Probabilistic Data Structures

Amit Kol
Membership testing
An array of \( m \) elements and a hash function \( h \)

- How do we keep track of collisions?
  - How expensive is it?
  - What if we don’t keep track?
Use $k$ hash functions $h_1, h_2, \ldots, h_k$ on a bit array

- No false negatives
- Saves space
- Constant time to add an element
Bloom filter – false positives

After \( n \) insertions,

\[
Pr(bit = 0) = \left(1 - \frac{1}{m}\right)^{kn}
\]

Probability of false positive:

\[
\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k
\]
Use in streaming scenarios
Scalable Bloom filter

- Add an arbitrary number of elements
- Constant bound on false positives
- Becomes expensive in terms of space
Stable Bloom filter

Goals:

- Use constant memory
- Evict stale data
Stable Bloom filter

Goals:

- Use constant memory
- Evict stale data

Results:

- The number of 0s in the array converges
- We can use this to limit false positives
- False negatives are introduced
How can we save more information?
We’d like to get a histogram of the elements in the stream

- Point queries
We’d like to get a histogram of the elements in the stream

- Point queries
- Range queries
We’d like to get a histogram of the elements in the stream

- Point queries
- Range queries
- “Heavy hitters”
Multisets – stream summary

We’d like to get a histogram of the elements in the stream

- Point queries
- Range queries
- “Heavy hitters”
- Quantile queries
Count-min sketch

- Split each of $k$ hash functions of bloom filter into separate array of size $m$
- Use counters
- We gain the ability to delete
Questions?