**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ**

**ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

**Факультет программной инженерии и компьютерной техники**

ОТЧЕТ

НА ТЕМУ «Bloom filters»

ПО ДИСЦИПЛИНЕ «Структуры и алгоритмы индексации данных»

Студент: Минина Дарья Николаевна

Группа: P4135

Преподаватель: к.т.н.

    Платонов Алексей Владимирович

Санкт-Петербург

2024

СОДЕРЖАНИЕ

[1. Описание алгоритма LSH **Ошибка! Закладка не определена.**](#_Toc178206603)

[2. Результаты 6](#_Toc178206604)

[ЗАКЛЮЧЕНИЕ 8](#_Toc178206605)

# **Bloom filters**

Bloom-filter — это вероятностная структура данных, которая используется для быстрой проверки принадлежности элемента к множеству. Рассмотрим ключевые моменты о Bloom-filter и его разновидностях.

Основные характеристики Bloom-filter

* Использует массив битов для хранения информации.
* Применяет несколько независимых хеш-функций для маппирования элементов в позиции массива.
* Предоставляет вероятностную проверку наличия элемента, допускающую ложноположительные результаты.

Разновидности Bloom-filter:

1. Counting Bloom filters:
   * Позволяют реализовать операцию удаления без создания заново структуру.
   * Каждая позиция массива теперь представляет собой многобитный счетчик.
   * Используют больше памяти, чем обычные Bloom filters (3-4 раза).
2. Space-efficient variants:
   * Путце и др. (2007): использует одну хеш-функцию и компрессию для экономии места.
   * Ротенштрейх и др. (2012): применяют переменные инкременты для улучшения точности.
3. Scalable Bloom filters:
   * Алмейда и др. (2007): адаптируются динамически к количеству элементов.
4. Compact approximators:
   * Больди и Вигна (2005): базируются на решетках вместо битов.
5. Spatial Bloom filters:
   * Пальмери и др. (2014): используются для хранения информации о местоположении.
6. Parallel-partitioned Bloom filters:
   * Используют отдельный массив для каждой хеш-функции, что позволяет параллельному вычислению.

Преимущества Bloom-filter:

1. Экономия памяти по сравнению с точными структурами данных.
2. Быстрое выполнение операций.

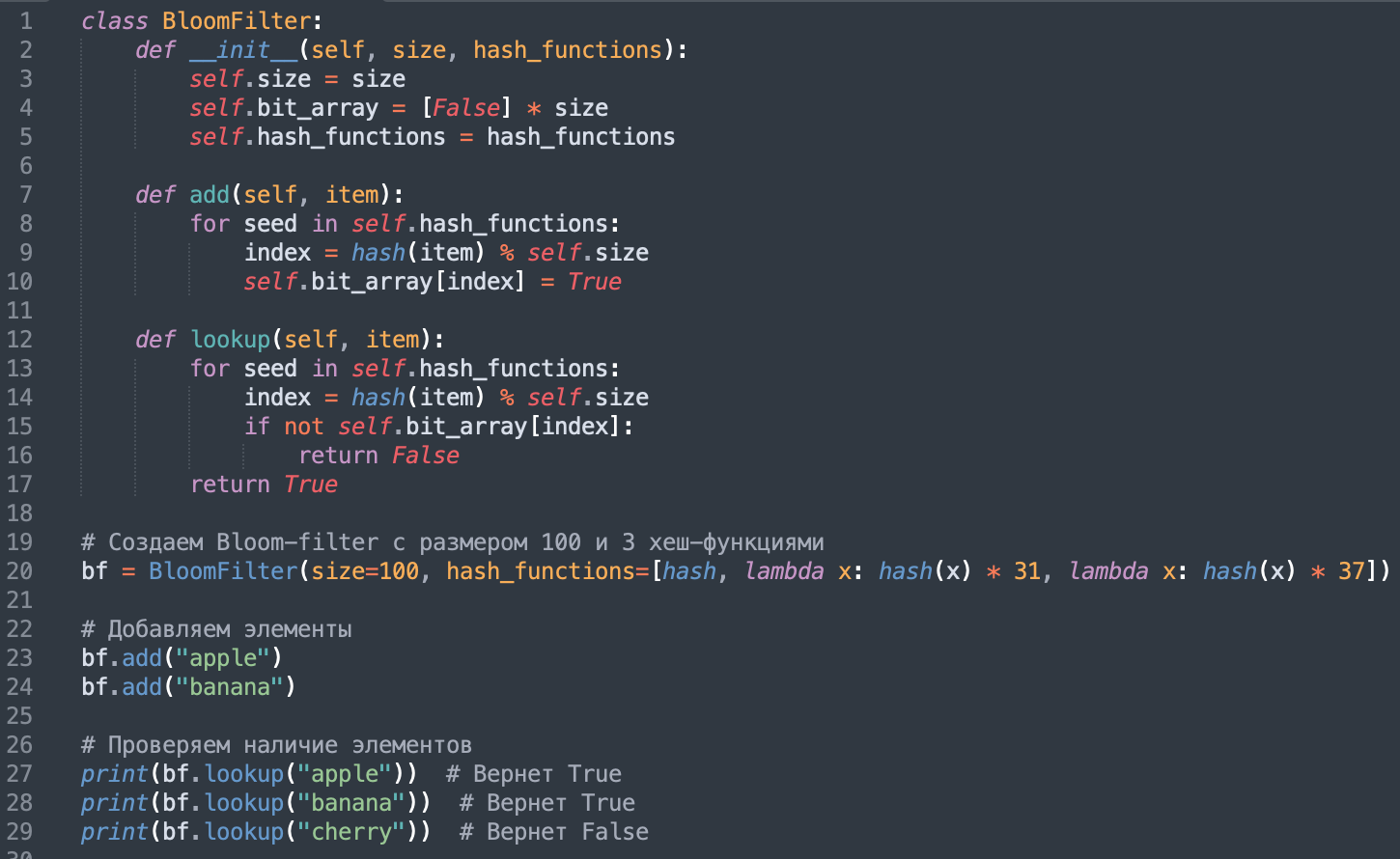
Недостатки:

1. Допускают ложноположительные результаты.
2. Не поддерживают операции удаления (в большинстве случаев).

Bloom-filter широко применяется в различных областях, таких как обработка больших данных, кэширование, сетевая аналитика и другие задачи, где важнее скорость и эффективность, чем абсолютная точность.

**Пример работы Bloom-filter**

Пример реализации и использования Bloom-filter на Python:



1. Мы создаем класс **BloomFilter** с параметрами размера массива бит и количества хеш-функций.
2. Метод **add(item)** добавляет элемент в Bloom-filter:
   * Для каждого элемента мы применяем заданные хеш-функции.
   * На основе результатов хеш-функций мы устанавливаем соответствующие биты в массиве в состояние True.
3. Метод **lookup(item)** проверяет наличие элемента в Bloom-filter:
   * Опять же, используем заданные хеш-функции для получения индексов.
   * Если хотя бы один из индексов содержит False, мы знаем, что элемент точно отсутствует.
   * Если все индексы содержат True, мы можем сказать, что элемент вероятно присутствует.
4. В нашем примере мы создаем Bloom-filter, добавляем несколько элементов и затем проверяем их наличие.

Особенности:

* Bloom-filter никогда не дает ложноотрицательных результатов, но может давать ложноположительные результаты (false positives).
* Более крупный Bloom-filter имеет меньший шанс на ложноположительные результаты, но требует больше памяти.
* Выбор числа хеш-функций влияет на баланс между скоростью и точностью.

Этот простой пример демонстрирует основную идею работы Bloom-filter и его использование. В реальных сценариях Bloom-filter могут быть значительно сложнее и оптимизированы для конкретных задач.

# **Результаты**

В лабораторной работе были реализованы Bloom filter и Scalable bloom filter и проведено сравнение их работы.

*``bloom``* — это модуль, который включает структуру данных *`bloom filter`* вместе с

реализацией *`Scalable bloom filter`* (Almeida, C. Baquero, N. Preguiça, D. Hutchison, Scalable Bloom Filters,

(GLOBECOM 2007), IEEE, 2007)

- **\*\****`Bloom filters`***\*\*** используются, если мы примерно понимаем, какое количество битов нужно выделить заранее для хранения всего набора.

- **\*\****`Scalable bloom filters`***\*\*** позволяют битам bloom filter расти в зависимости от вероятности ложного срабатывания и размера.

Фильтр "полный", когда он заполнен:

M \* ((ln 2 ^ 2) / abs(ln p)),

где M — количество битов, а p — вероятность ложного срабатывания.

Когда емкость достигнута, создается новый фильтр, экспоненциально больший, чем предыдущий, с более низкой вероятностью ложных срабатываний и большим количеством хэш-функций.

.. code-block:: python

>>> from bloom import BloomFilter

>>> f = BloomFilter(capacity=1000, error\_rate=0.001)

>>> [f.add(x) for x in range(10)]

[False, False, False, False, False, False, False, False, False, False]

>>> all([(x in f) for x in range(10)])

True

>>> 10 in f

False

>>> 5 in f

True

>>> f = BloomFilter(capacity=1000, error\_rate=0.001)

>>> for i in xrange(0, f.capacity):

... \_ = f.add(i)

>>> (1.0 - (len(f) / float(f.capacity))) <= f.error\_rate + 2e-18

True

>>> from bloom import ScalableBloomFilter

>>> sbf = ScalableBloomFilter(mode=ScalableBloomFilter.SMALL\_SET\_GROWTH)

>>> count = 10000

>>> for i in xrange(0, count):

... \_ = sbf.add(i)

...

>>> (1.0 - (len(sbf) / float(count))) <= sbf.error\_rate + 2e-18

True

Результаты бенчмарков:

Для *`Bloom filter`*:

*```*

*0.328 seconds to add to capacity, 305294.96 entries/second*

*Number of Filter Bits: 479256*

*Number of slices: 4*

*Bits per slice: 119814*

*------*

*Fraction of 1 bits at capacity: 0.566*

*0.252 seconds to check false positives, 397415.19 checks/second*

*Requested FP rate: 0.1000*

*Experimental false positive rate: 0.1048*

*Projected FP rate (Goel/Gupta): 0.102603*

*```*

Для *`Scalable bloom filter`*:

*```*

*0.798 seconds to add to capacity, 125347.20 entries/second*

*0.292 seconds to check false positives, 342620.67 checks/second*

*Requested FP rate: 0.1000*

*Experimental false positive rate: 0.0099*

*Final capacity: 100000*

*Count: 99814*

*```*

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения лабораторной работы были изучены bloom filter и проведено их сравнение.