DS 503: Advanced Data Analytics

Lecture 8: Streams

Instructor: Dr. Gagan Raj

More algorithms for streams:

- **More algorithms for streams:**
  - **(1) Filtering a data stream: Bloom filters**
    - Select elements with property **x** from stream
  - **(2) Counting distinct elements: KMV**
    - Number of distinct elements in the last ***k*** elements of the stream
  - **(3) Estimating moments: AMS method**
    - Estimate std. dev. of last ***k*** elements

Filtering Data Streams/Approximate Set Membership

- **Each element of data stream is a tuple**
- Given a list of keys **S**
- **Determine which tuples of stream are in *S***
- **Obvious solution: Hash table**
  - But suppose we **do not have enough memory** to store all of ***S*** in a hash table
  - E.g., we might be processing millions of filters on the same stream

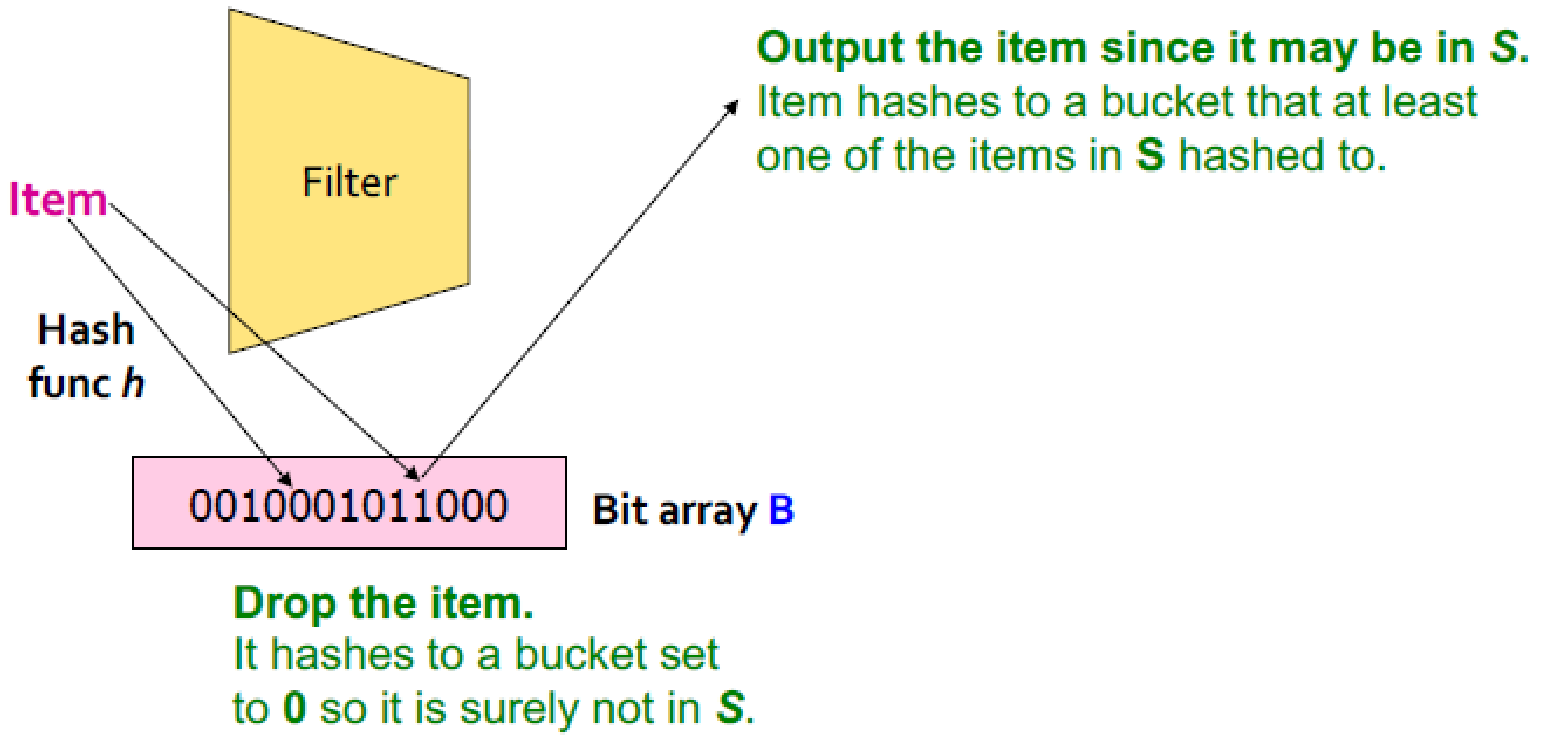
Applications

- Example: Email spam filtering
  - We know 1 billion “good” email addresses
    - Or, each user has a list of trusted addresses
  - If an email comes from one of these, it is **NOT** spam
- Publish-subscribe systems
  - You are collecting lots of messages (news articles)
  - People express interest in certain sets of keywords
  - Determine whether each message matches user’s interest
- Content filtering:
  - You want to make sure the user does not see the same ad multiple times
- Web cache filtering:
  - Has this piece of content been requested before? Then cache it now.

First Cut Solution (1)

- **Given a set of keys *S* that we want to filter**
  - Create a **bit array *B*** of ***n*** bits, initially all ***0*s**
  - Choose a **hash function *h*** with range **[*0,n*)**
  - Hash each member of $s \in S$ to one of ***n*** buckets, and set that bit to **1**, i.e., ***B[h(s)]=1***
  - Hash each element ***a*** of the stream and output only those that hash to bit that was set to **1**
    - **Output *a* if B[h(a)] == 1**

First Cut Solution (2)


Item
Hash func *h*
Filter
0010001011000
Bit array B
Output the item since it may be in *S*.
Item hashes to a bucket that at least one of the items in **S** hashed to.
Drop the item.
It hashes to a bucket set to **0** so it is surely not in ***S***.

- **Creates false positives but no false negatives**
  - If the item is in ***S*** we surely output it, if not we may still output it (depends on the hash function collisions)

First Cut Solution (3)

- **|S| = 1 billion email addresses**
- **|B|= 1GB = 8 billion bits**
- If the email address is in ***S***, then it surely hashes to a bucket that has the bit set to **1**, so it always gets through (***no false negatives***)
- Approximately **1/8** of the bits are set to **1**, so about **1/8th** of the addresses not in ***S*** get through to the output (***false positives***)
- Actually, less than **1/8th**, because more than one address might hash to the same bit

Analysis: Throwing Balls into Bins

- **More accurate analysis for the number of false positives**
- **Consider:** If we throw ***m*** balls into ***n*** equally likely bins, **what is the probability that a bin gets at least one ball?**
- **In our case:**
  - **Bins** = bits/buckets, **Balls** = hash values of items
- P[A given bin is empty] = $\left(1-\frac{1}{n}\right)^{m}$
- P[A given bin has at least one ball] = $1-\left(1-\frac{1}{n}\right)^{m} \sim 1-e^{-\frac{m}{n}}$
  - As $n \rightarrow \infty$, $\left(1-\frac{1}{n}\right)^{n} = e^{-1} = \frac{1}{e}$

<u>Example:</u>
M=1 Billion
N= 8 Billion
Fraction of 1s in array B
$= 1 - e^{-\frac{1}{8}} = 0.1175$

Bloom Filter

- Consider: **|S| = *m*, |B| = *n***
- Use ***k*** independent hash functions $h_1, h_2, \dots, h_k$
  - Simple hash functions such as H={h=[(ax+b) mod p] mod n} will work
- **Initialization:**
  - Set **B** to all **0s**
- **Update**:
  - Hash each element $s \in S$ using each hash function h_i
    - Set $B[h_i(s)] = 1$ (for each ***i* = 1,.., *k***)
- **Run-time:**
  - When a stream element with key ***x*** arrives
  - If $B[h_i(x)] = 1$ **for all *i* = 1,..., *k*** then declare that ***x*** is in ***S***
    - That is, ***x*** hashes to a bucket set to **1** for every hash function $h_i(x)$
  - Otherwise discard the element ***x***

Note 1
We have the same array B for all hash functions

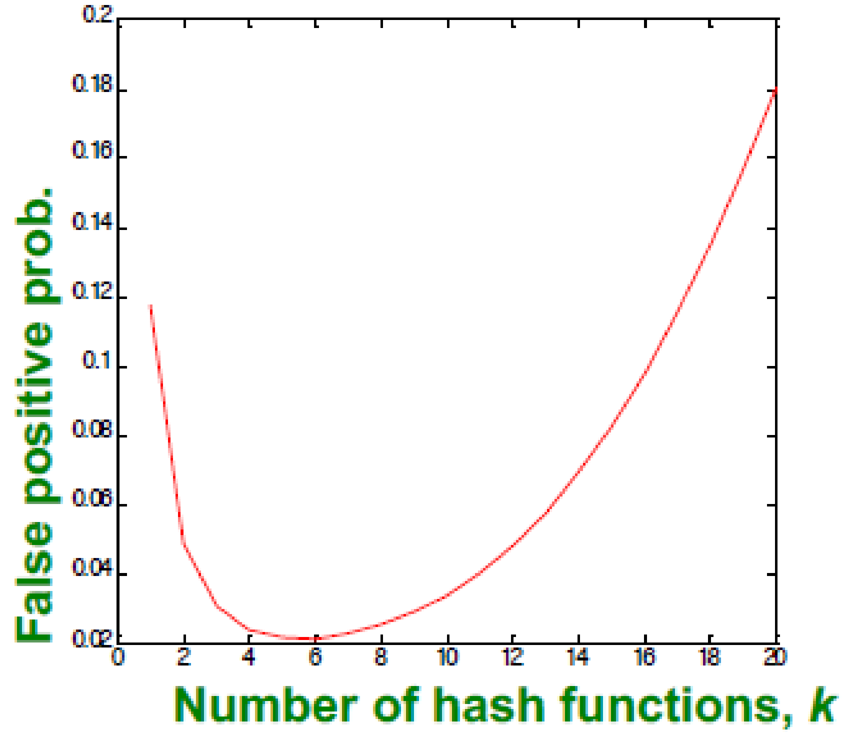
Note 2
Bloom filters support only insertions, but no deletions to the Set

Bloom Filter - Analysis

- **What fraction of the bit vector B are 1s?**
- Throwing ***k·m*** balls into ***n bins***
  - Fraction of **1**s is $1 - e^{-\frac{k.m}{n}}$
- We have ***k*** independent hash functions and we only let the element ***x*** through **if all *k*** hash element ***x*** to buckets of value **1**
- So, **false positive probability =** $\left(1 - e^{-\frac{k.m}{n}}\right)^k$
- Question: Given, m and n, what is the optimal value of k?
  - Differentiate w.r.t k and set derivative to 0.
  - Minima at $k = \frac{n}{m}\ln(2)$

Bloom Filter: Example

- ***m* = 1 billion, *n* = 8 billion**
- **k = 1**: $\left(1 - e^{-\frac{1}{8}}\right)$ = **0.1175**
- **k = 2**: $\left(1 - e^{-\frac{1}{4}}\right)^{2}$ = **0.0493**
- **Optimal k = 8 ln(2) = 5.54 ~ 6**
- **Error at k=6:** $\left(1 - e^{-\frac{3}{4}}\right)^{6} = 0.0216$



Bloom Filter: Summary

- Bloom filters allow for filtering / set membership
- **Bloom filters guarantee no false negatives, and use limited memory**
  - Great for pre-processing before more expensive checks
- **Suitable for hardware implementation**
  - Hash function computations can be parallelized
- Is it better to have **1** big **B** or ***k*** small **B**s?
  - **It is the same:** $\left(1-e^{-\frac{k.m}{n}}\right)^{k}$ vs. $\left(1-e^{-\frac{m}{\left(\frac{n}{k}\right)}}\right)^{k}$
  - But keeping **1 big B** is simpler

Counting Distinct Elements

- **Problem:**
- Data stream consists of a universe of elements chosen from a set of size ***N***
- Maintain a count of the number of distinct elements seen so far
- **Obvious approach:**
  - That is, keep a hash table of all the distinct elements seen so far
- It will take up a lot of storage

- **How many different words are found among the Web pages being crawled at a site?**
  - Unusually low or high numbers could indicate artificial pages (spam?)
- **How many different Web pages does each customer request in a week?**
- **How many distinct products have we sold in the last week?**
- **Number of unique users in the system**
- **Number of users affected by an outage**
- **Number of users using a particular service**

Allow for intersections and unions
How many products/users in both categories or at least one?

Naïve approach to distinct counting

- How many distinct users saw a news clip?
- Database query: select count(distinct user) from very_large_stream
- Execution (Naive):
  - Insert each user into a hash table
  - Compute the number of keys in the table
- Cost:
  - N = 1 million, 64 bits / user
  - Hash table size per website/issue: 16 MB
  - 100 websites/issues/ day ⇒ 1.6 GB / day
  - Over 1 month ⇒ 50 GB / month
  - With dimension breakdowns (user demographics), much higher costs
- Uniform sampling will be misleading, if some user watched it many times.

Using Small storage

- **Real problem:** **What if we do not have space to maintain the set of elements seen so far?**
- **Estimate the count in an unbiased way (E[estimate]=actual)**
- **Accept that the count may have a little error, but limit the probability that the error is large**
- **Several algorithms have been proposed for this problem, all of them try to make use of (one or more) hash functions and design an estimator that is unbiased and has low variance.**

KMV Sketch

Data (sequence of items): $x_1, x_2, x_3, \dots, x_t$

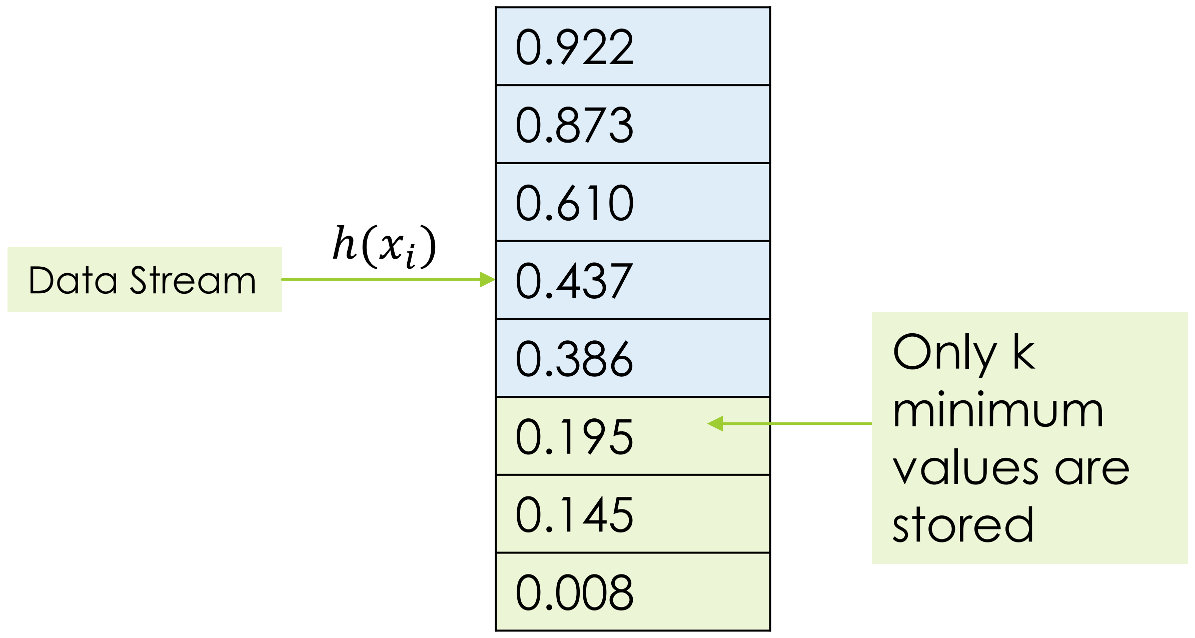
Universal hash function 'h': $x_i \rightarrow Z_i \sim Uniform(0,1)$

- Results in **n** distinct values (eg. 2^{32} or 2^{64})
- Distribution is known and depends only on the desired value **n**

Summary: Maintain k minimum values of h(x_k)
$Z_1 \leq Z_2 \leq Z_3 \dots \leq Z_k$

Estimate: $N_{hat} = \frac{k-1}{Z_k}$

Error: std dev $\sigma(N_{hat}) = \frac{N}{\sqrt{k-2}}$



Properties: It is mergeable, i.e. distributed computations, over time and space are allowed
Operations: Union, Intersection, Set Differences
Example: Multiple dimensions can be implemented as intersections, but errors increase

Intuition

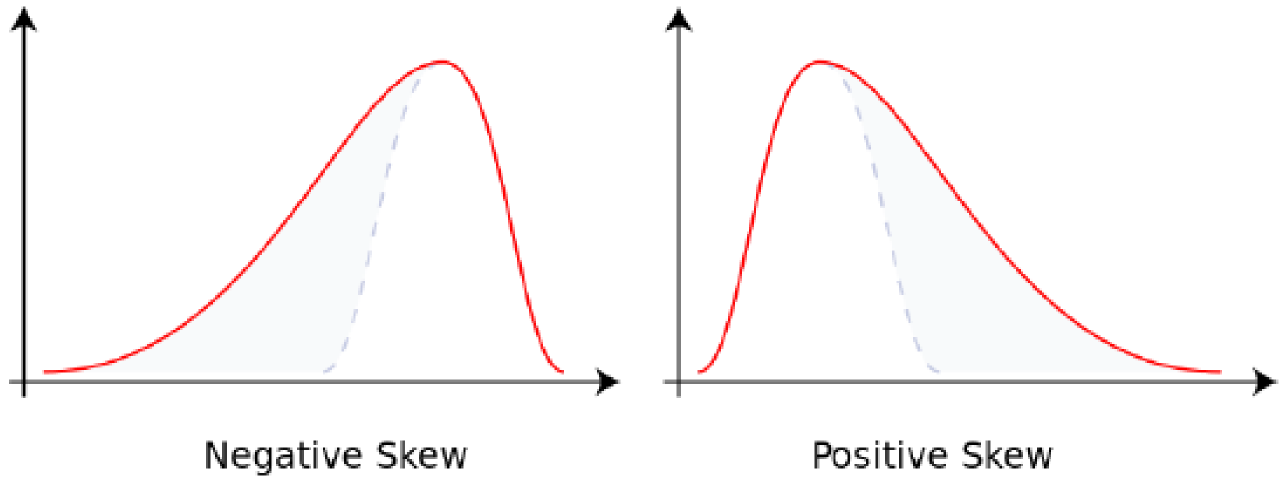
- Hash function maps the element to any of the n values with equal probability
- After d items, the spacing between them is roughly 1/d
- The value of the minimum is close to 1/d
- Thus estimate of d = 1/(min)
- But, there may be a lot of variation in this. We may get a small values, which may disturb the estimate
- Keep many values (k), and the estimate is k/(kth minimum)
    - To make it unbiased, we have to use k-1/(kth minimum)

Problem 3: Computing Moments

- Suppose a stream has elements chosen from a set *A* of *N* values
- Let m_i be the number of times value *i* occurs in the stream
- The k^{th} (frequency) *moment* is : $\sum_{i \in A} (m_i)^k$
  - This is the same way as moments are defined in statistics. But there one typically "centers" the moment by subtracting the mean
- **0th moment =** number of distinct elements
  - The problem just considered
- **1st moment =** count of the numbers of elements = length of the stream
  - Easy to compute, so not particularly useful
- **2nd moment =** ***surprise number S*** **=** a measure of how uneven the distribution is
  - Very useful

Moments

- Third Moment is Skew:



- Fourth Moment: Kurtosis
  - peakedness (width of peak), tail weight, and lack of shoulders (distribution primarily peak and tails, not in between).

Example: Surprise Number

- **Measure of how uneven the distribution is**
- **Stream of length 100 (sum of first moment)**
  - **11 distinct values**
- Item **counts mi**: **10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9**

  Surprise *S* = 910
- Item **counts mi** : **90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1**

  Surprise *S* = 8,110

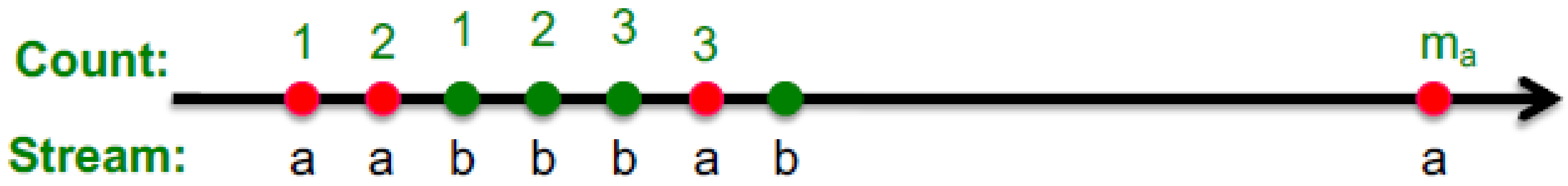
AMS Method

- **AMS method works for all moments**
- **Gives an unbiased estimate**
- **We will just concentrate on the 2nd moment**
  - Can be generalized
- **We pick and keep track of many variables *X*:**
  - For each variable ***X*** we store ***X.el*** and ***X.val***
    - ***X.el*** corresponds to the item ***i***
    - ***X.val*** corresponds to the **count** $\boldsymbol{m_i}$ of item ***i***
- Note this requires a count in main memory, so number of ***X***s is limited
- **Our goal is to compute** $\boldsymbol{S} = \sum_i m_i^2$

One Random Variable (X)

- **How to set *X.val* and *X.el*?**
  - Assume stream has length ***n*** (we relax this later)
  - Pick some random time ***t*** (***t<n***) to start, so that any time is equally likely
  - Let at time ***t*** the stream have item ***i***. ***We set X.el = i***
  - Then we maintain count ***c*** (***X.val = c***) of the number of ***i*s** in the stream starting from the chosen time ***t***
- **Then the estimate of the 2nd moment ($\sum_i m_i^2$) is:**
  - $S = f(X) = n\,(2 \cdot c - 1)$
  - Note, we will keep track of multiple **X**s, $(X_1, X_2, \ldots, X_k)$ and our final estimate will be $S = \frac{1}{k}\sum_j^k f(X_j)$

Expectation Analysis



- **2nd moment is $S = \sum_i m_i^2$**
- c_t … number of times item at time ***t*** appears from time **t** onwards

  $(c_1 = m_a, c_2 = m_a - 1, c_3 = m_b)$

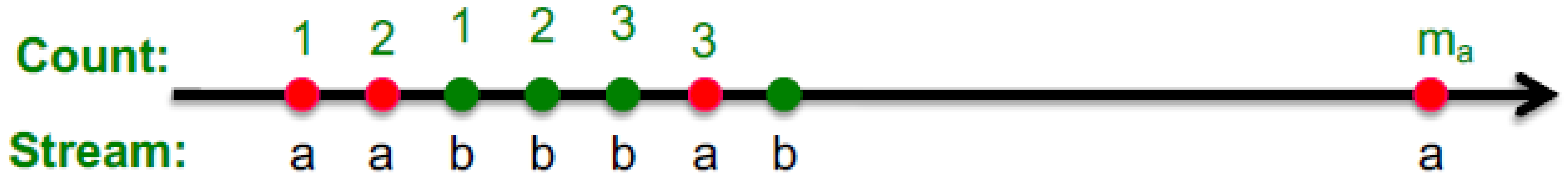
  m_i … total count of item i in the stream (we are assuming stream has length n)
- $E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) = \frac{1}{n}\sum_i n(1 + 3 + 5 + \cdots + 2m_i - 1)$

Group times by value seen

Time t when the last item i is seen ($c_t = 1$)

Time t when the first i is seen ($c_t = m_i$)

Expectation Analysis



- $E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) = \frac{1}{n}\sum_i n(1 + 3 + 5 + \cdots + 2m_i - 1)$
  - $1 + 3 + 5 + \cdots + 2m_i - 1 = \sum_{i=1}^{m_i}(2i - 1) = \frac{2m_i(m_i+1)}{2} - m_i = (m_i)^2$
- Then $E[f(X)] = \frac{1}{n}\sum_i n(m_i)^2$
- So, $E[f(X)] = \sum_i (m_i)^2 = S$
- We have the second moment (in expectation)!

Higher-Order Moments

- **For estimating kth moment we essentially use the same algorithm but change the estimate f(X):**
  - For **k=2** we used ***n* (2·c – 1)**
  - For **k=3** we use: $n(3.c^2 - 3.c + 1)$ (where **c=X.val**)
- **Why?**
  - **For k=2:** (1 + 3 + 5 + ⋯+ $2m_i$ – 1) sum to m_i^2
    - $\sum_{c=1}^{m}(2c-1) = \sum_{c=1}^{m} c^2 - \sum_{c=1}^{m}(c-1)^2 = m^2$
    - **Because:** $2c - 1 = c^2 - (c-1)^2$
  - **For k=3:** $c^3 - (c-1)^3 = 3c^2 - 3c + 1$
- **Generally:** Estimate f(X) = $n(c^k - (c-1)^k)$

Combining Samples

- **In practice:**
  - Compute $f(X) = n(2\,c - 1)$ for as many variables ***X*** as you can fit in memory
  - Average them in groups
  - Take median of averages
- **Problem: Streams never end**
  - We assumed there was a number ***n***, the number of positions in the stream
  - But real streams go on forever, so ***n*** is a variable – the number of inputs seen so far

Never ending streams: Solution

- The variables ***X*** have ***n*** as a factor – keep ***n*** separately; just hold the count in ***X***
- Suppose we can only store ***k*** counts. We must throw some ***X***s out as time goes on:
- **Objective:** Each starting time ***t*** is selected with probability ***k/n***
- **Solution: (fixed-size / reservoir sampling!)**
  - Choose the first ***k*** times for ***k*** variables
  - When the n^{th} element arrives ($n > k$), choose it with probability ***k/n***
  - If you choose it, throw one of the previously stored variables **X** out, with equal probability