*Overview of Bloom Filters:*

For effective set membership queries, probabilistic data structures called bloom filters are utilized. As they can represent a large set using a comparatively small amount of memory, this makes bloom filters especially helpful when memory is a constraint.

A Bloom filter's fundamental data structure is a bit array, in which each bit denotes a position in the set. Initialization sets all bits to 0. One or more hash functions are used to construct a set of places in the bit array for each element before adding it to the set. Then, we set each of these positions to 1.

Moreover, in terms of querying the same hash functions are applied to the element and the relevant points in the bit array are inspected to query the Bloom filter for set membership of the element. The element is assumed to be a member of the set with a high probability if all the bits are set to 1 in these positions.

The difference between bloom filters and counting bloom filters is that CBF utilizes the use of counters. This counter is in place of the standard (0 or 1) in each slot in the regular bloom filters. Therefore, unlike Bloom Filters, we can remove elements from the data structure (Alsayed, 2020).

*A deep dive into Counting Bloom Filters (CBFs):*

To efficiently storing and retrieving set membership, counting bloom filters (CBFs), a sort of probabilistic data structure, is used. Although they are similar to Bloom filters, they also can keep a counter for each hash value in the filter. The number of times an item has been added to the set may now be estimated with greater accuracy (Alsayed, 2020).

CBFs are built using an array of counters with a zero initial value. A set of hash functions are applied to each item when it is entered into the CBF, and the relevant counters in the array are increased. The number of bits that are set in the CBF for each item depends on the number of hash functions that are employed. A position in the array of counters is represented by each bit in the CBF (Kirushikesh, 2022).

The same hash functions are applied to the item to be queried to query the CBF for set membership, and the associated counters in the array are then analyzed. The item is assumed to be a member of the set with a high probability if all the counters for the hash values connected to it are non-zero. The number of times an item has been added to the set can also be calculated using CBFs. To achieve this, check the values of the counters connected to the item's hash values. The item is regarded as having been added to the set at least once if each counter has a value other than zero. The total of the counter values provides an approximation of how many times the item has been added to the set overall.

It is important to note that false positives are possible, though, because many things can hash to the same CBF counts. The size of the counters and the quantity of hash algorithms employed can be changed to reduce the likelihood of a false positive.

*A look into the use of hashing functions:*

An arbitrary-sized input (such as a string or file) is mapped to a fixed-size output (a hash value or hash code) using a hash function. Typically, the output is a series of bits that serve as a distinct digital fingerprint of the incoming data. The hash function should be deterministic, which means it should consistently generate the same output for a given input.

In a Counting Bloom Filter (CBF), the hash function's job is to map an element to one or more locations in the array that corresponds to the CBF. Every location in the array acts as a counter that can be increased or decreased to indicate if an element is present or absent from the CBF.

The hash function is used to generate one or more hash values when an element is added to the CBF. A place in the array is indexed using a hash value for each element, and the counter at that position is incremented to indicate the element's presence in the CBF.

The hash function is once more applied to the element to produce one or more hash values for determining whether an element is a member of the CBF. Next, we determine whether the counters at the hash value positions are greater than 0. The element is probably a member of the CBF if all the counters are larger than 0 (however there is a potential of false positives). We can conclude that an element is unquestionably not a member of the CBF if any of the counters are 0.

From the video that professor showed I will draw a diagram of the hash functions on how it works

Analysis of operations asymptotic order of growth:

The three main operations utilized in CBFs are: insertion, search and delete

*Insert:* To add a new item to the CBF, increase the counters corresponding to its hash values. This operation has an O(k) time complexity, where k is the total number of hash functions employed.

*Search:* Check the counters linked to the item's hash values to see if it is present in the CBF. The item is most likely part of the set if all the counters are greater than zero. The item is not part of the set if any of the counters are zero. This operation's temporal complexity is also O(k).

*Delete:* Decrease the counters linked to an item's hash values to remove it from the CBF. This operation's temporal complexity is also O(k).

*Practical, real-life computational applications of CBF:*

Plagiarism

By presenting each document as a set of distinctive words or phrases that occur in the document, counting Bloom Filters (CBFs) can be utilized for plagiarism detection. CBFs can be used to represent the sets, enabling effective set membership queries to determine whether two documents share any text and may therefore be plagiarized. We can represent two documents D1 and D2 as sets S1 and S2, respectively, to identify plagiarism between them. We can add the S1 items to a CBF and then determine whether S2 elements are already present. If any element is discovered, it means that S1 and S2 share at least one word or phrase, and more investigation can be done to evaluate whether plagiarism has taken place. To determine the level of plagiarism, CBFs can effectively estimate the number of times an item has been added to a set. For instance, if a term or phrase appears more than once in both texts, that could be a sign of more plagiarism than if it only appears once. This is especially useful for schools and universities when trying to check the extent of plagiarism in a student’s assignment.

Add another example

Networking traffic/accidents/bandwidth consumption optimization

*Conclusion:*

In conclusion, CBFs are an effective data structure for approximation set membership queries, particularly when the size of the set is huge or unknowable. They do, however, have a significant false positive rate, which must be considered while creating and utilizing CBFs.

*Implementation of the CBFs in Python using the code template, with appropriate docstrings and meaningful comments and a thorough justification for your choice of hash functions.*

*References:*

<https://medium.com/analytics-vidhya/cbfs-44c66b1b4a78#:~:text=Counting%20Bloom%20Filters%20use%20the,not%20possible%20in%20Bloom%20Filters>.

<https://www.geeksforgeeks.org/counting-bloom-filters-introduction-and-implementation/>