Introduction to Data Mining

Lecture #7: Mining Data Streams-2

U Kang
Seoul National University

Today’s Lecture

- **More algorithms for streams:**
  - **(1) Filtering a data stream: Bloom filters**
    - Select elements with property **x** from stream
  - **(2) Counting distinct elements: Flajolet-Martin**
    - Number of distinct elements in the last ***k*** elements of the stream

Outline

- ☐ **Filtering Data Stream**
- ☐ Counting Distinct Elements

Motivating Applications

- **Example: Email spam filtering**
  - We know 1 billion “good” email addresses
  - If an email comes from one of these, it is **NOT** spam
- **Publish-subscribe systems**
  - You are collecting lots of messages (news articles)
  - People express interest in certain sets of keywords
  - Determine whether each message matches user’s interest

Filtering Data Streams

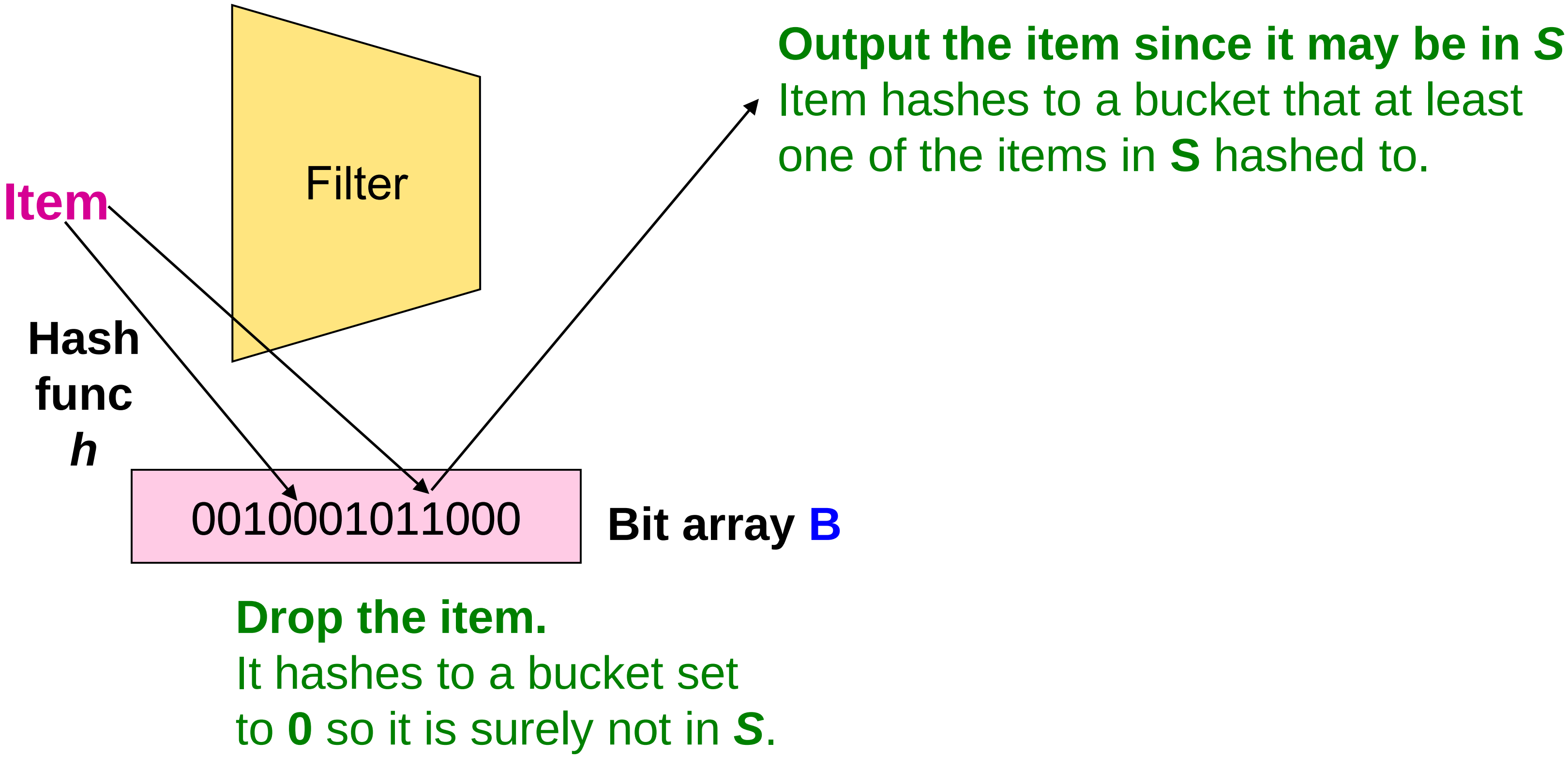
- **Each element of data stream is a tuple**
- Given a list of keys **S**
- **Determine which tuples of stream are in *S***

- **Obvious solution: Hash table**
  - But suppose we **do not have enough memory** to store all of ***S*** in a hash table
    - E.g., we might be processing millions of filters on the same stream

First Cut Solution (1)

- **Given a set of keys *S* that we want to filter**
- Create a **bit array *B*** of ***n*** bits, initially all ***0*s**
- Choose a **hash function *h*** with range **[*0,n*)**
- Hash each member of $s \in S$ to one of ***n*** buckets, and set that bit to **1**, i.e., ***B[h(s)]=1***
- Hash each element ***a*** of the stream and output only those that hash to bit that was set to **1**
  - **Output *a* if B[h(a)] == 1**

First Cut Solution (2)



- **Creates false positives but no false negatives**
  - If the item is in ***S*** we surely output it, if not we may still output it

First Cut Solution (3)

- **|S| = 1 billion email addresses**
  |B|= 1GB = 8 billion bits

- If the email address is in ***S***, then it surely hashes to a bucket that has the bit set to **1**,
  so it always gets through (***no false negatives***)

- Approximately **1/8** of the bits are set to **1**, so about **1/8th** of the addresses not in ***S*** get through to the output (***false positives***)
  - Actually, less than **1/8th**, because more than one address might hash to the same bit

Analysis: Throwing Darts (1)

- **More accurate analysis for the number of false positives**

- **Consider:** If we throw ***m*** darts into ***n*** equally likely targets, **what is the probability that a target gets at least one dart?**

- **In our case:**
  - **Targets** = bits/buckets
  - **Darts** = hash values of items

Analysis: Throwing Darts (2)

- We have ***m*** darts, ***n*** targets
- **What is the probability that a target gets at least one dart?**

Equals **1/e** as **n** →∞

Equivalent

$$1 - (1 - 1/n)^{n(m/n)}$$

$$\mathbf{1 - e^{-m/n}}$$

Probability some target **X** not hit by a dart

Probability at least one dart hits target **X**

Analysis: Throwing Darts (3)

- **Fraction of 1s in the array B =**
  = probability of false positive = $1 - e^{-m/n}$

- **Example:** **10^9** darts, **$8 \cdot 10^9$** targets
  - Fraction of **1s** in **B = $1 - e^{-1/8}$ = 0.1175**
    - Compare with our earlier estimate: **1/8 = 0.125**

Bloom Filter

- Consider: **|S| = *m*, |B| = *n***
- Use ***k*** independent hash functions $h_1, \ldots, h_k$
- **Initialization:**
  - Set **B** to all **0s**
  - Hash each element $s \in S$ using each hash function h_i, set **B[$h_i(s)$] = 1** (for each $i = 1, .., k$)

(**note:** we have a single array B!)

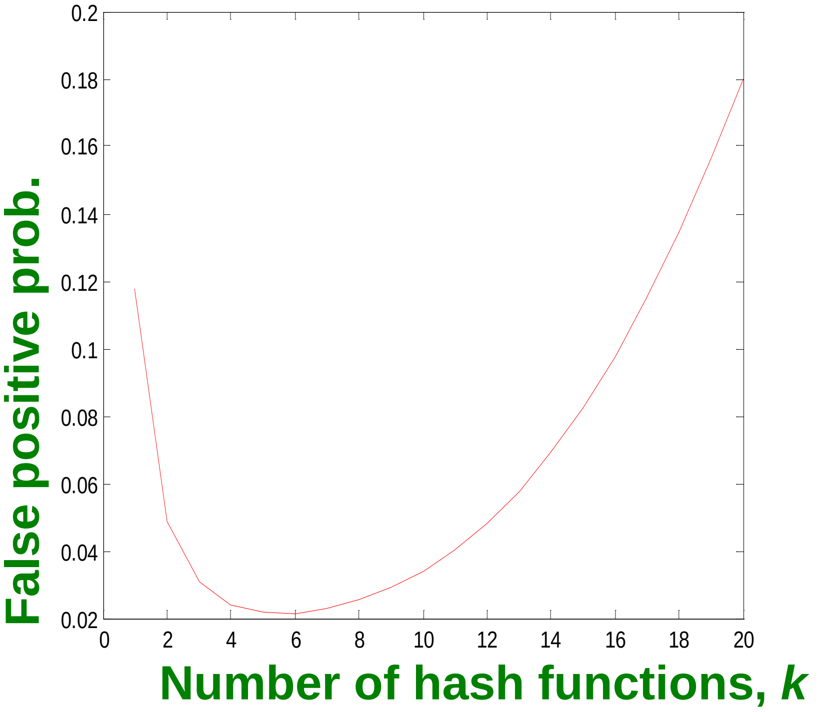
- **Run-time:**
  - When a stream element with key ***x*** arrives
    - If **B[$h_i(x)$] = 1** <u>for all</u> $i = 1, \ldots, k$ then declare that ***x*** is in ***S***
      - That is, ***x*** hashes to a bucket set to **1** for every hash function $h_i(x)$
    - Otherwise discard the element ***x***

Bloom Filter -- Analysis

- **What fraction of the bit vector B are 1s?**
  - Throwing $k \cdot m$ darts at n targets
  - So fraction of **1**s is $(1 - e^{-km/n})$
- But we have k independent hash functions and we only let the element x through **if all** k hash element x to a bucket of value **1**
- So, **false positive probability = $(1 - e^{-km/n})^k$**

Bloom Filter – Analysis (2)

- ***m* = 1 billion, *n* = 8 billion**
  - **k = 1**: $(1 - e^{-1/8})$ = **0.1175**
  - **k = 2**: $(1 - e^{-1/4})^2$ = **0.0493**
- **What happens as we keep increasing *k*?**



- "Optimal" value of ***k***: ***n/m* ln(2)**
  - **In our case:** Optimal **k = 8 ln(2) = 5.54 ≈ 6**
    - **Error at k = 6**: $(1 - e^{-1/6})^2$ = **0.0235**

Bloom Filter: Wrap-up

- **Bloom filters guarantee no false negatives, and use limited memory**
  - Great for pre-processing before more expensive checks
- **Suitable for hardware implementation**
  - Hash function computations can be parallelized

- Is it better to have **1** big **B** or ***k*** small **B**s?
  - **It is the same:** $(1 - e^{-km/n})^k$ vs. $(1 - e^{-m/(n/k)})^k$
  - But keeping **1 big B** is simpler

Outline

- ☑ Filtering Data Stream
- ☐ **Counting Distinct Elements**

Motivating Applications

- **How many different words are found among the Web pages being crawled at a site?**
  - Unusually low or high numbers could indicate artificial pages (spam?)
- **How many different Web pages does each customer request in a week?**
- **How many distinct products have we sold in the last week?**

Counting Distinct Elements

- **Problem:**
  - Data stream consists of a universe of elements chosen from a set of size ***N***
  - Maintain a count of the number of distinct elements seen so far

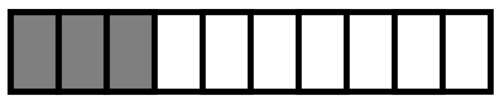
- **Obvious approach:**
  Maintain the set of elements seen so far
  - That is, keep a hash table of all the distinct elements seen so far

Using Small Storage

- **Real problem:** **What if we do not have space to maintain the set of elements seen so far?**
- **Estimate the count in an unbiased way**
- **Accept that the count may have a little error, but limit the probability that the error is large**

Flajolet-Martin Approach

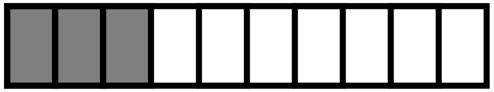
- Hash each item x to a bit, using exponential distribution
  - ½ map to bit 0, ¼ map to bit 1, …



- Let R be the position of the least '0' bit
- [Flajolet, Martin] : the number of distinct items is $2^R/\phi$, where ϕ is a constant

Intuition

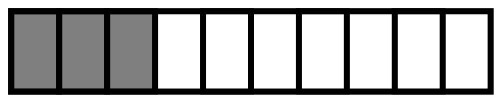
- Hash each item x to a bit, using exponential distribution: ½ map to bit 0, ¼ map to bit 1, …



- Intuition
  - The 0^{th} bit is accessed with prob. 1/2
  - The 1^{st} bit is accessed with prob. 1/4
  - … The k^{th} bit is accessed with prob. $O(1/2^k)$
- Thus, if the k^{th} bit is set, then we know that an event with prob. $O(1/2^k)$ happened
  - => We inserted distinct items $O(2^k)$ times

Improving Accuracy

- Hash each item x to a bit, using exponential distribution: ½ map to bit 0, ¼ map to bit 1, ...

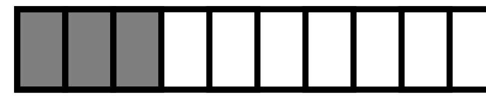


- Map each item to *k* different bitstrings, and we compute the average least '0' bit position b: # of items = $2^b/\phi$
  - => decrease the variance

- The final estimate: $2^b / (0.77351 * bias)$
  - *b : average least zero bit in the bitmask*
  - *bias : 1+.31/k for k different mappings*

Random Hash Function

- Hash each item x to a bit, using exponential distribution
  - ½ map to bit 0, ¼ map to bit 1, …



- How can we get this function?
  - Typically, a hash function maps an item to a random bucket
- Answer: use linear hash functions. Pick random (a_i, b_i) and then the hash function is:
  - $lhash_i(x) = a_i * x + b_i$
  - This gives uniform distribution over the bits
- To make this exponential, define
  - $hash_i(x)$ = least zero bit index in $lhash_i(x)$ (in binary format)

Storage Requirement

- Flajolet-Martin:
  - Let R be the position of the least '0' bit
  - The number of distinct items is $2^R/\phi$, where ϕ is a constant
- How much storage do we need?
  - R bits are required to count a set with $2^R/\phi = O(2^R)$ distinct items.
  - Thus, given a set with N distinct items, we need only O(log N) bits

Questions?