МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
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 №1.4
по дисциплине: «Дискретная математика»
по теме: «**Теоретико-множественные уравнения**»

Выполнил/а: ст. группы ПВ-231
Чупахина София Александровна
Проверили:
Островский Алексей Мичеславович
Рязанов Юрий Дмитриевич

Белгород, 2024

Цель работы:

Научиться решать теоретико-множественные уравнения с применением ЭВМ.

Вариант 9

Дано:

Исходное уравнение:

$(A \Delta X) \cap (B \cup X) - C = \overline{A \cup X} \cup (C \Delta X)$

$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

$A = \{1, 3, 5, 6, 8\}$

$B = \{2, 4, 6, 9\}$

$C = \{2, 6, 7, 10\}$

X — ?

Задания:

1. Преобразовать исходное уравнение в уравнение с пустой правой частью.

2. Преобразовать левую часть уравнения к виду $\overline{X} \cap \varphi^{\varnothing} \cup X \cap \varphi^{U}$ используя разложение Шеннона по неизвестному множеству X.

3. Написать программу, вычисляющую значения множеств $\varphi^{\varnothing}$ и $\overline{\varphi^{U}}$ при заданных исходных множествах.

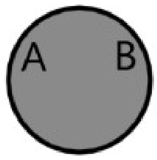
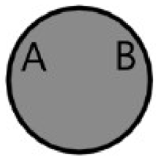
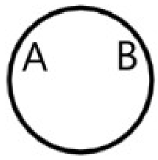
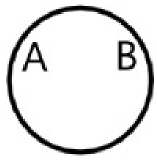
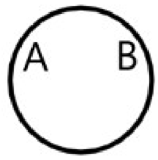
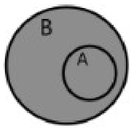
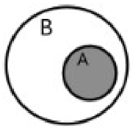
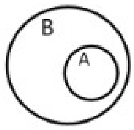
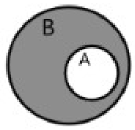
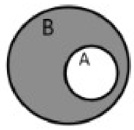
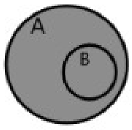
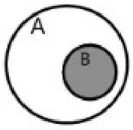
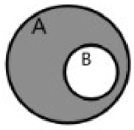
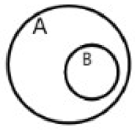
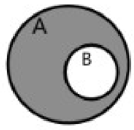
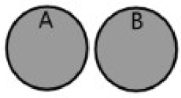
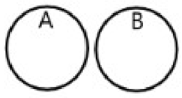
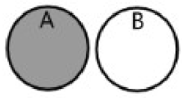
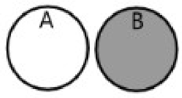
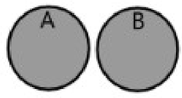
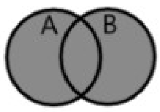
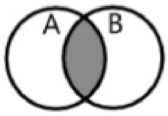
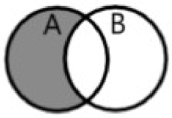
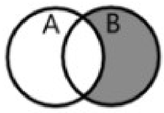
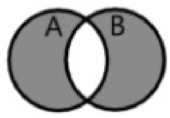
4. Вычислить значения множеств $\varphi^{\varnothing}$ и $\overline{\varphi^{U}}$, сделать вывод о существовании решения уравнения. Если решения уравнения не существует, то выполнить п.п. 1—4 для следующего (предыдущего) варианта.

5. Определить мощность общего решения, найти некоторые (или все) частные решения, в том числе частные решения наименьшей и наибольшей мощности.

6. Написать программу для проверки найденных решений

Задание 1:

Если мы хотим преобразовать уравнение так, чтобы в правой части его было пустое множество, мы должны перенести выражение справа в левую часть уравнения, связав их такой операцией, которая бы возвращала пустое множество, когда операнды равны, и непустое множество — во всех остальных случаях. Какая это будет операция? Для наглядности вставим сюда таблицу, отображающую результаты выполнения базовых операций над множествами в разных взаимных расположениях.

	A ∪ B	A ∩ B	A – B	B – A	A ∆ B
A и B равны					
A строго включено в B					
B строго включено в A					
A и B не пересекаются					
A и B находятся в общем положении					

Как видим, если множества равны, пустое множество будет возвращать операция разности (с любым порядком операндов) и симметрической разности. Однако операция разности будет возвращать пустое множество и в том случае, если множества не равны, но «уменьшаемое» множество является подмножеством «вычитаемого». А вот симметрическая разность, если множества не равны, возвращает непустое множество. Так что результат переноса правой части уравнения в левую будет следующим: $((A \,\Delta\, X) \cap (B \cup X) - C) \,\Delta\, (\overline{A \cup X} \cup (C \,\Delta\, X)) = \varnothing$.

Задание 2.

В общем виде разложение Шеннона производится следующим образом: если F(A1, A2, …, An) – некоторое выражение, в котором используются первичные термы A1, A2, …, An, то его можно преобразовать в вид $\overline{A1} \cap F(\varnothing, A2, \ldots, An) \cup A1 \cap F(U, A2, \ldots, An)$. Точно так же произвести разложение можно по любому множеству от A2 до An.

Мы же произведем разложение Шеннона над левой частью уравнения, полученного в пункте 1, по множеству X. В первом пересечении дополнение множества X пересекается с исходным выражением, где вместо X подставлено пустое множество; во втором — само множество X пересекается с исходным выражением, где вместо X подставлен универсум. Получим:

$\overline{X} \cap ((A \Delta \varnothing) \cap (B \cup \varnothing) - C) \Delta (\overline{A \cup \varnothing} \cup (C \Delta \varnothing)) \cup X \cap ((A \Delta U) \cap (B \cup U) - C) \Delta (\overline{A \cup U} \cup (C \Delta U))$

Выражение $((A \Delta \varnothing) \cap (B \cup \varnothing) - C) \Delta (\overline{A \cup \varnothing} \cup (C \Delta \varnothing))$ в дальнейшем и будет обозначаться как $\varphi^{\varnothing}$, а выражение $((A \Delta U) \cap (B \cup U) - C) \Delta (\overline{A \cup U} \cup (C \Delta U))$ — как φ^{U}. Так как значения множеств A, B, C известны, а X в этих выражениях не используется, их значения можно вычислить.

Задание 3.

Какие условия должны выполняться, чтобы выражение $\overline{X} \cap \varphi^{\varnothing} \cup X \cap \varphi^{U}$ было равно пустому множеству? Это выражение состоит из объединения двух частей, и чтобы это объединение было пустым множеством, каждая его часть должна быть пустым множеством.

Итак, когда пересечение $\overline{X} \cap \varphi^{\varnothing}$ будет равно пустому множеству? Операцию разности, $A - B$ можно переписать в виде $A \cap \overline{B}$, значит, и пересечение $\overline{X} \cap \varphi^{\varnothing}$ можно переписать как $\varphi^{\varnothing} - X$. А разность $\varphi^{\varnothing} - X$ будет равна пустому множеству, если в множестве $\varphi^{\varnothing}$ нет таких элементов, которые бы не входили в множество X, то есть $\varphi^{\varnothing}$ нестрого включено в X. Это условие прослеживается и в таблице в пункте 1, если просмотреть колонку «A – B».

По этой же таблице, просмотрев колонку «$A \cap B$», мы можем увидеть, что результатом пересечения будет пустое множество только в том случае, если множества не пересекаются. Но если ни один элемент множества A не находится в множестве B, то значит, все его элементы находятся в множестве $\overline{B}$. Значит, можно говорить о том, что другое обязательное условие - множество X нестрого включено в множество $\overline{\varphi^{U}}$. Таким образом, для нахождения всех возможных множеств X, являющихся корнями уравнения, необходимо найти множества $\varphi^{\varnothing}$ и $\overline{\varphi^{U}}$. Напишем для этого программу на основе библиотеки ordered_array_set, написанной в рамках курса ОП. Следующим образом будет выглядеть ее заголовочный файл, **ordered_array_set.h**:

```c
#ifndef INC_ORDERED_ARRAY_SET_H
#define INC_ORDERED_ARRAY_SET_H

#include <stdint.h>
#include <assert.h>
#include <memory.h>
#include <malloc.h>
#include <stdio.h>
#include <stdbool.h>
```

```c
typedef struct ordered_array_set {
    int *data; //элементы множества
    size_t size; //количество элементов в множестве
    size_t capacity; //максимальное количество элементов в множестве
} ordered_array_set;

//возвращает пустое множество, в которое можно вставить capacity
элементов
ordered_array_set ordered_array_set_create (size_t capacity);

//возвращает множество, состоящее из элементов массива a размера
size
ordered_array_set ordered_array_set_create_from_array (const int *a,
size_t size);

//Уменьшает емкость capacity множества по адресу a до его размера
size
//При этом перераспределяет память, отведенную под массив value,
чтобы он занимал минимальное ее количество
static void ordered_array_set_shrinkToFit (ordered_array_set *a);

//Увеличивает емкость capacity множества по адресу a на slots единиц
static void ordered_array_set_increaseCapacity (ordered_array_set
*a, size_t slots);

//возвращает значение позицию элемента в множестве,
//если значение value имеется в множестве set, иначе - n
size_t ordered_array_set_in (ordered_array_set *set, int value);

//возвращает значение ’истина’, если элементы множеств set1 и set2
равны, иначе - ’ложь’
bool ordered_array_set_isEqual (ordered_array_set set1,
ordered_array_set set2);

//возвращает значение ’истина’, если subset является подмножеством
set, иначе - ’ложь’
bool ordered_array_set_isSubset (ordered_array_set subset,
ordered_array_set set);

//возбуждает исключение, если в множество по адресу set нельзя
вставить элемент
void ordered_array_set_isAbleAppend (ordered_array_set *set);

//добавляет элемент value в множество set
void ordered_array_set_insert (ordered_array_set *set, int value);
```

```
//удаляет элемент value из множества set
void ordered_array_set_deleteElement (ordered_array_set *set, int
value);

//возвращает объединение множеств set1 и set2
ordered_array_set join (ordered_array_set set1, ordered_array_set
set2);

//возвращает пересечение множеств set1 и set2
ordered_array_set intersection (ordered_array_set set1,
ordered_array_set set2);

//возвращает разность множеств set1 и set2
ordered_array_set difference (ordered_array_set set1,
ordered_array_set set2);

//возвращает симметрическую разность множеств set1 и set2
ordered_array_set symmetricDifference (ordered_array_set set1,
ordered_array_set set2);

//возвращает дополнение до универсума universumSet множества set
ordered_array_set complement (ordered_array_set set,
ordered_array_set universumSet);

//вывод множества set
void ordered_array_set_print (ordered_array_set set);

//освобождает память, занимаемую множеством set
void ordered_array_set_delete (ordered_array_set *set);
#endif
```

А следующим образом будет выглядеть файл реализации, **ordered_array_set.c**:

```
#ifndef INC_ORDERED_ARRAY_SET_C
#define INC_ORDERED_ARRAY_SET_C

#include "ordered_array_set.h"
#include "C:\Users\sovac\Desktop\ОП, преимущественно лабы\
second_semester\libs\algorithms\array\array.c"
#include "C:\Users\sovac\Desktop\ОП, преимущественно лабы\
second_semester\libs\algorithms\math_basics\math_basics.c"

ordered_array_set ordered_array_set_create (size_t capacity) {
   return (ordered_array_set) {
       malloc(sizeof(int) * capacity),
       0,
```

```
        capacity
    };
}

ordered_array_set ordered_array_set_create_from_array (const int *a,
size_t size) {
    ordered_array_set result = ordered_array_set_create(size);
    for (size_t i = 0; i < size; i++) {
        ordered_array_set_insert(&result, a[i]);
    }
    qsort(result.data, size, sizeof(int), compare_ints);
    ordered_array_set_shrinkToFit(&result);
    return result;
}

static void ordered_array_set_shrinkToFit (ordered_array_set *a) {
    if (a->size != a->capacity) {
        a->data = (int *) realloc(a->data, sizeof(int) * a->size);
        a->capacity = a->size;
    }
}

static void ordered_array_set_increaseCapacity (ordered_array_set
*a, size_t slots) {
    a->data = (int *) realloc(a->data, sizeof(int) * (a->size +
slots));
    a->capacity += slots;
}

size_t ordered_array_set_in (ordered_array_set *set, int value) {
    size_t index = binarySearch_(set->data, set->size,value);
    return (index != SIZE_MAX) ? index : set->size;
}

bool ordered_array_set_isEqual (ordered_array_set set1,
ordered_array_set set2) {
    return memcmp(set1.data, set2.data, sizeof(int) * set1.size) ==
0;
}

bool ordered_array_set_isSubset (ordered_array_set subset,
ordered_array_set set) {
    bool is_subset = 1;
    for (size_t i = 0; i < subset.size; i++) {
        if (ordered_array_set_in(&set, subset.data[i]) == set.size)
```

```
{
            is_subset = 0;
            break;
        }
    }
    return is_subset;
}

void ordered_array_set_isAbleAppend (ordered_array_set *set) {
    assert(set->size < set->capacity);
}

void ordered_array_set_insert (ordered_array_set *set, int value) {
    ordered_array_set_isAbleAppend(set);
    if (set->size == 0 || value > set->data[(set->size)-1]) {
        append_(set->data, &(set->size), value);
    } else if (ordered_array_set_in(set, value) == set->size) {
        size_t start_index = binarySearchMoreOrEqual_(set->data,
set->size, value);
        insert_(set->data, &(set->size), start_index, value);
    }
}

void ordered_array_set_deleteElement (ordered_array_set *set, int
value) {
    if (ordered_array_set_in(set, value) != set-> size) {
        deleteByPosSaveOrder_(set->data, &(set->size),
ordered_array_set_in(set, value));
    }
}

ordered_array_set join (ordered_array_set set1, ordered_array_set
set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity + set2.capacity);
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
        if (set1.data[index1] == set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
```

```c
        } else {
            ordered_array_set_insert(&result, set2.data[index2]);
            index2++;
        }
    }
    while (index1 < set1.size) {
        ordered_array_set_insert(&result, set1.data[index1]);
        index1++;
    }
    while (index2 < set2.size) {
        ordered_array_set_insert(&result, set2.data[index2]);
        index2++;
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}

ordered_array_set intersection (ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity);
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
        if (set1.data[index1] == set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            index1++;
        } else {
            index2++;
        }
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}

ordered_array_set difference (ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity);
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
```

```c
        if (set1.data[index1] == set2.data[index2]) {
            index1++;
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
        } else {
            index2++;
        }
    }
    while (index1 < set1.size) {
        ordered_array_set_insert(&result, set1.data[index1]);
        index1++;
    }
    ordered_array_set_shrinkToFit(&result);
    return result;
}

ordered_array_set symmetricDifference (ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result =
ordered_array_set_create(set1.capacity + set2.capacity);
    size_t index1 = 0;
    size_t index2 = 0;
    while (index1 < set1.size && index2 < set2.size) {
        if (set1.data[index1] == set2.data[index2]) {
            index1++;
            index2++;
        } else if (set1.data[index1] < set2.data[index2]) {
            ordered_array_set_insert(&result, set1.data[index1]);
            index1++;
        } else {
            ordered_array_set_insert(&result, set2.data[index2]);
            index2++;
        }
    }
    while (index1 < set1.size) {
        ordered_array_set_insert(&result, set1.data[index1]);
        index1++;
    }
    while (index2 < set2.size) {
        ordered_array_set_insert(&result, set2.data[index2]);
        index2++;
    }
    ordered_array_set_shrinkToFit(&result);
```

```
    return result;
}

ordered_array_set complement (ordered_array_set set,
ordered_array_set universumSet) {
    return difference(universumSet, set);
}

void ordered_array_set_print (ordered_array_set set) {
    outputArray_(set.data, set.size);
}

void ordered_array_set_delete (ordered_array_set *set) {
    free(set->data);
    set->size = 0;
    set->capacity = 0;
}

#endif
```

Далее вне функции main определим функции, высчитывающие значение выражения, полученного выше:

```
#include "ordered_array_set\ordered_array_set.c"

//Возвращает множество - левую часть данного в варианте 9 уравнения
при заданных множествах X, A, B, C и универсуме universum
ordered_array_set left_part (ordered_array_set X, ordered_array_set
A, ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    ordered_array_set exp1 = symmetricDifference(A, X);
    ordered_array_set exp2 = join(B, X);
    ordered_array_set exp3 = intersection(exp1, exp2);
    ordered_array_set result = difference(exp3, C);
    return result;
}

//Возвращает множество - правую часть данного в варианте 9 уравнения
при заданных множествах X, A, B, C и универсуме universum
ordered_array_set right_part (ordered_array_set X, ordered_array_set
A, ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    ordered_array_set exp1 = join(A, X);
    ordered_array_set exp2 = complement(exp1, universum);
    ordered_array_set exp3 = symmetricDifference(C, X);
    ordered_array_set result = join(exp2, exp3);
    return result;
```

```
}

//Возвращает множество - уравнение варианта 9 с пустой правой частью
при заданных множествах X, A, B, C и универсуме universum
ordered_array_set equation (ordered_array_set X, ordered_array_set
A, ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    return symmetricDifference(left_part(X, A, B, C, universum),
right_part(X, A, B, C, universum));
}
```

И в теле функции main для вычисления значений множеств $\varphi^{\varnothing}$ и $\overline{\varphi}^{U}$ напишем следующий код:

```
int main() {
    //Задаются массивы с элементами множеств A, B, C, а также
элементами универсума
    int array_a[5] = {1, 3, 5, 6, 8};
    int array_b[4] = {2, 4, 6, 9};
    int array_c[4] = {2, 6, 7, 10};
    int array_u[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    //Из них создаются непосредственно множества
    ordered_array_set A =
ordered_array_set_create_from_array(array_a, 5);
    ordered_array_set B =
ordered_array_set_create_from_array(array_b, 4);
    ordered_array_set C =
ordered_array_set_create_from_array(array_c, 4);
    ordered_array_set universum =
ordered_array_set_create_from_array(array_u, 10);

    //Вычисляется значение множества φ∅: на место X в функцию
equation подставляется пустое множество
    ordered_array_set phi_empty =
equation(ordered_array_set_create(1), A, B, C, universum);

    //Вычисляется значение множества φ̄U: на место X в функцию
equation подставляется универсум
    //Потом над значением функции equation производится операция
дополнения до универсума
    ordered_array_set complement_phi_universum =
complement(equation(universum, A, B, C, universum), universum);

    //Полученные множества выводятся на экран
    ordered_array_set_print(phi_empty);
    ordered_array_set_print(complement_phi_universum);
```

```
    return 0;
}
```

Задание 4.

Итак, программа написана, и запустив ее, мы можем вычислить значения искомых множеств. На вывод она подаст следующее:

```
"C:\Users\sovac\Desktop\дискретная математика\ДИСКРЕТКА ЛАБА 4\phi_count.exe"
2 4 6 7 9 10
2 4 6 7 9 10

Process finished with exit code 0
```

Таким образом, можем записать, что $\varphi^{\varnothing} = \{2, 4, 6, 7, 9, 10\}$ и $\overline{\varphi}^{U} = \{2, 4, 6, 7, 9, 10\}$. Для того, чтобы решение уравнения существовало, необходимо, чтобы множество $\varphi^{\varnothing}$ являлось подмножеством $\overline{\varphi}^{U}$. Ведь если $\varphi^{\varnothing} \subseteq X$ и $X \subseteq \overline{\varphi}^{U}$, то из этого следует, что $\varphi^{\varnothing} \subseteq \overline{\varphi}^{U}$. Данное условие выполняется (множества равны, но вхождение $\varphi^{\varnothing}$ в $\overline{\varphi}^{U}$ и не должно быть строгим), значит, существует единственное решение.

Задание 5.

Итак, множеством решений уравнения будет множество таких множеств X, которые являются надмножествами $\varphi^{\varnothing}$ и подмножествами $\overline{\varphi}^{U}$, причем в обоих случаях включение одного множества в другое нестрогое. Иначе говоря, множество решений уравнения можно выразить как множество объединений $\varphi^{\varnothing}$ со всеми подмножествами разности $\overline{\varphi}^{U} - \varphi^{\varnothing}$. Однако в нашем частном случае $\varphi^{\varnothing}$ и $\overline{\varphi}^{U}$ равны, и их разность равна пустому множеству. Значит, его единственное подмножество — пустое множество. Результатом объединения пустого множества с $\varphi^{\varnothing}$ будет то же самое множество $\varphi^{\varnothing} = \{2, 4, 6, 7, 9, 10\}$.

Значит, множество решений будет записано как $\{\{2, 4, 6, 7, 9, 10\}\}$, и мощность общего решения будет равна 1 (поскольку решение лишь одно). Единственное частное решение, $\{2, 4, 6, 7, 9, 10\}$, является решением наименьшей и наибольшей мощности одновременно, и эта мощность равна 6 (во множестве X шесть элементов).

Задание 6.

Поскольку решение уравнения, данного в этом варианте, у нас лишь одно, мы можем задать его в программе с помощью литералов, оставив задачу генерации всех возможных подмножеств разности $\overline{\varphi}^{U} - \varphi^{\varnothing}$ на следующую лабораторную работу. Для проверки решения запустим следующий код (все так же использующий функции left_part и right_part):

```
int main() {
    //Задаются массивы с элементами множеств A, B, C, а также
элементами универсума
```

```
    int array_a[5] = {1, 3, 5, 6, 8};
    int array_b[4] = {2, 4, 6, 9};
    int array_c[4] = {2, 6, 7, 10};
    int array_u[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    //Из них создаются непосредственно множества
    ordered_array_set A =
ordered_array_set_create_from_array(array_a, 5);
    ordered_array_set B =
ordered_array_set_create_from_array(array_b, 4);
    ordered_array_set C =
ordered_array_set_create_from_array(array_c, 4);
    ordered_array_set universum =
ordered_array_set_create_from_array(array_u, 10);

    //Задаем массив с множествами-решениями уравнения
    //Можно было бы создать отдельную переменную вместо массива, но
такое решение делает программу более универсальной
    //Позволяя вставлять в массив решений больше множеств при
надобности
    ordered_array_set solutions[1] = {
            ordered_array_set_create_from_array((int[6]) {2, 4, 6,
7, 9, 10}, 6)
    };
    int solutions_len = 1;

    //Создаем флаг, контролирующий, все ли решения являются
истинными
    bool are_all_solutions_correct = true;

    //перебираем все решения в массиве
    for (int i = 0; i < solutions_len; i++) {
        //высчитываем значения левой и правой части уравнения
        ordered_array_set left_part_calculated =
left_part(solutions[i], A, B, C, universum);
        ordered_array_set right_part_calculated =
right_part(solutions[i], A, B, C, universum);
        //если они не равны, выводим сообщение об ошибочном решении
на экран и устанавливаем в 0 флаг
        if (!ordered_array_set_isEqual(left_part_calculated,
right_part_calculated)) {
            printf("This set isn't a solution of the equation: ");
            ordered_array_set_print(solutions[i]);
            are_all_solutions_correct = false;
        }
    }
```

```
    //Если флаг все еще истинен, выводим на экран сообщение о том,
что все множества проверены и верны
    if (are_all_solutions_correct) {
        printf("All solutions are checked and correct");
    }

    return 0;
}
```

Вывод на экран будет следующим:

```
"C:\Users\sovac\Desktop\дискретная математика\ДИСКРЕТКА ЛАБА 4\check_solutions.exe"
All solutions are checked and proven
Process finished with exit code 0
```

Что показывает, что наше единственное решение является верным.

Вывод:

В ходе лабораторной работы научились решать теоретико-множественные уравнения наиболее формализованным и универсальным способом — через разложение Шеннона, применив для этого написанный самостоятельно программный код.