More algorithms for streams:

1. Filtering a data stream: Bloom filters
   - Select elements with property \( x \) from stream

2. Counting distinct elements: Flajolet-Martin
   - Number of distinct elements in the last \( k \) elements of the stream

3. Estimating moments: AMS method
   - Estimate std. dev. of last \( k \) elements

4. Counting frequent items
(1) Filtering Data Streams
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
  - 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
Given a set of keys $S$ that we want to filter

- Create a **bit array** $B$ of $n$ bits, initially all 0s
- 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)

- Creates false positives but no false negatives
  - If the item is in $S$ we surely output it, if not we may still output it

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 $S$.

Item

Hash func $h$

0010001011000

Bit array $B$
First Cut Solution (3)

- \( |S| = 1 \text{ billion email addresses} \)
  \( |B| = 1 \text{GB} = 8 \text{ billion bits} \)

- If the email address is in \( S \), then it surely hashes to a bucket that has the big set to 1, so it always gets through (no false negatives)

- Approximately 1/8 of the bits are set to 1, so about 1/8\(^{th}\) of the addresses not in \( S \) get through to the output (false positives)
  - Actually, less than 1/8\(^{th}\), because more than one address might hash to the same bit
More accurate analysis for the number of false positives

Consider: If we throw $m$ darts into $n$ equally likely targets, what is the probability that a target gets at least one dart?

In our case:

- **Targets** = bits/buckets
- **Darts** = hash values of items
Analysis: Throwing Darts (2)

- We have $m$ darts, $n$ targets
- What is the probability that a target gets at least one dart?

$$1 - (1 - 1/n)^n \approx 1 - e^{-m/n}$$

Probability some target $X$ not hit by a dart

Probability at least one dart hits target $X$

Approximation is especially accurate when $n$ is large
Fraction of 1s in the array B = 

= probability of false positive = $1 - e^{-m/n}$

Example: $10^9$ darts, $8 \cdot 10^9$ targets

- Fraction of 1s in B = $1 - e^{-1/8} = 0.1175$
  - Compare with our earlier estimate: $1/8 = 0.125$
Consider: $|S| = m$, $|B| = n$

Use $k$ independent hash functions $h_1, \ldots, h_k$

**Initialization:**
- Set $B$ to all $0$s
- Hash each element $s \in S$ using each hash function $h_i$, set $B[h_i(s)] = 1$ (for each $i = 1, \ldots, k$)

**Run-time:**
- When a stream element with key $x$ arrives
  - If $B[h_i(x)] = 1$ for all $i = 1, \ldots, k$ 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: we have a single array $B$!)
What fraction of the bit vector B are 1s?

- Throwing $k \cdot m$ darts at $n$ targets
- So fraction of 1s is $(1 - e^{-km/n})$

But we have $k$ independent hash functions and we only let the element $x$ through if all $k$ hash element $x$ to a bucket of value 1

So, false positive probability $= (1 - e^{-km/n})^k$
$m = 1$ billion, $n = 8$ billion
- $k = 1$: $(1 - e^{-1/8}) = 0.1175$
- $k = 2$: $(1 - e^{-1/4})^2 = 0.0493$

What happens as we keep increasing $k$?

Optimal value of $k$: $n/m \ln(2)$
- In our case: Optimal $k = 8 \ln(2) = 5.54 \approx 6$
- Error at $k = 6$: $(1 - e^{-1/6})^2 = 0.0235$

Optimal $k$: $k$ which gives the lowest false positive probability
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 Bs?

- It is the same: \((1 - e^{-km/n})^k\) vs. \((1 - e^{-m/(n/k)})^k\)
- But keeping 1 big B is simpler
(2) 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:
Maintain the set of elements seen so far
- That is, keep a hash table of all the distinct elements seen so far
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?
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

Accept that the count may have a little error, but limit the probability that the error is large
Flajolet-Martin Approach

- Pick a hash function $h$ that maps each of the $N$ elements to at least $\log_2 N$ bits

- For each stream element $a$, let $r(a)$ be the number of trailing 0s in $h(a)$
  - $r(a) =$ position of first 1 counting from the right
  - E.g., say $h(a) = 12$, then 12 is 1100 in binary, so $r(a) = 2$

- Record $R =$ the maximum $r(a)$ seen
  - $R = \max_a r(a)$, over all the items $a$ seen so far

- Estimated number of distinct elements $= 2^R$
Why It Works: Intuition

- **Very very rough and heuristic intuition why Flajolet-Martin works:**
  - $h(a)$ hashes $a$ with equal prob. to any of $N$ values
  - Then $h(a)$ is a sequence of $\log_2 N$ bits, where $2^{-r}$ fraction of all $a$s have a tail of $r$ zeros
    - About 50% of $a$s hash to ***0
    - About 25% of $a$s hash to **00
    - So, if we saw the longest tail of $r=2$ (i.e., item hash ending *100) then we have probably seen about 4 distinct items so far
  - So, it takes to hash about $2^r$ items before we see one with zero-suffix of length $r$
Now we show why Flajolet-Martin works

Formally, we will show that probability of finding a tail of $r$ zeros:

- Goes to 1 if $m \gg 2^r$
- Goes to 0 if $m \ll 2^r$

where $m$ is the number of distinct elements seen so far in the stream

Thus, $2^R$ will almost always be around $m!$
Why It Works: More formally

- What is the probability that a given \( h(a) \) ends in at least \( r \) zeros? It is \( 2^{-r} \)
  - \( h(a) \) hashes elements uniformly at random
  - Probability that a random number ends in at least \( r \) zeros is \( 2^{-r} \)
- Then, the probability of NOT seeing a tail of length \( r \) among \( m \) elements:
  \[
  (1 - 2^{-r})^m
  \]
  - Prob. all end in fewer than \( r \) zeros.
  - Prob. that given \( h(a) \) ends in fewer than \( r \) zeros
Note: \((1 - 2^{-r})^m = (1 - 2^{-r})^{2r(m2^{-r})} \approx e^{-m2^{-r}}\)

Prob. of NOT finding a tail of length \(r\) is:

- If \(m \ll 2^r\), then prob. tends to 1
  - \((1 - 2^{-r})^m \approx e^{-m2^{-r}} = 1\) as \(m/2^r \to 0\)
  - So, the probability of finding a tail of length \(r\) tends to 0

- If \(m \gg 2^r\), then prob. tends to 0
  - \((1 - 2^{-r})^m \approx e^{-m2^{-r}} = 0\) as \(m/2^r \to \infty\)
  - So, the probability of finding a tail of length \(r\) tends to 1

Thus, \(2^R\) will almost always be around \(m!\)
Why It Doesn’t Work

- $E[2^R]$ is actually infinite
  - Probability halves when $R \rightarrow R+1$, but value doubles
- Workaround involves using many hash functions $h_i$ and getting many samples of $R_i$
- How are samples $R_i$ combined?
  - Average? What if one very large value $2^{R_i}$?
  - Median? All estimates are a power of 2
- Solution:
  - Partition your samples into small groups
  - Take the median of groups
  - Then take the average of the medians
(3) Computing Moments
Generalization: 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 $k^{\text{th}}$ moment is

\[
\sum_{i \in A} (m_i)^k
\]

This is the same way as moments are defined in statistics. But there we many times “center” the moment by subtracting the mean.
Special Cases

\[
\sum_{i \in A} (m_i)^k
\]

- **0\textsuperscript{th} moment** = number of distinct elements
  - The problem just considered
- **1\textsuperscript{st} moment** = count of the numbers of elements = length of the stream
  - Easy to compute
- **2\textsuperscript{nd} moment** = surprise number \( S \) = a measure of how uneven the distribution is
Moments

- Third Moment is Skew:

  ![Graph showing Negative and Positive Skew](image)

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

- **Stream of length 100**
- **11 distinct values**

**Item counts:** 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9  
Surprise $S = 910$

**Item counts:** 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 2\textsuperscript{nd} moment $S$
- 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 $m_i$ of item $i$
  - Note this requires a count in main memory, so number of $X$s is limited
- Our goal is to compute $S = \sum_i m_i^2$
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 2$^{nd}$ moment ($\sum_i m_i^2$) is:

$$S = f(X) = n (2 \cdot c - 1)$$

- Note, we will keep track of multiple Xs, $(X_1, X_2, \ldots X_k)$ and our final estimate will be $S = 1/k \sum_{j=1}^{k} f(X_j)$
2\textsuperscript{nd} moment is $S = \sum_i m_i^2$

$c_t$ ... number of times item at time $t$ appears from time $t$ onwards ($c_1=m_a$, $c_2=m_a-1$, $c_3=m_b$)

$E[f(X)] = \frac{1}{n} \sum_{t=1}^{n} n(2c_t - 1)$

$= \frac{1}{n} \sum_i n \left(1 + 3 + 5 + \cdots + 2m_i - 1\right)$

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

- Little side calculation: \( (1 + 3 + 5 + \cdots + 2m_i - 1) = \sum_{i=1}^{m_i} (2i - 1) = 2 \frac{m_i(m_i+1)}{2} - m_i = (m_i)^2 \)

- 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)!
For estimating $k^{th}$ moment we essentially use the same algorithm but change the estimate:

- For $k=2$ we used $n \ (2 \cdot c - 1)$
- For $k=3$ we use: $n \ (3 \cdot c^2 - 3c + 1)$ (where $c=X.val$)

**Why?**

- For $k=2$: Remember we had $(1 + 3 + 5 + \cdots + 2m_i - 1)$ and we showed terms $2c-1$ (for $c=1,...,m$) sum to $m^2$
  - $\sum_{c=1}^{m} 2c - 1 = \sum_{c=1}^{m} c^2 - \sum_{c=1}^{m} (c - 1)^2 = m^2$
  - So: $2c - 1 = c^2 - (c - 1)^2$
- For $k=3$: $c^3 - (c-1)^3 = 3c^2 - 3c + 1$

**Generally:** Estimate $= n \ (c^k - (c - 1)^k)$
In practice:

- Compute $f(X) = n(2c - 1)$ 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
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 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.
Counting Itemsets
New Problem: Given a stream, which items appear more than \( s \) times in the window?

Possible solution: Think of the stream of baskets as one binary stream per item

- \( 1 \) = item present; \( 0 \) = not present
- Use DGIM to estimate counts of \( 1 \)s for all items

At least 1 of size 16. Partially beyond window.
Extensions

- In principle, you could count frequent pairs or even larger sets the same way
  - One stream per itemset

- Drawbacks:
  - Only approximate
  - Number of itemsets is way too big
Exponentially Decaying Windows

- Exponentially decaying windows: A heuristic for selecting likely frequent item(sets)
  - What are “currently” most popular movies?
    - Instead of computing the raw count in last $N$ elements
    - Compute a smooth aggregation over the whole stream
  - If stream is $a_1, a_2, ...$ and we are taking the sum of the stream, take the answer at time $t$ to be:
    \[
    = \sum_{i=1}^{t} a_i (1 - c)^{t-i}
    \]
    - $c$ is a constant, presumably tiny, like $10^{-6}$ or $10^{-9}$
  - When new $a_{t+1}$ arrives:
    Multiply current sum by $(1-c)$ and add $a_{t+1}$
Example: Counting Items

- If each $a_i$ is an “item” we can compute the **characteristic function** of each possible item $x$ as an Exponentially Decaying Window
  - That is: $\sum_{i=1}^{t} \delta_i \cdot (1 - c)^{t-i}$
  - where $\delta_i = 1$ if $a_i = x$, and 0 otherwise
  - Imagine that for each item $x$ we have a binary stream (1 if $x$ appears, 0 if $x$ does not appear)
  - **New item $x$ arrives:**
    - Multiply all counts by $(1-c)$
    - Add +1 to count for element $x$
  - Call this sum the “weight” of item $x$
Important property: Sum over all weights
\[ \sum_t (1 - c)^t \] is \[ 1/[1 - (1 - c)] = 1/c \]
What are “currently” most popular movies?

Suppose we want to find movies of weight $> \frac{1}{2}$

- Important property: Sum over all weights $\sum_t (1 - c)^t$ is $1/[1 - (1 - c)] = 1/c$

Thus:

- There cannot be more than $2/c$ movies with weight of $\frac{1}{2}$ or more

So, $2/c$ is a limit on the number of movies being counted at any time
Count (some) itemsets in an E.D.W.

- What are currently “hot” itemsets?
  - Problem: Too many itemsets to keep counts of all of them in memory

When a basket B comes in:

- Multiply all counts by $(1-c)$
- For uncounted items in B, create new count
- Add 1 to count of any item in B and to any itemset contained in B that is already being counted
- Drop counts $< \frac{1}{2}$
- Initiate new counts (next slide)
Start a count for an itemset $S \subseteq B$ if every proper subset of $S$ had a count prior to arrival of basket $B$

- **Intuitively**: If all subsets of $S$ are being counted this means they are “frequent/hot” and thus $S$ has a potential to be “hot”

- **Example**:
  - Start counting $S=\{i, j\}$ iff both $i$ and $j$ were counted prior to seeing $B$
  - Start counting $S=\{i, j, k\}$ iff $\{i, j\}$, $\{i, k\}$, and $\{j, k\}$ were all counted prior to seeing $B$
Counts for single items < (2/c) \cdot (\text{avg. number of items in a basket})

Counts for larger itemsets = ??

But we are conservative about starting counts of large sets

- If we counted every set we saw, one basket of 20 items would initiate 1M counts