Boolm filter

space-efficient probabilistic DS.

uesd to test whether an elem is a member of a set

BF.find(elem) == BE.end(), result may return true

~~BF.find(elem) != BE.end(), result return false~~

deleting elements from filter is not possible

```
init() {
        an bit array with m bits
        k hash function
}
```

```
add(elem) {
        for each (K in k-hash-function) {
                setBit(K(elem) % m);
        }
}
```

```
find(elem) {
        let res = true
        for-each(K in k-hash-function) {
                res &= K(elem);
        }
        return res;
}
```

use-case:

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 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

HW: use bayen to prove probability of false positivity:

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 :

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

Streaming algorithms

streaming algorithms are algorithms designed to process a sequence of data in a single pass. In such tasks, memory is always limited (the complexity is typically constant or logarithmic with respect to the input size.

Window task

where you are given a stream of numbers and a number k, and after each new number a_id in the stream, you need to output a function computed over the last min(k, id) numbers

rigorously /ˈrɪɡ.ər.əs.li/ (adv): một cách nghiêm khác
def: in a careful way so that every part of something is looked at or considered to make certain it is correct or safe

:((

The practical appication of the problem of counting 1 bits in a window is quite important. For example, using this problem, we can determinate the amount of good traffic (e.g., succcessfully completed requests) out of the last n requests.

Count number of distinct elems

Solution:

let assign a random hash-value to each number

find the "maximum" number of leading zeros among all hash-values

then the number of unique elems is ~ 2 ^ zeros

because every prefix has the same probability

in practice, instead of simply by using 2^zeros, we ususally take 2^zeros/phi

(phi ~ 0.77351)

the value of phi is determine empirically /ɪmˈpɪr.ɪ.kəl.i/ (dựa trên quan sát, thực nghiệp hơn là dựa trên lý thuyết)