CS 261: Data Structures

Week 4: Streaming and sketching

Lecture 4b: Heavy hitters, count-min sketch and invertible Bloom filter

David Eppstein
University of California, Irvine

Spring Quarter, 2025

Majority and heavy hitters

Impossibility of most-frequent item

We've seen the mean and median; what about the mode (most frequent item) in a sequence?

Impossible in general!

Consider the state of a sketch after $n - 2$ items

Suppose we have already seen one item appear three times, and $(n - 3)/2$ pairs of other items

If the next two items are equal, with value x, then the result should be

- The triple, if x was not already seen
- The triple, if x has the same value
- x, if x has the same value as one of the pairs

So the sketch must know all the pairs, too much memory

Impossibility of majority

Majority element: one that occurs as more than half of the sequence elements

Naive formulation of the problem:

Either find a majority element if one exists

Or return “None” if there is no majority

Still impossible!

If first $n/2$ elements are distinct, any one could become majority

We don’t have enough memory to store all of their values

Boyer–Moore majority vote algorithm

Maintain two variables, m and c, initially None and 0

For each x in the given sequence:

- If $m = x$: increment c
- Else if $c = 0$: set $m = x$ and $c = 1$
- Else decrement c

Claims:

- If sequence has a majority, it will be the final value of m
- If no majority, m may be any element of the sequence (might not be frequently occurring)
- Algorithm cannot tell us whether m is majority element or not

[Boyer and Moore 1991]

Streaming majority example

With the sequence elements in left-to-right order:

x:		B	A	C	A	A	A	C	B	A
m:	None	B	B	C	C	A	A	A	A	A
c:	0	1	0	1	0	1	2	1	0	1
majority?	N	Y	N	N	N	Y	Y	Y	N	Y

(We know when there is a majority but the algorithm doesn't)

After the first step, B is in the majority, and the algorithm correctly reports B as its value of m.

After the third A (out of five sequence elements), A is in the majority, and stays in the majority for most of the remaining steps; the algorithm correctly reports A as its value of m.

When there is no majority (each element has at most half of the sequence) the algorithm may choose any element as its value of m.

Heavy hitters

Generalization of the same algorithm [Misra and Gries 1982]:

Store two arrays M and C, both of length k

Initialize cells of M to empty and C to zero

For each sequence item x:

- If x equals $M[i]$ for some i: increment $C[i]$
- Else if some $C[i]$ is zero: set $M[i] = x$ and $C[i] = 1$
- Else decrement $C[i]$ for all choices of i

Claim: In a sequence of n elements, if x occurs more than $n/(k+1)$ times, then x will be one of the elements stored in M

(Special case $k = 1$ is correctness of majority algorithm)

Why does it work?

This algorithm approximately counts occurrences for all sequence elements (not just the ones it stores)!

Define count(x) = actual number of occurrences of x,
estimate(x) = $C[i]$ (if $M[i] = x$ for some i) or 0 otherwise.

Then $\text{count}(x) - \dfrac{n}{k+1} \leq \text{estimate}(x) \leq \text{count}(x)$.

Proof idea: Induction on number of steps of the algorithm

If a step ends with x in M, we increase its count and estimate

Otherwise, we decrease all $k + 1$ nonzero estimates

Number of increases $\leq n \Rightarrow$ number of decreases $\leq n/(k+1)$

Because heavy hitters have big estimates, they must be stored

Count-min sketch

Main idea

We want to estimate frequencies of individual items (like heavy hitter data structure) in the turnstile model, allowing removals

Use hashing to map each item to a multiple cells in a small table
(like Bloom filter, but smaller)

Each cell counts how many elements map to it

- Include another copy of x: increment all cells for x
- Remove a copy of x: decrement all cells for x

Estimate of frequency of x: minimum of counts in cells for x

[Cormode and Muthukrishnan 2005]

Details

Set up according to two parameters ε and δ

- ε: how accurate the estimates should be
- δ: how likely estimates are to be accurate

2d table $C[i, j]$, initially all zero

- Each element maps to exactly one cell in each row
- Horizontal dimension (i): $\lceil e/\varepsilon \rceil$ cells
- Vertical dimension (j): $\lceil \ln 1/\delta \rceil$ cells

Hash functions h_j (for $j = 0, \ldots \lceil \ln 1/\delta \rceil - 1$)

- Include x: increment $C[h_j(x), j]$ for all j
- Remove x: decrement $C[h_j(x), j]$ for all j

Estimate of count(x): $\min_j C[h_j(x), j]$

(the smallest count among the cells for x in each row)

Accuracy

Clearly, estimate $\geq$ count; let $N =$ total number of elements

Claim: With probability $\geq 1 - \delta$, estimate $\leq$ count $+ \varepsilon N$

Follows from:

With probability $\geq 1 - \frac{1}{e}$, for any j, $C[h_j(x), j] \leq$ count $+ \varepsilon N$

(rows independent so ok to multiply probabilities $\Rightarrow (1/e)^{\#\text{rows}} \leq \delta$)

Overestimate comes from collisions with other elements

Expected number of collisions $= \epsilon N/e$. Instead of Chernoff,
use Markov's inequality: $Pr[X > c\, E[X]] \leq 1/c$

Turnstile medians

Data structure for approximate medians in the turnstile model,
of a collection S of integers $0, 1, 2, \ldots n$ (allowing repetitions):

For each $i = 0 \ldots \log_2 n$, round each number down to a multiple of 2^i, using the formula `round(x,i) = x &~ ((1<<i)-1)`

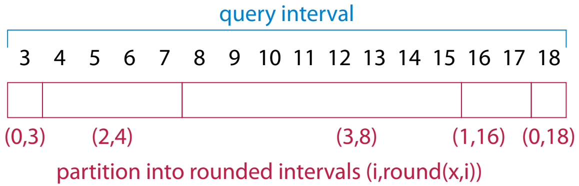
Construct $\log n$ count-min sketches,
one for each sequence of rounded numbers

(but strengthen accuracy and failure probability bounds by a factor of $\log n$ so that when we query all of them and combine the answers, the overall accuracy is what we want)

Finding the approximate median

To estimate the number of elements in the interval $[\ell, r]$:

Partition the interval into $O(\log n)$ rounded intervals
(sequences of 2^i elements that all round to the same value in S_i)



Use the rounded sketches to count elements in each rounded interval and sum the results

To find median: binary search for an interval $[0, m]$ containing approximately half of the elements

Set reconciliation

Find the missing element

Toy problem:

I give you a sequence of 999 different numbers, from 1 to 1000

Which number is missing?

Streaming solution: return $500500 - \sum(\text{sequence})$

Straggler sketching

Given a parameter k, maintain a turnstile collection of objects (allowing repetitions), such that

- Total space is $O(k)$
- The set can have arbitrarily many elements
- Whenever it has $\leq k$ distinct elements, we can list them all (with their numbers of repetitions)

E.g. could solve the toy problem by:

- Initialize structure with $k = 1$
- Insert all numbers from 1 to 1000
- Remove all members of sequence
- Ask which element is left

Invertible Bloom filter

Structure:

- Table with $O(k)$ cells
- $O(1)$ hash functions $h_i(x)$ mapping elements to cells (like a Bloom filter)
- "Fingerprint" hash function $f(x)$ (like a cuckoo filter)

Each cell stores three values:

- Number of elements mapped to it (like count-min sketch)
- Sum of elements mapped to it
- Sum of fingerprints of elements mapped to it

[Eppstein and Goodrich 2011]

How to list the elements

“Pure cell”: stores only a single distinct element (but maybe more than one copy of it)

If a pure cell stores element x, we can compute x as (sum of elements)/(number of elements)

Test whether a cell is pure: compute x and check that stored sum of fingerprints equals number of copies times fingerprint(x)

Decoding algorithm:

- While there is a pure cell, decode it, output it, and remove its elements from the collection
- If we reach an empty table (all cells zero), return
- Otherwise, decoding fails (table is too full)

Application: Set reconciliation

Problem: Two internet servers both have slightly different copies of the same collection (e.g. versions of a git repository)

We want to find how they differ, using communication proportional to the size of the difference (rather than the size of the whole set)

If we already know the size of the difference, k:

- Server A builds an invertible Bloom filter for its collection, with parameter k, and sends it to server B
- Server B removes everything in its collection from the filter and then decodes the result
- (Some elements might have negative numbers of copies! But the data structure still works)

[Eppstein et al. 2011]

How to estimate the size of the difference?

Use MinHash to find samples of server B's set with numbers of elements smaller by factors of $1/2$, $1/4$, $1/8$, $\ldots$

Sample i: elements whose hash has first i bits = zero

Server B chooses parameter $k = O(1)$ and sends invertible Bloom filters of all the samples in a single message to server A

Server A uses the same hash function to sample its set, and removes sampled subsets from B's filters

Estimate of the size of the difference = 1/sampling rate of largest sample whose difference can be decoded

Summary

Summary

- Sketch: Structure with useful information in sublinear space
- Stream: Loop through data once, using a sketch
- Cash register (addition only) and turnstile (addition and removal) models of sketching and streaming
- Mean and median; impossibility of exact median
- Maintaining a random sample of a stream and using it for approximate medians
- Majority voting, heavy hitters, and frequency estimation
- MinHash-based sampling and estimation of Jaccard distance
- Count-min sketch turnstile-model frequency estimation
- Using count-min sketch for turnstile median estimation
- Stragglers, invertible Bloom filters, and set reconciliation

References and image credits, I

Robert S. Boyer and J Strother Moore. MJRTY: a fast majority vote algorithm. In Robert S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–118. Kluwer Academic Publishers, 1991. Originally published as a technical report in 1981.

Goudey Gum Co. Baseball card of Babe Ruth. Public-domain image, 1933. URL `https://commons.wikimedia.org/wiki/File:BabeRuthGoudeycard.jpg`.

Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi: 10.1016/J.JALGOR.2003.12.001.

David Eppstein and Michael T. Goodrich. Straggler identification in round-trip data streams via newton's identities and invertible Bloom filters. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):297–306, 2011. doi: 10.1109/TKDE.2010.132.

References and image credits, II

David Eppstein, Michael T. Goodrich, Frank C. Uyeda, and George Varghese. What's the difference?: efficient set reconciliation without prior context. In Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul, editors, *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, pages 218–229, 2011. doi: 10.1145/2018436.2018462.

Jayadev Misra and David Gries. Finding Repeated Elements. *Science of Computer Programming*, 2(2):143–152, 1982. doi: 10.1016/0167-6423(82)90012-0.