College of Computer & Information Science
Northeastern University
CS 7880: Network Algorithms

Spring 2016
15 March 2016
Scribe: Giorgos Zirdelis

- Streaming
  - Distinct items (cont.)

1 Distinct items

In the streaming model, we see a stream σ of m elements drawn for a universe $\{1, \ldots, n\}$. We process the stream in order, and compute a function ϕ over the stream σ. The function ϕ is essentially a function of the frequency distribution of σ, namely $f = \langle f_1, \ldots, f_n \rangle$.

The goal is to approximate ϕ using low space. Specifically, assume that $\mathcal{A}$ is a randomized algorithm with output $\mathcal{A}(\sigma)$. We say that $\mathcal{A}$ is a (ε, δ) approximation of ϕ, if for all streams it holds that

$$(1 - \varepsilon)\phi(\sigma) \leq \mathcal{A}(\sigma) \leq (1 + \varepsilon)\phi(\sigma)$$

with probability $1 - \delta$ for some $\varepsilon, \delta > 0$. Ideally, we want to use space $O(\log m + \log n)$ or in general sublinear in terms of n and m.

To approximate the number of distinct elements in a stream σ, we want to do something different for each distinct element and do the same thing for the same elements. For that reason, we will use hashing. Assume we have a universal hash function $h : [n] \to [n]$ and suppose that k is the number of distinct elements in σ.

Before we continue with the analysis we recall the problem of throwing k balls into n bins, where $k \leq n$. Let X denote the number of bins that will be occupied. We want to know the value of X in expectation. We have that,

$$\Pr[\text{Bin is empty}] = \left(1 - \frac{1}{n}\right)^k.$$

Therefore,

$$\mathbb{E}[X] = n\left(1 - \left(1 - \frac{1}{n}\right)^k\right).$$

Because we have k balls we know that $\mathbb{E}[X] \leq k$ so we want a lower bound. Expand $\left(1 - \frac{1}{n}\right)^k$ for two steps using the binomial formula:

$$\overbrace{1 - \frac{k}{n} + \binom{k}{2}\frac{1}{n^2}}^{\text{two step expansion}} = 1 - \frac{k}{n} + \frac{k(k-1)}{2n^2} \geq \left(1 - \frac{1}{n}\right)^k$$

Hence,

$$\mathbb{E}[X] \geq n\left(1 - \left(1 - \frac{k}{n} + \frac{k(k-1)}{2n^2}\right)\right) = k - \frac{k(k-1)}{2n^2} \geq \frac{k}{2}$$

Returning back to counting distinct elements, one can make an analogy of the previous balls & bins problems and hashing. Instead of counting occupied bins, assume we focus on a small fraction of n, i.e. a bucket of size αn, where $\alpha < 1$ and count how many elements land in that bucket. On expectation we will have αk elements in a bucket of size αn. By setting $\alpha = \frac{1}{k}$ and relying only on expectation we can say that if there is one element in that bucket we have approximately k distinct elements. If there is more that one than one element then approximately we have more that k elements, and if there is no element then we have less than k elements.

The algorithm that uses hashing works as follows: let $\mathsf{zeros}(x)$ denote the number of trailing zeros of x,

```
μ ← 0;
while a_j ∈ σ do
    z ← zeros(h(a_j));
    if μ < z then
        μ ← z;
    end
end
return 2^μ;
```

The intuition for this algorithm can be summarized with the following table:

# trailing zeros	# elements in $[n]$	# elements in area $\frac{n}{\#\text{ trailing zeros}}$ (expected)
1	n/2	k/2
2	n/4	k/4
$\vdots$	$\vdots$	$\vdots$
$\log k$	n/k	1

Next we calculate the probability that k is close to the estimate 2^μ of the algorithm using the following cases:

Case 1

$$\Pr[k > 2^r 2^\mu] \leq \Pr[\text{no element was mapped with } 1+\mu \text{ trailing zeros}]$$

Let Y denote the event: "number of elements that were mapped with $1+\mu$ trailing zeros".

Case 2

$$\Pr[k < 2^{-r} 2^\mu] \leq \Pr[\text{some element was mapped with } \mu \text{ trailing zeros}]$$

Let Z denote the event: "number of elements that were mapped with μ trailing zeros".

Note that r is a constant that we will substitute later.

First we calculate the expectation of events Y, Z and then use Chebyshev's and Markov's inequality respectively to bound the probability.

Case 1

$$\mathbb{E}[Y = 0] = \frac{k}{2^{1+\mu}} > \frac{2^r 2^\mu}{2^{1+\mu}} = \frac{2^r}{2} = 2^{r-1}$$

Case 2

$$\mathbb{E}[Z \geq 1] = \frac{k}{2^\mu} < \frac{2^{-r} 2^\mu}{2} = \frac{1}{2^r}$$

By Markov's inequality we have that $\Pr[Z \geq 1]$ is no more than $\mathbb{E}[Z \geq 1] = 2^{-r}$.

Define $X_i = \begin{cases} 1, & \text{if the i-th distinct elem. is mapped to } \mu+1 \text{ trailing zeros (with prob. } 1/2^{\mu+1}) \\ 0, & \text{otherwise} \end{cases}$

Then it is the case that $Y = \sum_{i=1}^{k} X_i$ and $\mathrm{Var}[Y] = \sum_{i=1}^{k} \mathrm{Var}[X_i]$. But $\mathrm{Var}[X_i] \leq \mathbb{E}[X_i^2] = \mathbb{E}[X_i] = 1/2^{\mu+1}$, and so $\sum_{i=1}^{k} \mathrm{Var}[X_i] = k/2^{\mu+1}$.

Using Chebyshev's inequality $\Pr[|Y - \mathbb{E}[Y]| \geq \varepsilon \mathbb{E}[Y]] \leq \frac{\mathrm{Var}[Y]}{\varepsilon^2}$ for $\varepsilon = 1$, we obtain that

$$\Pr[Y = 0] \leq \frac{k/2^{\mu+1}}{(k/2^{\mu+1})^2} = \frac{2^{\mu+1}}{k}$$

and because we are in the event where $k > 2^{\mu}2^{r}$ it follows that

$$\Pr[Y = 0] \leq 1/2^{r-1}.$$

By setting $r = 3$, the previous algorithm will return an estimate that is with probability $\frac{3}{4}$ between $\frac{k}{8}$ and k, and with probability $\frac{7}{8}$ between k and $8k$. In total the probability of the estimate landing in $\frac{k}{8}$ and $8k$ is no less than $1 - \frac{1}{8} - \frac{1}{4} = \frac{5}{8}$.

To amplify the probability of landing in the above interval, we run the same algorithm in parallel with a different hash function each time, for $t = c\log(\frac{1}{\delta})$ times and return the median of all returned values. The probability that the median lands outside the interval $[\frac{k}{8}, 8k]$ is less than $\binom{t}{t/2}\left(\frac{3}{8}\right)^{t/2} \leq \delta$ by setting c properly.

While Markov's inequality requires no independence for events, Chebyshev's inequality requires pairwise independence. Hence, we need a 2-wise independent hash function for the algorithm. There are 2-wise independent hash function families that have polynomial number of hash functions so we need $\log(n^{c'}) = c' \log n$ bits to represent them for some constant c'. We also need $O(t \log k)$ bits to store the returned values of the all the parallel runs of the algorithm. In total we need logarithmic space in terms of n.