Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 22.

Randomization and Hashing

M. Oguzhan Kulekci

- Bloom Filters
- Minwise Hashing

Hash Functions and Randomized Algorithms ?

- A hash function should be completely deterministic. No randomization!
- Remember in randomization, we repeated the random process a couple of times to reduce the false alarms.
- Can using multiple *randomly chosen hash functions* make sense to reduce the verification complexity in hashing ?
- Bloom filter is a good example of such multiple randomly chosen hash functions.

Bloom Filters

- On a document collection free of duplicates, when a new document will be added, check whether a copy of it was previously added.
- Using a hash table means computing the hash of the new item and check it in the hash table.
- In case of a match, perform **verification** since collisions are unavoidable

How can we reduce the need for verification ?

- By using more than one hash function
  - That will increase the space usage
- **Bloom filters** provide a solution
  - Multiple hash functions with a **bitmap** structure

Bloom Filters

Bloom filter :

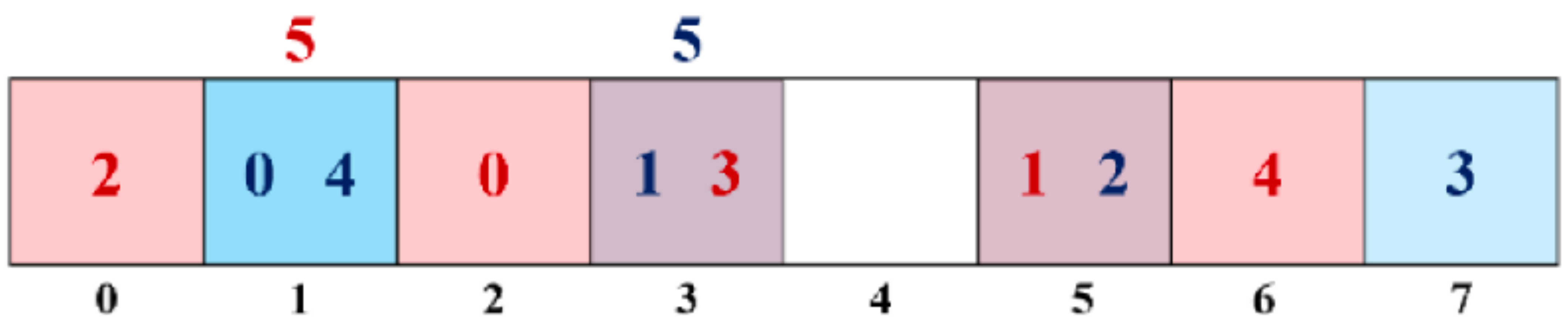
- 8-bits long and two hash functions.
- Insert 0,1,2,3,4 in order

x=0	x=1	x=2	x=3	x=4
1	3	5	7	1
2	5	0	3	6

$$h_1(x) = 2x + 1 \mod 8$$
$$h_2(x) = 3x + 2 \mod 8$$

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0
0	1	1	**1**	0	**1**	0	0
1	1	1	1	0	**1**	0	0
1	1	1	**1**	0	1	0	**1**
1	**1**	1	1	0	1	**1**	1

- How to search for an item?
- Compute the hash bit positions and check whether all are set.
- If so, a match is reported.
- Notice that still it can be a false alarm.



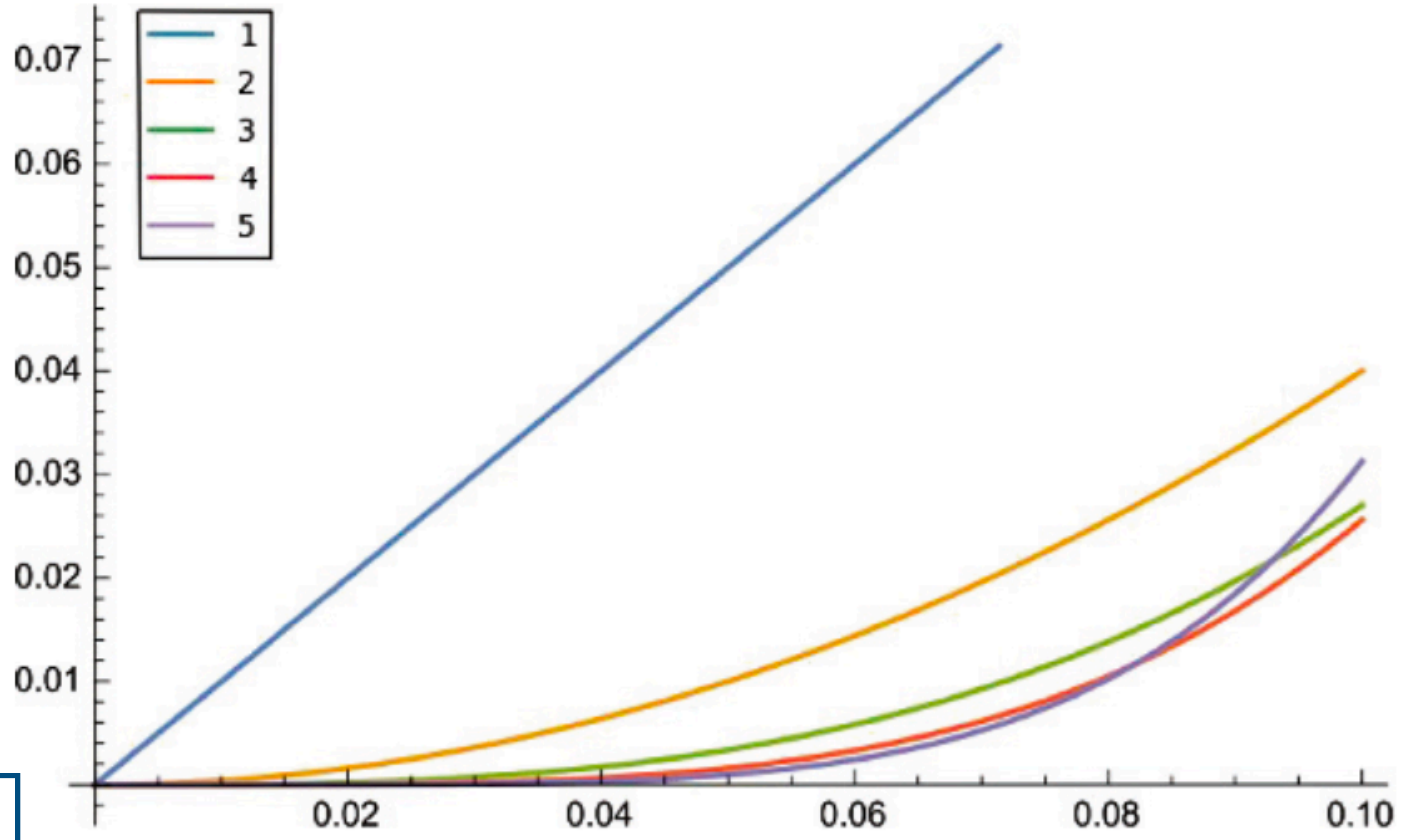
What to do to reduce that possibility ?

Bloom Filters

- Assume we have k functions and m items are inserted into a n bits hash table.
- Each insertion marks k bit positions and thus at most km bits are set among n bits.
- The probability that a bit is set is then km/n.
- The probability that an item which has not been inserted into hash table but have all corresponding bits set is $(km/n)^k$.
- This value gets small with increased k.
- With further derivations, it arrives

For a target false alarm probability ϵ,
$k = -\log_2 \epsilon$ and
bits per element (inverse load) is $-1.44 \log_2 \epsilon$

k



load factor $= \dfrac{m}{n}$

Minwise Hash

How can we check the similarity of two documents? $J(D_1, D_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|}$

Sort the words in D_2, and then check every word of D_1 on the sorted list $O(n \log n)$

Hash may help to achieve $O(n)$:

- Create a hash table for all the words in D_2, and search the words of D_1 on that hash

How about finding the similarity between many documents and a queried document ?

Minwise Hash

- Assume we select **one** representative word from each document, and keep an index of those selected words.
- If the representative words of the two queried documents are the same, we say they are similar.
- How would you select the representative word?
- Most frequent, least frequent, **random**, TF-IDF…?
- Think about maintaining k selected words per document.

Compute the hash of each word in a document, and just select the words that has the minimum hash values.

s:	The	cat	in	the	hat
h(s):	17	128	56	17	4

The	hat	in	the	store
17	4	56	17	96

Minwise Hash

- Since we use the same hash function, minwise hashing provides us **selecting the same random words**. Why?
  - Assume the vocabularies of two documents are same and we are selecting a random word from each document.
    - If we do it regularly, the probability of selecting same is $1/|V|$.
    - With min hash values, it is guaranteed to select a randomly picked word as they both will have the same hash value.
  - Assume the vocabularies are not the same.
    - Then the probability that min hash selects the same word depends on the number of common words in between them.
    - Larger the intersection, greater the probability, which reflects the measurement we need, the Jaccard similarity

Minwise Hash

- Notice we again need to pass over all documents and compute the hash values
- So, it seems **not very advantageous** in terms of **time** over the classical approach
- However, there is a very significant **space saving**
  - Assume n documents have m words on the average, thus we will need $O(nm)$ space, but with the minwise hash values, it requires $O(kn)$ space, where $k \ll m$

Compute the hash of each word in a document, and just select the words that has the minimum hash values.

s:	The	cat	in	the	hat
h(s):	17	128	56	17	4

The	hat	in	the	store
17	4	56	17	96

Minwise Hash - Another Example

- How can we estimate the number of distinct values in a streaming integer array S, which includes duplicates, using only a constant amount of memory?
- Regular approach requires linear space.
- With minwise hash, it turns to be constant space. How?

- **Compute the hash of the each value and store it if it is less than the current minimum value.**
- **At the end the estimated value of distinct numbers k is** $k = \dfrac{M}{minhash} - 1$, **where M is the hash size and $minhash$ is the minimum hash observed.**
- **Why ?**

Minwise Hash - Another Example

- Assume we have randomly selected k distinct items from $[1..M]$. Here k denotes the number of distinct elements of S, since their hashes will be distinct.
- What is the expected minimum of these k values?
    - If $k = 1$, it should be $M/2$
    - If $k = 2$, it should be $M/3$
    - In general we expect the minimum value to be $M/(k+1)$, since the interval will be split into (k+1) pieces, which are expectedly to be of equal length.
- Therefore, given the smallest element a_1, k can be estimated as $a_1 = \dfrac{M}{k+1}$ is the expected minimum value.
- *Accuracy can be improved by using more than one hash function ?*

Reading assignment

- Skiena chapter 8.
- Goodrich et al. 10.2