**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 ∪ (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).

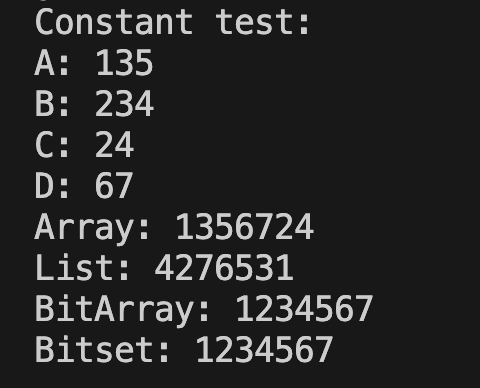
Битсет: все операции реализуются побитовыми операциями над 32-битным целым числом, поэтому сложность строго O(1) для операций над множествами.

**Контрольные тесты**

**Test № 1**

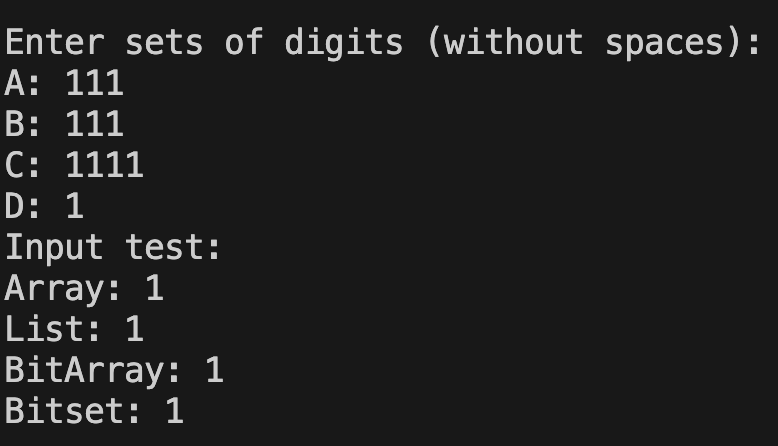
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (константные) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 135 | 234 | 24 | 67 | 1234567 |

Результат:



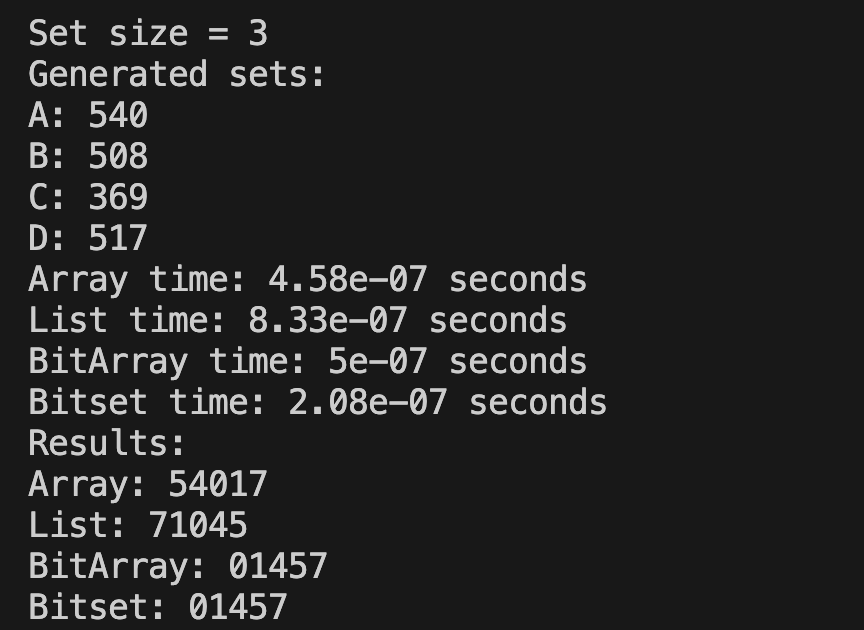
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (пользовательский ввод) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 111 | 111 | 1111 | 1 | 1 |

Результат:



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

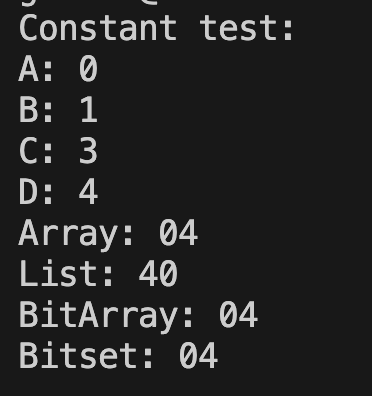
Результат:



**Test № 2**

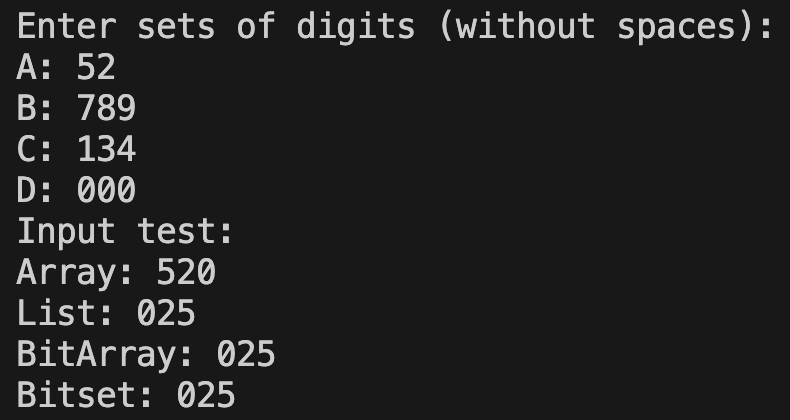
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (константные) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 0 | 1 | 3 | 4 | 04 |

Результат:



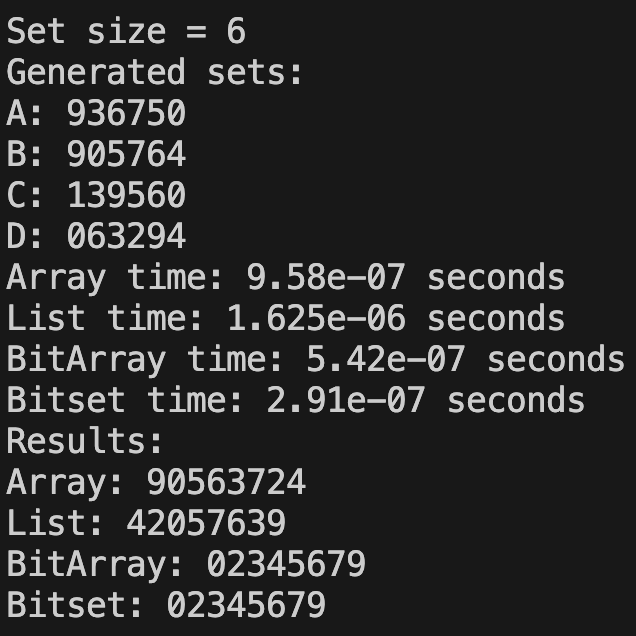
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (пользовательский ввод) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 52 | 789 | 134 | 000 | 025 |

Результат:



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

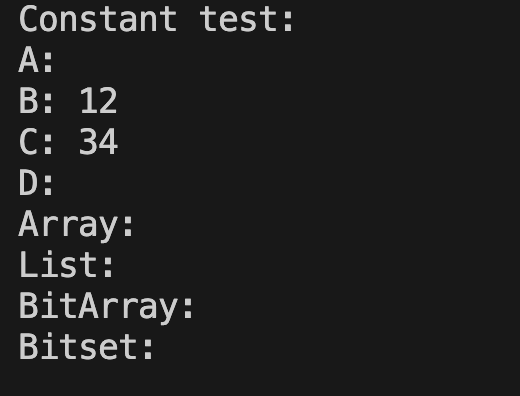
Результат:



**Test № 3**

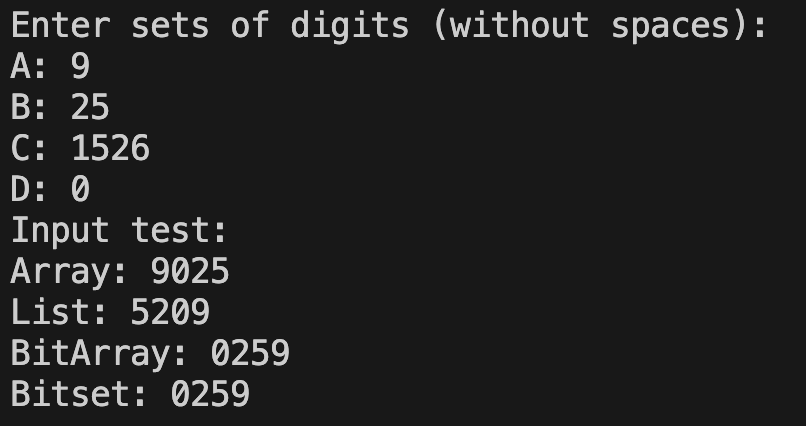
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (константные) | | | | Ожидаемый результат |
| A | B | C | D | E |
| ∅ | 12 | 34 | ∅ | ∅ |

Результат:



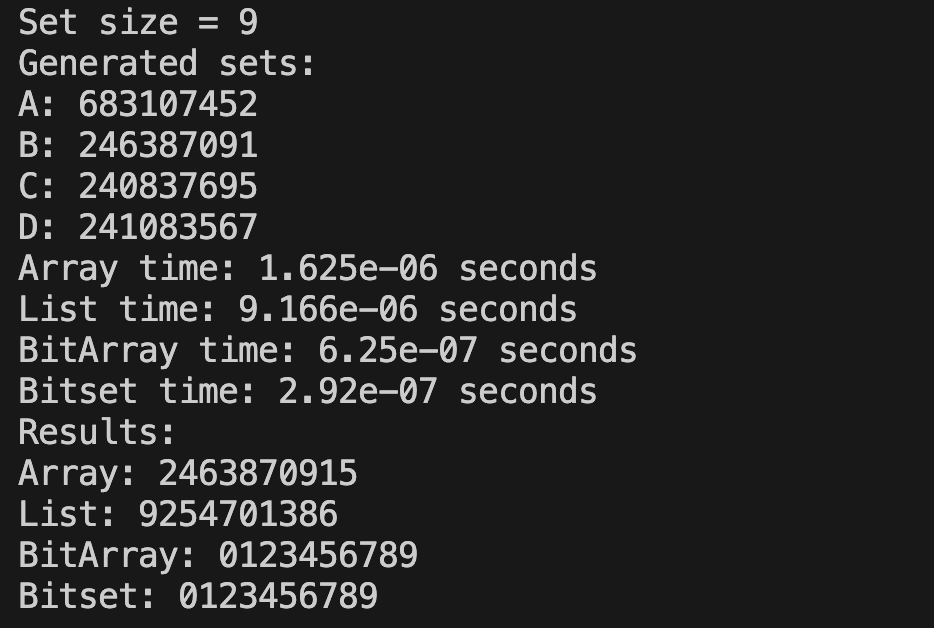
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные (пользовательский ввод) | | | | Ожидаемый результат |
| A | B | C | D | E |
| 9 | 25 | 1526 | 0 | 0259 |

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 | 1.67e-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;

}