```
MCMD L8.5 : Streaming | Count Min Sketch

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
F_2 = sqrt{sum_j f_j^2}  == RMS count

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

Goal:

eps-FREQUENCY-ESTIMATION:  Build data structure S.
For any j in [n], hat{f}_j = S(j) s.t.
 f_j - eps*F_1 <= hat{f}_j <= f_j        MG
 f_j <= hat{f}_j <= f_j + eps F_1    CMS (today)
 |f_j - hat{f}_j| <= eps F_2         CS (maybe)

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

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

Count-Min Sketch [Cormode + Muthukrishnan '05]

t independent hash functions {h_1, ..., h_t}
each h_i : [n] -> [k]

2-d array of counters:
h_1 -> [C_{1,1}] [C_{1,2}] ... [C_{1,k}]
h_2 -> [C_{2,1}] [C_{2,2}] ... [C_{2,k}]
...    ...                      ...
h_t -> [C_{t,1}] [C_{t,2}] ... [C_{t,k}]

for each a \in A -> increment C_{i,h_i(a)} for i in [t].

hat{f}_a = min_{i in [t]} C_{i,h_i(a)}

Set t = log(1/delta)
Set k = 2/eps
```

```
---------------------------

----
How to implement h : IR -> [m]?
  h_a in H.  Let a be a large real number:  say a = Unif(0,1) * 10000
  Define h_a(x) = floor(m * frac-part(x*a))
        frac-part(15.324) = 0.234
        floor(15.324) = 15
----

Clearly f_a <= hat{f}_a

hat{f}_a <= f_a + W.  What is W?

One hash function h_i.
Adds to W when there is a collision h_i(a) = h_i(j).  wp 1/k

random variable Y_{i,j}
Y_{i,j} = {f_j wp 1/k, 0 wp 1-1/k}
E[Y_{i,j}] = f_j/k

random variable X_i = sum_{j in [n], j!=a} Y_{i,j}
E[X_i] = E[sum_j Y_{i,j}] = sum_j f_j/k = F_1/k = eps * F_1/2

++++++++++++++++++++++
Markov Inequality

X a rv and a>0
Pr[|X| >= a] <= E[|X|]/a
++++++++++++++++++++++

X_i > 0 so |X_i| = X_i
setting a = eps F_1  then
        E[|X|]/a = (eps*F_1 /2)/(eps F_1) = 1/2
Pr[X_i >= eps F_1] <= 1/2

-----

Now for t *independent* hash functions:

Pr[hat{f}_a - f_a >= eps F_1]
        = Pr[min_i X_i >= eps F_1]
        = Pr[forall_{i in [t]} (X_i >= eps F_1)]
        = Prod_{i in [t]} Pr[X_i >= eps F_1)]
       <= 1/2^t
        = delta   (since t = log(1/delta) )
```

Hence:
f_a <= hat{f}_a <= f_a + eps F_1
 - first inequality always holds
 - second inequality holds wp > 1-delta

Space:
each of k*t counters requires log m space
O(k*t*log m)
Store t hash functions: log n each
O((k log m + log n)*t) = O((1/eps) log m + log n) log (1/delta))

----

turnstile model: add or subtract (as long as is there)

Count Sketch:

t independent hash functions {h_1, ..., h_t}
each h_i : [n] -> [k]

t independent secondary hash functions {g_1, ... g_t}
each g_i : [n] -> {-1,+1}

2-d array of counters:
```
h_1 -> [C_{1,1}] [C_{1,2}] ... [C_{1,k}]
h_2 -> [C_{2,1}] [C_{2,2}] ... [C_{2,k}]
...    ...                      ...
h_t -> [C_{t,1}] [C_{t,2}] ... [C_{t,k}]
```

for each a \in A -> adds g_i(a) to C_{i,h_i(a)} for i in [t].

hat{f}_a = median_{i in [t]} C_{i,h_i(a)}

Set t = 2*log(1/delta)
Set k = 4/eps^2

One hash function pair h_i,g_i.
E[hat{f}_a] = g_i(a) f_a

random variable : Y_{i,j} expected error caused by f_j on hat{f}_a

```
Y_{i,j} = {f_j wp 1/2k, -f_j wp 1/2k, 0 wp 1-1/k}

random variable : X_i expected error of hat{f}_a
X_i = sum_j Y_{i,j}
E[X_i] = 0

Y_{i,j} pairwise independent, so
Var[X] = sum_j Var[Y_{i,j}]
Var[Y_{i,j}] = E[Y_{i,j}^2] - E[Y_{i,j}]^2
             = E[Y_{i,j}^2]
             = f_j^2 / k

Var[X_i] = sum_j f_j^2/k <= F_2^2/k.

++++++++++++++++++++++
Chebyshev's Inequality:

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

using b = eps F_2

Pr[|X_i| >= eps F_2] <= (F_2^2/k) / (eps F_2)^2
                      = 1/(k * eps^2) <= 1/4
                        since k = 4/eps^2

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

t *independent* hash function pairs:

Recall:  hat{f}_a = median_i {(f_a + X_i)/g_i(a)}

Pr[|f_a - hat{f}_a| < eps F_2]
         = Pr[median_i X_i > eps F_2]
        <= 2 * Pr[t/2 {i in [t]} (X_i >= eps F_2)]
        <= 2 * Prod_{i in [t/2]} Pr[X_i >= eps F_2)]
        <= 2 * 1/4^{t/2}
        <= delta   (since t = 2*log(1/delta) )


-----
Space:
each of k*t counters requires log m space
O(k*t*log m)
Store t hash function pairs:  log n each
O((k log m + log n)*t)
```

```
       = O((1/eps^2) log m + log n) log (1/delta))


--------------------------------------------
--------------------------------------------
CMS:  eps F_1 error
      space O(((1/eps) log m + log n) log (1/delta))
CS :  eps F_2 error
      space O(((1/eps^2) log m + log n) log (1/delta))

F_2 < F_1 (generally), but 1/eps << 1/eps^2
CMS very practical because of only (1/eps) term.
```