УДК 004.021

ВЕРОЯТНОСТНЫЕ СТРУКТУРЫ ДАННЫХ – ФИЛЬТР БЛУМА

♦♦♦♦♦

PROBABILISTIC DATA STRUCTURES – BLOOM FILTER

Вергунова Алина Евгеньевна
студентка,
Кубанский государственный технологический университет
vergunova2004@gmail.com

Долунц Анастасия Станиславовна
студентка,
Кубанский государственный технологический университет
11naaastiaaa@gmail.com

Вонарх Юлия Сергеевна
старший преподаватель кафедры
информационных систем и программирования,
Кубанский государственный технологический университет
jul_nosova@maill.ru

Аннотация. Представлены методы для решения задач хранения и поиска информации на примере фильтра Блума. Приведено описание внутренней структуры фильтра Блума и принципы его работы. Такая структура данных используется в обработке больших объемов данных для эффективного определения принадлежности элемента к множеству, когда задача должна решаться в сжатом размере памяти и быстро. Представлена зависимость частоты ложных утверждений от размера фильтра. Предложены способы улучшения фильтра, и обсуждаются стратегии для кластеризации и распараллеливание системы. Описан структура фильтра для возможности удаления его элементов – фильтр Блума с подсчетом – его проблемы и возможности.

Ключевые слова: большие данные, вероятностные структуры данных, хеш-функция, массив, хранение, фильтр Блума, эффективность, минимизация хранения.

Vergunova Alina Evgenievna
Student,
Kuban State Technological University
vergunova2004@gmail.com

Dolunts Anastasia Stanislavovna
Student,
Kuban State Technological University
11naaastiaaa@gmail.com

Vonarkh Yulia Sergeevna
Senior Lecturer of the Department
of Information Systems and Programming,
Kuban State Technological University
jul_nosova@maill.ru

Annotation. Methods for solving problems of storing and retrieving information using the Bloom filter as an example are presented. A description of the internal structure of the Bloom filter and its operating principles are provided. This data structure is used in processing large amounts of data to effectively determine the membership of an element in a set, when the problem must be solved in a compressed memory size and quickly. The dependence of the frequency of false statements on the filter size is presented. Methods for improving the filter are proposed, and strategies for clustering and parallelizing the system are discussed. The structure of the filter for the possibility of removing its elements – Bloom filter with counting - is described – its problems and possibilities.

Keywords: big data, probabilistic data structures, hash function, array, storage, Bloom filter, efficiency, storage minimization.

Фильтр Блума – это вероятностная структура данных, придуманная Бёртоном Говардом Блумом в 1970 году [1], позволяющая хранить некое множество элементов, а также быстро отвечать на запрос о том, есть ли данный элемент во множестве или нет. При этом существует возможность получить не правильный результат. Если система работы с данными толерантна к незначительным накладным расходам на ошибки, то с помощью фильтра Блума можно значительно повысить ее производительность. Поэтому фильтр Блума нашел применение во многих сферах, а именно, в больших данных [2, 3], в дедупликации [4] , в различных сетевых технологиях, включая информационную безопасность [5], файловых системах [2], индексировании и т.п. Фильтр Блума неприменим там, где предъявляются строгие требования к ответам системы, например, базам данных, содержащим данные пользовательской аутентификации (логины, пароли и т.п.).

Построение и реализация фильтра Блума

Фильтр Блума состоит из двух ключевых компонентов (рис. 1): комплекта из k хэш-функций $h_1(x)$, $h_2(x)$, $h_3(x)$, ..., $h_k(x)$ и регистра Rg (массива битов) – то, что является самой структура для хранения длиной в m разрядов, заполненном нулями в исходном состоянии.

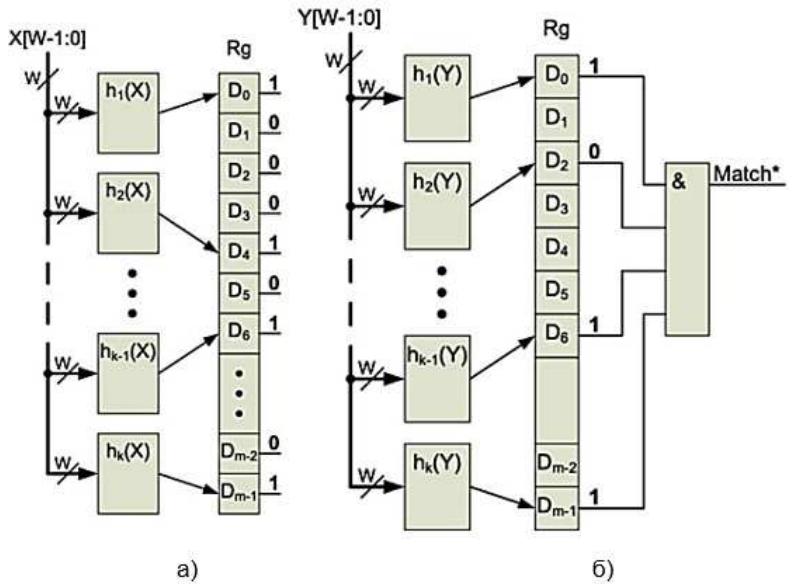

Рисунок 1 – Принцип работы фильтра Блума: а) запись; б) проверка вхождения

При записи входное значение подается на вход хэш-функции, которая возвращает индекс 0… m–1, и в данной позиции массива устанавливается «1». Так при $h_1(x) = 5$, результат первой функции установит «1» в 5-й индекс массива – в D5 (рис. 1а).

Проверка вхождения проходит аналогично (рис. 1 б): подаётся фрагмент входной последовательности символов. Для него вычисляются значения всех k хэш-функций. По полученным адресам осуществляется обращение к ячейкам битового массива. Если во всех позициях, на которые укажут хэш-функции, содержатся единицы, считается, что входная комбинация символов с определенной вероятностью совпадает с одним из паттернов, которое уже было записано. Но если хотя бы одна хэш-функция укажет на ячейку с нулевым значением, это гарантированно свидетельствует об отсутствии соответствующей последовательности символов среди множества участвовавших в программировании паттернов.

Если все проверяемые биты равны «1», то нельзя гарантировать, что такие данные уже были. Если был получен не правильный результат — это называется ложноположительное срабатывание [2].

Пусть задано множество элементов S, все элементы которого записаны в фильтр Блума. При проверке наличия элемента K в множестве S возможны четыре ситуации:

1. Срабатывание (ошибка первого рода): $K \notin S$, а фильтр Блума выдает, что $K \in B$.

2. Истинно положительное срабатывание: $K \in S$, и фильтр Блума выдает, что $K \in B$.

3. Ложноотрицательное срабатывание (ошибка второго рода): $K \in S$, а фильтр Блума выдает, что $K \notin B$.

4. Истинноотрицательное: $K \in S$, и фильтр Блума выдает, что $K \in B$.

Главной проблемой Фильтра Блума являются ложноположительные и ложноотрицательные ответы.

Большинство реализаций страдают от ложноположительных, но не ложноотрицательных срабатываний. Ложноотрицательные срабатывания представляют серьезную проблему в программных реализациях при операциях удаления, по этой причине большинство реализаций избегают возможности удаления элемента.

Интуитивно можно предположить, что с увеличением размера m фильтра Блума число наложений между отпечатками разных ключей должно уменьшаться, что, в свою очередь, приводит к меньшему числу ложных утверждений. Существует ли золотая середина, где и m, и частота ложноположительных срабатываний одновременно малы?

Ошибка первого рода

Значение вероятности ошибки первого рода для заданных значений количества паттернов n, размера битового массива m, и количества хэш- функций k находится согласно формуле (1):

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right) \approx (1-e^{-\frac{kn}{m}})^k = 1-e^{-\frac{k}{b}}, \quad (1)$$

где

$$b = \frac{m}{n}. \quad (2)$$

Формула (2) – часто называют пространством b, которое показывает количество бит на ключ.

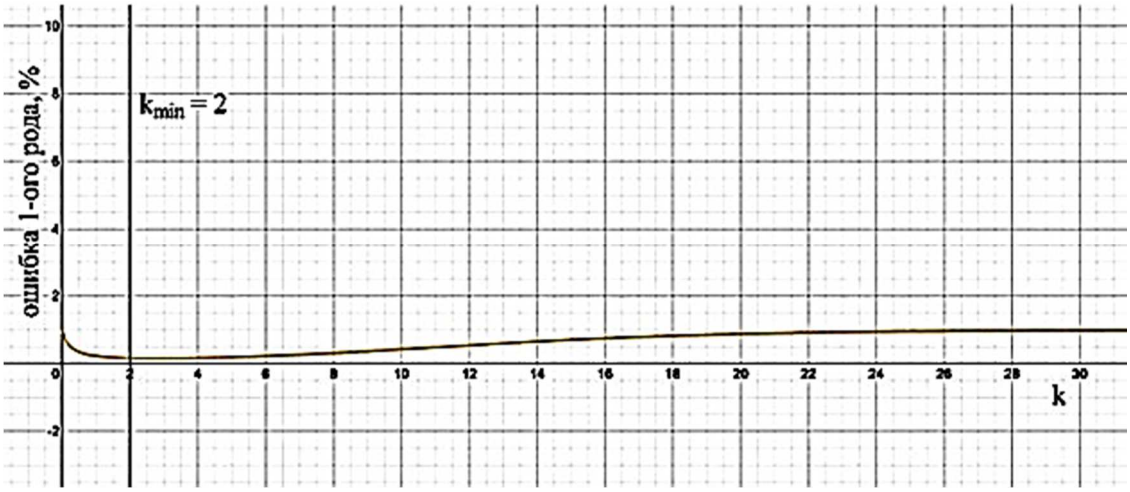

Рисунок 2 – График по формуле (1) при b = 4,
зависимость кол-ва хеш-функций и ошибки 1-города

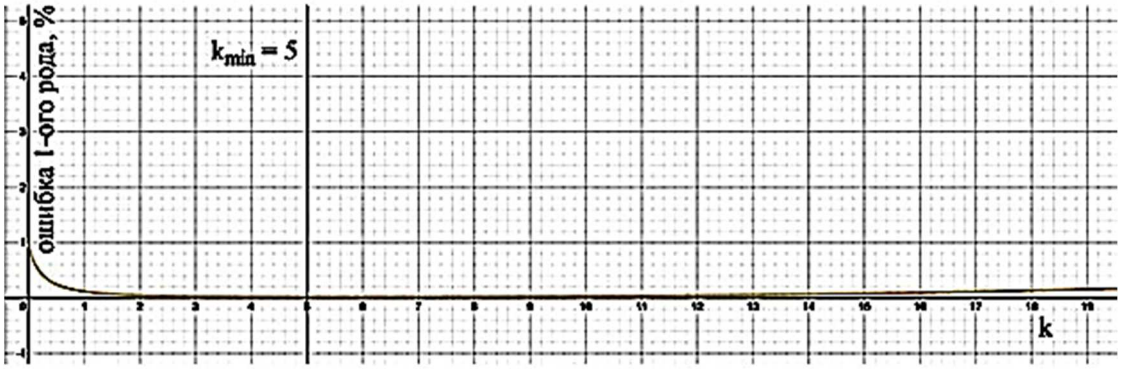

Рисунок 3 – График по формуле (1) при b = 8,
зависимость кол-ва хеш-функций и ошибки 1 – города

В дополнение к пространству в расчете на ключ b оценка (1) зависит от k, числа хеш-функций, которые используются фильтром Блума. Значение k находится под полным контролем проектировщика фильтра Блума, так почему бы не настроить его так, чтобы оно минимизировало оценочную частоту ошибок? То есть, оставляя b фиксированным, мы можем подобрать k для минимизации частоты ложных утверждений (рис. 2, 3). С помощью дифференциального исчисления можно подобрать наилучшее k:

$$k = \frac{m}{n} ln(2) = b \times (ln2). \quad (3)$$

Тогда вероятность ошибки при оптимальном k:

$$2^{-k} = 2^{-(ln2)*b}. \quad (4)$$

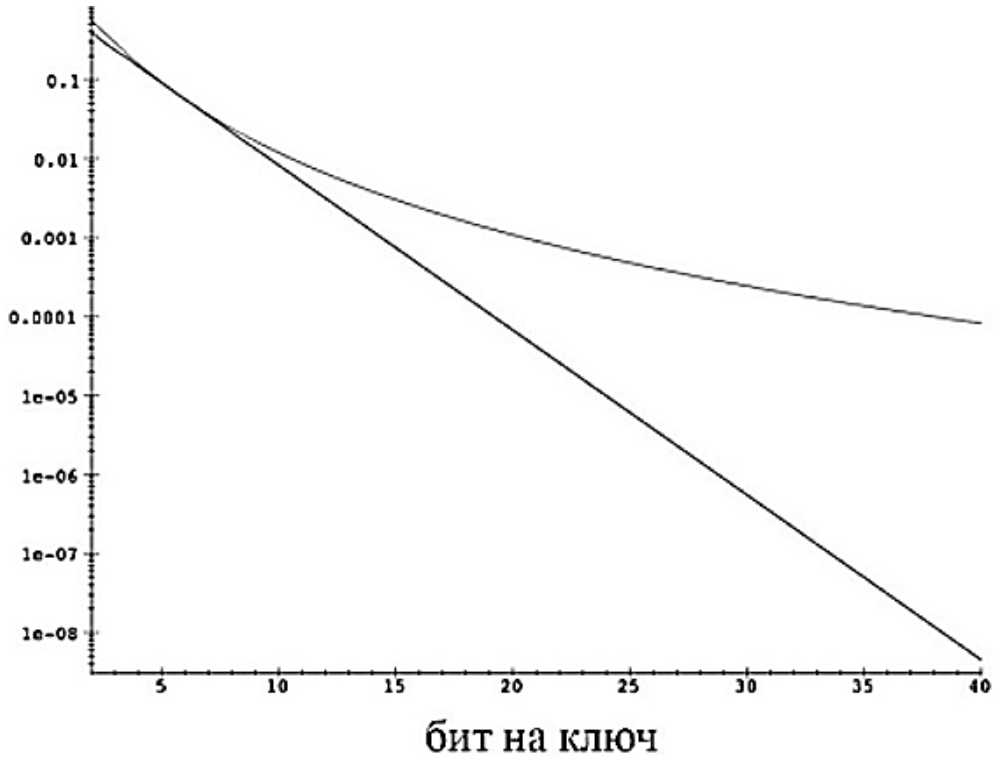

Рисунок 4 – Вероятность ложноположительных срабатываний от кол-ва бита на ключ: верхняя кривая – работа фильтра с 4-мя хеш-функциями; нижняя – оптимальное число функций

Это именно то, чего мы хотели с самого начала. У нас есть формула, которая выдает ожидаемую частоту ложных утверждений как функцию объема пространства, которое мы готовы использовать. Данная формула уменьшается экспоненциально вместе с пространством b в расчете на ключ, и именно поэтому существует золотая середина, где и размер фильтра Блума, и его частота ложных утверждений малы одновременно. Например, при хранении только 8 бит на ключ (b = 8) эта оценка составляет чуть более 2 % (табл. 1). В таблице 1 число k нужно привести к целому числу, для разных вариантов построения фильтра может быть выбран свой способ округления.

Таблица 1 – Данные, приведенные по формулам (3) и (4)

b, бит	k	ошибка 1-ого рода, %
1	2	3
4	2,77	14,63415427
8	5,54	2,141584712
16	11,09	0,045863851
24	16,63	0,000982213
32	22,18	2,10349E-05
40	27,72	4,50481E-07
48	33,27	9,64743E-09
56	38,81	2,06608E-10
64	44,36	4,42468E-12
72	49,90	9,47583E-14
80	55,45	2,02933E-15
88	60,99	4,34598E-17
96	66,54	9,30729E-19
104	72,08	1,99323E-20
112	77,63	4,26868E-22
120	83,17	9,14174E-24
128	88,72	1,95778E-25

Способы улучшения фильтра Блума

Вот несколько способов улучшить фильтр Блума:

1. Размер фильтра: можно увеличить размер фильтра, чтобы уменьшить вероятность ложноположительных результатов.

2. Выбор хеш-функций: нужно выбрать такие хеш-функции, которые будут равномерно распределять значения по битам фильтра.

3. Множественные фильтры: использовать несколько фильтров, в каждом из которых будут свои хеш-функции или же можно ограничиться одной функцией. Первый способ такой организации – можно каждую функцию привязать к одному фильтру, каждый новый паттерн обрабатываться каждой хеш-функцией, она устанавливает единицу в свой фильтр, тогда паттерн будет записан во всех фильтрах, но разными функциями. Если использовать такой подход, получается, что частота ложноположительных срабатываний будет зависеть только от работы конкретной хеш-функции, так как каждая имеет свой фильтр для записи, но возникают проблемы, нужно увеличивать его размер для избежания коллизий. Это будет хорошо только для малого числа данных. Второй способ, в отличие от первого, предлагает не записывать каждый паттерн во все фильтры, а разделить их по категориям в зависимости от их свойств – длине, значению, или же хешировать для определения категории. Такой способ называется кластеризация, о нем можно подробнее будет рассказано ниже.

Варианты с множественными фильтрами предполагает, что функции могут работать в одной массиве, в котором есть несколько фильтров, тогда результатом работы такой функции будет лежать в диапазоне своего указанного фильтра.

Динамическое изменение размера: рассмотрите возможность динамического изменения размера фильтра Блума в зависимости от количества элементов, которые вы хотите добавить. Это может помочь уменьшить расход памяти, когда ты заранее не знаешь сколько данных будет.

Подсчет ошибок: измеряйте вероятность ложноположительных и ложноотрицательных результатов. Это позволит вам оптимизировать параметры фильтра Блума.

Использование дополнительных проверок: если ложноположительные результаты могут иметь серьезные последствия, рассмотрите дополнительные методы проверки, чтобы исключить ложные срабатывания.

Кластеризация

Организация хеш-функции в контексте кластеризации фильтра Блума означает разделение данных на кластеры. Разделение, может быть, по свойствам входящих данных или же использовать нескольких хеш-функций, которые применяются для разделения данных на несколько кластеров.

Вот как это можно реализовать при использовании нескольких хеш-функций:

1. Хеш-функции первого уровня: начните с использования нескольких хеш-функций первого уровня для разделения вашего набора данных на несколько кластеров.

2. Фильтры Блума внутри каждого кластера: для каждого кластера создайте свой собственный фильтр Блума, в котором будут храниться элементы этого кластера. Выберите соответствующий размер и хеш-функции для каждого фильтра Блума.

3. Поиск в кластерах: когда вы хотите проверить, принадлежит ли элемент множеству, сначала примените хеш-функции первого уровня к элементу, чтобы определить кластер, в котором он, возможно, находится. Затем используйте соответствующий фильтр Блума для проверки наличия элемента внутри этого кластера.

Преимущество такого подхода заключается в том, что он может помочь уменьшить вероятность ложноположительных результатов, поскольку хеширование на два уровня позволяет более точно разбивать данные. Однако это также требует больше памяти и вычислительных ресурсов для поддержки множества фильтров Блума.

Распараллеливание

Распараллеливание фильтра Блума – это один из способов улучшения его производительности, особенно в многопоточных или распределенных системах. Способы такой реализации:

1. Распределенное хранение: если вы используете фильтр Блума в распределенной среде, вы можете хранить несколько копий фильтра Блума на разных серверах или узлах. Это позволит параллельно обрабатывать запросы на проверку принадлежности элемента к множеству.

2. Многопоточное добавление: если у вас есть многозадачное приложение, вы можете использовать несколько потоков для одновременного добавления элементов в фильтр Блума. Это ускорит процесс наполнения фильтра и обеспечит параллельную обработку операций.

3. Параллельный поиск: при выполнении операции проверки принадлежности элемента к множеству вы можете использовать несколько потоков или ядер процессора для одновременного применения хеш-функций и проверки результатов в фильтре Блума.

4. Распределенная обработка запросов: если у вас есть большое количество запросов на проверку принадлежности элемента к множеству, вы можете распределить эти запросы между несколькими узлами или серверами, каждый из которых использует свой собственный фильтр Блума. Это снизит нагрузку на каждый фильтр и позволит параллельно обрабатывать больше запросов.

Важно правильно согласовать доступ к фильтру Блума из разных потоков или узлов, чтобы избежать состояний гонки и других проблем с совместным доступом к данным. Также следует учитывать, что параллельное выполнение может повлечь за собой дополнительные затраты на согласование и управление потоками или узлами.

Фильтр Блума с подсчётом

Впервые подобная модификация была предложена в работе [3].

Такая структура данных используется для добавления и удаления элементов в наборе данных.

В отличие от фильтра Блума, фильтр Блума с подсчётом представляет собой m счётчиков с несколькими битами вместо битов. Когда элемент добавляется или удаляется из набора данных, к элементу применяются k хеш-функций и все из k местоположений в массиве увеличиваются или уменьшаются на единицу. Операция поиска проверяет, что каждый из требуемых счётчиков не равен нулю.

Арифметическое переполнение счётчиков является проблемой, и счётчики должны быть достаточно большими, чтобы сделать этот случай редким. А слишком большие счетчики будут занимать лишнее место. Поэтому нужно правильно взять.

Если элемент больше не требуется учитывать, можно уменьшить соответствующие счетчики на единицу. Существует реализация под названием.

«Фильтр Кукушки» [4], имеющая меньшую вероятность ошибки при удалении, но обладающая низкой производительностью.

Литература / References

1. Bloom Burton H. Space/time trade-offs in hash coding with allowable errors, Communications of the ACM. – 1970. – Vol. 13(7). – P. 422–426.
2. ACMTrans. Comput Bigtable: A distributed storage system for structured data / F. Chang [et al.] // Syst. – 2008. – Vol. 26 (2). Art. 4. – 26 p.
3. Patgiri R. Role of Bloom Filter in Big Data Research: A Survey / R. Patgiri, S. Nayak, S.K. Borgohain // International Journal of Advanced Computer Science and Applications (IJACSA). – 2018. – Vol. 9. – № 11. – P. 655–661.
4. Zhang Y. Droplet: A distributed solution of data deduplication / Y. Zhang, Y. Wu, G. Yang // ACM/IEEE 13th International Conference on Grid Computing. – 2012. – P. 114–121.
5. Patgiri R. Preventing DDoS using bloom filter: A survey / R. Patgiri, S. Nayak, S.K. Borgohain // EAI Endorsed Transaction on Scalable Information Systems. – 2018. – Vol. 13. – P. e3.