Weekplan: Streaming III

Philip Bille Inge Li Gørtz

References and Reading

[1] Inge Li Gørtz. Lecture notes on the CountMin sketch.

[2] An improved data stream summary: the count-min sketch and its applications, G. Cormode and S. Muthukrishnan, J. Algorithms, Volume 55, Issue 1, April 2005, Pages 58-75

[3] Amit Chakrabarti: *Data Stream Algorithms* 2011 (updated July 2020) Section 5.5, 5.2, and 5.4.

We recommend reading [1] and [2] (except section 5.1) in detail. [3] is supplementary reading.

Exercises

1 Heavy hitters using dyadic intervals Solve the following exercises:

1.1 Explain how to calculate which bucket element a_i lands in at level j in constant time.

1.2 Explain the insert algorithm in detail and analyse its running time.

2 Range queries with a single CountMin sketch In this exercise we consider range queries of the form:

- `range`(a, b): Return an estimate for the number of elements with value between a and b (both included).

Explain how we can use a single CountMin sketch to solve range queries. What is the query time and how large is the expected error?

3 Range queries In this exercise we will use the tree of dyadic intervals to give a sketch for range queries. We maintain a separate CountMin sketch for each level in the tree, where for level j the jth CountMin sketch treats two elements that fall into the same interval in level j as the same element. For all intervals i in the tree, let $C(i)$ denote the value that the appropriate CountMin sketch returns for i.

3.1 Show that any interval in the range from 1 to n can be expressed as the disjoint union I of at most $2 \lg n$ dyadic intervals. *Hint:* Use the tree.

3.2 Explain how to use I to return an estimate for the range query `range`(a, b).

3.3 We want to have an additive error of at most ϵm as in the original CountMin sketch. Let $f_{[a,b]}$ be the total frequency of the elements in the range $[a, b]$. Show that if we set $w = \frac{2 \lg n}{\epsilon}$ in all the CountMin sketches then the expected error is ϵm, that is $\mathbb{E}\left[\sum_{i \in I} C(i)\right] \leq f_{[a,b]} + \epsilon m$.

3.4 What is the total space and query time? (we still use $d = \lg \frac{1}{\delta}$).

4 Combine Given two streams S_1 and S_2 and the CountMin sketches CM_1 and CM_2 of the two streams, explain how to combine the two sketches to get a sketch for the stream $S_1 \cup S_2$.

What do you need to assume about the two CountMin sketches for this to work?

5 Idea behind CountSketch Consider the following simple algorithm algorithm for finding estimates of all f_i. Let s be a pairwise independent hash function from $[n]$ to $\{-1,1\}$ and let c be a counter. While processing the stream, each time we encounter an item i, update the counter $c = c + s[i]$. When answering a query we return $\hat{f}_i = c \cdot s[i]$.

5.1 Compute the expected value of $s[j]$ for $j \in [n]$.

5.2 Show that $\hat{f}_i = f_i + \sum_{j \neq i} f_j \cdot s[j] \cdot s[i]$.

5.3 For two independent random variables X and Y we have: $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$. Use this and linearity of expectation to show that $\mathbb{E}[\hat{f}_i] = f_i$.

The variance of this algorithm is very large and many elements have estimates that are wrong by more than the variance. In the CountSketch algorithm there are t hash functions $h_j : [n] \to [w]$ and t hash functions $s_j : [n] \to \{-1,1\}$ and a $t \times w$ array of counters. The counters in row j are saved in array C_j. When an element q arrives the data structure is update by setting $C_j[h_j[q]] += s_j[q]$ for all $j \in [t]$. To estimate the frequency of element q return $\texttt{median}_j\{C_j[h_j[q]] \cdot s_j[q]\}$.

6 Heavy hitters using a single CountMin sketch

Heavy hitters Recall an element i in a stream is a *heavy hitter* if it has frequency $f_i > m/k$ for some given k. We define an *infrequent element* to be an element with frequency less than $\frac{m}{2k}$.

In this exercise we will use a single CountMin sketch and a binary min heap to get a streaming algorithm that outputs heavy hitters. The goal is to get an algorithm that returns all heavy hitters and with probability $1-\gamma$ returns no infrequent elements. The algorithm should be space and time efficient.

The algorithm will use a binary min heap to maintain some of the most frequent elements seen so far. Every time an element is encountered the algorithm updates the CountMin sketch and the heap. By the end of the stream the algorithm outputs all elements in the heap that has an estimated frequency more than m/k.

6.1 Explain how to update the heap when an element is encountered in the stream to ensure all heavy hitters are output. Remember to argue that your algorithm is correct.

6.2 We want to choose an error probability δ in the CountMin sketch such that the probability of the algorithm outputting an element with frequency $f_i < (1/k - \epsilon)m$ is bounded by γ. Show that this can be obtained by setting $\delta = \gamma/n$.

6.3 We want to ensure that with probability at least $1-\gamma$ the algorithm outputs no infrequent element. Explain how we can obtain this.

6.4 Let X be the number of elements with estimated frequency greater than m/k. Argue that $E[X] \leq 2k + n \cdot \gamma$. What should γ be if we want $E[X] = O(k)$?

6.5 What is the update time and expected space usage of your algorithm? What is the time complexity for reporting the heavy hitters?