CSEP 521: Applied Algorithms
Lecture 9
Algorithms for Streams

Richard Anderson

February 2, 2021

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

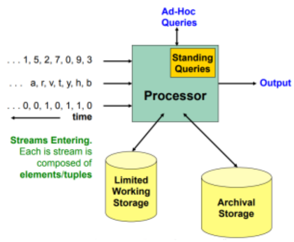
- RAM (Random Access Machine)
- Idealized computer
- Natural instructions
- Unit cost models
- Compute functions of inputs
- Develop runtime functions

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

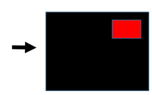
Some trivial examples of Stream Algorithms

- 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 Ω(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
- ε-Heavy Hitters: Approximation algorithm for Heavy Hitters
- Parameters k and ε:
  - 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 – εn times in A.
- In other words
  - At least n/k times. One the list
  - Between n/k – εn and n/k times. Maybe on the list
  - Fewer than n/k – εn 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

```
for each X in the stream
      Count[Hash[X]] = Count[Hash[X]] + 1
```

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

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

	$h_1(x)$	$h_2(x)$	$h_3(x)$	$h_4(x)$	$h_5(x)$
A	9	2	10	2	5
B	1	7	4	7	3
C	3	10	7	10	9
D	7	3	10	7	3
E	1	10	7	4	2
F	6	9	8	4	1
G	10	6	9	5	2
H	7	4	7	6	1

1		1				2		3	
	3		2			1			1
			1			1			3+2
	3					1+2			1
		1+2		3				1	

Sequence: A, B, C, D, A, D, A

Example

	$h_1(x)$	$h_2(x)$	$h_3(x)$	$h_4(x)$	$h_5(x)$
A (16)	9	2	10	2	5
B (10)	1	7	4	7	3
C (6)	3	10	7	10	9
D (17)	7	3	10	7	3
E (2)	1	10	7	4	2
F (16)	6	9	8	4	1
G (6)	10	6	9	5	2
H (6)	7	4	7	6	1

12		6			26	24		16	
	16	17	6		6	10		16	8
			10			14	16	6	33
	16		18	6	6	27			6
24	8	27		16				6	

Sequence: A, B, C, D, A, D, A, A, D, B, F, G, G, H, E, B, A, A, B, A, A, D, D, D, C, A, B, C, D, E, F, G, H, G, H, B, D, D, H, A, B, D, A, D, A, B, C, C, D, A, F, F, F, F, F, F, G, H, A, B, D, D, D, D, A, B, C, F, G, H, F, F, F, D, F, A, F, F, F

A short, incorrect proof is often more convincing than a longer correct one – J.D.Ullman

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≠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

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

- 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 δ, we need k ≥ log (1/δ)
- For δ = .01 this is k = 7
- For ε-Heavy Hitters, we want error at most εn, we take b = 1/ε

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→[1..m]
- Universal property
  - For all x, y in U, with x ≠ y, if h is chosen at random from H

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

  - 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 ≥ 1 is a constant

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

- Crude method from converting from expectation to probability

ε-Heavy Hitters

- Find elements which occur at least n/k times with error range εn
- Parameters k and ε:
  - 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 – εn times in A.
- Choose ε = 1/2k
- For CountMin, we take b = 1/ε and with j hash functions, where j = log (1/δ)
- Reasonable practical values are k=100 and δ=.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 ε-heavy hitters list in a heap as there are at most 2k of these values (independent of n)

Coming attractions . . .

- Hyperloglog