Indian Institute of Technology Gandhinagar

Accuracy and Applications of Probablistic Data Structures

MA202 Project Report

Members

Aaryan Darad (21110001)
Abdul Qadir Ronak (21110003)
Abhay Kumar Upparwal (21110004)
Vaibhavi Sharma (21110231)
Trushika Parmar (21110150)

Under the guidance of

Prof. Anirban Dasgupta

CONTENTS

1) Abstract
2) Introduction
3) Methodology
4) Implementation
5) Applications
6) Acknowledgments and References

ABSTRACT

There are many Data Structures that we come across from stacks, queues, heaps to sets and hashmaps. The applications in which they are used are immense and quite useful for various problems in not only computer science but in real life as well. In this project we are working on the implementation of various advanced data structures such as Count-Min Sketch, HyperLogLog and Bloom Filter. These are also called probabilistic or randomized data structures. We have studied various aspects of probability and statistics and now we are going to check the statistical accuracy of these data structures with respect to the space of data used.

INTRODUCTION

Count-Min Sketch:

Count-Min Sketch is a probabilistic data structure used to estimate the frequency of elements in a data stream. It is often used in situations where it is not feasible to store the entire data stream due to its size or the high rate of incoming data.

The Count-Min Sketch consists of an array of counters, where each counter corresponds to a particular hash function. When an element is added to the sketch, it is hashed by each of the hash functions, and the corresponding counters are incremented by one. To estimate the frequency of an element, the sketch hashes the element again using the same hash functions and returns the minimum value of the counters corresponding to those hash functions. [1]

The Count-Min Sketch has the benefit of just requiring a fixed amount of memory, irrespective of the size of the data stream. The maximum number of unique elements in the stream and the desired level of accuracy in the frequency estimations are used to define the size of the sketch. It is also mentioned in a source[2] “The data structure is designed in a way that allows freely trading off accuracy and space requirements.”

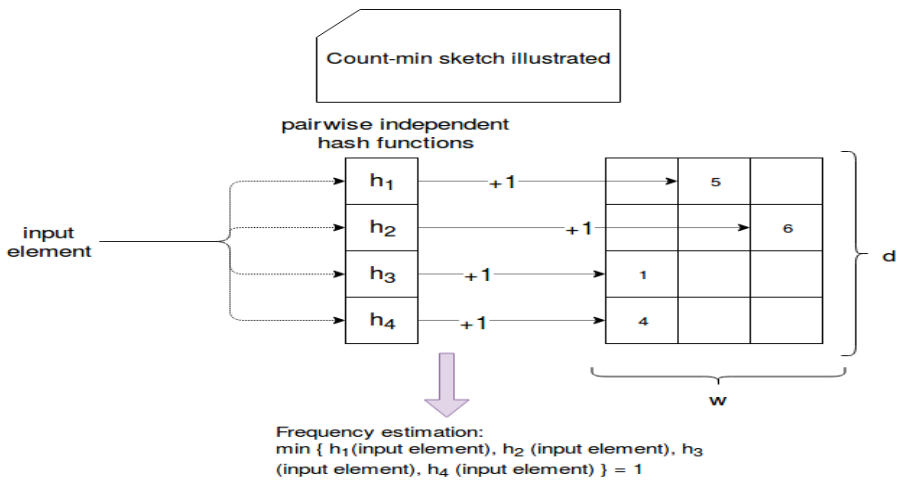
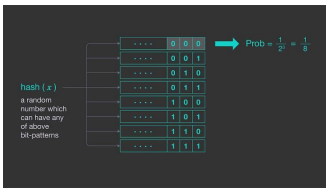


Image source: https://www.waitingforcode.com/public/images/articles/count_min_sketch.png

HyperLogLog:

HyperLogLog is a probabilistic algorithm used to estimate the cardinality of a large set. One way to explain how it works is to say that it takes the average of the number of trailing zeroes in the binary representation of a set of hashed values.



To elaborate, the algorithm works by first hashing the input values and then mapping the hash values to a binary string. The algorithm then examines the binary representation of each hash value and counts the number of trailing zeroes in the string. The maximum number of trailing zeroes is then recorded as an estimate of the cardinality of the set.

The reason this works is that the number of trailing zeroes in the binary representation of a number is related to the magnitude of the number. Specifically, for a uniformly distributed set of hash values, the number of values with a certain number of trailing zeroes will be proportional to 2^(−k), where k is the number of trailing zeroes. By taking the average of the number of trailing zeroes across all the hash values, HyperLogLog is able to estimate the overall magnitude of the set.
Overall, the algorithm is able to provide a good estimate of the cardinality of large sets with a relatively small amount of memory, making it a useful tool for data analysis and processing. [10]

Memory Requirement - log(log(list))

Bloom Filter:

Bloom filter is a probabilistic data structure, invented by Burton Bloom. It is used to test whether an element is a member of a set or not. It works on the concept of hashing. A bloom filter is a n-bit array with multiple number of hash functions. Each element added is hashed and is mapped to the n-bit array. Bloom filters are space-efficient, but there is a problem of false positives. The bloom filter can generate false positives, i.e. it can show that an element is present in the input dataset, though actually it is not a member. Thes false positives have a probability and it depeends on multiple paraemters of the bloom filters. So we can predict the rate of false positive, by changing these parameters.

METHODOLOGY

Count-Min Sketch:

The Count-Min Sketch data structure is a probabilistic algorithm for summarising and estimating the frequency of items in a data stream. The methodology for implementing a Count-Min Sketch can be summarised in the following steps as referred by a source[4]:

1. Initialise an m x k array of counters, where m is the number of hash functions used and k is the size of each counter.
2. For each item in the data stream, hash it with each of the m hash functions, and increment the corresponding counter in the array.

Image source: https://www.researchgate.net/publication/325965443/figure/fig2/AS:641247416954880@1529896706559/Examples-of-count-min-Sketch.png

3. To estimate the frequency of an item, hash it with each of the m hash functions and take the minimum value of the counters.

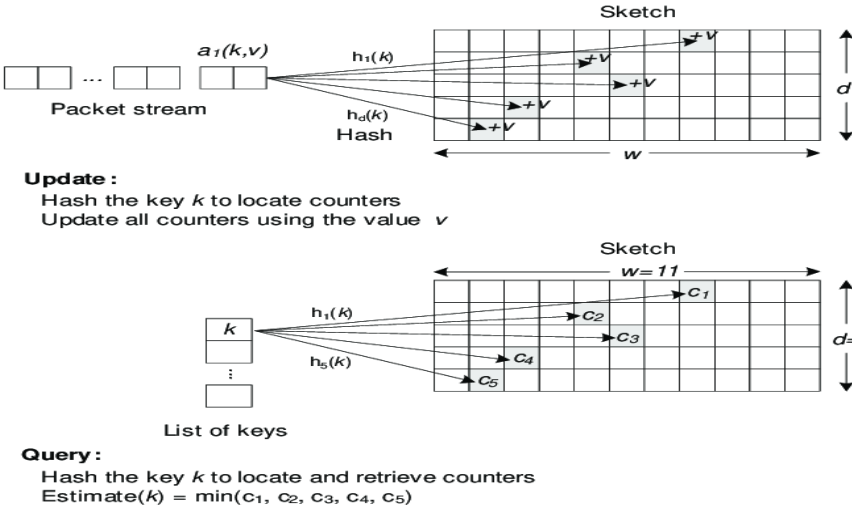


Image source:
https://www.researchgate.net/publication/334379141/figure/fig2/AS:779283882577923@1562807164367/Sketch-update-and-query-operations-Count-Min-sketch.png

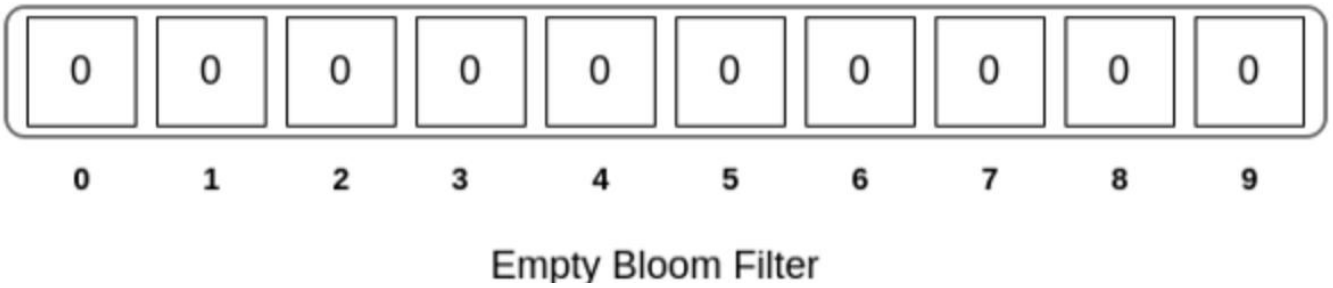
4. To handle deletions, decrement the corresponding counters instead of incrementing them.

The size of the array and the number of hash functions used are chosen to balance the trade-off between accuracy and space complexity.

Bloom Filter:

The Bloom filter is a probabilistic data structure that can efficiently test whether an element is a member of a set. However, it may occasionally report false positives, which means that it may mistakenly indicate that an element is in the set when it is not. Bloom filter's working is given in the points below

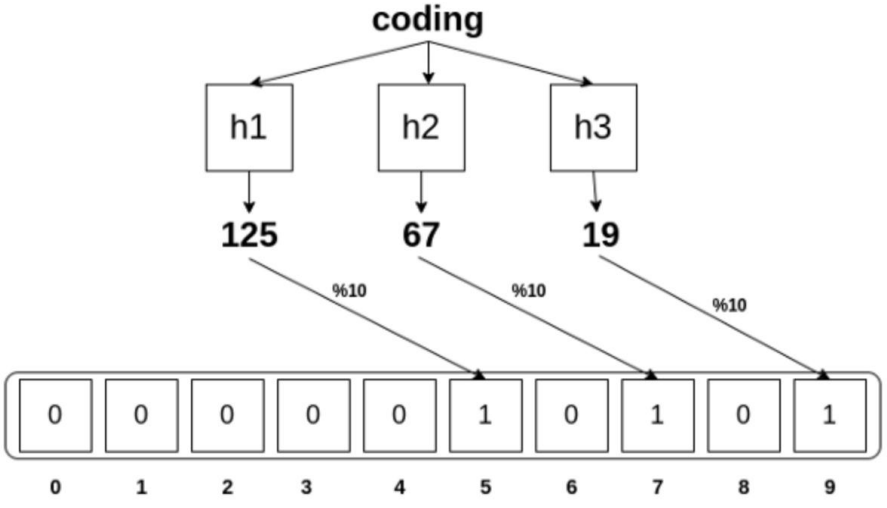
A bloom filter comprises a bit array of size m and k hash functions. The hash functions map each input data element to k positions in the m-bit array. The hash functions povide k different indices , corresponding to which the bits in the bit array are turned to 1.



Empty Bloom Filter

Image source: [7]

When testing for certain elements, the element is again hashed k times giving k different indices, which are then checked in the m-bit array. If the bit value of filter(array) of the correspondong indices are 1, then, the element is a member of our input dataset. But there are chances for getting false positives, i.e. the filter can show that an element is present in the input dataset, though it is not there.



Inserting in bloom filter

Image source: [7]

The false positives happen because of collisions of hash values. The filter can show that an element is a member of dataset by checking the indices given by the hash functions. But the indices could have been changed to 1 by other elements since each element is mapped to k different indices, thus though an element is not present in the set, its corresponding indices by hash functions can falsely show that it is present in the dataset. But the fase positives can be predicted. We can calculate the probability of false positives. The probability depends on size(width) of filter, number of hash functions and total number of elements in the set.

HyperLogLog

The basic idea behind the algorithm is to first count the number of leading zeroes in the binary representation of a hash value for each element in the set. The maximum number of leading zeroes found for any element is then used to estimate the cardinality of the set.

However, the above statement of "first counting the number of zeroes then taking their expected value" requires a bit more explanation. In the HyperLogLog algorithm, the number of leading zeroes is not counted directly, but rather approximated by calculating the position of the first non-zero bit in the hash value. This value is then used to assign the element to one of several virtual buckets based on the number of leading zeroes in its binary representation.

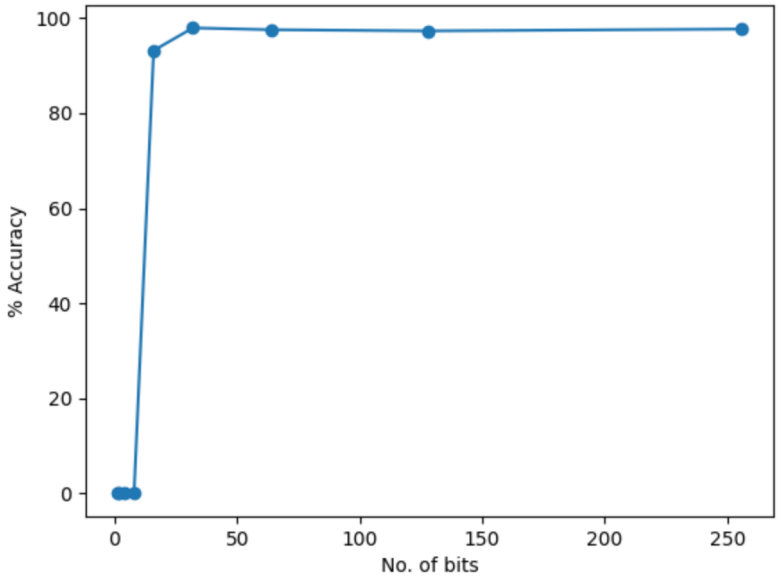
The number of buckets and their size is determined by a parameter called the precision. A higher precision allows for a more accurate estimate of the cardinality, but requires more memory and processing time.

Once all the elements have been assigned to virtual buckets, the algorithm calculates the expected value of the cardinality based on the number of buckets that have at least one element assigned to them.

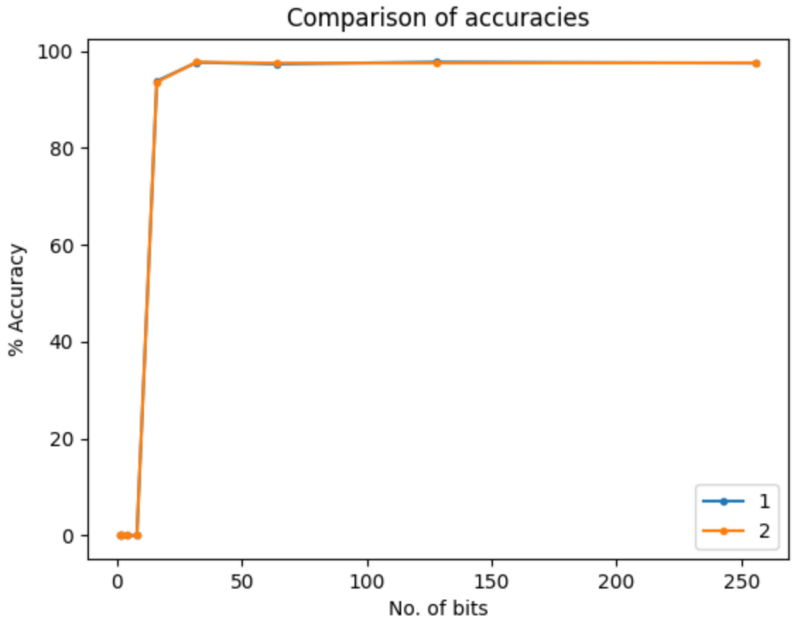
1. Generate test sets: Generate num_sets of sets with varying sizes (e.g., 1000, 10000, 100000 elements) using a random number generator. The maximum size of the set should be 2^(2^(j-1)) - 1, where j is a variable that ranges from 1 to 9.
2. Estimate the cardinality: For each set, use the HyperLogLog algorithm to estimate the cardinality of the set.
3. Calculate the error: Calculate the absolute error between the estimated cardinality and the actual cardinality of the set. % Error = 100* abs(actual_cardinality - estimated_cardinality) / actual_cardinality. % Accuracy = 100 - % Error
4. Compute the mean error: Compute the mean error across all sets for each value of j.
5. Plot the results: Plot the mean error against the number of bits (b). The plot will show how the accuracy of the algorithm changes as the number of bits changes.

IMPLEMENTATION

HyperLogLog



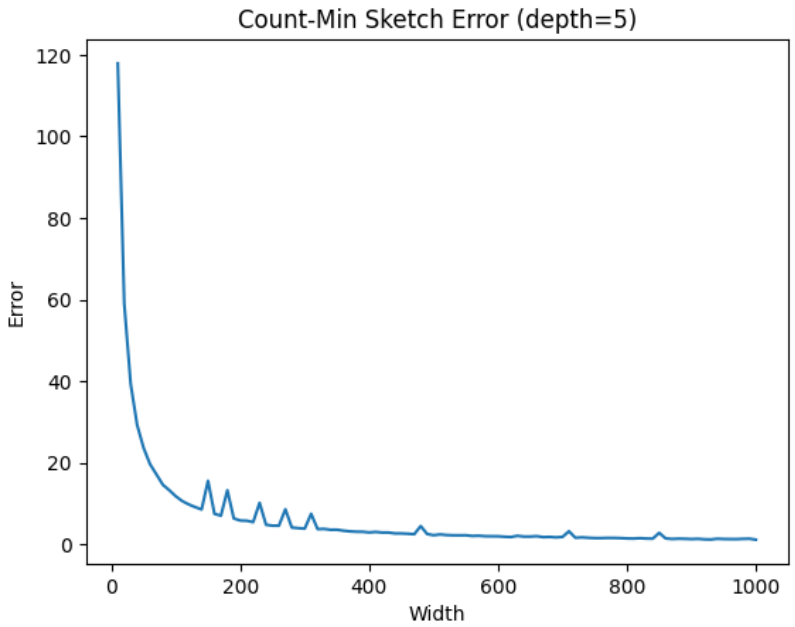
In the above plot we can see as the no. of bits increases accuracy increases, there is a sharp rise near 16 bits.



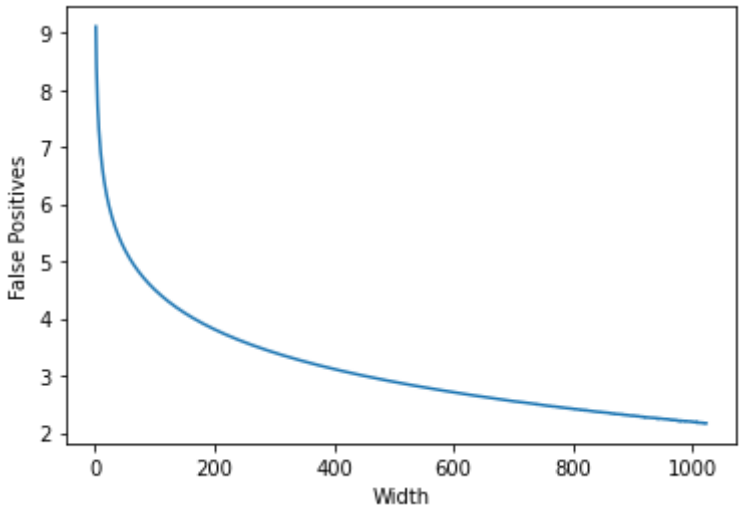
In the above plot blue represents the normal plot whereas the orange represents the expected value of least 70% trailing zeroes.

Count-Min Sketch

As width increases, so does the frequency estimating error decrease. This is because, as the number of total possible values of hashing a number (width) increases, the possibility of overlapping of hash values decreases and leads to a more accurate representation. However, this also results in the increase in time complexity significantly.



Bloom Filter:



The y-axis represents the rate of false positives and the x- axis represents the width(size) of the filter. As shown in the graph, the rate of false positives decreases with respect to increasing size of filter.

APPLICATIONS

Count-Min Sketch:

The Count-Min Sketch is widely used in a variety of applications, including network traffic monitoring, web analytics, and natural language processing.

According to a source[2], we can say that Count-Min Sketch is helpful whenever we are just interested in approximating counts of the most crucial components. The fact that the data structure can be filled in first and then afterwards queried with appropriate keys is a feature that enables a wide range of applications. We don't first need to be aware of the keys that interest us in the initial stage.

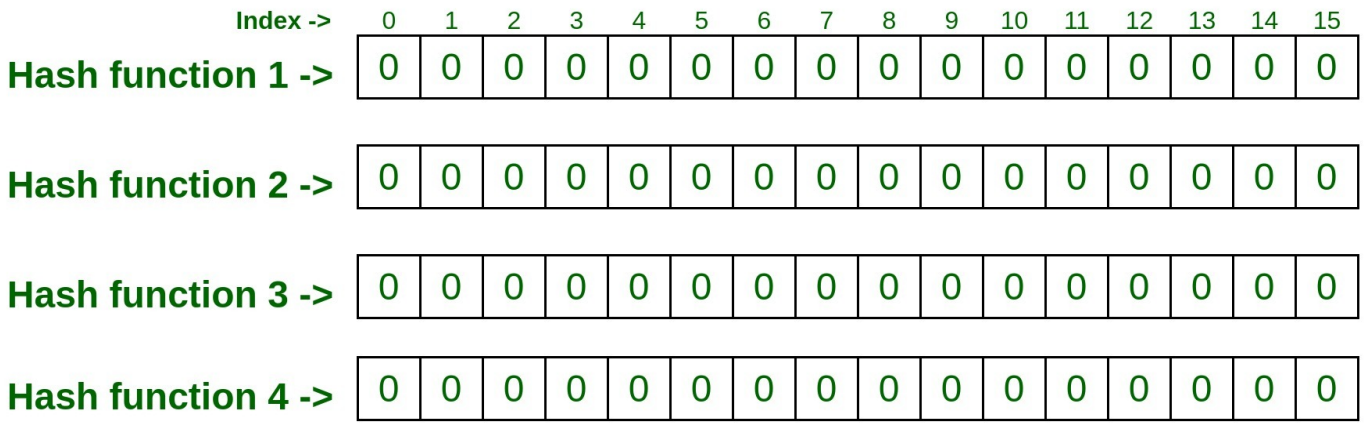


Image source: https://media.geeksforgeeks.org/wp-content/uploads/20200414202244/Count-min-sketch.jpg

1. **Database Query Planning**

   Database engines exploit the count-min sketch to identify effective execution paths.
   In order to retain two drawings, one for the occurrences of n in a and the other for n in b, we can maintain a query like select * from a, b where a.n = b.n. Hence, to determine the size of the join, we may then query these sketches.

2. **To make a password strong**

   For a certain website, we can keep track of how frequently a password was used.

In this case, the frequency is the amount of times the supplied string was used as a password, and the password is the object. [3]

3. **Finding Heavy Hitters**

Heavy Hitters are the elements with great frequency. For instance, we might want to identify the most popular pages and how frequently they were visited given a massive log of website visitors. Furthermore, if there is a big tail of unpopular pages with low traffic,

building up a comprehensive hash table may not scale well. We only need to iterate through the log once to develop the sketch and address the issue with Count-Min Sketch. We need to think of potential keys to search for in order to query the sketch. If we don't already have a candidate set, we can just go over the log once more and look up each page in the Count-Min Sketch, keeping in mind the most crucial ones. [2]

Bloom Filter:

A Bloom filter is a data structure with many practical applications. By using Bloom filter, an element can be rapidly found among a huge number of other elements. This data structure is very convenient and helpful when a large amount of data needs to be searched and there is very limited memory on the device. Its applications can be found in network routers, databases, weak password detection, cyber security and web browsers, just to name a few. [6][7] Some of these applications are explained in detail below:

1. Network related applications

Bloom filters have many network-related applications. Here are some examples:

Caching: Bloom filters can be used in caching to quickly determine whether a requested item is in the cache or not. This can reduce the number of expensive disk or network accesses needed to retrieve the item.

Denial of service (DoS) protection: Bloom filters can be used to detect and block IP addresses that are sending a high volume of requests, which can help prevent DoS attacks.

Web search: Bloom filters can be used to quickly check whether a search term appears in a search index, allowing for fast retrieval of search results. [9]

2. Wrong password detection

Bloom filters can be used in wrong password detection systems to efficiently detect and block attempts to login with incorrect passwords. To start, the system stores a set of hashed passwords in a Bloom filter. Each hash value is stored as a bit in the filter. When a user attempts to login with a password, the system hashes the password and checks whether the resulting hash value is present

in the Bloom filter. If the hash value is not present, the password is likely incorrect and the system can reject the login attempt.

The use of a Bloom filter allows for fast and efficient checking of password hashes, as the filter can quickly indicate whether a hash value is likely to be in the set or definitely not in the set. [7]

3. Cyber Security

Bloom filters have several cyber security applications. Few examples of this are as follows:

Malware detection: Bloom filters can be used in antivirus software to efficiently test whether a given file or program is malware, by storing a set of known malware signatures, and checking whether a file's signature matches any of the malware signatures.

Spam filtering: Bloom filters can be used to identify known spam email messages by maintaining a set of known spam phrases or sender addresses. Any incoming email that matches any of the phrases or addresses in the filter can be flagged as spam.

File deduplication: Bloom filters can be used to identify duplicate files or data within a system, which can help to prevent redundant storage and reduce the risk of data breaches. [7]

HyperLogLog :

Here are some common applications of HyperLogLog:

1. Web analytics: To calculate the number of distinct visitors to a website, HyperLogLog is frequently used in web analytics. This helps companies in better understanding their target market and streamlining their marketing initiatives.
2. Network traffic analysis: The number of unique IP addresses in a network flow can be estimated using HyperLogLog, which is helpful for finding anomalies, spotting potential security concerns, and improving network efficiency.
3. Database indexing: Compact indexes for huge datasets are produced using HyperLogLog, which can drastically minimise the amount of memory needed to store the indexes.
4. Big data processing: By breaking up huge datasets into smaller chunks and generating the HLL estimate for each chunk, HyperLogLog is used to handle large datasets parallelly. The processing time is sped up, and less memory is needed as a result.
5. Search engine optimization: In order to increase the relevance of search results and enhance search engine performance, HyperLogLog is used to estimate the number of unique queries in a search engine.

ACKNOWLEDGEMENTS

The members of our group would like to thank Prof. Anirban Dasgupta, as we learned a lot about advanced Data Structures from this project to understand the application of Probability and Statistics. We would also like to thank Prof. Tanya Kaushal Srivastava to give us an opportunity to apply what we learnt in the first half of course MA202. The YouTube channel Niema Moshiri helped us a lot to get a better understanding of the working principle of these advanced data strcutures, in addition to this. Few other online sources like Wikipedia and Github Pages helped us understand the concept in detail.

REFERENCES

[1] "*Count-Min Sketch*" , From Wikipedia, the free encyclopedia, available at: https://en.wikipedia.org/wiki/Count%E2%80%93min_sketch

[2] *Count-Min Sketch* Available at: https://florian.github.io/count-min-sketch/ (Accessed: March 7, 2023).

[3] "*Count-Min Sketch for Beginners*" Available at: https://medium.com/@nehasingh18.9/count-min-sketch-for-beginners-f1e441bbe7a4 (Accessed: Mar 7, 2023).

[4] Cormode, G., & Muthukrishnan, S. (2005). *An improved data stream summary: the count-min sketch and its applications.* Journal of Algorithms, 55(1), 58-75.

[5] Bloom filter (2023) Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Bloom_filter#Extensions_and_application (Accessed: March 10, 2023)

[6] Prasanna (no date) Bloom filter data structure: Implementation and application, enjoyalgorithms. Available at: https://www.enjoyalgorithms.com/blog/bloom-filter/ (Accessed: March 10, 2023).

[7] Foundation, O.G. (2020) Applications of bloom filter, OpenGenus IQ: Computing Expertise & Legacy. OpenGenus IQ: Computing Expertise & Legacy. Available at: https://iq.opengenus.org/applications-of-bloom-filter/ (Accessed: March 10, 2023).

[8] Bloom filters - introduction and implementation (2022) GeeksforGeeks. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/ (Accessed: March 10, 2023).

[9] Network applications of Bloom Filters: A survey - harvard university (no date). Available at: http://eecs.harvard.edu/~michaelm/CS222/bloomsurvey.pdf (Accessed: March 10, 2023).

[10] (no date) YouTube. YouTube. Available at: https://www.youtube.com/?watch= (Accessed: March 10, 2023).

[11] 胡程維, C.-W.H.丨 (2021) HyperLogLog: A simple but powerful algorithm for Data scientists, Medium. Towards Data Science. Available at: https://towardsdatascience.com/hyperloglog-a-simple-but-powerful-algorithm-for-data-scientists-aed50fe47869 (Accessed: March 10, 2023).