Streaming Algorithms

Stream : \( A = <a_1,a_2,...,a_m> \)
\( a_i \in [n] \) size \( \log n \)
Compute \( f(A) \) in \( \text{poly}(\log m, \log n) \) space

Let \( f_j = |\{a_i \in A \mid a_i = j\}| \)

\( F_1 = \sum_j f_j = m \) == total count

Goal:

\( \epsilon\)-RELATIVE-RANK: Build data structure \( S \).
\( \text{rank}(v) = 1 + \# \text{ items in } A \text{ smaller than } v \)
\( \text{relative-rank}(v) = \text{Rrank}(v) = \text{rank}(v)/|A| \) in \([0,1]\)

\( \epsilon\)-RELATIVE-RANK \( S \) returns \( S(v) \) such that
\( \text{Rrank}(v) - \epsilon < S(v)/m < \text{Rrank}(v) + \epsilon \)
correct wp \( 1-\delta \)

Why relative rank?
\( \text{median}(A) = \{* \mid \text{Rrank}(*) = 1/2\} \)
quantiles: \( \text{Rrank}(*) = 1/4, 3/4 \)
- more "robust" than mean
- captures distribution

Warm Up:
Random Sampling | Reservoir Sampling \( S \) w/ \( k = O(1/\epsilon^2 \log (1/\delta)) \)
For any interval \( I_v = (-\infty, v) \) --> \( \text{Rrank}(v) = \text{count}(I)/m \)
Estimate \( \text{count}(I_v) \) w/ \( S(v) \) s.t.
\( \text{Rrank}(v) - \epsilon < S(v)/m < \text{Rrank}(v) + \epsilon \)
correct wp \( 1-\delta \)

Want \( S \) of size \( \sim O(1/\epsilon) \) instead of \( \sim O(1/\epsilon^2) \)

| \( \epsilon \) | \( .1 \) | \( .01 \) | \( .001 \) |
|------------------|------------------|------------------|
| \( 1/\epsilon \)   | 10               | 100              | 1000             |
| \( 1/\epsilon^2 \) | 100              | 10000            | 1000000          |

Greenwald-Kanna Algorithm \([G,K '01]\)
\[ \text{min-rank}(v) = \text{smallest possible rank of } v \text{ (to } S\text{'s knowledge)} \]
\[ \text{max-rank}(v) = \text{largest possible rank of } v \text{ (to } S\text{'s knowledge)} \]
\[ m = \text{size of stream up to know} \]
\[ A = \langle a_1, a_2, \ldots, a_m \mid a_{m+1}, \ldots \rangle \]

Maintain collection of tuples \((v_i, g_i, \Delta_i)\)
\[ \text{s.t. } v_i \leq v_{i+1} \]
\[ g_i = \text{min-rank}(v_i) - \text{min-rank}(v_{i-1}) \]
\[ \Delta_i = \text{max-rank}(v_i) - \text{min-rank}(v_i) \]
Maintain \(\text{MIN} = \min\{a_j \in A\}\) (seen so far)
\[ \text{MAX} = \max\{a_j \in A\}\) (seem so far)
Maintain \(m = \text{total count}\)

##################################

Process: \(a_m\)
Find \(i\) s.t. \(v_i < a_m < v_{i+1}\)
\(a_m\) becomes \(v_{i+1}\) \((\text{don't actually keep indices})\)
\((v_{i+1} = a_m, g_{i+1} = 1, \Delta_i = L 2 \text{ eps m J})\)
\(m \leftarrow m+1\)
Update \{\text{MIN, MAX}\} as needed
\[ \rightarrow \Delta_i = 0 \]
Check if we can compress:
if \((i \text{ s.t. } g_i + g_{i+1} + \Delta_{i+1} \leq L 2 \text{ eps m } - 1)\) then
Remove \((v_i, g_i, \Delta_i)\)
Set \(g_{i+1} \leftarrow g_i + g_{i+1}\)

##################################

Query : \(\text{rank}(v)\)?
Find \(i\) s.t. \(v_i < v < v_{i+1}\)
Return \((\text{min-rank}(v_i) + \text{max-rank}(v_{i+1}))/2\)
\[ = \text{min-rank}(v_i) + 1 + \sum_{j=1}^i g_i \]
\[ = \text{max-rank}(v_{i+1}) = \text{min-rank}(v_{i+1}) + \Delta_{i+1} \]

Error ?
- we know : \(\text{max-rank}(v_{i+1}) - \text{min-rank}(v_i) = g_i + g_{i+1} + \Delta_{i+1}\)
\[ g_i + g_{i+1} + \Delta_{i+1} \leq 1 + L 2 \text{ eps m } - 1 J < 2 \text{ eps m} \]
\[ \text{INDUCTION: Base case true on insertion} \]
\[ \Delta < 2 \text{ eps m, by induction as well...} \]
\[ \text{Insertion does not increase} \]
\[ \text{Compression only allowed if above holds} \]
\[ 2 \text{ eps m } / 2 < \text{ eps m } \rightarrow \text{Rrank(v) within eps.} \]

Space : \(O((1/\text{eps}) \log(\text{eps } \times m) \log n)\)
- each tuple space : \(O(\log n + \log m) \rightarrow O(\log n + \log (\text{eps m}))\)
- how many tuples? \(O((1/\text{eps}) \log (\text{eps m}))\)
INTUITION: <analysis quite complicated>
- Delta starts 2 eps m, but as m grows, this shrinks relative to m.
- g_i starts at 1. If g_i + g_{i+1} < eps m,
  eventually will compress as Delta_{i+1} decreases relative to m.
- So g_i + g_{i+1} > eps m unless g_{i+1} is new
  (second half of stream).
  <this is hard part to show,
  need amortized slack of log (eps m)>
  - if g_i + g_{i+1} > eps m, then g_i > i * eps m /2
    -> max i < 2 / eps.

So O((1/eps) * log (eps m)) * O(log n + log (eps m))
  <second log (eps m) can be absorbed into
  the O((1/eps) log (eps m)) term>
  --> O((1/eps) log(eps m) log n)

-----------------------------
Median Exactly?
1-pass Omega(min{m,n}) space
2-pass ~O(sqrt{n}) space
p-pass ~O(n^{1/p}) space

p = O(log n) passes --> ~O(1) space

2-pass set eps ~ 1/sqrt{n} in GK.
- Space:  O(sqrt{n} * log (m/sqrt{n}) log n)
- Estimate rank of all elements within sqrt{n} on pass 1
  Find max i s.t. Rrank(v_i) < 1/2
  max j s.t. Rrank(v_j) > 1/2
  I_{i,j} = (v_i , v_j) and count(I_{i,j}) = O(sqrt{n})
- Second pass keep all elements in I_{i,j} and count a in A  s.t. a < v_i

In p-pass narrow range and approximate each pass...

A few more passes can get much more accurate!