CS7960 L9 : Streaming | Heavy Hitters = Approximate Counts

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}|

MAJORITY: if some f_j > m/2, output j
           else,   output NULL

one-pass requires Omega(min{m,n}) space

Simpler:
FP-MAJORITY: if some f_j > m/2,
output j
else,
output anything

How good w/ O(log m + log n) (one counter c + one location l)?
...

```
############################
c = 0, l = X
for (a_i \in A)
  if (a_i = l) c += 1
  else         c -= 1
  if (c <= 0)  c = 1, l = a_i
return l
############################
```

Analysis: if f_j > m/2, then
  if (l != j) then c decremented at most < m/2 times, but c > m/2
  if (l == j) can be decremented < m/

2, but is incremented > m/2
if f_j < m/2 for all j, then any
answer ok.

----- another view of analysis ------
Let f_j > m/2, and k = m - f_j.
After s steps, let g_s = unseen
elements of index j
               let k_s = unseen
elements != index j
               let c_s = c if l!=j,
and -c if l==j
Claim: g_s > c+k_s
  base case (s=0, or even s=1) easily
true.
  Inductively 4 cases:
   a_i = l = j : (g_s decremented, c
decremented)
   a_i = l != j: (c incremented, k_s
decremented)
   a_i !=l != j: (c decremented, k_s
decremented)
   a_i !=l = j : (k_s decremented,

maybe c incremented)

Since at the end g_s = k_s = 0, then
0 > c + 0, implies c < 0, and
l==j.

FREQUENT: for k, output the set {j :
f_j > m/k}
also hard.

k-FREQUENCY-ESTIMATION: Build data
structure S.
For any j in [n], hat{f}_j = S(j)
s.t.
f_j - m/k <= hat{f}_j <= f_j

aka eps-approximate phi-HEAVY-
HITTERS:
Return all f_j s.t. f_j > phi
Return no f_j s.t. f_j < phi -
eps*m

```
  (any f_j s.t. phi-eps*m < f_j < phi
is ok)


-------------------------

Misra-Gries Algorithm [Misra-Gries
'82]

Solves k-FREQUENCY-ESTIMATION in
O(k(log m + log n)) space.

Let C be array of k counters C[1],
C[2], ..., C[k]
Let L be array of k locations L[1],
L[2], ..., L[k]

############################
Set all C = 0
Set all L = X

for (a_i in A)
  if (a_i in L)  <at index j>
```

```
     C[j] += 1
  else           <a_i !in L>
    if (|L| < k)
      C[j] = 1
      L[j] = a_i
    else
      C[j] -= 1 forall j in [k]
  for (j in [k])
     if (C[j] <= 0) set L[j] = X
#############################
On query q in [n]
  if (q in L {L[j]=q}) return hat{f}
_q = C[j]
  else                     return hat{f}
_q = 0
#############################
```

Analysis

A counter C[j] representing L[j] = q is only incremented if a_i = q hat{f}_q <= f_q

If a counter C[j] representing L[j] = q is decremented,
then k-1 other counters are also decremented.
This happens at most m/k times.
A counter C[j] representing L[j] = q is decremented at most m/k times.

f_q - m/k <= hat{f}_q

How do we get an additive eps-approximate FREQUENCY-ESTIMATION ?
i.e. return hat{f}_q s.t.
|f_q - hat{f}_q| <= eps*m

Set k = 2/eps, return C[j] + (m/k)/2

Space O((1/eps) (log m + log n))

Also:
eps-approximate phi-HEAVY-HITTERS for any phi > m*eps in space O((1/eps) (log m + log n))

Can solve k-FREQUENT optimally in two passes w/ O(k(log n + log m)) space.
Run M-G algorithm w/ k counters.
For each stored location, make second pass and count exactly.