Lecture 7: Heavy Hitters, Misra-Gires Algorithms, and CountMin

09-16-2025 Lecturer: Ali Vakilian | Scribe: Achintya Sunil | Editor: Ali Vakilian

In a previous lecture, we discussed the Boyer-Moore Voting algorithm for efficiently finding a single majority item (i.e., an item with frequency $f_i > m/2$). We will now extend the core "canceling" intuition of this algorithm to solve the more general heavy hitters problem: identifying all items whose frequency is at least m/k.

1 Misra-Gries Algorithm

The premise of this algorithm is simple. Instead of keeping one word of memory and a counter, we keep track of an array of size k instead.

Algorithm 1 Misra Gries (k):

```
1:  let $D$ be an empty array
2:  for each item $e_j$ in the stream do
3:      if $e_j$ is a key in $D$ then
4:          $D[e_j] \leftarrow D[e_j] + 1$
5:      else
6:          if the number of keys in $D$ is less than $k$ then
7:              add $e_j$ to $D$ with value 1
8:          else
9:              for each key $\ell$ in $D$ do
10:                 $D[\ell] \leftarrow D[\ell] - 1$
11:             remove all keys from $D$ whose value is 0
12: return the key-value pairs in $D$
```

The space complexity of this algorithm is $O(k \log m)$, where M is the size of the stream. The following theorem provides a bound on our estimator $\hat{f}_i$.

Theorem 1.1. *For each $i \in [n]$, $f_i - \frac{m}{k+1} \leq \hat{f}_i \leq f_i$. In particular, any item with $f_i > \frac{m}{k}$ will be in D.*

Proof. The algorithm maintains at most k counters. When an item e_j arrives, one of three cases occurs:

(i) If e_j is currently being tracked, its counter is incremented.

(ii) If e_j is not tracked and there are fewer than k counters, a new counter is created for e_j with value 1.

(iii) If e_j is not tracked and all k counters are in use, all k counters are decremented by 1.

Let ℓ be the total number of times Case (iii) (the "decrement-all" step) is triggered. Each time this step occurs, it is caused by the arrival of an item that is not among the k items being tracked. This single new item, along with the k items whose counters are decremented, account for at least $k+1$ items from the stream. Since this happens ℓ times, we can bound the total items involved:

$$\ell \cdot (k+1) \leq m \implies \ell \leq \frac{m}{k+1}.$$

Define the under-count for an item i as $\alpha_i = f_i - \hat{f}_i$. Let's analyze how α_i changes when an item e_j arrives:

(i) **($e_j = i$, counter exists):** f_i and $\hat{f}_i$ both increase by one, so α_i is unchanged.

(ii) **($e_j = i$, no counter, counters full):** f_i increases by one, but $\hat{f}_i$ (currently 0) does not. α_i increases by one. This is a decrement event.

(iii) **($e_j \neq i$, counters full):** f_i is unchanged, but $\hat{f}_i$ (if it exists) is decremented by one. α_i increases by one. This is also a decrement event.

In all cases, the under-count α_i increases by at most 1, and only when one of the ℓ decrement events occurs. The total under-count for any item i is bounded by the total number of decrement events. Therefore, we can combine our bounds:

$$f_i - \hat{f}_i \leq \ell \leq \frac{m}{k+1}$$

□

Note that Misra and Gries [1982] is a purely deterministic algorithm. Surprisingly, we cannot make any changes to significantly improve the space complexity of $O(k \log m)$. This is quite impressive for a purely deterministic algorithm. However, we are still on the lookout for randomized solutions as they have additional properties such as supporting both insertions and deletions. Before that, let's introduce some useful notions on hashing.

2 Sketching-Based Approach: Hashing Ideas

We can introduce hashing for solving the heavy hitters problem using the following idea: Let's say we have k heavy hitters $b_1, b_2, \ldots, b_k$. Suppose we pick a hash function such that $h : [n] \to [ck]$ for some $c > 1$ (the hash function is mapping the heavy hitters to different buckets. Note that each heavy hitter will be mapped to the same bucket, but we can expect collision to occur between between different items). Then we can estimate the frequency of one of our heavy hitters by using a count of the items from one of the buckets. Repeating this idea with independent hashes, we can improve our estimate. This is the premise behind CountMin Sketch [Cormode and Muthukrishnan, 2005].

2.1 CountMin Sketch

To setup CountMin sketch, we need to use the following data structures.

- d pairwise independent hash functions $h_1, \ldots, h_d$; each mapping $[n] \to [w]$
- Store one counter per entry in the table, $C[\ell, s]$, which is keeping the aggregate frequency of items mapped into the bucket s by the hash function ℓ.

Equivalently, this data structure is table of counters with d rows and w columns.

Note that for $\ell \in [d]$ and $s \in [w]$, $C[\ell, s] = \sum_{i:h_\ell(i)=s} f_i$. Since we allow for collisions between the items, for every $\ell \in [d]$, $C[\ell, h_\ell(i)]$ is an overestimate of f_i. If we have d such estimates, the closest one to the true frequency of the item is given by the minimum $C[\ell, h_\ell(i)]$.

Algorithm 2 CountMin Sketch (stream):

```
1: let $h_1, \ldots, h_k$ be pairwise independent hash functions from $[n] \to [w]$
2: for each item $e_i = (i_t, \Delta_t)$ in the stream do
3:     for $\ell = 1$ to $d$ do $C[\ell, h_\ell(i_t)] \leftarrow C[\ell, h_\ell(i_t)] + \Delta_t$
                                                        ▷ Once the stream is over
4: for each $i \in [n]$ do set $\tilde{f}_i = \min_{\ell \in [d]} \; C[\ell, h_\ell(i)]$
```

Theorem 2.1. *Consider strict turnstile streaming (i.e., always $f \geq 0$). Let $d = \Omega(\log \frac{1}{\delta})$ and $w > \frac{2}{\varepsilon}$. Then, for fixed $i \in [n]$, $f_i \leq \tilde{f}_i$ and $\Pr[\tilde{f}_i \geq f_i + \epsilon ||f||_1] \leq \delta$.*

We remark that unlike Misra-Gries, CountMin is overestimating the frequency of items. The items are not stored, they have to be accessed by querying CountMin to find their frequency estimate. It is a more robust algorithm as it handles deletions ($\Delta_t = -1$) too, and the space complexity is $O(\varepsilon^{-1} \log(1/\delta) \log m)$ bits.

Proof. Let's define $Z_\ell = C[\ell, h_\ell(i)]$ the value of the counter in row ℓ that i is being hashed to. Then

$$\mathbb{E}[Z_\ell] = f_i + \sum_{i \neq j} Pr[h_\ell(j) = h_\ell(i)] f_j = f_i + \sum_{j \neq i} \frac{1}{w} f_j$$

This last equality comes from the pairwise independence of h_ℓ (the probability of a collision between two items is $1/w$). Since $\sum_{j \neq i} f_j \leq \|f\|_1$ and choosing $\varepsilon > 2/w$, $\mathbb{E}[Z_\ell] \leq \varepsilon \|f\|_1/2$. Applying Markov's inequality, $\Pr[Z_\ell - f_i \geq \varepsilon \|f\|_1] \leq 1/2$, and since d hash functions are independent,

$$\Pr[\min_{\ell \in [d]} Z_\ell \geq f_i + \varepsilon ||f||_1] \leq \Pr[\bigwedge_{\ell \in [d]} Z_\ell \geq f_i + \varepsilon ||f||_1] \leq 2^{-d} \leq \delta$$

□

If we set $\delta = 1/n^2$, for every $i \in [n]$, $\Pr[\tilde{f}_i > f_i + \varepsilon \|f\|_1] \leq n^{-2}$. Applying a union bound on all n elements, with probability at least $1 - 1/n$, for all $i \in [n]$, $\tilde{f}_i \leq f_i + \varepsilon \|f\|_1$.

2.2 CountMin as a Linear Sketch

To prove that Count-Min is a linear sketch, we must show that the sketch vector it produces can be obtained by multiplying a fixed sketch matrix by the input's frequency vector. Specifically, a hash function h that maps a universe of n items to w buckets corresponds to a $w \times n$ *binary matrix*, let's call it Π. Each column i of this matrix represents an item from the universe. The column contains exactly one non-zero entry, a '1', at the row corresponding to the bucket the item is mapped to. In other words, the entry $\Pi_{j,i} = 1$ if $h(i) = j$, and is 0 otherwise. When this matrix Π is multiplied by the frequency vector f, the result is a vector of size w representing the counters in the hash buckets. The example in Figure 1-(a) illustrates this for a single hash function.

The CountMin sketch uses d independent hash functions, $h_1, \ldots, h_d$, each mapping from $[n]$ to $[w]$. Each of these functions can be represented by its own $w \times n$ matrix, $\Pi_1, \ldots, \Pi_d$. The entire CountMin data structure can be viewed as a single, larger matrix, Π_{CM}, formed by *vertically concatenating (stacking)* these d individual hash matrices.

$$\Pi_{CM} = \begin{bmatrix} \Pi_1 \\ \Pi_2 \\ \vdots \\ \Pi_d \end{bmatrix}$$

The full sketch is then the product of this large $(d \cdot w) \times n$ matrix and the frequency vector f. The resulting vector of size $d \cdot w$ is the "flattened" $d \times w$ table of counters. Figure 1-(b) provides a perfect visual of this concatenation.

$$\text{(a)}\quad H^{(1)} \in \{0,1\}^{3\times 5} \qquad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} = \begin{bmatrix} f_4 \\ f_2 + f_3 \\ f_1 + f_5 \end{bmatrix}$$

$$\text{(b)}\quad R = \begin{bmatrix} H^{(1)} \\ H^{(2)} \\ H^{(3)} \end{bmatrix} \in \{0,1\}^{9\times 5} \qquad \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} = \begin{bmatrix} f_4 \\ f_2 + f_3 \\ f_1 + f_5 \\ f_1 + f_5 \\ f_2 + f_4 \\ f_3 \\ f_1 + f_2 \\ f_3 + f_4 \\ f_5 \end{bmatrix}$$

Figure 1: An illustration of the CountMin sketch as a linear projection. (a) A single hash function ($d = 1, w = 3$) represented by a 3×5 matrix. (b) A full Count-Min sketch ($d = 3, w = 3$) represented by a single 9×5 matrix formed by vertically stacking the three individual hash matrices.

References

Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.