MPEI 2018 -2019

Aula 13

Set Membership problem(s)
&
Bloom Filters

Set membership problems

- Consider the following question:
- Given a string **s** (of arbitrary size) and a **set S**, does **s belong to S ?**
- For small sets it is easy
- **For the huge sets that occur in big data problems it is not as easy**

Applications

- In many situations and applications in the area of Informatics/ Computer Science we need to have an efficient way to determine if something belongs to a set

- Examples:
- Determine if candidate words are members of the set of words in a dictionary
- Determine if an email address is in a list of SPAM
- Check if one web page is cached in a caching proxy server

Application: Filtering Data Streams

- 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 ***S*** in a hash table
  - E.g., we might be processing millions of filters (keys) on the same stream

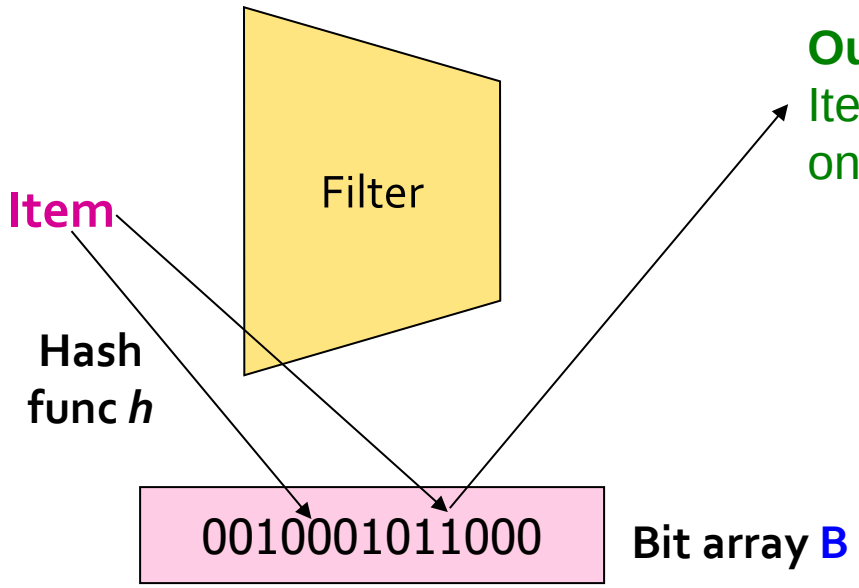
A partial solution

- **Given a set of keys *S* that we want to filter**
- Create a **bit array *B*** of ***n*** bits
  - initially all ***0s***
- 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**
  - ***B[h(s)]=1***

A partial solution (continuation)

- Hash each element ***a*** of the stream …
- and output only those that hash to a bit that was set to **1**
  - **Output *a* if B[h(a)] == 1**

A partial solution (continuation)



Output the item since it may be in *S*.
Item hashes to a bucket that has at least one of the items in **S** hashed to.

Drop the item.
It hashes to a bucket set to **0** so it is surely not in ***S***.

Limitations

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

Another Application: Email spam

- **Example:**
- We know 1 billion “good” email addresses
- If an email comes from one of these, it is **NOT** spam

First "Solution"

- **Apply the same idea as before**

- |S| = 1 billion email addresses
- n=|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

Bloom Filters

Bloom Filters

- The Bloom Filter is a way of using hashing to determine set membership
- They generalize the solution presented for SPAM and stream filtering

Definition

- Bloom filters use hash transforms to **compute a vector (the filter)** that is representative of the data set

- Membership is tested by comparing the results of hashing on the potential members to the vector

- In its simplest form the vector is composed of n elements, each a bit
- An element is set if and only if some hash transform hashes to that location for some key

Bloom Filter

- We will consider:
  - **S** as being the Set
  - **B** the Bloom Filter
  - **|S| = *m***
  - **|B| = *n***

- We will use ***k*** independent hash functions
  - $h_1, \ldots, h_k$

Bloom Filter: Initialization

- Set all **B** to **0s**
- Hash each element $s \in S$ using each hash function h_i,
  - And set $B[h_i(s)] = 1$ (for each $i = 1,.., k$)

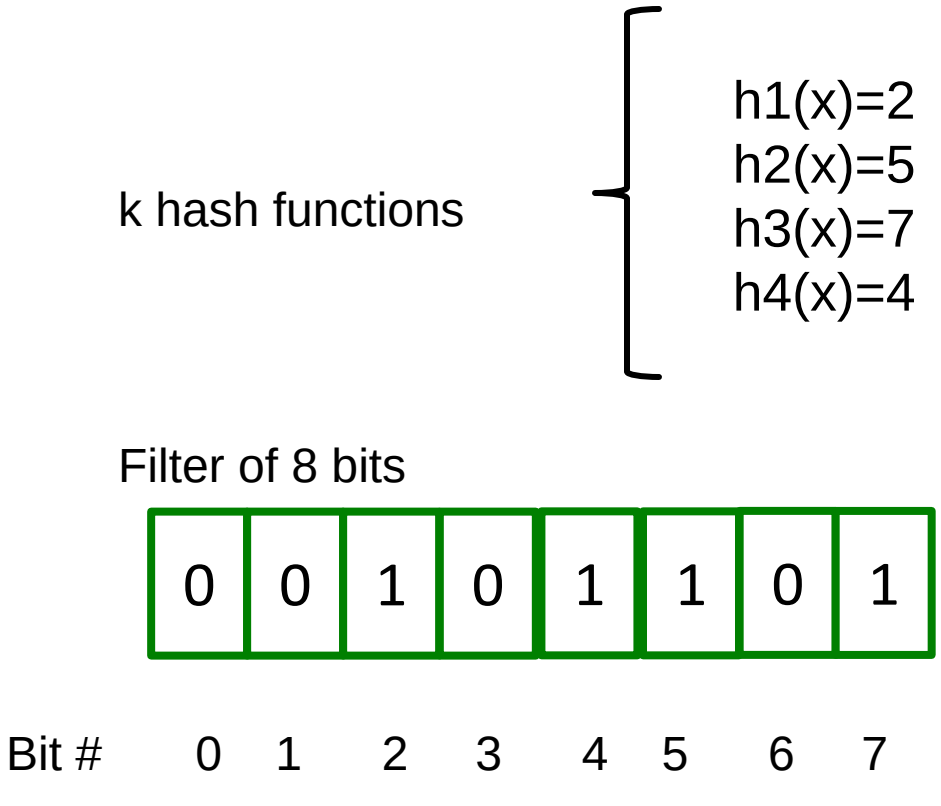
(**note:** we have a single array B!)

Bloom Filter: test membership

- At run-time, to test key ***x***:
- If **B[$h_i(x)$] = 1 for all *i* = 1,..., *k***
  - That is, ***x*** hashes to a bucket set to **1** for every hash function $h_i(x)$
- Then declare that ***x*** is probably in ***S***
- Otherwise discard the element ***x***

Example

- A filter with k = 4 hash functions and n= 8 bits



Errors

- Errors can occur when two or more transforms map to the same element.

- The membership test for a key **x** works by checking the elements that would have been updated if the key had been inserted into the vector
- If all the appropriate flag bits have been set by hashes then **x** will be reported as a member of the set
- If the elements have been updated by hashes on other keys - and not **x** - then the membership test will incorrectly report **x** as a member

Example of false positive error

A portion of the filter

···	P	C	P	P	C	—	C	···
···	—	T	—	T	T	—	—	···

Results of hashing *tomato*

Key	Meaning
P	updated by *potato*
C	updated by *cabbage*
T	would be set by *tomato*
—	unset

- Filter at left contains potato and cabbage but not tomato
- It will incorrectly identify tomato as a member
- Such an error occur because all the bits that would be set by tomato have already been set in the Filter

Superimposed coding

- Knuth [1] described Bloom Filters as a type of superimposed coding because all of the hash transforms map to the same table

- [1] Sorting and Searching, volume 3 of The Art of Computer Programming. Addison-Wesley Publishing Company, 1973.

Filter Parameters

- The behaviour of a Bloom Filter is determined by four parameters:
  - n :
    - The number of elements (or cells) in the Filter
  - k:
    - The number of hash transforms to be used
  - m:
    - The number of set members
  - f:
    - The fraction of elements (or cells) that are set in the Filter

Implementation

Hashing

- Hashing transforms are typically pseudo-random mathematical transforms used to compute addresses for lookup

Hashing

- The time complexity of searches by hashing can be as low as O(1) or as high as O(N), for a hash table with N elements

- The worst-case behaviour occurs when two or more distinct keys $K_i \neq K_j$ collide, i.e., $h(K_i) = h(K_j)$, and the entire table must be searched to find the correct entry

Hashing

- Bloom Filters are a fast method in which the hash transforms always have constant time complexity
- There is no attempt at collision resolution
- As several hash functions are used it is important to select efficient hash functions capable of producing non-correlated outputs

Testing hash functions

- A simple and basic test consists in:

1. Generating a large set of random keys
2. Process all the keys with the k hash functions
   - And storing all hash codes produced
3. Analyze the histogram of each hash function
   - For uniformity of distribution of the codes
4. Calculate, display and analyze the correlations between the codes produced by the several hash functions

Basic implementation: operations

- Bloom Filters have three basic operations:
  - Initialization
  - A membership test
  - and Insert/Update
- Initialize clears all the elements in the vector
- Insert computes the values of the k hash transforms for a key and updates the appropriate elements.
  - In the simplest case the update sets the element's flag bit
  - It requires time proportional to the number of hash functions
  - In more complicated cases additional information would also be placed in the element

Operations

- IsMember
  - computes the same hash values as Insert but instead of updating the elements it checks if they have been set
  - By definition, only members have their keys inserted into the vector
  - If any of the hash transforms, hi(**x**), compute a vector element that has not been set, then the key **x** could not have been inserted into the vector and therefore cannot be a member of the set
  - Worst time complexity for IsMember occurs for members (and non-members that are erroneously reported as members)

Time Complexities of Filter Steps

Procedure	Parameters	Time complexity
INITIALIZE	Table of N cells	$O(N)$
SET	Cell in Table	$O(1)$
CLEAR	Cell in Table	$O(1)$
ISSET	Cell in Table	$O(1)$
INSERT	Table, Key, and m hash transforms	$O(m)$
ISMEMBER	Table, Key, and m hash transforms	$O(m)$

Insert()

Insert(Table, Key)

- For i=1:m do
  - hi is the ith hash transform
  - Set(Table[hi(Key)])
- endfor
- end.

IsMember()

IsMember(Table, Key) -> Boolean

- i=0
- repeat
  - i=i+1
  - hi is the ith hash transform (1<i<=m)
- until ((i=m) | !(IsSet(Table[hi(Key)])))

- if i==m then
  - return (IsSet(Table[hi(Key)]))
- else
  - return(False)
- end.

Bloom Filters applications

- Bloom Filters find application wherever fast set membership tests on large data sets are required.
- Such applications include spell checking, differential file updating, distributed network caches, and textual analysis.
- It is a probabilistic method with a set error rate. Errors can only occur on the side of inclusion
  - a true member will never be reported as not belonging to a set,
  - but some non-members may be reported as members.

Applications: Spell checkers

- Determine if candidate words are members of the set of words in a dictionary
- Used in programs such as **cspell**
  - The Filter size was chosen to be large enough to allow the inclusion of additional words added by the user.
- Bloom Filters perform very well in such cases

Applications: Network applications

- Bloom Filters have recently found many applications in networks

- Example:
  - Routers use Bloom Filters and labels to probabilistically determine which hosts continue to send more than their share of traffic even when some of their data are dropped by the router.
  - Hosts which continue to operate in this non-cooperative fashion have more of their packets dropped.

Applications: Web

- Bloom Filters are used in caching proxy servers on the World Wide Web (WWW)
- Caching improves performance when clients obtain copies of files from neighboring servers instead of from the originating server (which may be several slow network links away)
- Proxy servers intercept requests from clients and either fulfill the requests themselves or re-issue them to servers

Application: Text Analysis

- Example: Find related passages in monograph
- Works by constructing a Bloom Filter of all the words in each passage of a monograph and then computing the normalized dot product of all pairs of them
- The result of every dot product is a similarity measure

Demonstrations

Online

- **Bloom Filters by Example**
  - http://billmill.org/bloomfilter-tutorial/
- Bloom Filters
  - https://www.jasondavies.com/bloomfilter/

Examples of Application (Matlab)

- Simple vegetables set
- Dictionary
- Random strings
  - And effect of the number of hash functions (k)
- Students enrolled in MPEI course (2018/2019)

Part of the slides adapted from: Mining Data Streams (Part 2)

Mining of Massive Datasets
Jure Leskovec, Anand Rajaraman, Jeff Ullman
Stanford University
http://www.mmds.org