Announcements

• Homework 5 is out
  • Its raining, there’s homework . . .

• All students should be on the Ed discussion board now
  • Apologies for this Snafu – the problem was with CSEM registrations
Algorithms

• Designing computational processes

• Abstract expression of instructions for a task built on a set of computational primitives

• A key part of this class is thinking about algorithms across different computational settings

• Some problems become interesting (and important) in novel settings
Models of Computation

• Ground rules for defining computation
• Expression of key operations and resources
• Link with mathematics
• Abstract away grim reality of real devices
• Capture setting and constraints
Standard Model

- RAM (Random Access Machine)
- Idealized computer
- Natural instructions
- Unit cost models
- Compute functions of inputs
- Develop runtime functions
Memory Based Models

• Real systems are based on a memory hierarchy, and optimization for storage may be a central concern

• External Storage models
  • Consider costs for external access, as well as paged access

• Data Base Systems
  • View computations as interacting with internal state through DBMS
Stream Models

- Reacting to ongoing data sources
- Viewing data a single time
- Large quantities of data
- Limited local resources
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 bounds
Some trivial examples of Stream Algorithms

• Count the elements
• Find a specific element
• Count by property
• Average value
• Maximum

• Pick a random element
  • Not trivial, so its homework
Element Distinctness

• Determine if there are any duplicate elements
  • Yes/No question: Are all of the elements distinct

...,4,19,11,21,93,0,1,15,46,18,31,41,51,96,42,19,33
Theorem:
Element distinctness requires $\Omega(N)$ space

• Make the assumption you are drawing elements from a large domain
  • Assume elements are from 1..$N^2$ for N elements in the stream

• Heuristic argument:
  • You need to save all of the items, since the last one could match any one you have seen

• Rigorous argument:
  • This is much more work, as you need to define the model of computation to prevent cheating in storage – you need a model that counts the bits of storage
  • To make this work requires tools such as information theory (which is very cool!)
Majority Element

• Given a sequence of n elements, is there an element that occurs at least \( n/2 + 1 \) times.

• GME, MSFT, GME, GME, GME, AMZN, AMZN, GME, GME, GME, FB

• This is a standard exercise in Divide and Conquer algorithms

• Or if you are allowed to compare elements, you can sort, or compute the median and verify

• But there is a better way
Counter based algorithm

Find a majority element in array A of length n

Counter = 0;
Current = null;

for j = 1 to n
    if Counter == 0
        Current = A[j]
        Counter = Counter + 1
    else if A[j] == Current
        Counter = Counter + 1
    else
        Counter = Counter - 1

return Current

*I hate specifying algorithms as just code*
Correctness Proof

- If X is a majority element, it will be found by the counter based algorithm

- Counter for X increases + Counter for non-X decreases is at least $n/2 + 1$
How about this one? Breakout groups

• Determine if a stream algorithm can find an element that occurs more than n/3 times
Impossibility results
Heavy Hitters Problem

• Find elements which occur at least \( n/k \) times
• \( \varepsilon \)-Heavy Hitters: Approximation algorithm for Heavy Hitters

• 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 \( A \).

• In other words
  • At least \( n/k \) times. One the list
  • Between \( n/k - \varepsilon n \) and \( n/k \) times. Maybe on the list
  • Fewer than \( n/k - \varepsilon n \) times. Not on the list
Applications

• Most common items in a stream
• Detecting common searches
• Frequent stock trades
• TCP flows – identifying DOS attacks
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
Count values in a hash table

- First idea, store the counts in a cell indexed by the hash of a value
- What could go wrong? Do we worry about $\text{Hash}[x] = \text{Hash}[y]$?

\[
\text{for each } X \text{ in the stream} \\
\text{Count[Hash}[X] \text{]} = \text{Count[Hash}[X] \text{]} + 1
\]
Multiple hash functions (think Bloom filter)

• 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 $b << n$
  • We can think of $b = 1000$
  • Int $HT[k][b]$

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

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

• This is an approximation
• We are interested in estimating counts of most common items
• Counts for rare items will be garbage
• Epsilon approximation for Heavy Hitters
### 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>3</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>6</th>
<th>26</th>
<th>24</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>17</td>
<td>6</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14</td>
<td>16</td>
<td>6</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>18</td>
<td>6</td>
<td>6</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>8</td>
<td>27</td>
<td>16</td>
<td>6</td>
</tr>
</tbody>
</table>

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 $\mathbb{E}[X] = C$
• The probability that $X$ is greater than $2C$ is at most one half
Error analysis

• Applying the lemma

Now consider $k$ hash tables

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

• 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
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).
Coming attractions . . .

• Hyperloglog