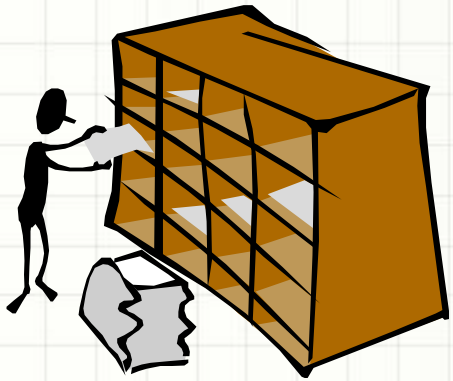
Bloom Filter

By Robert Gutierrez

Background: Hash Table

- Maps keys into an array
- Hash function create a unique key for each input
- Look up speed on average is O(1)
- Problem: Collision Occur often



Bloom Filter

- Created by Burton Howard Bloom
- Can determine if an object is in not in the set
- Space Efficient and Time Efficient
- 100% recall rate
- Issue: Not able to delete

How it works

- A bloom filter is constructed using a bit array
- Contains more than one hash function
- Caution: There needs to be a perfect balance, between Hash function and array size.

Calculating: Bit Array

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

- P = acceptable false positive
- n = items expected

Calculating: Hash Functions

$$k = \frac{m}{n} \ln(2)$$

- *m* = bits needed
- *n* = Number of elements

False Positive Rate

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

- k = number of hash functions
- n = Number of elements
- m = bits in filter

Hashing the Bit Array

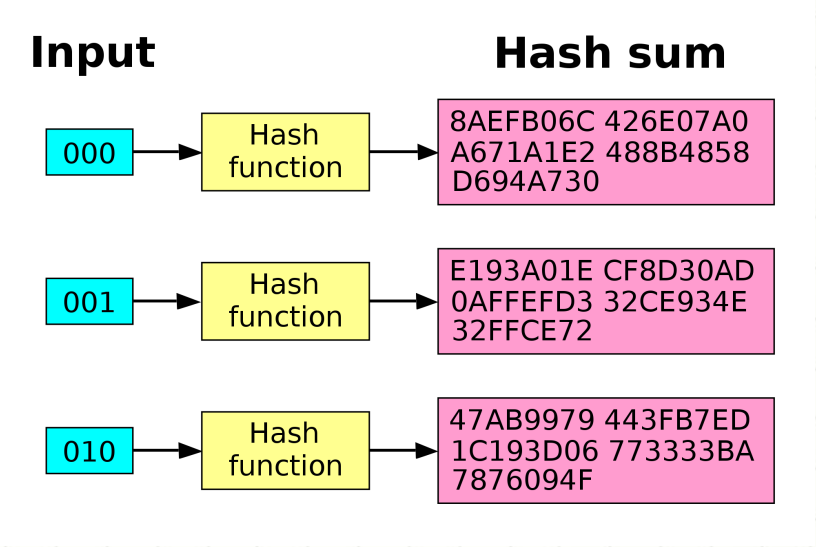
```
def mark(self, string):
    mark_here = self.encrypt(string) # create hash key
    # end mark

def encrypt(self, string):
    crypt_sha1 = hashlib.sha1() # call secure hash algorithm
    crypt_sha1.update(string)
    key = crypt_sha1.hexdigest() #  gives a hexadecimal number
    key = int(key, 16)           #  convert the number

    crypt_mmh3 = mmh3.hash128(string) # second hash function

    # Hash Equation: gi(x) = h1(x) + ih2(x) % m
    for i in range(self.num_hash):
        location = (key + i * crypt_mmh3) % self.size
        self.bloom_array[location] = True

    # end encrypt
```

Does it exist?

```
def lookup(self, string):
    am_I_here = True                  # Assume the function exist
    am_I_here = self.decrypt(string) # create hash key and search

    if am_I_here == True:
        return True

    else:
        return False

    # end lookup


def decrypt(self, string):  # simlar to encrypt
    crypt_sha1 = hashlib.sha1() # secure hash Alg. method use again
    crypt_sha1.update(string)
    key = crypt_sha1.hexdigest() #  gives a hexadecimal number
    key = int(key, 16)           #  convert the number

    crypt_mmh3 = mmh3.hash128(string) # second hash algo

    # if this loop fails  once, we can be certain it does not exist
    for i in range(self.num_hash ):

        location = (key + i * crypt_mmh3) % self.size
        if self.bloom_array[location] == False:
            return False

    return True             # return True if found otherwise breaks
```

So what is mmh3?

- Murmur Hash is a non crypto graphic hash function
- Performs well in Random distribution
- Performs well in Avalanche behavior
- Layman's term: it multiplies and rotates the bits.



Cryptographic Hash Functions

- Some names are MD5 and SHA1
- Convert strings into ASCII, then into binary.
- Then use logical operation in order to scramble them. Ex. *x and y, x or y, x xor y, not x*
- *Goal is to be a multiple of 512 (sha1)*
- *They always digest a 160 bit number (sha1)*

Bloom Filters in the Wild!

- Web browser: Commonly used to detect if a website is malicious
- Facebook: Search Bar, to fetch friends and friends of friends to add them to a users query.
- Yahoo: request a bloom filter when accessing your email. To verify that people are in your address book.

Thank You !

Questions?