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»
по теме: «Операции над множествами»

Выполнил: ст. группы ВТ-231
Кисиль Николай Владимирович

Проверили:
Рязанов Юрий Дмитриевич
Островский Алексей Мечиславович

Белгород, 2024 г.

Цель: изучить и научиться использовать операции над множествами, изучить различные способы представления множеств в памяти ЭВМ, научиться программно реализовывать операции над множествами и вычислять значения теоретико-множественных выражений.

Задания

1. Вычислить значение выражения (см. ―Варианты заданий‖, п. а).

Решение изобразить с помощью кругов Эйлера.

2. Записать теоретико-множественное выражение, 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 ∈ A, то Ai = true, иначе Ai= false.

4. Написать программы для вычисления значений выражений (см. ―Задания, п.1 и п.2).

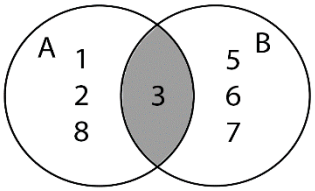
5. Используя программы (см. ―Задания, п.4), вычислить значения выражений (см. ―Задания, п.1 и п.2).

Вариант 7

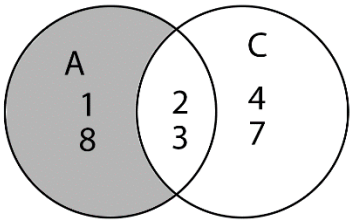
Задание №1

D = (A ∩ B) ∪ (A-C) ∪ (B-C)
A = {1, 2, 3, 8} B = {3, 5, 6, 7} C = {2, 3, 4, 7}

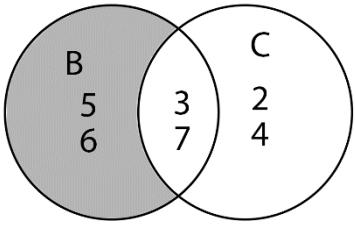
1. A ∩ B = {1, 2, 3, 8} ∩ {3, 5, 6, 7} = {3}



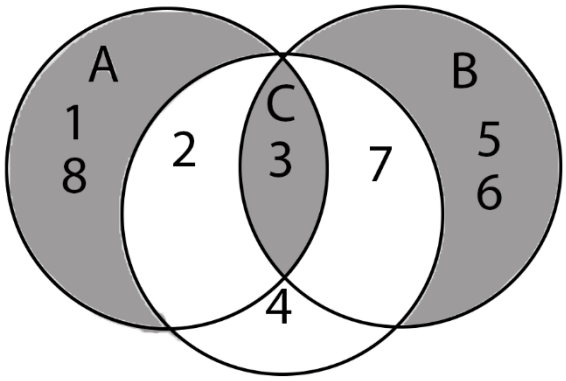
2. A – C = {1, 2, 3, 8} – {2, 3, 4, 7} = {1, 8}



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



4. (A ∩ B) ∪ (A-C) ∪ (B-C) = {3} ∪ {1, 8} ∪ {5, 6} = {1, 3, 5, 6, 8}



Задание №2

A = {1, 2, 3, 4, 5, 6, 7} B = {2, 5, 6, 9, 10} C = {4, 7, 8, 11, 12}

D = {2, 4, 5, 6, 7}

Ответ: A ∩ (B ∪ C)

1. A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)
2. A ∩ B = {1, 2, 3, 4, 5, 6, 7} ∩ {2, 5, 6, 9, 10} = {2, 5, 6}
3. A ∩ C = {1,2,3,4,5,6,7} ∩ {4,7,8,11,12} = {4, 7}
4. (A ∩ B) ∪ (A ∩ C) = {2, 5, 6} ∪ {4, 7} = {2, 4, 5, 6, 7}

Задание №3 а)

```c
#ifndef INC_UNORDERED_ARRAY_SET_H
#define INC_UNORDERED_ARRAY_SET_H

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

typedef struct unordered_array_set {
    int *data;
    size_t size;
    size_t capacity;
} unordered_array_set;

size_t unordered_array_set_in(unordered_array_set set, int value) {
    for (size_t i = 0; i < set.size; i++) {
        if (set.data[i] == value) {
            return i;
        }
    }
    return set.size;
}

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;
}

bool unordered_array_set_isEqual(unordered_array_set set1,
unordered_array_set set2) {
    if (set1.size != set2.size) {
        return 0;
    }
    qsort(set1.data, set1.size, sizeof(int), compare);
    qsort(set2.data, set2.size, sizeof(int), compare);
    return memcmp(set1.data, set2.data, sizeof(int) * set1.size) == 0;
}


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
```

Задание №3 б)

```c
#ifndef INC_ORDERED_ARRAY_SET_H
#define INC_ORDERED_ARRAY_SET_H

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



typedef struct ordered_array_set {
    int *data;
    size_t size;
    size_t capacity;
} ordered_array_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;
}

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;
}

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;
}

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
```

Задание №3 в)

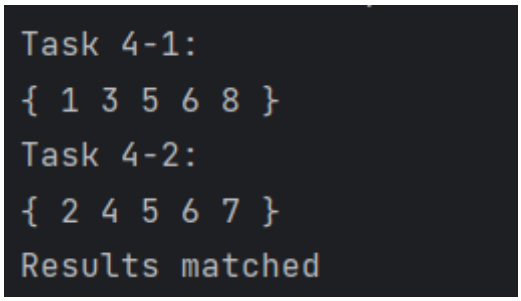
```c
#ifndef INC_BITSET_H
#define INC_BITSET_H

#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;

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);
}

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
```

Задание №4

```c
#include "ordered_array_set.h"

void task4_1() {
    printf("Task 4-1:\n");
    ordered_array_set A = ordered_array_set_create_from_array((int []) {1, 2,
3, 8}, 4);
    ordered_array_set B = ordered_array_set_create_from_array((int []) {3, 5,
6, 7}, 4);
    ordered_array_set C = ordered_array_set_create_from_array((int []) {2, 3,
4, 7}, 4);

    ordered_array_set f_action = ordered_array_set_intersection(A, B);
    ordered_array_set s_action = ordered_array_set_difference(A, C);
    ordered_array_set th_action = ordered_array_set_difference(B, C);
    ordered_array_set D = ordered_array_set_union(f_action, s_action);
    D = ordered_array_set_union(D, th_action);

    ordered_array_set_print(D);

    ordered_array_set_delete(f_action);
    ordered_array_set_delete(s_action);
    ordered_array_set_delete(th_action);
}

void task4_2() {
    printf("Task 4-2:\n");
    ordered_array_set A = ordered_array_set_create_from_array((int [])
{1,2,3,4,5,6,7}, 7);
    ordered_array_set B = ordered_array_set_create_from_array((int [])
{2,5,6,9,10}, 5);
    ordered_array_set C = ordered_array_set_create_from_array((int [])
{4,7,8,11,12}, 5);
    ordered_array_set D = ordered_array_set_create_from_array((int [])
{2,4,5,6,7}, 5);

    ordered_array_set f_action = ordered_array_set_intersection(A, B);
    ordered_array_set s_action = ordered_array_set_intersection(A, C);
    ordered_array_set result = ordered_array_set_union(f_action, s_action);

    ordered_array_set_print(result);

    if(ordered_array_set_isEqual(result, D)) {
        printf("Results matched");
    } else {
        printf("Results didn`t matched");
    }


    ordered_array_set_delete(f_action);
    ordered_array_set_delete(s_action);
    ordered_array_set_delete(result);
}

int main() {
    task4_1();
    task4_2();

    return 0;
}
```

Задание №5



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