Advanced Algorithms November 19, 2024

Lecture 17: Finding Frequent Items, Distinct Elements

Notes by Ola Svensson[1]

1 Finding Frequent Items Deterministically

(This is the Google search query problem).

We have a stream $\sigma = \langle a_1, \ldots, a_m \rangle$, with each $a_i \in [n]$. This implicitly defines a frequency vector $\mathbf{f} = (f_1, \ldots, f_n)$ (describing the number of times each query has been searched). Note that $f_1 + f_2 + \cdots + f_n = m$.

MAJORITY problem: If exists j such that $f_j > m/2$, then output j, otherwise, output "$\perp$".

FREQUENT problem with parameter k: Output the set $\{j : f_j > m/k\}$.

Unfortunately, both these problems requires space $\Omega(\min\{m, n\})$ if we limit ourselves to deterministic one-pass algorithms. However, we will see the Misra-Gries Algorithm'82 that solves the related problem of estimating the frequencies f_j and can also be used to solve the above problems in two passes.

1.1 The Misra-Gries Algorithm

The algorithms is a one-pass data stream algorithm. It consists of three sections.

- An initialization section, executed before we see the stream.
- A processing section, executed each time we see an element.
- An output section, where we answers question(s) about the stream.

The Misra-Gries Algorithm uses a parameter k that controls the quality of the answers it gives. It can be described as follows:

Initialization: $A \leftarrow$ (empty associative array).

Process j:

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove ℓ from A.

Output: On query a, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example 1 *It is instructive to run the algorithm on the stream $\langle 1, 1, 2, 2, 4, 4, 1, 4, 4, 1 \rangle$ with $k = 3$.*

We now analyze the space requirement and solution quality of the algorithm.

[1] **Disclaimer:** These notes were written as notes for the lecturer. They have not been peer-reviewed and may contain inconsistent notation, typos, and omit citations of relevant works.

1.1.1 Space requirement of algorithm

We store at most $k-1$ key/value pairs. Each key requires $\log n$ bits to store and each value at most $\log m$ bits. Hence the amount of space we use is $O(k(\log n + \log m))$. So we are happy with the space requirement of the algorithm.

1.1.2 Solution quality

- Let us pretend that A consists of n key/value pairs, with $A[j] = 0$ whenever j is not actually stored in A by the algorithm.
- Notice that the counter $A[j]$ is incremented only when we process an occurrence of j in the stream. Thus

$$\hat{f}_j \leq f_j.$$

- On the other hand, how often can we decrement the counters? Whenever $A[j]$ is decremented (we pretend that $A[j]$ is incremented from 0 to 1, and then immediately decremented back to 0), we also decrement $k-1$ other counters.
- Since the stream consists of m elements, there can be at most m/k such decrements. Therefore

$$f_j - \frac{m}{k} \leq \hat{f}_j.$$

We summarize the facts about the Misra-Gries algorithm in the following theorem.

Theorem 1 *The Misra-Gries algorithm with parameter k uses one pass and $O(k(\log m + \log n))$ bits of space, and provides, for any token j, an estimate $\hat{f}_j$ satisfying*

$$f_j - \frac{m}{k} \leq \hat{f}_j \leq f_j.$$

How can you use the Misra-Gries Algorithm to solve the FREQUENT problem with one additional pass? If some token j has $f_j > m/k$ then its corresponding counter $A[j]$ will be positive in the end of the Misra-Gries Algorithm. Thus we can make a second pass over the input stream, counting exactly the frequencies f_j for all $j \in keys(A)$, and then output the desired set of items.

2 Estimating the Number of Distinct Elements

(This is the Facebook problem, i.e., the number of different cities on Facebook.)

DISTINCT-ELEMENTS problem Our goal is to output an approximation to then number $d(\sigma) = |\{j : f_j > 0\}|$ of distinct elements that appear in the stream σ.

It is provably impossible to solve this problem in sublinear space if one is restricted to either deterministic algorithms or exact algorithms. Thus we shall seek a randomized approximation algorithms. More specifically, we give a guarantee of the following type

$$\Pr[d(\sigma)/3 \leq A(\sigma) \leq 3d(\sigma)] \geq 1 - \delta,$$

i.e., with probability $1-\delta$ we have a 3-approximate solution. (With more work the 3 can be improved to $1+\varepsilon$ for any $\varepsilon > 0$.) The amount of space we use will be $O(\log(1/\delta) \log n)$.

2.1 Ingredients

2.1.1 Pairwise independent hash family

A family $\mathcal{H}$ of functions of the type $[n] \to [n]$ is said to be a pairwise independent hash family if the following property holds, with $h \in \mathcal{H}$ picked uniformly at random:

for any $x \neq x' \in [n]$ and $y, y' \in [n]$ we have

$$\Pr_{h \sim \mathcal{H}}[h(x) = y \wedge h(x') = y'] = 1/n^2.$$

Note that this implies that $\Pr_h[h(x) = y] = 1/n$. For us the following fact will be important (which follows from the construction in last lecture):

Lemma 2 *There exists a pairwise independent hash family so that h can be sampled by picking $O(\log n)$ random bits. Moreover, $h(x)$ can be calculated in space $O(\log n)$.*

2.1.2 The zero function

For an integer $p > 0$ let $zeros(p)$ denote the number of zeros that the binary representation of p ends with. Formally,

$$zeros(p) = \max\{i : 2^i \text{ divides } p\}.$$

Examples are $zeros(2) = 1, zeros(3) = 0, zeros(4) = 2, zeros(6) = 1, zeros(7) = 0$.

2.2 The Algorithm

The basic intuition of the algorithm is as follows:

- The probability that a random number x has $zeros(x) \geq \log d$ is $1/d$.
- So if we have d distinct numbers we would expect $zeros(h(j)) \geq \log d$ for some element j.

The algorithm is now very simple:

Initialization: Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

Process j: If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

Output: $2^{z+1/2}$.

We only use $O(\log n)$ space. Let's now analyze the quality of the output.

2.3 Analysis of Algorithm

- For each $j \in [n]$ and each integer $r \geq 0$, let $X_{r,j}$ be an indicator random variable for the event "$zeros(h(j)) \geq r$," and let $Y_r = \sum_{j:f_j>0} X_{r,j}$.
- Let t denote the value of z when algorithm terminates. By definition,

$$Y_r > 0 \iff t \geq r. \tag{1}$$

[2]To ease calculations we assume that n is a power of two and so we hash a value p to a uniformly at random binary string of length $\log_2(n)$.

- It will be useful to restate this fact as follows:

$$Y_r = 0 \iff t \leq r - 1. \tag{2}$$

In the next lecture we will compute the expectation and variance of Y_r, and use these bounds to prove that Y_r is likely positive for r such that $2^{r+1/2} < d/3$ and Y_{r+1} is likely zero for r such that $2^{r+1/2} > 3d$. This will establish the required approximation guarantee.