CS7960 L11 : Streaming | Approximate Rank

Streaming Algorithms

Stream : A = <a1,a2,...,am>
  ai in [n] size log n
Compute f(A) in poly(log m, log n) space

Let f_j = |{a_i in A | a_i = j}|

F_1 = sum_j f_j = m == total count

Goal:

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

eps-RELATIVE-RANK S returns S(v) such that
 Rrank(v) - eps <= S(v) <= Rrank(v) + eps

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

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

Want S of size ~O(1/eps) instead of ~O(1/eps^2)

eps	.1	.01	.001
1/eps	10	100	1000
1/eps^2	100	10000	1000000

Greenwald-Kanna Algorithm [G,K '01]

```
min-rank(v) = smallest possible rank of v (to S's knowledge)
max-rank(v) = largest possible rank of v (to S's knowledge)
m = size of stream up to know
    |  A = <a1, a2, ..., a_m | a_{m+1}, ... >

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

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

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

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


Space : O((1/eps) log(eps * m) log n)
 - each tuple space : O(log n + log m) ->  O(log n + log (eps m))
 - how many tuples?  O((1/eps) log (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!
```