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

ОТЧЕТ
по зачетной работе № 1
по дисциплине «Алгоритмы и структуры данных»
НА ТЕМУ: «Множество в памяти ЭВМ»

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

Санкт-Петербург
2025

Цель работы: сравнительное исследование четырёх способов хранения множеств в памяти ЭВМ.

Содержание работы: поэтапная разработка программы для обработки множеств 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, опытные программисты могут 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).

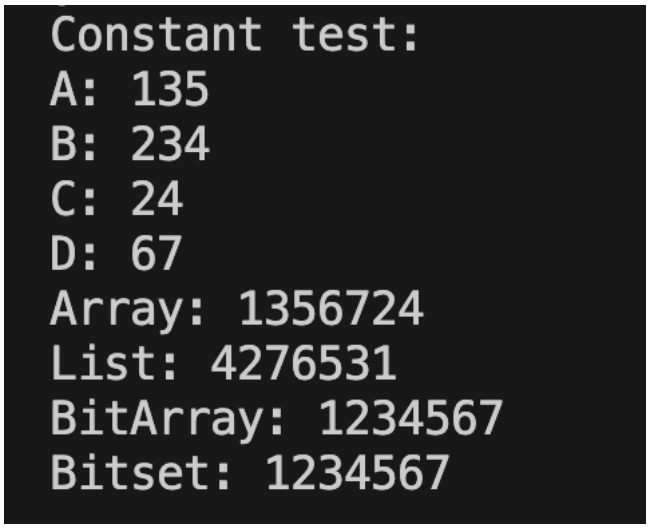
Битсет: все операции реализуются побитовыми операциями над 32-битным целым числом, поэтому сложность строго O(1) для операций над множествами.

Контрольные тесты

Test № 1

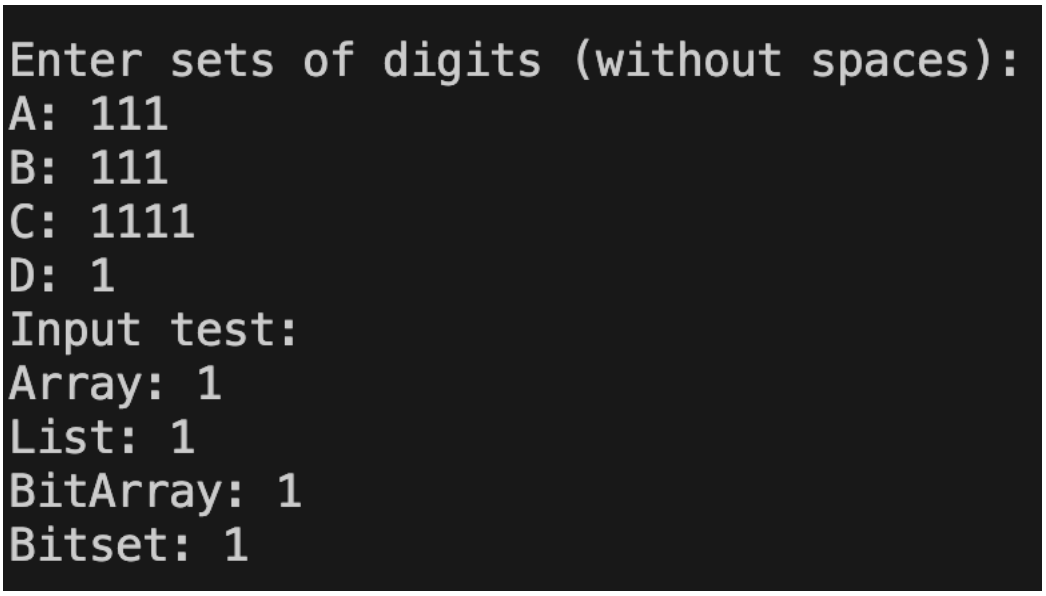
Входные данные (константные)				Ожидаемый результат
A	B	C	D	E
135	234	24	67	1234567

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

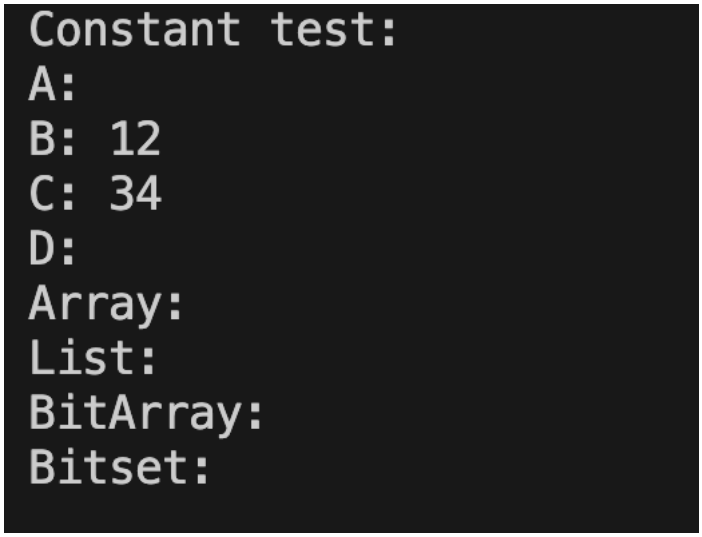
Результат:

```
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

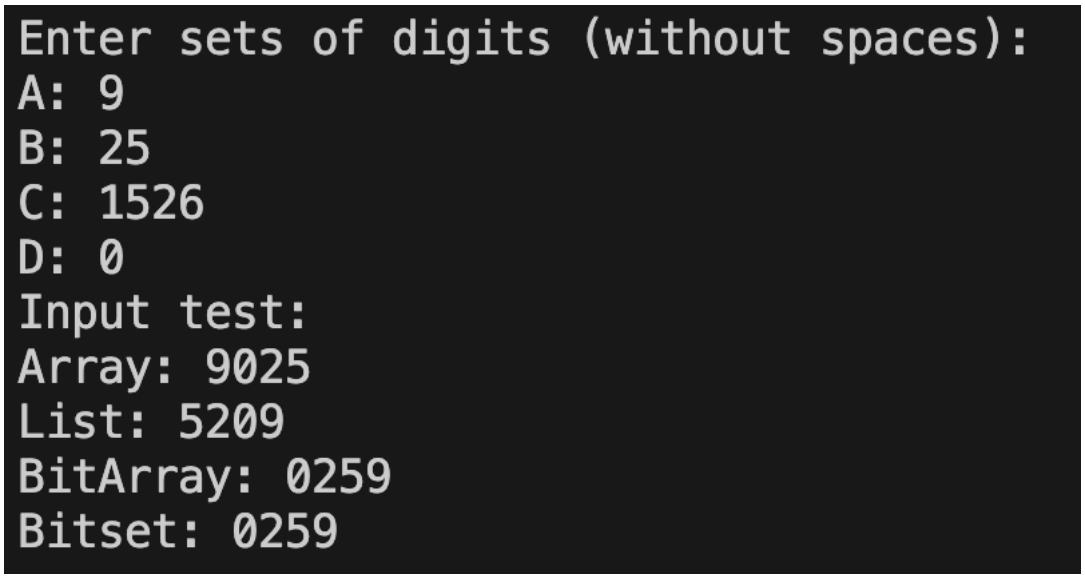
Входные данные (константные)				Ожидаемый результат
A	B	C	D	E
∅	12	34	∅	∅

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	1.67e-07	4.58e-07	6.25e-07	1.66e-07
2	2.91e-07	6.25e-07	5.42e-07	1.66e-07
3	4.59e-07	8.33e-07	5.42e-07	1.67e-07
4	7.09e-07	1.333e-06	4.58e-07	2.08e-07
5	8.75e-07	1.875e-06	5.42e-07	2.92e-07
6	9.17e-07	1.708e-06	6.25e-07	2.92e-07
7	1e-06	1.917e-06	5.83e-07	3.33e-07
8	1.292e-06	2.833e-06	5.83e-07	2.92e-07
9	1.25e-06	2.291e-06	5.83e-07	3.34e-07

Вывод

В результате выполнения работы мы пришли к следующим выводам:

Самый быстрый способ — это использование машинного слова (битовых масок). Он идеально подходит, когда мы заранее знаем все возможные элементы множества (универсум) и их количество не больше 64 (для 64-битных систем). В этом случае все операции над множествами выполняются буквально за одну команду процессора 0(1).

Битовые массивы - тоже очень быстрый способ, похожий на машинное слово, но позволяющий работать с универсумами больше 64. Оба этих метода хороши тем, что автоматически исключают повторы элементов.

Списки оказались самым медленным способом. Их стоит использовать только в ситуациях, когда мы не знаем, сколько элементов будет в множестве, и не можем заранее выделить нужный объем памяти.

Обычные массивы лучше применять, когда универсум слишком велик для битовых методов, но примерно представляем максимальный размер множества.

Итог: выбор способа хранения множества зависит от конкретной задачи. Нужно учитывать размер универсума, требуемую скорость работы и то, насколько мы можем предсказать размер множества.

Список использованных источников:

1. Колинько, П.Г. Пользовательские структуры данных [Текст]: методи-ческие указания по дисциплине «Алгоритмы и структуры данных, часть 1» /П.Г. Колинько, Н. Т. – Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020. – 64 с.

2. Множества в памяти ЭВМ // Алгоритмы и структуры данных. Лекция от 23.09.2025.

Код программы:

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

using namespace std;

const int UNIVERSAL_SIZE = 10;

// Структура для узла списка
struct Node
{
    char digit;
    Node *next;
    Node(char d, Node *n = nullptr) : digit(d), next(n) {}
};

// Функции для работы с массивами символов
bool is_in_array(char c, const char *arr)
{
    for (int i = 0; arr[i] != '\0'; ++i)
    {
        if (arr[i] == c)
            return true;
    }
    return false;
}

void string_copy(char *dest, const char *src)
{
    int i = 0;
```

```
    while (src[i] != '\0')
    {
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0';
}

void compute_with_array(const char *A, const char *B, const char *C, const char *D, char *E)
{
    char temp[11] = {0};
    int index = 0;

    for (int i = 0; B[i] != '\0'; ++i)
    {
        if (is_in_array(B[i], C) && !is_in_array(B[i], temp))
        {
            temp[index++] = B[i];
        }
    }

    for (int i = 0; A[i] != '\0'; ++i)
    {
        if (!is_in_array(A[i], temp))
        {
            temp[index++] = A[i];
        }
    }

    for (int i = 0; D[i] != '\0'; ++i)
    {
        if (!is_in_array(D[i], temp))
        {
            temp[index++] = D[i];
        }
    }

    temp[index] = '\0';
    string_copy(E, temp);
}
```

```
// Функции для работы с односвязными списками
bool contains(Node *head, char c)
{
    Node *p = head;
    while (p != nullptr)
    {
        if (p->digit == c)
            return true;
        p = p->next;
    }
    return false;
}

void insert(Node *&head, char c)
{
    if (contains(head, c))
        return;
    Node *newNode = new Node(c, head);
    head = newNode;
}

void clear_list(Node *&head)
{
    Node *p = head;
    while (p != nullptr)
    {
        Node *temp = p;
        p = p->next;
        delete temp;
    }
    head = nullptr;
}

void list_to_string(Node *head, char *str)
{
    int i = 0;
    Node *p = head;
    while (p != nullptr)
    {
        str[i++] = p->digit;
        p = p->next;
```

```
    }
    str[i] = '\0';
}

void compute_with_list(const char *A, const char *B, const char *C, const char *D, Node *&E)
{
    clear_list(E);
    for (int i = 0; A[i] != '\0'; ++i)
    {
        insert(E, A[i]);
    }
    for (int i = 0; D[i] != '\0'; ++i)
    {
        insert(E, D[i]);
    }
    for (int i = 0; B[i] != '\0'; ++i)
    {
        if (is_in_array(B[i], C) && !contains(E, B[i]))
        {
            insert(E, B[i]);
        }
    }
}

// Функции для работы с битовыми массивами
void str_to_bitarr(const char *str, bool *bits)
{
    for (int i = 0; i < UNIVERSAL_SIZE; ++i)
        bits[i] = false;
    for (int i = 0; str[i] != '\0'; ++i)
    {
        if (str[i] >= '0' && str[i] <= '9')
        {
            int index = str[i] - '0';
            bits[index] = true;
        }
    }
}

void bitarr_to_str(const bool *bits, char *str)
{
```

```
    int index = 0;
    for (int i = 0; i < UNIVERSAL_SIZE; ++i)
    {
        if (bits[i])
        {
            str[index++] = '0' + i;
        }
    }
    str[index] = '\0';
}

void compute_with_bitarr(const char *A, const char *B, const char *C, const char *D, bool *E)
{
    bool A_bits[UNIVERSAL_SIZE], B_bits[UNIVERSAL_SIZE], C_bits[UNIVERSAL_SIZE],
D_bits[UNIVERSAL_SIZE];
    str_to_bitarr(A, A_bits);
    str_to_bitarr(B, B_bits);
    str_to_bitarr(C, C_bits);
    str_to_bitarr(D, D_bits);

    for (int i = 0; i < UNIVERSAL_SIZE; ++i)
    {
        E[i] = A_bits[i] || D_bits[i] || (B_bits[i] && C_bits[i]);
    }
}

// Функции для работы с машинным словом
void str_to_m_word(const char *str, unsigned int &w)
{
    w = 0;
    for (int i = 0; str[i] != '\0'; ++i)
    {
        if (str[i] >= '0' && str[i] <= '9')
        {
            int index = str[i] - '0';
            w |= (1 << index);
        }
    }
}

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';
}

void compute_with_bitset(const char *A, const char *B, const char *C, const char *D, unsigned int &wE)
{
    unsigned int wA, wB, wC, wD;
    str_to_m_word(A, wA);
    str_to_m_word(B, wB);
    str_to_m_word(C, wC);
    str_to_m_word(D, wD);
    wE = wA | wD | (wB & wC);
}

// Простой генератор случайных чисел (линейный конгруэнтный метод)
static unsigned int next_rand = 1;

int my_rand()
{
    next_rand = next_rand * 1103515245 + 12345;
    return (unsigned int)(next_rand / 65536) % 32768;
}
// замена srand из cstdlib
void my_srand(unsigned int seed)
{
    next_rand = seed;
}

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

```
}

// 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()
{

    // int set_size = 3;
    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 << "\nB: " << B_str << "\nC: " << C_str << "\nD: " << D_str << endl;

        // Можно изменить кол-во повторений = домножить на 10^n степени скорость работы
        const int repetitions = 1000000;
```

```
char E_array[11];
Node *E_list = nullptr;
bool E_bitarray[UNIVERSAL_SIZE];
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 < repetitions; ++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 < repetitions; ++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 < repetitions; ++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 < repetitions; ++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;
    }
}

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

    char E_array[11];
    Node *E_list = nullptr;
    bool E_bitarray[UNIVERSAL_SIZE];
    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
    {
        cout << "Enter you'r choice how to run the programm:\n1 - run programm with constant data\n2 - run programm
with console data\n3 - run programm with random data (10 test)" << endl;
        cin >> choice;
    } while (choice != 1 && choice != 2 && choice != 3);

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

```
    const char *A = "0";
    const char *B = "1";
    const char *C = "3";
    const char *D = "4";

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

    // 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";
    getchar();
    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)
    {
        // Замер времени на случайных данных
        cout << "\nTime measurement with random sets:\n";
        measure_time();
    }

    return 0;
}
```