DON'T CROSS THE STREAMS

COUNTING: AS EASY AS 1, 2, 3, 5

MARC MILLSTONE
LEAD, ANALYTICS AND MACHINE LEARNING (ANIML)
MARC@SOCRATA.COM

COUNTING IS HARD

COUNTING IS IMPORTANT

1. Service alerting
2. Filter Recommendations
3. (Big) Data analysis
4. ...

Evergreen

New York City 311 data

All requests from New York City's 311 system. From 2010 to present. The telephone number 3-1-1 is a special telephone number supported in many communities in the United States which provides access to non-emergency municipal services.

LAST UPDATED **7 Days Ago**

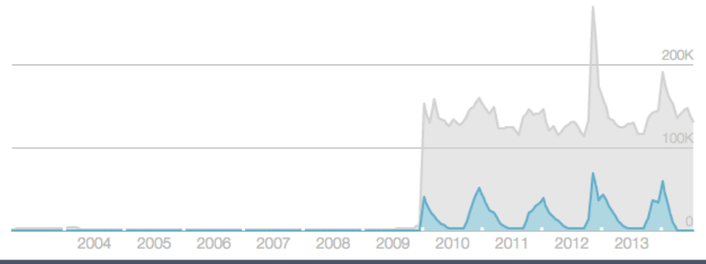
SOURCE DATASET **311 Service Requests From 2010...**

Save | Save As | Customize | Download | API

Showing all Requests where complaint type is HEATING | Clear All ×

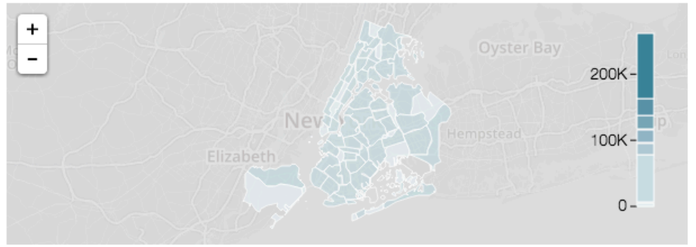
CREATED DATE

Date ticket was created



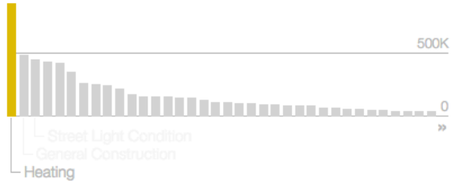
COMMUNITY DISTRICTS

Community district reporting 311 request



COMPLAINT TYPE

Type of 311 complaint



LOCATION TYPE

Type of location the issue concerns

AGENCY NAME

Agency responsible for addressing the issue

DATA TABLE

facility ty...	intersection street 2	city	school phone num...	school or citywide complaint	location	park borou...	latitude
N/A		BROOKLYN	Unspecified		(40.669422°, -73.98855...	Unspecified	40.6694217844600.
N/A		BRONX	Unspecified		(40.835444°, -73.92895...	Unspecified	40.8354438653203
N/A		BRONX	Unspecified		(40.8738°, -73.885955°)	Unspecified	40.8737995407732.
N/A		BRONX	Unspecified		(40.823192°, -73.89685...	Unspecified	40.8231919185387.
N/A		BRONX	Unspecified		(40.840709°, -73.92615°)	Unspecified	40.8407092045185.
N/A		BROOKLYN	Unspecified		(40.677505°, -73.96245...	Unspecified	40.6775050811210
N/A		ARVERNE	Unspecified		(40.59165°, -73.795036°)	Unspecified	40.5916498730791
N/A		BROOKLYN	Unspecified		(40.638571°, -73.94900...	Unspecified	40.638570680126
N/A		STATEN ISLAND	Unspecified		(40.635601°, -74.12047...	Unspecified	40.6356011645854
N/A		BRONX	Unspecified		(40.863151°, -73.86433...	Unspecified	40.8631514269651.
N/A		FOREST HILLS	Unspecified		(40.723713°, -73.84887...	Unspecified	40.7237134908036

Showing 1 to 11 of 903785 (Total: 7668243)

THREE APPROACHES TO COUNTING AT SCALE

1. Use engineering
2. Use math
3. Use ignorance

SPECIFIC PROBLEMS WE WILL LOOK AT

ESTIMATING THE CARDINALITY OF A DATA STREAM

Given a data stream, how many unique elements are there

HEAVY HITTERS, MOST FREQUENT ELEMENTS

Given a data stream, which elements appear most often

ESTIMATING THE FREQUENCY (COUNT) OF ITEMS

Given a data stream, answer queries as to the count of specific items

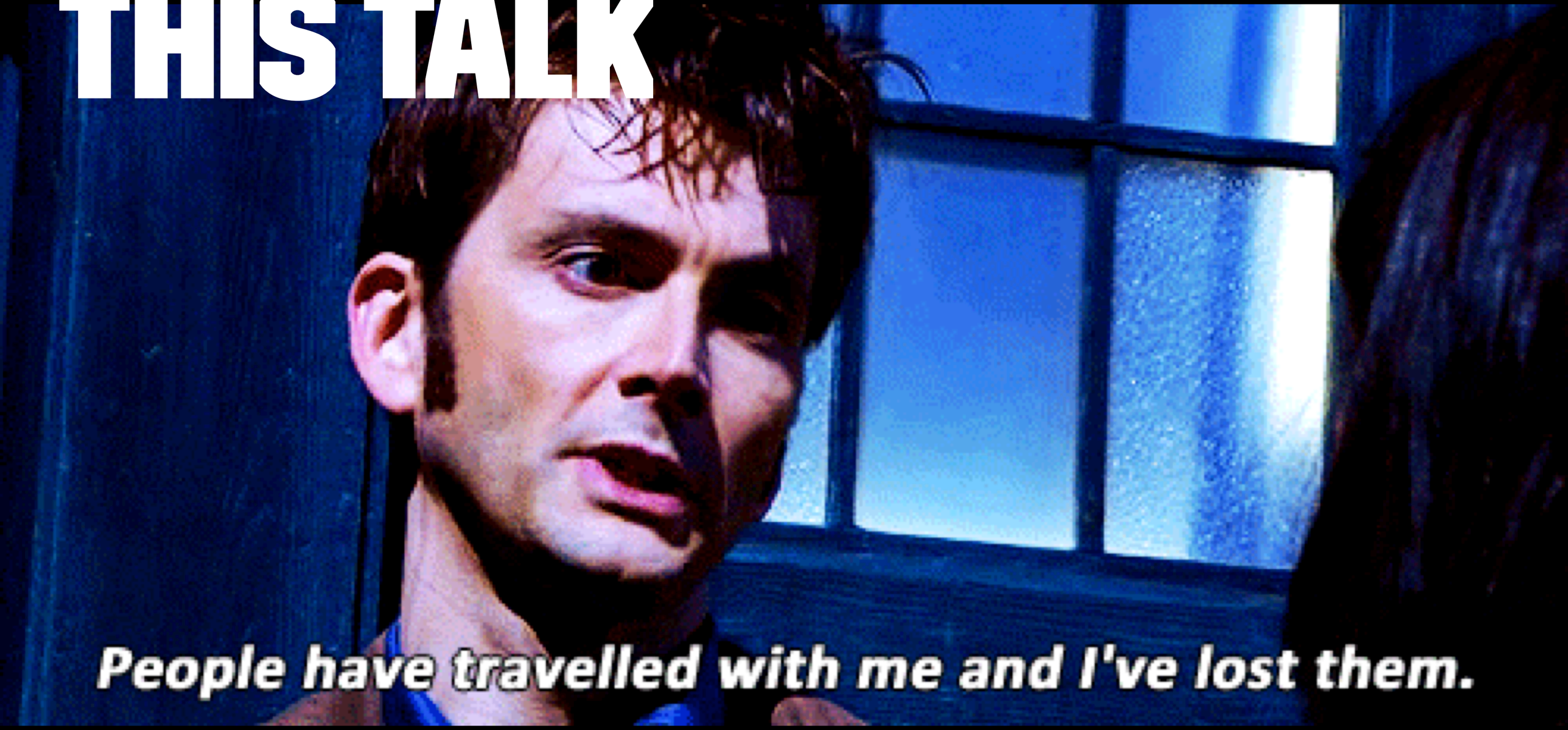
TOP K ELEMENTS (WITH COUNTS)

Given a data stream, output the top k elements with counts

FOLLOW ALONG

Tallyho: http://github.com/splittingfield/tallyho
Slides: in papers directory above
Algebird: https://github.com/twitter/algebird
Stream-lib: https://github.com/addthis/stream-lib

THIS TALK



BEFORE WE TALK

1. Talk is purposely short, with room to chat
2. Always stop to ask questions (we can learn together)
3. Goal is to be a discussion, not a lecture

AND REMEMBER

OUR RULES OF STREAMING DATA

» Data presented as a sequence of elements

» Each element can be processed only once

» Data structures may only consume constant memory overhead, independent of the stream size.

FIRST: OUR DATA PIPELINE

How we stitch together our computations.

THE PIPELINE DATA STRUCTURE

```
trait Pipeline[-I, +M, O <: HList]
```

» I: The type of the element

» M: The type after processing

» O: HList for combining results of multiple pipelines

THE PIPELINE IN MORE DETAIL

```
trait Pipeline[-I, +M, O <: HList] { self =>
  def pipeTo[M2, O2 <: HList](next:Pipeline[M, M2, O2]):Pipeline[I,M2,O2] = {
    new Pipeline[I,M2,O2] {
      override def process(elem:I): M2 = next.process(self.process(elem))
      override def results = next.results
    }
  }

  def alongWith[U <: I, M2, O2 <: HList](and:Pipeline[U,M2,O2])
      (implicit prepend : Prepend[O, O2]):Pipeline[U,(M,M2),prepend.Out] = {
    new Pipeline[U,(M, M2),prepend.Out] {
      override def process(elem:U) = {
        val f1 = self.process(elem)
        val f2 = and.process(elem)
        (f1,f2)
      }
      override def results = self.results ::: and.results
    }
  }
  def process(elem: I): M
  def results: O
}
```

SIMPLIFYING THE DATA STRUCTURE

```
abstract class AbstractSimplePipeline[I, M, O <: HList] extends Pipeline[I, M, O] {
    def step(item: I): M
    final def process(item: I)= step(item)
}

abstract class SimplePipeline[I,M,O] extends AbstractSimplePipeline[I,M,O :: HNil] {
    def result: O
    final def results = result :: HNil
}

abstract class Transformer[I, O] extends AbstractSimplePipeline[I, O, HNil] {
    final def results = HNil
}
```

COMPUTING THE CARDINALITY OF A STREAM

PROBLEM STATEMENT

Given a stream of data, return the number of distinct elements

A LOWER BOUND ON MEMORY USAGE

1. Let $a_1, a_2, \ldots, a_n$ be integers in the range 1 to m
2. Supposed we have seen $k > m$ values in the sequence
3. Subset seen could be 2^m subsets of $\{1, 2, \ldots, m\}$
4. m bits for each possible subset.

~~COMPUTING~~ APPROXIMATING CARDINALITY OF A STREAM

AN ASIDE ON HASHING

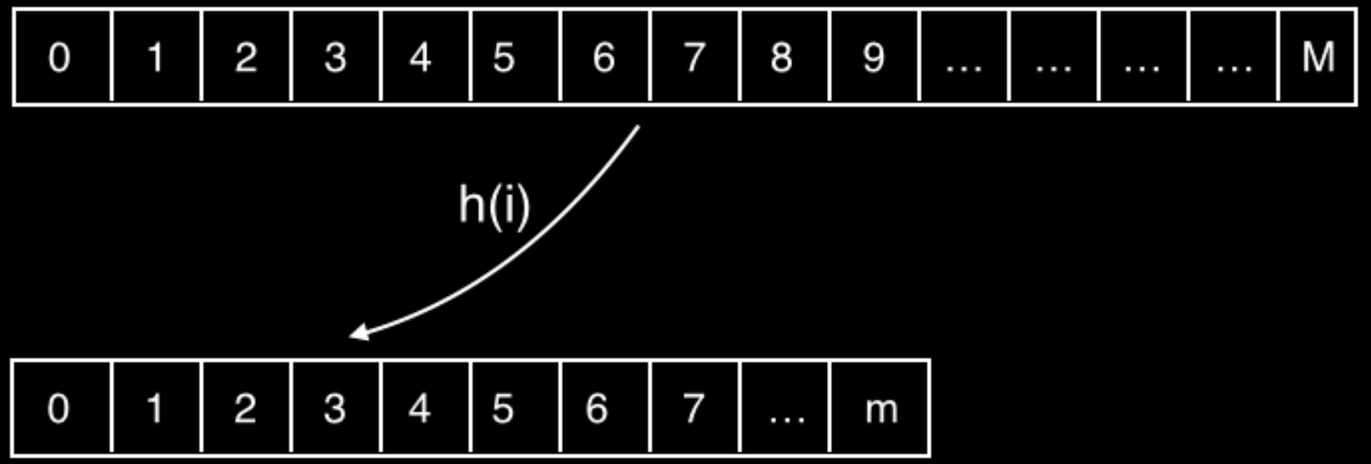
» Let U be a universe of symbols, $\|U\| = M$

» Map $U \to \{0, 1, 2, \ldots, m-1\} = [m]$, where $m \ll M$

» The set of functions $H = \{h : U \to [m]\}$ is a 2-universal hash family if $\forall x, y \in U, x \neq y$

$$Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{m}$$

» Basically, we expect collisions equivalent to random


0
1
2
3
4
5
6
7
8
9
…
…
…
…
M
h(i)
0
1
2
3
4
5
6
7
…
m

OUR HASH FAMILY

» i is an integer

» a, b chosen at random

» $p = 2^{31} - 1$ or $p = 2^{61} - 1$

» m is range of hash functions

$$h(i) = (a * i + b \bmod p) \bmod m$$

» if m is a power of 2, we can compute this quickly

A PROOF

$$h(i) = (a * i + b \bmod p) \bmod m$$ **IS 2-UNIVERSAL**

$$\begin{aligned} h(i) &= h(j) \quad (i \neq j) \\ ai + b &= (aj + b \bmod p) \bmod m \\ ai + b &= aj + b + l * m \bmod p \\ ai - aj &= l * m \bmod p \\ a &= (i - j)^{-1}(l * m) \bmod p \end{aligned}$$

» $p - 1$ possible values for a

» l varies in range $\left\lfloor \frac{p}{m} \right\rfloor \longrightarrow \frac{\left\lfloor \frac{p}{m} \right\rfloor}{p - 1}$ collision probablity

WAIT, WHAT ABOUT OBJECTS?[1]

» Let x be an object (string)

» Given two hash functions, $h_1(x)$ and $h_2(x)$

» Define $g_i(x) = h_1(x) + ih_2(x)$ for $i \in (0, \ldots, d-1)$

» For strings, use MurmurHash.

[1] Kirsch, Adam, and Michael Mitzenmacher. "Building a better bloom filter." In Proceedings of the 14th Annual European Symposium on Algorithms (ESA. 2005.)

WHY DO WE CARE ABOUT HASHING?

A SIMPLE APPROACH TO COUNTING

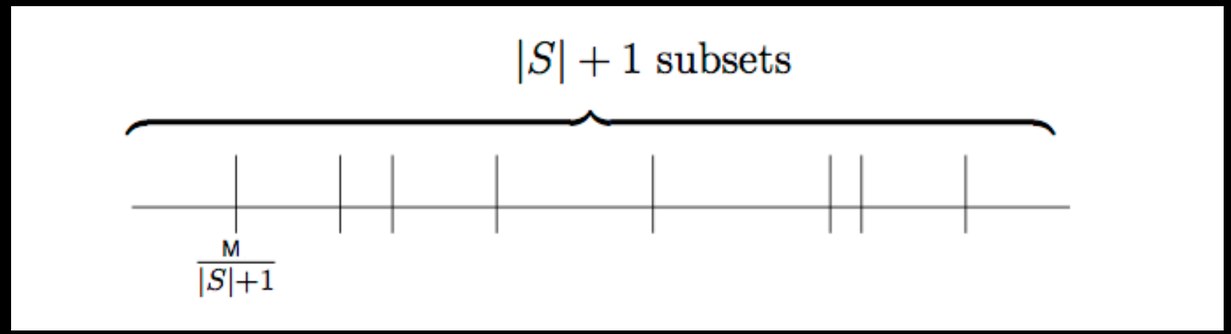
1. $E \in \{1, 2, \ldots, M\}$ be our sequence.
2. Let S be the set of distinct elements seen
3. Suppose we choose elements of S uniformly at random from E.
4. What is the minimum value of S?
5. How is this useful?

THE ANSWER

1. $\mathcal{S}$ partitions E into $|\mathcal{S}| + 1$ subsets.

2. Minimum is approximately $\dfrac{M}{|\mathcal{S}| + 1}$

3. $|\mathcal{S}| = \dfrac{M}{\min(\mathcal{S})} - 1$



WE CAN'T RANDOMIZE OUR SEQUENCE

HASHING PROVIDES THAT RANDOMIZATION

1. Hash each element in our sequence, with a 2-universal family
2. Apply this heuristic.

Let d be the number of distinct elements. Assume $m > 100d$. With probability at least $\frac{2}{3}$,

$$\frac{d}{6} \leq \frac{m}{min} \leq 6d$$

WE COULD RANDOMLY SAMPLE OUR STREAM THOUGH...

RANDOM SAMPLING

1. Let $U, (|U| = M)$ be our sequence of elements.
2. Goal: Select $S, (|S| = m)$ to approximate M.
3. So pick $u \in U$ with probability $\frac{m}{M}$.
4. Uhm... we do not know M.

SAMPLING FROM A STREAM[Vitt]

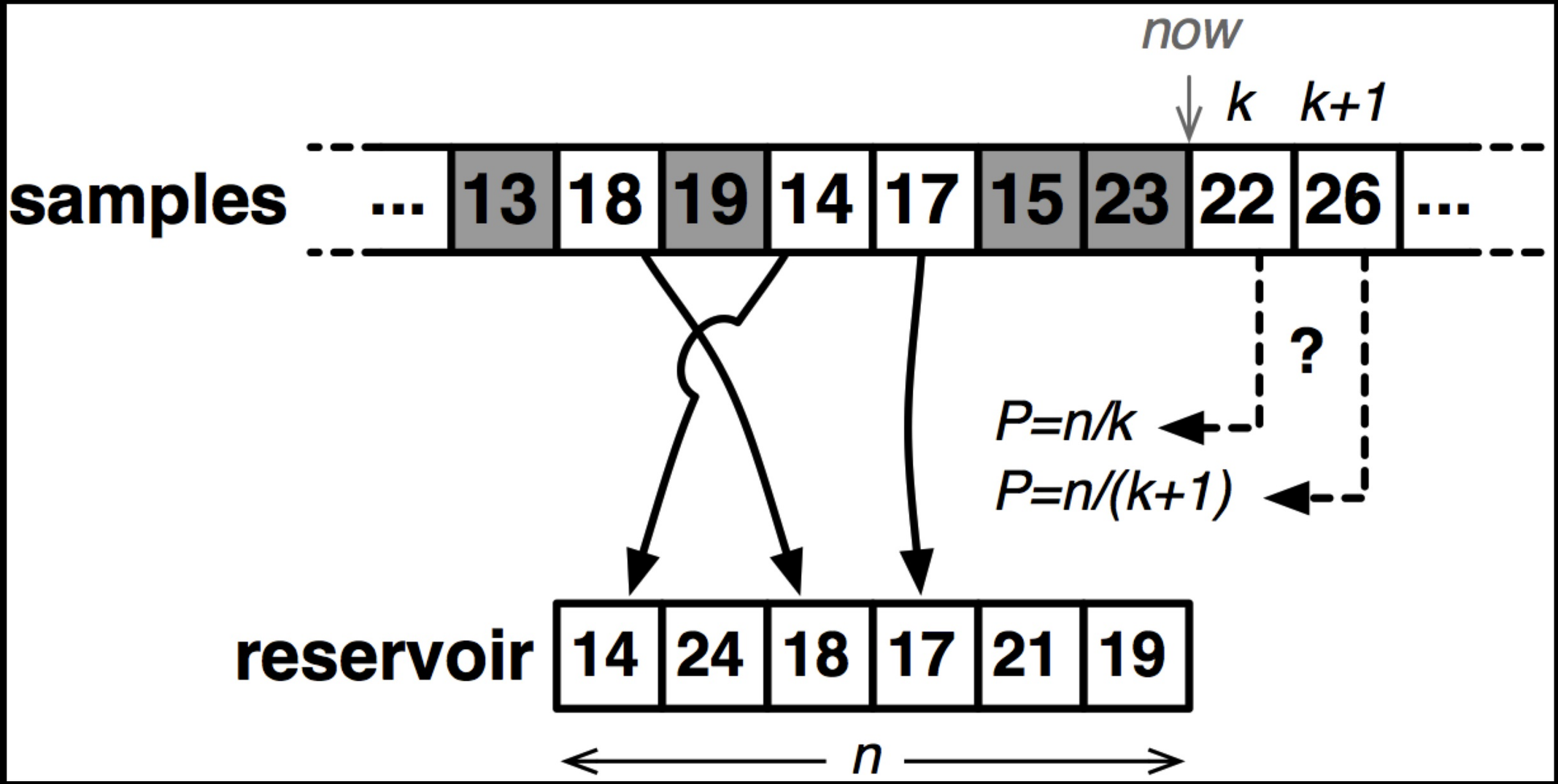
[Vitt] Vitter, Jeffrey S. "Random sampling with a reservoir." ACM Transactions on Mathematical Software (TOMS) 11.1 (1985): 37-57.
APA

RESERVOIR SAMPLING

1. $U = \{a\}$
   - » Return a with probability 1.
2. $U = \{a, b\}$
   - » Store a (assuming it comes first)
   - » When we see b
   - » Generate random number between 0 and 1.
   - » If smaller than .5, return a, else b

IN GENERAL

1. Fix sample size k and let R be an array storing our sample
2. Store the first k elements into R
3. For the j'th element, a_j
   - » Let p_j be a random number in $(0, 1)$
   - » Replace a random element in R with a_j if $p_j \leq \frac{k}{j}$



Maldonado, Walther, et al. "Deadline-aware scheduling for software transactional memory." Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on. IEEE, 2011.
APA

CLAIM:

After processing the stream, an element will be contained in the reservoir with probability $\frac{k}{M}$

PROOF

Left as an exercise for the passionate reader. [Hint]

[Hint] Use induction.

We keep the ith element with probablity $\frac{k}{i}$

Probablity that an element is removed is $\frac{1}{k}$

What is overall probablity of an element being in the reservoir after i rounds?

Reservoir sampling can be extended to when the distribution is weighted. Still need to know the weights though...

BACK TO PROBLEM AT HAND

~~COMPUTING~~ APPROXIMATING THE CARDINALITY OF A STREAM

HYPERLOGLOG [2]

[2] Flajolet, Philippe, et al. "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm." DMTCS Proceedings 1 (2008).

SOME OBSERVATIONS

1. Imagine the set of bit strings $x_7 x_6 x_5 \ldots x_0$
2. Denote (number of leading 0's) + 1 as rank (ρ)
3. What fraction are of the form $1 x_6 x_5 \ldots x_0$? (rank 1)
4. What fraction are of the form $01 x_5 \ldots x_0$? (rank 2)
5. What fraction are of the form $000 \ldots 1$? (rank r)

$2^{\max rank}$ approximates cardinality (poorly)

MAKING HYPERLOGLOG WORK (AN EXAMPLE)

1. $x = x_7 x_6 x_5 \ldots x_0$ (our bit strings)
2. Let $p = 2$. Choose p bits from the start of x
3. $m = 2^2$ counters, $m_{00}, m_{01}, m_{10}, m_{11}$
4. For each x, update max rank($x_5 x_4 x_3 x_2 x_1 x_0$) in $m_{x_7 x_6}$

EVEN MORE CONCRETELY

1. $x_1 = 1000000\,,\quad x_2 = 01110001\,,\quad x_3 = 01001111\,,\quad x_4 = 11000011$

2. $m_{00} = 0\,,\quad m_{01} = \max\{3, 1\}\,,\quad m_{10} = 7\,,\quad m_{11} = 5$

MAKING HYPERLOGLOG WORK (IN GENERAL)

1. Choose p bits from the start of each hashed value
2. Choose $m = 2^p$
3. Divide input stream S into m substreams, S_i
4. $$\begin{aligned} x \longrightarrow h(x) &= x_{n-1}x_{n-2}\dots x_0 \\ &= (x_{n-1}x_{n-2}\dots x_{n-p})x_{n-(p+1)}x_{n-(p+2)}\dots x_0 \end{aligned}$$
5. Let $M[i] = \max_{x \in S_i} \rho(x_{n-(p+1)}x_{n-(p+2)}\dots x_0)$

HLL CONTINUED

We can then approximate the cardinality via

$$cardinality = \alpha_m m^2 \left(\sum_{j=1}^{m} 2^{-M[j]} \right)$$

where

$$\alpha_m = \left(m \int_0^\infty \left(\log_2 (\frac{2+u}{1+u}) \right)^m \partial u \right)^{-1}$$

$$\alpha_{16} = .673, \alpha_{32} = .697, \alpha_{64} = .709, \alpha_{m \geq 128} = \frac{.7213}{1 + 1.079/m}$$

HUH? WHAT?

There is large variability in using the rank, so we apply a technique called stochastic smoothing

Lots of other tricks for a robust implementation...

1. Use linear counting when the cardinality is small...
2. When the cardinality is giant, realize that hash collisions happen more often.

IMPLEMENTATIONS OF HYPERLOGLOG

COUNTING ELEMENTS ABOVE A CERTAIN FREQUENCY IN A STREAM [3]

[3] Karp, Richard M., Scott Shenker, and Christos H. Papadimitriou. "A simple algorithm for finding frequent elements in streams and bags." ACM Transactions on Database Systems (TODS) 28.1 (2003): 51-55.

THE SETUP

» An alphabet Σ with $\|\Sigma\| = n$

» A sequence $x = x_1, x_2, \ldots, x_N \in \Sigma^*$

» A threshold θ satisfying $0 < \theta < 1$ (frequency)

» Assume $N \gg n \gg \frac{1}{\theta}$

» $f_x(a)$ the number of times a appears in sequence x

» $I(x, \theta) = \{a \in \Sigma : f_x(a) > \theta N\}$

EXAMPLES

» $x = (1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 3, 4, 5, 6), \quad N = 15$

$$
\begin{aligned}
I(x, .5) &= \{a \in \Sigma : f_x(a) > .5 * 15\} \\
&= \{1\} \\
I(x, .1) &= \{a \in \Sigma : f_x(a) > .1 * 15\} \\
&= \{1, 2\}
\end{aligned}
$$

OBSERVATIONS

» Determining $I(x, \theta)$ exactly requires $\Omega(n)$ memory

SO WHAT DO WE DO?

» Compute a superset, K, of $I(x, \theta)$ using $O\left(\frac{1}{\theta}\right)$ space

» Remember, $n \gg \frac{1}{\theta}$

A TRICK WHEN $\theta = .5$

» Pick two unique symbols, remove them from list

» Repeat

» The symbol remaining is the majority element

» (1,1,1,2,3)

  » Remove (1,2) $\longrightarrow$ (1,1,3)

  » Remove (1,3) $\longrightarrow$ (1,1)

  » 1 is majority element

THE PSEUDOCODED ALGORITHM

```
x[1]...x[N] is the input sequence
K is a set of symbols initially empty
count is an array of integers indexed by K
for i:= 1,...,N do
  if x[i] is in K then count[x[i]] := count[x[i]] + 1
  else insert x[i] in K, set count[x[i]] := 1
  if |K| > 1/theta then
    for all a in K do
      count[a] := count[a] - 1,
      if count[a] = 0 then delete a from K
output K
```

A SCALA IMPLEMENTATION

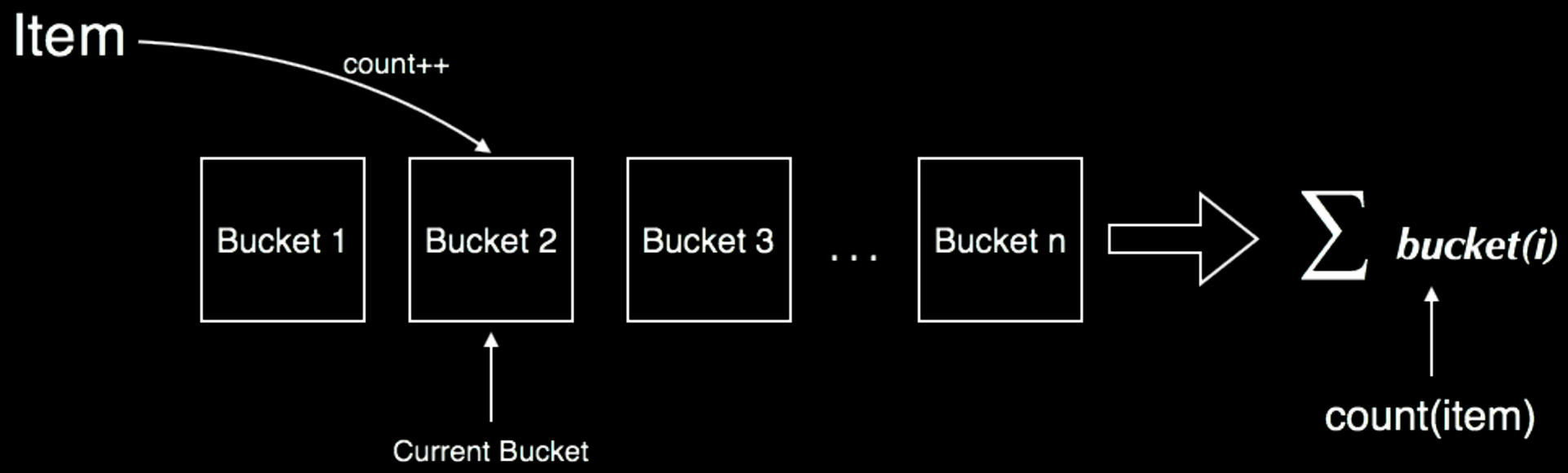
COUNTING A SMALL NUMBER OF THINGS OVER A ROLLING WINDOW

OBSERVATIONS

» Few number of distinct elements

» Must handle concurrency

» Nothing fancy, just be careful


Item
count++
Bucket 1
Bucket 2
Bucket 3
. . .
Bucket n
Current Bucket
∑ bucket(i)
count(item)

LET'S LOOK AT SOME CODE

COUNTING AN UNKNOWN NUMBER OF THINGS IN A STREAM

COUNTMIN SKETCH [GMCMS]

[GMCMS] Cormode, Graham, and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications." Journal of Algorithms 55.1 (2005): 58-75. APA

(SIMPLIFIED) PROBLEM STATEMENT

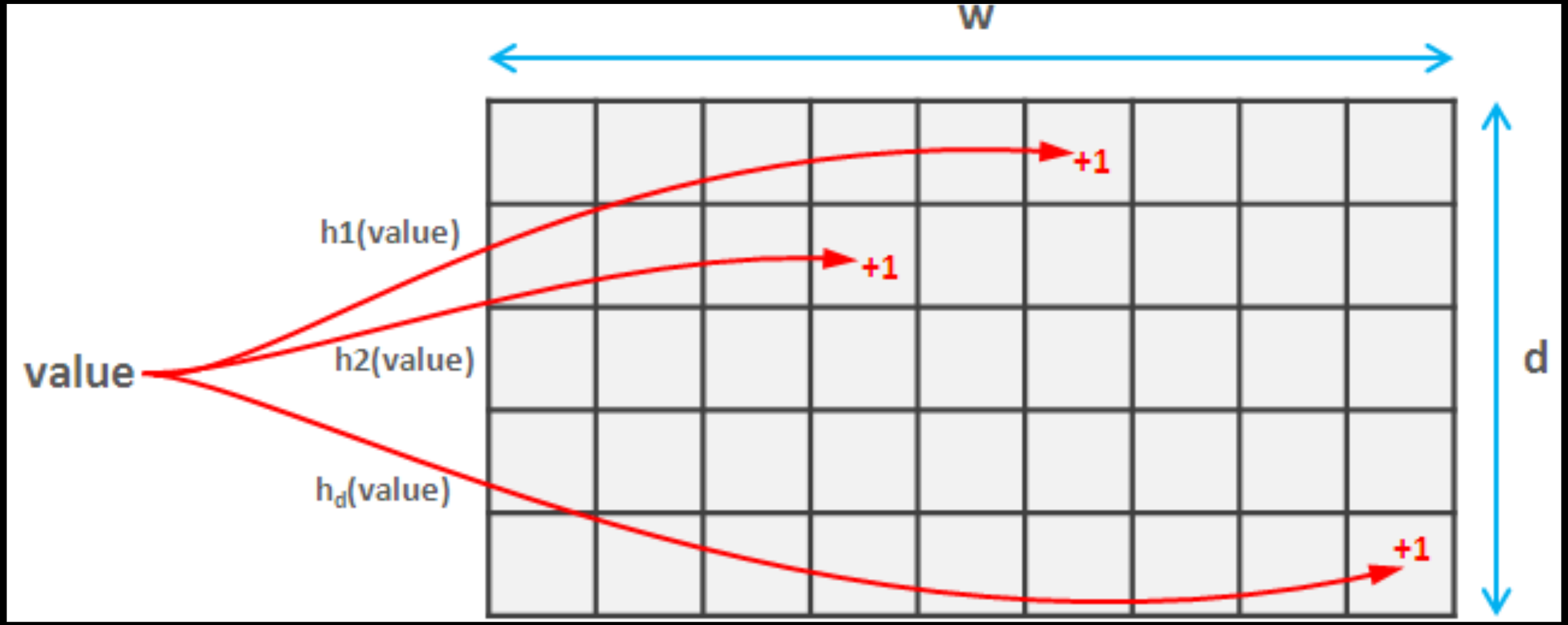
1. Given a stream of data, $a_1, a_2, \ldots a_n$
2. Answer queries of the form $count(k)$

OUR UNIVERSAL HASH FAMILY

$$h(i) = (a * i + b \bmod p) \bmod m$$

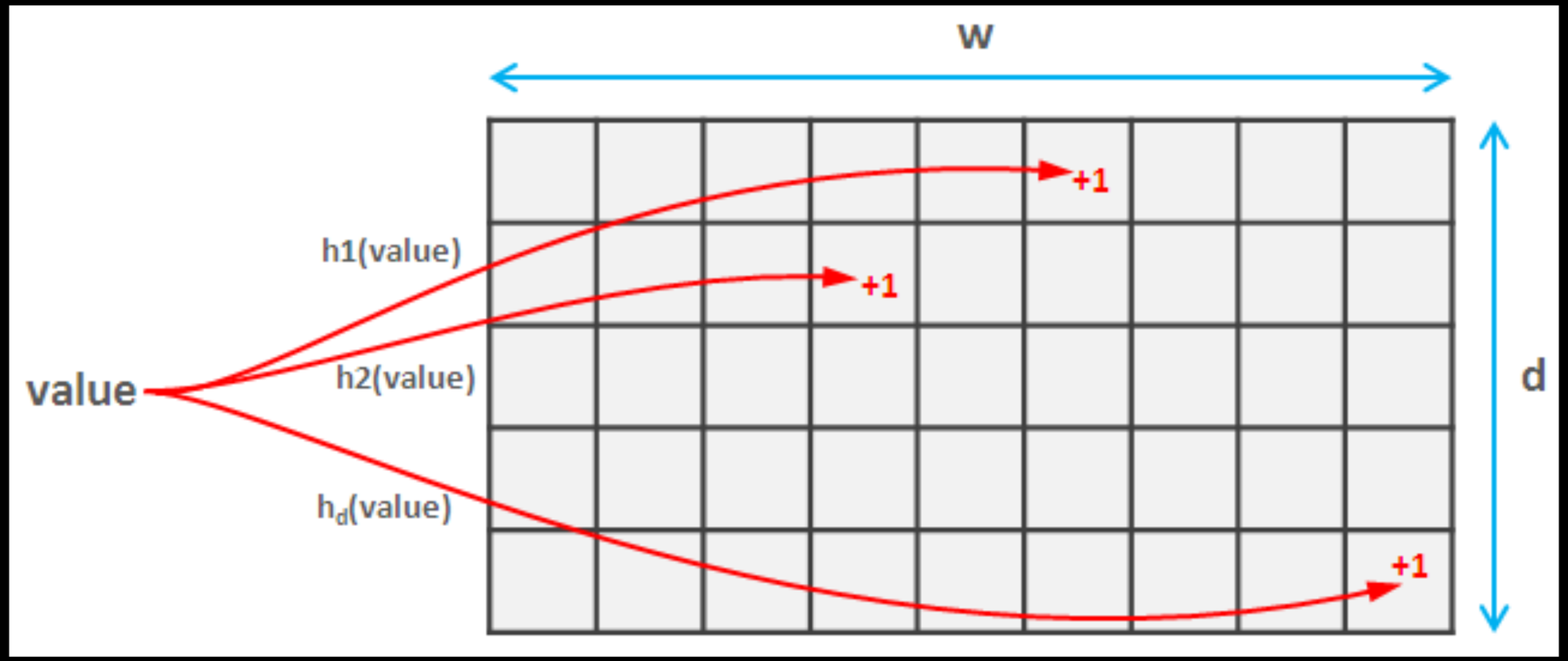
THE SETUP

1. Depth d (corresponding to d hash functions)
2. Width w (Each w is a counter)
3. $CMS[i, j]$ is the count of counter j for hash function i



http://highlyscalable.files.wordpress.com/2012/04/count-min-sketch.png

ESTIMATE THE FREQUENCY



$$count(k) \approx \min_{j=1,2,\ldots d} CMS[j, h_j(k)]$$

AND STRINGS?

Use the same hashing trick as above

LETS LOOK AT SOME CODE

SAVING SPACE AND SUMMARIZING STREAMS[MAA05]

[MAA05] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In Proceedings of the 10th international conference on Database Theory (ICDT'05) PDF

OVERVIEW

1. Counter based technique
2. Update counters in a way that accurately reflects frequency of significant elements
3. Estimates frequencies of significant elements and stores them in an always sorted structure
4. Great algorithms/engineering with some strong math

THE SETUP

1. An alphabet, A
2. A stream S of size N
3. F_i is the frequency of the element E_i

Goal: Find the k elements with the highest frequency

PERFECT WORLD

1. We have $m \geq k$ counters (data structure to be described later)
2. An element E_i, with rank i is stored in the i'th counter
3. Top k: Output elements at counter $0, 1, 2, \ldots, k-1$

REAL WORLD

1. Order of elements in counter data structure may not reflect actual ranks
2. $count_i$ is counter at the i position, estimates frequency f_i of element e_i
3. If the i'th position of the data structure has the element with rank i, then $e_i = E_i$ and $f_i = F_i$

ALGORITHM IDEA

1. Monitor m elements with m counters
2. If a monitored element is observed, increment corresponding counter
3. If a non monitored element is observed
   - » Let min equal count of least frequent element
   - » Eject the element
   - » Add new element with count as $min + 1$

ALGORITHM WALKTHROUGH

» $k = m = 2\,;\;\; count[i] = \{element, count\}$

» `Stream` $\{1, 1, 3, 2, 3\}$

$$
\begin{aligned}
1 &\rightarrow count[0] = \{1, 1\} \\
1 &\rightarrow count[0] = \{1, 2\} \\
3 &\rightarrow count[0] = \{1, 2\};\; count[1] = \{3, 1\} \\
2 &\rightarrow count[0] = \{1, 2\};\; count[1] = \{2, 2\} \\
3 &\rightarrow count[0] = \{1, 2\};\; count[1] = \{3, 3\}
\end{aligned}
$$

ADDITIONAL INFORMATION

For each monitored element e_i, we store its maximum over-estimation ε_i (the counter value of the evicted element)[Detail]

[Detail] Implementations in stream-lib do not use this fact, Algebird does. Can be used to provide a guarantee of correctness

STREAM SUMMARY DATA STRUCTURE

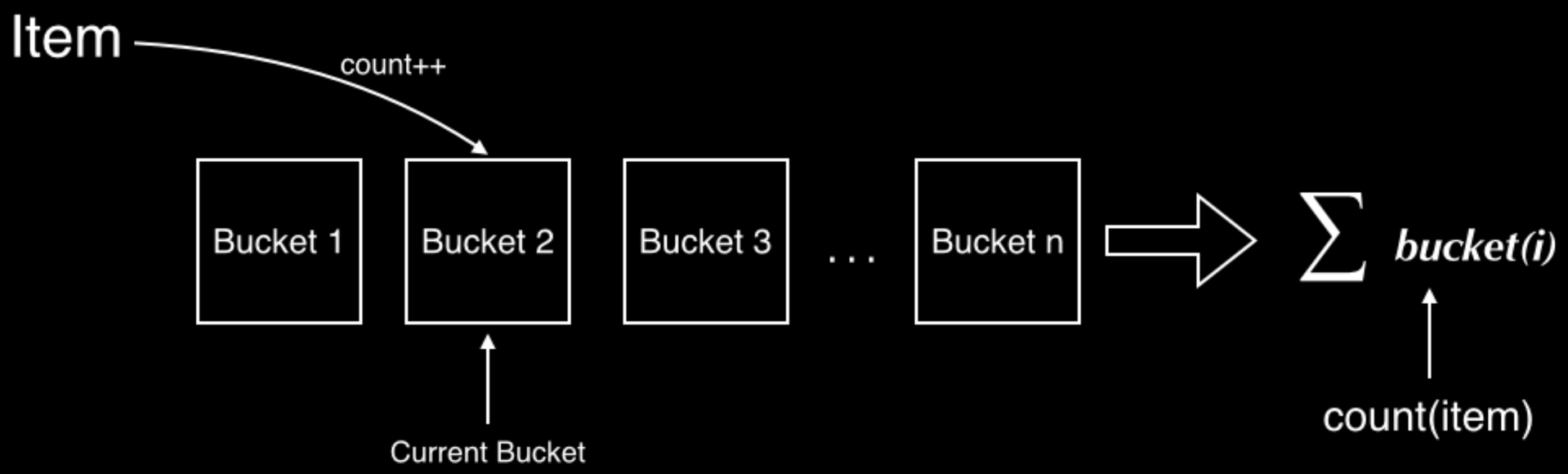
» All elements with same counter value are in linked list, with a pointer to parent bucket

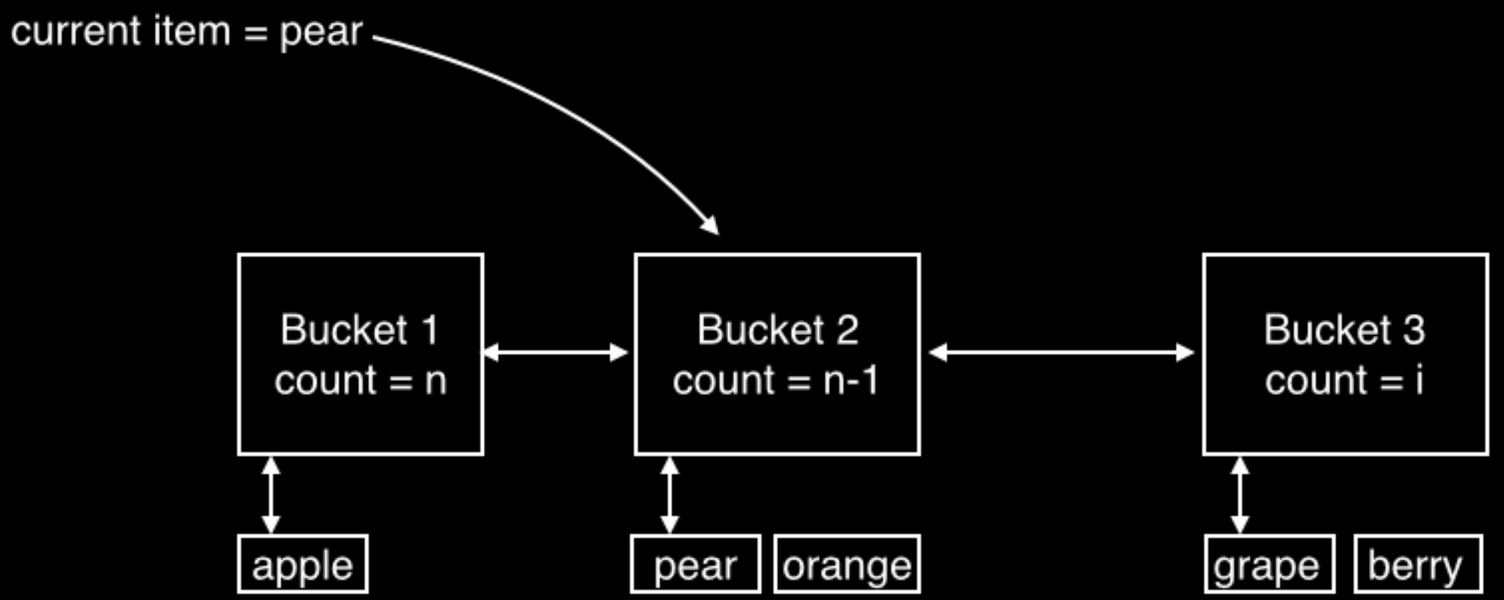
» Parent bucket has counter equal to all values in list

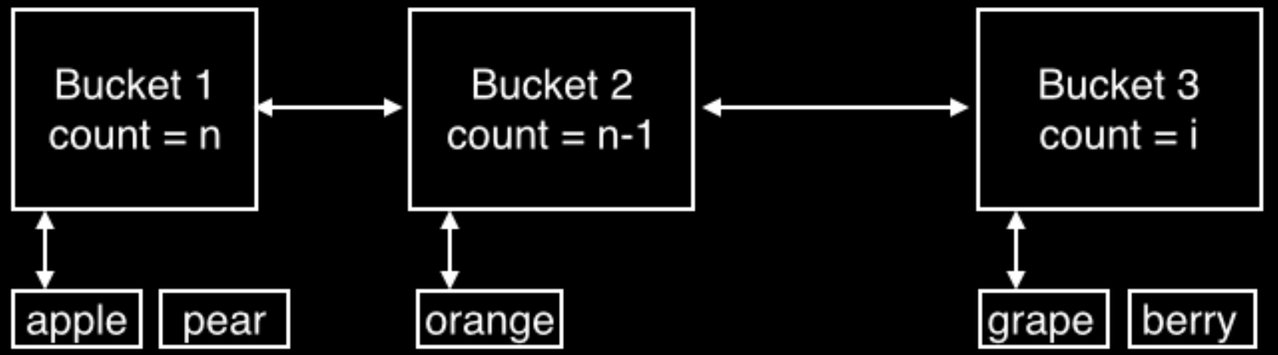
» Each parent bucket points to exactly one child element

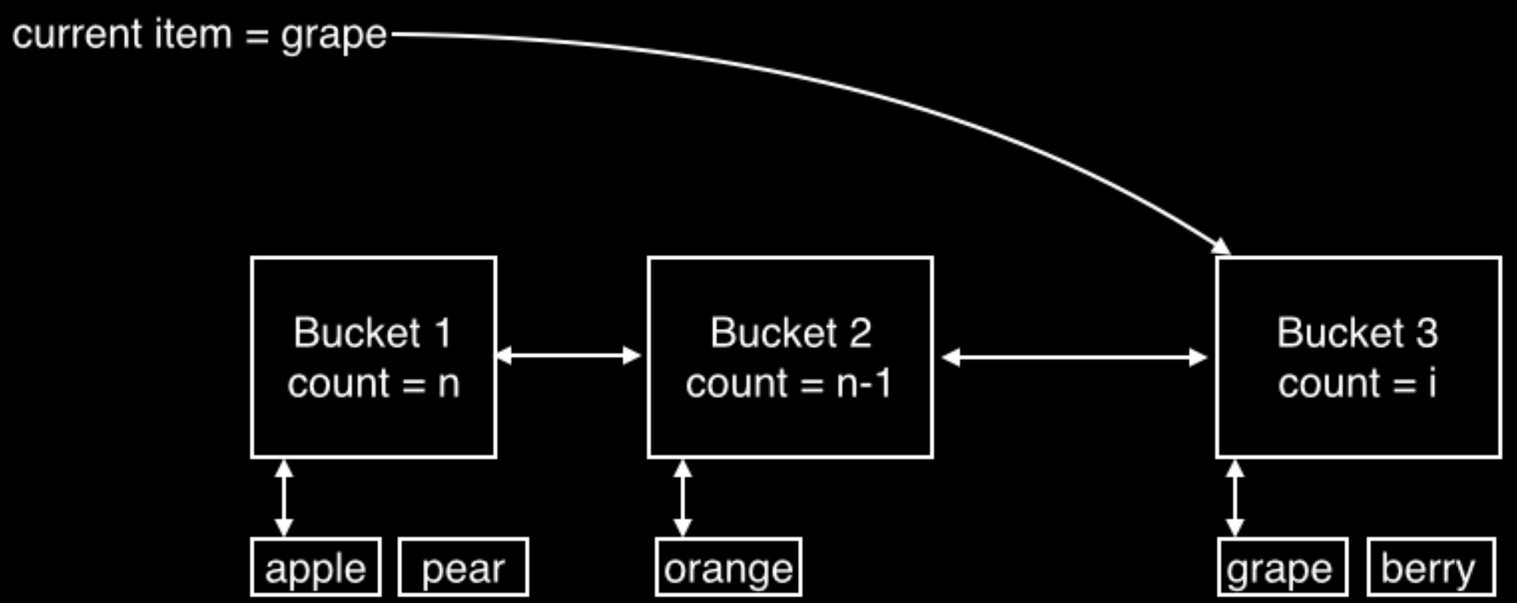
» Parent buckets linked in sorted doubly-linked list

THE ALGORITHM IN PICTURES


Item
count++
Bucket 1
Bucket 2
Bucket 3
. . .
Bucket n
Current Bucket
∑ bucket(i)
count(item)


current item = pear
Bucket 1
count = n
Bucket 2
count = n-1
Bucket 3
count = i
apple
pear
orange
grape
berry


Bucket 1
count = n
Bucket 2
count = n-1
Bucket 3
count = i
apple
pear
orange
grape
berry


current item = grape
Bucket 1
count = n
Bucket 2
count = n-1
Bucket 3
count = i
apple
pear
orange
grape
berry

BUCKET UPDATE ALGORITHM

```
let Bucket[i] be the Bucket of count[i]
let Bucket[i+1] be Bucket[i]'s neighbor of larger value
Detach count[i] from Bucket[i]’s child-list;
count[i]++;
//Finding the right bucket for count[i]
If (Bucket[i+1] does exist AND count[i] = Bucket[i+1])
  Attach count[i] to Bucket[i+1]’s child-list;
else
  //A new bucket has to be created
  Create a new Bucket Bucket[new];
  Assign Bucket[new] the value of count[i];
  Attach count[i] to Bucket[new]’s child-list;
  Insert Bucket[new] between Bucket[i] and Bucket[i+1];

//Cleaning up
If Bucket[i]’s child-list is empty
  Detach Bucket[i] from the Stream-Summary;
  Delete Bucket[i];
```

COMPUTING THE TOP-K

1. IN stream-lib, just output the biggest k elements
2. A little more complicated in algebird
   - » Why does algebird not have a SSZero

SOME PROPERTIES OF THE STREAM SUMMARY

The length, N, of the stream is equal to the sum of all the counters in the Stream Summary data structure

STREAM SUMMARY IN PRACTICE

AREAS NOT COVERED

1. Bloom filters
2. Streaming percentiles/median
3. Distributed counting
4. Other things that I do not know exist

Socrata

WE ARE HIRING