Count-Min Sketch

Anil Maheshwari

anil@scs.carleton.ca
School of Computer Science
Carleton University
Canada

Outline

Majority element

Count-Min Sketch

Complexity Analysis

Markov's Inequality

Proof of the claim

Conclusions

Majority element

Problem

Finding the Majority Element

Input: A stream consisting of n elements and it is given that it has a majority element, i.e. it occurs at least $1 + \lfloor \frac{n}{2} \rfloor$ times

Output: The majority element.

An Example: $n = 19$

Input Stream = [3 2 4 7 2 2 3 2 2 1 4 2 2 2 1 1 2 3 2]

Straightforward Solutions

Solution 1: Store the stream in an array A.
Sort and pick the middle element.
Complexity: $O(n \log n)$ time and $O(n)$ space

Solution 2: Count frequency of each element.
Input: 3 2 4 7 2 2 3 2 2 1 4 2 2 2 1 1 2 3 2

Element	1	2	3	4	7
Frequency	3	10	3	2	1

Complexity: ?

Do we need that much space?

Finding the Majority Element

Input: A stream consisting of n elements and it is given that it has a majority element.
Output: The majority element.

Memory required in Solutions 1 & 2 $\geq$ Number of distinct elements in the stream.

What if we can only use $O(1)$ space?

Majority Algorithm

Input: Array A of size n consisting a majority element

Output: The majority element

```
c ← 0
for i = 1 to n do
    if c = 0 then
        current ← A[i]; c ← c + 1
    end
    else
        if A[i] = current then
            c ← c + 1
        end
        else
            c ← c − 1
        end
    end
end
return current
```

$A[i]$		3	2	4	7	2	2	3	2	...
current										...
c	0									...

Analysis of Majority Algorithm

Observations

1. Algorithm maintains only two variables: c and current.
2. Correctness: Each non-majority element can ‘kill’ at most one majority element.

Claim

By performing a single pass, using only $O(1)$ additional space, we can report the majority element of A (if it exists).

Misra $\&$ Gries [82] Algorithm

Finding Heavy Hitters

Input: A stream consisting of n elements and fixed integer $k < n$.

Output: Report all heavy hitters, i.e. elements that occur $\geq n/k$ times.

---

1. Initialize k bins, each with null element and a counter with 0.
2. **For** each element x in the stream **do**
   if $x \in$ Bin b **then** increment bin b's counter

   elseif find a bin whose counter is 0 and

   - Assign x to this bin
   - Assign 1 to its counter

   else decrement the counter of every bin.
3. Output elements in the bins.

Analysis of Misra and Gries Algorithm

Claim

Let $f_x^* =$ Frequency of x in the stream. Each heavy hitter x is in one of the bins with counter value $\geq f_x^* - n/k$.

Correctness: What can be the minimum value of the counter of a heavy hitter?

Running Time:
Initializing k bins: $O(k)$ time
Processing each element requires looking at $O(k)$ bins.
Total Run Time = $O(nk)$

Space: $O(k)$

Reference: J. Misra and D. Gries,“Finding repeated elements” in Science of Computer Programming, Vol. 2 (2): 143 -152, 1982.

Count-Min Sketch

Count-Min Sketch

Problem

For a data stream, using very little space, we are interested to report

1. All the elements that occur frequently, e.g at least 2% times.
2. For each element, its (approximate) frequency.

Count-Min Sketch Data Structure

Input: An array (stream) A consisting of n numbers and r hash functions $h_1, \ldots, h_r$, where $h_i : \mathbb{N} \to \{1, \ldots, b\}$

Output: $CMS[\cdot, \cdot]$ table consisting of r rows and b columns

```
for i = 1 to r do
    for j = 1 to b do
        CMS[i, j] ← 0
    end
end
for i = 1 to n do
    for j = 1 to r do
        CMS[j, h_j(A[i])] ← CMS[j, h_j(A[i])] + 1
    end
end
return CMS[·, ·]
```

Illustration of Algorithm

Let $b = 10$ and $r = 3$.

Assume that stream $A = xyy$.

Assume the following h-values for x and y:

For x: $h_1(x) = 3, h_2(x) = 8$, and $h_3(x) = 5$

For y: $h_1(y) = 6, h_2(y) = 8$, and $h_3(y) = 1$

$CMS[*, *] =$

	1	2	3	4	5	6	7	8	9	10
1										
2										
3										

```
for i = 1 to n do
    for j = 1 to r do
        CMS[j, h_j(A[i])] ← CMS[j, h_j(A[i])] + 1
    end
end
```

Complexity Analysis

Observations

Let $n =$ Total number of items in the stream.
$f_x^* =$ True frequency of x in the stream.

Let $f_x = \min\{CMS[1, h_1(x)], \ldots, CMS[r, h_r(x)]\}$. Report f_x as the estimate on the frequency of x.

Observations:

1. The size of CMS table ($= br$) is independent of n.
2. CMS table can be computed in $O(br + nr)$ time.
3. For any $x \in A$, and for any $j = 1, \ldots, r$, $CMS[j, h_j(x)] \geq f_x^*$
4. f_x is an overestimate as $f_x \geq f_x^*$

Assume - Proof comes later

Claim

Let $b = \frac{2}{\epsilon}$. Then $Pr[f_x - f_x^* \geq \epsilon n] \leq \frac{1}{2^r}$

Corollary

With probability at least $1 - 1/2^r$, $f_x^* \leq f_x \leq f_x^* + \epsilon n$

Reporting Frequent Elements

Suppose we want to report all the elements of A that occur approximately $\geq n/k$ times for some integer k.

- In the Claim, set $\epsilon = 1/3k$. Then $b = \frac{2}{\epsilon} = 6k$.
- Construct CMS table of size $br = 6kr$
- Scan A and compute the entries in the CMS table
- Maintain a set of $O(k)$ items that occur most frequently among all the elements in A scanned so far.

Heap Data Structure

The items are stored in a HEAP with f_x values as the key.

What is a Heap?

An array that stores n elements and supports:

- Find Max or Min: Report the element with the smallest/largest key value in Heap in $O(1)$ time.
- Insert(x, k): Insert element x with key k in Heap in $O(\log n)$ time.
- Delete(x): Delete element x from Heap in $O(\log n)$ time.
- $\ldots$

Reporting Frequent Elements contd.

Assume we have scanned $i - 1$ items and have updated the CMS table and the heap.
Consider the i-th item (say $x = A[i]$) and we perform the following:

1. For $j = 1$ to r: update the CMS table by executing $CMS[j, h_j(x)] \leftarrow CMS[j, h_j(x)] + 1$.
2. Let $f_x = \min\{CMS[1, h_1(x)], \ldots, CMS[r, h_r(x)]\}$.
   If $f_x \geq i/k$, do:
   2.1 If $x \in$ heap, delete x and re-insert it again with the updated f_x value.
   2.2 If $x \notin$ heap, then insert it in the heap and remove all the elements whose count is less than i/k.

Reporting Frequent Elements contd.

Claim [Cormode and Muthukrishnan 2005]

Elements that occur approx. n/k times in a data stream of size n can be reported in $O(kr + nr + n \log k)$ time using $O(kr)$ space with high probability.

Proof.

Recall Corollary: $f_x^* \leq f_x \leq f_x^* + \epsilon n = f_x^* + n/3k$.
This implies:

- Heap contains elements whose frequency is at least $n/k - n/3k = 0.667n/k$ (with high probability).
- Size of heap = $O(k)$
- Time Complexity: $O(br + nr + n \log k) = O(kr + nr + n \log k)$ as $b = \frac{2}{\epsilon} = 6k$.
- Total Space= $O(br + k) = O(kr)$

$\square$

Markov's Inequality

Markov's Inequality

Theorem

Let X be a non-negative discrete random variable and $s > 0$ be a constant. Then $P(X \geq s) \leq E[X]/s$.

Proof of the claim

Bounding f_x

Claim

Let $b = \frac{2}{\epsilon}$. Then $Pr[f_x - f_x^* \geq \epsilon n] \leq \frac{1}{2^r}$

Conclusions

Conclusions

What if we wanted to report exactly?
Do we need $\Omega(n)$ space?

Simple idea with important applications.

Consider a vector $v = (v_1, v_2, \ldots, v_n)$. Initially $v = 0$.
Update at time t is a pair (j, c): $v_j \leftarrow v_j + c$.
Using only small space, answer queries of the form

1. Point Query: Report v_i
2. Range Query $[l, r]$: Report $\sum_{i=l}^{r} v_i$
3. Inner product of two vectors: $u \cdot v$

Reference: An improved data stream summary: the count-min sketch and its applications, G. Cormode and S. Muthukrishnan, Jl. Algorithms 55(1): 58-75, 2005