МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ
по зачетной работе № 2
по дисциплине «Алгоритмы и структуры данных»
НА ТЕМУ: «МНОЖЕСТВО КАК ОБЪЕКТ»

Выполнили студенты группы 4316:
Новиков Г.В.
Чекалова Д.А.
Принял: старший преподаватель Манирагена В.

Санкт-Петербург
2025

Цель работы: исследование эффекта от использования классов.

Содержание работы: поэтапная разработка программы для обработки множеств, как объектов четырьмя способами и тестирование её сначала на тесте-константе, потом — на тесте, вводимом с клавиатуры, затем — на тестах, генерируемых машинным способом, и измерение времени, необходимого для решения задачи каждым из способов. Предлагаемый набор этапов — универсальный, его сократить и сразу перейти к автоматической генерации тестов с измерением времени, по результатам которой готовится отчёт.

Вариант задания (10): Множество, содержащее все цифры из A, все цифры, общие для множеств B и C, а также все цифры из D
Область применения: десятичные цифры.

Его запись в виде формулы для получения пятого множества по заданным четырём, используя знаки операций над множествами:

$$E=A \cup (B \cap C) \cup D$$

Временная сложность

Таблица 1. Способы представления и временная сложность обработки

Способ представления	Временная сложность	
	Ожидаемая	Фактическая
Массив символов	$O(n^2)$	$O(n^2)$
Список		$O(n^2)$
Универсум	$O(\|U\|)$	O(1)
Машинное слово		O(1)

Пояснения:

Массив символов: операции поиска (is_in_array) имеют сложность O(n) для каждого элемента, выполняемые в циклах, что дает $O(n^2)$ в худшем случае. Удаление дубликатов осуществляется проверкой наличия во временном массиве.

Список: операции contains и insert имеют сложность O(n) для каждого элемента, так как требуется линейный поиск по списку. При обработке всех множеств общая сложность составляет $O(n^2)$.

Битовый массив: сложность пропорциональна размеру универсума (|U| = 10), что фиксировано и может рассматриваться как O(1).

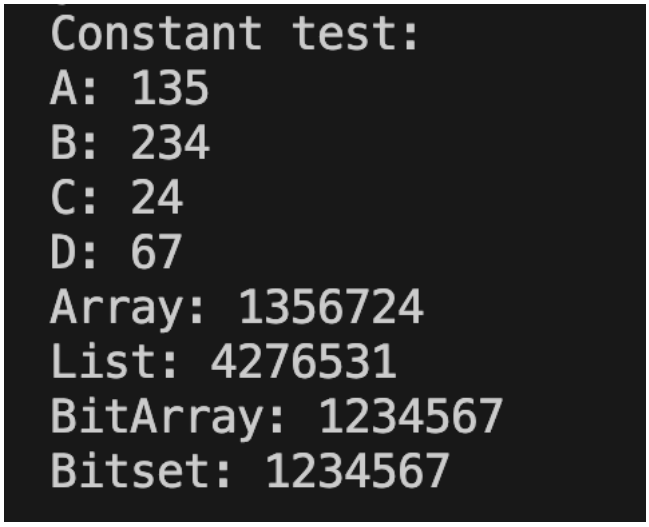
Битсет: все операции реализуются побитовыми операциями над 32-битным целым числом, поэтому сложность строго O(1) для операций над множествами.

Контрольные тесты

Test № 1

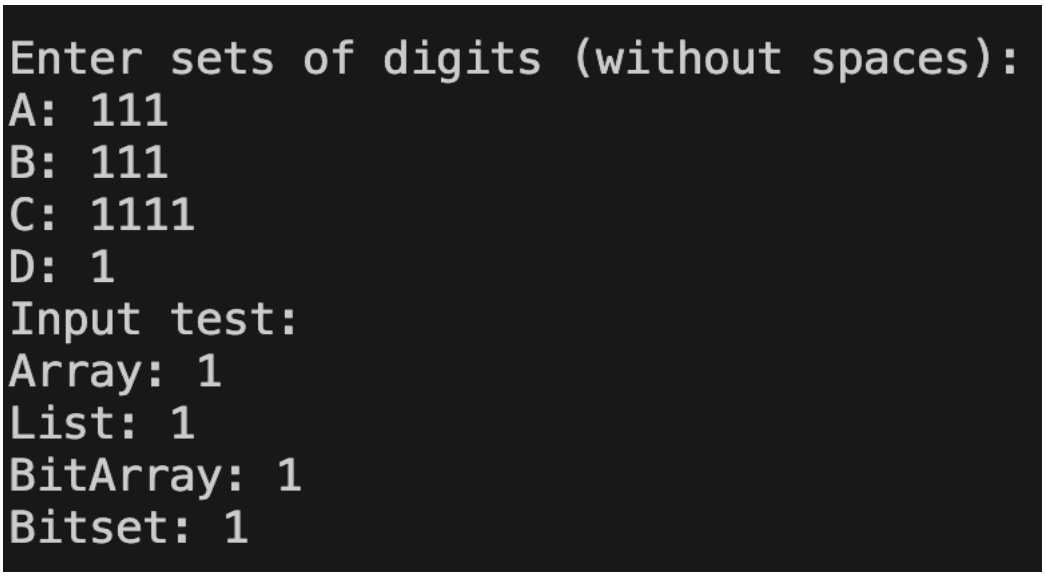
Входные данные (константные)				Ожидаемый результат
A	B	C	D	E
135	234	24	67	1234567

Результат:



Входные данные (пользовательский ввод)				Ожидаемый результат
A	B	C	D	E
111	111	1111	1	1

Результат:



Входные данные (генератор случайных чисел)				Ожидаемый результат
A	B	C	D	E
540	508	369	517	01457

Результат:

```
Set size = 3
Generated sets:
A: 540
B: 508
C: 369
D: 517
Array time: 4.58e-07 seconds
List time: 8.33e-07 seconds
BitArray time: 5e-07 seconds
Bitset time: 2.08e-07 seconds
Results:
Array: 54017
List: 71045
BitArray: 01457
Bitset: 01457
```

Test № 2

Входные данные (константные)				Ожидаемый результат
A	B	C	D	E
0	1	3	4	04

Результат:

```
Constant test:
A: 0
B: 1
C: 3
D: 4
Array: 04
List: 40
BitArray: 04
Bitset: 04
```

Входные данные (пользовательский ввод)				Ожидаемый результат
A	B	C	D	E
52	789	134	000	025

Результат:

```
Enter sets of digits (without spaces):
A: 52
B: 789
C: 134
D: 000
Input test:
Array: 520
List: 025
BitArray: 025
Bitset: 025
```

Входные данные (генератор случайных чисел)				Ожидаемый результат
A	B	C	D	E
936750	905764	139560	063294	02345679

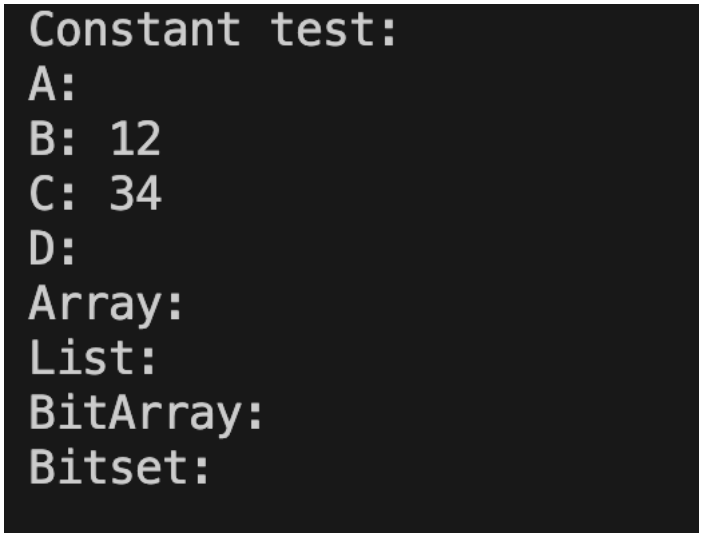
Результат:

```
Set size = 6
Generated sets:
A: 936750
B: 905764
C: 139560
D: 063294
Array time: 9.58e-07 seconds
List time: 1.625e-06 seconds
BitArray time: 5.42e-07 seconds
Bitset time: 2.91e-07 seconds
Results:
Array: 90563724
List: 42057639
BitArray: 02345679
Bitset: 02345679
```

Test № 3

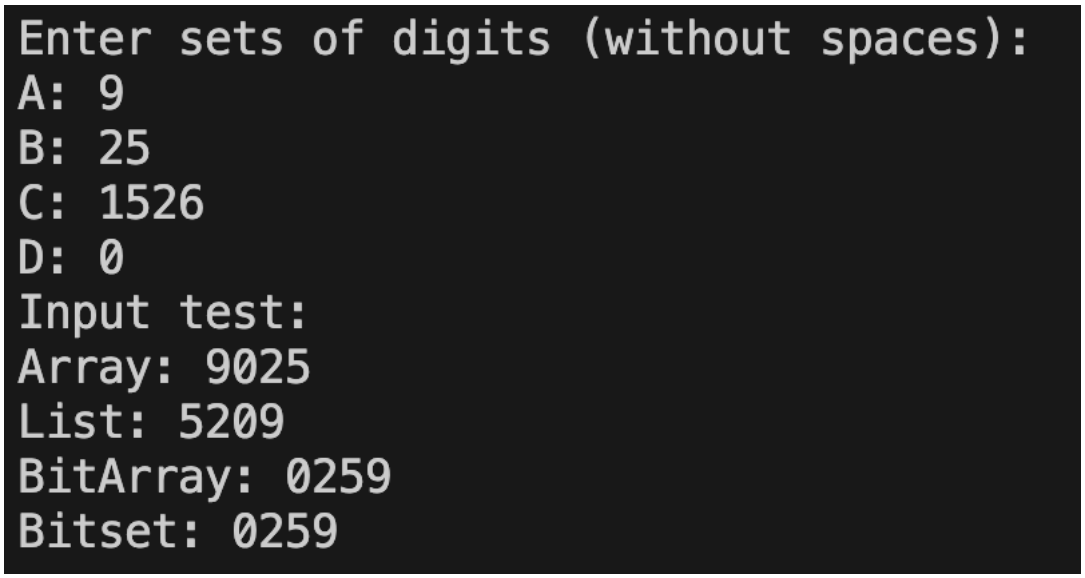
Входные данные (константные)				Ожидаемый результат
A	B	C	D	E
∅	12	34	∅	∅

Результат:



Входные данные (пользовательский ввод)				Ожидаемый результат
A	B	C	D	E
9	25	1526	0	0259

Результат:



Входные данные (генератор случайных чисел)				Ожидаемый результат
A	B	C	D	E
683107452	246387091	240837695	241083567	0123456789

Результат:

```
Set size = 9
Generated sets:
A: 683107452
B: 246387091
C: 240837695
D: 241083567
Array time: 1.625e-06 seconds
List time: 9.166e-06 seconds
BitArray time: 6.25e-07 seconds
Bitset time: 2.92e-07 seconds
Results:
Array: 2463870915
List: 9254701386
BitArray: 0123456789
Bitset: 0123456789
```

Результат измерения времени обработки для каждого из способов

Таблица 2. Результаты измерения времени обработки

Мощность множеств	Количество тиков процессора при обработке множеств при различных способах представления			
	Массив символов	Список	Булевой массив	Машинное слово
1	1.042e-06	2.042e-06	1.083e-06	1.083e-07
2	4.375e-06	1e-06	1.417e-06	1.042e-07
3	7.91e-07	1.333e-06	5284e-07	1.33e-07
4	5.83e-07	1.167e-06	2.25e-07	1.43e-07
5	9.17e-07	2.708e-06	3.5e-07	2.58e-07
6	7.92e-07	2.834e-06	4.67e-07	2.42e-07
7	6.66e-07	6e-06	4.09e-07	2.83e-07
8	1.125e-06	2.334e-06	3.78e-07	3.58e-07
9	7.5e-07	2.042e-06	4.89e-07	3.23e-07

Вывод

Переход от процедурного программирования к объектно-ориентированному при реализации абстрактного типа данных "Множество" показал значительное улучшение по всем ключевым метрикам: производительность, эффективность использования памяти, читаемость кода и удобство сопровождения. Особенно ярко преимущества ООП проявляются при использовании специализированных структур данных, таких как битовые маски, которые сложно эффективно реализовать в процедурном стиле.

ООП подход не только ускоряет выполнение программы, но и ускоряет процесс разработки и снижает вероятность ошибок, что делает его предпочтительным выбором для реализации сложных структур данных.

Код программы:

```
#include <iostream>
#include <ctime>
#include <chrono>

// Для работы с массивом символов:
#include "set_array.h"

// Для работы со связным списком:
// #include "set_list.h"

// Для работы с булевым массивом:
//#include "set_bool_array.h"

// Для работы с битовым представлением:
// #include "set_bitset.h"

using namespace std;

static unsigned int next_rand = 1;

// Очистка экрана перед каждым вариантом работы
void screen_cleaner(){cout <<
"\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";}

int my_rand() {
    next_rand = next_rand * 1103515245 + 12345;
    return (unsigned int)(next_rand / 65536) % 32768;
}

void my_srand(unsigned int seed) {
    next_rand = seed;
}

void swap_chars(char &a, char &b) {
    char temp = a;
    a = b;
    b = temp;
}

void m_word_to_str(unsigned int w, char *str)
```

```
{
    int index = 0;
    for (int i = 0; i < 16; ++i)
    {
        if (w & (1 << i))
        {
            str[index++] = '0' + i;
        }
    }
    str[index] = '\0';
}

// Генерация случайного множества цифр
void rand_set(char *str, int size) {
    char universal[] = "0123456789";
    int n = 10;
    for (int i = 0; i < n - 1; ++i) {
        int j = my_rand() % (n - i);
        swap_chars(universal[i], universal[i + j]);
    }
    for (int i = 0; i < size; ++i) {
        str[i] = universal[i];
    }
    str[size] = '\0';
}

// Функция для замера времени выполнения
void measure_time(const int count_of_tests) {
    // Отключаем отладочный вывод для замера времени
    Set::setDebug(false);

    char A_str[11], B_str[11], C_str[11], D_str[11];

    for (int set_size = 1; set_size < 10; ++set_size) {
        rand_set(A_str, set_size);
        rand_set(B_str, set_size);
        rand_set(C_str, set_size);
        rand_set(D_str, set_size);

        cout << "Set size = " << set_size << endl;
        cout << "Generated sets:\n";
```

```
cout << "A: " << A_str << " B: " << B_str << " C: " << C_str << " D: " << D_str << endl;

char E_array[11];
Node *E_list = nullptr;
bool E_bitarray[10];
unsigned int E_bitset = 0;
char E_str_from_list[11], E_str_from_bitarray[11], E_str_from_bitset[11];

// Замер времени для массива символов
auto start = chrono::high_resolution_clock::now();
for (int i = 0; i != count_of_tests; ++i) {
    compute_with_array(A_str, B_str, C_str, D_str, E_array);
}
auto end = chrono::high_resolution_clock::now();
chrono::duration<double> duration = end - start;
cout << "Array time: " << duration.count() << " seconds\n";

// Замер времени для списка
start = chrono::high_resolution_clock::now();
for (int i = 0; i != count_of_tests; ++i) {
    compute_with_list(A_str, B_str, C_str, D_str, E_list);
}
end = chrono::high_resolution_clock::now();
duration = end - start;
cout << "List time: " << duration.count() << " seconds\n";
list_to_string(E_list, E_str_from_list);
clear_list(E_list);

// Замер времени для битового массива
start = chrono::high_resolution_clock::now();
for (int i = 0; i != count_of_tests; ++i) {
    compute_with_bitarr(A_str, B_str, C_str, D_str, E_bitarray);
}
end = chrono::high_resolution_clock::now();
duration = end - start;
cout << "BitArray time: " << duration.count() << " seconds\n";
bitarr_to_str(E_bitarray, E_str_from_bitarray);

// Замер времени для битовой маски
start = chrono::high_resolution_clock::now();
for (int i = 0; i != count_of_tests; ++i) {
```

```
            compute_with_bitset(A_str, B_str, C_str, D_str, E_bitset);
        }
        end = chrono::high_resolution_clock::now();
        duration = end - start;
        cout << "Bitset time: " << duration.count() << " seconds\n";
        m_word_to_str(E_bitset, E_str_from_bitset);

        // Вывод результатов
        cout << "Results:\n";
        cout << "Array: " << E_array << endl;
        cout << "List: " << E_str_from_list << endl;
        cout << "BitArray: " << E_str_from_bitarray << endl;
        cout << "Bitset: " << E_str_from_bitset << endl;
        cout << endl;
    }

    // Включаем отладочный вывод обратно
    Set::setDebug(true);
}

int main() {
    my_srand(time(0));

    char E_array[11];
    Node *E_list = nullptr;
    bool E_bitarray[10];
    unsigned int E_bitset = 0;
    char E_str_from_list[11], E_str_from_bitarray[11], E_str_from_bitset[11];
    int choice = 0;

    do
    {
        do {
        cout << "Enter your choice how to run the program:\n1 - run program with constant data\n2 - run program with
console data\n3 - run program with random data (10 tests)\n4 - exit" << endl;
        cin >> choice;
    } while (choice != 1 && choice != 2 && choice != 3);

    if (choice == 1) {
        screen_cleaner();
        // Константные данные
```

```
    const char *A = "013";
    const char *B = "123";
    const char *C = "234";
    const char *D = "345";

    cout << "Constant test with tracing:\n";
    cout << "A: " << A << "\nB: " << B << "\nC: " << C << "\nD: " << D << endl;

    compute_with_array(A, B, C, D, E_array);
    compute_with_list(A, B, C, D, E_list);
    compute_with_bitarr(A, B, C, D, E_bitarray);
    compute_with_bitset(A, B, C, D, E_bitset);

    list_to_string(E_list, E_str_from_list);
    bitarr_to_str(E_bitarray, E_str_from_bitarray);
    m_word_to_str(E_bitset, E_str_from_bitset);

    cout << "Results:\n";
    cout << "Array: " << E_array << endl;
    cout << "List: " << E_str_from_list << endl;
    cout << "BitArray: " << E_str_from_bitarray << endl;
    cout << "Bitset: " << E_str_from_bitset << endl;

    clear_list(E_list);
}
else if (choice == 2) {
    screen_cleaner();
    // Тест с вводом с клавиатуры
    char A_input[80], B_input[80], C_input[80], D_input[80];
    cout << "\nEnter sets of digits (without spaces):\n";
    cin.ignore();
    cout << "A: ";
    cin.getline(A_input, 80);
    cout << "B: ";
    cin.getline(B_input, 80);
    cout << "C: ";
    cin.getline(C_input, 80);
    cout << "D: ";
    cin.getline(D_input, 80);

    compute_with_array(A_input, B_input, C_input, D_input, E_array);
```

```
        compute_with_list(A_input, B_input, C_input, D_input, E_list);
        compute_with_bitarr(A_input, B_input, C_input, D_input, E_bitarray);
        compute_with_bitset(A_input, B_input, C_input, D_input, E_bitset);

        list_to_string(E_list, E_str_from_list);
        bitarr_to_str(E_bitarray, E_str_from_bitarray);
        m_word_to_str(E_bitset, E_str_from_bitset);

        cout << "Input test:\n";
        cout << "Array: " << E_array << endl;
        cout << "List: " << E_str_from_list << endl;
        cout << "BitArray: " << E_str_from_bitarray << endl;
        cout << "Bitset: " << E_str_from_bitset << endl;

        clear_list(E_list);
    }
    else if (choice == 3) {
        screen_cleaner();
        // Замер времени на случайных данных
        int count_of_tests;
        do
        {
            cout << "Enter the number of tests for each set: ";
            cin >> count_of_tests;

        } while (count_of_tests <= 0);

        cout << "\nTime measurement with random sets:\n";
        measure_time(count_of_tests);
    }

    } while (choice != 4);

    return 0;
}
```