Variants of the Streaming Model:

Notation: number of stream updates (aka stream length): m
- data universe size: n
- stream itself: \( \sigma \)

- Vanilla stream model aka insert-only, unit-weight updates:
  \( \sigma = \langle a_1, a_2, \ldots a_m \rangle \), each \( a_i \in \{1, \ldots n\} \)

- Cash register model aka insert-only model:
  \( \sigma = \langle (a_1, \delta_1), \ldots, (a_m, \delta_m) \rangle \), each \( a_i \in \{1, \ldots n\} \), each \( \delta_i \in \mathbb{Z}_+ \)
  (interpreted as \( \delta_i \) copies of \( a_i \))

Frequencies and frequency vector:
- Let \( f_j = \sum_{i: a_i = j} \delta_i \) (frequency of item \( j \))
- Let \( f = (f_1, \ldots, f_n) \) (frequency vector of \( \sigma \))
- Let \( M = \sum \delta_i \) be the L1-norm of the frequency vector

- Turnstile model: same as cash register, but \( \delta_i \in \mathbb{Z} \)
- Strict turnstile model: same as turnstile model, but promised that at end of stream, \( \forall j \), \( f_j \geq 0 \)

---

**Information Guarantee:**

\( \exists \delta \) we want to compute.
Determine Algorithms For Approximating Frequency in Insert-Only streams.

Goal: Compute a summary of so that, for any \( j \in \mathcal{E} \), one can use the summary to output an estimate \( \hat{f}_j \) of \( f_j \) such that 
\[ |\hat{f}_j - f_j| \leq \varepsilon M. \]

A question of the form "What is \( f_j \)" is also called a pointque.

An algorithm for the vanilla stream model (where \( M = M \)).

Mina-Evils, 1982:

* Algorithm maintains \( K \approx \frac{1}{\varepsilon} \) counters, Initially unassigned.

* Each counter is assigned a universe item \( j \in \mathcal{E} \) that's appeared in the stream.

* On stream update, the algorithm looks to see if \( a_i \) is assigned a counter.

  * If so, the count is incremented.

  * If not, and there are any unassigned counters, \( a_i \) is assigned a counter, and the count is incremented to 1.

  * Otherwise, all counters are decremented and any counters that become 0 are marked unassigned.

* When asked to provide an estimate \( \hat{f}_j \) for \( f_j \), the algorithm returns 0 if \( j \) is unassigned a counter, and returns the value of the counter otherwise.

* Claim: For each \( j \in \mathcal{E} \), \[ 0 \leq f_j - \hat{f}_j \leq \frac{M}{K} \varepsilon M. \]
Proof: It is obvious that $0 \leq f_i - f_j$ for all $i$. Let us prove that $f_i - f_j \leq \frac{m}{(k+1)}$.

Claim: The total number of decrement operations over the course of the algorithm is at most $\frac{m}{(k+1)}$.

Proof: If there are $d$ decrement operations then since each decrement operation affects all $k$ counters, the sum of the counter values at the end of the algorithm is $m - d(k+1)$. Since no counter value is ever negative, it must be that $m - d(k+1) > d \geq \frac{m}{(k+1)}$.

Remarks: A space usage of algorithm 3 is $O(k \cdot \log(k \cdot m))$ bits, $O(\frac{\log(k \cdot m)}{\epsilon})$.

This is optimal up to a factor of $\log(k \cdot m)$.

(For any set $S$ of size $\frac{1}{\epsilon}$, there is a stream for which $f_j \geq c \cdot m \cdot \log(k \cdot m)$ and $|S| = \frac{1}{\epsilon}$. Hence, any sketch achieving our goal can identify an arbitrary subset $S$ of $[m]$ s.t. $|S| = \frac{\log(k \cdot m)}{\epsilon}$.

This "clearly" takes $\log \left( \left( \frac{\log(k \cdot m)}{\epsilon} \right)^{\epsilon} \right) \approx \log(c \cdot m \cdot \log(k \cdot m)) = O(\frac{\log(\log(m))}{\epsilon})$ bit.

\[
\frac{m \cdot \log(k \cdot m)}{k} \leq \left( \frac{m}{k} \right) \leq m
\]

The error guarantee $D \leq f_c - f_i \leq \frac{m}{k+1}$ can be improved to $0 \leq f_c - f_i \leq F_{i,\text{result}}$ for $F_i = \sum_{j} A_{j,k}$ the sum of frequencies of all but $\log k$ most frequent items for any $i$. This will be on the homework.
Space Saving is same as Misra-Gries except if a_i is not assigned a counter, the smallest counter is reassigned to a_i and incremented. Its estimates are always over estimates, not underestimates.

These were treated as distinct algorithms for years, but actually they are isomorphic in the sense that given a "MG-sketch" with \( k \) counters one can compute estimates that would be returned by the space-saving algorithm when run with \( k+1 \) counters, and vice versa.

**Homework**

Implementations of Misra-Gries:

- Store all assigned counters in a hash table.
  - Keys are assigned items
  - Values are counts
- Assume all hash table operations (lookup, insert, delete) take \( O(1) \) time (amortized)
- Assume enumerating all key-value pairs can be done mod 1

Then every stream update can be processed in \( O(1) \) amortized time, since each stream update requires \( k \) \( O(1) \) time.

1. A hash table (lookup and delete)
2. Either changing a key's value or inserting a new (unassigned) counter with value 1 or decrement operation (requires enumerating all \( k \) \( (key,value) \) pairs and deleting up to \( k \) of them)

\( O(k) \) time, and it only happens a total of at most \( k \) times.
What about weighted stream updates?

Possibilities: Process an update \((a_i, \delta_i)\) as \(\delta_i\) unit-weight update

- Berinde et al., 2010: When processing \((a_i, \delta_i)\):
  - if \(a_i\) is assigned a counter increment \(i\) by \(\delta_i\)
  - if not and there are unassigned counters assign \(i\) to
    get its value to \(\delta_i\)
  - otherwise decrement all counters by the minimum
    counter value. Easy to show \(0 \leq \delta_i - \delta_{\min} \leq \frac{m}{k+1}\)

Shortcoming: Doesn't run in \(O(1)\) amortized time per stream update. Why?

Proposal (upcoming work): decrement by the median counter value. Then the does run in \(O(1)\) amortized time per stream update. Can also prove

\[0 \leq \delta_i - \delta_{\min} \leq \frac{2M}{k+1}\]

Really only approximate medians are necessary;
we will see easy sampling-based algorithm for computing approximate medians in a few lectures.

If time, talk about heavy hitters, itemsets.