CS630 Graduate Algorithms

November 5, 2024

by Dora Erdos and Jeffrey Considine

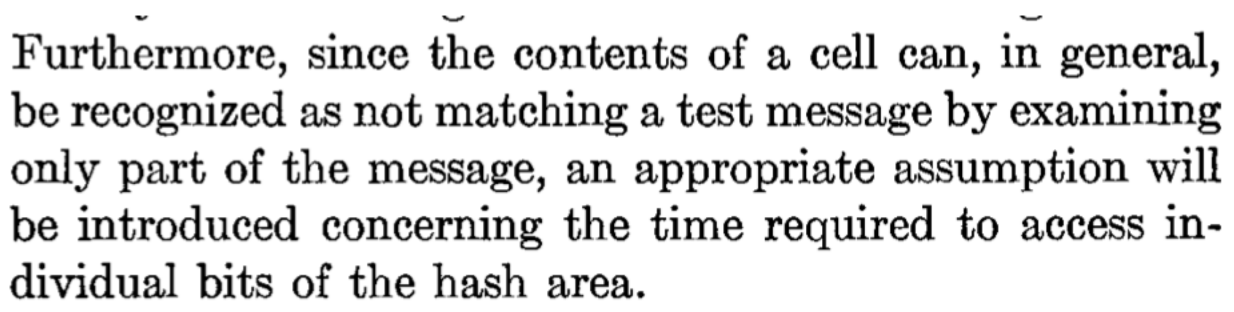
- Bloom Filters

slides by Dora Erdos and Jeffrey Considine, based on Kevin Wayne, Adam Smith

Hyphenation

A long time ago, in a galaxy with fixed width fonts…

- Hyphenation can break up a word into pieces, separated by a hyphen.



Furthermore, since the contents of a cell can, in general, be recognized as not matching a test message by examining only part of the message, an appropriate assumption will be introduced concerning the time required to access in-dividual bits of the hash area.

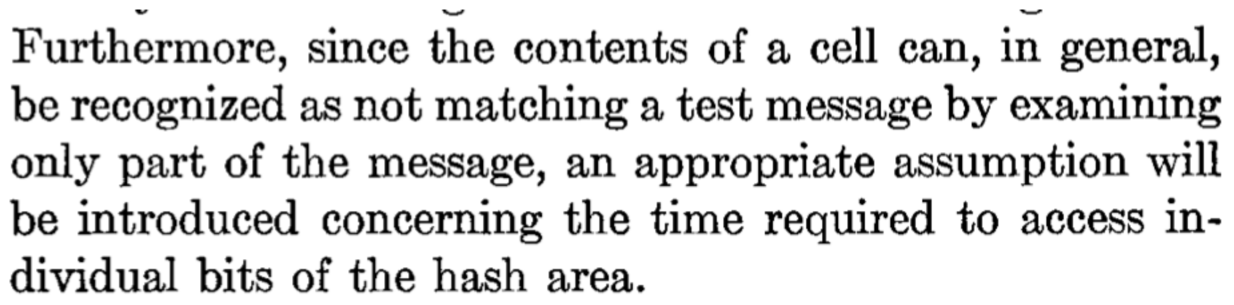
This is hyphenation.

Image source: “Space/Time Trade-offs in Hash Coding with Allowable Errors” by Bloom (1970)

Hyphenation

A long time ago, in a galaxy with fixed width fonts…

- Hyphenation can break up a word into pieces, separated by a hyphen.


Furthermore, since the contents of a cell can, in general, be recognized as not matching a test message by examining only part of the message, an appropriate assumption will be introduced concerning the time required to access in-
dividual bits of the hash area.

This is hyphenation.

- This used to be done a lot more often before modern word processors managed the line wrapping automatically…
- The trailing hyphen at the end of last word of a line signals that the word need-
ed to be split because it was too long…
  - Most hyphenation can be done automatically with some very simple rules.
  - But there are a few exceptions.
  - Usual approach was to have a small database of those exceptions.
  - But accessing that database was slow compared to the rest of the program.

Image source: “Space/Time Trade-offs in Hash Coding with Allowable Errors” by Bloom (1970)

Approximate Membership Queries

Bloom’s proposal:

- Make a small sketch of the exception database with the following properties.
    - This sketch must fit in memory.
    - If a word is in the exception database, always return true.
    - If a word is not in the exception database, usually return false.

Approximate Membership Queries

Bloom's proposal:

- Make a small sketch of the exception database with the following properties.
  - This sketch must fit in memory.
  - If a word is in the exception database, always return true.
  - If a word is not in the exception database, usually return false.

- This sketch provides approximate membership queries with one-sided error.
  - If the sketch returns false, the word is definitely not in the database.
  - If the sketch returns true, the word might be in the database.

Approximate Membership Queries

Bloom's proposal:

- Make a small sketch of the exception database with the following properties.
  - This sketch must fit in memory.
  - If a word is in the exception database, always return true.
  - If a word is not in the exception database, usually return false.

- This sketch provides approximate membership queries with one-sided error.
  - If the sketch returns false, the word is definitely not in the database.
  - If the sketch returns true, the word might be in the database.

- If the sketch returns false, then the expensive database check is skipped.
- If the sketch returns true, then the expensive database check is performed.
  - Want to reduce "false positives" where the sketch returns true but the word is not in the database.

Approximate Membership?

How could this work?

Approximate Membership?

How could this work?

Hint: paper title was “Space/Time Trade-offs in Hash Coding with Allowable Errors”.

Approximate Membership?

How could this work?

Hint: paper title was “Space/Time Trade-offs in Hash Coding with Allowable Errors”.

Is hash function output longer or shorter than a word?

- We can make it shorter by truncate it, but then we get collisions.
- Maybe make a set of hash function outputs?
- Maybe make a set of truncated hash function outputs?

Approximate Membership?

How could this work?

Hint: paper title was “Space/Time Trade-offs in Hash Coding with Allowable Errors”.

Is hash function output longer or shorter than a word?

- We can make it shorter by truncate it, but then we get collisions.
- Maybe make a set of hash function outputs?
- Maybe make a set of truncated hash function outputs?

Taking a step back, how do we represent a set?

- A sorted tree?
- A hash table?

Bloom’s Idea

Imagine a hash table where each bucket is represented by one bit.

- This bit is 1 if anything is in the bucket, and 0 otherwise.

Bloom's Idea

Imagine a hash table where each bucket is represented by one bit.

- This bit is 1 if anything is in the bucket, and 0 otherwise.

What can we do with this?

- If we query an item and the bit is 0,
  - Then it definitely was not inserted.
- If we query an item and the bit is 1,
  - Then something was inserted with the same hash code.
  - But it might have been a different item than we queried.

Bloom Filters (incomplete)

Initialize:

- Make an array B of m bits initialized to 0.

Insert(x):

- Set $B[h(x)] = 1$

Check(x):

- If $B[h(x)] = 0$, return false.
- Return true.

TopHat

Which of the following are true of the Bloom filter construction so far assuming the array has m bits and n items were inserted?

1. Insertions take constant time.
2. If Check(x) returns true, then Insert(x) was previously called.
3. If Check(x) returns false, then Insert(x) was never called.
4. The probability of a false positive where Check(x) returns true but Insert(x) was never called is $(1 - 1/m)^n$.

TopHat Review

1. Insertions take constant time.

2. If Check(x) returns true, then Insert(x) was previously called.

3. If Check(x) returns false, then Insert(x) was never called.

4. The probability of a false positive where Check(x) returns true but Insert(x) was never called is $(1 - 1/m)^n$.

Bloom Filter Optimization

If we will insert n items, how big should m be?

Bloom Filter Optimization

If we will insert n items, how big should m be?

- The probability of a false positive is $1 - (1 - 1/m)^n \approx 1 - (1/e)^{n/m}$.
- So grows bigger with n and smaller with m.
- Unhelpful answer: m should be as big as possible.

Bloom Filter Design

Actually one more parameter to consider - how many hash functions?

Bloom Filter Design

Actually one more parameter to consider - how many hash functions?

What if we use k hash functions instead of just one?

- Multiple checks to reduce false positives?

Initialize:

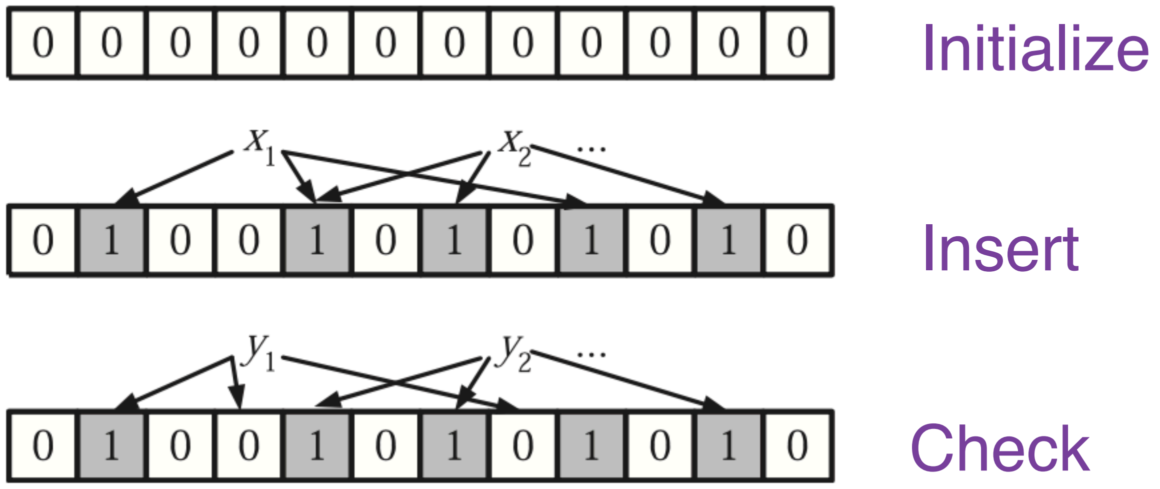
- Make an array B of m bits initialized to 0.

Insert(x):

- For i = 1 to k,
  - Set $B[h_i(x)] = 1$

Check(x):

- For i = 1 to k,
  - If $B[h_i(x)] = 0$, return false.
- Return true.



Are false positives or negatives possible with these changes?

Image source: Network Applications of Bloom Filters: A Survey (2005)

Bloom Filter Analysis

Given a Bloom filter with m bits, k hash functions, and n items inserted, what is the false positive probability?

Let $z[i, j]$ be the event that bit $B[i]$ is still zero after j items are inserted.

The probability of $z[i, j]$ is $(1 - 1/m)^{kj}$.

For distinct i and i',

- $z[i, j]$ and $z[i', j]$ are negatively correlated.

The expected number of bits that are still zero after j items are inserted is $n(1 - 1/m)^{kj}$.

- Tight distribution around this mean because of negative correlations.

Bloom Filter Analysis

What is the probability of a false positive after n insertions?

Bloom Filter Analysis

What is the probability of a false positive after n insertions?

- Simplification:
  - Assume exactly $n(1 - 1/m)^{kn}$ zeros in the array.
  - This greatly simplifies the independence handling below.

Bloom Filter Analysis

What is the probability of a false positive after n insertions?

- Simplification:
  - Assume exactly $n(1 - 1/m)^{kn}$ zeros in the array.
  - This greatly simplifies the independence handling below.
- Use $p = (1 - 1/m)^{kn}$ as the probability of any $z[i, n]$.
- For each hash function check,
  - The probability $B[h_i(x)] = 0$ and the function returns false is p.
  - The probability of continuing is $1 - p$.

Bloom Filter Analysis

What is the probability of a false positive after n insertions?

- Simplification:
  - Assume exactly $n(1 - 1/m)^{kn}$ zeros in the array.
  - This greatly simplifies the independence handling below.
- Use $p = (1 - 1/m)^{kn}$ as the probability of any $z[i, n]$.
- For each hash function check,
  - The probability $B[h_i(x)] = 0$ and the function returns false is p.
  - The probability of continuing is $1 - p$.
- The probability of continuing for all k hash functions is $(1 - p)^k$.
  - This is where the assumption of the number of zeros kicks in.
  - This also is implicitly an assumption on the fraction of ones.
  - Assuming the fraction of ones lets us ignore dependences around whether particular bucket bits are 1 or 0.
  - Just need hash function to (pseudorandomly) pick a one each time.

Bloom Filter Analysis

What is the probability of a false positive after n insertions?

- Simplification:
  - Assume exactly $n(1-1/m)^{kn}$ zeros in the array.
  - This greatly simplifies the independence handling below.
- Use $p = (1-1/m)^{kn}$ as the probability of any $z[i,n]$.
- For each hash function check,
  - The probability $B[h_i(x)] = 0$ and the function returns false is p.
  - The probability of continuing is $1-p$.
- The probability of continuing for all k hash functions is $(1-p)^k$.
  - This is where the assumption of the number of zeros kicks in.
  - This also is implicitly an assumption on the fraction of ones.
  - Assuming the fraction of ones lets us ignore dependences around whether particular bucket bits are 1 or 0.
  - Just need hash function to (pseudorandomly) pick a one each time.
- So false positive probability f for x not inserted is $(1-p)^k$.

Bloom Filter Analysis

What is the probability of a false positive after n insertions?

- Simplification:
  - Assume exactly $n(1 - 1/m)^{kn}$ zeros in the array.
  - This greatly simplifies the independence handling below.
- Use $p = (1 - 1/m)^{kn}$.
- So false positive probability f for x not inserted is $(1 - p)^k$.

So,

- $p = (1 - 1/m)^{kn} \approx (1/e)^{kn/m} = e^{-kn/m}$
- $f \approx \left(1 - (1/e)^{kn/m}\right)^k = \left(1 - e^{-kn/m}\right)^k$

Simplification:

- Will treat that as an equality going forward.

What k minimizes the false positive probability f ?

Bloom Filter Analysis

So,

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

What k minimizes the false positive probability f ?

Bloom Filter Analysis

So,

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

What k minimizes the false positive probability f ?

Minimizing f is the same as minimizing $g = \ln f$…

- $g = k \ln \left(1 - e^{-kn/m}\right)$

Bloom Filter Analysis

So,

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

What k minimizes the false positive probability f ?

Minimizing f is the same as minimizing $g = \ln f$...

- $g = k \ln\left(1 - e^{-kn/m}\right)$

Taking the derivative,

- $\dfrac{\partial g}{\partial k} = \ln\left(1 - e^{-kn/m}\right) + \dfrac{kn}{m}\dfrac{e^{-kn/m}}{1 - e^{-kn/m}}$

Bloom Filter Analysis

So,

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

What k minimizes the false positive probability f?

Minimizing f is the same as minimizing $g = \ln f$...

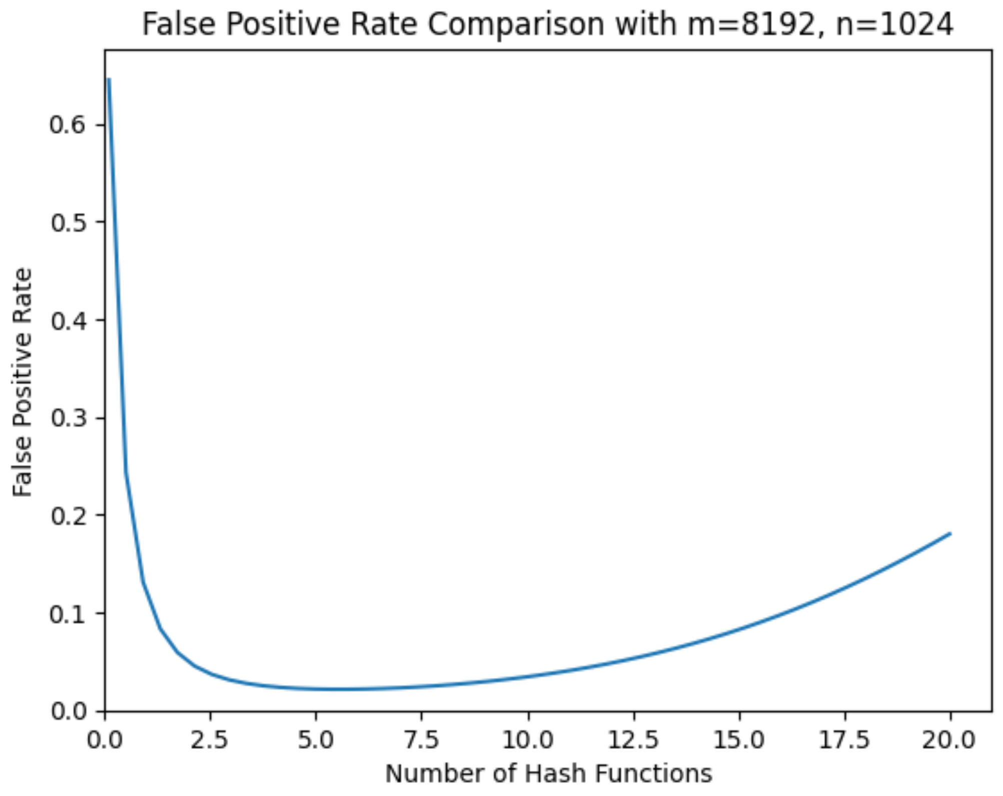
- $g = k \ln\left(1 - e^{-kn/m}\right)$

Taking the derivative,

- $$\frac{\partial g}{\partial k} = \ln\left(1 - e^{-kn/m}\right) + \frac{kn}{m}\frac{e^{-kn/m}}{1 - e^{-kn/m}}$$

This is zero when $k = \ln 2 \cdot (m/n)$.

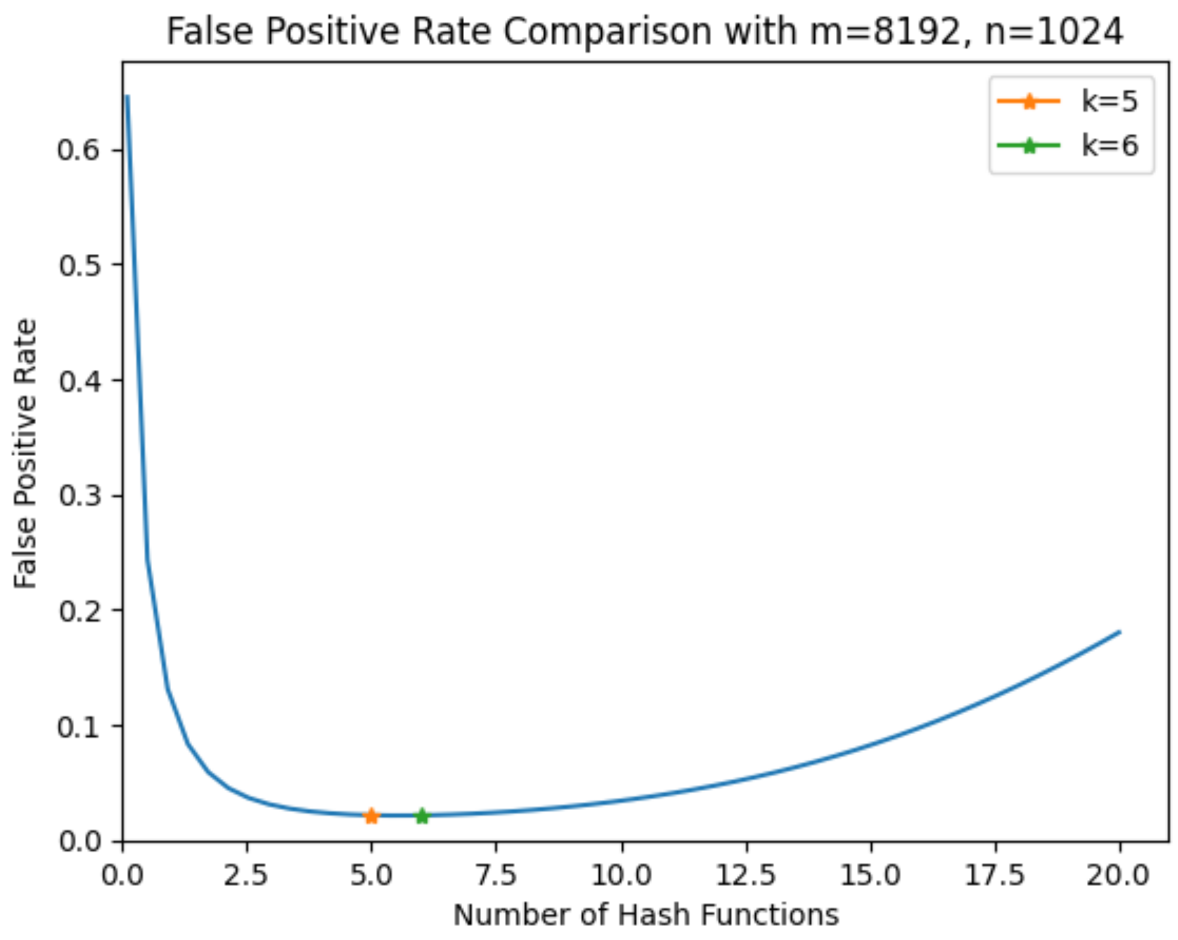
$$\ln\left(1 - e^{-\ln 2}\right) + \ln 2 \cdot \frac{e^{-\ln 2}}{1 - e^{-\ln 2}} = \ln(1 - 1/2) + \ln 2 \cdot \frac{1/2}{1 - 1/2} = \ln 1/2 + \ln 2$$

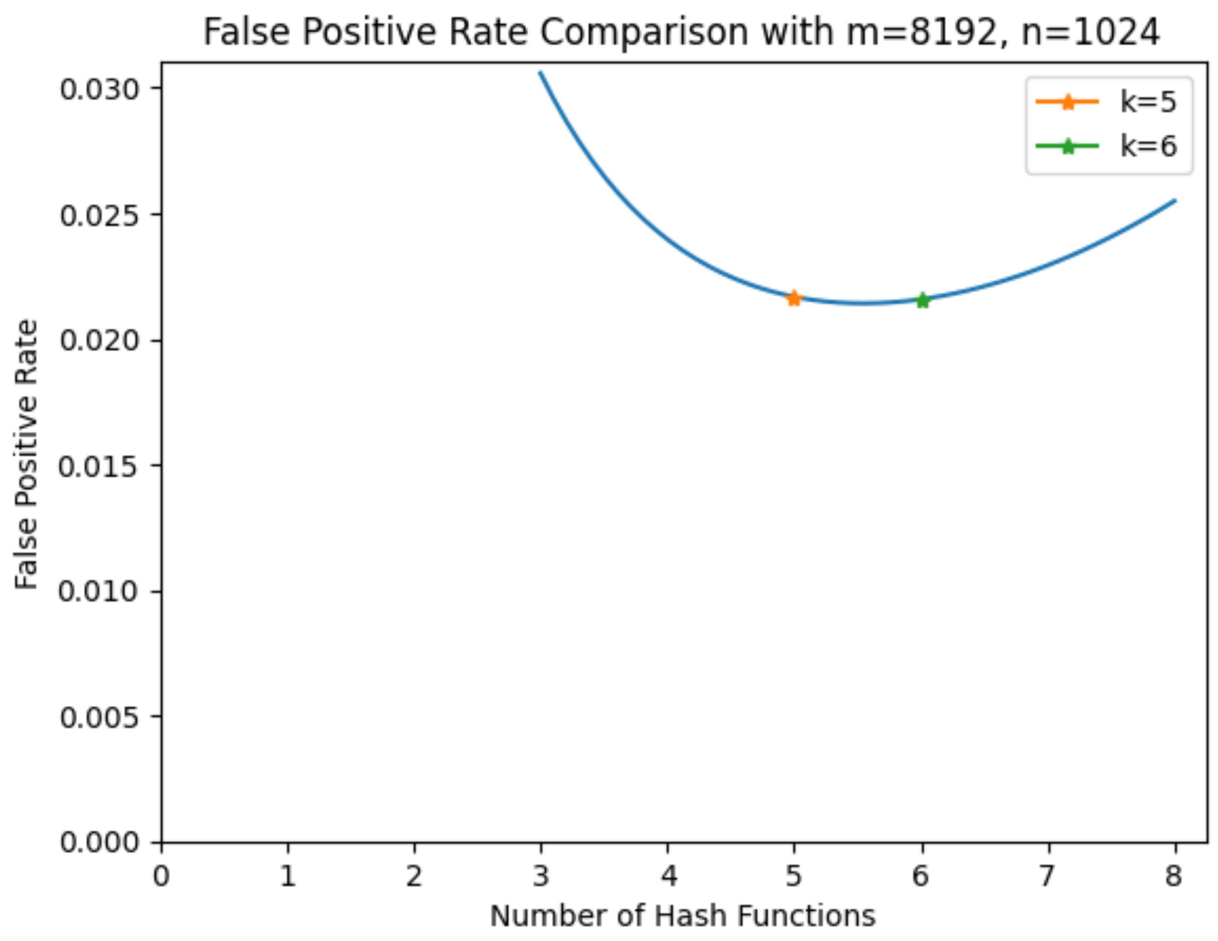


Bloom Filter Analysis

Given a Bloom filter with m bits and n items inserted, the false positive probability is minimized using $k = \ln 2 \cdot (m/n)$ hash functions.

- Well, if k is an integer.
- Optimal choice is one of $\lfloor \ln 2 \cdot (m/n) \rfloor$ or $\lceil \ln 2 \cdot (m/n) \rceil$
- Common to round down to reduce hash computations even if slightly suboptimal false positive rates.





Bloom Filter Analysis

Given a Bloom filter with m bits and n items inserted, the false positive probability is minimized using $k = \ln 2 \cdot (m/n)$ hash functions.

- Well, if k is an integer.
- Optimal choice is one of $\lfloor \ln 2 \cdot (m/n) \rfloor$ or $\lceil \ln 2 \cdot (m/n) \rceil$
- Common to round down to reduce hash computations even if slightly suboptimal false positive rates.

Previously saw $p = (1 - 1/m)^{kn} \approx (1/e)^{kn/m} = e^{-kn/m}$…

So with $k = \ln 2 \cdot (m/n)$, then

- $p = 1/2$
- $f = 1/2^k \approx 0.62^{m/n}$

Bloom Filter Analysis

Given a Bloom filter with m bits and n items inserted, the false positive probability is minimized using $k = \ln 2 \cdot (m/n)$ hash functions.

- Well, if k is an integer.
- Optimal choice is one of $\lfloor \ln 2 \cdot (m/n) \rfloor$ or $\lceil \ln 2 \cdot (m/n) \rceil$
- Common to round down to reduce hash computations even if slightly suboptimal false positive rates.

Previously saw $p = (1 - 1/m)^{kn} \approx (1/e)^{kn/m} = e^{-kn/m}$…

So with $k = \ln 2 \cdot (m/n)$, then

- $p = 1/2$
- $f = 1/2^k \approx 0.62^{m/n} \longleftarrow$ False positive rate drops exponentially with bits per item.

$\uparrow$

Nice round numbers.

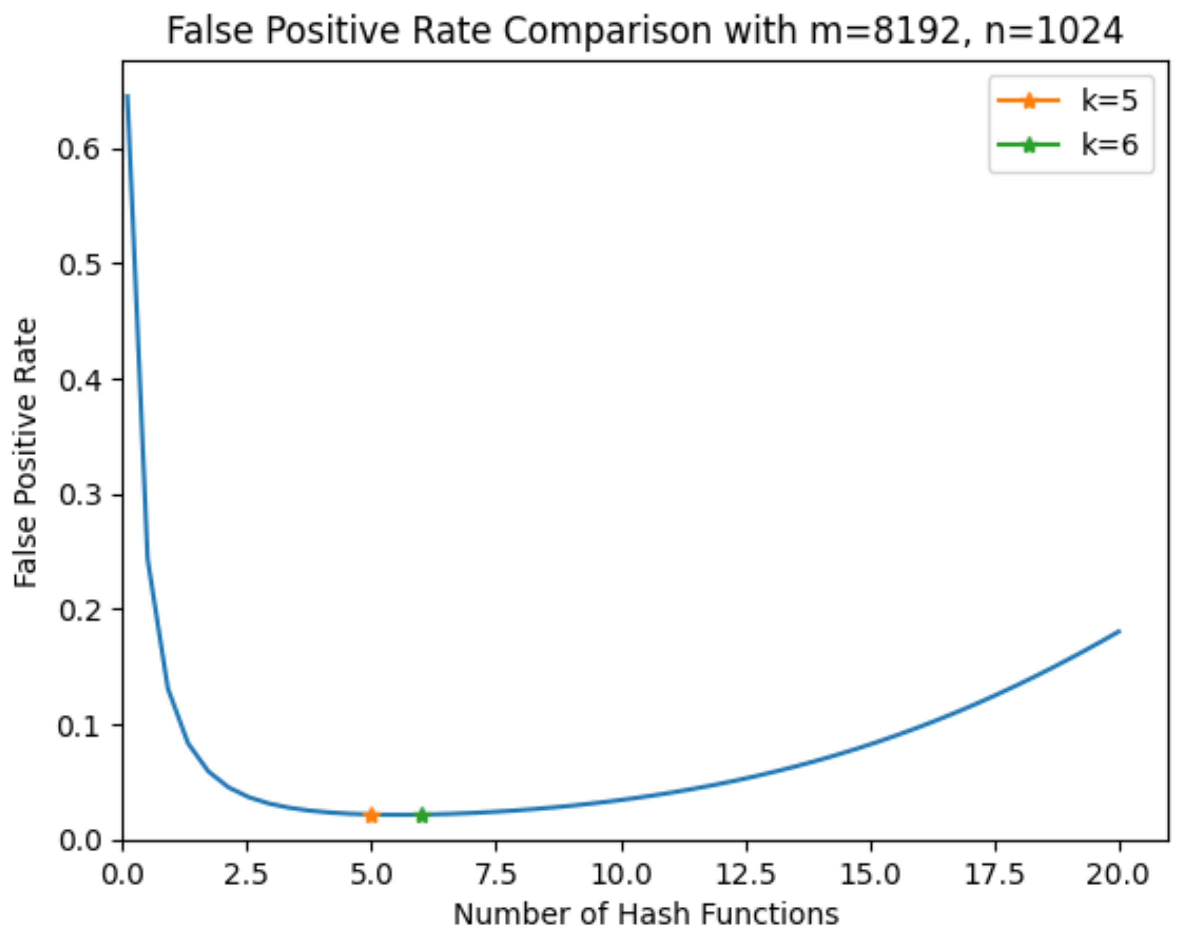
Still get 1/2 if you do the math more carefully.

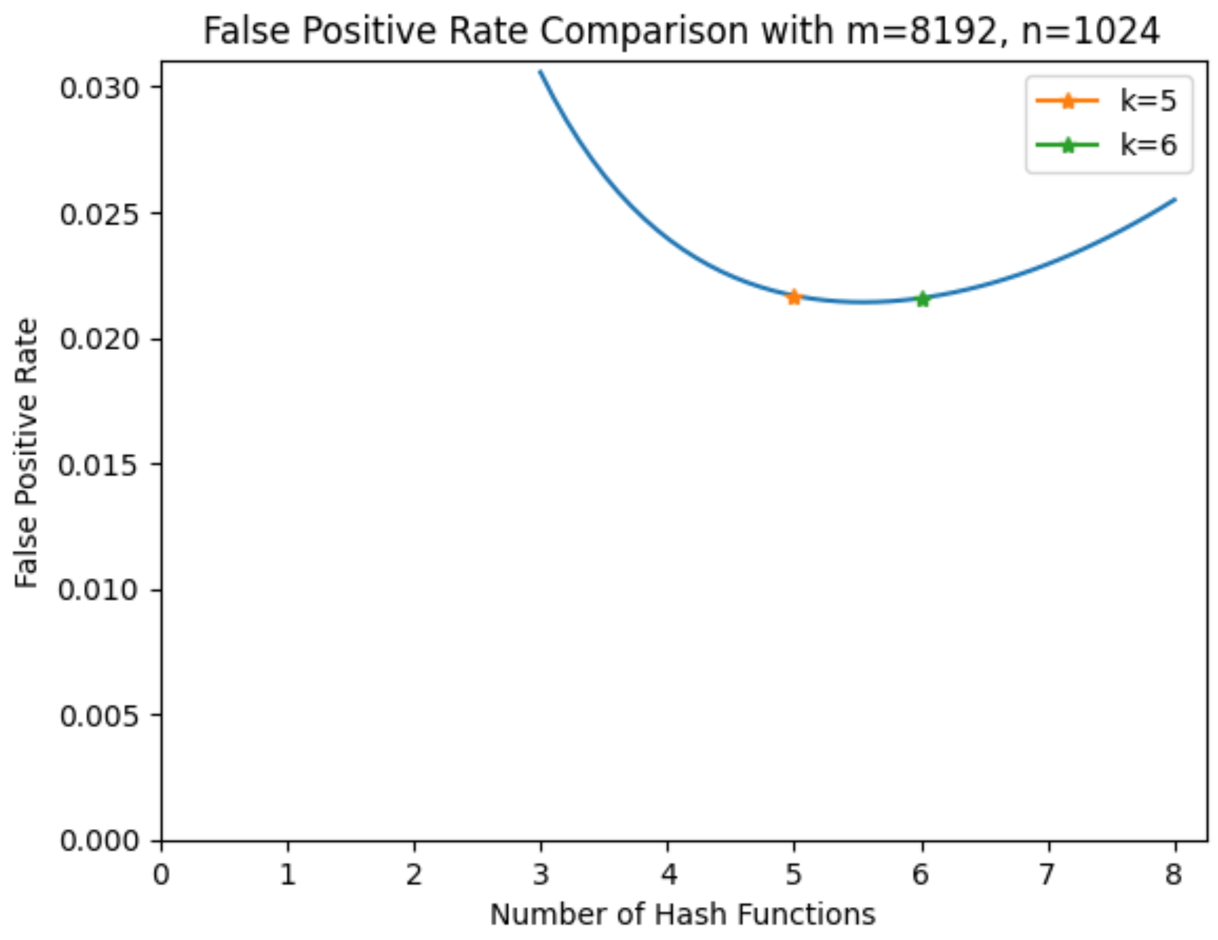
Bloom Filters in Practice

If no particular space target, standard recommendation is

- $m = 8n$
- $k = \lfloor \ln 2 \cdot 8 \rfloor = 5$
- $p = (1 - 1/m)^{kn} \approx e^{-kn/m} = e^{-5/8} \approx 0.535$
- $f = (1 - p)^k \approx 0.022$

So one byte per item and ~2% false positive rate.





Bloom Filter Application - Reduce Expensive Database Queries

Generic version:

- Store database keys in a Bloom filter.
- Check Bloom filter to see if database key is present.
- Query database if Bloom filter returns true. ← Fast
- False positives result in extra database queries. ← Slow

From my industry experience:

- Running a real-time bidder for ad inventory.
  - Very low ms response times required.
- Bid requests came with cookie data that we could lookup.
  - Database too slow to respond to every bid request.
  - Some cookies not in the database.
  - Some cookies not associated with valuable data.
  - Wanted to focus bidding on a small valuable subset of cookies.
- Bloom filter for valuable cookie ids
  - Many fewer database requests even with false positives.
  - Reduced requests → reduced queuing → faster response times

Bloom Filter Application - Peer-to-Peer Sharing

Circa 2000, lots of interest in peer-to-peer (p2p) applications.

- Approximately all actual p2p applications were file sharing.
- Mostly music.
- Mostly illegal.
- Mostly trying not to get sued for copyright infringement.

Bloom filters often included in designs for two reasons.

- Smaller representation of what you could share
- Somewhat deniable what you were actually offering
- BitTorrent later had a proposal to integrate Bloom filters, but neither approved or implemented…

Bloom Filter Variant - Minimum Path Lengths

Also used for peer-to-peer design -

- Replace each bit with a counter
  - How many steps to the nearest item hashing to this location?
- How does it work?
  - Counters initialize to a MAX or ∞ value.
  - Peers insert their items by setting counters selected by hashes to zero.
  - These Bloom filters are all 0 or ∞ before sharing.
  - Then share to neighbors.
  - Neighbors add one to all received counters.
  - Then combine with own counters taking the element-wise minimum.
  - Check an item’s distance by taking the max of all counters selected by hash functions.
- If you have taken a networking class,
  - This is distance vector routing with the hashes obfuscating addresses.

Bloom Filter Variant - Counting Bloom Filter

A Bloom filter weakness - no deletion support.

Counting Bloom filter

- Instead of individual bits, maintain (small) counters for each entry.
- Inserts increment all counters selected by hashes.
- Deletes decrement all counters selected by hashes.

- Do not delete items that were not inserted!

- Space utilization is worse.
- But can convert to normal Bloom filter if it needs to be transmitted.

Bloom Filter Variant - Compressed Bloom Filters

Bloom filters usually optimized by picking $k = \ln 2 \cdot (m/n)$ hash functions.

What if you compress the resulting Bloom filter?

- Access time will be worse.
- But maybe better false positive rates for the same space usage?

If intending to compress,

- Use $k = 1$ hash functions.
- Uncompressed Bloom filter will have higher density of zeros.
- Easy to compress.
- False positive rate ~ $0.50^{\text{(bits/item)}}$ after compression
  vs previous $0.62^{\text{(bits/item)}}$ without compression

- Deferring this analysis until we talk about compression later.

Bloom Filter Variant - Combine with Power of Two Choices?

This actually works, but let’s stop here.

- We consider the combination of two ideas from the hashing literature: the power of two choices and Bloom filters. Specifically, we show via simulations that, in comparison with a standard Bloom filter, using the power of two choices can yield modest reductions in the false positive probability using the same amount of space and more hashing. While the improvements are sufficiently small that they may not be useful in most practical situations, the combination of ideas is instructive; in particular, it suggests that there may be ways to obtain improved results for Bloom filters while using the same basic approach they employ, as opposed to designing new, more complex data structures for the problem.

Using the Power of Two Choices to Improve Bloom Filters (2007)