15-853: Algorithms in the Real World

Announcements:

- HW 3 will be released today
- Due on Wednesday Nov 20

Reminder for last week’s announcements:

- Project reports due on Dec 3 2:30pm
- Format announced in last lecture. We will share a template this week.
- Project presentations are in class on Dec 3 and 5
Recall: Bloom filter

Representing a dictionary with far fewer bits when only need membership query.

Possible if we:
- Allow to make mistakes on membership queries
- No deletions

- Only false positives; no false negatives
  - may report that a key is present when it is not
Recall: Bloom filter

Space efficient data structure for *approximate* membership queries.
- Only false positives; no false negatives

- Keep an array $T$ of $M$ bits
  - Initially all entries are zero.
- $k$ hash functions: $h_1, h_2, \ldots, h_k : U \rightarrow [M]$  
  - Assume completely random hash functions for analysis

Adding a key:
- To add a key $x \in S \subseteq U$, set bits $T[h_1(x)], T[h_2(x)], \ldots, T[h_k(x)]$ to 1
Bloom filter

Membership query:
• For a query for key $x \in U$: check if all the entries $T[h_i(x)]$ are set to 1
• If so, answer Yes else answer No.

Q: Why no false negatives?
If an item $x$ is present, then corresponding bits will be set.

Q: Why false positives?
Other elements could have set the same bits.

Let’s analyze the probability of false positives.
Bloom filter

A false positive for a query occurs when all k bits in T corresponding to a query is set.

Let $p =$ probability that a bit in T is not set

$$p = \left(1 - \frac{1}{M}\right)^{kN} = \left(1 - \frac{1}{m}\right)^m \cdot \frac{kN}{m} \leq e^{-\frac{kN}{m}}$$

Prob. of false positive = all k bits set = $(1 - p)^k$

$$\left(1 - e^{-\frac{kN}{m}}\right)^k$$
Bloom filter

Q: What value of k minimizes prob. of false positives?

Differentiate and set to 0: Take \( \ln \left( \left(1 - e^{-\frac{kn}{m}}\right)^k\right) \)

\[
\frac{d}{dk} \left( k \ln \left(1 - e^{-\frac{kn}{m}}\right) \right) = -\frac{kn}{m} \ln (1 - e^{-\frac{kn}{m}}) + \frac{k e^{-\frac{kn}{m}}}{1 - e^{-\frac{kn}{m}}} \cdot \frac{n}{m}
\]

\[
k = \frac{M}{N} \ln(2)
\]

is a minima

Let \( \varepsilon \) denote the prob. of false positives.

Then <write>..

\[
\varepsilon = \left(\frac{1}{2}\right)^{\frac{M}{N} \ln 2}
\]

\[
\Rightarrow M = 1.44 N \log \left(\frac{1}{\varepsilon}\right)
\]
Bloom filter

Thus

\[ M \approx 1.44 N \log \left( \frac{1}{e} \right) \]

1.44 log(\(1/e\)) bits per element

E.g.: For 1% false positive probability, \( M \approx 10N \) and \( k = 7 \). Significantly smaller space than \( N \cdot \log(|U|) \) required to store the elements.
15-853: Algorithms in the Real World

Hashing:

- Concentration bounds
- Load balancing: balls and bins
- Hash functions (cont.)
- Data streaming model
Data streaming model

- Different computational model: elements going past in a “stream”
- Limited storage space: Insufficient to store all the elements

Assumptions:
- Denote the elements of the stream as $a_1, a_2, \ldots$
- Each element is from an alphabet $U$
- Each element takes $b$ bits to represent
  - E.g. 32-bit IP addresses
- The question: what functions of input stream can we compute with what time and space overhead.
Data streaming model

- Functions of interest:
  - Sum of all elements seen (easy)
  - Max of the elements seen (easy)
  - Median (tricky to do with small space)
  - Heavy-hitters, i.e., element(s) that have appeared most often)
  - Number of distinct elements seen

- Example application:
  - Switch or a router where packets are passing through.
Sampling vs. Hashing

Sampling is a natural option (since it helps reduce the amount of data)
But can lead to incorrect answers if not done correctly.

Example from [1]:
Suppose we want to figure out

```
#“uniques” = elements that occur exactly once.
```

Consider this sampling approach:

- Sample 10% of the stream by picking each element with probability 0.1.
- Count uniques and scale up the answer by 10

Sampling vs. Hashing

This will lead to incorrect answer:
Suppose stream length is $n$ and $n/2$ are uniques and $n/4$ appear twice.
Q: Correct answer is? $n/2$

In the sampled stream,
Expected length = $n/10$

$\text{#uniques} = 0.1 \times n/2 + n/4 \times (2 \times 0.1 - 0.1^2)$

(approx.) $n/10$

So our estimate of $\text{#uniques} = n$ (incorrect)

This is in expectation, but will hold with high probability as $n$ gets large (by Chernoff bound)
Sampling vs. Hashing

Q: What was the problem here?
Sampling decision was being made independently on each element of the stream.

Q: What we should have done?
If an element is sampled, all its copies are also sampled.

Q: How can we achieve this via hashing?
Hash the elements to the range \([10]\) and take elements that map to one value, say 0.
If we have at least 1-wise independence then we get \(1/10\) fraction of the stream along with duplicates.
Streams as vectors

Useful abstraction: viewing streams as vectors (in high dimensional space)

Stream at time t as a vector \( x^t \in \mathbb{Z}^{\mid U \mid} \)

\[
x^t = (x^t_1, x^t_2, ..., x^t_{\mid U \mid})
\]

Element \( i = \) number of times \( i^{th} \) element of \( U \) has been seen until time \( t \)

If next element is \( j \), then \( x_j \) is incremented by 1

Leads to an extension of the model where each element of the stream is either

(1) A new element or (2) old element departing (i.e. deletions).
Streams as vectors

That is, updates to the stream looks like (add e) or (del e).

Assumption: \#deletes for any element \(\leq\) \#additions.

\[\Rightarrow\] running count for each element is non-zero

This vector notation makes it easy to formulate some of the data stream problems:

- Heavy hitters = estimate “large” entries in the vector \(x\)
- Total number of elements seen = Sum of the elements of \(x\)
  \[\langle\text{write}\rangle\] (easy one)
- \#distinct elements = \#non-zero entries in \(x\)
Heavy hitters

Many ways to formalize the heavy hitters problem.

ε-heavy-hitters: Indices i such that $x_i > \varepsilon \| x \|_1$

Let us consider a simpler problem first.

**Count-Query:**
At any time t, given an index i, output the value of $x^t_i$ with an error of at most $\varepsilon\|x^t\|_1$. I.e., output an estimate

$$y_i \in x_i \pm \varepsilon \| x \|_1$$

Q: Given an algorithm for Count-Query, how to get heavy hitters?
To first order: we can look for i's s.t. $y_i > 0$
(at least a good first step)
Q: Would sampling work for Count-query?
No. Example: N copies of A arrives and then they all depart. Then sqrt(N) copies of B arrives.
At the end, heavy hitter = only B
But if we sample the elements with any prob. less that sqrt(N), we don’t expect to see any B.

Next:
Hashing-based solution: Count-Min Sketch
Hashing-based solution: Count-Min Sketch

On board.