CS7960 L8 : Streaming-Counting Distinct Elements

Streaming Algorithms

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

Flajolet + Martin '85
Alon, Matias, Szegedy '99

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

Goal: F_0 = |{j \in [n] | f_j >= 0}|
 number of distinct elements

zeros(p) = max{i | 2^i divides p}

####################
Init:
Choose random hash h : [n] -> [n]

```
z := 0

Stream:A
 if (zeros(h(ai)) > z) then z := zeros(h(ai))

Output: 2^{z+1/2}
####################
```

Let there be k distinct elements.
 - we don't know answer, but used in analysis

Expect 1/k distinct elements to have zeros(ai) >= log k
Expect no elements to have zeros(ai) >> log k

Let $X_{r,j}$ == indicator random variable for [zeros(h(j)) > r]

$Y_r = \sum_{j \text{ s.t. } a_i=j} X_{r,j}$

Let t = z at end of stream.

Y_r > 0 <==> t >= r
Y_r = 0 <==> t < r

E[X_r,j] = Pr[zeros(h(j)) >= r] = Pr[2^r divides h(j)] = 1/2^r

E[Y_r] = sum_{j s.t. ai=j} E[X_r,j] = k/2^r

```
Var[Y_r]  = sum_{j s.t. ai=j} Var[X_r,j]
(= E[(X_r,j)^2)] - E[X_r,j]^2)
         <= sum_{j s.t. ai=j} E[X^2_r,j]
          = sum_{j s.t. ai=j} E[X_r,j]
          = k/2^r
```

++++++++++++++++++++++++++++

Markov Inequality

X a rv and a>0

Pr[|X| >= a) <= E[|X|]/a

++++++++++++++++++++++++++++

++++++++++++++++++++++++++++

Chebyshev's Inequality:

Y a rv and b>0
Pr[|Y-E[Y]| >= b) <= Var(Y)/b^2
++++++++++++++++++++++++++++

using MI with X = (Y-E[Y])^2 and a=b^2

MI : Pr[Y_r > 0] = Pr[Y_r >= 1] <=
E[Y_r]/1 = k/2^r (E1)
given r < log k then
CI : Pr[Y_r = 0] = Pr[|Y_r - E[Y_r]| >=
k/2^r]
<= Var[Y_r]/(k/2^r)^2
<= 2^r/k
(E2)

Algorithm output: hat{k}
hat{k} = 2^{t+1/2}

Let a == smallest integer s.t. 2^{a+1/2}
>= 3k.
Pr[hat{k} > 3d] = Pr[t >= a] = Pr[Y_a
> 0] <= k/2^a <= sqrt{2}/3 < 1/2

Let b == largest integer s.t. 2^{b+1/2} < k/3.
   Pr[hat{k} <= d/3] = Pr[t <= b] = Pr[Y_{b+1} = 0] <= 2^{b+1}/k <= sqrt{2}/3 < 1/2

(eps=3,delta=1/2)-approximation

Median Trick
 (make delta arbitrary small)

Run s parallel, independent hash functions on the above procedure.
 output: hat{K} = { hat{k}_1, hat{k}_2, ..., hat{k}_s }
let bar{k} = median[hat{K}]

bar{k} > 3k only if s/2 values in hat{K} > 3k.
Each <= 3k wp 1/2   -- all independent

      1/2^{s/2} <= delta   (where we choose delta)

solve for s :
      2^{s/2} >= 1/delta
      s/2 >= log(1/delta)
      s >= 2 log (1/delta)

Similar for lower bound: delta -> delta/2

Using s = 2 log (2/delta), take median bar{k} is an
  (eps=3, delta)-approximation of # distinct elements.

O(log log n) bits to store t
O(log (1/delta)) hash functions
So: O(log(1/delta) * log log n) right?

oops, forgot to store hash function:
O(log n) bits to store hash function
So: O(log(1/delta) * log n)

Better algorithm:

Space: O(log m + (1/eps^2) (log(1/eps) + log log m))

(eps,delta)-approximation

Hashes to smaller number of bins

Takes average to drive eps down