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 №1.3
по дисциплине: «Дискретная математика»
по теме: «**Теоретико-множественные тождества**»

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

Белгород, 2024

Цель: изучить методы доказательства теоретико-множественных тождеств.

Содержание:

Вариант 9

Задание:

Дано множество ТМВ:

$\{A \cap B \cup \overline{A} \cap C \cup B \cap C,$

$\overline{A \cap B} \cap C \cup A \cap \overline{C},$

$A \cap \overline{C} \cap (A \cup B \cup \overline{C}) \cup (\overline{A} \cup C) \cap (\overline{A} \cap \overline{B} \cap C),$

$(\overline{A \cup B} - \overline{C}) \cup (\overline{C} - \overline{A}),$

$(A - (A - B)) \Delta (C \cap A) \Delta C\}$

Нужно получить все двухэлементные подмножества этого множества, состоящие из тождественных ТМВ, и составить из них тождества. Для проверки тождественности ТМВ использовать методы доказательства теоретико-множественных тождеств:

1) метод эквивалентных преобразований;

2) теоретико-множественный метод.

Применяя метод эквивалентных преобразований, нужно ТМВ преобразовать в совершенную нормальную форму Кантора, используя разложение Шеннона.

Задание 1: нахождение тождественных пар с помощью метода эквивалентных преобразований.

Метод эквивалентных преобразований сводится к тому, что два теоретико-множественных выражения, проверяемых на тождество, с помощью законов алгебры множеств и свойств операций над множествами преобразуются к одному виду. Более эффективным использованием этого метода будет приведение обоих теоретико-множественных выражений к совершенной НФК. Каждое выражение имеет только одну совершенную НФК, и потому приведение к ней универсально и сразу дает ответ на вопрос, являются ли множества тождественно равными: они являются таковыми тогда и только тогда, когда равны их совершенные НФК, то есть каждая конституента в совершенной НФК первого выражения есть в совершенной НФК второго, и наоборот.

Как мы уже знаем из курса дискретной математики, совершенная НФК любого теоретико-множественного выражения может быть составлена с помощью разложения Шеннона. Теоретико-множественное выражение из трех первичных термов (не учитываются первичные термы с дополнениями) расписывается следующим образом:

$\overline{A} \cap \overline{B} \cap \overline{C} \cap F(\varnothing, \varnothing, \varnothing) \cup$

$\overline{A} \cap \overline{B} \cap C \cap F(\varnothing, \varnothing, U) \cup$

$\overline{A} \cap B \cap \overline{C} \cap F(\varnothing, U, \varnothing) \cup$

$\overline{A} \cap B \cap C \cap F(\varnothing, U, U) \cup$

…

$A \cap B \cap \overline{C} \cap F(U, U, \varnothing) \cup$

$A \cap B \cap C \cap F(U, U, U)$,

где F(A, B, C) – само теоретико-множественное выражение, и значения A, B, C – переданные функции F аргументы. То есть перебираются все возможные конституенты, и если значение функции F (с аргументами Ø вместо первичного терма с дополнением и U вместо первичного терма без дополнения) будет равно U, то конституента входит в совершенную НФК.

Преобразовывать пять выражений в совершенную НФК вручную может быть немного утомительно. К счастью, разработка ПО для автоматизации этого процесса в данной лабораторной работе только поощряется. Базой для программ станет библиотека для работы с множествами, представленными на упорядоченном массиве, которая была разработана в рамках курса ОП.
Приведем здесь заголовочный файл, **ordered_array_set.h:**

```
#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>

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 isEqual (ordered_array_set set1, ordered_array_set set2);

//возвращает значение ’истина’, если subset является подмножеством
set
//иначе - ’ложь’
bool 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
```

...И 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) {
```

```c
        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 isEqual (ordered_array_set set1, ordered_array_set set2) {
    if (set1.size == set2.size) {
        return memcmp(set1.data, set2.data, sizeof(int) *
set1.size) == 0;
    } else {
        return 0;
    }
}

bool 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);
```

```c
    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++;
        } 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) {
        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 напишем программу, которая выводит на экран совершенные НФК тех пяти теоретико-множественных выражений, которые мы получили в задании. Похожий код мы уже писали во время выполнения лабораторной работы 1.2. Во-первых, вне функции main определим функции для вычисления данных пяти выражений при заданных множествах A, B, C и заданном универсуме universum.

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

ordered_array_set first_expression(ordered_array_set A,
ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    ordered_array_set exp1 = intersection(A, B);
    ordered_array_set exp2 = intersection(complement(A,
universum), C);
    ordered_array_set exp3 = intersection(B, C);
    ordered_array_set exp4 = join(exp1, exp2);
    ordered_array_set result = join(exp4, exp3);
    return result;
}

ordered_array_set second_expression(ordered_array_set A,
ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    ordered_array_set exp1 = intersection(A, B);
    ordered_array_set exp2 = intersection(complement(exp1,
universum), C);
    ordered_array_set exp3 = intersection(A, complement(C,
universum));
    ordered_array_set result = join(exp2, exp3);
    return result;
}

ordered_array_set third_expression(ordered_array_set A,
ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    ordered_array_set exp1 = join(A, B);
    ordered_array_set exp2 = join(exp1, complement(C, universum));
    ordered_array_set exp3 = intersection(A, complement(C,
universum));
    ordered_array_set exp4 = intersection(exp3, exp2);
```

```
    ordered_array_set exp5 = join(complement(A, universum), C);
    ordered_array_set exp6 = intersection(complement(A,
universum), complement(B, universum));
    ordered_array_set exp7 = intersection(exp6, C);
    ordered_array_set exp8 = intersection(exp5, exp7);
    ordered_array_set result = join(exp4, exp8);
    return result;
}

ordered_array_set fourth_expression(ordered_array_set A,
ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    ordered_array_set exp1 = join(A, B);
    ordered_array_set exp2 = difference(complement(exp1,
universum), complement(C, universum));
    ordered_array_set exp3 = difference(complement(C, universum),
complement(A, universum));
    ordered_array_set result = join(exp2, exp3);
    return result;
}

ordered_array_set fifth_expression(ordered_array_set A,
ordered_array_set B, ordered_array_set C, ordered_array_set
universum) {
    ordered_array_set exp1 = difference(A, B);
    ordered_array_set exp2 = difference(A, exp1);
    ordered_array_set exp3 = intersection(C, A);
    ordered_array_set exp4 = symmetricDifference(exp2, exp3);
    ordered_array_set result =  symmetricDifference(exp4, C);
    return result;
}
```

Далее поступим следующим образом. Если конституента состоит из трех первичных термов, то можно представить ее как двоичное число, где 0 – первичный терм с дополнением, 1 – первичный терм без дополнения, а расположение нулей и единиц соответствует алфавитному порядку множеств. Как пример, число 010 обозначает конституенту $\overline{A} \cap B \cap \overline{C}$.

Составим отдельную функцию для вывода конституенты на экран по данному двоичному числу и длине конституенты.

```
//Выводит конституенту из len множеств по заданному ее номеру -
двоичному числу, где 0 - множества с дополнением, 1 - множества
без дополнений
void printConstitution(int len, int number) {
    printf("(");
```

```
    for (int i = 0; i < len; i++) {
        if ((number >> (len-1-i) & 1) == 0) {
            printf("!");
        }
        printf("%c", 'A' + i);
        if (i != len-1) {
            printf(" ");
        }
    }
    printf(")");
}
```

И наконец, напишем функцию main, которая выведет на экран все СНФК, со следующим содержанием:

```
int main() {
    //создаем массив функций, в котором по порядку идут данные в 
условии теоретико-множественные выражения
    ordered_array_set (*functions[5])(ordered_array_set, 
ordered_array_set, ordered_array_set, ordered_array_set) = {
            first_expression,
            second_expression,
            third_expression,
            fourth_expression,
            fifth_expression
    };

    //создаем массив из двух элементов
    ordered_array_set nilAndOne[2];
    //первый элемент - пустое множество
    nilAndOne[0] = ordered_array_set_create_from_array((int[]) {},
0);
    //второй элемент - множество с одним элементом, единицей
    //так как результат выражения, операнды в котором - только 
пустое множество и универсум, может быть только пустым множеством 
или универсумом, одного элемента в универсуме хватит
    nilAndOne[1] = ordered_array_set_create_from_array((int[]) 
{1}, 1);

    //перебираем все функции в массиве функций
    for (int i_exp = 0; i_exp < 5; i_exp++) {
        //для каждой функции перебираем все возможные номера 
конституент
        for (int cur_const_ind = 0; cur_const_ind < 8; 
cur_const_ind++) {
            //высчитываем значение функции с пустым множеством и 
```

```
универсумом в качестве аргументов
            ordered_array_set i_cur_const = (functions[i_exp])
(nilAndOne[(cur_const_ind >> 2) & 1], nilAndOne[(cur_const_ind >>
1) & 1], nilAndOne[cur_const_ind & 1], nilAndOne[1]);
            //если значение функции - универсум, соответствующая
конституента войдет в совершенную НФК
            //выводим ее на экран, отделив от следующей табуляцией
            if (isEqual(i_cur_const, nilAndOne[1])) {
                printConstitution(3, cur_const_ind);
                printf("\t");
            }
        }
        //отделим совершенную НФК одной функции от совершенной НФК
другой переносом строки
        printf("\n");
    }
    return 0;
}
```

На экран будет выведено следующее:

```
"C:\Users\sovac\Desktop\дискретная математика\ЛАБА 3\perfect form print.exe"
(!A !B C)       (!A B C)        (A B !C)        (A B C)
(!A !B C)       (!A B C)        (A !B !C)       (A !B C)        (A B !C)
(!A !B C)       (A !B !C)       (A B !C)
(!A !B C)       (A !B !C)       (A B !C)
(!A !B C)       (!A B C)        (A B !C)        (A B C)

Process finished with exit code 0
```

Построчно сравнив совершенные НФК, можно заметить, что эквивалентны они:

1) У первого и пятого выражений,
2) У третьего и четвертого выражений.

Таким образом, подмножества тождественно равных выражений будут следующими: $\{A \cap B \cup \overline{A} \cap C \cup B \cap C, (A - (A - B)) \Delta (C \cap A) \Delta C\}$, $\{A \cap \overline{C} \cap (A \cup B \cup \overline{C}) \cup (\overline{A} \cup C) \cap (\overline{A} \cap \overline{B} \cap C), (\overline{A \cup B} - \overline{C}) \cup (\overline{C} - \overline{A})\}$, и записать тождества можно так:

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

Проверку НФК на равенство можно было бы автоматизировать, сделав необязательным их вывод на экран. Для этого слегка изменим функцию main:

```
int main() {
    //создаем массив функций, в котором по порядку идут данные в условии теоретико-множественные выражения
    ordered_array_set (*functions[5])(ordered_array_set, ordered_array_set, ordered_array_set, ordered_array_set) = {
            first_expression,
            second_expression,
            third_expression,
            fourth_expression,
            fifth_expression
    };

    //создаем массив из двух элементов
    ordered_array_set nilAndOne[2];
    //первый элемент - пустое множество
    nilAndOne[0] = ordered_array_set_create_from_array((int[]) {}, 0);
    //второй элемент - множество с одним элементом, единицей
    //так как результат выражения, операнды в котором - только пустое множество и универсум, может быть только пустым множеством или универсумом, одного элемента в универсуме хватит
    nilAndOne[1] = ordered_array_set_create_from_array((int[]) {1}, 1);

    //перебираем все возможные пары функций двумя вложенными циклами
    for (int i_exp = 0; i_exp < 4; i_exp++) {
        for (int j_exp = i_exp+1; j_exp < 5; j_exp++) {
            //устанавливаем флаг, который контролирует равенство совершенных НФК, по умолчанию истинный
            bool are_perfect_forms_equal = true;
            //для данной пары функций перебираем все возможные номера конституент
            for (int cur_comp_ind = 0; cur_comp_ind < 8; cur_comp_ind++) {
                //высчитываем значение каждой функции с пустым множеством и универсумом в качестве аргументов
                ordered_array_set i_cur_comp = functions[i_exp](nilAndOne[(cur_comp_ind >> 2) & 1], nilAndOne[(cur_comp_ind >> 1) & 1], nilAndOne[cur_comp_ind & 1], nilAndOne[1]);
                ordered_array_set j_cur_comp = functions[j_exp](nilAndOne[(cur_comp_ind >> 2) & 1], nilAndOne[(cur_comp_ind >> 1) & 1], nilAndOne[cur_comp_ind & 1], nilAndOne[1]);
                //если они не равны, значит, в какую-то из совершенных НФК войдет конституента, не входящая в другую: они не
```

```
равны
                if (!isEqual(i_cur_comp, j_cur_comp)) {
                    are_perfect_forms_equal = false;
                    break;
                }
            }
            //если совершенные НФК равны, выводим на экран сообщение
            if (are_perfect_forms_equal) {
                printf("Expression %d and %d are identically equal\
n", i_exp+1, j_exp+1);
            }
        }
    }
    return 0;
}
```

После выполнения этой программы вывод будет следующим:

```
"C:\Users\sovac\Desktop\дискретная математика\ДИСКРЕТКА ЛАБА 3\perfect form method.exe"
Expression 1 and 5 are identically equal
Expression 3 and 4 are identically equal

Process finished with exit code 0
```

Как видим, он совпадает с результатами, которые мы получили, используя предыдущую програму, так что мы можем записать все те же тождества:

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

Задание 2: нахождение тождественных пар с помощью теоретико-множественного метода.

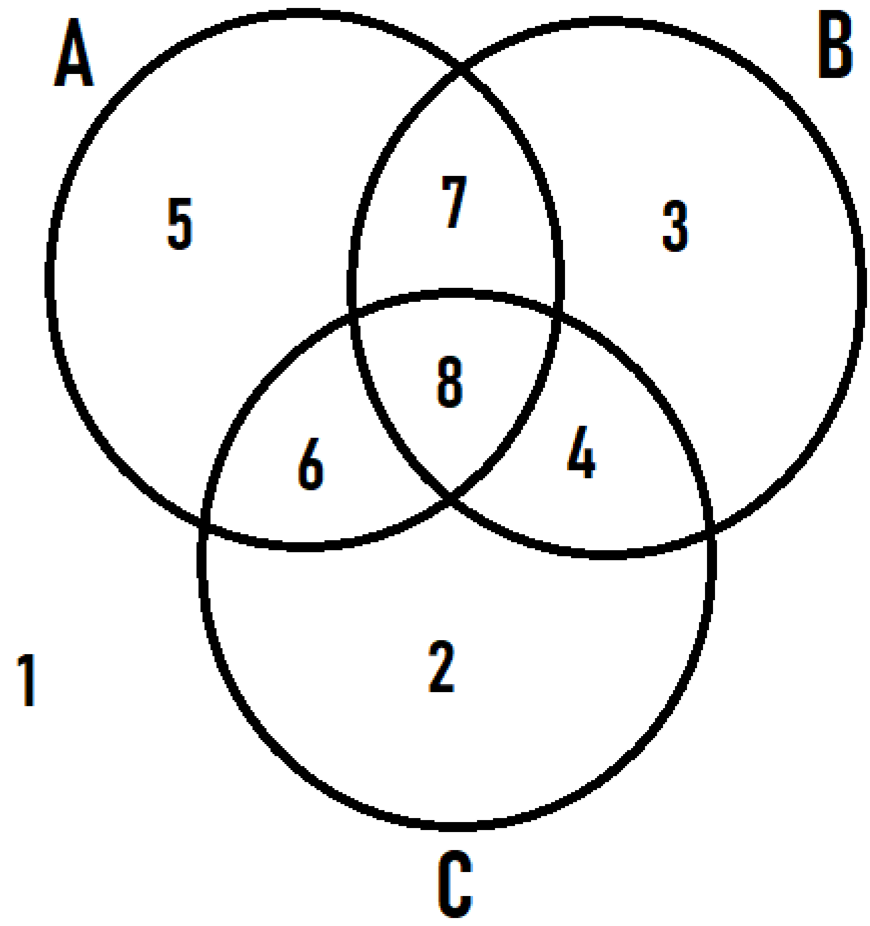
Если первичных термов (без дополнения) три, то они разделят универсум на несколько частей, и каждой части можно будет присовить двоичный вектор длины 3, каждая компонента которого обозначает, принадлежит ли эта часть определенному первичному терму. Всего таких векторов будет $2^3 = 8$, и универсум окажется разбит на 8 частей.

Составим таблицу, которая будет отображать, какая область каким множествам принадлежит:

Область/ первичный терм	A	B	C
1	0	0	0
2	0	0	1
3	0	1	0

4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Отобразим эту картину на кругах Эйлера.



Таким образом, получим множества A = {5, 6, 7, 8}, B = {3, 4, 7, 8}, C = {2, 4, 6, 8} при U = {1, 2, 3, 4, 5, 6, 7, 8}. Все множества находятся друг относительно друга в общем положении, все варианты вхождения и не-вхождения элемента универсума в множества представлены. Следовательно, можно говорить о том, что если значения теоретико-множественных выражений для этих значений A, B, C равны, то они будут равны для любых значений и являются тождественно равными. Нам остается только вычислить значения пяти данных теоретико-множественных выражений и сравнить их между собой.

Для удобства сразу обозначим значения множеств $\overline{A}$, $\overline{B}$, $\overline{C}$: $\overline{A}$ = {1, 2, 3, 4}, $\overline{B}$ = {1, 2, 5, 6}, $\overline{C}$ = {1, 3, 5, 7}.

Тогда для первого выражения, $A \cap B \cup \overline{A} \cap C \cup B \cap C$:

1. $A \cap B = \{5, 6, 7, 8\} \cap \{3, 4, 7, 8\} = \{7, 8\}$
2. $\underline{A} \cap C = \{1, 2, 3, 4\} \cap \{2, 4, 6, 8\} = \{2, 4\}$
3. $B \cap C = \{3, 4, 7, 8\} \cap \{2, 4, 6, 8\} = \{4, 8\}$
4. $A \cap B \cup \overline{A} \cap C = \{7, 8\} \cup \{2, 4\} = \{2, 4, 7, 8\}$
5. $A \cap B \cup \overline{A} \cap C \cup B \cap C = \{2, 4, 7, 8\} \cup \{4, 8\} = \{2, 4, 7, 8\}$

Для второго выражения, $\overline{A \cap B} \cap C \cup A \cap \overline{C}$:

1. $A \cap B = \{5, 6, 7, 8\} \cap \{3, 4, 7, 8\} = \{7, 8\}$
2. $\overline{A \cap B} = \{7, 8\} = \{1, 2, 3, 4, 5, 6, 7, 8\} - \{7, 8\} = \{1, 2, 3, 4, 5, 6\}$
3. $\overline{A \cap B} \cap C = \{1, 2, 3, 4, 5, 6\} \cap \{2, 4, 6, 8\} = \{2, 4, 6\}$
4. $A \cap \overline{C} = \{5, 6, 7, 8\} \cap \{1, 3, 5, 7\} = \{5, 7\}$
5. $\overline{A \cap B} \cap C \cup A \cap \overline{C} = \{2, 4, 6\} \cup \{5, 7\} = \{2, 4, 5, 6, 7\}$

Для третьего выражения, $A \cap \overline{C} \cap (A \cup B \cup \overline{C}) \cup (\overline{A} \cup C) \cap (\overline{A} \cap \overline{B} \cap C)$:

1. $A \cup B = \{5, 6, 7, 8\} \cup \{3, 4, 7, 8\} = \{3, 4, 5, 6, 7, 8\}$
2. $A \cup B \cup \overline{C} = \{3, 4, 5, 6, 7, 8\} \cup \{1, 3, 5, 7\} = \{1, 3, 4, 5, 6, 7, 8\}$
3. $A \cap \overline{C} = \{5, 6, 7, 8\} \cap \{1, 3, 5, 7\} = \{5, 7\}$
4. $A \cap \overline{C} \cap (A \cup B \cup \overline{C}) = \{5, 7\} \cap \{1, 3, 4, 5, 6, 7, 8\} = \{5, 7\}$
5. $\overline{A} \cup C = \{1, 2, 3, 4\} \cup \{2, 4, 6, 8\} = \{1, 2, 3, 4, 6, 8\}$
6. $\overline{A} \cap \overline{B} = \{1, 2, 3, 4\} \cap \{1, 2, 5, 6\} = \{1, 2\}$
7. $\overline{A} \cap \overline{B} \cap C = \{1, 2\} \cap \{2, 4, 6, 8\} = \{2\}$
8. $(\overline{A} \cup C) \cap (\overline{A} \cap \overline{B} \cap C) = \{1, 2, 3, 4, 6, 8\} \cap \{2\} = \{2\}$
9. $A \cap \overline{C} \cap (A \cup B \cup \overline{C}) \cup (\overline{A} \cup C) \cap (\overline{A} \cap \overline{B} \cap C) = \{5, 7\} \cup \{2\} = \{2, 5, 7\}$

Для четвертого выражения, $(\overline{A \cup B} - \overline{C}) \cup (\overline{C} - \overline{A})$:

1. $A \cup B = \{5, 6, 7, 8\} \cup \{3, 4, 7, 8\} = \{3, 4, 5, 6, 7, 8\}$
2. $A \cup B = \{3, 4, 5, 6, 7, 8\} = \{1, 2, 3, 4, 5, 6, 7, 8\} - \{3, 4, 5, 6, 7, 8\} = \{1, 2\}$
3. $A \cup B - C = \{1, 2\} - \{1, 3, 5, 7\} = \{2\}$

4. $C - A = \{1, 3, 5, 7\} - \{1, 2, 3, 4\} = \{5, 7\}$
5. $(\overline{A \cup B} - \overline{C}) \cup (\overline{C} - \overline{A}) = \{2\} \cup \{5, 7\} = \{2, 5, 7\}$

И наконец, для пятого выражения, $(A - (A - B)) \Delta (C \cap A) \Delta C$:

1. $A - B = \{5, 6, 7, 8\} - \{3, 4, 7, 8\} = \{5, 6\}$
2. $A - (A - B) = \{5, 6, 7, 8\} - \{5, 6\} = \{7, 8\}$
3. $C \cap A = \{2, 4, 6, 8\} \cap \{5, 6, 7, 8\} = \{6, 8\}$
4. $(A - (A - B)) \Delta (C \cap A) = \{7, 8\} \Delta \{6, 8\} = \{6, 7\}$
5. $(A - (A - B)) \Delta (C \cap A) \Delta C = \{6, 7\} \Delta \{2, 4, 6, 8\} = \{2, 4, 7, 8\}$

Сравнив итоговые результаты, мы заметим, что одинаковыми являются результаты теоретико-множественных выражений 1 и 5 (результатом является множество $\{2, 4, 7, 8\}$) и для выражений 3 и 4 (результатом является множество $\{2, 5, 7\}$). То есть мы можем выделить все те же подмножества тождественных теоретико-множественных выражений, $\{A \cap B \cup \overline{A} \cap C \cup B \cap C, (A - (A - B)) \Delta (C \cap A) \Delta C\}$, $\{A \cap \overline{C} \cap (A \cup B \cup \overline{C}) \cup (\overline{A} \cup C) \cap (\overline{A} \cap \overline{B} \cap C), (\overline{A \cup B} - \overline{C}) \cup (\overline{C} - \overline{A})\}$, и записать тождества можно так:

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

Проверку получившихся при заданных значениях A, B, C множеств можно также автоматизировать, если, используя написанные выше функции first_expression, second_expression, third_expression, fourth_expression, fifth_expression, вставить в качестве функции main следующий код:

```
int main() {
    //задаем множество-универсум и множества A, B, C в соответствии с ранее составленной таблицей
    ordered_array_set universum = ordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6, 7, 8}, 8);
    ordered_array_set a = ordered_array_set_create_from_array((int[]){5, 6, 7, 8 }, 4);
    ordered_array_set b = ordered_array_set_create_from_array((int[]){3, 4, 7, 8}, 4);
    ordered_array_set c = ordered_array_set_create_from_array((int[]){2, 4, 6, 8}, 4);

    //задаем массив функций, вычисляющих данные ТМВ при заданных значениях A, B, C
```

```
    ordered_array_set (*functions[5])(ordered_array_set,
ordered_array_set, ordered_array_set, ordered_array_set) = {
            first_expression,
            second_expression,
            third_expression,
            fourth_expression,
            fifth_expression
    };

    //создаем массив множеств для хранения результатов вычисления
данных ТМВ
    ordered_array_set results[5];
```

```
    //для каждого выражения находим значение и сохраняем его в
массив results
    for (int i = 0; i < 5; i++) {
        results[i] = functions[i](a, b, c, universum);
    }
```

```
    //перебираем все возможные пары результатов, и если множества-
результаты равны, выводим на экран сообщение о том, что тожественно
равны соответствующие выражения
    for (int i = 0; i < 4; i++) {
        for (int j = i+1; j < 5; j++) {
            if (isEqual(results[i], results[j])) {
                printf("Expression %d and %d are identically equal\
n", i+1, j+1);
            }
        }
    }
    return  0;
}
```

На экран будет выведено следующее:

```
"C:\Users\sovac\Desktop\дискретная математика\ДИСКРЕТКА ЛАБА 3\set-theoretic method.exe"
Expression 1 and 5 are identically equal
Expression 3 and 4 are identically equal

Process finished with exit code 0
```

Из чего мы можем сделать все тот же вывод о тождественности выражений:

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

Завершить выполнение работы можно таблицей, в которой наглядно видно, одинаковы или отличаются друг от друга СНФК данных в условии выражений и множества, полученные в ходе доказательства теоретико-множественным методом.

Выражение	СНФК	Значение
$A \cap B \cup \overline{A} \cap C \cup B \cap C$	$(\overline{A} \cap \overline{B} \cap C) \cup (\overline{A} \cap B \cap C) \cup (A \cap B \cap \overline{C}) \cup (A \cap B \cap C)$	{2, 4, 7, 8}
$\overline{A \cap B} \cap C \cup A \cap \overline{C}$	$(\overline{A} \cap \overline{B} \cap C) \cup (\overline{A} \cap B \cap C) \cup (A \cap \overline{B} \cap \overline{C}) \cup (A \cap \overline{B} \cap C) \cup (A \cap B \cap \overline{C})$	{2, 4, 5, 6, 7}
$A \cap \overline{C} \cap (A \cup B \cup \overline{C}) \cup (\overline{A} \cup C) \cap (\overline{A} \cap \overline{B} \cap C)$	$(\overline{A} \cap \overline{B} \cap C) \cup (A \cap \overline{B} \cap \overline{C}) \cup (A \cap B \cap \overline{C})$	{2, 5, 7}
$(\overline{A \cup B} - \overline{C}) \cup (\overline{C} - \overline{A})$	$(\overline{A} \cap \overline{B} \cap C) \cup (A \cap \overline{B} \cap \overline{C}) \cup (A \cap B \cap \overline{C})$	{2, 5, 7}
$(A - (A - B))\ \Delta\ (C \cap A)\ \Delta\ C$	$(\overline{A} \cap \overline{B} \cap C) \cup (\overline{A} \cap B \cap C) \cup (A \cap B \cap \overline{C}) \cup (A \cap B \cap C)$	{2, 4, 7, 8}

Вывод: в ходе лабораторной работы закрепили знания о различных методах доказательства теоретико-множественных тождеств и смогли частично автоматизировать этот процесс.