1 Introduction

The class is about algorithms. Historically, the golden standard for algorithms has always been $O(n)$ (linear time and space). The issue is that data we work with is much larger than computer resources we have. We need to think of algorithms in models which limit the access to data. These limitations can be either limited time and/or limited space.

1.0.1 A common scenario: limited space

Suppose we have a router with a lot of traffic passing through it. This could be a router which receives some information, along with a source and an IP address. We would like to be able to compute certain statistics with the information that passes through the router. For example, we would like to know the number of times a certain IP address makes a request. Another example could be to compute the IP address with the maximum number of requests.

The amount of information which passes through the router greatly exceeds the available storage in the router. Therefore, we cannot simply store copies of data passing through the router.

Many of these tasks are simply impossible to compute. We relax the guarantees. Instead of requiring exact computations, we can require approximate computations. One way to approximate is to say:

$$\text{true answer} \leq \text{output} \leq \alpha \times \text{true answer} \quad (1)$$

We think of $\alpha = 1 + \epsilon$, where $\epsilon$ is very small. Another way to relax the guarantees is to require the approximations holds with a certain probability. We will say, with probability at least $1 - \delta$, Equation 1 holds. We think of $\delta$ as being a very small number, so the approximations hold often.

The class will include:

- Streaming algorithms. Data passes through an algorithm. The algorithm sees all of it, but cannot store it.
- Dimension reduction, sketching. We work with distributed data, where no processor has access to the entire data. We need to compute small summaries of the data in order to perform computations on the entire dataset.
- High dimensional nearest neighbor search. We want to find the most similar objects in a database, where the dimensionality of the individual objects is very large.
- Sampling, property testing. We cannot even look at all the data. Instead, we will need to take small pieces of the data to deduce overall properties of the data set.
Parallel algorithms. Many processors will partake in the computation, but no processor will have access to all the data.

Grading:

- Scribing (2-3) students (%10).
- 5 Homeworks. First assignment is worth %7. The rest of the assignments are worth %12 each. There are 5 total days of lateness (120 hours) to use as you wish.
- Project research based (%35). Projects can be to solve an open problem or make progress in the field, to apply algorithms to your research area, or to survey some papers in the field and synthesize the results.

1.1 Probability Review

These are a few probability tools we will use in the analysis of algorithms.

We will let $X$ be a random variable.

- **Expectation:** For discrete random variables, $E[X] = \sum_a a \Pr[X = a]$. For continuous random variables, $E[X] = \int a\phi(a)da$ where $\phi$ is the probability density function of $X$.

- **Linearity of expectation:** $E[X + Y] = E[X] + E[Y]$.

- **Markov Inequality:** Suppose we know $X \geq 0$. For all $\lambda > 0$, $\Pr[X > \lambda] \leq \frac{E[X]}{\lambda}$.

- **Variance:** $\text{var}[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$.

- **Chebyshev inequality:** For all $\lambda > 0$, $\Pr[|X - E[X]| > \lambda] \leq \frac{\text{var}[X]}{\lambda^2}$. A corollary is that $X = E[X] \pm \sqrt{10\text{var}X}$ with probability at least $\frac{0}{10}$.

1.2 Problem: counting

The problem could be phrased as follows. In the streaming model, where the data passes through an algorithm, we want to count the number of times an event happens. For example, in the above scenario, we could ask to count the number of times a certain IP address makes a request.

Let’s say we know $n$ is the upper bound on the count. So the event happens at most $n$ times. How much space does it take to compute the count? Well, we need to store $n$, which we can do in $O(\log n)$ bits.

Can we do better? In general, no. But we can if we approximate. We will see an algorithm which uses $O(\log \log n)$ bits to approximate the count.

1.2.1 Morris Algorithm

We maintain a counter $X$.

1. Initialize $X = 0$. 


2. When the event happens: Update $X \leftarrow X + 1$ with probability $\frac{1}{2^X}$ and do nothing with probability $1 - \frac{1}{2^X}$.

3. When done, output $2^X - 1$.

First, we will prove the algorithm works.

The strategy to proving the algorithm works will be to:

1. Establish an estimator for a given quantity we want.
2. Prove that the expected value of the estimator is what it should be.
3. Compute the variance.
4. Use the Chebyshev inequality with the variance computed to say that the estimator is usually close to its expected value.

In our case, the value we want to estimate is the number of times an event happens. Call this value $n$. The estimator we have in the algorithm is $2^X - 1$. So intuitively, we will show that the algorithm is correct by showing that very often, $2^X - 1 \approx n$.

In order to prove that at the end of the algorithm $E[2^X - 1] = n$, we will analyze how $X$ evolves as the event happens. So we’ll let $X_0 = 0$, be the original value. In general, $X_i$ is the value of $X$ after $i$ events. Finally, we will have $X = X_n$.

**Claim 1.** Let $X_n$ be the random variable equal to $X$ after $n$ increments. Then $E[2^{X_n} - 1] = n$.

**Proof.** We will write $E[2^{X_n}]$ in terms of $E[2^{X_{n-1}}]$ and then use induction to get our desired claim. In the base case, $E[2^{X_0}] = 1$.

$$E[2^{X_n}] = \sum_i 2^i \Pr[X_n = i]$$

$$\Pr[X_n = i] = \Pr[X \text{ is incremented and } X_{n-1} = i - 1] + \Pr[X \text{ is not incremented and } X_{n-1} = i]$$

$$= \frac{1}{2^{i-1}} \Pr[X_{n-1} = i - 1] + (1 - \frac{1}{2^i}) \Pr[X_{n-1} = i]$$

Since the probability that $X$ is incremented at step $n$ is $\frac{1}{2^{X_{n-1}}}$. So

$$E[2^{X_n}] = \sum_i 2^i \left( \frac{1}{2^{i-1}} \Pr[X_{n-1} = i - 1] + (1 - \frac{1}{2^i}) \Pr[X_{n-1} = i] \right)$$

$$= \sum_i \left( 2 \Pr[X_{n-1} = i - 1] + 2^i \Pr[X_{n-1} = i] - \Pr[X_{n-1} = i] \right)$$

$$= \sum_i 2^i \Pr[X_{n-1} = i] + 2 \sum_i \Pr[X_{n-1} = i - 1] - \sum_i \Pr[X_{n-1} = i]$$

$$= E[2^{X_{n-1}}] + 2 - 1$$

$$= E[2^{X_{n-1}}] + 1$$
So by induction, and noting that the \( X_0 \) case adds 1,
\[
\mathbb{E}[2^{X_n}] = n + 1
\]
(10)

This completes the proof.

Claim 2. \( \text{var}[2^X - 1] \leq \frac{3n(n-1)}{2} + 2 = O(n^2) \)

**Proof.** The first thing to see is that \( \text{var}[2^X - 1] = \text{var}[2^X] \). So we will compute that.
\[
\text{var}[2^{X_n}] = \mathbb{E}[2^{2X_n}] - \mathbb{E}[2^{X_n}]^2
\]
(11)

\[
\leq \mathbb{E}[2^{2X_n}]
\]
(12)
since \( \mathbb{E}[2^{2X_n}] \geq 0 \).
\[
\mathbb{E}[2^{2X_n}] = \sum_i 2^{2i} \Pr[X_n = i]
\]
(13)

\[
= \sum_i 2^{2i} \left( \frac{1}{2^{i-1}} \Pr[X_{n-1} = i - 1] + (1 - \frac{1}{2^i}) \Pr[X_{n-1} = i] \right)
\]
(14)

\[
= \sum_i 2^{i+1} \Pr[X_{n-1} = i - 1] + \sum_i 2^{2i} \Pr[X_{n-1} = i] - \sum_i 2^i \Pr[X_{n-1} = i]
\]
(15)

\[
= \mathbb{E}[2^{2X_{n-1}}] + 4 \sum_i 2^{i-1} \Pr[X_{n-1} = i - 1] - \sum_i 2^i \Pr[X_{n-1} = i]
\]
(16)

\[
= \mathbb{E}[2^{2X_{n-1}}] + 3 \sum_i 2^i \Pr[X_{n-1} = i]
\]
(17)

\[
= \mathbb{E}[2^{2X_{n-1}}] + 3 \mathbb{E}[2^{X_{n-1}}]
\]
(18)

So by induction and noting that \( \mathbb{E}[2^{2X_0}] = 1 \), it follows that \( \mathbb{E}[2^{2X_n}] = 3 \sum_{i=0}^{n-1} \mathbb{E}[2^{X_i}] + 1 \). By Claim 1, this value is at most \( \frac{3n(n-1)}{2} + 2 \).

Now we apply the Chebyshev bound. Although we don’t have the variance computed exactly, an upper bound on the variance will still give us a Chebyshev bound. Our estimator, \( 2^X - 1 \) is close to its expectation \( \pm \sqrt{O(n^2)} \) with high constant probability. Specifically, we can say that with probability at least \( \frac{9}{10} \), we have that
\[
2^X - 1 = n \pm O(n)
\]
This is good on the upper bound, but not good enough. We usually want an \( \epsilon \) approximation, for small \( \epsilon \) which we pick.

1.3 Morris+

In order to achieve a \( 1 + \epsilon \)-approximation. We will repeat Morris’s algorithm many times, and then take the average of all the outputs. So in Morris+,

1. Run \( k \) independent copies of Morris’s algorithm. Keeping \( (X_1, \ldots, X_k) \).
2. At the end, output \( \frac{1}{k} \sum_{i=1}^{k} (2^{Y_i} - 1) \).

For the analysis, we’ll let \( Y_i = 2^{X_i} - 1 \) and \( Y = \frac{1}{k} \sum Y_i \). This will allow us to leverage all the analysis done for the regular Morris algorithm.

**Claim 3.** \( E[Y] = n \).

**Proof.** The proof follows from linearity of expectation.

\[
E[Y] = E\left[\frac{1}{k} \sum Y_i\right] \tag{19}
\]

\[
= \frac{1}{k} \sum_{i=1}^{k} E[Y_i] \tag{20}
\]

\[
= \frac{1}{k} \sum_{i=1}^{k} n \tag{21}
\]

\[
= n \tag{22}
\]

\( \square \)

**Claim 4.** \( \text{var}[Y] \leq \frac{1}{k} O(n^2) \)

**Proof.**

\[
\text{var}[Y] = \text{var}\left[\frac{1}{k} \sum Y_i\right] \tag{23}
\]

\[
\leq \sum_{i=1}^{k} \text{var}[Y_i/k] \tag{24}
\]

\[
\leq \sum_{i=1}^{k} \frac{1}{k^2} O(n^2) \tag{25}
\]

\[
\leq O(n^2/k) \tag{26}
\]

\( \square \)

Again, we apply the Chebyshev inequality. We have that with constant probability,

\[
Y = n \pm \frac{1}{\sqrt{k}} O(n)
\]

Letting \( k = O(\frac{1}{\epsilon^2}) \) gives us the \((1 + \epsilon)\)-approximation.

But this algorithm only works with high constant probability, for example, only \( \frac{9}{10} \) probability.

### 1.3.1 Morris++

In order to achieve the \((1 + \epsilon)\)-approximation from Morris+ with probability \( 1 - \delta \), we can do run Morris+ with \( k = O(\frac{1}{\epsilon^2 \delta}) \). Using the Chebyshev bound, we have that

\[
Y = (1 \pm \epsilon)n
\]
except with failure probability at most \( \frac{\text{var}[Y]}{e^2 n^2} \leq O \left( \frac{n^2 \epsilon^2 \delta}{e^2 n^2} \right) \). So with the appropriate constants, we achieve the approximation with probability at least \( 1 - \delta \).

In fact, we can even do total space \( O \left( \frac{1}{\epsilon^2} \log(\frac{1}{\delta}) \log \log n \right) \).

The idea is to take \( O(\log(1/\delta)) \) counters like \( Y \) and take the median of the \( Y \)'s. For the analysis, we will need Chernoff/Hoeffding bounds. We will describe these bounds, and how to apply them, in the next lecture.