(CS602) Applied Algorithms 14 Jan, 2015

Lecture 3: Introduction

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

This course will focus on algorithms for large data. The course is vaguely divided into four modules.

- Algorithms for large datasets – frequency moments computations.
- Algorithms for problems in linear algebra – dimension reduction, norm computation.
- Algorithms for graph problems.
- Lower bound results via information theory.

The first two modules will be discussed before the midsem and the last two after midsem. The grading scheme will be uploaded on the Moodle page of the course.

3.1 Introduction

We begin with two simple problems:
Problem 1: Given a stream of numbers, output the length of the input stream.
Problem 2: Given a graph as a set of edges, output the maximum matching in a graph.

Problem 1 is easily solvable in space $O(\log n)$, where n is the length of the input. Problem 2 has a polynomial time algorithm. Therefore, both the problems are *easy* in the classical model of computation. However, the model that we will study in this course is slightly restrictive:

- The input data is *very large* as compared to the space available to the algorithm.
- Rereading input bits is *very expensive*.
- Time for computation per input bit is *very small*.

Typically, if the input is of length, say n, we will be required to design algorithms which use space $o(n)$ and read every input bit $O(1)$ times. Under these assumptions, let us first design a 2-approximation algorithm for Problem 2.

3.2 Problem 2

Consider a simple greedy algorithm which starts with an empty set M and adds a new edge say e to M iff $M \cup \{e\}$ is a matching. This algorithm looks at every edge exactly once, uses space $O(n \log n)$ which is $o(n^2)$. However, this does not necessarily output the maximum matching. For every edge it adds to M, it may in the worst case exclude 2 edges from the optimal matching. Therefore, this algorithm gives a 2-approximation.

3.3 Problem 1

For Problem 1 the naive algorithm already has all the desired properties, i.e. it looks at every input element exactly once, and uses space $O(\log n)$ which is $o(n)$. However, one could ask whether we can reduce the space used by the algorithm even further. It is easy to see that if we wish to compute the length of the input exactly, then $\Omega(\log n)$ bits are necessary. We now discuss a randomized algorithm design by Morris [1], which gives a 2-approximation with probability at least $3/4$ and uses space $O(\log \log n)$.

$Y \leftarrow 0$;
while *there exists* x_i*, an input element* **do**
  $Y \leftarrow Y + 1$ w.p. $\frac{1}{2^Y}$;
  $Y \leftarrow Y$ w.p. $1 - \frac{1}{2^Y}$;
end
Output 2^Y

We will analyze the algorithm in the next class.

References

[1] R. Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, Oct. 1978.

(CS602) Applied Algorithms 16 Jan, 2014

Lecture 4: Computing the length of the input

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

In the last class, we started with two problems:
Problem 1: Given a stream of numbers, output the length of the input stream.
Problem 2: Given a graph as a set of edges, output the maximum matching in a graph.

We saw a $O(n \log n)$ space 2-approximation algorithm for Problem 2. We gave an algorithm for Problem 1. Today we will analyze the algorithm.

4.1 Algorithm for Problem 1 and analysis

Let us start by recalling the algorithm.

$Y \leftarrow 0$;
while *there exists* x_i*, an input element* **do**
  $Y \leftarrow Y + 1$ w.p. $\frac{1}{2^Y}$;
  $Y \leftarrow Y$ w.p. $1 - \frac{1}{2^Y}$;
end
Output $2^Y - 1$

The algorithm increments the counter with lower and lower probability as the length of the input increases. We will first analyze the expected value and the variance of the output after i steps.

Lemma 4.1.1. $\mathbb{E}(2^{Y_i}) = i + 1$

Proof.

$$\begin{aligned}
\mathbb{E}(2^{Y_i}) &= \sum_{j=0}^{\infty} \left(\left[\mathbb{E}(2^{Y_i}) | Y_{i-1} = j \right] \Pr\left[Y_{i-1} = j \right] \right) \\
&= \sum_{j=0}^{\infty} \left(2^{j+1} \frac{1}{2^j} + 2^j (1 - \frac{1}{2^j}) \right) \Pr\left[Y_{i-1} = j \right] \\
&= \sum_{j=0}^{\infty} 2^j \Pr\left[Y_{i-1} = j \right] + \sum_{j=0}^{\infty} \Pr\left[Y_{i-1} = j \right] \\
&= \mathbb{E}(2^{Y_{i-1}}) + 1
\end{aligned}$$

Using the recurrence, we get the lemma. □

Lemma 4.1.2. $\mathbb{V}ar(2^{Y_i}) = \frac{i(i-1)}{2}$

Proof.

$$\begin{aligned}
\mathbb{E}(2^{2Y_i}) &= \sum_{j=0}^{\infty} \left(\left[\mathbb{E}(2^{2Y_i}) | Y_{i-1} = j \right] \Pr\left[Y_{i-1} = j \right] \right) \\
&= \sum_{j=0}^{\infty} \left(2^{2(j+1)} \frac{1}{2^j} + 2^{2j}(1 - \frac{1}{2^j}) \right) \Pr\left[Y_{i-1} = j \right] \\
&= \sum_{j=0}^{\infty} 2^{2j} \Pr\left[Y_{i-1} = j \right] + 3 \sum_{j=0}^{\infty} 2^j \Pr\left[Y_{i-1} = j \right] \\
&= \mathbb{E}(2^{2Y_{i-1}}) + 3i
\end{aligned}$$

Solving the recurrence, we get that $\mathbb{E}(2^{2Y_i}) = 1 + \sum_{k=1}^{i} 3k = 1 + \frac{3i(i+1)}{2}$. Therefore, $\mathbb{V}ar(2^{Y_i}) = 1 + \frac{3i(i+1)}{2} - (i+1)^2 = \frac{i(i-1)}{2}$ □

Lemmas 4.1.1 indicates that the expected value of the output of the algorithm is equal to the actual length of the input. This is a good sign. Lemma 4.1.2 indicates that the variance is not too large. This again is useful. Now using Chebyshev we have $\Pr\left[|2^{Y_{n+1}} - (n+1)| \geq 0.9(n+1) \right] \leq \frac{n(n-1)}{1.62(n+1)^2} < \frac{3}{4}$. This tells us that with probability at least 1/4 the algorithm gives a 0.9 approximation.

Definition 4.1.3. *A randomized algorithm $\mathcal{A}$ computing a function f is said to be an (ε, δ) algorithm for f if for every input x, $\Pr\left[(1-\varepsilon)f(x) \leq A(x) \leq (1+\varepsilon)f(x) \right] \geq 1 - \delta$.*

In this sense, the algorithm we have is an $(0.9, 3/4)$ algorithm for computing the length of the input.

4.2 Improving approximation guarantee

We now describe the standard trick used to increase the approximation gurantee. Let us call the algorithm designed in Section 4.1 as A_1. Given an $\varepsilon > 0$ and algorithm A_1, we give another algorithm A_2 such that $\Pr\left[(1-\varepsilon)f(x) \leq A_2(x) \leq (1+\varepsilon)f(x) \right] \geq 2/3$.

for $j = 1$ *to* t **do**
  $Y^{(j)} \leftarrow 0$;
end
while *there exists* x_i*, an input element* **do**
  for $j = 1$ *to* t **do**
    $Y^{(j)} \leftarrow Y^{(j)} + 1$ w.p. $\frac{1}{2^{Y^{(j)}}}$;
    $Y^{(j)} \leftarrow Y^{(j)}$ w.p. $1 - \frac{1}{2^{Y^{(j)}}}$;
  end
end
Output $\frac{\sum_{i=1}^{t} 2^{Y^{(i)}} - 1}{t}$

Here, t is a parameter, which we will fix shortly. Let Z_n denote the output of A_2 for inputs of length n. It is easy to see that $\mathbb{E}(Z_n) = n$ and $\mathbb{V}ar(Z_n) = \frac{n(n-1)}{2t}$. Therefore, by applying Chebyshev's inequality we get $\Pr[|Z_n - n| \geq \varepsilon n] \leq \frac{n(n-1)}{2t\varepsilon^2 n^2} < \frac{1}{2t\varepsilon^2}$. By setting $t = \lceil \frac{3}{2\varepsilon^2} \rceil$ we get that $\Pr[|Z_n - n| \geq \varepsilon n] < 1/3$. That is, we get for every input x $\Pr[(1-\varepsilon)f(x) \leq A_2(x) \leq (1+\varepsilon)f(x)] \geq 2/3$. Suppose $s(n)$ is the space used by A_1 for inputs of length n, then as $t = O(1/\varepsilon^2)$, the space used by A_2 is $ts(n) = O(s(n)/\varepsilon^2)$.

Remark 4.2.1. *Observe that the error probability that we obtained here could be made small enough by setting t appropriately. Suppose we needed the error probability to be δ, the space used by the algorithm would take an additional blow of $O(1/\delta)$. In the next section, we show how to reduce the error by increasing the space by only $O(\log(\frac{1}{\delta}))$.*

4.3 Decreasing the error probability

Consider the following modified algorithm:

for $j = 1$ *to* t **do**
  for $\ell = 1$ *to* k **do**
    $Y^{(j,\ell)} \leftarrow 0$;
  end
end
while *there exists x_i, an input element* **do**
  for $j = 1$ *to* t **do**
    for $\ell = 1$ *to* k **do**
      $Y^{(j,\ell)} \leftarrow Y^{(j,\ell)} + 1$ w.p. $\frac{1}{2^{Y^{(j,\ell)}}}$;
      $Y^{(j,\ell)} \leftarrow Y^{(j,\ell)}$ w.p. $1 - \frac{1}{2^{Y^{(j,\ell)}}}$;
    end
  end
end
Output Median of $\left(\frac{\sum_{j=1}^{t} 2^{Y^{(j,1)}} - 1}{t}, \frac{\sum_{j=1}^{t} 2^{Y^{(j,2)}} - 1}{t}, \ldots, \frac{\sum_{j=1}^{t} 2^{Y^{(j,k)}} - 1}{t} \right)$

Let us call this algorithm A_3. Here, let t be as fixed in Section 4.2, i.e. $t = \frac{3}{2\varepsilon^2}$. Let us define $Z_\ell = \frac{\sum_{j=1}^{t} 2^{Y^{(j,\ell)}} - 1}{t}$ for $1 \leq \ell \leq k$. And let Y_ℓ be a 0-1 random variable which is set to 1 if $(1+\varepsilon)n \leq Z_\ell \leq (1+\varepsilon)n$ for $1 \leq \ell \leq k$. Then we know that for $1 \leq \ell \leq k$, $\mathbb{E}(Y_\ell) = 2/3$. That is, in expectation, about 2/3rd of the Y_ℓs are in the correct range of values. The algorithm A_3 outputs the medial of these Y_ℓs. If more than half of the Y_ℓs have the values in the right range, the median will be in the right range. Therefore, to bound the error of the algorithm we need to bound the probability of the event that strictly less than half of the Y_ℓs are in the right range. Let $Y = \sum_{\ell=1}^{k} Y_\ell$. We wish to bound the probability that $Y < k/2$. Note that $\mathbb{E}(Y) = 2k/3$ by linearity of expectations. $\Pr[Y < k/2] = \Pr[|Y - \mathbb{E}(Y)| \geq k/6] = \Pr\left[|Y - \mathbb{E}(Y)| \geq \frac{\mathbb{E}(Y)}{4}\right]$. This can be bounded by

using the Chernoff bound as follows: $\Pr\left[|Y - \mathbb{E}(Y)| \geq \frac{\mathbb{E}(Y)}{4}\right] \leq 2.e^{-\frac{2k}{16 \cdot 3}}$. To make this smaller than $\delta > 0$, we need to $k = O(\log(\frac{1}{\delta}))$.

To summarize, we have designed an algorithm A_3 which runs for $O\left(\frac{1}{\varepsilon^2} \log\left(\frac{1}{\delta}\right)\right)$ iterations and has the following guarantee: $\Pr\left[(1-\varepsilon)n \leq A_3(x) \leq (1+\varepsilon)n\right] \geq 1 - \delta$.

(CS602) Applied Algorithms 28 Jan, 2015

Lecture 7: Computing the number of distinct elements

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

In the last class we gave a randomized (ε, δ) algorithm[1] for approximating the length of the input. In this class we will build towards coming up with an algorithm for approximating the number of distinct elements in the input stream.

Given an input stream $x_1, x_2, \ldots, x_n$, where each $x_i \in [m]$ and $\varepsilon, \delta > 0$ constants, our goal is to design a randomized (ε, δ) algorithm for approximating the number of distinct elements using space $O(\log m)$. We will give two approaches for this, which will introduce you to two ideas. Unfortunately, none of them will manage to achieve the correct space bounds. Finally, the exact space bounds will be achieved in the next class.

7.1 Approach 1

Pick h uniformly randomly from $\mathcal{H} = \{h : [m] \to [0, 1]\}$;
$Z \leftarrow 1$;
while *there exists x, an input element* **do**
  if $Z > h(x)$ **then**
    $Z \leftarrow h(x)$;
  end
end

The algorithm assigns minimum over all x, $h(x)$ to Z. For a fixed input string, suppose $D \subseteq [m]$ is the set of distinct elements in the string. Let $d = |D|$. The above algorithm takes the set D and maps it to the real interval $[0, 1]$. As this happens uniformly in the interval, intuitively the elements in D will be equispaced inside the interval $[0, 1]$. Therefore, suppose $min_{x \in D} h(x)$ is equal to 0.25 then it is very likely that there were $3 = (1/0.25 - 1)$ elements in D: one which got mapped to 0.25, another to 0.5 and the last to 0.75. It seems that by simply randomly hashing into the $[0, 1]$ interval, and computing the min of the hash values, we can get a good idea about the number of distinct elements in the input.

We will now make this intuition concrete in the next few steps of calculations. We will compute the expectation and variance of Z. Note that the probability is over the random choice of $h \in \mathcal{H}$.

[1] Defined in last lecture note.

$$
\begin{aligned}
\mathbb{E}(Z) &= \int_0^1 \Pr\left[Z > \lambda\right] d\lambda \\
&= \int_0^1 \Pr\left[\prod_{x \in D} h(x) > \lambda\right] d\lambda && (Z \text{ is the min among all } h(x)) \\
&= \int_0^1 (1-\lambda)^d d\lambda && (\text{As } h(x)\text{s are uniformaly distributed over } [0,1] \text{ and } |D| = d) \\
&= \frac{u^{d+1}}{d+1}\Big|_0^1 && (\text{By change of variables}) \\
&= \frac{1}{d+1}
\end{aligned}
$$

To compute $\mathbb{V}ar(Z)$, we first compute $\mathbb{E}(Z^2)$ similarly.

$$
\begin{aligned}
\mathbb{E}(Z^2) &= \int_0^1 \Pr\left[Z^2 > \lambda\right] d\lambda \\
&= \int_0^1 \Pr\left[Z > \sqrt{\lambda}\right] d\lambda \\
&= \int_0^1 \Pr\left[\prod_{x \in D} \left(h(x) > \sqrt{\lambda}\right)\right] d\lambda && (Z \text{ is the min among all } h(x)) \\
&= \int_0^1 (1-\sqrt{\lambda})^d d\lambda && (\text{As } h(x)\text{s are uniformaly distributed over } [0,1] \text{ and } |D| = d) \\
&= 2\left(\frac{u^{d+1}}{d+1} - \frac{u^{d+2}}{d+2}\right)\Big|_0^1 && (\text{By change of variables}) \\
&= \frac{2}{(d+1)(d+2)}
\end{aligned}
$$

From this we get that $\mathbb{V}ar(Z) = O(1/d^2)$.

To bring down the variance by a factor of q, we run q parallel copies of the algorithm and take average of their outputs and declare that as our new output. This is called the averaging trick and is decsribed in Lecture 2, Section 2.2. Let $\tilde{Z}$ denote the averaged output. Then it is easy to see that $\mathbb{E}(\tilde{Z}) = \mathbb{E}(Z)$ and $\mathbb{V}ar(\tilde{Z}) = \frac{\mathbb{V}ar(Z)}{q}$. By Chebyshev's inequality we will get the following:

$$
\Pr_{h \in \mathcal{H}}\left[\left|\tilde{Z} - \frac{1}{d+1}\right| \geq \frac{\varepsilon}{d+1}\right] \leq O\left(\frac{1}{q\varepsilon^2}\right)
$$

Now choosing q appropriately, we get the above probability is strictly smaller than $1/3$.

This finishes the description of this approach. There are a couple of problems with this approach:

- As the range of the functions is $[0,1]$, the family $\mathcal{H}$ is uncountable. Therefore, we will not be able to implement the step of picking a random function from $\mathcal{H}$.

- Again, as Z takes value in $[0,1]$ infinite precision may be required to store Z.

Our next approach partially addresses these issues.

7.2 Approach 2

We first change the family of functions. Let $\mathcal{H}' = \{h : [m] \to \{0,1\}^K\}$, where K is such that $2^K > m$. Note that by changing the range of the functions, we manage to address the problems related to Approach 1. However, if we implement the exact same algorithm as in Section 7.1 with respect to this family then the algorithm will use $\log(2^{mK}) = O(m^2)$ bits to pick a random function.[2] This is better than Approach 1, however not good enough — as we already know a $O(m)$ algorithm for exactly computing the number of distinct elements in any input stream. So this change of family of functions is not quite enough to achieve good space bounds.

In any case, we will use this family $\mathcal{H}'$ to design a different randomized algorithm, say $\mathcal{A}_2$, to approximate the number of distinct elements. We will then observe some properties of the algorithm. These properties will indicate that to achieve the same performce guarantee — that is the same approximation factor and the same amount of error — we need not pick a random function from the family $\mathcal{H}'$ but from a sub-family $\mathcal{H}'' \subset \mathcal{H}'$. In the next class we will construct a sub-family $\mathcal{H}''$ and an algorithm $\mathcal{A}'_2$ such that $|\mathcal{H}''| = O(m^2)$ and $\mathcal{A}'_2$ is the same as $\mathcal{A}_2$, except that it chooses a random function from $\mathcal{H}''$. Note that picking a random function from $\mathcal{H}''$ will need only $O(\log m)$ bits.

We now describe the algorithm.

Pick h uniformly randomly from $\mathcal{H}' = \{h : [m] \to \{0,1\}^K\}$;
$Z \leftarrow 0$;
while *there exists x, an input element* **do**
  if $Z <$ *#traling zeroes in* $h(x)$ **then**
    $Z \leftarrow h(x)$;
  end
end
Output 2^Z;

Let D denote the distinct elements in the streams and $|D| = d$. The algorithm spreads the elements of D uniformaly in the range $\{0,1\}^K$. It then stores the maximum over $x \in D$ the number of trailing 0s observed in $h(x)$. The rationale behind counting the trailing zeroes can be explained as follows: consider the random experiment in which you pick a random subset from $\{1, 2, \ldots, n\}$. In this case, about half the elements will be divisible by 2, about 1/4th will be divisible by 4 and so on. Turning it around, if d elements are mapped uniformly and randomly to $\{1, 2, \ldots, n\}$ then at least 1 of them will be divisible by $2^{\log d}$.

We now work out the calculations. For the ease of calculations, let us introduce some notation. Let $Y_{x,\ell}$ be a 0-1 random variable, which is defined to be 1 iff #traling zeroes in $h(x) > \ell$. And let $\tilde{Y}_\ell$ denote $\sum_{x \in D} Y_{x,\ell}$.

[2] This is because $|\mathcal{H}'| = (2^K)^m$.

Observation 7.2.1. *$\tilde{Y}_\ell = 0$ if and only if $Z < \ell$.*

The proof of the above observation is left as an easy exercise. To bound the error probability of the algorithm, we would like to show that the following probability is high: $\Pr\left[\frac{d}{c} \leq 2^Z \leq cd\right]$. We will do this by bounding the probabilities of the following two quantities:

$$\Pr\left[2^Z > cd\right] \leq \Pr\left[Z > \lceil \log(cd) \rceil\right] \qquad (A)$$

$$\Pr\left[2^Z < d/c\right] \leq \Pr\left[Z < \lfloor \log(d/c) \rfloor\right] \qquad (B)$$

Let U denote $\lceil \log(cd) \rceil$ and L denote $\lfloor \log(d/c) \rfloor$ in $(A), (B)$, respectively. Using Observation 7.2.1 we know that $\Pr\left[Z < L\right] = \Pr\left[\tilde{Y}_L = 0\right]$. Similarly, we also know that $\Pr\left[Z > U\right] = \Pr\left[\tilde{Y}_U > 0\right]$. Therefore, now we are left to upper bound the probability of these two events. We bound them as follows:

$$\begin{aligned}\Pr\left[\tilde{Y}_U > 0\right] &= \Pr\left[\tilde{Y}_U \geq 1\right] \\ &\leq \mathbb{E}(\tilde{Y}_U)/1 \qquad \text{(By Markov's inequality)}\end{aligned}$$

$$\begin{aligned}\Pr\left[\tilde{Y}_L = 0\right] &= \Pr\left[\left|\tilde{Y}_L - \mathbb{E}(\tilde{Y}_L)\right| \geq \mathbb{E}(\tilde{Y}_L)\right] \\ &\leq \mathbb{V}ar(\tilde{Y}_L)/(\mathbb{E}(\tilde{Y}_L))^2 \qquad \text{(By Chebyshev's inequality)}\end{aligned}$$

Lemma 7.2.2. *For every $1 \leq \ell \leq K$, $\mathbb{E}(\tilde{Y}_\ell) = \frac{d}{2^l}$ and $\mathbb{V}ar(\tilde{Y}_\ell) < \frac{d}{2^l}$*

Proof. $\mathbb{E}(\tilde{Y}_\ell) = \mathbb{E}(\sum_{x \in D} Y_{x,\ell}) = \sum_{x \in D} \mathbb{E}(Y_{x,\ell}) = F_0 \cdot \mathbb{E}(Y_{x,\ell}) = F_0/2^\ell$. Here, the first equality comes from the definition of $\tilde{Y}_\ell$, the second equality comes from linearity of expectation, the third equality comes from the fact that $\mathbb{E}(Y_{x,\ell})$ is the same for every $x \in D$ and the final equality comes from the fact that $\Pr\left[Y_{x,\ell} = 1\right] = 1/2^\ell$.

For computing variance, we first compute $\mathbb{E}(\tilde{Y}_\ell^2)$.

$$\begin{aligned}\mathbb{E}(\tilde{Y}_\ell^2) &= \mathbb{E}\left((\sum_{x \in D} Y_{x,\ell})^2\right) \\ &= \mathbb{E}\left(\sum_{x \in D} Y_{x,\ell}^2 + \sum_{x \neq x' \in D} Y_{x,\ell} \cdot Y_{x',\ell}\right) \\ &= \sum_{x \in D} \mathbb{E}(Y_{x,\ell}^2) + \sum_{x \neq x' \in D} \mathbb{E}(Y_{x,\ell} \cdot Y_{x',\ell})\end{aligned}$$

□

The proof was worked out in detail in class. It is now left as an exercise. Now putting it all together we get that $\Pr\left[\tilde{Y}_U > 0\right] \leq \frac{d}{2^U} \leq 1/c$. The first inequality is due to Lemma 7.2.2 and the second inequality is by using the value of U. Similarly, we get $\Pr\left[\tilde{Y}_L = 0\right] < \frac{2^L}{d} \leq 1/c$. Again, the first inequality is due to Lemma 7.2.2 and the second inequality is by using the value of L.

7.3 Exercise

Exercise 1. *Suppose the algorithm for Approach 1 is changed as follows:*

Pick h uniformly randomly from $\mathcal{H} = \{h : [m] \to [0, 1]\}$;
$Z \leftarrow 0$;
while *there exists x, an input element* **do**
  if $Z < h(x)$ **then**
    $Z \leftarrow h(x)$;
  end
end

Compute $\mathbb{E}(Z)$ and $\mathbb{V}ar(Z)$.

Exercise 2. *Prove Observation 7.2.1.*

Exercise 3. *Prove Lemma 7.2.2.*

(CS602) Applied Algorithms 30 Jan, 2015

Lecture 8: Pairwise Independence

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

In the last class, we considered the following problem: given an input stream $x_1, x_2, \ldots, x_n$, where each $x_i \in [m]$ and $\varepsilon, \delta > 0$ constants, design a randomized (ε, δ) algorithm for approximating the number of distinct elements using space $O(\log m)$. We gave two approaches for this. however, none of them will managed to achieve the correct space bounds.

Today we will introduce the notion of pairwise independent hash functions and use it in the second algorithm studied in the last class. This will give a randomized (ε, δ) algorithm for approximating the number of distinct elements using space $O(\log^2 m)$.

8.1 Modified Approach 2

The details regarding the definition of pairwise independence and their construction can be found in Lecture 0 (we will not rewrite them here).

In this lecture note we will simply observe some properties of Approach 2 from the last class and show one can use pairwise independence to bring the space usage from $O(m^2)$ to $O(\log^2 m)$.

Recall Approach 2 from last class.

Pick h uniformly randomly from $\mathcal{H}' = \{h : [m] \to \{0,1\}^K\}$;
$Z \leftarrow 0$;
while *there exists x, an input element* **do**
  if $Z <$ *#traling zeroes in* $h(x)$ **then**
    $Z \leftarrow h(x)$;
  end
end
Output 2^Z;

The only change we make to this algorithm is that we use a family of pairwise independent hash functions which is a subfamily of $\mathcal{H}$ of size $O(2^K m)$. Therefore, the number of bits needed to choose a function from this family is $O(K \log m) = O(log^2 m)$ if we choose K such that $m \leq 2^K \leq 2m$. Therefore, the upper bound on the space is enough. Now, let us see why the same analysis can go through for the modified algorithm.

The only property of the randomly chosen hash function we used was that $\forall \ell \in [K]$, $\mathbb{V}ar(\tilde{Y}_\ell) \leq \mathbb{E}(\tilde{Y}_\ell)$, where $\tilde{Y}_\ell$ is a sum of 0-1 random variables. From Exercise 9 of lecture 0, sum of pairwise independent 0-1 random variables also has this property. And hence, the same analysis as n Approach 2 will go through even for this modified approach.

(CS602) Applied Algorithms 23 Jan, 2015

Lecture 5: Second frequency moment, F_2

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

In the last class we gave a randomized (ε, δ) algorithm for approximating the number of distinct elements using space $O(\frac{1}{\varepsilon} \cdot \log\left(\frac{1}{\delta}\right) \cdot \log^2 m)$.

Today we will define the notion of frequency moments and give (ε, δ) algorithm for approximating the second frequency moment using space $O(\frac{1}{\varepsilon^2} \cdot \log\left(\frac{1}{\delta}\right) \cdot \log m)$.

5.1 Frequency Moments

Let $x_1, x_2, \ldots, x_n$ be input stream and for each $i \in [n]$ let $x_i \in [m]$. Let f_j denote the number of times the element $j \in [m]$ appears in the stream. The kth frequency moment is defined as follows:

$$F_k = \sum_{j \in [m]} f_j^k$$

As per this definition, F_0 is the number of distinct elements in the stream and F_1 is the length of the stream. We gave space efficient algorithms to approaximate these quantities over the last few lectures. Today we will give an algorithm to approximate F_2.

Pick h uniformly randomly from 4-wise independent family of functions
$\mathcal{F} = \{h : [m] \to \{\pm 1\}\}$;
Sum $\leftarrow 0$;
while *there exists x, an input element* **do**
  Sum $\leftarrow$ Sum $+ h(x)$;
end
Output $Z \leftarrow (\text{Sum})^2$;

We will first analyse the expected value of the output of the algorithm.

Lemma 5.1.1. $\mathbb{E}(Z) = F_2$

Proof.

$$\begin{aligned}
\mathbb{E}(Z) &= \mathbb{E}\left(\text{Sum}^2\right) \\
&= \mathbb{E}\left(\left(\sum_{x \in \text{stream}} h(x)\right)^2\right) && \text{(From the definition of Sum)} \\
&= \mathbb{E}\left(\left(\sum_{j \in [m]} f_j h(j)\right)^2\right) && \text{(From the definition of } h(x)\text{)}
\end{aligned}$$

From here we see that,

$$\begin{aligned}
\mathbb{E}(Z) &= \mathbb{E}\left(\sum_{j\in[m]} f_j^2 h(j)^2 + \sum_{j\neq\ell} f_j f_l h(j)h(\ell)\right) \\
&= \sum_{j\in[m]} f_j^2 \mathbb{E}(h(j)^2) + \sum_{j\neq\ell} f_j f_l \mathbb{E}(h(j)h(\ell)) && \text{(By linearity of expectation)} \\
&= \sum_{j\in[m]} f_j^2 \cdot 1 + \sum_{j\neq\ell} f_j f_l \cdot 0 && \text{(As } h(j)^2 = 1 \ \forall j \text{ and Pairwise independence of } \mathcal{F}) \\
&= F_2 && \text{(By the definition of } F_2)
\end{aligned}$$

□

Lemma 5.1.2. $\mathbb{V}ar(Z) \leq 2F_2^2$.

To reduce the variance even further, we use the *averaging trick*. If we run t copies of the same algorithm and let the output, say Z', be the average of the outputs of all the t algorithms then we will get the following:

Lemma 5.1.3. $\mathbb{E}(Z') = F_2$ *and* $\mathbb{V}ar(Z') \leq 2F_2^2/t$.

Now using Chebyshev's inequality we know that

$$\Pr\left[|Z' - \mathbb{E}(Z')| \geq \varepsilon F_2\right] \leq \frac{2F_2^2}{t\varepsilon^2 F_2^2} \leq 1/3 \text{ (for appropriate choice of } t)$$

We can further reduce the probability of error to be bounded above by δ by using the *median trick*.

We now argue the space bound. To compute Z, we need to keep track of the variable Sum, which can be stored in $O(\log n)$ space. The number of bits required to pick a random function from the family of 4-wise independent hash functions equals $\log(|\mathcal{F})$. It is known that for any family of functions $\mathcal{H} = \{h : [m] \to [k]\}$, there exists a subfamily $\mathcal{F} \subset \mathcal{H}$ of 4-wise independent hash functions of size $k^{\log m}$. Therefore, he number of bits required to pick a random function from the family of 4-wise independent hash functions equals $\log(|\mathcal{F}) = O(\log k \log m)$. As $k = 2$ here, we can choose a random function using $O(\log m)$ bits. As we saw in Lecture 1, the use of the averaging trick and the median trick along with this space bound we get that the randomized (ε, δ) approximation algorithm for F_2 uses space $O(\frac{1}{\varepsilon^2} \cdot \log\left(\frac{1}{\delta}\right) \cdot \log m)$.

This algorithm presented here is from a seminal paper by Alon, Matias and Szegedy.

5.2 Exercises

Exercise 1. *Prove Lemmas 5.1.2, 5.1.3.*

(CS602) Applied Algorithms 11 Feb, 2015

Lecture 11: Count-Min and Count sketches

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

In the last class we defined the notion of frequency moments and gave (ε, δ) approximation algorithm for approximating the second frequency moment using space $O(\frac{1}{\varepsilon^2} \cdot \log\left(\frac{1}{\delta}\right) \cdot \log m)$.

Today we will present two different randomized approximation algorithm which can compute all frequency moments using $O(\frac{1}{\varepsilon} \cdot \log\left(\frac{1}{\delta}\right) \cdot \log m)$ space, albeit their approximation guarantee will be weaker: i.e. approximation error will be *absolute* instead of *relative* (as in the previous classes).

11.1 Algorithm 1: Count-Min sketch algorithm

Recall some notation from the last time. Let $x_1, x_2, \ldots, x_n$ be the input stream and for each $i \in [n]$ let $x_i \in [m]$. Let f_j denote the number of times the element $j \in [m]$ appears in the stream. After the algorithm processes the entire stream, the task is to answer queries of the following form: *how many times has element "a" appeared in the stream?* or *what is the most frequently occuring element in the stream?* etc.

We now give the first algorithm.

Pick $h_1, h_2, \ldots, h_t$ uniformly randomly from pairwise independent family of functions $\mathcal{F} = \{h : [m] \to [k]\}$;
for $i = 1$ *to* t **do**
  for $j = 1$ *to* k **do**
    $C[i][h_i(j)] \leftarrow 0$;
  end
end
while *there exists* x*, an input element* **do**
  for $i = 1$ *to* t **do**
    $C[i][h_i(x)] \leftarrow C[i][h_i(x)] + 1$;
  end
end
On query a, output $\hat{f}_a \leftarrow min_i\{C[i][h_i(a)]\}$;

This algorithm is called the Count-Min sketch algorithm for obvious reasons and has been designed by Cormode and Muthukrishnan in 2003 [2].

In the algorithm, k, t are parameters which will be fixed later. In terms of these parameters, the space used by the algorithm can analysed easily. The number of bits needed to pick pairwise independent hash functions is $O(t \log m \log k)$ and the number of bits needed to store the table C is $O(tk \log n)$. Therefore, the total space requirement is $O(kt(\log m + \log n))$. This indicates that to minimize the space usage, we need to keep both parameters as low as possible.

Suppose $k \ll m$, then a random function from $[m]$ to $[k]$, maps around m/k elements of the domain to the same element in the range. Suppose the stream has a lot of occurences of one element, say a_1, while very few of the other, say a_2, and say under a random function they get mapped to the same bucket (which can happen with probability $1/k$), then the algorithm's estimate for the number of a_2 will be very erroneous. To counter this effect, the algorithm chooses t (pairwise) random functions. Now the probability that a_1 and a_2 collide under all t functions is small.

One thing we can quickly observe is that f_a, which is the number of times the element 'a' appears in the stream, is cerianly upper bounded by $\hat{f}_a$. This is because if under some h_i, no other element gets mapped to $h_i(a)$ then the count $C[i][h_i(a)]$ will be equal to f_a and all other $C[j][h_j(a)] \geq C[i][h_i(a)]$.

We will now prove the following lemma:

Lemma 11.1.1. *For every constant $\varepsilon, \delta > 0$, $\Pr\left[\left(\hat{f}_a - f_a\right) \geq \varepsilon n\right] \leq \delta$.*

Remark 11.1.2. *Ideally, we would have liked to prove that for every constant $\varepsilon, \delta > 0$, $\Pr\left[\left(\hat{f}_a - f_a\right) \geq \varepsilon f_a\right] \leq \delta$. That is, we would have liked to have a multiplicative approximation error, but we get an additive error. In fact, it is known that any randomized approximation algorithm (with multiplicative approximation error) for $\max_i \{f_i\}$ (which trivially is a restriction of the function we are computing here) requires space $\Omega(n)$ [1].*

Proof of Lemma 11.1.1: For $j \in [n] \setminus \{a\}$ let $Y_{i,j}$ denote the excess in the counter $C[i][h_i(a)]$. Then we have, $Y_{i,j} = \begin{cases} f_j & \text{if } h_i(a) = h_i(j) \quad \text{(with probability } 1/k\text{)} \\ 0 & \text{otherwise} \qquad\quad \text{(with probability } 1 - 1/k\text{)} \end{cases}$

Let $Y_i := \sum_j Y_{i,j}$. Then we have that $\hat{f}_a = min_i \{Y_i\} + f_a$. Therefore, $\hat{f}_a - f_a = min_i \{Y_i\}$. Therefore, we need to prove that for every constant $\varepsilon, \delta > 0$, $\Pr[min_i \{Y_i\} \geq \varepsilon n] \leq \delta$. That is, we need to prove that for every constant $\varepsilon, \delta > 0$, $\Pr[\forall i : Y_i \geq \varepsilon n] \leq \delta$. Now, note that all Y_is are independent, therefore, it suffices to prove that for every constant $\varepsilon, \delta > 0$, $\Pr[Y_i \geq \varepsilon n] \leq \delta^{1/t}$. We will analyze the expected value of Y_i in order to prove this. From the definition of $Y_{i,j}$ and linearity of expectation, we get that $\mathbb{E}(Y_i) = \sum_{j \in [m] \setminus \{a\}} \mathbb{E}(Y_{i,j}) = \frac{\sum_{j \in [m] \setminus \{a\}} f_j}{k} = \frac{n - f_a}{k}$. Therefore, using Markov's inequality we get that $\Pr[Y_i \geq \varepsilon n] \leq \frac{n - f_a}{k \varepsilon n}$. If k is chosen such that $k = \lceil \frac{2}{\varepsilon} \rceil$ then we get that $\Pr[Y_i \geq \varepsilon n] < \frac{1}{2}$. And if t is chosen to be $\lceil 2 \log \frac{1}{\delta} \rceil$, we get the lemma.

□

11.2 Count Sketch

In this section, we present a small modification of the algorithm presented in the previous section. We will analyze the algorithm in this lecture and compare and contrast the performance of the two algorithms in the next lecture.

Like in the previous section, let for $j \in [n] \setminus \{a\}$ let $Y_{i,j}$ denote the excess in the counter $C[i][h_i(a)]$.

Then we have, $Y_{i,j} = \begin{cases} f_j & \text{if } h_i(a) = h_i(j) \text{ and } g_i(j) = +1 \quad \text{(with probability } 1/2k\text{)} \\ -f_j & \text{if } h_i(a) = h_i(j) \text{ and } g_i(j) = -1 \quad \text{(with probability } 1/2k\text{)} \\ 0 & \text{otherwise} \qquad\qquad\qquad\quad \text{(with probability } 1 - 1/k\text{)} \end{cases}$

Pick $h_1, h_2, \ldots, h_t$ uniformly randomly from pairwise independent family of functions $\mathcal{F} = \{f : [m] \to [k]\}$ and pick $g_1, g_2, \ldots, g_t$ uniformly randomly from pairwise independent family of functions $\mathcal{G} = \{g : [m] \to \{\pm\}\}$;
for $i = 1$ *to* t **do**
  for $j = 1$ *to* k **do**
    $C[i][h_i(j)] \leftarrow 0$;
  end
end
while *there exists* x*, an input element* **do**
  for $i = 1$ *to* t **do**
    $C[i][h_i(x)] \leftarrow C[i][h_i(x)] + g_i(x)$;
  end
end
On query a, output $\text{Median}_{1 \le i \le t} \{g_i(a) C[i][h_i(a)]\}$;

Let $Y_i := \sum_j Y_{i,j}$. The output of the algorithm and the Y_is are related in the following way: Y_i counts the total error terms in $C[i][h_i(a)]$. Therefore, $C[i][h_i(a)] = Y_i + g_i(a) \cdot f_a$. In order to bound the error in the output, let us first compute the expected value of Y_i and variance of Y_i.

Lemma 11.2.1. *For every* $1 \le i \le t$ $\mathbb{E}(Y_i) = 0$ *and* $\mathbb{V}ar(Y_i) = \frac{\sum_{j \in [m] \setminus \{a\}} f_j^2}{k}$.

Proof.

$$\begin{aligned}
\mathbb{E}(Y_i) &= \mathbb{E}\left(\sum_{j \in [m] \setminus \{a\}} Y_{i,j}\right) && \text{(By the definition of } Y_i\text{)} \\
&= \sum_{j \in [m] \setminus \{a\}} \mathbb{E}(Y_{i,j}) && \text{(By linearity of expectation)} \\
&= \sum_{j \in [m] \setminus \{a\}} \frac{f_j}{2k} + \frac{-f_j}{2k} && \text{(By the definition of } Y_{i,j}\text{)} \\
&= 0
\end{aligned}$$

$$
\begin{aligned}
\mathbb{E}(Y_i^2) &= \mathbb{E}\left(\sum_{j\in[m]\setminus\{a\}} Y_{i,j}^2 + \sum_{j\neq j'\in[m]\setminus\{a\}} Y_{i,j}Y_{i,j'}\right) && \text{(By the definition of } Y_i\text{)}\\
&= \sum_{j\in[m]\setminus\{a\}} \mathbb{E}(Y_{i,j}^2) + \sum_{j\neq j'\in[m]\setminus\{a\}} \mathbb{E}(Y_{i,j}Y_{i,j'}) && \text{(By linearity of expectation)}\\
&= \sum_{j\in[m]\setminus\{a\}} \frac{f_j^2}{2k} + \frac{(-f_j)^2}{2k} + \sum_{j\neq j'\in[m]\setminus\{a\}} \mathbb{E}(Y_{i,j})\mathbb{E}(Y_{i,j'}) && \text{(By the pairwise independence of } Y_{i,j}\text{)}\\
&= \frac{\sum_{j\in[m]\setminus\{a\}} f_j^2}{k}
\end{aligned}
$$

□

In the next class we will bound the expectation and variance of the output of the algorithm using Lemma 11.2.1.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. ACM.

[2] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

11.3 Exercises

Exercise 1. *Make approrpiate modifications to the algorithms presented here so that they work in the turnstile model. Work out all the calculations for the modified algorithms. Read about the turnstile model at:*
http://en.wikipedia.org/wiki/Streaming_algorithm#Models

(CS602) Applied Algorithms 13 Feb, 2015

Lecture 12: Count-Min and Count sketches

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

In the last class we studied two different randomized approximation algorithm for computing all frequency moments using $O(\frac{1}{\varepsilon} \cdot \log\left(\frac{1}{\delta}\right) \cdot \log m)$ space and additive approximation error. We fully analyzed the first algorithm. Recall that the algorithm used a sketch which is known as Count-Min sketch, which was developed by Cormode and Muthukrishnana [2]. Towards the end of the last class, we presented the second algorithm and we were in the process of analysing the second algorithm. The second algorithm was developed by Charikar, Chen and Farach-Colton [1]. It also uses a sketch, which is known as Count sketch.

Both these sketching algorithms have found a plethora of applications in varied areas of computer science including compressed sensing, natural language processing, databses, and networking. (See for example: `https://sites.google.com/site/countminsketch/`)

We will finish the analysis of the second algorithm in this class.

12.1 Recalling the Count sketch algorithm

Let us start by recalling the algorithm we presented last time, the notation we introduced, and the Lemmas we proved.

Pick $h_1, h_2, \ldots, h_t$ uniformly randomly from pairwise independent family of functions $\mathcal{F} = \{f : [m] \to [k]\}$ and pick $g_1, g_2, \ldots, g_t$ uniformly randomly from pairwise independent family of functions $\mathcal{G} = \{g : [m] \to \{\pm\}\}$;
for $i = 1$ *to* t **do**
  for $j = 1$ *to* k **do**
    $C[i][h_i(j)] \leftarrow 0$;
  end
end
while *there exists* x*, an input element* **do**
  for $i = 1$ *to* t **do**
    $C[i][h_i(x)] \leftarrow C[i][h_i(x)] + g_i(x)$;
  end
end
On query a, output $\text{Median}_{1 \le i \le t} \{g_i(a) C[i][h_i(a)]\}$;

Like in the previous section, let for $j \in [n] \setminus \{a\}$, $Y_{i,j}$ denote the excess in the counter $C[i][h_i(a)]$.

Then we have, $Y_{i,j} = \begin{cases} f_j & \text{if } h_i(a) = h_i(j) \text{ and } g_i(j) = +1 \quad \text{(with probability } 1/2k\text{)} \\ -f_j & \text{if } h_i(a) = h_i(j) \text{ and } g_i(j) = -1 \quad \text{(with probability } 1/2k\text{)} \\ 0 & \text{otherwise} \qquad\qquad\qquad\quad \text{(with probability } 1 - 1/k\text{)} \end{cases}$

Let $Y_i := \sum_j Y_{i,j}$. The output of the algorithm and the Y_is are related in the following way: Y_i counts the total error terms in $C[i][h_i(a)]$. Therefore, $C[i][h_i(a)] = Y_i + g_i(a) \cdot f_a$.

In order to bound the error in the output, let us first compute the expected value of Y_i and variance of Y_i.

Lemma 6.2.1. For every $1 \leq i \leq t$, $\mathbb{E}(Y_i) = 0$ and $\mathbb{V}ar(Y_i) = \frac{\sum_{j \in [m] \setminus \{a\}} f_j^2}{k}$.

The proof of this emma was presented in the last lecture note.

12.2 Analysis of the algorithm

We now use Lemma 6.2.1 to analyze the performance of the algorithm. Let $\hat{\hat{f}}_a$ denote the output of the algorithm. Therefore, $\hat{\hat{f}}_a = \text{Median}\{\hat{\hat{f}}_{a,i}\}$, where $\hat{\hat{f}}_{a,i} = g_i(a) \times (Y_i + g_i(a) \cdot f_a)$. We will first compute the expectation and variance of $\hat{\hat{f}}_{a,i}$.

Lemma 12.2.1. *For every $1 \leq i \leq t$, $\mathbb{E}(\hat{\hat{f}}_{a,i}) = f_a$ and $\mathbb{V}ar(\hat{\hat{f}}_{a,i}) = \frac{\sum_{j \in [m] \setminus \{a\}} f_j^2}{k}$*

Proof.

$$\begin{aligned}
\mathbb{E}(\hat{\hat{f}}_{a,i}) &= \mathbb{E}(g_i(a) \times (Y_i + g_i(a) \cdot f_a)) \\
&= \mathbb{E}(g_i(a)Y_i + f_a) && (\text{As } g_i(a)^2 = 1) \\
&= \mathbb{E}(g_i(a)Y_i) + f_a \\
&= f_a + \sum_{j=1}^{k} \mathbb{E}(g_i(a)Y_{i,j}) \\
&= f_a + \sum_{j=1}^{k} \frac{+f_j}{4k} + \frac{-f_j}{4k} + \frac{+f_j}{4k} + \frac{-f_j}{4k} \\
&= 0
\end{aligned}$$

Remark 12.2.2. *Note that as an intermediate step, the above proof also proved that $\mathbb{E}(g_i(a)Y_i) = 0$. We will use this fact in the analysis of the variance.*

$$\begin{aligned}
\mathbb{E}(\hat{\hat{f}}_{a,i}^2) &= \mathbb{E}\left((g_i(a)Y_i + f_a)^2\right) && (\text{From Step 2 above}) \\
&= f_a^2 + \mathbb{E}\left(g_i(a)^2 Y_i^2 + 2 f_a g_i(a) Y_i\right) \\
&= f_a^2 + \mathbb{E}\left(Y_i^2\right) + 2 f_a \mathbb{E}(g_i(a)Y_i) && (\text{As } g_i(a)^2 = 1) \\
&= f_a^2 + \mathbb{E}\left(Y_i^2\right) + 0 && (\text{By Remark 12.2.2}) \\
&= f_a^2 + \frac{\sum_{j \in [m] \setminus \{a\}} f_j^2}{k} && (\text{By proof of Lemma 6.2.1})
\end{aligned}$$

Therefore, we get $\mathbb{V}ar(\hat{\hat{f}}_{a,i}) = \frac{\sum_{j \in [m] \setminus \{a\}} f_j^2}{k}$ □

By using Chebyshev's inequality, we get $\Pr\left[|\hat{\hat{f}}_{a,i} - f_a| \geq \varepsilon\sqrt{F_2}\right] \leq \frac{\sum_{j\in[m]\setminus\{a\}} f_j^2}{k\varepsilon^2 F_2}$, where $F_2 = \sum_{j\in[m]} f_j^2$. By choosing k appropriately, we can bound this probability by $1/3$, that is $\Pr\left[|\hat{\hat{f}}_{a,i} - f_a| \geq \varepsilon\sqrt{F_2}\right] \leq 1/3$. Now by the standard median trick, by choosing $t = O\left(\log \frac{1}{\delta}\right)$, we get $\Pr\left[|\hat{\hat{f}}_a - f_a| \geq \varepsilon\sqrt{F_2}\right] \leq \delta$ for any constant $\delta > 0$.

12.3 Comparison of the two algorithms

Recall that in the last lecture note we proved that $\Pr\left[|\hat{f}_a - f_a| \geq \varepsilon F_1\right] \leq \delta$ for any constant $\delta > 0$. That is, in the Count-Min algorithm, the additive approximation error was proportional to F_1 where as here it is proportional to $\sqrt{F_2}$. However, the dependence on ε is $1/\varepsilon$ for Count-Min and $1/\varepsilon^2$ for Count sketch.

References

[1] Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3 – 15, 2004. ¡ce:title¿Automata, Languages and Programming¡/ce:title¿.

[2] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

12.4 Exercises

Exercise 1. *In the algorithm presented here, say the last line is changed from*

$$\text{“On query } a\text{, output Median}_{1\leq i\leq t}\{g_i(a)C[i][h_i(a)]\}\text{”}$$

to

$$\text{“On query } a\text{, output } \frac{\sum_{i=1}^{t}\{g_i(a)C[i][h_i(a)]\}}{t}$$

The rest of the algorithm is kept as it is. Analyze the performance of this modified algorithm.

(CS602) Applied Algorithms 06 Feb, 2015

Lecture 9: Sampling based approach for distinct elements

Lecturer: Nutan Limaye *Scribe: Nutan Limaye*

In the last class we completed the analysis of Count sketch algorithm. Today we will give a sampling based approach for estimating distinct elements.

Recall the distinct element problem deals with given a stream of data $x_1, x_2, \ldots, x_n$, where for all i $x_i \in [m]$, counting the number of distinct elements in the stream. As a first step towards solving this problem using sampling, we will look at the restricted version of the same problem and design a sampling algorithm for it. We call this version, the gap version of the problem, $\mathsf{GapDist}_k$.

Given: $\tilde{x} = x_1, x_2, \ldots, x_n$, where for all $1 \leq i \leq n$, $x_i \in [m]$, and $k \in \mathbb{N}$
Output "Yes" if the number of distinct elements in $\tilde{x}$ is $> 2^{k+2}$
"No" if the number of distinct elements in $\tilde{x}$ is $< 2^{k-2}$

9.1 Naive Sampling algorithm for $\mathsf{GapDist}_k$

We first give an algorithm which uses (in the worst case) $O(m)$ number of *independent* random bits. Later we show how one can raplace independent random bits by pairwise random bits.

Pick every element of $[m]$ into the set S with probability $\frac{1}{2^k}$;
Sum $\leftarrow 0$;
while *there exists x, an input element* **do**
  if $x \in S$ **then**
    Sum $\leftarrow$ Sum $+1$;
  end
end
Output "Yes" iff Sum > 0;

Algorithm 1: Algorithm with independent random bits

We now argue the correctness the above algorithm and bound its error probability. Let $\tilde{x}$ be the given input. Let D denote the set of distinct elements in $\tilde{x}$. Let F_0 denote $|D|$.

Suppose $F_0 < 2^{k-2}$. Then the probability that the algorithm makes an error is:

$$\begin{aligned}
\Pr[\text{Algorithm makes an error}] &= \Pr[\text{Sum } > 0] \\
&= \Pr[\exists x \in D \text{ s.t. } x \in S] \\
&\leq \sum_{x \in D} \Pr[x \in S] && \text{(By union bound)} \\
&= \frac{|D|}{2^k} \\
&< \frac{1}{4} && \text{(By our assumption that } |D| < 2^{k-2}\text{)}
\end{aligned}$$

Suppose $F_0 > 2^{k+2}$. Then the probability that the algorithm makes an error is:

$$\begin{aligned}
\Pr[\text{Algorithm makes an error}] &= \Pr[\text{Sum } = 0] \\
&= \Pr[\forall x \in D : x \notin S] \\
&\leq \prod_{x \in D} \Pr[x \notin S] && \text{(As the samples are independent)} \\
&= \left(1 - \frac{1}{2^k}\right)^{2^{k+2}} && \text{(By our assumption that } |D| > 2^{k+2}\text{)} \\
&< \left(\frac{1}{e}\right)^4 && \text{(Using } \left(1 - \frac{1}{x}\right)^x = \frac{1}{e}\text{)}
\end{aligned}$$

By the above calculations, we get that the algorithm correctly decides $\mathsf{GapDist}_k$ with probability at least 3/4.

Note that, in the above calculations we used the fact that our samples are independent. Let us do the calculations once again, but in such a way that the analysis will go through even if we draw samples using pairwise independence. Let X_j be a 0-1 random variable defined as follows: $X_j = 1$ if $j \in S$ and $X_j = 0$ otherwise. Let $X = \sum_{j \in D} X_j$. Note that $\Pr[X_j = 1] = \frac{1}{2^k}$ for all j. Therefore, $\mathbb{E}(X_j) = \frac{1}{2^k}$ and $\mathbb{E}(X) = \frac{|D|}{2^k}$. Suppose X_js are either purely independent or pairwise independent, we know that $\mathbb{V}ar(X) \leq \mathbb{E}(X)$ (by the property of pairwise independent random variables).

Suppose $F_0 < 2^{k-2}$. Then the probability that the algorithm makes an error is:

$$\begin{aligned}
\Pr[\text{Algorithm makes an error}] &= \Pr[\text{Sum } > 0] \\
&= \Pr[X > 0] \\
&= \Pr[X \geq 1] \\
&\leq \frac{|D|}{2^k} && \text{(By Markov's inequality)} \\
&< \frac{1}{4} && \text{(By our assumption that } |D| < 2^{k-2}\text{)}
\end{aligned}$$

On the other hand, suppose $F_0 > 2^{k+2}$. Then the probability that the algorithm makes an error is:

$$\begin{aligned}
\Pr[\text{Algorithm makes an error}] &= \Pr[X = 0] \\
&\leq \Pr[|X - \mathbb{E}(X)| \geq \mathbb{E}(X)] \\
&\leq \frac{\mathbb{V}ar(X)}{\mathbb{E}(X)^2} && \text{(By Chebyshev's inequality)} \\
&\leq \frac{1}{\mathbb{E}(X)} && (\mathbb{V}ar(X) \leq \mathbb{E}(X)) \\
&< \frac{1}{4} && \text{(Using } |D| > 2^{k+2} \text{ and } \mathbb{E}(X) = \frac{|D|}{2^k}\text{)}
\end{aligned}$$

Once again, by the above calculations, we get that the algorithm correctly decides $\mathsf{GapDist}_k$ with probability at least 3/4.

By using standard Chernoff argument, we can bring down the error probability down to δ using at most $O(\log \frac{1}{\delta})$ bits.

Now, we change the algorithm so that independent samples can now be changed by pairwise independent samples.

Pick h from a family of pairwise independent random functions
$\mathcal{F} = \{h : [m] \to \{0,1\}^k\}$.;
Sum $\leftarrow 0$;
while *there exists x, an input element* **do**
  if $h(x) = 0^k$ **then**
    Sum $\leftarrow$ Sum $+1$;
  end
end
Output "Yes" iff Sum > 0;

Algorithm 2: Algorithm with pairwise independent random variables

For the analysis, we define $X_j = 1$ iff $h(j) = 0^k$ and $X = \sum_{j \in D} X_j$ as before. The analysis of the algorithm is the same as our second analysis.

Let $\mathcal{A}_\delta^k$ denote this randomized algorithm for $\mathsf{GapDist}_k$ with error at most δ. In the next section we use this algorithm to approximate F_0.

9.2 Approximating F_0 using $\mathcal{A}_\delta^k$

In this section we will use $\mathcal{A}_\delta^1, \mathcal{A}_\delta^2, \ldots, \mathcal{A}_\delta^{\lceil \log m \rceil}$ to get an 8-approximation for F_0. In the exercise, you are asked to improve it to $(1+\varepsilon)$-approximation.

Let $\mathcal{A}_{\delta'}$ be the following algorithm:

for $i = \lceil \log m \rceil$ *downto 1* **do**
  if $\mathcal{A}_\delta^i$ *outputs 0* **then**
    next i;
  end
  else
    Output 2^i;
  end
end

Suppose on some fixed input $\mathcal{A}_\delta^i$ outputs 0 for all $i \geq j$ but $\mathcal{A}_\delta^{j-1}$ outputs 1 and suppose also that all the answers are correct. Then this tells us that the answer must be certainly smaller than 2^{j+2} and definitely more than 2^{j-3}. Therefore, if the algorithm outputs 2^j then it will be 8-approximation. But unfortunately, not all answers may be correct. $\Pr[A_{\delta'} \text{ makes an error}] \leq \Pr\left[\exists A_\delta^i \text{ makes an error}\right] \leq \lceil \log m \rceil \cdot \delta$. By making $\delta = \frac{\delta'}{\lceil \log m \rceil}$, we can make the error bounded by δ'.

9.3 Space analysis of $\mathcal{A}_\delta^i$ and $\mathcal{A}_{\delta'}$

To pick a random function from a family of pairwise independent functions, we need $O(k \cdot \log m)$ bits and to store 'Sum' we need $O(\log n)$ bits. To bring down the overall error to δ, we need to run $O(\log(\frac{1}{\delta}))$ copies of Algorithm 2. Therefore, total number of bits stored by A_δ^i is $O(\log(\frac{1}{\delta}) \cdot (k \cdot \log m + \log n))$. Say $s = O(\log(\frac{1}{\delta}) \cdot (k \cdot \log m + \log n))$.

Now, the algorithm $\mathcal{A}_{\delta'}$ simultaeously runs $\lceil \log m \rceil$ copies of $\mathcal{A}_\delta^i$, one for every $1 \leq i \leq \lceil \log m \rceil$. This takes space $O(\lceil \log m \rceil \cdot s)$. Finally, for the error to be bounded by δ', we need to set $\delta = \frac{\delta'}{\lceil \log m \rceil}$. Putting it together, we get that the space used by $\mathcal{A}_{\delta'}$ can be bounded by $O\left(\lceil \log m \rceil \cdot \log(\frac{\lceil \log m \rceil}{\delta'}) \cdot (k \cdot \log m + \log n)\right)$. This gives us an 8-approximation for F_0 with probability $1 - \delta'$.

9.4 Exercises

Exercise 1. *Modify Algorithm 2, $\mathcal{A}_\delta^i$ and $\mathcal{A}_{\delta'}$ to obtain for every $\varepsilon > 0$, $(1+\varepsilon)$-approximation algorithm for F_0. Analyze the space used by your algorithm.*