CS 49/149: 21st Century Algorithms (Fall 2018): Lecture 13

Date: 25th October, 2018

Topic: Streaming. Frequency Moments. Heavy Hitters.

Scribe: Barry Yang

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors. Please email errors to barry.a.yang.gr@dartmouth.edu.

Streaming Algorithms

- Think of data as a very large array. Elements of an array come in as a stream. Once an element leaves the stream, it is gone.
- The stream can be thought of as an array of n things. Thus, we have $Array[1...n]$.
- Elements of the array are selected from the set ranging from 1 to m. Thus, $A[i] \in \{1, 2, ..., m\}$.
- Both n and m are very large.

Motivation We have an array for IP addresses. In this case, both m and n are very large. For example, m may be the number of IP addresses, and n represents m websites. The amount of memory available to us is much much smaller than n nor m. (memory $<< n, m$). If we had m space, then these problems would be trivial, but here we assume we have $< m$ space to count.

Frequency Moments

Suppose we are interested in how many times people are looking at a website. For each $i \in \{1, 2, ..., m\}$, f_i is the number of occurrences of i in A. In this case, frequency is the number of times a number appears in a stream.

As a check, we know that the sum of frequencies must equal to n, or

$$\sum_{i=1}^{m} f_i = n$$

Here is a question: suppose we fix i, what is f_i? Suppose we have $O(\log n), O(\log m)$ memory. For any fixed i, f_i can be exactly found in $O(\log n)$ bits for any fixed i. ✍

Question: *Find the frequencies of all elements in this trivial algorithm. This is $m \log n$ counters. How would you estimate, calculate the frequencies?*

Here are the kind of questions that we're interested in:

- Evaluation/estimating f_i for all i
- Maybe we don't want to estimate each f_i, instead we might want a broader statistic!

What are the settings of f_i that minimize the following? These indicate how balanced the frequency statistic is (for example, if f_x^2 is smaller, then the frequencies are more balanced).

- $\sum_{i=1}^{m} f_i^2 = F_2 = ||\vec{f}||_2^2$, where $\vec{f} = \{f_1, f_2, ..., f_m\}$
- $\sum_{i=1}^{m} f_i^k = F_k = ||\vec{f}||_k^k$
- 0-norm: F_0 Count the number of distinct entries in A.
- $F_\infty = \max_{i=1}^{m} f_1$ = Frequency of the most frequent item

Problem: We have an array, and we want to find the frequent elements.

Suppose in array, one element appears appears strictly more than $\frac{n}{2}$ times. Suppose this is element x. At some point in the sequence within the stream, there has to be two xs that are next to each other (pigeonhole principle).

For example, suppose we have the sequence {31145117}. In this case there is not one element that appears strictly more than $\frac{n}{2}$ times. In this case, we have divide the pairs into {31}, {14}, {51}, and {17}. Here, it is plain to see we do not have a pairing in which there are two 1s, which is the element that appears the most.

To find the element appearing more than $\frac{n}{2}$ times, we can divide and conquer, assuming the element exists. This can be done by finding all the elements that are paired, and then recursing on the subsequent sequences.

This type of algorithm is more formalized in the **Bayer-Moore Algorithm**.

BAYER-MOORE ALGORITHM: For finding elements appearing $> \frac{n}{2}$ times.

- initialize an empty "box" B
- while there are elements in the stream
  - $A \rightarrow$ step j: ($A[j]$ arrives)
  - if $B \neq 0$:
    - Put ($A[j]$, 1) in it
  - else if ($B = A[j]$):
    - Increment counter of B
  - else
    - decrement counter of B
    - if counter = 0, delete from B
- return just the number (the key in the dictionary)

However, this algorithm does not always necessarily find the majority element. What if there is no majority element? Then the number returned is not empty. For example, suppose we have the stream {4151617}. Here 7 is returned.

Why would the frequent element not be in the box?

- It cancels out another element
- It was kicked out by another element

In both cases, these are not added to the box. It is important to note that the majority element will never kick out its own element. We also note that for every element that has been kicked out, that element is paired with a different number responsible for kicking the element out. ✍

Question: *How would you modify the algorithm to find elements appearing at least $\frac{n}{10}$ times? Or more broadly, $\frac{n}{k}$ times, where k is a non-negative integer?*

For $\frac{n}{2}$, we need 1 box. So, for $\frac{n}{k}$, we need $k-1$ boxes. In this case, when a number kicks another number out of the box, then it empties every box as well. For example, suppose $k = 3$ and we have the stream $\{41316321912\}$. In this case, as the stream begins 4 and 1 fill seperate boxes.

Anything with appears more than $\frac{n}{3}$ cannot be grouped up.

MISRA-GRIES 1982 ALGORITHM: For finding elements appearing $> \frac{n}{k}$ times, it remains in the same box.

- initialize $k-1$ empty boxes
- while there are elements in the stream
  - $A \rightarrow$ step j: ($A[j]$ arrives)
  - check if $A[j]$ is in any box:
    - if so, increment counter
  - check if any box is empty
    - if so put $(A[j], 1)$ in that box
  - decrement count of all boxes
  - if some count = 0 empty that box
- return just the number (the key in the dictionary)

Boxes have distinct elements that are kicked or is kicked out. However, take note there are cases in the algorithm where there is junk that remains in the box (two numbers may be returned, and only one represents the frequent number).

Corollary: For each element $a \in$ box, $\hat{f}_a$ = estimate of count in the box.

$$\hat{f}_a \leq f_a \leq \hat{f}_a + \frac{n}{k}$$

Note that the above does not make any assumptions. If we want the estimate to satisfy $\hat{f}_a \leq f_a \leq \hat{f}_a + \varepsilon k$, we can set $k = \frac{1}{\varepsilon}$. This algorithm is deterministic and takes up $O(\frac{1}{\varepsilon}\log(n+m))$ space.

How to do better? ε is the l_1-norm of the frequencies.

$$\hat{f}_a \leq f_a \leq \hat{f}_a + \varepsilon||\vec{f}||$$

An Aside For any vector, which is bigger? l_1-norm or l_2-norm?

- $||v||_1 = \sum v_i$
- $||v||_2 = \sqrt{\sum v_i}$

l_1-norm in 2 dimensions is larger than l_2. In general, $||v||_1 \geq ||v||_2$.

Thus, we would like to find a randomized algorithm where ε is l_2-norm instead of l_1-norm.

Hashing

We have numbers between 1 and m. Suppose we have a hash function $h : [m] \to [k]$, where $[m] = \{1...m\}$ and $[n] = \{1...k\}$.

An algorithm is:

- maintain k counters
- a arrives, increment $h(a)$'s counter
- $\hat{f}_a =$ counter($h([a])$

Here, we note that for all a, will has to $h(a)$. Also note that $\hat{f}_a > f_a$, as two distinct elements $a \neq b$ may hash such that $h(a) = h(b)$.

Formally, $\forall a : \hat{f}_a = f_a + \sum_{b:h(a)=h(b),b\neq a} f_b$, where $\sum f_b$ represents an extra term. For a better estimate, we want to minimize $\sum f_b$. Thus, what is the chance of $h(a) = h(b)$?

If h is chosen from H, which is a universal hash function, $Pr_{h\in H \text{ for } a\neq b}[h(a) = h(b)] \leq \frac{1}{k}$.

$$E[\hat{a}] = f_a + E_h[\sum_{b:h(a)=h(b)} f_b]$$

$$= f_a + \sum_{b\neq a} f_b \cdot Pr[h(a) = h(b)]$$

$$\leq f_a + \frac{1}{k}\sum_{b\neq a} f_b$$

What we want: $E_h[\hat{f}_a] \leq f_a + \varepsilon n$

If $k = \frac{1}{\varepsilon}$,

$$E_h[\hat{f}_a] \leq f_a + \varepsilon \sum_{b\neq a} f_a$$

Idea 2

- h: $[m] \to [k]$
- when a arrived in stream

  counter [h(a)] = counter [h(a)] + 1
- $\hat{f}_a$ = counter [h(a)]

Let's say we decrement counter. When a arrived in stream. counter $[h(a)]$ = counter($h(a)$) - 1 and return negative.

$\hat{f}_a$ = -counter ($h(a)$)

Suppose for some counter do +1 and others do -1.

New Idea vs. Old Idea If I have collisions that pile up. To remove, have a +1, -1 in counter.

How to set +/- 1 for each a.

- $g : [m] \to \{-1, 1\}$
- When a arrives in stream, count [$h(a)$] = count($h(a)$) + g(a)
- $\hat{f}_a = g(a) \cdot \text{count}[h(a)]$

Algorithm Summary

- Instead of m counters, maintain k counters
- Maintain sign: $[m] \to \{-1, 1\}$
- count[h(a)] = count[h(a)] + g(a)
- $\hat{f}_a = g(a) \cdot \text{count}[h(a)]$

Note: can get negative number. Will be within +/- of two frequencies.

Count-Sketch Algorithm Claim:

$$E_{h \in H, g \in G}[\hat{f}_a] = f_a$$

$$p_f : E_{h,g}[g(a) \cdot \text{count}[h(a)]]$$

$$= E_{h,g}[g(a) \cdot (g(a) \cdot f_a + \sum_{b \neq a, h(a)=h(b)} g(b) \cdot f_b)]$$

$$= E_{h,g}[f_a + \sum_{b \neq a, h(a)=h(b)} g(a)g(b) \cdot f_b]$$

$$= f_a + \sum_{b \neq a, h(a) \neq h(b)} f_b \cdot E_{g,h}[g(a)g(b)]$$

$E_{g,h}[g(a)g(b)]$ is **not independent**. Instead it is 0, because $g(a)g(b)$ is +1 half the time and -1 half the time.

$$= f_a$$

Thus, this is an unbiased estimator!
When we have an unbiased estimator,

$$Pr[|\hat{f}_a - f_a| \geq \varepsilon ||\vec{f}||_2] \leq \delta$$

$$\text{Var}(\hat{f}_a) \cdot \frac{1}{\varepsilon^2 ||\vec{f}||_2^2} \cdot \ln(\frac{1}{\delta})$$

Note that $\text{Var}(\hat{f}_a) = E[\hat{f}_a{}^2] - (E[\hat{f}_a])^2 = E_{h,g}[\text{count}(h(a))^2]$

$$= E_{h,g}[((g(a) \cdot f_a + \sum_{a \neq b, h(a)=h(b)} g(b) \cdot f_b)^2]$$

$$= E_{h,g}[f^2 + \sum_{a \neq b, h(a)=h(b)} f_b^2 + \sum g(b)g(b')f_b f_{b'}]$$

Here, we note that $\sum g(b)g(b')f_b f_{b'} = 0$

$$= E_{h,g}[f_a^2 + \sum_{a \neq b, h(a)=h(b)} f_b^2]$$

$$\leq \sum_{j=1}^{m} f_i^2$$

$$\implies \text{Var}(\hat{f}_a) \leq ||\vec{f}||_2^2$$

Remember

$$\cancel{\text{Var}(\hat{f}_a)} \cdot \frac{1}{\varepsilon^2 \cancel{||\vec{f}||_2^2}} \cdot \ln(\frac{1}{\delta})$$

$$= \frac{1}{\varepsilon^2} \cdot \ln(\frac{1}{\delta})$$

This represents "parallel runs." However, the catch is that space is $\frac{1}{\varepsilon^2}$, not $\frac{1}{\varepsilon}$.