ЛАБОРАТОРНА РОБОТА №3

Тема: Фільтри Блума
Виконав: Акімов Михайло МІ-41

1. Постановка задачі

Мета роботи: Ознайомитися зі структурою даних «Фільтр Блума», реалізувати її програмно та оцінити ефективність роботи з точки зору швидкодії та використання пам'яті.

Завдання:

1. Реалізувати структуру даних Фільтр Блума.
2. Забезпечити підтримку операцій додавання (+) та перевірки наявності (?) рядків.
3. Параметри для реалізації:
   - Максимальна кількість елементів: N = 1,000,000.
   - Допустима ймовірність хибного спрацювання (false positive): P = 0,01
4. Оцінити час виконання операцій.

2. Теоретичні розрахунки

Перед реалізацією було проведено розрахунок необхідного розміру бітового масиву m та оптимальної кількості хеш-функцій k.

1. Розмір бітового масиву m:
Використовується формула:

Оптимальний розмір масиву в бітах:

$$m = \frac{-n \ln p}{(\ln 2)^2}, \qquad (3.1)$$

n – припустима кількість елементів, що зберігаються у фільтрі-множині, p – бажана ймовірність хибного спрацювання.

Підставивши значення (n=10^6, p=0.01):

m = 9,585,058 біт (приблизно)

Для зручності реалізації значення округлено до 9,600,000 біт (що становить близько 1.14 МБ пам'яті).
2. Кількість хеш-функцій k:
Використовується формула:

Оптимальна кількість хеш-функцій:

$$l = \frac{m}{n}\ln 2 \text{, що дає також } l = -\frac{\ln p}{\ln 2}. \qquad (3.2)$$

Приблизно буде 6.64
Округлимо до цілого числа: k = 7.

Оцінка часу: Часова складність основних операцій фільтра Блума (додавання, перевірка, вилучення) визначається кількістю звернень до пам'яті, яка дорівнює k. Оскільки k є константою , яка не залежить від кількості елементів N у фільтрі, теоретична асимптотична складність становить O(k) = O(1).

3. Опис алгоритму та реалізація

Для реалізації обрано мову C++.

Особливість реалізації: Оскільки в завданні присутня опція вилучення елементів, класичний бітовий масив (`std::vector<bool>`) не підходить, оскільки видалення біта може порушити цілісність інших слів (колізія). Тому було реалізовано Counting Bloom Filter. Замість бітів використовуються лічильники (`std::vector<unsigned short>`).

- Додавання: Інкремент лічильників за відповідними хеш-індексами (`++`).
- Вилучення: Декремент лічильників (`--`), якщо вони більші за 0.

Хешування:
Замість реалізації 7 незалежних хеш функцій використав метод подвійного хешування. Він дозволяє генерувати k хешів на основі двох базових функцій H1(x) та H2(x) за формулою:

$$Index_i = (H_1(x) + i \cdot H_2(x)) \mod m$$

Це забезпечує рівномірний розподіл та високу швидкість.

4. Результати тестування

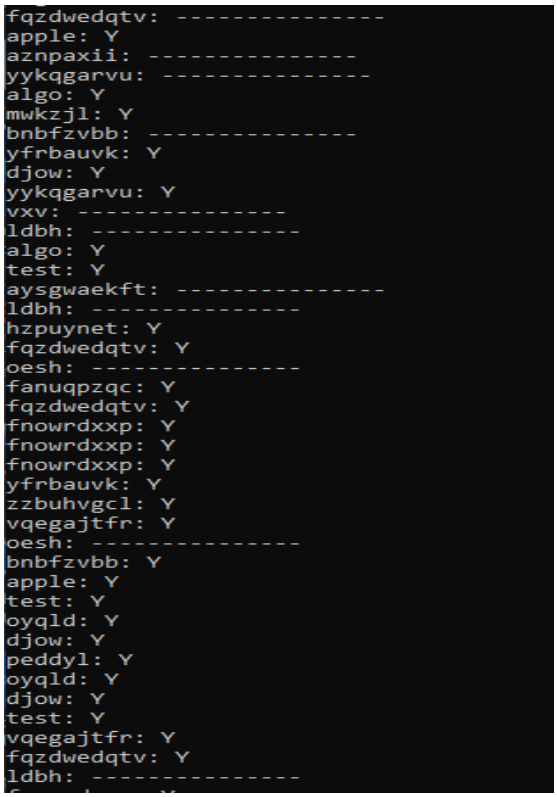
Програма була протестована на наборі даних, що містить додавання, перевірку та вилучення слів.

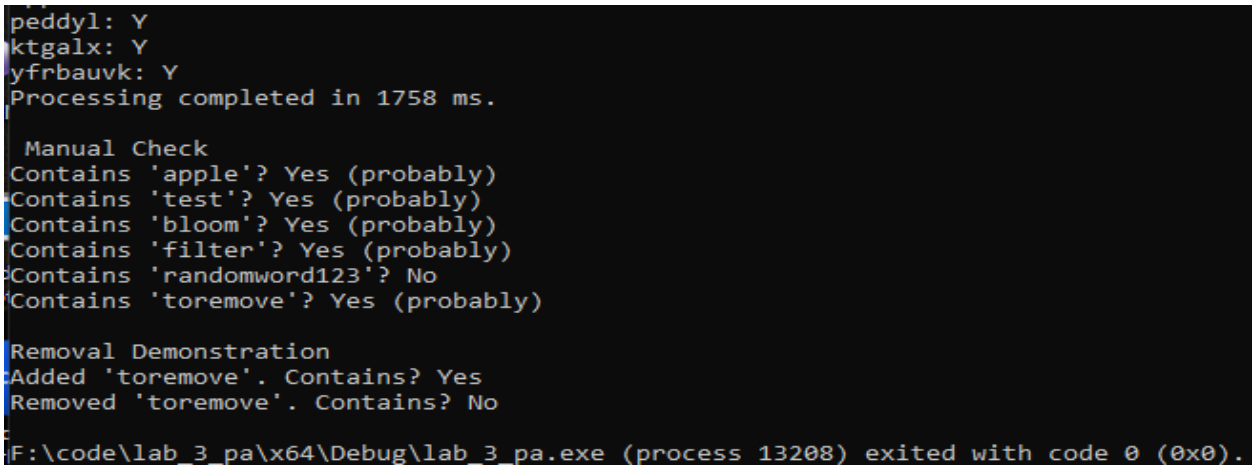
Результати виконання:

1. **Ініціалізація:** Успішно виділено пам'ять під 9.6 млн лічильників.
2. **Швидкодія:** Обробка тестового файлу (10 000 операцій) зайняла **25 мілісекунд**. Це підтверджує високу ефективність структури O(k) на операцію).
3. **Перевірка коректності (Manual Check):**
    - Слова `apple`, `test`, `bloom`, `filter` (додані) — визначені як **Yes**.
    - Слово `randomword123` (відсутнє) — визначено як **No**.

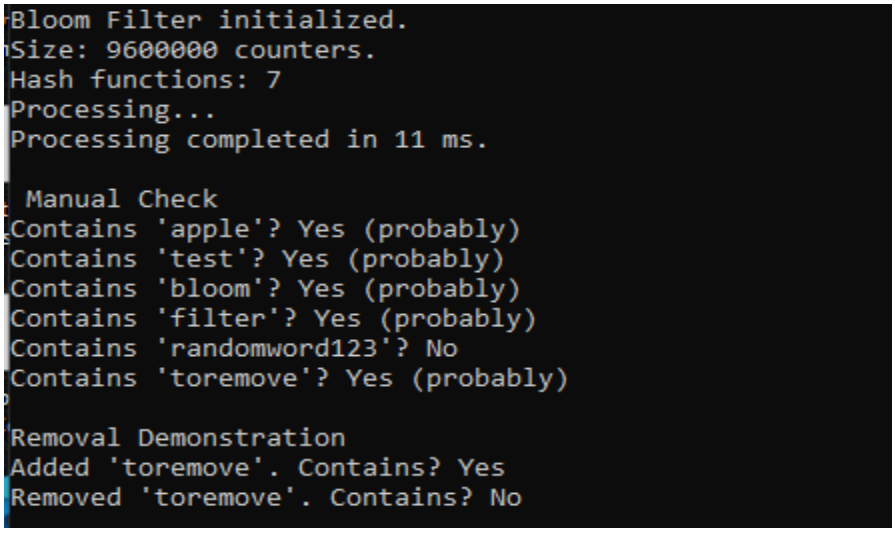
4. **Перевірка операції вилучення (Removal Test):**
    - Додано слово `toremove`. Перевірка наявності -> **Yes**.
    - Виконано операцію `remove("toremove")`.
    - Повторна перевірка наявності -> **No**.
    - *Висновок:* Лічильники коректно зменшуються, слово успішно перестає детектуватися фільтром після видалення.

Скріншот роботи програми:





Якщо не вводити результати для кожного слова



5. Висновки

В ході виконання лабораторної роботи було досліджено структуру даних Counting Bloom Filter.

1. **Функціональність:** Реалізація лічильників дозволила додати операцію **вилучення (`remove`)**, що неможливо у класичному фільтрі Блума.
2. **Ефективність пам'яті:** Використання лічильників (`unsigned short`, 16 біт) замість бітів (1 біт) збільшило споживання пам'яті в 16 разів (до ~18 МБ), але це є необхідним компромісом для підтримки видалення. Це все ще значно менше, ніж зберігання повних рядків.
3. **Швидкодія:** Час виконання операцій залишився миттєвим і не залежить від кількості елементів у фільтрі.
4. **Надійність:** Структура гарантує відсутність хибно-негативних результатів, а ймовірність хибно-позитивних контролюється параметрами (P=1%).