Invertible Bloom Lookup Tables (aka Invertible Bloom Filters)

Hashing large sets in small memory for future reconciliation
IBLT: Invertible Bloom Lookup Table

- $U$: universe of possible elements
- $K$: subset of elements, $|K| = n$
- IBLT: table $T[0..m-1]$
- $T[i]$ has fields $T[i].\text{keysum}$ and $T[i].\text{counter}$
- $d$ hash functions $h_1, \ldots, h_d: U \rightarrow \{0, \ldots, m-1\}$
IBLT: Invertible Bloom Lookup Table

- $U$: universe of possible elements
- $K$: subset of elements, $|K| = n$
- IBLT: table $T[0..m-1]$
- $T[i]$ has fields $T[i].\text{keysum}$ and $T[i].\text{counter}$
- $d$ hash functions $h_1, \ldots, h_d$: $U \rightarrow \{0, \ldots, m-1\}$

**INSERT($k$):** for every $j$,

- $T[h_j(k)].\text{keysum} \leftarrow T[h_j(k)].\text{keysum} + k$  \hspace{1cm} (+ or $\oplus$)
- $T[h_j(k)].\text{counter} \leftarrow T[h_j(k)].\text{counter} + 1$

**DELETE($k$):** for every $i$,

- $T[h_j(k)].\text{keysum} \leftarrow T[h_j(k)].\text{keysum} - k$  \hspace{1cm} (− or $\oplus$)
- $T[h_j(k)].\text{counter} \leftarrow T[h_j(k)].\text{counter} - 1$
IBLT: Invertible Bloom Lookup Table

- $U$ : universe of possible elements
- $K$ : subset of elements, $|K| = n$
- IBLT: table $T[0..m - 1]$
- $T[i]$ has fields $T[i].\text{keysum}$ and $T[i].\text{counter}$
- $d$ hash functions $h_1, ..., h_d: U \rightarrow \{0, ..., m - 1\}$

- **LIST_KEYS($T$):**
  - while there exists index $i$ with $T[i].\text{counter} = 1$
  - for some $i$ do DELETE($T[i].\text{keysum}$)
- **LIST_KEYS($T$) succeeds** if the resulting table is empty
LIST_KEYS($A$): properties

- LIST_KEYS($A$) can be implemented in $O(m)$ time
- success of LIST_KEYS is closely related to peelability of underlying (hyper)graph
- phase transition ($c_d = m/n$)

<table>
<thead>
<tr>
<th>$d$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_d$</td>
<td>1.222</td>
<td>1.295</td>
<td>1.425</td>
<td>1.570</td>
<td>1.721</td>
</tr>
</tbody>
</table>

- if $m > (c_d + \varepsilon)n$, then LIST_KEYS($T$) succeeds with probability $1 - o(1)$ when $n \to \infty$
Application: set reconciliation

- Assume Alice and Bob hold respectively sets $S_A$ and $S_B$ such that $S_B \subseteq S_A$ and $\Delta = S_A \setminus S_B$ is small compared to $S_A, S_B$.
Application: set reconciliation

- Assume A(lice) and B(ob) hold respectively sets $S_A$ and $S_B$ such that $S_B \subseteq S_A$ and $\Delta = S_A \setminus S_B$ is small compared to $S_A, S_B$

- to find out $S_A \setminus S_B$,
  - A builds a IBLT for $S_A$ of size $O(|\Delta|)$ and sends it to B
  - B deletes all elements of $S_B$
  - the resulting IBLT can be used to list $S_A \setminus S_B$
Application: set reconciliation

- Assume Alice and Bob hold respectively sets $S_A$ and $S_B$ such that $S_B \subseteq S_A$ and $\Delta = S_A \setminus S_B$ is small compared to $S_A, S_B$
- to find out $S_A \setminus S_B$,
  - A builds a IBLT for $S_A$ of size $O(|\Delta|)$ and sends it to B
  - B deletes all elements of $S_B$
  - the resulting IBLT can be used to list $S_A \setminus S_B$

Comments:
- condition $S_B \subseteq S_A$ is not necessary, we can consider symmetric difference $\Delta = S_A \setminus S_B \cup S_B \setminus S_A$
- $\Delta$ can even be recovered from IBLTs of $S_A$ and $S_B$
IBLT: more comments

- IBLT can also store a satellite ‘value’ information via an additional *valuesum* field.
- Retrieval of the key value (or key membership) is not supported in general, but elements with $A[h_j(k)].\text{counter} = 1$ for some $j$ can be retrieved.
- Supports deletion of an element that is in the current set (otherwise the data structure is corrupted); can be handled by an additional *hashsum* field.
Count-Min sketch (aka Spectral Bloom filter)

Storing count information
How to support deletions in Bloom filters?
How to support deletions in Bloom filters?

- **Counting Bloom filter**: Bloom filter that, instead of 0 and 1, stores (small) counters

  - INSERT\((k)\): \(B[h_i(k)] \leftarrow B[h_i(k)] + 1\) for all \(i\)
  - LOOKUP\((k)\): check \(B[h_i(k)] > 0\) for all \(i\)
  - DELETE\((k)\): \(B[h_i(k)] \leftarrow B[h_i(k)] - 1\) for all \(i\)

- Also works for multi-sets

- Analysis shows that if Bloom filter is properly dimensioned (e.g. \(m/n \approx 10\)), then counters remain small (e.g. \(< 2^4 = 16\) [Fan et al. IEEE/ACM Trans. on Networking, 2000]
CountMin sketch

- What if we want to estimate the multiplicities (number of occurrences) of elements of a multi-set stored in a counting Bloom filter?
- Streaming framework
- *Example:* \( m = 8, \ C[0..7] \)

<table>
<thead>
<tr>
<th></th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
<th>( e )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_1 )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>( h_2 )</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

\( \text{b a d a e f c a ...} \)

#a? #e? #f? \( \uparrow \)

#c? #e? #f? \( \uparrow \)
CountMin: operations

- **UPDATE**(k): \( C[h_i(k)] \leftarrow C[h_i(k)] + 1 \) for all \( i \)
- count estimate of \( k \): \( \hat{f}(k) = \min_i h_i(k) \)

Let \( f(k) \) be the true count of \( k \)

Property: \( \hat{f}(k) \geq f(k) \)
CountMin sketch: analysis

- **Theorem**: if \( d = \log_2 \frac{1}{\delta} \) and \( m = \frac{ed}{\varepsilon} \), then
  \[
P[\hat{f}(k) \geq f(k) + \varepsilon N] \leq \delta,
  \]
  where \( d \) is the number of hash functions, \( f(k) \) is the true count of \( k \) and \( N \) is the total number of elements in the stream

**Proof**: \( C[h_i(k)] = f(k) + X_i(k) \)

\[
E[X_i(k)] = \frac{1}{m} \sum_{l \neq k} f(l) + \sum_{j \neq i} \frac{1}{m} \sum_k f(k) \leq \frac{d}{m} N = \frac{\varepsilon}{e} N
\]

\[
P[X_i(k) > \varepsilon N] < \frac{1}{e}
\]

Since \( \hat{f}(k) = f(k) + \min_i X_i(k) \), we have

\[
P[\hat{f}(k) - f(k) \geq \varepsilon N] \leq e^{-d} = \delta
\]
CountMin: properties

- Total space $m = \frac{e}{\varepsilon} \log \frac{1}{\delta}$
- Bound in Theorem is in terms of $N$
- CM-sketch also applies to increments $> 1$
- Decrements (“deletions”) are also supported provided that counters remain non-negatives
Application: **Heavy Hitters**

- Assume we want to output all elements that occur $N/50$ of times.
- Set $\varepsilon = 1/100$, build CountMin sketch ($m = 271 \cdot \log \frac{1}{\delta}$) and output all elements with CountMin estimate $\geq N/50$ ($N$ known in advance).
- Then we will output all desired elements, but also some elements occurring less, but not less than $N/100$, with $p = 1 - \delta$. 
What if $N$ is not known in advance?

- **Idea**: Maintain a min-heap of current frequent items, update after each element.
- After processing each element $x$, estimate $\hat{f}(x)$.
- If $\hat{f}(x) \geq \varepsilon N$ ($N$ current stream size), insert $x$ to the heap with value $\hat{f}(x)$ (or update if it was already there).
- If the smallest value of the heap (computed in $O(1)$) is $< \varepsilon N$, delete it from the heap.
- At the end, output all elements of the heap.
Sketches for mining big data streams
Processing big streaming data

- Big data is often streamed from a data source (sensors, cameras, internet, phone calls, …) or has to be processed in a streaming fashion (genomes, …)

- Common characteristics:
  - data comes at a (very) high rate
  - data cannot be stored, small working memory used
  - data should be processed online, in low time per item
  - approximate answers are often ok
Example 1: Sampling a stream

- **General idea**: sample a stream (e.g. consider 1/10 of the items) in hope that the sampled stream will have similar properties.
- **Cannot be done by simply sampling 1 over 10 items!**
- **Why?**
**Example 1: Sampling a stream**

- **General idea:** sample a stream (e.g. consider 1/10 of the items) in hope that the sampled stream will have similar properties.
- Cannot be done by simply sampling 1 over 10 items!
- **Why?** Assume we have a stream where $s$ items occur once and $d$ occur twice, and we want to estimate the fraction of repeated elements (right answer $\frac{d}{s+d}$).

If we sample each element with $p = \frac{1}{10}$, then out of $d$ repeated items, $\frac{d}{100}$ will occur twice in the sample, $\frac{81 \cdot d}{100}$ will disappear, and $\frac{18d}{100}$ will become unique.

The estimate will be $\frac{d}{s + \frac{18d}{100} + \frac{d}{100}} = \frac{d}{10s + 19d}$. 

Sampling a stream (cont)

- How to solve this problem?
Sampling a stream (cont)

- How to solve this problem? Use hash functions!

- Hash items to numbers, sample those whose hash ends with 0 (in decimal notation)
Example 2: Checking if an element has been “seen before”
Example 2: Checking if an element has been “seen before”

- Can be done with Bloom filters!
Example 3: Keeping multiplicities of elements and maintaining “heavy hitters”

- Find frequent (often occurring) elements in a stream

- *Example*: frequently viewed products in an online shop
Example 3: Keeping multiplicities of elements and maintaining “heavy hitters”

- Can be done with **CountMin** sketch!
Example 4: Counting distinct elements in a stream

- **Count-distinct problem**: count the number of distinct elements in a (very large) stream

  - **Examples**:
    - estimating the cardinality for memory allocation (Bloom filter)
    - unique users of a web site,
    - distinct IP addresses (routers, web servers, …)
    - detecting DoS attacks
    - number of distinct words ($k$-mers) in a (streamed) text (DNA sequence)
    - …
Approximate (probabilistic) counting (Morris 1977)

- Robert Morris (Bell Labs): maintain the logs of a very large number of events in small registers

**Algorithm:**
- maintain $K$ that stores (approx value of) $\log(n)$, i.e. size of $K$ is $\log \log(n)$
- initialize $K = 0$
- when a new event arrives, increment $K$ with probability $2^{-K}$
- $K$ reaches a value $k$ after expected $1 + 2 + 4 + \cdots + 2^{k-1} = 2^k - 1$ steps, i.e. $E[2^K - 1] = N$ ($N$ is true count)
- $\sigma^2 = N(N - 1)/2$ i.e. $\sigma \approx N/\sqrt{2}$ (70% of $N$)
Try this

- Implement this and run several times, acquire statistics:

  pick \( N \) (e.g. \( N = 15 \))
  \( c = 0 \)
  for \( i = 1 \) to \( 2^N \) do
    increment \( c \) with probability \( 2^{-c} \)
  print \( c \) // compare \( c \) with \( N \)
Try this

- Implement this and run several times, acquire statistics:

pick $N$ (e.g. $N = 15$)
$c = 0$
for $i = 1$ to $2^c$ do
    increment $c$ with probability $2^{-c}$
print $c$ // compare $c$ with $N$
How to improve?

- **Improvement 1**: change base of logarithms from 2 to some smaller \( b \ (b > 1) \), count \( \log_b N \)
  - increment \( K \) with probability \( b^{-K} \)
  - \( E[b^K - 1] = (b - 1)N, \sigma^2 = (b - 1)N(N - 1)/2 \)
  - price: space increased from \( \log \log n \) to \( \log \log_b n \)

- **Improvement 2**: keep \( m \) counters instead of just one, then compute the average/median …
Counting distinct elements in a stream
[Flajolet & Martin 85]

Main idea:

- hash elements into (binary) numbers $[0..2^L - 1]$ using a good hash function $h$
- hashes $\ldots x\ldots x\ldots$ are expected to occur $1/2$ time, $\ldots x\ldots x\ldots x\ldots 10$ are expected $1/4$ time, $\ldots x\ldots x\ldots x\ldots 100$ $1/8$ time, etc.
- $\max_k[\text{hash } \ldots 10^i]$ are all observed for $0 \leq i \leq k]+1$ is a good indication of $\log N$ ($N$ nb of distinct elements)
Flajolet & Martin algorithm: implementation

- maintain a bitmap of size $L$, set all bits to 0
- for each hash, compute position of the rightmost 1 and set the corresponding bit of bitmap to 1
- let $i$ be the position (from right) of the rightmost 0 in bitmap
- then the nb $N$ of unique elements is estimated as $2^{i-1}/\varphi$, $\varphi=0.77351..$

**Intuition:** if $i \ll \log N + 1$, then it is almost certainly 1; if $i \gg \log N + 1$, then $i$-th bit of bitmap is almost certainly 0

- $Pb$: big variance of results (accuracy within a factor of $\sim 2$)
- **Solution 1**: run the algo $m$ times, then take average/median/combination (accuracy $O(1/\sqrt{m})$)
- **Solution 2 [FM]**: stochastic averaging
  - use first $k$ bits of hash to dispatch elements into $m = 2k$ bins, then average; accuracy $\approx 0.78/\sqrt{m}$