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

По дисциплине: «Дискретная математика»

Выполнил: студент группы ВТ-231

Борченко Александр Сергеевич

Проверили:

Островский Алексей Мячиславович

Рязанов Юрий Дмитриевич

Белгород 2024

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.

Задания

1. Вычислить значение выражения (см. “Варианты заданий”, п. а).

Во всех вариантах считать U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. Решение

изобразить с помощью кругов Эйлера.

2. Записать выражение в алгебре подмножеств, значение которого

при заданных множествах A, B и C равно множеству D (см. “Варианты

заданий”, п. б).

3. Программно реализовать операции над множествами, используя следующие способы представления множества в памяти ЭВМ:

а) элементы множества A хранятся в массиве A. Элементы массива A неупорядочены;

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

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

4. Написать программы для вычисления значений выражений

(см. “Задания”, п.1 и п.2).

5. Используя программы (см. “Задания”, п.4), вычислить значения

выражений (см. “Задания”, п.1 и п.2).

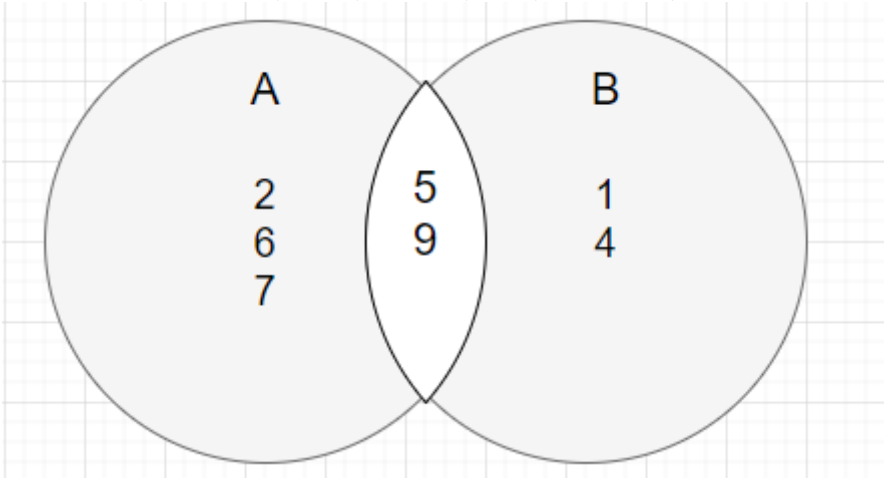
Вариант 2

Задание №1. Вычислить значение выражения (см. “Варианты заданий”, п. а). Во всех вариантах считать U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. Решение изобразить с помощью кругов Эйлера.

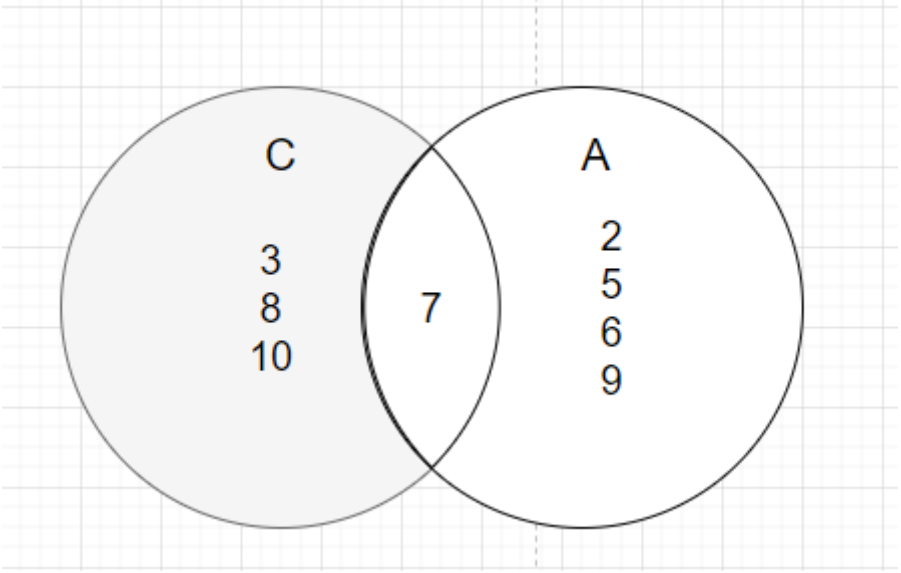
$D = B \cap (A \triangle B) \cup (C - A)$

A = {2,5,6,7,9} B = {1,4,5,9} C = {3,7,8,10}

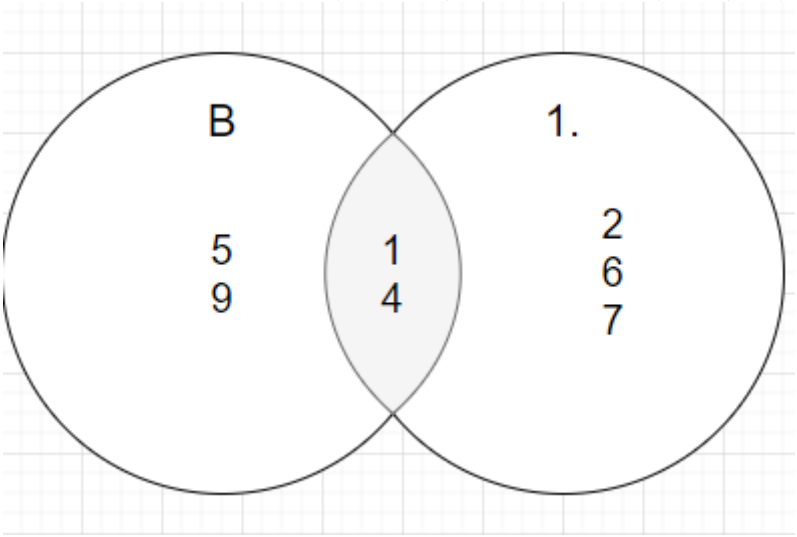
1. $A \triangle B = \{2,5,6,7,9\} \triangle \{1,4,5,9\} = \{1,2,4,6,7\}$



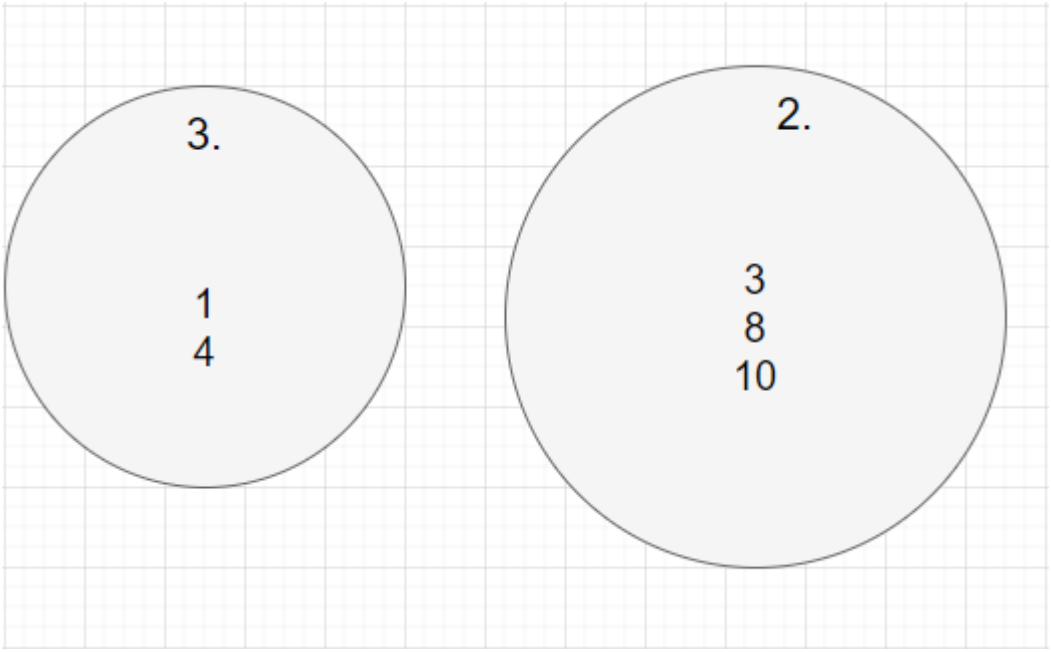
2. $C - A = \{3,7,8,10\} - \{2,5,6,7,9\} = \{3,8,10\}$



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



4. $B \cap (A \triangle B) \cup (C - A) = 3. \cup 2. = \{1,4\} \cup \{3,8,10\} = \{1,3,4,8,10\}$



Задание №2. Записать выражение в алгебре подмножеств, значение которого при заданных множествах A, B и C равно множеству D (см. “Варианты заданий”, п. б).

A = {1,2,3,8}

B = {3,6,7}

C = {2,3,4,5,7}

D = {1,3,4,5,6,8} = **(B-C) ∪ (A △ (C-B))**

Поэтапное решение:

1. Только в множестве B присутствует нужный мне элемент для множества D (6). Поэтому: вычту из множества B все лишние элементы, кроме 6. Из множества B вычитаю (операция разность) множество C, так как в этом множестве (в множестве C) есть повторяющиеся элементы множества B, кроме нужной мне 6:
   B-C = {3,6,7} – {2,3,4,5,7} = {6}
2. В множестве C присутствуют нужные мне элементы для множества D (4, 5). Избавлюсь от неудовлетворяющих элементов путем разности множеств C и B:
   C-B = {2,3,4,5,7} - {3,6,7} = {2,4,5}
3. В множестве A присутствуют нужные мне элементы для множества D (1,3). Также стоит заметить, что в итоговом множестве D нет элемента 2. Поэтому произведу симметрическую разность между элементом A и результатом второго действия (2.). Таким образом я получу удовлетворяющие элементы множества A и уберу ненужный элемент 2:
   A △ 2. = {1,2,3,8} △ {2,4,5} = {1,3,4,5,8}
4. Операцией объединение между результатом первого действия и третьего действия получу элементы множества D:
   1. ∪ 3. = {6} ∪ {1,3,4,5,8} = {1,3,4,5,6,8}

Задание №3 Программно реализовать операции над множествами, используя следующие способы представления множества в памяти ЭВМ.

а) Элементы множества A хранятся в массиве A. Элементы массива A неупорядочены;

```c
#ifndef C_UNORDERED_SET_H
#define C_UNORDERED_SET_H
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <assert.h>
#include <malloc.h>

typedef struct unordered_array_set {
    int *data;// элементы множества
    size_t size; // количество элементов в множестве
    size_t capacity; // максимальное количество элементов в множестве,
вместимость, мощность
} unordered_array_set;

unordered_array_set unordered_array_set_create(size_t capacity) {
    unordered_array_set set;
    set.data = (int *)malloc(capacity * sizeof(int));
    set.size = 0;
    set.capacity = capacity;
    return set;
}

size_t linearSearch(const int a[], const size_t n, int x) {
    for (size_t i = 0; i < n; i++)
        if (a[i] == x)
            return i;
    return n;
}

size_t unordered_array_set_in(unordered_array_set* set, int value) {
    return linearSearch(set -> data, set -> size, value);
}

bool unordered_array_set_isSubset(unordered_array_set subset,
unordered_array_set set) {
    for (size_t i = 0; i < subset.size; i++) {
        bool found = false;
        for (size_t j = 0; j < set.size; ++j) {
            if (subset.data[i] == set.data[j]) {
                found = true;
                break;
            }
        }

        return found;
    }
}

int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

bool unordered_array_set_isEqual(unordered_array_set set1,
unordered_array_set set2) {
    if (set1.size != set2.size) {
```

```
        return false;
    }

    qsort(set1.data, set1.size, sizeof(int), compare);
    qsort(set2.data, set2.size, sizeof(int), compare);

    for (size_t i = 0; i < set1.size; i++) {
        if (set1.data[i] != set2.data[i]) {
            return false;
        }
    }
    return true;
}

void unordered_array_set_isAbleAppend(unordered_array_set *set) {
    assert(set->size >= set->capacity);
}

void unordered_array_set_insert(unordered_array_set *set, int value) {
    if (set->size >= set->capacity) {
        size_t new_capacity = (set->capacity == 0) ? 1 : set->capacity * 2;
        int *new_data = realloc(set->data, new_capacity * sizeof(int));

        set->data = new_data;
        set->capacity = new_capacity;
    }

    set->data[set->size++] = value;
}

unordered_array_set unordered_array_set_create_from_array(const int *a,
size_t size) {
    unordered_array_set set = {NULL, 0, 0};
    for (size_t i = 0; i < size; ++i) {
        unordered_array_set_insert(&set, a[i]);
    }
    return set;
}

void unordered_array_set_deleteElement(unordered_array_set *set, int value) {
    for (size_t i = 0; i < set->size; ++i) {
        if (set->data[i] == value) {
            for (size_t j = i; j < set->size - 1; ++j) {
                set->data[j] = set->data[j + 1];
            }
            set->size--;
            return;
        }
    }
}

unordered_array_set unordered_array_set_union(unordered_array_set set1,
unordered_array_set set2) {
    unordered_array_set result = {NULL, 0, 0};

    for (size_t i = 0; i < set1.size; i++) {
        if (unordered_array_set_in(result, set1.data[i]) == result.size) {
            unordered_array_set_insert(&result, set1.data[i]);
        }
    }

    for (size_t i = 0; i < set2.size; i++) {
        if (unordered_array_set_in(result, set2.data[i]) == result.size) {
            unordered_array_set_insert(&result, set2.data[i]);
```

```
        }
    }

    return result;
}

unordered_array_set unordered_array_set_intersection(unordered_array_set
set1, unordered_array_set set2) {
    unordered_array_set intersection = {NULL, 0, 0};
    for (size_t i = 0; i < set1.size; i++) {
        if (unordered_array_set_in(set2, set1.data[i]) != set1.size) {
            unordered_array_set_insert(&intersection, set1.data[i]);
        }
    }
    return intersection;
}

unordered_array_set unordered_array_set_difference(unordered_array_set set1,
unordered_array_set set2) {
    unordered_array_set difference = {NULL, 0, 0};
    for (size_t i = 0; i < set1.size; i++) {
        if (unordered_array_set_in(set2, set1.data[i]) == set1.size) {
            unordered_array_set_insert(&difference, set1.data[i]);
        }
    }
    return difference;
}

unordered_array_set unordered_array_set_complement(unordered_array_set set,
unordered_array_set universumSet) {
    unordered_array_set complement = {NULL, 0, 0};
    for (size_t i = 0; i < universumSet.size; i++) {
        if (unordered_array_set_in(set, universumSet.data[i]) == set.size) {
            unordered_array_set_insert(&complement, universumSet.data[i]);
        }
    }
    return complement;
}

void unordered_array_set_delete(unordered_array_set set) {
    free(set.data);
}

unordered_array_set
unordered_array_set_symmetricDifference(unordered_array_set set1,
unordered_array_set set2) {
    unordered_array_set diff1 = unordered_array_set_difference(set1, set2);
    unordered_array_set diff2 = unordered_array_set_difference(set2, set1);
    unordered_array_set symmetric_difference =
unordered_array_set_union(diff1, diff2);
    unordered_array_set_delete(diff1);
    unordered_array_set_delete(diff2);
    return symmetric_difference;
}

void unordered_array_set_print(unordered_array_set set) {
    printf("{ ");
    for (size_t i = 0; i < set.size; ++i) {
        printf("%d ", set.data[i]);
    }
    printf("}\n");
}

#endif //C_UNORDERED_SET_H
```

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

```
#ifndef БИБЛИОТЕКИ_ORDERED_SET_H
#define БИБЛИОТЕКИ_ORDERED_SET_H
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <assert.h>
#include <malloc.h>
#include "array.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,
вместимость, мощность
} ordered_array_set;

ordered_array_set ordered_array_set_create(size_t capacity) {
    ordered_array_set set;
    set.data = (int *) malloc(capacity * sizeof(int));
    set.size = 0;
    set.capacity = capacity;
    return set;
}

ordered_array_set ordered_array_set_create_from_array(const int *a, size_t
size) {
    ordered_array_set set;
    set.data = (int *) malloc(size * sizeof(int));
    for (size_t i = 0; i < size; ++i) {
        set.data[i] = a[i];
    }
    set.size = size;
    set.capacity = size;
    return set;
}

size_t ordered_array_set_in(ordered_array_set *set, int value) {
    for (size_t i = 0; i < set->size; ++i) {
        if (set->data[i] == value) {
            return i;
        }
    }
    return set->size;
}

bool ordered_array_set_isEqual(ordered_array_set set1, ordered_array_set
set2) {
    if (set1.size != set2.size) {
        return false;
    }
    for (size_t i = 0; i < set1.size; ++i) {
        if (set1.data[i] != set2.data[i]) {
            return false;
        }
    }
    return true;
}

bool ordered_array_set_isSubset(ordered_array_set subset, ordered_array_set
set) {
```

```
    for (size_t i = 0; i < subset.size; ++i) {
        if (ordered_array_set_in(&set, subset.data[i]) == set.size) {
            return false;
        }
    }
    return true;
}

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) {
    size_t i;
    for (i = set->size; i > 0 && set->data[i - 1] > value; --i) {
        set->data[i] = set->data[i - 1];
    }
    set->data[i] = value;
    ++set->size;
}

void ordered_array_set_deleteElement(ordered_array_set *set, int value) {
    size_t i = ordered_array_set_in(set, value);
    if (i < set->size) {
        for (; i < set->size - 1; ++i) {
            set->data[i] = set->data[i + 1];
        }
        set->size--;
    }
}

ordered_array_set ordered_array_set_union(ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result = ordered_array_set_create(set1.size +
set2.size);
    size_t i = 0, j = 0, k = 0;
    while (i < set1.size && j < set2.size) {
        if (set1.data[i] < set2.data[j]) {
            result.data[k++] = set1.data[i++];
        } else if (set2.data[j] < set1.data[i]) {
            result.data[k++] = set2.data[j++];
        } else {
            result.data[k++] = set1.data[i++];
            ++j;
        }
    }
    while (i < set1.size) {
        result.data[k++] = set1.data[i++];
    }
    while (j < set2.size) {
        result.data[k++] = set2.data[j++];
    }
    result.size = k;
    return result;
}

ordered_array_set ordered_array_set_intersection(ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result = ordered_array_set_create(set1.size);
    size_t i = 0, j = 0, k = 0;
    while (i < set1.size && j < set2.size) {
        if (set1.data[i] < set2.data[j]) {
            i++;
        } else if (set2.data[j] < set1.data[i]) {
```

```
            j++;
        } else {
            result.data[k++] = set1.data[i++];
            j++;
        }
    }
    result.size = k;
    return result;
}

ordered_array_set ordered_array_set_difference(ordered_array_set set1,
ordered_array_set set2) {
    ordered_array_set result = ordered_array_set_create(set1.size);
    size_t i = 0, j = 0, k = 0;
    while (i < set1.size && j < set2.size) {
        if (set1.data[i] < set2.data[j]) {
            result.data[k++] = set1.data[i++];
        } else if (set2.data[j] < set1.data[i]) {
            j++;
        } else {
            i++;
            j++;
        }
    }
    while (i < set1.size) {
        result.data[k++] = set1.data[i++];
    }
    result.size = k;
    return result;
}

ordered_array_set ordered_array_set_symmetricDifference(ordered_array_set
set1, ordered_array_set set2) {
    ordered_array_set result1 = ordered_array_set_difference(set1, set2);
    ordered_array_set result2 = ordered_array_set_difference(set2, set1);
    ordered_array_set result = ordered_array_set_union(result1, result2);
    free(result1.data);
    free(result2.data);
    return result;
}

ordered_array_set ordered_array_set_complement(ordered_array_set set,
ordered_array_set universumSet) {
    ordered_array_set result = ordered_array_set_difference(universumSet,
set);
    return result;
}

void ordered_array_set_print(ordered_array_set set) {
    printf("{ ");
    for (size_t i = 0; i < set.size; ++i) {
        printf("%d ", set.data[i]);
    }
    printf("}\n");
}

void ordered_array_set_delete(ordered_array_set set) {
    free(set.data);
}

#endif //БИБЛИОТЕКИ_ORDERED_SET_H
```

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

```
#ifndef БИБЛИОТЕКИ_BIT_SET_H
#define БИБЛИОТЕКИ_BIT_SET_H
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <assert.h>
#include <malloc.h>

typedef struct bitset {
    uint32_t *values;
    uint32_t maxValue;
} bitset;

bitset bitset_create(unsigned maxValue) {
    bitset set;
    set.maxValue = maxValue;

    uint32_t arraySize = (maxValue + 31) / 32;

    set.values = (uint32_t *)calloc(arraySize, sizeof(uint32_t));

    return set;
}

bool bitset_in(bitset set, uint32_t value) {
    if (value > set.maxValue) {
        return false;
    }
    uint32_t index = value / 32;
    uint32_t bitOffset = value % 32;

    return (set.values[index] & (1 << bitOffset)) != 0;
}

void bitset_insert(bitset *set, uint32_t value) {
    if (value > set->maxValue) {
        return;
    }

    uint32_t index = value / 32;
    uint32_t bitOffset = value % 32;

    set->values[index] |= (1 << bitOffset);
}

bool bitset_isEqual(bitset set1, bitset set2) {
    if (set1.maxValue != set2.maxValue) {
        return false;
    }

    uint32_t arraySize = (set1.maxValue + 31) / 32;
    for (uint32_t i = 0; i < arraySize; ++i) {
        if (set1.values[i] != set2.values[i]) {
            return false;
        }
    }
    return true;
}

bool bitset_isSubset(bitset set1, bitset set2) {
```

```
    if (set1.maxValue > set2.maxValue) {
        return false;
    }

    uint32_t arraySize = (set1.maxValue + 31) / 32;
    for (uint32_t i = 0; i < arraySize; ++i) {
        if ((set1.values[i] & ~set2.values[i]) != 0) {
            return false;
        }
    }
    return true;
}

void bitset_deleteElements(bitset *set) {
    free(set->values);
    set->values = NULL;
}

bitset bitset_union(bitset set1, bitset set2) {
    bitset result = bitset_create(set1.maxValue > set2.maxValue ?
set1.maxValue : set2.maxValue);

    for (uint32_t i = 0; i < result.maxValue; ++i) {
        if (bitset_in(set1, i) || bitset_in(set2, i)) {
            bitset_insert(&result, i);
        }
    }

    return result;
}

bitset bitset_intersection(bitset set1, bitset set2) {
    bitset result = bitset_create(set1.maxValue < set2.maxValue ?
set1.maxValue : set2.maxValue);

    for (uint32_t i = 0; i < result.maxValue; ++i) {
        if (bitset_in(set1, i) && bitset_in(set2, i)) {
            bitset_insert(&result, i);
        }
    }

    return result;
}

bitset bitset_difference(bitset set1, bitset set2) {
    bitset result = bitset_create(set1.maxValue);

    for (uint32_t i = 0; i < set1.maxValue; ++i) {
        if (bitset_in(set1, i) && !bitset_in(set2, i)) {
            bitset_insert(&result, i);
        }
    }

    return result;
}

bitset bitset_symmetricDifference(bitset set1, bitset set2) {
    bitset result = bitset_create(set1.maxValue > set2.maxValue ?
set1.maxValue : set2.maxValue);

    for (uint32_t i = 0; i < result.maxValue; ++i) {
        if ((bitset_in(set1, i) && !bitset_in(set2, i)) || (!bitset_in(set1,
i) && bitset_in(set2, i))) {
            bitset_insert(&result, i);
```

```
        }
    }

    return result;
}

bitset bitset_complement(bitset set) {
    bitset result = bitset_create(set.maxValue);

    for (uint32_t i = 0; i < set.maxValue; ++i) {
        if (!bitset_in(set, i)) {
            bitset_insert(&result, i);
        }
    }

    return result;
}


void bitset_print(bitset set) {
    printf("{ ");
    for (uint32_t i = 0; i <= set.maxValue; ++i) {
        if (bitset_in(set, i)) {
            printf("%u ", i);
        }
    }
    printf("}\n");
}

#endif //БИБЛИОТЕКИ_BIT_SET_H
```

Задание №4. Написать программы для вычисления значений выражений (см. “Задания”, п.1 и п.2).

Задание №4. П1 (программа к заданию №1).

```c
#include <stdio.h>
#include "bit_set.h"

// Добавляет значение с клавиатуры в множетво set, размера size
void fill_bit_set(bitset *set, size_t size) {
    int k;
    for(size_t i = 0; i < size; i++) {
        scanf("%d", &k);

        bitset_insert(set, k);
    }
}

int main() {
    // A = {2,5,6,7,9} (размер 5, макс значение - до 10 (9).
    bitset A = bitset_create(11);
    fill_bit_set(&A, 5);
    // B = {1,4,5,9} (размер 4, макс значение - до 10 (9)
    bitset B = bitset_create(11);
    fill_bit_set(&B, 4);
    // C = {3,7,8,10} (размер 4, макс значение - до 11 (10)
    bitset C = bitset_create(11);
    fill_bit_set(&C, 4);
    //1) Нахожу симметрическую разность A и B
    bitset A_B_sym_diff = bitset_symmetricDifference(A, B);
    //2) Нахожу разность C и A
    bitset C_A_diff = bitset_difference(C, A);
    //3) Нахожу пересечение B и результата 1 действия
    bitset B_1_interesection = bitset_intersection(B, A_B_sym_diff);
    //4) Нахожу объединение действия 3 и 2, что и будет множеством D
    bitset D = bitset_union(C_A_diff,B_1_interesection);
    //Вывод ответа (множества D)
    bitset_print(D);

    return 0;
}
```

Задание №4. П2 (программа к заданию №2).

```c
#include <stdio.h>
#include "bit_set.h"

// Добавляет значение с клавиатуры в множетво set, размера size
void fill_bit_set(bitset *set, size_t size) {
    int k;
    for(size_t i = 0; i < size; i++) {
        scanf("%d", &k);

        bitset_insert(set, k);
    }
}

int main() {
    // A = {1,2,3,8} (размер 4, макс значение - до 9 (8).
    bitset A = bitset_create(9);
    fill_bit_set(&A, 4);
    // B = {3,6,7} (размер 3, макс значение - до 8 (7)
    bitset B = bitset_create(8);
    fill_bit_set(&B, 3);
    // C = {2,3,4,5,7} (размер 5, макс значение - до 8 (7)
    bitset C = bitset_create(11);
    fill_bit_set(&C, 5);
    //1) Нахожу разность между множествами B и C
    bitset B_C_diff = bitset_difference(B,C);
    //2) Нахожу разность между множествами C и B
    bitset C_B_diff = bitset_difference(C,B);
    //3) Нахожу симметрическую разность между A и 2 действием
    bitset A_2_sym_diff = bitset_symmetricDifference(A, C_B_diff);
    //4) Нахожу результат объединения 1 и 3 действия, что и будет являться
множеством D
    bitset D = bitset_union(B_C_diff, A_2_sym_diff);

    //Вывод ответа (множества D, результат должен быть {1,3,4,5,6,8})
    bitset_print(D);

    return 0;
}
```

Задание №5. Используя программы (см. “Задания”, п.4), вычислить значения выражений (см. “Задания”, п.1 и п.2).

П1 (программа к заданию №1). Результат выполнения программы:

```
"C:\Users\Александр\CLionProjects\ДМ (1.1 лаба)\Задание п1 Вариант 2.exe"
2 5 6 7 9
1 4 5 9
3 7 8 10
{ 1 3 4 8 10 }

Process finished with exit code 0
```

Ответ совпадает.

П2 (программа к заданию №2). Результат выполнения программы:

(D = {1,3,4,5,6,8} по условию из учебника)

```
"C:\Users\Александр\CLionProjects\ДМ (1.1 лаба)\Задание п2 Вариант 2.exe"
1 2 3 8
3 6 7
2 3 4 5 7
{ 1 3 4 5 6 8 }

Process finished with exit code 0
```

Ответ совпал.

Вывод: в ходе выполнения лабораторной работы были получены навыки использования алгебры подмножеств в памяти ЭВМ, реализованы программно операции над множествами и выражениями в алгебре подмножеств