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 №1.1
Операции над множествами

по дисциплине: Дискретная математика

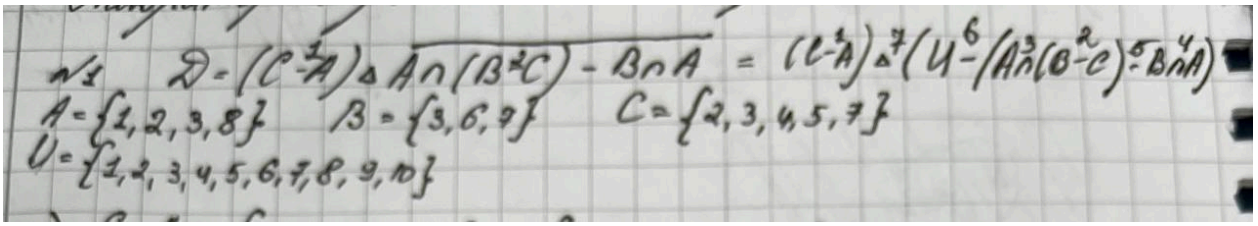
Выполнил: студент ПВ-233
Мороз Роман Алексеевич

Проверил: Островский Алексей
Мичеславович

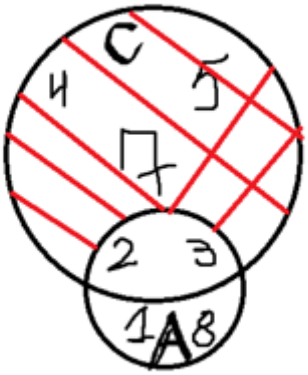
Белгород 2024 г.

Цель работы: изучить и научиться использовать алгебру подмножеств, изучить различные способы представления множеств в памяти ЭВМ, научиться программно реализовывать операции над множествами и выражения в алгебре подмножеств.

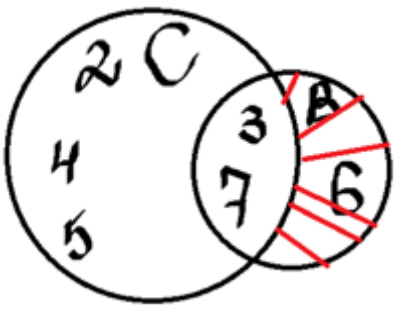
Задание 1 Вариант 8



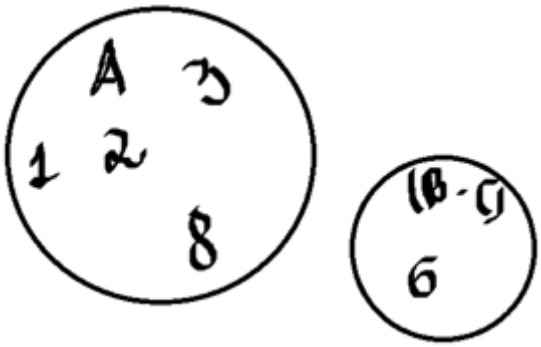
1) C-A = {2, 3, 4, 5, 7} – {1, 2, 3, 8} = {4, 5, 7}



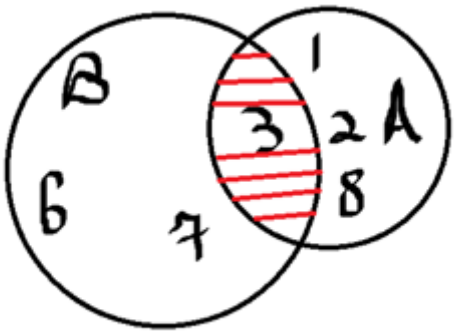
2) B-C = {3, 6, 7} – {2, 3, 4, 5, 7} = {6}



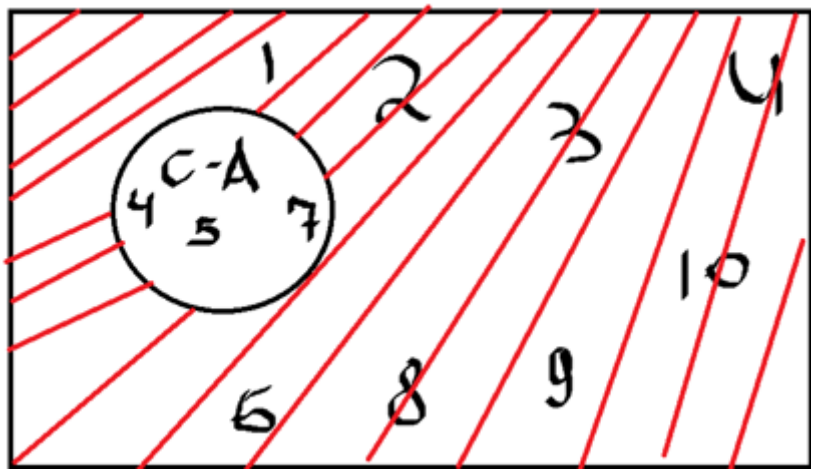
3) A∩(B-C) = {1, 2, 3, 8}∩{6} = {}



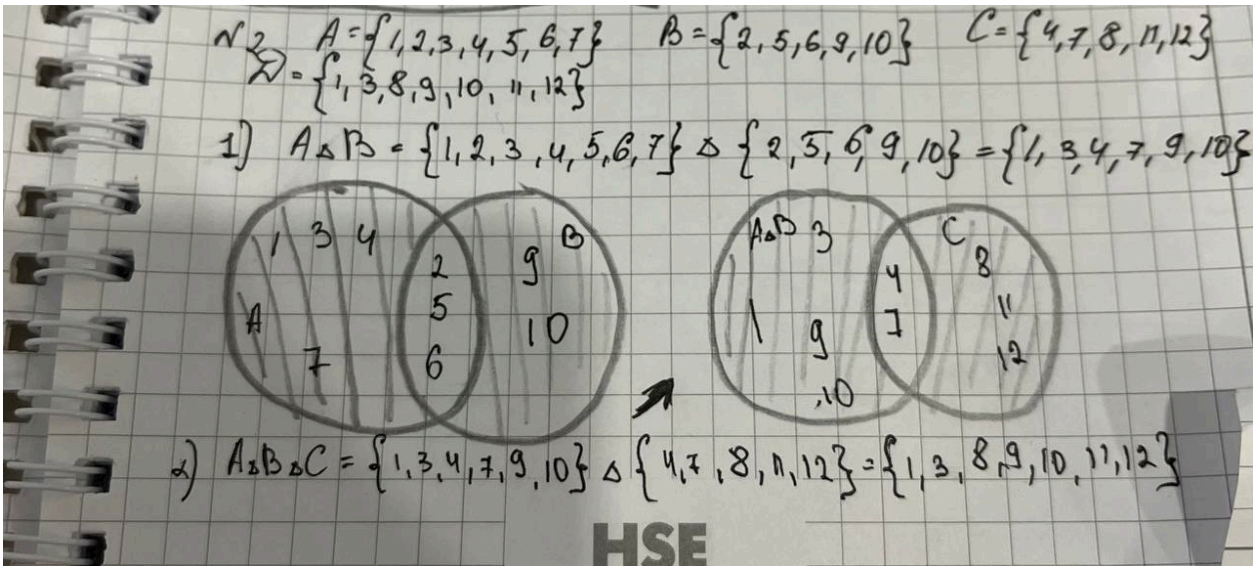
4) B∩A = {3, 6, 7} ∩ {1, 2, 3, 8} = {3}



5) A∩(B-C)-B∩A = {} – {3} = {}
6) U – (A∩(B-C)-B∩A) = U
7) (C-A) ∆(U – (A∩(B-C)-B∩A)) = {4, 5, 7} ∆{1,2,3,4,5,6,7,8,9,10} = {1, 2, 3, 6, 8, 9, 10}



Задание 2



A 1	2	3	4	5	6	7			
B	2			5	6		9 10		
C			4		6	7 8		11	12
D 1		3				8	9 10	11	12

Чтобы решить подобную задачу, нужно найти вход в решение. В данной ситуации заметим, что итоговое множество D содержит элементы 1 и 3, которые есть только в множестве A. Значит будет рационально взять за первый операнд множество A и множество B, ведь в нем присутствуют также элементы, которые есть в множестве D (9, 10). Не трудно заметить, что элементы множеств A и B такие как 2, 5, 6 не присутствуют в итоговом множестве D, соответственно от них необходимо избавиться, но так, чтобы оставить нужные нам элементы 1, 3, 9, 10. Для подобной проблемы воспользуемся операцией симметрической разницы множеств A и B. В итоге получим новое множество {1, 3 , 4, 7, 9, 10}. Приступим к анализу полученного множества и множества C. Абсолютно очевидно, чтобы получить D = {1, 3, 8, 9, 10, 11, 12} из множеств {1, 3, 4, 7, 9, 10} и C = {4, 7, 8, 11, 12} , нужно избавиться от элементов одинаковых элементов (4, 7, 8) и объединить уникальные. В этом случае снова используем операцию симметрической разности множеств и на выходе получаем нужное нам множество D

Задание 3.

а) элементы множества A хранятся в массиве A. Элементы масси-
ва A неупорядочены;

```
#include <stdio.h>
#include <stdbool.h>
#include "libs/algorithms/array/array.c"
```

```c
#define MAX_SIZE 100

typedef struct unordered_array_set{
    int data[MAX_SIZE];
    size_t size;
} unordered_array_set;

unordered_array_set unordered_array_set_create(size_t size) {
    unordered_array_set set = {
            {0},
            size,
    };
    return set;

}

// возвращает множество, состоящее из элементов массива a размера size
unordered_array_set unordered_array_set_create_from_array(const int *a, size_t
size) {
    unordered_array_set set = unordered_array_set_create(size);
    for (size_t i = 0; i < size; i++)
        set.data[i] = a[i];

    return set;
}

// возвращает позицию элемента в множестве,
// если значение value имеется в множестве set, иначе - n
int unordered_array_set_in(unordered_array_set *set, int value) {
    return linearSearch_(set->data, set->size, value);
}


bool containsArray(int *a, int *b, size_t n, size_t m) {
    bool contains = false;
    for (int i = 0; i <= n - m; i++) {
        bool match = true;
        for (int j = 0; j < m; j++) {
            if (a[i + j] != b[j]) {
                match = false;
                break;
            }
        }
        if (match) {
            contains = true;
            break;
        }
    }
    return contains;
}



// возвращает истину если subset является подмножеством set, иначе ложь
bool unordered_array_set_isSubset(unordered_array_set subset,
unordered_array_set set) {
    return (containsArray (subset.data, set.data, subset. size, set.size)) ?
true : false;
}
```

```
bool check_Same_Arrays(int *a, int *b, size_t n, size_t m) {
    if (n != m) {
        return false;
    }

    for (int i = 0; i < n; i++) {
        if (a[i] != b[i]) {
            return false;
        }
    }

    return true;
}



// возвращает значение 'истина', если элементы множеств set1 и set2 равны
// иначе - 'ложь'
bool unordered_array_set_isEqual(unordered_array_set set1, unordered_array_set
set2) {
    return (check_Same_Arrays(set1.data, set2.data, set1.size, set2.size)) ?
true : false;
}


// добавляет элемент value в множество set
void unordered_array_set_insert(unordered_array_set *set, int value) {
    if (unordered_array_set_in(set, value) != -1) {
        append_(set->data, (size_t *) set->size, value);
    }
}

// удаляет элемент value из множества set
void unordered_array_set_deleteElement(unordered_array_set *set, int value) {
    size_t pos = linearSearch_(set->data, set->size, value);
    deleteByPosSaveOrder_(set->data, (size_t *) set->size, pos);
}


// возвращает объединение множеств set1 и set2
unordered_array_set unordered_array_set_union(unordered_array_set set1,
unordered_array_set set2) {
    unordered_array_set set3;
    set3.size = 0;
    for (size_t i = 0; i < set1.size; i++) {
        set3.data[set3.size++] = set1.data[i];
    }
    for (size_t i = 0; i < set2.size; i++) {
        int flag = 0;
        for (size_t j = 0; j < set3.size; j++) {
            if (set3.data[j] == set2.data[i]) {
                flag = 1;
                break;
            }
        }
        if (!flag) {
            set3.data[set3.size++] = set2.data[i];
        }
    }

    return set3;
}

int CheckSameElements(int const *a, int const *b, size_t elements_a, size_t
```

```
elements_b) {
    int counter = 0;
    for (size_t i = 0; i < elements_a; i++) {
        for (int j = 0; j < elements_b; j++) {
            if (a[i] == b[j])
                counter++;
        }
    }
    return counter;
}

// возвращает пересечение множеств set1 и set2
unordered_array_set unordered_array_set_intersection(unordered_array_set set1,
unordered_array_set set2) {
    unordered_array_set set3;
    set3.size = 0;
    //set3.capacity = CheckSameElements(set1.data, set2.data, set1.size,
set2.size);
    for (size_t i = 0; i < set1.size; i++)
        for (size_t j = 0; j < set2.size; j++) {
            if (set1.data[i] == set2.data[j]) {
                set3.data[set3.size] = set1.data[i];
                set3.size++;
            }
        }
    return set3;
}

int Check_Value(int value, int *b, size_t n) {
    for (size_t i = 0; i < n; i++) {
        if (value == b[i])
            return 1;
    }
    return 0;
}

// возвращает разность множеств set1 и set2
unordered_array_set unordered_array_set_difference(unordered_array_set set1,
unordered_array_set set2) {
    unordered_array_set set3;
    set3.size = 0;
    for (size_t i = 0; i < set1.size; i++) {
        if (!Check_Value(set1.data[i], set2.data, set2.size)) {
            set3.data[set3.size] = set1.data[i];
            set3.size++;
        }
    }
    return set3;
}


// возвращает симметрическую разность множеств set1 и set2
unordered_array_set unordered_array_set_symmetricDifference(unordered_array_set
set1, unordered_array_set set2) {
    unordered_array_set set3 =unordered_array_set_difference(set1,set2);
    unordered_array_set set4 =unordered_array_set_difference(set2,set1);
    unordered_array_set set_res = unordered_array_set_union(set3, set4);
    return set_res;
}

// вывод множества set
void unordered_array_set_print(unordered_array_set set) {
    outputArray_(set.data, set.size);
}
```

```
unordered_array_set ordered_array_set_universum_create (size_t size) {
    ordered_array_set universum_set;
    universum_set.size = size;
    int value = 1;
    for (size_t i = 0; i < size; i++) {
        universum_set.data[i] = value++;
    }
    return universum_set;
}

// Проверка элемента на принадлежность множеству (нестрогое включение)
int isInSet(unordered_array_set set, int element) {
    for (int i = 0; i < set.size; i++) {
        if (set.data[i] == element) {
            return 1; // Возвращаем 1, если элемент содержится в множестве
        }
    }
    return 0; // Возвращаем 0, если элемент не найден в множестве
}
// Проверка элемента на строгое включение в множество
int isProperSubset(unordered_array_set set1, unordered_array_set set2) {
    if (set1.size >= set2.size) {
        return 0; // Если размер первого множества больше или равен размеру
второго, нет
       // строгого включения
    }
    for (int i = 0; i < set1.size; i++) {
        if (!isInSet(set2, set1.data[i])) {
            return 0; // Если хотя бы один элемент первого множества не
содержится во
            //втором, нет строгого включения
        }
    }
    return 1; // В противном случае, есть строгое включение
}
```

б) элементы множества A хранятся в массиве A. Элементы массива A упорядочены по возрастанию;

```
#include <stdio.h>
#include <stdbool.h>
#include "libs/algorithms/array/array.c"

# include <memory.h>

#define MAX_SIZE 100

// Сортировка выбором
void swapArrayElements(int *a, int pos1, int pos2) {
    int temp = a[pos1];
    a[pos1] = a[pos2];
    a[pos2] = temp;
}

void arraySort(int *a, size_t *n) {
    for (int i = 0; i < *n; i++) {
        int min_element = a[i];
        int min_element_index = i;
```

```
        for (int j = i; j < *n; j++) {
            if (a[j] < min_element) {
                min_element = a[j];
                min_element_index = j;
            }
        }
        swapArrayElements(a, i, min_element_index);
    }
}


typedef struct ordered_array_set{
    int data[MAX_SIZE];
    size_t size;
} ordered_array_set;

ordered_array_set ordered_array_set_create(size_t size) {
    ordered_array_set set = {
            {0},
            size,
    };
    return set;

}

// возвращает множество, состоящее из элементов массива a размера size
ordered_array_set ordered_array_set_create_from_array(const int *a, size_t
size) {
    ordered_array_set set = ordered_array_set_create(size);
    for (size_t i = 0; i < size; i++)
        set.data[i] = a[i];

    return set;
}

// возвращает позицию элемента в множестве,
// если значение value имеется в множестве set, иначе - n
int ordered_array_set_in(ordered_array_set *set, int value) {
    return linearSearch_(set->data, set->size, value);
}

//верно ли, что массив B содержит каждый элемент массива A.
int Check_Every_A_Elements(int *a, int *b, size_t n, size_t k) {
    int i = 0;
    int j = 0;
    int counter = 0;
    while (i < k ) {
        if (b[i] == a[j]) {
            counter++;
            i++;
            j++;
        } else {
            i++;
            j++;
        }
    }
    return counter == n ? 1 : 0;
}

// возвращает значение 'истина', если subset является подмножеством set
// иначе - 'ложь'
bool ordered_array_set_isSubset(ordered_array_set subset, ordered_array_set
set) {
```

```
    return (Check_Every_A_Elements(subset.data, set.data, subset.size,
set.size)) ? true : false;
}

// 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)
{
    return (!memcmp(set1.data, set2.data, set1.size)) ? true : false;
}


// добавляет элемент value в множество set
void ordered_array_set_insert(ordered_array_set *set, int value) {
    if (ordered_array_set_in(set, value) != -1) {
        append_(set->data, (size_t *) set->size, value);
    }
}

// удаляет элемент value из множества set
void ordered_array_set_deleteElement(ordered_array_set *set, int value) {
    size_t pos = linearSearch_(set->data, set->size, value);
    deleteByPosSaveOrder_(set->data, (size_t *) set->size, pos);
}

int CheckSameElements(int const *a, int const *b, size_t elements_a, size_t
elements_b) {
    int counter = 0;
    for (size_t i = 0; i < elements_a; i++) {
        for (int j = 0; j < elements_b; j++) {
            if (a[i] == b[j])
                counter++;
        }
    }
    return counter;
}

// возвращает объединение множеств set1 и set2
ordered_array_set ordered_array_set_union(ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set set3;
    set3.size = set1.size + set2.size - CheckSameElements(set1.data, set2.data,
set1.size, set2.size);
    int i = 0;
    int j = 0;
    int l = 0;

    while ( l < set3.size ) {
        if (set2.size > j && set1.data[j] < set2.data[i]) {
            set3.data[l] = set1.data[j];
            j++;
        } else if (set1.data[j] == set2.data[i]) {
            set3.data[l] = set1.data[j];
            i++;
            j++;
        } else {
            set3.data[l] = set2.data[i];
            i++;
        }
        l++;
    }
    arraySort(set3.data, &set3.size);

    return set3;
```

```
}

// возвращает пересечение множеств set1 и set2
ordered_array_set ordered_array_set_intersection(ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set set3;
    set3.size = CheckSameElements(set1.data, set2.data, set1.size, set2.size);
    int i = 0;
    int j = 0;
    int l = 0;

    while ( l < set3.size ) {
        if (set2.data[i] < set1.data[j])
            i++;
        else if(set2.data[i] > set1.data[j])
            j++;
        else {
            set3.data[l++] = set2.data[i];
            i++;
            j++;
        }
    }
    for (size_t f = 0; f < set3.size; f++) {
        if (set3.data[f] == set3.data[f + 1])
            deleteByPosSaveOrder_(set3.data, &set3.size, f);
    }
    return set3;
}

int Check_Different_Elements(int const *a, int const *b, size_t elements_a,
size_t elements_b) {
    int counter = 0;
    for (int i = 0; i < elements_a; i++) {
        int flag = 0;
        for (int j = 0; j < elements_b; j++) {
            if (a[i] == b[j]) {
                flag = 1;
            }
        }
        if (!flag)
            counter++;
    }
    return counter;
}


// возвращает разность множеств set1 и set2
ordered_array_set ordered_array_set_difference(ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set set3;
    set3.size = Check_Different_Elements(set1.data, set2.data, set1.size,
set2.size);
    int i = 0;
    int j = 0;
    int l = 0;

    while ( l < set3.size ) {
        if (j >= set2.size || set1.data[i] < set2.data[j]) {
            set3.data[l] = set1.data[i];
            l++;
            i++;
        } else if (set1.data[i] > set2.data[j]) {
            j++;
        } else {
```

```
            i++;
            j++;
        }

    }
    return set3;
}


// возвращает симметрическую разность множеств set1 и set2
ordered_array_set ordered_array_set_symmetricDifference(ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set set3 = ordered_array_set_difference(set1, set2);
    ordered_array_set set4 = ordered_array_set_difference(set2, set1);
    ordered_array_set set_res = ordered_array_set_union(set3, set4);
    return set_res;
}

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

ordered_array_set ordered_array_set_universum_create (size_t size) {
    ordered_array_set universum_set;
    universum_set.size = size;
    int value = 1;
    for (size_t i = 0; i < size; i++) {
        universum_set.data[i] = value++;
    }
    return universum_set;
}

// Проверка элемента на строгое включение в множество
int isProperSubsetOrdered(ordered_array_set set1, ordered_array_set set2) {
    if (set1.size >= set2.size) {
        return 0; // Если размер первого множества больше или равен размеру
второго, нет
        //строгого включения
    }
    int i = 0, j = 0;
    while (i < set1.size && j < set2.size) {
        if (set1.data[i] == set2.data[j]) {
            i++;
            j++;
        } else if (set1.data[i] < set2.data[j]) {
            return 0; // Если первое множество содержит элемент, которого нет
во втором,
            //то нет строгого включения
        } else {
            j++;
        }
    }
    if (i == set1.size) {
        return 1; // Если все элементы первого множества входят во второе и
размеры
        //различны, есть строгое включение
    }
    return 0;
}
```

в) элементы множества A хранятся в массиве A, элементы которого типа boolean. Если i принадлежит A, то Ai=true, иначе Ai=false.

```c
#include <stdio.h>
#include <assert.h>

#include "bitset.h"


bitset bitset_create(unsigned setMaxValue) {
    assert (setMaxValue < 32);
    return (bitset) {0, setMaxValue};
}

int bitset_checkValue(bitset *a, unsigned value) {
    return value >= 0 && value <= a->maxValue;
}

bool bitset_in(bitset set, unsigned int value) {
    return set.values &= (1 << value) ? true : false;
}


bool bitset_isEqual(bitset set1, bitset set2) {
    return set1.values == set2.values && set1.maxValue == set2.maxValue ? true
: false;
}

bool bitset_isSubset(bitset subset, bitset set) {
    return (set.values & subset.values == set.values && set.maxValue ==
subset.maxValue) \
 && (set.values != subset.values && set.maxValue != subset.maxValue) ? true :
false;
}

void bitset_insert(bitset *set, unsigned int value) {
    set->values |= (1 << value);
}

void bitset_deleteElement(bitset *set, unsigned int value) {
    set->values |= ~(1 << value);
}

bitset bitset_union(bitset set1, bitset set2) {
    bitset set3 = {set1.values | set2.values , set1.maxValue | set2.maxValue};
    return set3;
}

bitset bitset_intersection(bitset set1, bitset set2) {
    assert (set1.maxValue == set2.maxValue);
    return (bitset) {set1.values & set2.values, set1.maxValue};
}

bitset bitset_difference(bitset set1, bitset set2) {
    bitset set3 = {set1.values & ~set2.values , set1.maxValue &
~set2.maxValue};
    return set3;
}

bitset bitset_symmetricDifference(bitset set1, bitset set2) {
    bitset set3 = {set1.values ^ set2.values , set1.maxValue ^ set2.maxValue};
    return set3;
}
```

```
bitset bitset_complement(bitset set) {
    bitset result;
    result.values = ~set.values & ((1 << (set.maxValue + 1)) - 1);
    result.maxValue = set.maxValue;
    return result;
}

void bitset_print(bitset set) {
    printf("{");
    int isEmpty = 1;
    for (int i = 0; i <= set.maxValue; ++i) {
        if (bitset_in(set, i)) {
            printf("%d, ", i);
            isEmpty = 0;
        }
    }
    if (isEmpty)
        printf("}\n");
    else
        printf("\b\b}\n");
}
```

Задание 4.

Решение задания 1:

```
int main() {
    ordered_array_set A;
    A.size = 4;

    ordered_array_set B;
    B.size = 3;

    ordered_array_set C;
    C.size = 5;

    printf("input A elements\n");
    inputArray_(A.data, A.size);

    printf("input B elements\n");
    inputArray_(B.data, B.size);

    printf("input C elements\n");
    inputArray_(C.data, C.size);

    ordered_array_set universum_set = ordered_array_set_universum_create(10);
    ordered_array_set set1 = ordered_array_set_difference(C, A);
    ordered_array_set set2 = ordered_array_set_intersection(A,
ordered_array_set_difference(B, C));
    ordered_array_set set3 = ordered_array_set_difference(set2,
ordered_array_set_intersection(B, A));
    ordered_array_set set4 = ordered_array_set_difference(universum_set, set3);
    ordered_array_set D = ordered_array_set_symmetricDifference(set4, set1);
    ordered_array_set_print(D);

    return 0;
}
```

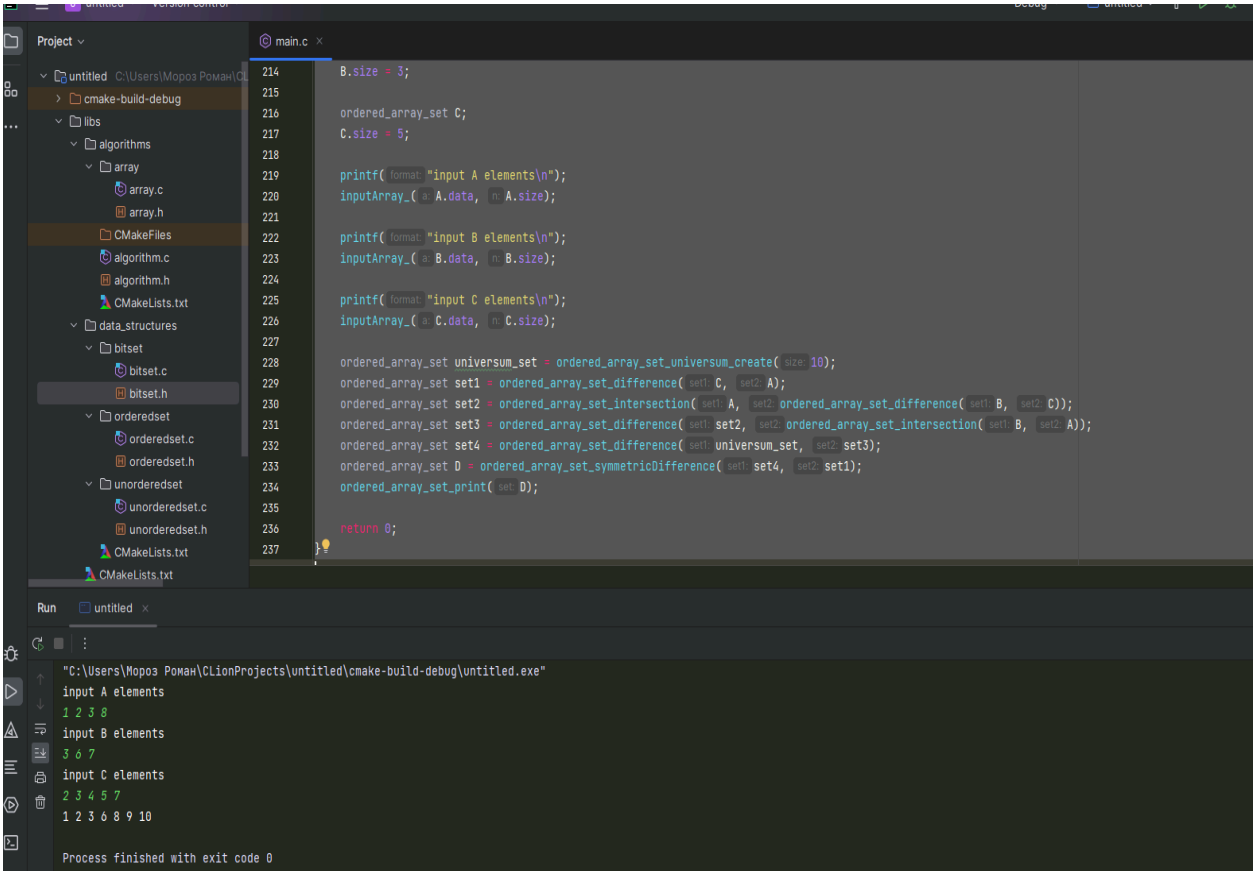
Решение задания 2:

```
int main() {
    ordered_array_set A;
    A.size = 7;
```

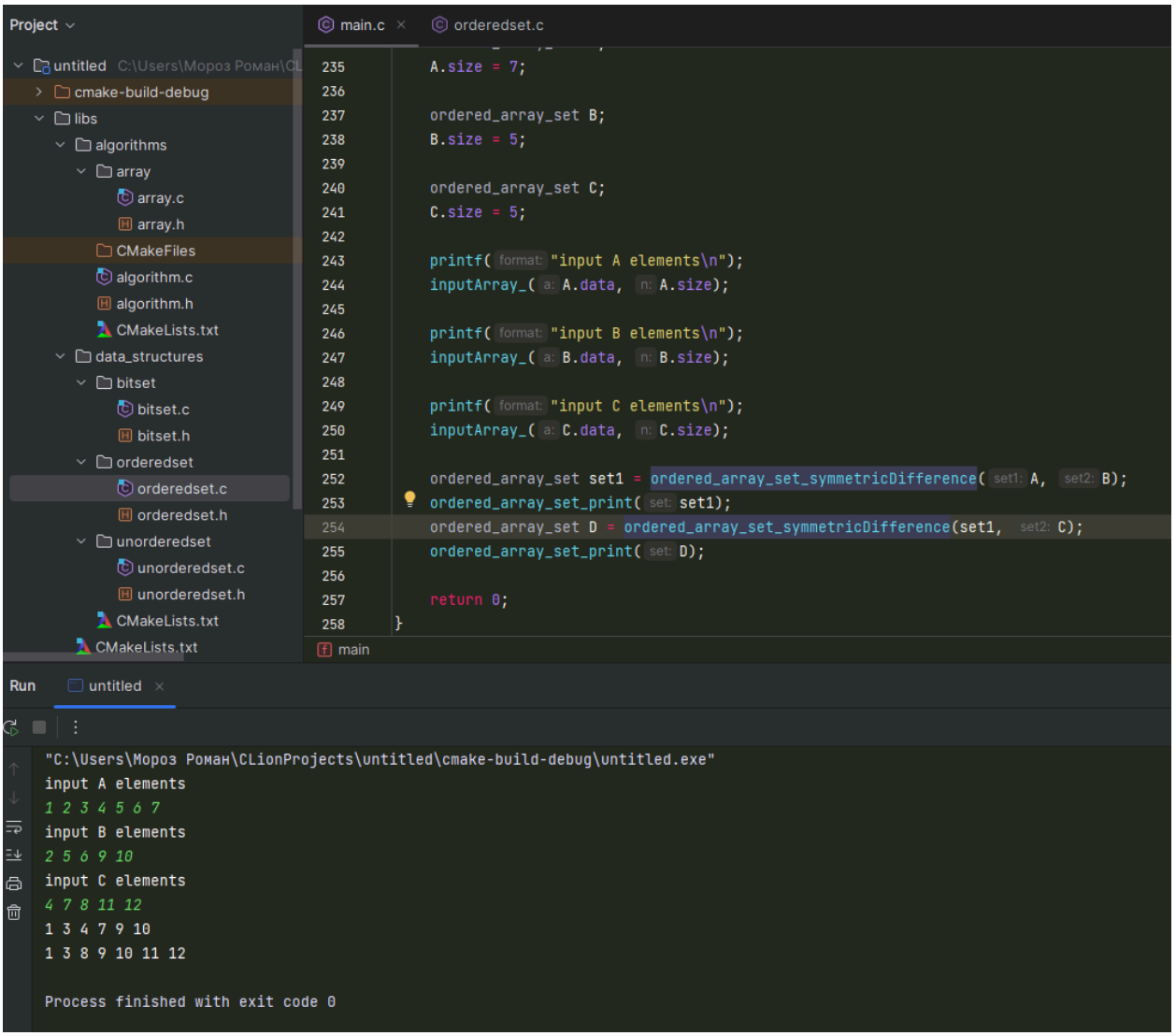
```c
    ordered_array_set B;
    B.size = 5;

    ordered_array_set C;
    C.size = 5;

    printf("input A elements\n");
    inputArray_(A.data, A.size);

    printf("input B elements\n");
    inputArray_(B.data, B.size);

    printf("input C elements\n");
    inputArray_(C.data, C.size);

    ordered_array_set set1 = ordered_array_set_symmetricDifference(A, B);
    ordered_array_set_print(set1);
    ordered_array_set D = ordered_array_set_symmetricDifference(set1, C);
    ordered_array_set_print(D);

    return 0;
}
```

Задание 5.

Задание 1:



Задание 2:



Вывод: изучили и научились использовать алгебру подмножеств, изучили различные способы представления множеств в памяти ЭВМ, научились программно реализовывать операции над множествами и выражения в алгебре подмножеств.