Test Task – Bloom Filter

Overview

A Bloom filter is a **space-efficient probabilistic** data structure that is used to test whether an element is a member of a set. The tradeoff for that space efficiency is that it's probabilistic nature that means, there might be some False Positive results.

When you look up an item in a bloom filter, the possible answers are:

- It's definitely not in the set. This is a true negative.
- It might be in the set. This could be a false positive, or it could be a true positive.

Algorithm

An empty Bloom filter is a bit array of m bits, all set to 0. There must also be k different hash functions defined, each of which maps or hashes some set element to one of the m array positions, generating a uniform random distribution.

To add an element, feed it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

To query for an element (test whether it is in the set), feed it to each of the k hash functions to get k array positions. If any of the bits at these positions is 0, the element is definitely not in the set; if it were, then all the bits would have been set to 1 when it was inserted. If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive.

How big should be the Bloom Filter and Optimum no: of Hash functions

We can control the probability of getting a false positive by controlling the size of the Bloom filter. More space means fewer false positives. If we want to decrease probability of false positive result, we have to use more number of hash functions and larger bit array. This would add latency in addition to the item and checking membership.

Probability of False positivity: Let **m** be the size of bit array, k be the number of hash functions and **n** be the number of expected elements to be inserted in the filter, then the probability of false positive **p** can be calculated as:

$$P = \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^{k}$$

Size of Bit Array: **If expected number of elements** n **is known and desired false positive probability is** p **then the size of bit array** m **can be calculated as:**

$$m = -\frac{n \ln P}{(ln2)^2}$$

Optimum number of hash functions: The number of hash functions **k** must be a positive integer. If **m** is size of bit array and **n** is number of elements to be inserted, then k can be calculated as:

$$k = \frac{m}{n} ln2$$

Unnecessarily large number of hash functions can set more bits per element and ultimately increase the false positive probability. More generally, fewer than 10 bits per element are required for a 1% false positive probability, independent of the size or number of elements in the set.

Special Characteristics

- Unlike a standard hash table, a Bloom filter of a fixed size can represent a set with an arbitrarily large number of elements.
- Adding an element never fails. However, the false positive rate increases steadily as elements are added until all bits in the filter are set to 1, at which point all queries yield a positive result.
- Bloom filters never generate false negative result.
- Deleting elements from filter is not possible because, if we delete a single element by clearing bits at indices generated by k hash functions, it might cause deletion of few other elements.

Strengths:

- Space-efficient. Bloom filters take up O(m) space given m is the size of bitmap, regardless of the number of items inserted. (But their accuracy goes down as more elements are added.)
- Fast. Insert and lookup operations are both O(k) time given k is number of hash functions.

Weaknesses

- Probabilistic. Bloom filters can only definitively identify true negatives. They cannot identify true positives. If a bloom filter says an item is present, that item might actually be present (a true positive), or it might not (a false positive).
- Limited Interface. Bloom filters only support the insert and lookup operations. You can't iterate through the items in the set or delete items.

Usage

- Check whether a username is already occupied
- Check whether an IP is blacklisted
- Check whether a URL is malicious
- Identify whether some promotion or recommendation is already shown to a user.

Approach for Implementing Bloom Filter based Spell Checker (Kata 05)

We have been provided with a word list containing 338882 items as the dictionary to be used for this task. According to the theoretical details described above, we have come up with below figures to keep false positive probability at 0.05.

- Bitmap size (m) – 2113006
- No: of Hash functions (k) – 4.32

Since those are kind of odd figures, we have rounded those figures as below keeping the false positive probability as close as possible to 0.05.

- No: of elements to be inserted (n) = 338882
- False positive probability (p) = 0.051161858 (1 in 20)
- Bitmap size (m) = 2100000 (256.35KiB)
- No: of Hash functions (k) = 4

We could go for some more large figures to reduce the false positive probability, but that could increase the space utilization and reduce the space efficiency as opposed to the characteristics of Bloom Filter.

Implementation of Bloom Filter Data Structure `(BloomFilter.java)`

As described, Bloom filter is a that keeps an arbitrary set of elements as a bit array and facilitates inserts and lookups in a space efficient manner. In our implementation we have utilized BitSet class in Java to get the functionality of a bit array. One can use this BloomFilter class to implement a membership test of their own by providing bitmap size (m) and no: of hash functions (k).

Calculating hash values is implemented as below.

1. For an element, take the MD5 hash value.
2. Divide the MD5 hash value into k parts (dividing into equal length portions, last portion can be in a different length)
3. For each portion, get the numeric value and get the remainder of division by m (hash portion %m). This will ensure generated hash value falls inside the bitmap index range.
4. This will produce k values representing k indexes in bitmap.

Based on this hash mechanisms, Bloom filter will support its core operations.

- Insert(x): Get k hash values for x and set related indexes in bitmap.
- Lookup(x): Get k hash values for x and check whether all related indexes are set.

Implementation of Spell Checker Library `(SpellChecker.java)`

This class uses the above BloomFilter class as its data structure to store the provided data set and interfaces to do the lookup and determine false positivity. This class uses below default values.

- Bitmap size (m) = 2100000
- No: of Hash functions (k) = 4

- Data set – Word list provided in Kata 05.

One can use this library class and implement custom interfaces to do custom lookups and generate custom reports on the lookup results. Further, anyone can use desired values for above properties and use a data set of their own to utilize the functionalities provided by this library class. If those are not specified, default values will be used.

At the initialization, this will initialize the Bloom Filter with provided values or default values and then load the provided data set or default data set into Bloom Filter. Based on the loaded Bloom Filter, below operations will be available for anyone to utilize.

- Lookup(x): Wrapped interface that returns the result of Bloom Filter's lookup operation for x.
- isFalsePositive(x): Returns true if x is not available in the original data set or false otherwise.

Implementation of Part Two – Spell Checker Application

This part is implemented using two classes `Main.java` and `SpellCheckerApplication.java`. Main.java provide the main method to run at the execution. It can accept bitmap size, hash count and dictionary file as command line arguments. Based on the parameters it will initialize a SpellChekerApplication instance and invoke the start method. SpellCheckerApplication will initialize the SpellChecker library based on the inputs or null values will be passed to SpellChecker so that it will use default values. Once SpellChecker initialization is completed with loading the dictionary into Bloom Filter, SpellCheckerApplication will provide user with 3 options to do lookup as follows.

- 1 - Lookup a Custom Word
- 2 - Lookup a Random Word (5-Character)
- 3 - Lookup 10 Random Words (5-Character)

Once user enters the preferred option, it will do the lookup and displays the result as 'Found' or 'Not Found' along with small summary including total lookup count, total success count and total false positive count.

By running the application with different values to bitmap size and hash count, we can see accuracy is getting changes. Refer README.md to run the application.

Ex: - When we use 2000000 as bitmap size, lookup for 'aswgn' outputs a false positive. But when we use 2100000 as the bitmap size, lookup for 'aswgn' outputs a true negative.