, 54 и 3, 108 и 6

несравнимые: 9 и 6

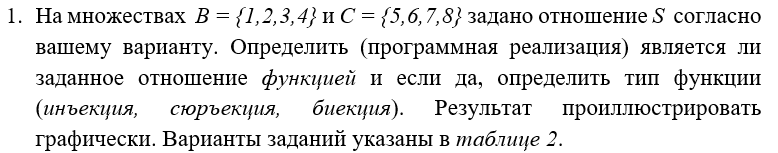
3. 24

4. 1

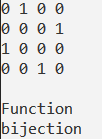
5. 108 и 1

6. одновременно верхняя и нижняя полурешётка

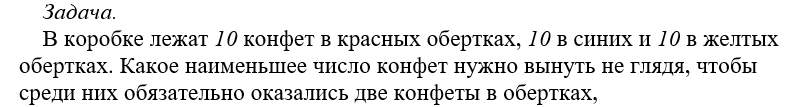
**Функции.**  
Задание 1.







Задание 2



11 конфет

**Текст программы:**

#include <iostream>

using namespace std;

int\*\* a\_b(int\* A, int size\_A, int& size\_R1) //отношение R1

{

int num\_col = 2;

int\*\* R1 = new int\* [size\_A \* size\_A];

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

R1[i] = new int[num\_col];

}

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

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

if ((A[i] \* A[j]) % 2 == 1) {

R1[size\_R1][0] = A[i];

R1[size\_R1][1] = A[j];

size\_R1++;

}

}

}

return R1;

}

int\*\* b\_a(int\* A, int size\_A, int& size\_R2) //отношение R2

{

int num\_col = 2;

int\*\* R2 = new int\* [size\_A \* size\_A];

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

R2[i] = new int[num\_col];

}

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

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

if ((A[i] + 4) == A[j]) {

R2[size\_R2][0] = A[i];

R2[size\_R2][1] = A[j];

size\_R2++;

}

}

}

return R2;

}

void show(int\*\* R, int size) //вывод отношения в консоль

{

int count = 0;

cout << "{";

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

cout << "(";

for (int j = 0; j < 2; j++) {

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

}

cout << "\b) ";

count++;

if (count == 10) {

count = 0;

cout << endl;

}

}

cout << "\b}" << endl;

}

int\*\* matrix(int\*\* R, int size\_A, int size\_R) //построение матрицы отношений

{

int\*\* matrix = new int\* [size\_A];

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

matrix[i] = new int[size\_A];

}

bool is = false;

cout << endl;

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

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

for (int c = 0; c < size\_R; c++) {

if (R[c][0] == i + 1 && R[c][1] == j + 1) {

is = true;

break;

}

else {

is = false;

}

}

if (is == true){

matrix[i][j] = 1;

}

else {

matrix[i][j] = 0;

}

}

}

return matrix;

}

void show\_matrix(int\*\* matrix, int size\_A) //вывод матрицы в консоль

{

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

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

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

}

cout << endl;

}

}

int\*\* R\_(int\*\* R, int size) //обратное отношение

{

int\*\* R\_1 = new int\* [size];

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

R\_1[i] = new int[2];

}

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

R\_1[i][0] = R[i][0];

R\_1[i][1] = R[i][1];

}

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

swap(R\_1[i][0], R\_1[i][1]);

}

return R\_1;

}

int\*\* Addition1(int\*\* R1, int size\_R1, int\* A, int size\_A, int& size\_Ra) //дополнение отношения

{

int\*\* R\_a1 = new int\* [size\_A \* size\_A];

bool is\_of = false;

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

R\_a1[i] = new int[2];

}

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

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

is\_of = false;

for (int c = 0; c < size\_R1; c++) {

if ((A[i] != R1[c][0] && A[j] != R1[c][1]) || (A[i] == R1[c][0] && A[j] != R1[c][1]) || (A[i] != R1[c][0] && A[j] == R1[c][1])) {

continue;

}

else if (A[i] == R1[c][0] && A[j] == R1[c][1]) {

is\_of = true;

break;

}

}

if (is\_of == false) {

R\_a1[size\_Ra][0] = A[i];

R\_a1[size\_Ra][1] = A[j];

size\_Ra++;

}

}

}

return R\_a1;

}

int\*\* Fl\_Yo(int\*\* matrix\_of) //алгоритм Флойда-Уоршолла

{

for (int k = 0; k < 10; k++) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

matrix\_of[i][j] = matrix\_of[i][j]

|| matrix\_of[i][k]

&& matrix\_of[k][j];

}

}

}

return matrix\_of;

}

int\*\* del\_el(int\*\* R, int size, int index) //"удаление" элемента по индексу

{

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

R[i] = R[i + 1];

}

return R;

}

int\*\* Compos\_R1oR2(int\*\* R1, int\*\* R2, int size\_R1, int size\_R2, int& size\_C) //Находит композицию двух отношений

{

int\*\* Comp = new int\* [size\_R1 \* size\_R2];

for (int i = 0; i < size\_R1 \* size\_R2; i++) {

Comp[i] = new int[2]{0};

}

for (int i = 0; i < size\_R1; i++) {

for (int j = 0; j < size\_R2; j++) {

if (R2[j][1] == R1[i][0]) {

Comp[size\_C][0] = R1[i][0];

Comp[size\_C][1] = R2[j][1];

size\_C++;

}

}

}

//поиск и "удаление" повторяющихся элементов

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

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

if (\*Comp[i] == \*Comp[j]) {

Comp = del\_el(Comp, size\_C, j);

size\_C--;

j--;

}

}

}

return Comp;

}

void type\_of\_function(int\* C, int\* B, int S[][2], int size\_C, int size\_B, int size\_S) //определяет тип функции

{

bool injection = true;

bool surjection = true;

bool bijection = true;

int\* size\_of\_row = new int[size\_C] {0};

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

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

for (int k = 0; k < size\_S; k++) {

if (B[j] == S[k][0] && C[i] == S[k][1]) {

size\_of\_row[i]++;

break;

}

}

}

}

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

if (size\_of\_row[i] > 1) {

injection = false;

}

if (size\_of\_row[i] == 0) {

surjection = false;

}

}

if (injection == true && surjection == true) {

cout << "bijection\n";

}

else {

if (injection == true) {

cout << "injection\n";

}

else {

cout << "ne injection\n";

}

if (surjection == true) {

cout << "surjection\n";

}

else {

cout << "ne surjection\n";

}

}

}

void is\_functin(int\* B, int\* C, int S[][2], int size\_B, int size\_S, int size\_C) //является ли отношение функцией

{

bool is\_func = true;

int\* size\_of\_line = new int[size\_B] {0};

int\*\* matrix = new int\* [size\_B];

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

matrix[i] = new int[size\_C];

}

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

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

for (int k = 0; k < size\_S; k++) {

if (B[i] == S[k][0] && C[j] == S[k][1]) {

matrix[i][j] = 1;

size\_of\_line[i]++;

break;

}

else {

matrix[i][j] = 0;

}

}

}

}

show\_matrix(matrix, 4);

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

if (size\_of\_line[i] != 1) {

is\_func = false;

break;

}

}

if (is\_func == true) {

cout << "\nFunction\n";

type\_of\_function(C, B, S, size\_C, size\_B, size\_S);

}

else {

cout << "\nne function";

}

}

int main()

{

//Task 1

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

int size\_A = sizeof(A) / sizeof(int);

int size\_R1 = 0, size\_R2 = 0, size\_Ra1 = 0, size\_Ra2 = 0, size\_C = 0;

int\*\* R1 = a\_b(A, size\_A, size\_R1);

int\*\* R2 = b\_a(A, size\_A, size\_R2);

int\*\* R\_1 = R\_(R1, size\_R1);

int\*\* R\_2 = R\_(R2, size\_R2);

int\*\* R\_a1 = Addition1(R1, size\_R1, A, size\_A, size\_Ra1);

int\*\* R\_a2 = Addition1(R2, size\_R2, A, size\_A, size\_Ra2);

cout << endl;

show(R1, size\_R1);

cout << endl;

show(R2, size\_R2);

show(R\_1, size\_R1);

cout << endl;

show(R\_2, size\_R2);

cout << endl;

show(R\_a1, size\_Ra1);

cout << endl;

show(R\_a2, size\_Ra2);

int\*\* matrix1 = matrix(R\_1, size\_A, size\_R1);

int\*\* matrix2 = matrix(R\_2, size\_A, size\_R2);

int\*\* matrix3 = matrix(R\_a1, size\_A, size\_Ra1);

int\*\* matrix4 = matrix(R\_a2, size\_A, size\_Ra2);

cout << endl;

show\_matrix(matrix1, size\_A);

cout << endl;

show\_matrix(matrix2, size\_A);

cout << endl;

show\_matrix(matrix3, size\_A);

show\_matrix(matrix3, size\_A);

cout << endl;

show\_matrix(matrix4, size\_A);

int\*\* Floid\_Yorshal = Fl\_Yo(matrix2); //or matrix2

int\*\* R1oR2 = Compos\_R1oR2(R1, R2, size\_R1, size\_R2, size\_C);

show(R1oR2, size\_C);

int\*\* matrix5 = matrix(R1oR2, size\_A, size\_C);

show\_matrix(matrix5, size\_A);

cout << endl;

show\_matrix(Floid\_Yorshal, size\_A);

cout << endl;

//Task 2

int B[] = { 1, 2, 3, 4 };

int C[] = { 5, 6, 7, 8 };

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

int size\_B = sizeof(B) / sizeof(int);

int size\_C = sizeof(C) / sizeof(int);

int size\_S = sizeof(S) / (sizeof(int) \* 2);

is\_functin(B, C, S, size\_B, size\_S, size\_C);

return 0;

}

Вывод: в ходе работы были изучены отношения и функции.