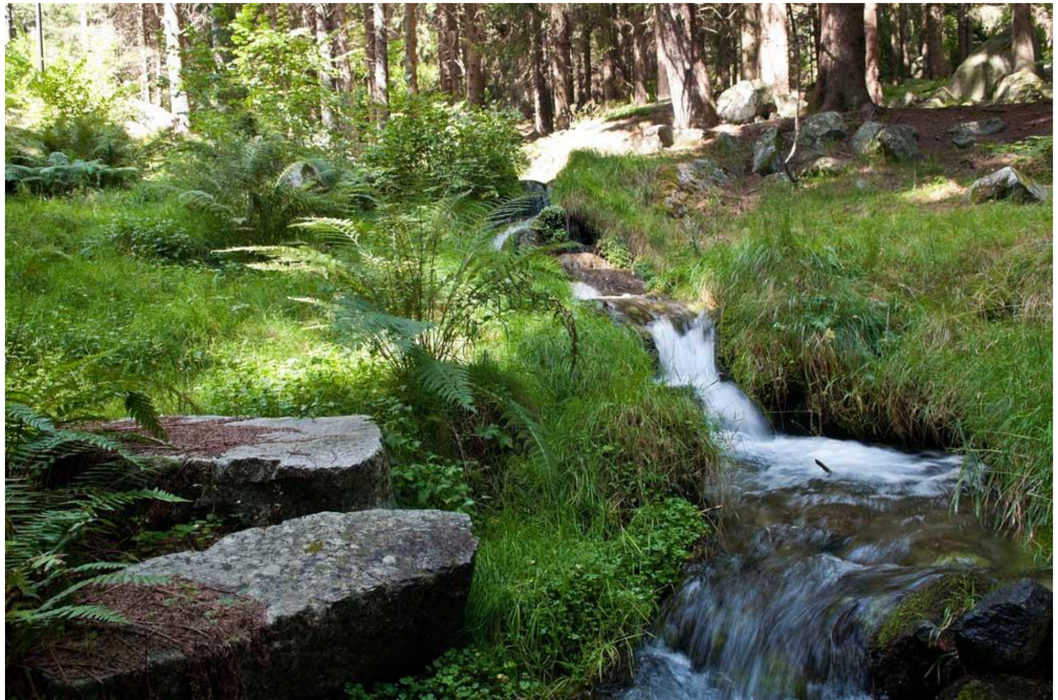
CSEP 521: Applied Algorithms
Lecture 10
Stream Algorithms: Heavy Hitters

Richard Anderson

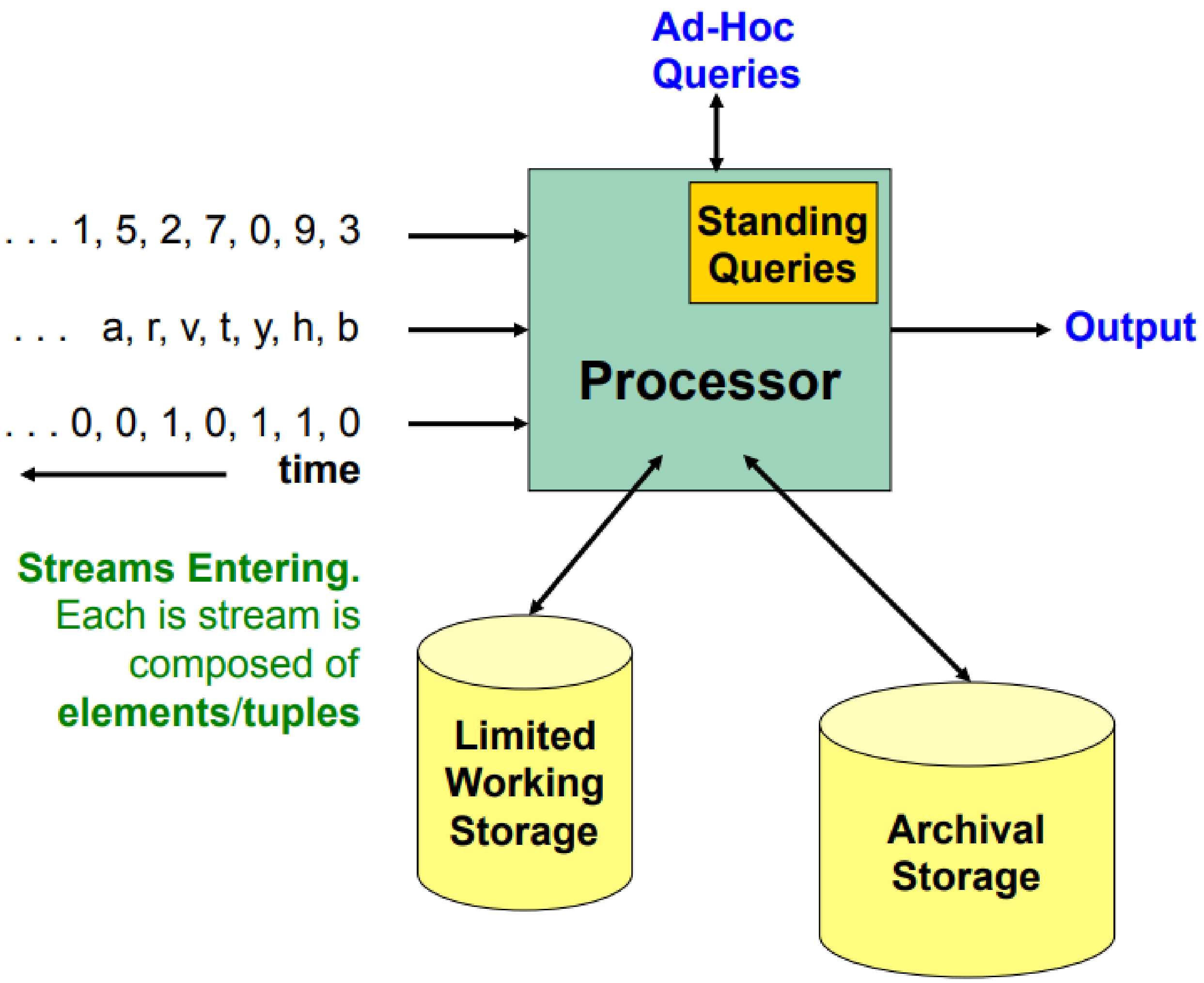
February 4, 2021



Announcements

Algorithms for data streams

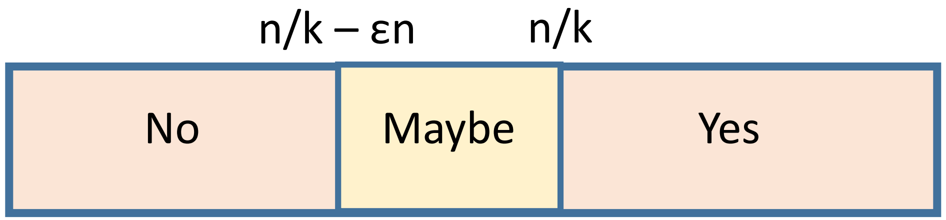
- Data items received one at a time, N is number of items received
- Computation performed on each data item
- Memory is limited to being much less than N
  - Memory in thousands
  - Data in millions



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 $k \geq 3$
- ε-Heavy Hitters: Approximation algorithm for Heavy Hitters

- Parameters k, ε and δ:
  - 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.
  - Probability of not achieving this is at most δ

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 $h_1(x)$ . . . $h_k(x)$
  - 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)

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)	6	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
I (4)	9	2	1	2	7
J (3)	4	9	3	9	8
K (5)	2	8	8	6	5

12	5		3		22	24		20	6
	22	17	6		6	10	5	19	8
4		3	10			14	21	6	33
	25		18	6	6	27		3	6
22	8	27		21		4	3	6	

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

Sequence: A, B, C, D, A, D, A, J, A, I, D, B, F, G, G, H, E, B, A, K, A, B, A, A, D, D, D, C, I, A, B, C, D, E, F, G, K, H, G, H, B, D, D, K, H, A, B, I, D, A, D, A, B, C, C, D, A, F, F, F, I, F, F, F, G, H, A, B, D, K, D, D, D, A, K, B, C, F, G, H, F, F, J, J, F, D, F, A, F, F, F

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

- Applying the lemma

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

- Now consider k hash tables

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

- 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

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

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

Carter-Wegman hash functions

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

- If a and b are chosen at random, $x \neq y$, then $\mathrm{Prob}[h_{ab}(x) = h_{ab}(y)] = 1/m$

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 $n/k - \varepsilon n$ 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 ε-heavy hitters list in a heap as there are at most 2k of these values (independent of n)

Big numbers

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

- 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

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

```
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