Bloom Filters

kind of a "probabilistic dictionary"

- maintain the "fingerprints" of the elements of a set S
- `search(S, x)`
  - returns `no` if $x \notin S$
  - returns `probably_yes` if $x \in S$, but also sometimes when $x \notin S$

Applications:

- hyphenation
- checking URLs in case they are malicious without storing entire set of malicious sites
  - let a user visit the URL if it is not malicious
  - warn the user if the URL might be malicious
  - can accomplish this with a 10 mb bloom filter, rather than a 500 mb list of sites

Implementation

- bit array `BF` of size m, so we have `BF[0]`, `BF[1]`, ... `BF[m-1]`
- t independent hash functions $h_1, h_2, \ldots, h_t$
  - $h_i : U \to \{0, 1, \ldots, m-1\}$
  - h_i conforms to simple uniform hashing assumption
- `insert(S, x)`

```
def insert(S, x):
    for i in range(t):
        BF[hi(x)] = 1
```

- `search(S, x)`

  ```
  def search(S, x):
      for i in range(t):
          if BF[hi(x)] == 0:
              return false
      else:
          return true
  ```

  - takes $\mathcal{O}(t)$ time

Probability of a false positive

If n is the number of insertions, t is the number of hash functions, m is the size of BF

$$P(BF[i]=0) = P\left(\bigcap_{k=1}^{n}\bigcap_{j=1}^{t} h_j(x_k) \neq 1\right)$$

$$= \prod_{k=1}^{n}\prod_{j=1}^{t} P(h_j(x_k) \neq 1)$$

$$= (1-1/m)^{nt}$$

$$\approx (e^{-1/m})^{nt} = e^{-nt/m}$$

Let $q = P(BF[i]=1) = 1 - e^{-nt/m}$

$$P(\text{false positive}) = P(BF[h_1(x)] = 1 \cap BF[h_2(x)] = 1 \cap \ldots \cap BF[h_t(x)] = 1)$$

$$\approx P(BF[h_1(x)] = 1) \cdot P(BF[h_2(x)] = 1) \cdot \ldots \cdot P(BF[h_t(x)] = 1)$$

$$= q^t = (1-e^{-nt/m})^t$$

Using calculus, to minimize $P(\text{false positive})$, we should choose $t = \frac{m \ln 2}{n} \approx 0.69\frac{m}{n}$