**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 ∪ (B ∩ C) ∪ D

# **Временная сложность**

Таблица 1. Способы представления и временная сложность обработки

|  |  |  |
| --- | --- | --- |
| Способ представления | Временная сложность | |
| Ожидаемая | Фактическая |
| Массив символов | O() | O(n2) |
| Список | O(n2) |
| 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(n²) 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(n²).

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

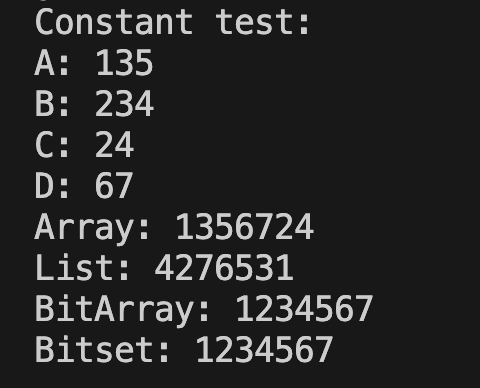
Битсет: все операции реализуются побитовыми операциями над 32-битным целым числом, поэтому сложность строго O(1) для операций над множествами.

**Контрольные тесты**

**Test № 1**

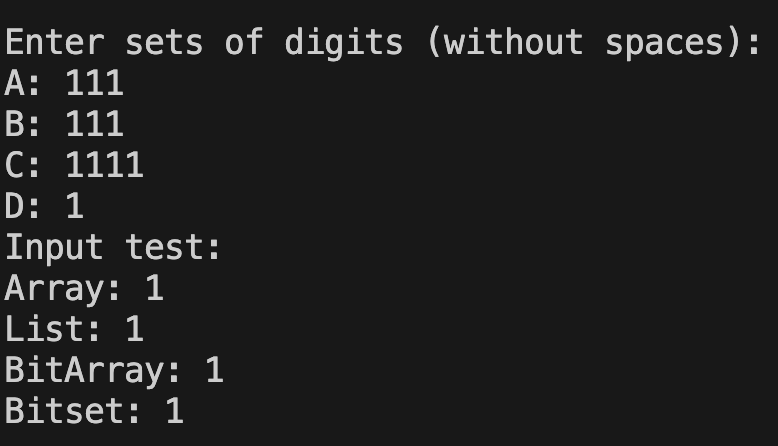
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (константные) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 135 | 234 | 24 | 67 | 1234567 |

Результат:



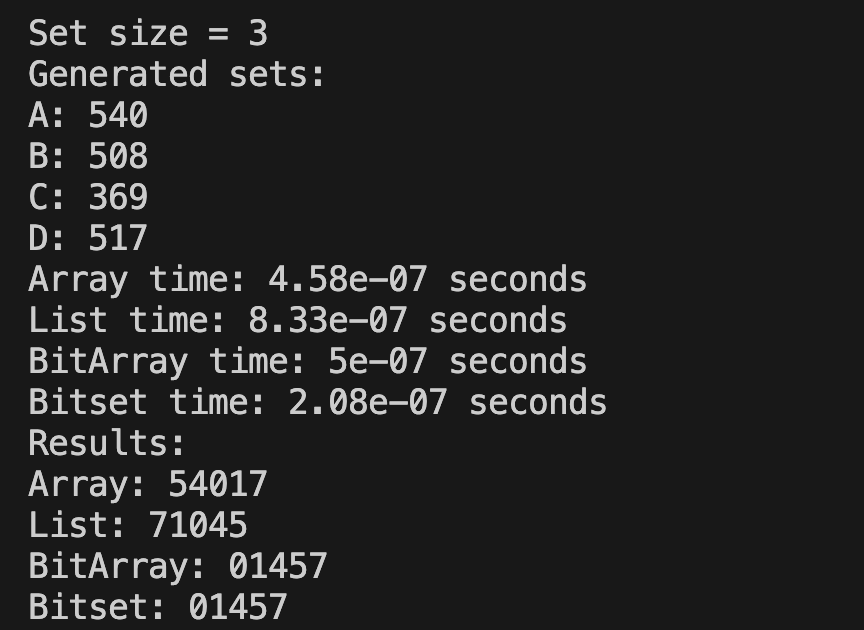
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (пользовательский ввод) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 111 | 111 | 1111 | 1 | 1 |

Результат:



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

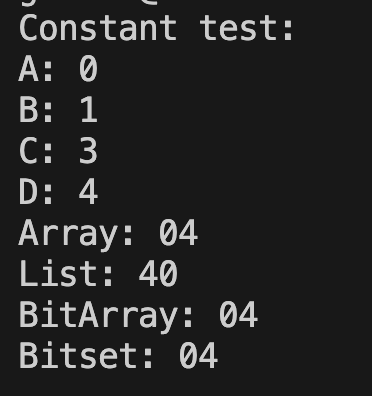
Результат:



**Test № 2**

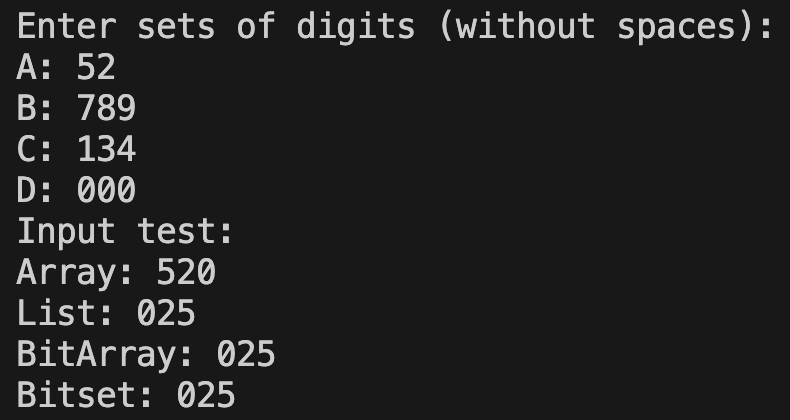
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (константные) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 0 | 1 | 3 | 4 | 04 |

Результат:



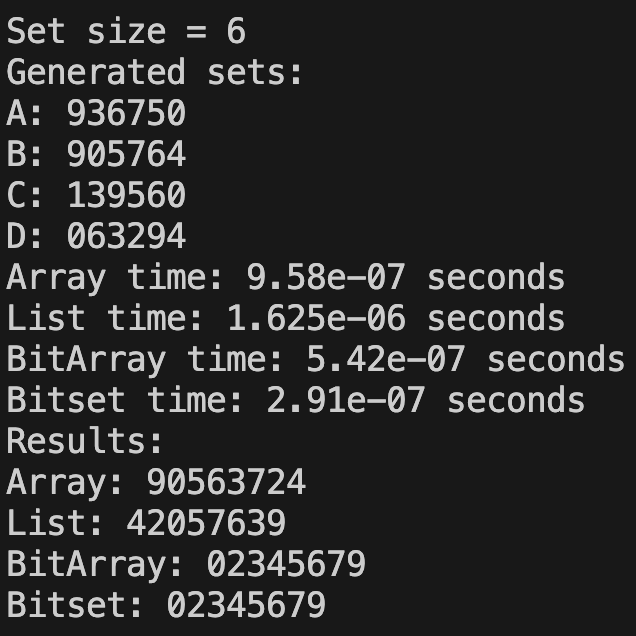
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (пользовательский ввод) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 52 | 789 | 134 | 000 | 025 |

Результат:



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

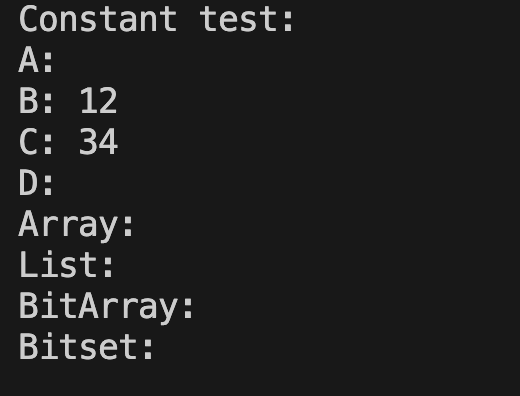
Результат:



**Test № 3**

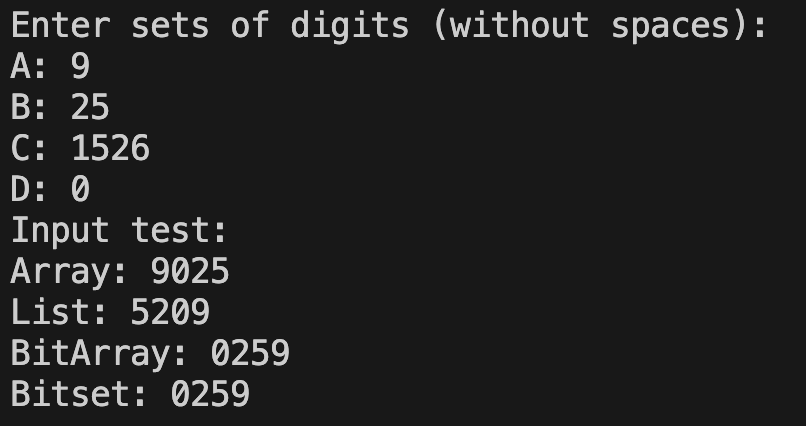
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (константные) | | | | Ожидаемый результат |
| A | B | C | D | E |
| ∅ | 12 | 34 | ∅ | ∅ |

Результат:



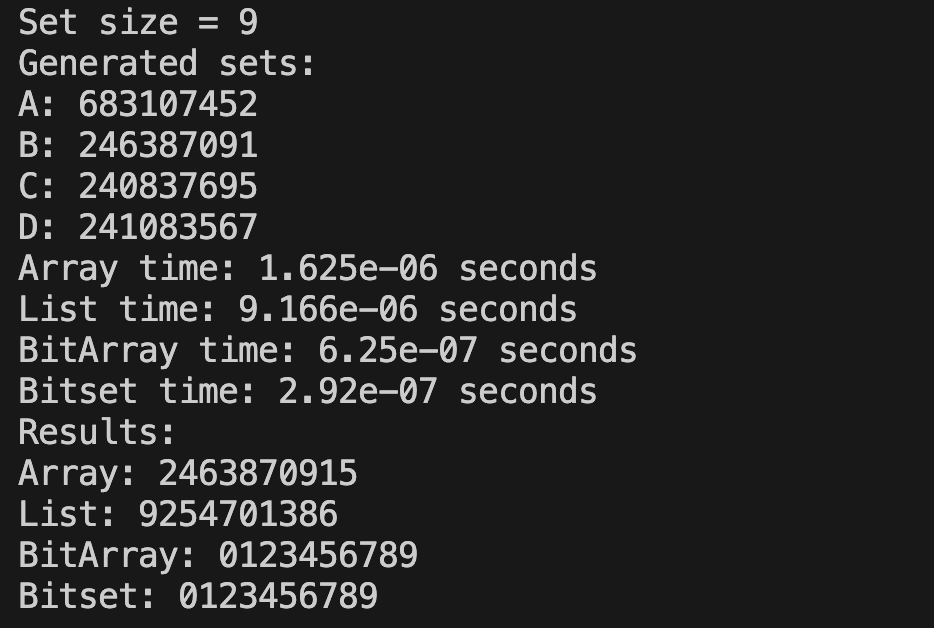
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (пользовательский ввод) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 9 | 25 | 1526 | 0 | 0259 |

Результат:



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

Результат:



# 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;

}