MCMD L7.5 : Streaming | Reservoir Sampling

Streaming Algorithms

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

Goal: randomly sample k elements from stream
O(k*log n + log m) space

Simpler question: randomly sample one element from stream
O(log n + log m) space

O(log n) to store element S
O(log m) to keep count of how many seen so far C

???

wp k/i keep a_i in register, replace old S w/ a_i
[Vitter '85]

Analysis:
What is probability a_m should be kept? k/m -- good.
What is probability a_{m-1} should be kept?
    (k/(m-1)) * (1 - (k/m)(1/k) = (m-1)/m)) = k/m -- good.
      [kept]   [not replaced by a_m]
Inductively, ignoring a_{i+1} ... a_m
  what is probability a_i should be kept to that point? k/i
  Assume a_{i+1} ... a_m kept with correct probability: total (m-i)/k * k/m = (m-i)/m
    a_i in S after processed wp k/i
    not replaced afterwards wp 1-(m-i)/m = i/m
    total (kept) * (not replaced) = (k/i) * (i/m) = k/m -- good.

(eps,delta)-Approximate Counts:

Consider Interval I subset [n]
  count(I) = |{ a_i in A | a_i in I}|

```
Goal:  Data structure S s.t. for query interval
   Pr[ | S(I) - count(I) | > eps * m ] < delta

++++++++++++++++++++++++++++
Chernoff Inequality

Let {X_1, X_2, ..., X_r} be independent RVs
Let Delta_i =  max(X_i) - min(X_i)
Let M = sum_i X_i

Pr[ | M - sum_i E[X_i] | > alpha ] < 2 exp(- 2 alpha^2 / sum_i (Delta_i)^2)

often:  Delta = max_i Delta_i   and  E[X_i] = 0  then:
Pr[ |M| > alpha ] < 2 exp(- 2 alpha^2/ r Delta^2)
++++++++++++++++++++++++++++

Let S be a random sample of size k = O((1/eps^2) log (1/delta))
S(I) = | {S cap I} | * (m/k)

Each s_i in I wp (count(I)/m)
  ->  RV  Y_i = {1 if s_i in I, 0 if s_i !in I}
          E[Y_i] = count(I)/m
  ->  RV  X_i = (Y_i - count(I)/m)/k
          E[X_i] = 0
          Delta < 1/k
M = sum_i X_i  == error on count estimate by S

Pr[ |M| > eps ] < 2 exp(- 2 eps^2 / (k *(1/k^2) ) < delta

Solve for k in eps,delta:
                  2 exp(- 2 eps^2 k) < delta
                  exp(2 eps^2 k) > 2/delta
                  2 eps^2 k > ln(2/delta)
                  k > (1/2) (1/eps^2) ln (2/delta)
                    = O((1/eps^2) log (1/delta)
```