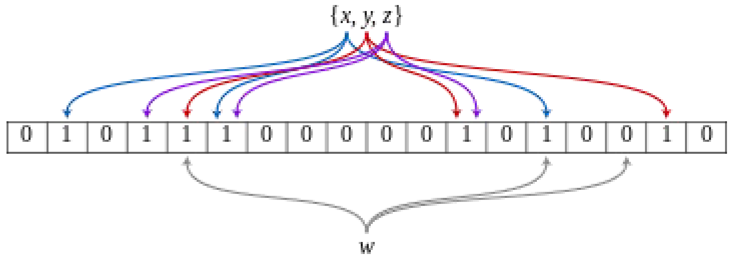
Algorithmic Problem Solving
17ECSE309

Blooms Filter

USN : 01FE15BCS130
Name : Pooja S Galer

The Problem

- A **Bloom filter** is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set.



- **Previous methods:**
  - Linear Search: Iterating through all data entries and checking if present
  - Binary Search: Storing all data values in one standard sequence (Increasing or decreasing order), and searching from the center.

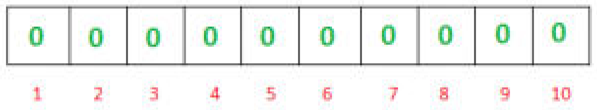
The Solution

- Consider a scenario of checking username while account creation:
  - 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.
  - Typically, *k* is a constant, much smaller than *m*, which is proportional to the number of elements to be added; the precise choice of *k* and the constant of proportionality of *m* are determined by the intended false positive rate of the filter
- For every k hash function the resulting index of bit array is set to 1. i.e., k indices are set to 1 for every insertion.
- The operations allowed are insertion and search.
- Every index value in bit array represents presence of an object.

- 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, i.e., telling you that a username doesn’t exist when it actually exists.
- Deleting from the bit array is not valid. Because the value of array can be indication of more than 1 objects. If for a particular object deletion happens in array the presence of another object fails.

Example

A empty bloom filter is a **bit array** of **m** bits, all set to zero



We define **k** hash functions:

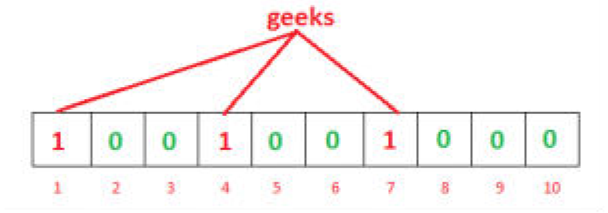
h1(x), h2(x), …. hk(x)

Insertion:

h1(geeks)=4

h2(geeks)=1

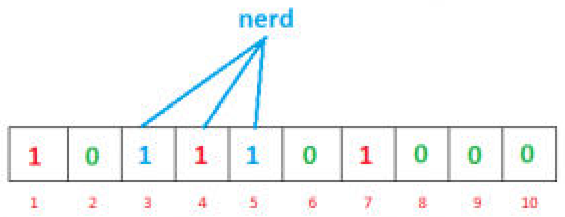
h3(geeks)=7



h1(nerd)=4

h2(nerd)=3

h3(nerd)=5

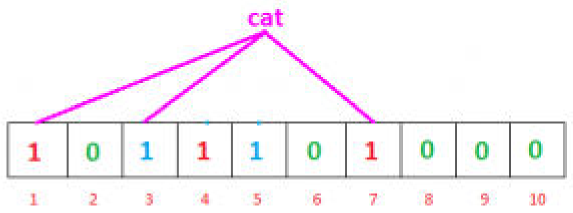


Search:

To check the presence of ‘geeks’, we do the same thing
i.e., calculate the hash function values and check the existence of that output bits as 1.
If all are 1 then the element is present.

False Positive results:

‘Probably present’
For example, check the presence of ‘cat’
h1(cat)=7
h2(cat)=1
h3(cat)=3



This is because of definition of **k** hash functions.

Note: The results cannot be **false negative** i.e., object exists but the search operation turns out to be not present

Insertion and Search modules:

```
def add(self, item):
        '''
        Add an item in the filter
        '''
        digests = []
        for i in range(self.hash_count):

            # create digest for given item.
            # i work as seed to mmh3.hash() function
            # With different seed, digest created is different
            digest = mmh3.hash(item,i) % self.size
            digests.append(digest)

            # set the bit True in bit_array
            self.bit_array[digest] = True
```

```
def check(self, item):
        '''
        Check for existence of an item in filter
        '''
        for i in range(self.hash_count):
            digest = mmh3.hash(item,i) % self.size
            if self.bit_array[digest] == False:

                # if any of bit is False then,its not present
                # in filter
                # else there is probability that it exist
                return False
        return True
```

Application

- Medium uses bloom filters for recommending post to users by filtering post which have been seen by user.
- Quora implemented a shared bloom filter in the feed backend to filter out stories that people have seen before.
- The Google Chrome web browser used to use a Bloom filter to identify malicious URLs
- Google BigTable, Apache HBase and Apache Cassandra, and Postgresql use Bloom filters to reduce the disk lookups for non-existent rows or columns.

References:

- https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/
- https://en.wikipedia.org/wiki/Bloom_filter