1 Lecture 1

Inequalities and Sampling

Theorem 1.1. Markov's Inequality $\mathbb{P}(X \geq t)t \leq \mathbb{E}[X]$. Easy to see why. Note that this is only true for non-negative random variables.

Remark 1.2. This inequality is sharp, meaning that it cannot be further tightened. For example, let X be a random variable that takes on 1 with probability $1-a$ and 0 with probability 0. Then $\mathbb{E}[X] = (1)(1-a) + 0(a) \geq (1)(\mathbb{P}[X \geq 1]) = 1-a$. In this case, the $\geq$ is, in fact, an equality $=$.

Theorem 1.3. Chebyshev's Inequality It is UNRESOLVED why we need for $\text{Var}(X) < \infty$

$$\mathbb{P}[(X - \mathbb{E}[X])^2 \geq t^2\sigma_x^2] \leq \frac{1}{t^2}$$

Note that this statement is also that

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq t] = \mathbb{P}[(X - \mathbb{E}[X])^2 \geq t^2] \leq \frac{\sigma_x^2}{t^2}$$

Proof.

$$\begin{aligned}
\mathbb{P}(X \geq t)t &\leq \mathbb{E}[X] \\
\mathbb{P}(X^2 \geq t^2)t^2 &\leq \mathbb{E}[X^2] \\
\mathbb{P}((X-\mu_x)^2 \geq (t\sigma_x)^2)(t\sigma_x)^2 &\leq \mathbb{E}[(X-\mu_x)^2] \\
\mathbb{P}((X-\mu_x)^2 \geq (t\sigma_x)^2) &\leq \frac{\mathbb{E}[(X-\mu_x)^2]}{(t\sigma_x)^2} = \frac{1}{t^2}
\end{aligned}$$

□

2 Lecture 2

Introduction to Randomization UNRESOLVED (MISSING MATRIX MULT AND TRIANGLES)

Problem Expected Vertex Count. Suppose a graph has n vertices and we randomly delete each vertex with probability 1/2. The expected number of vertices is:

Let w be an array consisting only of 0s and 1s that is as long as there are vertices in G. Assume $w[x] = 1$ if the graph obtained from G – let us call it H – still has the xth vertex. It is easy to see that the probability of H having those vertices denoted by w is $\frac{1}{2^n}$. Let $X(w)$ be the number of vertices that H would have, were it represented by w. Let X be the number of vertices H has.

Then $\mathbb{E}[X] = \sum_{w \in \Omega} X(w)\frac{1}{2^n}$.

The right hand side of the inequality can be replaced by

$$\sum_{k=0}^{n} \underbrace{(\text{The number of ways to get } k \text{ vertices multiplied by } k \text{ vertices})}_{\alpha} \frac{1}{2^n}$$

. Think about α for a second: for a given k , α is really $\binom{n}{k}(k)$.

Problem Expected Edge Count. Suppose a graph has n vertices and we randomly delete each vertex with probability 1/2. The expected number of edges is:

An edge e is a connection between two vertices u and v. Thus e will continue to exist in H if and only if uv exists in H; that is, u and v must not be deleted, the probability of which is 1/4. Let E be the set of edges in G. Then the number of edges in H is a random variable X whose value is

$$\sum_{(u,v)\in E} X_[(u,v) \in E_H]$$

where $X_{(u,v)\in E_H}$ is the binary random variable that takes on 1 if $(u,v) \in E_H$ and 0 otherwise. Its expectation is

$$\sum_{(u,v)\in E} \mathbb{P}[(u,v) \in E]$$

which is just

$$\sum_{(u,v)\in E} \frac{1}{4} = \frac{m}{4}$$

- Las Vegas algorithms are those algorithms that have a deterministic, correct ouput; however, these algorithms have a random running time depending on the input they're supplied with.
  - We are commonly interested in bounding their worst case expected running time. This quantity can be called $T(n)$, and it can be expressed as:

$$T(n) = \max_{X:|X|=n} \mathbb{E}[X] \qquad (\alpha)$$

- Monte Carlo algorithms have a random output but a deterministic running time. We want to calculate what is the minimum probability that the algorithm produces an accurate output.
  - This can be expressed as

$$T(n) = \min_{X:|X|=n} \mathbb{P}[f(X) \text{ "is correct" }]$$

I now present two ways to analyze randomized quick sort. Recall that quick sort chooses a pivot in an array, sorts all elements in the array according to the

pivot, and then recursively sorts the "sorted" halves. Let $Q(A)$ be the number of comparisons made in order to quick sort A. Suppose A_n has n elements; then

$$Q(A_n) = \sum_{i=1}^{n} X_i \left(([Q(A_{i-1})] + [Q(A_{n-i})] \right)$$

where $X_i = 1$ if i is a pivot and 0 otherwise.

$$Q(A_n) = \sum_{i=1}^{n} X_i \left(([Q(A_{i-1})] + [Q(A_{n-i})] \right)$$

Now apply expectation to get:

$$\mathbb{E}[Q(A_n)] = \sum_{i=1}^{n} \mathbb{E}[X_i] \left(\mathbb{E}[Q(A_{i-1})] + \mathbb{E}[Q(A_{n-i})] \right) + n$$

We skipped a step in that we jumped from $\mathbb{E}[Q(A_j)X_{j+1}]$ to $\mathbb{E}[Q(A_j)]\mathbb{E}[X_{j+1}]$. This is justified because $Q(A_j)$ is independent of X_{j+1}; for the coin flips[1] used to choose pivot $j+1$ are independent of the coin flips that will be used in A_j.

$$\mathbb{E}[Q(A_n)] = \sum_{i=1}^{n} \frac{1}{n} \left(\mathbb{E}[Q(A_{i-1})] + \mathbb{E}[Q(A_{n-i})] \right) + n \qquad (\beta)$$

This recurrence is true for all inputs A of size n and, in particular, for the input that would require the maximum running time. Thus, if we define $T(n)$ as we did in (α), then the expression above becomes

$$T(n) \leq n + \sum_{i=1}^{n} \frac{1}{n} \left(T(i-1) + T(n-i) \right)$$

Notice that we replaced $=$ with $\leq$, for the use of $T(n)$ means that we are working with maximums, and so the inequality in β becomes less tight. For first (UNRESOLVED).

At this point, we can (UNRESOLVED) verify that this run time follows $O(n \log n)$ by running an inductive proof.

Another approach is to define R_{ij} as the event that the ith rank element is compared with the jth rank element and to define X_{ij} as the associated binary random variable. Thus, the total number of comparisons (and, hence, the running time of randomized quick sort) is given by

$$X = \sum_{1 \leq i < j \leq n} X_{ij}$$

where X is the total number of comparisons; whence

$$\mathbb{E}[X] = \sum_{1 \leq i < j \leq n} \mathbb{E}[X_{ij}]$$

$$\mathbb{E}[X] = \sum_{1 \leq i < j \leq n} \mathbb{P}[R_{ij}]$$

Lemma 2.1. $\mathbb{P}[R_{ij}]$ is $\frac{2}{j-i+1}$.

[1] We use coinflips to simulate a uniform distribution

Proof. Identifying the ith through jth ranked elements as $\{i \dots j\}$, observe that i is compared with j if and only if the pivot chosen from among $\{i \dots j\}$ is either i or j. Since pivot selection happens uniformly at random, there are two desireable values to choose from and $j - i + 1$ total values, the probability of R_{ij} is as stated. □

Whence

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{\delta=2}^{n-i+1} \frac{2}{\delta} \\
&= 2 \left(\sum_{i=1}^{n-1} H_{n-i+1} - 1 \right) \\
&\leq 2 \left(\sum_{i=1}^{n-1} H_{n-i+1} \right) \\
&\leq 2 \left(\sum_{i=1}^{n-1} \log n \right) \\
&\leq 2n \log n
\end{aligned}$$

Note that

$$\sum_{x=1}^{n} \frac{1}{x+1} \leq \underbrace{\int_{1}^{n} \left(\frac{1}{x} \right)}_{\ln(n)} \leq \sum_{x=1}^{n} \frac{1}{x}$$

which establishes that $H(n) = \Theta(n)$, since both the left hand side and the right hand since

$$H_n = O(\sum_{x=1}^{n} \frac{1}{x+1}) \leq \underbrace{\int_{1}^{n} \left(\frac{1}{x} \right)}_{\ln(n)} \leq O(\sum_{x=1}^{n} \frac{1}{x}) = H_n$$

3 Lecture 3

Probabilistic Inequalities

Remark 3.1. If we toss a fair coin n times then the probability of getting k heads is

$$\binom{n}{k} (1/2^n)$$

Recall that this holds, because any one sequence of heads and tails has a $\frac{1}{2^n}$ probability of taking place; finally, there are $\binom{n}{k}$ such sequences that have k heads.

Remark 3.2. Suppose we know that a random variable Q satisfies $\mathbb{P}[Q \geq 10n \log n] \leq c$. And we know that $Q \leq n^2$. Argue that $\mathbb{E}[Q] \leq 10n \log n + (n^2 - 10n \log n)c$.

First note that

$$\mathbb{P}[Q < 10n \log n] > 1 - c$$

It, therefore, follows that:

$$\begin{aligned} \mathbb{E}[Q] &\leq n^2 \times (\mathbb{P}[Q \geq 10n \log n]) + (\mathbb{P}[Q < 10n \log n])(10n \log n) \\ &= \mathbb{E}[Q] \leq n^2 \times (c) + (1 - c)(10n \log n) \end{aligned}$$

This gives us what we desire.

Remark 3.3. Recall that if X and Y are independent, then $\text{Var}(X + Y) = \text{Var}(X) + \text{Vary}(Y)$ and that $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$.

Remark 3.4. The Chebyshev bound is not, in certain cases, strong enough:

Suppose that we simulate a one dimensional random walk; a random variable X_i takes on -1 at time i if we move left at time i and otherwise takes the value 1. We wish to put bounds on the variable $Y = \sum_{i=1}^{n} X_i$.

Observe that $\mathbb{E}[Y] = \sum_{i=1}^{n} \mathbb{E}[X_i] = 0$ and that, since all the X_i are independent,

$$\text{Var}[Y] = \sum_{i=1}^{n} \text{Var}[X] = \sum_{i=1}^{n} \mathbb{E}[X^2] - \mathbb{E}[X]^2 \sum_{i=1}^{n} (1) = n$$

Because of this, we can use Chebyshev's inequality.

$$\begin{aligned} \mathbb{P}[|Y - \mathbb{E}[Y]| \geq t] &\leq \frac{\sigma_x^2}{t^2} \\ \mathbb{P}[|Y - \mathbb{E}[Y]| \geq t\sqrt{n}] &\leq \frac{n}{nt^2} = \frac{1}{t^2} \\ \mathbb{P}[|Y| \geq t\sqrt{n}] &\leq \frac{n}{nt^2} = \frac{1}{t^2} \end{aligned}$$

Lemma

Let $X_1, \ldots, X_k$ be k independent binary random variables such that, for each $i \in [1, k]$, $\mathbf{E}[X_i] = \Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{k} X_i$. Then $\mathbf{E}[X] = \sum_i p_i$.

- *Upper tail bound: For any $\mu \geq \mathbf{E}[X]$ and any $\delta > 0$,*

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$$

- *Lower tail bound: For any $0 < \mu < \mathbf{E}[X]$ and any $0 < \delta < 1$,*

$$\Pr[X \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}$$

Remark 3.5. The chernoff bound given before can be simpliefied in the case that $0 < \delta < 1$:

Chernoff Bound: Non-negative case, simplifying

When $0 < \delta < 1$ an important regime of interest we can simplify.

Lemma

Let $X_1, \ldots, X_k$ be k independent random variables such that, for each $i \in [1, k]$, X_i equals 1 with probability p_i, and 0 with probability $(1 - p_i)$. Let $X = \sum_{i=1}^{k} X_i$ and $\mu = \mathbf{E}[X] = \sum_i p_i$. For any $0 < \delta < 1$, it holds that:

$$\Pr[|X - \mu| \geq \delta\mu] \leq 2e^{\frac{-\delta^2\mu}{3}}$$

$\Pr[X \geq (1+\delta)\mu] \leq e^{\frac{-\delta^2\mu}{3}}$ *and* $\Pr[X \leq (1-\delta)\mu] \leq e^{\frac{-\delta^2\mu}{2}}$

Remark 3.6. If we allow the random variables to take on negative values as well, then we find that the new bound depends on the number of variables (the dimension), whereas the theorem involving non-negative random variables was dimension free.

Lemma

Let $X_1, \ldots, X_k$ be k independent random variables such that, for each $i \in [1, k]$, $X_i \in [-1, 1]$. Let $X = \sum_{i=1}^{k} X_i$. For any $a > 0$,

$$\Pr[|X - \mathrm{E}[X]| \geq a] \leq 2exp(\frac{-a^2}{2n}).$$

Applying this new finding to the random walk example presened earlier, we see that we can revise the bounds to be

$$\mathbb{P}\left[|Y| \geq t\sqrt{n}\right] \leq 2\exp -t^2/2 = \frac{1}{t^2}$$

Example 3.7. Suppose that we toss n balls into n bins (or consider that we toss m balls into n bins). We wish to bound the random variable Y, which we set to the maximum number of balls in any bin. To do so, we employ the following procedure:

- Focus on one bin, and bound the probability that this one bin receives more than a certain number of balls.
  - Express this probability by first defining binary random variables that represent the probability that a ball falls into this chosen bin.
  - Use the chernoff bounds to define a bound (when we work forward, we will usually have to leave this bound expressed as a variable; at the end of the exercise, it will be clear what bound to set.
- Apply the union bound to bound the probability that all bins receive more than a certain number of balls.
  - Use this to define the maximum number of balls that fall into any one bin.
- Compute the expectation in a manner similar to the example shown earlier.

Let X_{ij} be the event that ball j falls into bin i. Then set

$$X_i = \sum_{i=1}^{n} X_{ij}$$

where X_i represents the number of balls that fall into bin i. One Chernoff bound tells us that for $\delta > 0$, we have

$$\mathbb{P}\left[|X_i > (1+\delta)\mu|\right] \leq (\frac{e^\delta}{(1+\delta)^{1+\delta}})^\mu$$

In this case note that $\mathbb{E}[X_i] = \sum_{i=1}^{n} 1/n = 1$.

Whence, this simplifies to.

$$\mathbb{P}[|X_i > (1+\delta)|] \leq (\frac{e^\delta}{(1+\delta)^{1+\delta}})$$

For a second (ie, we will be concrete about this later by choosing a suitable values for δ), assume that

$$(\frac{e^\delta}{(1+\delta)^{1+\delta}})^\mu < \frac{1}{n^3}$$

Now observe that if we let A_i be the event that $X_i > (1+\delta)$

$$\mathbb{P}\left[\bigcup_{\in}^{n} A_i\right] \leq \sum_{i=1}^{n} \mathbb{P}[A_i] = 1/n^2$$

Then the probability that the maximum of any bin Y is less than $(1+\delta)$ is precisely the probability that that all bins have less than $(1+\delta)$. This is at least $1 - 1/n^2$. Whence, we can say, *with high probability* that this works.

Given that $Y \leq n$, we can bound $\mathbb{E}[Y]$ by

$$(1 - 1/n^2)(1+\delta) + 1/n^2(n)$$

UNRESOLVED: It is not known how the quantity:

$$\left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu$$

can be massaged into $\frac{1}{n^3}$.

Remark 3.8. Recall that the variance of a binary random variable is $p^2 - p = pq$ since $\mathbb{E}[X]^2 - \mathbb{E}[X]^2$.

The notes leave open the question of, assuming that a single ball first (before all other balls) falls into bin j, then what is the expected number of balls that will fall into j afterwards, what is the variance of this quantity and can we give a high probability bound (the answer to the latter being: yes, use the chernoff bound where the LHS is $\mathbb{P}[X \leq (1+\delta)\mu]$.

Example 3.9. ϵ representative Median Array

Problem W. e wish to devise an algorithm such that given a list A we can choose an element x such that the rank of x is

$$(1-\epsilon)n/2 \leq \text{rank}(x) \leq (1+\epsilon)n/2$$

Devise a randomized algorithm for this that outputs a desireable element with high probability

Solution 3.10. Sample from the array k times with replacement. Output the median of the sample. This works:

Set S to be the set obtained. Set M to be

$$\{y|(1-\epsilon)n/2 \leq \text{rank}(y) \leq (1+\epsilon)n/2\},$$

L to be the left portion of A and R to be right portion of A. Observe that if this algorithm does not produce a desireable candidate, then necessarily either

$$|S \cap L| \geq k/2 \text{ or } |S \cap R| \geq k/2^2$$

The more helpful observation is that if both

$$|S \cap L| < k/2 \text{ and } |S \cap R| < k/2$$

then, the algorithm does produce a good solution.

WOLOG, let us make a probability bound that $|S \cap L| < k/2$.

Using the (UNRESOLVED) first cherner inequality, we can show that for a sufficiently large value of k, we have

$$\mathbb{P}\left[|S \cap L| \geq k/2\right] < \delta/2$$

Running the same argument, we find that

$$\mathbb{P}\left[|S \cap R| \geq k/2\right] < \delta/2$$

whence

$$\mathbb{P}\left[\text{That either of the preceding two events take place}\right] \leq \sum \mathbb{P}\left[\text{That either of the two events take place alone}\right.$$

It follows that neither event will take place with probability $1-\delta$.

We outline the strategy of the approach (UNRESOLVED)

- Realize that

Example 3.11. We established that randomized quick sort takes $O(n \log n)$ time in expectation and, otherwise, require $O(n^2)$ time. We wish to concretely bound

$$\mathbb{P}\left[Q > n \log n\right]$$

where Q is the number of comparisons that quick sort makes.

Our strategy is as follows:

- Observe that if k levels of recursion take place then $Q \leq kn$.[3]
  - Whence it suffices to show that $k \leq$ some constant $\times \log n$.
- We will make a bound on the depth of recursion by proving that, with high probability, the number of levels of recursion involving any one element is bounded by some constant $\times \log n$.

[2]For if this were not the case, then $|S \cap M|$ would not be non empty.

[3]We call k the depth of the recursion.

- Applying the union bound, we will conclude that the total depth of recursion is less than some constant multiplied by $\log n$ with high probability.

Fixing an element $s \in A$, we let S_i be the partition of A that contains s on the ith iteration of quick sort. Call an iteration i lucky if $|S_{i+1}| \leq \frac{3}{4}|S_i|$ and if $|S_{i+1} \setminus S_i| \leq \frac{3}{4}|S_i|$. The probability that any one iteration is lucky is independent of the probability that a subsequent or later iteration is lucky, and the probability that an interation is lucky is $\frac{1}{2}$. UNRESOLVED :It is not presently known why we care that $|S_{i+1} \setminus S_i| \leq \frac{3}{4}|S_i|$. We observe that

if we let ρ be the number of lucky iterations, desire that $|S_k| = 1$, and use the fact that

$$|S_k| \leq (3/4)^\rho n$$

then $(3/4)^\rho n = 1 \implies \rho \ln(3/4) = \ln(1/n) \implies \rho = \frac{\ln(n)}{\ln(4/3)} = \log_{4/3} n \leq 4\ln(n)$. We will show that within $k = M \log n$ iterations for some $M > 0$, at least $4\ln(n)$ lucky iterations take place (with high probability) for each member of A. This will allow us, with high probability, to bound the probability that all members of A have $4\ln(n)$ lucky iterations within $M\log(n)$ iterations.

Let X_i be a binary random variable that is 1 if the ith iteration is lucky and 0 otherwise. Observe that $\mathbb{E}[[]\, X_i] = 1/2$ (convince yourself). Thus the expectation is $k/2$. We can then use Chernoff to make a bound:

$$\text{Set } k = 32\ln n \text{ and } \delta = 3/4 \mathbb{P}[\rho \leq 4\ln n] = \mathbb{P}[\rho \leq \mu(1-\delta)]$$

$$\begin{aligned} (\textit{Chernoff}) \quad &\leq e^{\frac{-\delta^2\mu}{2}} \\ &= e^{-\frac{9k}{64}} \\ &= e^{-4.5 \ln n} \leq \frac{1}{n^4} \end{aligned}$$

The probability that any on element has the preceding fate befall them is $1/n^3$. Thus, with probability $1-1/n^3$, no element will have this fate befall them, and the algorithm will end after $32\ln n$ comparisons. UNRESOLVED (again, it is not known how we could have derived the bounds that we set earlier.

4 Lecture 4

i-wise Independence and Hashing

Remark 4.1. It is expensive to store n random bits. We need a way to obtain n random variables without paying for the price to store n random variables.

Definition 4.2. Givey random variables $Y_1 \ldots Y_k$ with range $[B]$, we say that the random variables are totally independent if

$$\mathbb{P}[\{Y_i = b_i\} \forall i] = \prod_i \mathbb{P}[Y_i = b_i]$$

Lemma 4.3. It is easy to check that if a set of totally random variables is totally random in any of its subsets.

Example 4.4. A set of pairwise random variables is not necessarily totally independent.

Suppose that X_1 and X_2 are independent. Let $X_3 = f(X_1, X_2)$ where f is some deterministic function THen X_3 is not independent with X_1 and X_2, since knowing the latter two implies knowing the former.

Theorem 4.5. Index k uniformly random bits by $[k]$ so that X_i is the random variable whose value is the value of bit i. Let T be a subset of $[k]$ an let $S_T = \oplus_T X_i$. Then if $T \neq Q$, we have that T and Q are independent.

Proof. Sketch: Consider $T \cap Q = \emptyset$, $T \subseteq Q$ and $T \cap Q \neq \emptyset$ but $T \not\subseteq Q$. In each case, let v_{TQ} be the value obtained by taking the $\oplus$ of indices common to T and Q. There is at least one index that T and Q differ by, call it s and assume without loss of generality that T has it. Then $Y = v_{TQ} \otimes X_s$ is a uniformly random bit, since it takes on 1 as often as it takes on 0; it is independent of v_{TQ} since v_{TQ} does not influence Y to be anything other than uniformly distributed. If T and Q differ by any other bits, then we can apply the foregoing analysis to Y and Q. □

Theorem 4.6. Given $\log n$ random bits, we can construct n random bits[4] on the fly.

Proof. Apply the preceding theorem. Notice that we only have to store $\log n$ bits as opposed to n. □

Remark 4.7. If we want to make n random variables that are each uniformly distributed in $\{1 \ldots k\}$, then convince yourself that we need $\log n \log k$ random bits. To see this imagine that we have laid out $\log m$ cubbies. When each of these $\log m$ cubbies takes a value, we will get a number in $\{1 \ldots k\}$. In order to get n such random numbers, it suffices that each of these cubbies have n different random functions that fill these cubbies. We need $\log n$ bits to do that. Thus, each cubby is fillable by the function obtained by taking the $\oplus$ of some subset of these $\log n$ bits. This is why we need $\log n \log k$ bits.

Remark 4.8. Always imagine in these scenarios that we want to construct some random variables from random bits (and not just random values). That is, using some small number of random bits, we want to be able to create independent random functions that we pass on to some application. These functions can share parts, but these must ultimately themselves be random. Also helpful is to remember that these functions will take on random values, each time you provide them with random inputs. That is, pretend that you shake out random bits from a cup, and then insert those bits into random functions.

Lemma 4.9. If $x \in \mathbb{Z}_p$ where p is prime then there is a unique $y \in \mathbb{Z}_p$ such that $xy = 1 \mod p$. The proof is on his notes around page 13.

Remark 4.10. As a consequence, $\mathbb{Z}_p$ is a field, and it admits division.

[4]Whenever we speak of random bits, we will implicitly assume that they are uniformly randomly distributed.

Lemma 4.11. If $x \neq y$ and $(r, s) \in \mathbb{Z}_p \times \mathbb{Z}_p$, there is equality one pair $(a, b) \in \mathbb{Z}_p \times \mathbb{Z}_p$ such that

$$ax + b = r \text{ and } ay + b = s$$

Proof. Solve for a and b using this equation:

$$ax + b = r \text{ and } ay + b = s$$

We have $b = r - ax$ and $a = \frac{r-s}{x-y}$.

This tells us that if we know (a, b) then we know (r, s) and vice versa. □

Theorem 4.12. We can create p random variables with range $[p]$ using only $2\lceil \log p \rceil$ random variables.

Proof. Twice, using $\log p$ bits each time, get a value in $[p]$. Define $X_i = ai + b \forall i \in [p]$.

Observe firstly that each X_i is uniformly distributed. That is, for each $y \in [p]$, we have $\left|X_i^{-1}(y)\right| = p$. This is easy to verify, since whatever the value of ai, as b varies over $[p]$ $ai + b$ will take on all values.

Now we claim that X_i and X_j are pairwise independent for $i \neq j$. We need to show that:

$$\mathbb{P}\left[X_i = m \text{ and } X_j = n\right] = \mathbb{P}\left[X_i = m\right] \mathbb{P}\left[X_j = m\right] = 1/p^2$$

From the previous theorem, we know that given a pair $(m, n) \in \mathbb{Z}_p \times \mathbb{Z}_p$, the pair (a, b) that satisfies the foregoing equations is unique. The probability that this unique pair is chosen is $\frac{1}{p^2}$ and, thus, we are done. □

Remark 4.13. Note that the claim that independent $\{X_i\}_{i \in [n]}$ where $X = \sum_{i=1}^{n} X_i$ satisfy

$$\text{Var}\left[X\right] = \sum_{i=1}^{n} \text{Var}\left[X_i\right]$$

is sufficient if the X_i are pairwise independent.

Proposition 4.14. If we want to construction n pairwise independent random variables that have range $[m]$, where $n \neq m$, the previous construction involving p random variables that have $[p]$ range can be modified.

Proof. Assume first that $n < m$:

There is always a prime $p \in [m, 2m]$. Using this p, invoke the previous construction but take only n random variables instead of all p.

Assume now that $n > m$.

Lemma 4.15. All finite fields are of the form p^k where p is prime $k \geq 1$. Given any prime p and $k \geq 1$, there is a field of order p^k that is unique up to isomorphism.

If we use this lemma, then we can at least provide a construction for when n and m are powers of 2. Invoke the construction using n. Now take the n resultant random variables and truncate them by removing their first $\log m - \log n$ bits. This will leave them in the range $[m]$. These random variables are still uniformly distributed (for every power k of 2 where $k > m$, all possible arrangements of bits occurring in $X_i[0 : m-1]$ will appear twice more), so no one arrangement is skewed. The random variables are still pairwise independent: X_i and X_j. Sketch: When the first $\log n - \log m$ bits existed, there was a unique (a, b) that allowed $X_i = x$ and $X_j = y$. Now there are $2n/m = 2^{1+\log n - \log m}$ copies; the probability is thus $2n/m \Big/ n^2$ that an acceptable of (a, b) is chosen. The probabiliy that $X_i = x$ or that $X_j = y$ is $2n/m = 2^{1+\log n - \log m}$. □

Definition 4.16. Suppose a family $\mathcal{H}$ of hash functions is given. Then $\mathcal{H}$ is 2-strongly universal if, for any distinct x, y and $h \in \mathcal{H}$ chosen uniformly at random, it holds that $h(x)$ and $h(y)$ are independent. Moreover, fixing x and letting h vary, $h(x)$ is uniformly distributed.

Definition 4.17. $\mathcal{H}$ is 2-universal if for all distinct x, y it holds that $x\mathbb{P}\left[h(x) = h(y)\right]_{h\in\mathcal{H}} \leq \frac{1}{n}$.

5 Lecture 5

More on Hashing and Morris's Algorithm

Theorem 5.1. Assuming that we use separate chaining (ie when a collission takes place at some bucket, just append to a linked list at that bucket to add the element that resulted in the collission), then if we hash $n = |S|$ elements to a table of size $m = |T|$, assuming that we also choose a hash function uniformly at random from a universal hash family, the probability that a collision takes place is n/m,

Proof. Let $l(x) = T(h(x))$ be the size of the linked list at the bucket at the bucket mapped to by x. We wish to calculate $\mathbb{E}\left[l(x)\right]$. For any other y, let D_y be the binary random variable representing the event that y maps to x. Then

$$\mathbb{E}\left[l(x)\right] = \sum_{y\in S} \mathbb{E}\left[D_y\right]$$
$$= \sum_{y\in S} \mathbb{P}\left[h(x) = h(y)\right]. \leq \sum_{y\in S} \frac{1}{m}.$$

Since $\mathcal{H}$ is universal, the probability that $h(x) = h(y)$ is at most $1/m$. □

Theorem 5.2. Assume that $N = |\mathcal{U}|, m = |T|, n = |S|$. Let $p \geq N$ be prime. Let $a, b \in [p]$. Then the set of hash functions given by $\mathcal{H} = \left\{h_{a,b} | h_{a,b} = ax + b \mod p \mod m\right\}$ is universal.

Proof. First observe that the probability of a bucket being chosen is not exactly uniformly distributed. While $h_{a,b}$ restricted to $\mod p$ is uniformly distributed, where a and b are chosen uniformly at random, $h_{a,b}$ fails to be truly uniformly

random because it involves an outer $\mod$ under m. This outer $\mod$ suggests that some buckets (bear in mind that the regime $p > m$ is at work) – in particular some of the buckets $p \mod m$ buckets that are left over – will be filled while the $m - p \mod m$ buckets will not be. In fact the probability that the $p \mod m$ buckets will be chosen – and, thus, the greatest probability that a bucket can be chosen – is

$$\frac{\lfloor p/m \rfloor + 1}{p} \leq \frac{\lceil p/m \rceil}{\lfloor p \rfloor / m \rfloor m}$$

In class, it was claimed that this function does have a collission probability of less than $1/m$, however. UNRESOLVED – they seem to claim that if $x \neq y$ then $ax \neq ay \mod p$ without requiring that $x \neq y \mod p$; then they argue that adding $b \mod p$ to the expression ax or ay will have the collission be less than $1/m$. □

Definition 5.3. A bloom filter is a collection $\mathcal{H}$ of hashing functions that records whether an object was seen or not. Given an input x and $h \in \mathcal{H}$, we mark the bucket $h(x)$ by 1. When we are given an element y, we claim that $h(y)$ exists if $h(y) = 1$ and otherwise claim it does not exist. We do this for each $h \in \mathcal{H}$. Let the false probability of y existing be α for any $h \in \mathcal{H}$ be α. The bloom filter claims that an element exists iff all $h \in \mathcal{H}$ map that element to 1. Thus the probability of error is $\alpha^{|\mathcal{H}|}$.

Remark 5.4. We need $\log n$ bits to deterministically count to n. What if we kept track of $\log n$, however? Well, there is no function in place to calculate $\log n$ (at least not cheaply) – so we will keep track of $\log n$ using a randomized algorithm. In expectation, it will turn out that this algorithm gives us the right answer.

Definition 5.5. Morris's Counting Algorithm

```
X ←0
while objects come in:
  Flip a coin with probability 1/2^X. If heads:
      X ←0
Return 2^X − 1
```

Proof. The proof proving that this works in expectation is given in Chekuri's Lecture 5 notes. □

Remark 5.6. Chekuri also shows that the variance of this algorithm is $O(n^2)$. We can define X_n to be the value of the counter X when the nth object comes in. When we work with a random variable like this, which is not a simple sum of binary random variables, if we have only information about the random variable's expectation and variance, we generally can only appeal to Chebhysehv's inequality for bounds on the variable's deviation from its expectation. We might later apply Chernoff's inequality to events of the form

$$|Y_i - \mu| > c \tag{α}$$

ie the binary random variable A_i will represent the probability that (α) takes place (for which Chebyshev's will provide us with a minimum probability).

Remark 5.7. If we can provide an expectation and variance for an algorithm's running time, then observe that if we operate that algorithm n times independently (or, perhaps, in parallel), and take the average of the algorithms' results, then the variance of the resulting algorithm's accuracy drops by a factor $1/n$.

Remark 5.8. The median trick presupposes that we have a bound of the form (α) – see an earlier equation. In particular, the bound appears like

$$\mathbb{P}\left[|Y_i - \mu| > c\right] \leq d$$

Assembling a collection $Q = \{Y_i\}$, we use the value of the median in Q, for the median is, with high probability, a good estimator of the expectation. Why? Let us say that a member Y_i is a good estimator if $|Y_i - \mu|$ is within c; Y_i is a bad estimator, by contrast, if the foregoing difference exceeds c. If the median is not a good estimator, then necessarily more than half of the members in the collection are bad members. Thus, whatever probability bound exists on the scenario that more than half the members are bad is also applicable to the scenario that the median is bad. Fortunately, we can apply Chernoff bounds to the former – meaning, effectively, the latter.

Example 5.9. In Chekuri's notes, we find that after defining Z^i to be an average of the Y_i, where $Y_i = 2^{X_i}$, the following bound is applies to Z^i:

$$\mathbb{P}\left[\left|Z^i - \underbrace{n}_{\mu}\right| > n\epsilon\right] \leq 1/4$$

We wish to have the probability bound above be δ accurate. To do so, we define A^i as the binary random variable where $\left|Z^i - \underbrace{n}_{\mu}\right| > n\epsilon$; that is A^i takes on 1 if Z^i is bad; otherwise, A^i is good. The probability that A^i is bad is, at most, $1/4$. Suppose that we have n such A^i. What is the probability that more than half of the A^i are bad. The probability can be bounded by the Chernoff bound:

For $\delta \in (0, 1)$ we have:

$$\mathbb{P}\left[|X - \mu| \geq \mu(\delta)\right] \leq 2\exp -\mu\delta^2/3$$

Here we let $X = \sum_{i=1}^{l} A^i$ where l is the number of events, we let $\mu = l/4$ and we let $\delta = 2$.

$$\begin{aligned} &= 2\exp -(l/4)(4/3) \\ &= 2\exp -l/3 \end{aligned}$$

Let $l = c\log(1/\delta)$. Let $c = 3$.

$$\begin{aligned} &= 2\exp -3(\log(1/\delta))/3 \\ &= 2\exp -\log(1/\delta) \\ &= 2\exp\log(\delta) \\ &= 2\delta \end{aligned}$$

6 Lecture 6

Frequency Moments and Counting Distinct Elements

Definition 6.1. Suppose that we receive m objects (not necessarily distinct) that are members of the set $[n]$. This set of m objects will, as they appear in a stream, be labeled σ; we have $B << m$ bits to process σ. Let g be a real valued non-negative function that operates on streams.

Definition 6.2. We say that a streaming algorithm $\mathcal{A}$ provides (ϵ, δ) additive approximation if

$$\mathbb{P}\left[|\mathcal{A} - g| > \epsilon\right] < \delta$$

J

Definition 6.3. We say that a streaming algorithm $\mathcal{A}$ provides (ϵ, δ) relative approximation if

$$\mathbb{P}\left[\frac{\mathcal{A}(\delta)}{g(\delta)} - 1 > \epsilon\right] < \delta$$

Remark 6.4. Many streaming problems are trivial to solve deterministically, if we have full access to the data that enters a stream. To solve them deterministically, however, there are lower bounds on the space and running time of working algorithms. These bounds are often unacceptable. To achieve useful space and running times, thus, streaming algorithms often end up using randomized algorithms.

Remark 6.5. The task of counting how many distinct elements appear in a stream has obvious, if costly, deterministic solutions: Put all objects in a binary search tree; when a new stream object appears, search the tree for the object and, if not found, add it to the tree. We call this task the *distinct elements problem*.

This requires $O(k)$ space and $O(m \log k)$ running time where k is the true number of distinct objects. Even if you were told in advance that there would be k objects, a hashing solution would require $O(k)$ space and $O(m)$ running time.

Remark 6.6. A cute idea is to assume the existence of a hash function $h : [n] \to [n^3]$ that closely "resembles" the theoretical, non-existent hash function that maps all objects with perfect randomness to some real in $[0, 1]$. Then keep track of the random varible $X = \min_{e_i \in \text{ all objects}} h(e_i)$. The minimum statistic (for k uniform random variables over $[0, 1]$) has the property that $\mathbb{E}\left[[]\, X\right] = \frac{1}{k+1}$. Thus, at the end of streaming, take X, invert it and subtract 1 from it to obtain a statistical estimator of the number of distinct objects.

Convince yourself that the following integral represents the expectation.

$$\mathbb{E}\left[X\right] = \int_0^1 (\frac{1}{1-0}) \binom{k}{1} (1-x)^{k-1}$$

Theorem 6.7. Using the definitions set above, we can achieve a $1+\epsilon$ accurate estimate of X with probability $1-\delta$; if we kept track of the minimum hash value seen using only $O(\log n)$ bits, then

6.1 BJKST

In what follows, I give a high level overview of how to understand the analysis of BJKST

- What does the algorithm do?
  - The algorithm repeatedly hashes elements as they appear and stores the t smallest hash values.We use some hash function taken from a 2-universal hash family; moreover, this hash function has the formula $[n] \mapsto [n^3]$. It returns a statistic that depends on the average hash value of the tth smallest hash value.
- How do we analyze the algorithm?
  - First, we need to understand the statistics at place. Assume that v is the hash value of the tth smallest hash. Then:
    - The minimum of the hash function, assuming d distinct elements would fall around $\frac{1}{d+1}$ if we were working with an ideal hash function.
    - Since we work over $[N]$ instead of $[0,1]^3$, however, it seems reaonable that this will instead be $\frac{N}{d+1}$. Moreover, the tth hash – let's call it v – will hash to $\frac{tN}{d+1}$ on average (we're waving our hands, here). So

$$v = \frac{tN}{d+1} \implies (d+1) = \frac{tN}{v} \implies \approx d = \frac{tN}{v}$$

- We will have the algorithm described above return $\frac{tN}{v}$ as our estimate for d. We would like to have a bound of the form:

$$\mathbb{P}\left[|D-d| > \epsilon d\right] \leq 1/3$$

  For then we can apply the median trick.
  - Show that there is a low chance that more than half of this experiment's trials are all bad. On average, $l/3$ of the trials will be bad; we want to bound the probability that more than $l/2$ of the trials will be bad, for that will imply the median is good. Use the bound $\mathbb{P}\left[X \geq (1+\delta)\mu\right] \leq \exp\frac{-\delta^2\mu}{3}$.
- This is decomposable into exactly two, disjoint events:

$$\mathbb{P}\left[D > d(\epsilon+1)\right] + \mathbb{P}\left[D < d(1-\epsilon)\right]$$

- Focus on

$$\mathbb{P}\left[D < d(1-\epsilon)\right]$$

  It is more tractable to explore this inequality when we work with the concrete objet v, the object whose hash was the tth smallest of all hashes.

$$D < d(1-\epsilon) \iff tN/v < d(1-\epsilon) \iff v > \frac{tN}{d(1-\epsilon)}$$

- The foregoing occurs iff lesss than t objects fell into $\frac{tN}{d(1-\epsilon)}$.
  - The probability that a single distinct element falls in this range is

$$\frac{t}{d(1-\epsilon)}$$

Let $X = \sum_{i=1}^{d} X_i$. We have that

$$\mathbb{E}[X_i] = \frac{t}{d(1-\epsilon)} > \frac{t(1+\epsilon)}{d}$$
$$\mathbb{E}[X] = \frac{t}{1-\epsilon} > \frac{t(1+\epsilon)}{1}$$
$$\mathrm{Var}[X_i] < \mathbb{E}[X_i] < \frac{t(1+3\epsilon/2)}{d}$$
$$\mathrm{Var}[X] < \mathbb{E}[X] < \frac{t(1+3\epsilon/2)}{1}$$

Notice that we were originally interested in the value $\mathbb{P}[X < t]$

$$\mathbb{P}[X < t] \leq \mathbb{P}[|X - \mathbb{E}[X]| > \epsilon t] \leq \mathrm{Var}[X]/(\epsilon t)^2 \leq \frac{t(1+3\epsilon/2)}{c}$$

If we make c sufficiently large, then we get a desireable inequality

---

- Now focus on

$$\mathbb{P}[D > d(\epsilon+1)]$$

- As before, we find that the event $D > d(\epsilon+1)$ is synonymous with the event that $v < \frac{tN}{(1+\epsilon)d}$.
  - This is the event that at least t values hash inside $[1 \ldots \frac{tN}{(1+\epsilon)d}]$
- The probability that any one value hashes inside this range is

$$\frac{t}{(1+\epsilon)d} \leq \frac{t(1-\epsilon/2)}{d}$$

The latter inequality follows from the fact that $\frac{1}{1+\epsilon}$ is an alternating series[5] and that $\epsilon < 1/2$. We want to bound

$$\mathbb{P}[X \geq t]$$

[5] You can prove that odd partial sums are decreasing and even partial sums are increasing.

$$\mathbb{P}[X \geq t] \leq \mathbb{P}[|X - \mathbb{E}[X]| > \epsilon t/2 - q] \leq \frac{4\text{Var}[X]}{(\epsilon t - 2q)^2} \approx \frac{4\text{Var}[X]}{c}$$

Now choose c to make RHS less than $\frac{1}{6}$

- The preceding holds, because as before, it can be shown that

$$\text{Var}[X] < \frac{t(1-\epsilon/2)}{}$$

- Recall that we need c/ϵ^2 elements. c here is fixed, once we find a sufficiently large value for it. Thus we need $\log(1/\epsilon^2)\log n$ bits; and we perform the median trick $\log(1/\delta)$ times.
  - In total we need $O(\frac{1}{\epsilon^2}\log(1/\delta)\log n)$ bits.

7 Lecture 7

Definition 7.1. A stream is a sequence $O = \{e_1 \dots e_m\}$ where $e_i \in [n]$. For each $j \in [n]$, $f_j = |O \cap j|$, which is the number of times that j appears in O.

Definition 7.2. A function $g : \mathbb{R} \to \mathbb{R}$ of a stream σ satisfies:

$$g(0) = 0$$
$$g(\sigma) = \sum_{i=1}^{n} g_i(f_i)$$

Example 7.3. Let $g_i = h(x) = x^k$ for each i. Then $g(\sigma)$ is the kth frequency moment.

Remark 7.4. AMS sampling does the following:

- Draw an element e_J uniformly at random from $\{e_1 \dots e_m\}$.
  - $e_J = i$ for some $i \in [n]$.
- Count how many times i appears out of $\{e_j | j \geq J\}$.
- Let

$$R = \left|\left\{e_j | j \geq J\right\}\right|$$

- Output $g_i(R) - g_i(R-1)$

Lemma 7.5. Let Y be the output of AMS. Then $\mathbb{E}[Y] = g(\sigma) = \sum_{i=1}^{n} g_i(f_i)$.

```
AMSEstimate:
    s ← null
    m ← 0
    R ← 0
    While (stream is not done)
        m ← m + 1
        a_m is current item
        Toss a biased coin that is heads with probability 1/m
        If (coin turns up heads)
            s ← a_m
            R ← 1
        Else If (a_m == s)
            R ← R + 1
    endWhile
    Output m(g_s(R) − g_s(R − 1))
```

Proof.

Let e_J be the random variable whose values is the element in $[n]$ that we choose.

$$\begin{aligned}
\mathbb{E}[Y] &= \mathbb{E}[\mathbb{E}[Y|e_J]] \\
&= \sum_{i=1}^{n} \mathbb{P}[e_J = i]\, \mathbb{E}[Y|e_J = i] \\
&= \sum_{i=1}^{n} \mathbb{P}[e_J = i]\, \mathbb{E}[Y|e_J = i] \\
&= \sum_{i=1}^{n} \frac{f_i}{m} \mathbb{E}[Y|e_J = i] \\
&= \sum_{i=1}^{n} \frac{f_i}{m} \sum_{l=1}^{f_i} \frac{1}{f_i} m\,(g(l) = g(l-1)) \\
&= \sum_{i=1}^{n} g_i(f_i)
\end{aligned}$$

□

Remark 7.6. At this point, calculations are given that bound $\mathrm{Var}[Y]$. The calculations are not very illuminating, so I omit showing them. They are at the very end of the February 5th slides:

`https://courses.engr.illinois.edu/cs498abd/sp2019/slides/07.pdf`

8 Lecture 8

Estimation of F_2

Remark 8.1. F_2 is special for "fundamental" reasons.

Unresolved 8.2. The previous lecture's AMS algorithm purportedly establishes that we can estimate F_2 in $\sqrt{n}\log n$ space. The analysis revealed that we need $O(\sqrt{n})$ versions of the algorithm to run in parallel. It is not clear, however, that each version requires $\log n$ space – indeed, the algorithm needs counters that are at most m, which would suggest $\log m$ space. In the event that $m \in O(n^k)$ for some k, then this would be okay.

Update: It is likely that we need $\log(n)$ space to store the hash function.

Lemma 8.3. Consider the following algorithm:

AMS-F_2-ESTIMATE:
Suppose that h is chosen from a 4-wise independent hash family with formula $[n] \to \{-1, 1\}$.
$z \leftarrow 0$
For each e_j:
    $z \leftarrow h(e_j)$
Output z^2.

This algorithm estimates F_2 with (ϵ, δ) accuracy if we use $O(\log(1/\delta)\frac{1}{\epsilon^2}\log n)$ bits.

Proof. See lecture notes 8. `https://courses.engr.illinois.edu/cs498abd/sp2019/slides/08.pdf` □

Remark 8.4. This algorithm can be extended to estimate not just the 2 norm of the frequency vector but the square of the euclidean norm of any vector (and hence the euclidean norm itself). This is again covered in his lecture notes. In fact, wherever in the proof of the AMS-F_2 algoritm one uses f_i, just replace it by x_i.

Remark 8.5. k-wise independent families satisfy the property that for any $S = \{a_1 ... a_k\} \subseteq [n]$, we have that all elements in S are independent. In particular, a k-wise independent hash family guarantees that the hash of any input is uniformly distributed and that any 4 hashes are independent.

Definition 8.6. A sketch c of a stream σ is a function such that $c(\sigma_1 \cdot \sigma_2) = c(\sigma_1) * c(\sigma_2)$. Here, $*$ is some operation and σ_i are subsets of the stream and $\cdot$ is concatenation.

Remark 8.7. The previous algorithm A is a linear sketch[6]. For if we use the same hash function over two streams σ_1 and σ_2 then $(A(\sigma_1) + A(\sigma_2))^2 = A(\sigma_1 \cdot \sigma_2)^2$.

Remark 8.8. If we wish to run this algorithm in order to attain (ϵ, δ) accuracy, then could formulate it as follows:

Create the matrix

$$M = \begin{bmatrix} h_1(1) ... h_1(n) \\ \vdots \\ h_{O(\frac{1}{\epsilon^2}\log(1/\delta))}(1) ... h_{O(\frac{1}{\epsilon^2}\log(1/\delta))}(n) \end{bmatrix}$$

[6]Note that the algorithm returns z and one must additionally post-process this return value by returning z^2, so that we treat the algorithm as solely returning z

Then when the token (i_t, δ_{i_t}) where $i_t \in [n]$ for $t \in [m]$ appears, update a vector Z of initially $O(\frac{1}{\epsilon^2}\log(1/\delta))$ zeros so that $Z = Z + \delta_i * M(e_{i_t})$. At the end, we will have a vector of algorithm estimates, which we should take the median q of and then output q^2. In this formulation, we see that if we havea single M across two runs of this experiment and let

$$x_q = \sum_{t=1}^{m} \delta_{i_t} e_{i_t}$$

for $q \in \{1, 2\}$ (for both runs) then,

$$M(x_1 + x_2) = Mx_1 + Mx_2$$

Here $x_1 + x_2$ represents the action of taking the concatenation of both streams. This gives some insight as to why the algorithm is called a linear sketch.

9 Lecture 9

Heavy Hitters

Remark 9.1. Note that the algorithm to the determine the majority element can be shown to output the correct member, if there is in fact a majority element. If there is no majority element, there may be garbage.

Remark 9.2. The deterministic algorithm that returns those elements whose frequency is greater than m/k for some k will at least output those elements $i \in [n]$ such that $f_i \geq m/k$; however, this algorithm may also erroneously return those i such that $f_i < m/k$.

Unresolved 9.3. A second pass of the output will suffice to weed out those elements whose frequency moment is small.

We stopped watching at minute 57 of Tuesdsay February 12th lecture. Professor was then giving a high level overview of the randomized algorithm that allows one to find heavy hitters (also allowing for negative frequency updates).