**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 1**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Множество в памяти ЭВМ

|  |  |
| --- | --- |
| Студенты гр. 4312 | Устинов В. А. Гущин К. Д. |
| Преподаватель | Аббас С. А. М. |

Санкт-Петербург

2025

Оглавление

[1 Цель работы 3](#_Toc209194192)

[2 Задание 35 3](#_Toc209194193)

[3 Формула для вычисления пятого множества 3](#_Toc209194194)

[4 Контрольные тесты 4](#_Toc209194195)

[4.1 Ввод данных 4](#_Toc209194196)

[4.2 Генерация 4](#_Toc209194197)

[4.3 Вывод данных 5](#_Toc209194198)

[4.4 Результирующая панель обработки 6](#_Toc209194199)

[4.5 Контрольные примеры 7](#_Toc209194200)

[5 Временная сложность 11](#_Toc209194201)

[5.1 Таблица 11](#_Toc209194202)

[5.2 Пояснения 11](#_Toc209194203)

[6 Результат измерения времени 11](#_Toc209194204)

[7 Выводы 12](#_Toc209194205)

[8 Приложение 13](#_Toc209194206)

# Цель работы

Сравнительное исследование четырёх способов хранения множеств в памяти ЭВМ.

# Задание 35

Множество, содержащее все буквы множества A, которых нет во множествах B, C или D

# Формула для вычисления пятого множества

Формализация задания: E = A \ (B ∪ C ∪ D)

# Контрольные тесты

## **4.1 Ввод данных**

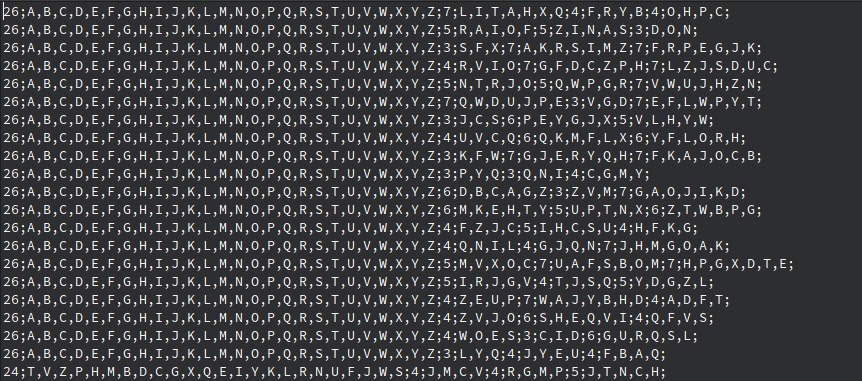
 Программа получает данные из файла *input.csv*. После запуска программы, в терминале появиться возможность сгенерировать новые данные, оставить уже введенные в файл либо в консоли. Файл *input.csv* представляет из себя построчно записанные массивы (A, B, C, D) в формате **длина массива;массив**

Рис. 1 input.csv

## **4.2 Генерация**

Если пользователь выбрал генерировать новые данные, то программа запросит ввести количество наборов (строк с множествами A, B, C, D), а цтакже минимальную и максимальную длину множества А.



Рис. 2 Ввод параметров генерации

## **4.3 Вывод данных**

После обработки входных данных, программа записывает результаты обработки множеств в 5 разных файлов:

* *idol\_output.txt* – ожидаемый результат обработки множеств. Подсчет логического выражения реализован с помощью контейнера **set** из библиотеки STL. С данным файлом сравниваются остальные 4 файла;
* *outputAr.txt* – результат обработки множеств через массив символов (const char\*);
* *outputBa.txt* – результат обработки множеств через массив битов;
* *outputBm.txt* – результат обработки множеств через машинное слово (битовую маску);
* *outputLi.txt* – результат обработки множеств через двусвязный список.

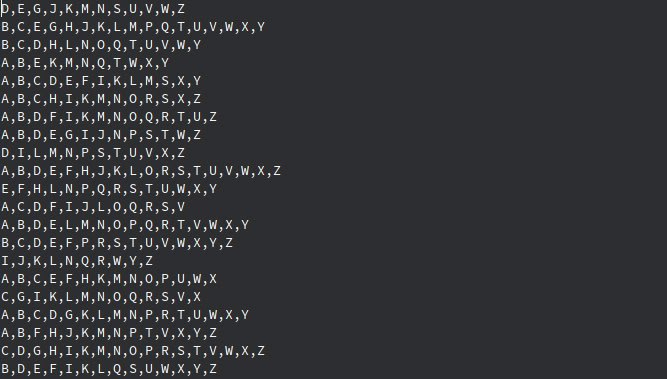
Вышеописанные 5 файлов представляют из себя построчно записанные результирующие множества E.

Рис. 3 idol\_output.txt

## **4.4 Результирующая панель обработки**

После обработки входных данных, в терминале выводятся результаты. А именно результат сравнения выходных данных реализации с ожидаемыми, время работы реализации и уведомление о записи данных в выходной файл реализации.

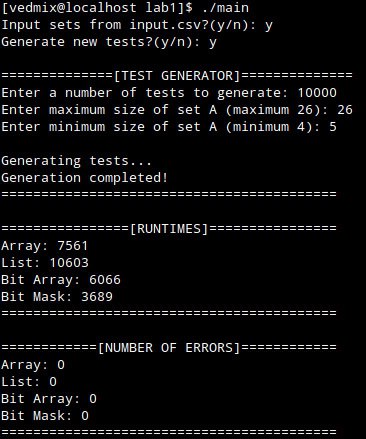


Рис. 4 Результаты обработки

## **4.5 Контрольные примеры**

Пример 1:

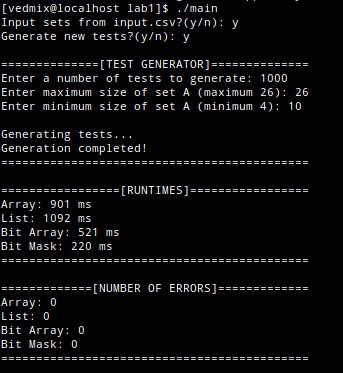


Рис. 5 Прогон 1000 тестов

Пример 2:

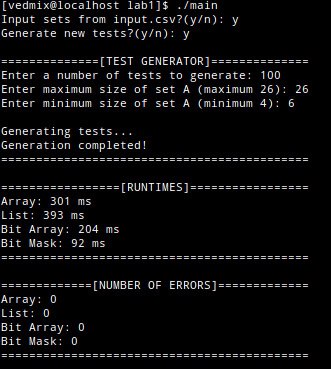


Рис. 6 Прогон 100 тестов

Пример 3:

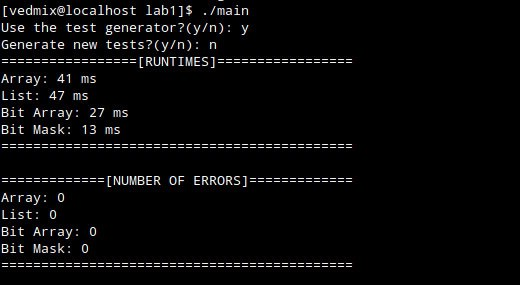


Рис. 7 Прогон 10 тестов

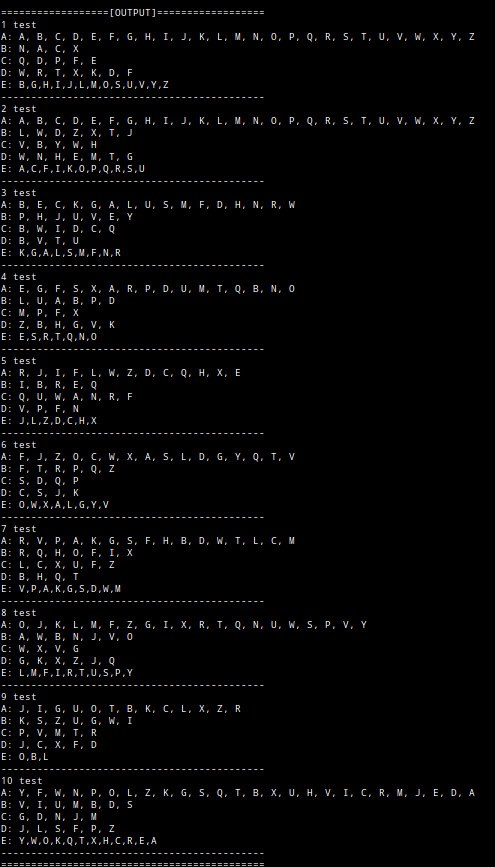


Рис. 8 После прогона менее 16 тестов, выводится результат

# Временная сложность

## **5.1 Таблица**

|  |  |  |
| --- | --- | --- |
| Способ представления | Временная сложность | |
| Ожидаемая | Фактическая |
| Массив символов | O(*n*2) | O(*n*2) |
| Двусвязный список | O(*n*2) |
| Массив битов | O(|*U*|) | O(1) |
| Машинное слово | O(1) |

## **5.2 Пояснения**

В ходе работы были реализованы и проанализированы четыре представления множеств, демонстрирующие различные классы временной сложности операции  **E = A \ (B ∪ C ∪ D)**. Массив символов и Двусвязный список показывают квадратичную сложность O(*n*2), поскольку для каждого элемента множества A требуется линейный поиск в трех других множествах.

Массив битов формально имеет сложность O(∣U∣), но благодаря фиксированному размеру универсума (26 символов) его фактическая сложность на практике является постоянной O(1).

Наиболее эффективной оказалась реализация на Машинном слове, где все операции над множествами сведены к побитовым операциям над целым числом, что обеспечивает константную сложность O(1) независимо от размера входных данных.

# Результат измерения времени

Для подтверждения теоретических оценок было проведено измерение времени выполнения программы на наборах данных различного объема. Результаты замеров (в микросекундах) представлены в пункте 4.5. Анализ результатов позволяет сделать следующие выводы:

Реализация на машинном слове (битовой маске) показала наилучшие результаты, что полностью подтверждает теоретическую оценку O(1). При росте объема входных данных в 10 раз (с 1 000 до 10 000 наборов) время выполнения увеличилось незначительно: с ~10-11 мкс (Рис. 5, 6, 7) до ~257-260 мкс (Рис. 8, 9, 10).

Реализация на массиве битов также продемонстрировала поведение, близкое к константной сложности O(1), благодаря фиксированному размеру универсума. Время ее выполнения ожидаемо немного выше, чем у машинного слова, но при увеличении нагрузки в 10 раз время возрастает сопоставимо с битовой маской (с ~11-12 мкс до ~270-275 мкс), что подтверждает эффективность подхода.

Реализации на массиве символов и двусвязном списке показали худшие результаты, что соответствует их квадратичной сложности O(n²). При увеличении количества тестов в 10 раз время их выполнения возросло не в 10, а приблизительно в 40-45 раз.

# Выводы

В ходе выполнения этой работы мы реализовали четыре разных способа хранения множеств и проверили, насколько быстро они работают.

Машинное слово оказалось самым быстрым и эффективным способом. Оно легко обрабатывает даже большие объемы данных, потому что все операции над множествами (проверка, объединение, вычитание) выполняются простыми действиями с битами.

Битовый массив тоже показал отличную скорость, близкую к константной, и надежно работал без ошибок. Он лишь немного уступил битовой маске из-за более сложной внутренней структуры.

Массив и список показали себя хорошо только на маленьких данных. Когда мы запустили тесты на 10000 наборов, они стали работать в разы медленнее. Это произошло из-за их квадратичной сложности O(n²), чтобы найти элемент, им приходилось перебирать все элементы внутри, и чем их больше, тем дольше шёл процесс.

# Приложение

1)

Рис. 9 Схема работы программы

2) Реализация множества через массив символов ([header](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\array_set.hpp), [source](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\array_set.cpp));

3) Реализация множества через массив битов ([header](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\barray_set.hpp), [source](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\barray_set.cpp));

4) Реализация множества через двусвязный список ([header](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\list.hpp));

5) Реализация множества через битовую маску ([header](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\bmask_set.hpp), [source](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\bmask_set.cpp));

6) Реализация генератора ([header](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\generator.hpp), [source](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\generator.cpp));

7) Реализация файлового менеджера ([header](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\fileManager.hpp), [source](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\fileManager.cpp));

8) Реализация ожидаемого множества ([header](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\idol_test.hpp), [source](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\idol_test.cpp));

9) Реализация main ([source](file:///C:\Users\vedmix\УЧЕБА\algosy\lab1\main.cpp)).