CSEP 521: Applied Algorithms
Lecture 10
Stream Algorithms: Heavy Hitters

Richard Anderson
February 4, 2021
Announcements
Algorithms for data streams

- Data items received one at a time, N is number of items received
- Computation performed on each data item
- Memory is limited to being much less than N
  - Memory in thousands
  - Data in millions
Formal stream model

• Data items received one at a time, N is number of items received
• Computation performed on each data item
• Memory is limited to being much less than N
• It may be a constant $b$, or $\log N$
• Think of $b$ in thousands (or millions), $N$ in billions (or gazillions)
• Low runtime per item and stay within memory bounds
Heavy Hitters Problem

• Find elements which occur at least $n/k$ times
  • Cannot be solved exactly for $k \geq 3$

• $\epsilon$-Heavy Hitters: Approximation algorithm for Heavy Hitters

• Parameters $k$, $\epsilon$ and $\delta$:
  • Every value that occurs at least $n/k$ times in $A$ is in the list
  • Every value on the list occurs at least $n/k - \epsilon n$ times in $A$.
  • Probability of not achieving this is at most $\delta$
Count Min Sketch

• Simple data structure for estimating the number of occurrence of items
• Looks like a counting Bloom filter
• Counts provide an UPPER BOUND for number of occurrences
• Only accurate for counting the most frequent values
• Term “sketch” is used in streaming to refer to saving just a small amount of info as the data goes by
Multiple hash functions

• k hash tables with independent hash functions \( h_1(x) \ldots h_k(x) \)
  • We can think of \( k = 5 \)
• Each table has \( b \) buckets, where \( b \ll n \)
  • We can think of \( b = 1000 \)
  • \( \text{Int \ HT}[k][b] \)

• \( \text{Inc}(x) \), add one to each counter for \( x \), \( \text{HT}[j][h_j(x)]++ \)
• \( \text{Count}(x) \), \( \min(\text{HT}[1][h_1(x)], \text{HT}[2][h_2(x)], \ldots \text{HT}[k][h_k(x)]) \)

• Upperbound on the count (but can easily be wrong)
Example

<table>
<thead>
<tr>
<th></th>
<th>$h_1(x)$</th>
<th>$h_2(x)$</th>
<th>$h_3(x)$</th>
<th>$h_4(x)$</th>
<th>$h_5(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>H</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Sequence: A, B, C, D, A, D, A
### Example

<table>
<thead>
<tr>
<th></th>
<th>$h_1(x)$</th>
<th>$h_2(x)$</th>
<th>$h_3(x)$</th>
<th>$h_4(x)$</th>
<th>$h_5(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (16)</td>
<td>9</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>B (10)</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>C (6)</td>
<td>6</td>
<td>10</td>
<td>7</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>D (17)</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>E (2)</td>
<td>1</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>F (16)</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>G (6)</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>H (6)</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>I (4)</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>J (3)</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>K (5)</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

### Sequence:


<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>5</th>
<th>3</th>
<th>22</th>
<th>24</th>
<th>20</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>22</td>
<td>17</td>
<td>6</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>3</td>
<td>10</td>
<td>14</td>
<td>21</td>
<td>6</td>
<td>33</td>
</tr>
<tr>
<td>F</td>
<td>25</td>
<td>18</td>
<td>6</td>
<td>6</td>
<td>27</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td>22</td>
<td>8</td>
<td>27</td>
<td>21</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

A: 20, 22, 33, 25, 21
D: 24, 17, 33, 27, 27
F: 22, 19, 21, 18, 22
Heuristic Error Analysis

• $f_x$ is the true frequency count for $x$
• Single row analysis
• If we’re lucky, $HT[h(x)]$ will be the true count, $f_x$
• If we’re unlucky, $y$ collides with $x$, then $f_y$ contributes to $HT[h(x)]$

• In general, $HT[h(x)] = f_x + \sum_s f_y$ where $s = \{y \neq x : h(x) = h(y)\}$
• With a good hash function $h$, $x$ collides with an expected $1/b$ elements
• Therefore, we expect

$$HT[h(x)] = f_x + \frac{1}{b} \sum_{y \neq x} f_y \leq f_x + \frac{n}{b}$$
Lemma

• Let X be a positive random variable with expectation $E[X] = C$
• The probability that X is greater than $2C$ is at most one half
Error analysis

• Applying the lemma

\[
\text{Prob} \left[ \min_{i=1}^{k} HT[i][h_i(x)] > f_x + \frac{2n}{b} \right] = \prod_{i=1}^{k} \text{Prob} \left[ HT[i][h_i(x)] > f_x + \frac{2n}{b} \right] \leq \left(\frac{1}{2}\right)^k
\]

• Now consider \( k \) hash tables

\[
\text{Prob} \left[ HT[h(x)] > f_x + \frac{2n}{b} \right] \leq \frac{1}{2}
\]

• If we want error \( \delta \), we need \( k \geq \log (1/\delta) \)

• For \( \delta = .01 \) this is \( k = 7 \)

• For \( \varepsilon \)-Heavy Hitters, we want error at most \( \varepsilon n \), we take \( b = 1/\varepsilon \)
Rigorous analysis (see 2.5 in the notes)

• What we’ve covered up: choosing random hash functions
• Universal family of hash functions
• Markov’s Inequality
Universal Family of Hash Functions

• Really good practical hash functions exist
  • Fast and good distribution of keys
  • Cryptographic hash functions are difficult to invert and more work

• Choose a random hash function
  • Set of hash functions $H: U \rightarrow [1..m]$

• Universal property
  • For all $x, y$ in $U$, with $x \neq y$, if $h$ is chosen at random from $H$

$$\Pr[h(x) = h(y)] \leq \frac{1}{m}$$

• This is a minimal property for good hash functions
• Practical university families exist, so mathematically sound algorithms could be implemented
Carter-Wegman hash functions

• Hashing from [0..m-1] to [0..m-1]
• Choose prime p,  p >> m
• $h_{ab}(x) = ((ax + b) \mod p) \mod m$ where $1 \leq a < p$ and $0 \leq b < p$

• If a and b are chosen at random, $x \neq y$, then $\text{Prob}[h_{ab}(x) = h_{ab}(y)] = 1/m$
Markov’s Inequality

• If $X$ is a non-negative random variable and $c \geq 1$ is a constant

$$\text{Prob}[X > c \cdot E[X]] \leq \frac{1}{c}$$

• Crude method from converting from expectation to probability
\( \varepsilon \)-Heavy Hitters

- Find elements which occur at least \( n/k \) times with error range \( \varepsilon n \)

- Parameters \( k \) and \( \varepsilon \):
  - Every value that occurs at least \( n/k \) times in \( A \) is in the list
  - Every value on the list occurs at least \( n/k - \varepsilon n \) times in \( A \).

- Choose \( \varepsilon = 1/2k \)

- For CountMin, we take \( b = 1/\varepsilon \) and with \( j \) hash functions, where \( j = \log (1/\delta) \)

- Reasonable practical values are \( k=100 \) and \( \delta=.01 \), so this is a table of size 1000 for an \( n \) as large as you want!
Tracking the heavy hitters

• Note that we don’t even need to know what n is.
• We track the values of the potential heavy hitters as the algorithm runs
• The easiest way is to just keep the values on the $\varepsilon$-heavy hitters list in a heap as there are at most $2k$ of these values (independent of n)
Big numbers

- What is the biggest value of $N$ we need to worry about as an input size
- What is an appropriate domain, $[0..m-1]$ for a hash function

$2^{10} \approx 10^3$

Answer: $2^{64}$

- Big numbers:
  - Number of US social security numbers $10^9$
  - Populations of India, China: 1.4 Billion
  - World Population: 7.8 Billion
  - Stars in Galaxy $10^{12}$
  - Galaxies in Universe $10^{12}$

$log \log 2^{64} = \log 64 = 6$
More on hash functions

- Hashing from [1..m] to [1..t]
- Considerations
  - Uniformity, “randomness”
  - Sensitive to small changes
  - Speed
- Cryptographic security
- Hashing from U to [1..m]
  - Avoid losing information
  - Avoid regular collisions
  - Combine words with operations such as SHIFT and XOR
- You shouldn’t have to invent or code your hash functions

- Multiplicative approaches to hash functions are common
  - But have some risks on poor choices of multipliers
  - Saving middle part of multiplication is common
- Bitwise operations such as XOR, <<, >>
- Common to have algorithms based on particular key numbers
  - a=11400714819323198485 for Fibonacci hashing
  - Power of two close to $2^{64}$
Sample hash function: Fowler-Noll-Vo

```plaintext
algorithm fnv-1 is
    hash := FNV_offset_basis
    for each byte_of_data to be hashed do
        hash := hash × FNV_prime
        hash := hash XOR byte_of_data
    return hash
```

- The FNV_offset_basis is the 64-bit FNV offset basis value: 14695981039346656037.
- The FNV_prime is the 64-bit FNV prime value: 1099511628211.
- 64-bit unsigned arithmetic, so multiplication is mod $2^{64}$
Coming attractions . . .

• Hyperloglog