Randomized Algorithms

CS6170

Jul - Nov 2021, Dept. of CSE, IIT Madras

Contents

Preface

These are my personal notes for the Randomized Algorithms course for the July - November semester, 2021. The material that is covered here is not new, and there are a number of references available for these topics. As is most often the case, I prefer the presentation of a particular topic better in one book than in another. Therefore, even though the material covered is basic, I have not restricted myself to one textbook. These notes collate the many different expositions and tries to make a consistent presentation.

Caveat Lector! These notes are meant for me to organize my thought before the lectures. It is likely that there are errors in the notes. Read the notes at your own risk!

If you feel that something in the notes do not make sense, then try to look it up on one of the textbooks. Let me know if there are any egregious errors.

1 Introduction

Randomization is ubiquitous in computer science. In many cases, we obtain faster, simpler and more elegant algorithms that better the "polynomial-time" algorithm known for the same problem. An example is the problem of primality testing. Suppose you are given an integer n as input and you want to check if n is prime or not. The trivial algorithm would be to check whether some number between 1 and $\sqrt{n}$ divides n or not. This is highly inefficient since for a number with 100 digits could be as large as 2^{100} and the naive algorithm has a running time for 2^{50}. In a breakthrough result in 2002, AKS showed that there is a polynomial time algorithm for testing primality, and currently the best upper bound known is $O(\log^7 n)$. While this running time does not look as prohibitive as the naive algorithm, it is, nonetheless, not very practical. As it turns out, there is a very simple algorithm to test primality, albeit one that can make an error occassionally. We will make the statement "make an error occassionally" precise in a moment. But, let us first look at the algorithm known as the Miller-Rabin test.

Algorithm 1: Miller-Rabin Primality Test

Input: Integer n

Let $n = 2^r s + 1$, where s is odd
Choose a uniformly at random from $\{2, \ldots, n-1\}$
if $a^s \neq 1 \pmod m$ **then** return `composite`;
for $0 \leq i \leq s-1$ **do**
  if $a^{2^i s} = -1 \pmod m$ **then** return `prime`;
end
return `composite`

By repeated squaring and computation of remainder modulo n, it is easy to see that the running time of the algorithm is $O(\log^2 n)$. As you can see, the algorithm is easy to implement in terms of the calculations that has to be done, and its running time is also good asymptotically. It can shown that if n is a prime, then irrespective of the choice of a then algorithm will always return "prime". When analyzing the algorithm (which is non-trivial and requires some basic algebra), we can show that for at least 3/4th fraction of numbers between 2 and $n-1$, the algorithm will return "composite". In other words, if the number is prime, then the probability that the algorithm errs is zero, whereas if it is composite, then the probability that it errs is at most $1/4$. It is possible to bring down with error probability to something as tiny as $1/2^40$ without changing the asymptotic complexity of the algorithm (we will see how to do this a little later). Note that while the algorithm is simple to explain,

its proof of correctness will not be easy, in most cases.

We will see various settings where randomized algorithms give simple, elegant and fast algorithms where fast deterministic algorithms are not known. Nonetheless, most theoretical computer scientists believe that randomness does not inherently add more computational power. In other words, they believe that if a computational problem has a fast randomized algorithm, then it also has a fast deterministic algorithm! Then why study randomized algorithms at all? Firstly, we are currently far away from that goal of converting every fast randomized algorithm to a fast deterministic algorithm. Secondly, fast here means polynomial-time. So, you could have a randomized primality test that runs in $O(\log^2 n)$ time and the best deterministic primality test may still require $O(\log^7 n)$. That is a considerable gap in many practical situations. Finally, there are scenarios where randomization is unavoidable. One such example is counting motifs in large graphs that are too big to run classical algorithms on. Any algorithm that can approximate these counts necessarily has to be randomized.

In the rest of this lecture, we will see a few more examples while refreshing some basic concepts in discrete probability.

1.1 Polynomial Identity Testing

Suppose that you are given a degree-d polynomial $p(x) = \sum_{i=0}^{d} c_i x^i$ in the explicit way. Your friend claims that the factorization of $p(x)$ is given by $q(x) = \prod_{i=1}^{d}(x - a_i)$ by using a program for polynomial factorization that she has written. How do you check if your friend is indeed telling the truth? The most straightforward way would be for you to expand the factorization $q(x)$ and check that it indeed gives $p(x)$. But, this is tedious since you might end up with far more terms than just the d terms that you will then have to cancel and reduce. With the power of randomness there is something very simple that you can do!

Let us assume for the moment that the numbers involved are all rationals. Consider the polynomial $p(x) - q(x)$. Notice that this polynomial is identically zero precisely when $q(x)$ is the factorization of $p(x)$. Furthermore, $p(x) - q(x)$ is a degree-d polynomials. Now, you know that every polynomial of degree d has at most d roots (this is known as the fundamental theorem of algebra). This gives you an idea for a solution! You choose a number a uniformly at random from the set $\{1, 2, \ldots, 100d\}$, and check if $p(a) - q(a) = 0$. Notice that if $q(x)$ is indeed the factorization of $p(x)$, then for every a, $p(a) - q(a) = 0$ and we will answer correctly. Moreover, if $q(x)$ is not the correct factorization of $p(x)$, then $p(x) - q(x)$ is a non-zero polynomial of degree d. Hence, it has at most d distinct roots, and the probability that a is one such root is at most $1/100$.

A generalization of this problem is the famous *Polynomial Identity Testing* problem (PIT for short). Here you have access to a multivariate polynomial $p(x_1, x_2, \ldots, x_n)$ of degree d. Observe that the number of monomials of this polynomial can be as large as $\binom{n+d-1}{n}$, and the polynomial is expressed succinctly as a product of a small number of polynomials. You want to check if this polynomial is identically zero. Since the explicit description of the polynomial can be exponentially larger than the given representation, it is inefficient to write out the full

polynomial and check whether it is indeed zero. Once again randomness comes to our rescue! We will state a lemma that generalized the fundamental theorem of arithmetic.

LEMMA 1.1 (DeMillo-Lipton-Schwartz-Zippel). *Let $p(x_1, x_2, \ldots, x_n)$ be a non-zero degree d polynomial over rationals. Let S be a subset of rational numbers. Then,*

$$\Pr_{a_1,a_2,\ldots a_n \in_r S}[p(a_1, a_2, \ldots, a_n) = 0] \leq \frac{d}{|S|}$$

Here by $a_1, a_2, \ldots, a_n \in_r S$ we mean that each a_i is picked from S with replacement. We will use this notation through the lecture notes. Observe that when $n = 1$, then the lemma follows from the fundamental theorem of algebra. The lemma can be proved by an induction on n (the number of variables). We will see this shortly.

1.1.1 Perfect matching in bipartite graphs

For a bipartite graph $G(L \cup R, E)$ with $|L| = |R| = n$, a perfect matching $M \subseteq E$ is a set of edges such that not two edges in M share a vertex, and for every vertex $v \in L \cup R$, there is an edge in M with one of the end points as v. The perfect matching thus gives a bijection between L and R via the edges in the graph. There are polynomial-time algorithms known for this problem (like the Edmonds-Karp algorithm). In fact, the idea of polynomial-time as the notion of efficiency was first described in a paper by Edmonds that gave a polynomial-time algorithm for the maximum matching problem. What we will see is that using randomization, and in particular PIT, we can give a simple algorithm for testing if a bipartite graph has a perfect matching.

Let A denote the bipartite adjacency matrix of G. I.e. A is an $n \times n$-matrix with the rows indexed by the vertices in L and the columns indexed by the vertices in R such that $A[i, j] = 1$ iff $(i, j) \in E$. Define an $n \times n$-matrix P as follows:

$$P[i, j] = A[i, j]x_{ij},$$

where x_{ij}s are variables. Recall that the determinant of P is defined as

$$\mathrm{Det}(P) = \sum_{\pi \in S_n} (-1)^{sgn(\pi)} \prod_{i=1}^{n} P[i, \pi(i)].$$

Observe that for the matrix P defined as above, the determinant $\mathrm{Det}(P)$ is a polynomial on n^2 variables (x_{ij}). The monomials of this polynomial are the products $\prod_{i=1}^{n} P[i, \pi(i)]$. A monomial is non-zero precisely when π is a perfect matching in the graph. Furthermore, two different monomials have at least one variable distinct from each other and hence do not cancel each other. Consequently, the polynomial $\mathrm{Det}(P)$ is non-zero precisely when G has a perfect matching. So, the problem of checking the existence of a perfect matching reduces

to polynomial identity testing, where the polynomial is the determinant polynomial. When values for the variables are fixed, then we know how to compute the determinant efficiently. Assuming the DeMillo-Lipton-Schwartz-Zippel lemma, this gives a simple randomized algorithm for checking whether a bipartite graph has a perfect matching.

1.2 Sample spaces, events, probability

Let us recall the notions of sample spaces, events and probability distributions. A randomized algorithm is a random process, and analyzing the algorithm will involve understand the sample space of the random process and the distribution over this sample space. Formally, a *sample space* Ω is the set of possible outcomes of a random experiment. An *event* E is a subset of Ω. A *probability space* is a 3-tuple $(\Omega, \mathcal{F}, \Pr)$ where $\mathcal{F}$ is the set of subsets of Ω and $\Pr : \mathcal{F} \to \mathbb{R}$ is a function that satisfy the following conditions:

1. For every event E, $0 \leq \Pr[E] \leq 1$.
2. $\Pr[\Omega] = 1$
3. For any countable sequence $E_1, E_2, \ldots$ of pairwise disjoint events, $\Pr[\cup E_i] = \sum \Pr[E_i]$.

The following statement, which follows from the inclusion-exclusion principle is very useful, and we will refer to it frequently.

Fact 1.2 (Union bound). *Let $E_1, E_2, \ldots, E_n$ be n events. Then*

$$\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr[E_i]$$

The *conditional probability* of an even E conditioned on another event F is denoted as $\Pr[E|F]$ and is defined as

$$\Pr[E|F] = \frac{\Pr[E \cap F]}{\Pr[F]}.$$

Notice that the conditional probability is well-defined only when $\Pr[F] \neq 0$. Intuitively, the sample space of interest is the set F, and hence we normalize it with $\Pr[F]$ so that the conditional probability remains a valid probability function.

We say that two events E and F are independent if $\Pr[E|F] = \Pr[E]$ and $\Pr[F|E] = \Pr[F]$. In other words, conditioned on the even E(or F) occurring, the probability of the occurrence of F(or E) does not change. An equivalent way to say this is that $\Pr[E \cap F] = \Pr[E] \Pr[F]$. Notice that $\Pr[E \cap F] = \Pr[E] \Pr[F|E] = \Pr[F] \Pr[E|F]$.

Another useful, and fairly straightforward, property described below is known as the law of total probability.

Theorem 1.3 (Law of Total Probability). *Let $E_1, E_2, \ldots, E_n$ be mutually disjoints events in the sample space Ω such that $\bigcup_{i=1}^{n} E_i = \Omega$. Let F be any event in the sample space Ω. Then,*

$$\Pr[B] = \sum_{i=1}^{n} \Pr[B \cap E_i]$$

Let us now look at the proof of Lemma 1.1. We will explicitly show the underlying sample space and the events of interest to illustrate the definitions and concepts described above.

Proof of Lemma 1.1. The proof is via an induction on n (the number of variables). The base case is when $n = 1$. Now the sample space of the experiment is the set S. Since we are sampling uniformly at random from the set S, the probability function Pr assigns the value $1/|S|$ to every element in the sample space. Now the event that we are interested in is the subset $Z \subseteq S$ such that for every $z \in Z$, $p(z) = 0$. By the fundamental theorem of algebra, we know that $|Z| \leq d$. Therefore,

$$\Pr_{a_1 \in_r S}[p(a_1) = 0] = \Pr[Z] = \frac{|Z|}{|S|} \leq \frac{d}{|S|}.$$

Let us prove the inductive step. The sample space Ω for this experiment is the set of n-tuples over S with Pr being the function that assigns the same value to every element in the sample space. We can write the polynomial $p(x_1, x_2, \ldots, x_n)$ as

$$p(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{d} x^i p_i(x_2, x_3, \ldots, x_n).$$

If p is not identically zero, then there exists an i such that $p_i(x_2, x_3, \ldots, x_n)$ is not identically zero. Consider the largest such i for which p_i is not identically zero. Let $Z_i \subseteq \Omega$ be tuples such that p_i evaluates to zero on these points. By the inductive hypothesis, $\Pr[Z_i] \leq (d - i)/|S|$. Let $Z \subseteq \Omega$ denote the tuples on which p evaluates to zero. By the law of total probability, we can write

$$\begin{aligned}
\Pr_{a_1, a_2, \ldots a_n \in_r S}[p(a_1, a_2, \ldots, a_n) = 0] = \Pr[Z] &= \Pr[Z \cap Z_i] + \Pr[Z \cap \bar{Z}_i] \\
&= \Pr[Z_i] \Pr[Z|Z_i] + \Pr[\bar{Z}_i] \Pr[Z|\bar{Z}_i] \\
&\leq \Pr[Z_i] + \Pr[Z|\bar{Z}_i]
\end{aligned}$$

If the event Z_i does not occur, then we have a polynomial $p(x_1, a_2, \ldots, a_n)$ of degree at most i and hence $\Pr[Z|\bar{Z}_i] \leq i/|S|$. Therefore, we have

$$\Pr_{a_1, a_2, \ldots a_n \in_r S}[p(a_1, a_2, \ldots, a_n) = 0] = \Pr[Z] \leq \frac{d}{|S|}.$$

□

We will see one more example problem of randomization that gives non-trivial savings in the running time while understanding the basic concepts of discrete probability better.

1.3 Verifying matrix multiplication

Suppose that you are provided with three $n \times n$ matrices A, B and C with the claim that $AB = C$. You want to verify whether C is indeed the product of the matrices A and B. The simplest way to do this is to actually perform the matrix multiplication and check AB and C entry-wise. The naive matrix multiplications algorithm takes time $O(n^3)$. Currently, the fastest matrix multiplication algorithm has a running time of $O(n^{2.37})$, though it is conjectured that there is an $O(n^{2+\epsilon})$-time algorithm for matrix multiplication for every $\epsilon > 0$. We will see an $O(n^2)$-time randomized algorithm to verify matrix multiplication over the field $\mathbb{F}_2$ (i.e. we do addition and multiplication modulo 2).

The algorithm is quite simple to state: Choose a random vector $r \in \{0,1\}^n$ and check if $ABr = Cr$. Since r is an $n \times 1$-matrix, the multiplication Br and Cr takes $O(n^2)$ time, and the result of Br is another $n \times 1$-matrix. Thus, the verification can be done in $O(n^2)$-time. It remains to show that the procedure does not err too much. Notice that if $AB = C$, then this procedure does not err at all. Hence all that remains is to show that if $AB \neq C$, then the probability that this process gives a wrong answer is small. The following lemma is sufficient, since if $AB \neq C$, then $D = AB - C$ is a non-zero matrix.

LEMMA 1.4. *Let D be a non-zero $n \times n$-matrix. Then, we have*

$$\Pr_{r \in_r \{0,1\}^n}[Dr = 0] \leq \frac{1}{2}$$

Proof. Let $r = (r_1, r_2, \ldots, r_n)$. If r is sampled uniformly at random from $\{0,1\}^n$, then it is equivalent to saying that each r_i is set to $0/1$ with probability $1/2$. Since D is non-zero, assume wlog that $D_{1,1} \neq 0$. Consider the $(1,1)$-th entry of Dr given by $\sum_{k=1}^n D_{1,k} r_k$. We will argue that $\Pr[\sum_{k=1}^n D_{1,k} r_k \neq 0] = 1/2$.

The sample space we are working with is the set $\{0,1\}^n$ with each element in the set being assigned the same probability. The event we are interested in the set of points such that $\sum_{k=1}^n D_{1,k} r_k \neq 0$. First notice that if $\sum_{k=1}^n D_{1,k} r_k \neq 0$, then $r_1 = -\sum_{k=2}^n D_{1,k} r_k / D_{1,1}$. Therefore, we can use the law of total probability to write our event of interest as

$$\begin{aligned}
\Pr[\sum_{k=1}^n D_{1,k} r_k \neq 0] &= \sum_{b_2, b_3, \ldots, b_n \in_r \{0,1\}} \Pr\left[r_1 = \frac{-\sum_{k=2}^n D_{1,k} r_k}{D_{1,1}} \cap (r_2, r_3, \ldots, r_n) = (b_1, b_2, \ldots, b_n)\right] \\
&= \sum_{b_2, b_3, \ldots, b_n \in_r \{0,1\}} \Pr\left[(r_2, r_3, \ldots, r_n) = (b_1, b_2, \ldots, b_n)\right] \Pr\left[r_1 = \frac{-\sum_{k=2}^n D_{1,k} b_k}{D_{1,1}}\right] \\
&= \frac{1}{2}
\end{aligned}$$

□

An intuitive way to understand the proof is as follows. Suppose that you don't choose all the r_is together; rather you choose them one at a time. Say that we choose $r_2, r_3, \ldots, r_n$

one at a time. Once all of these are fixed, then the probability that you will choose the value for r_1 which will make the sum non-zero is $1/2$. This intuition is sometimes referred to as the *principle of deferred decisions*, and the way to formalize it is through the conditional probability calculations as illustrated above.

Another bothersome point for you might be the guarantee of the algorithm. The analysis shows that the algorithm correctly answers with probability $\geq 1/2$, which does not seem much. Notice that this algorithm has *one-sided error*. Therefore, to reduce the error probability of the algorithm, we need to just repeat it many times with vectors $\mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_k$ chosen at random and with replacement and check whether $AB\mathbf{r}_i \neq C\mathbf{r}_i$ for even one of them. Let E_i be the event that $AB\mathbf{r}_i = C\mathbf{r}_i$. We are interested in the event $\bigcup_{i=1}^{k} \overline{E}_i$. First we compute

$$\begin{aligned}\Pr\left[\bigcap_{i=1}^{k} E_i\right] &= \prod_{i=1}^{k} \Pr[E_i], \text{ since the events } E_i \text{ are independent} \\ &\leq \frac{1}{2^k}.\end{aligned}$$

Therefore, we have

$$\begin{aligned}\Pr\left[\bigcup_{i=1}^{k} \overline{E}_i\right] &= 1 - \Pr\left[\bigcap_{i=1}^{k} E_i\right] \\ &\geq 1 - \frac{1}{2^k}.\end{aligned}$$

Observe that if we had sampled $\mathbf{r}_i$ without replacement, then the probability of error can only reduce further since if in the first iteration you do not find a vector $\mathbf{r}$ such that $AB\mathbf{r} \neq C\mathbf{r}$, then in the next iteration you have reduced the space from which you are sampling. But, the number of vectors $\mathbf{r}$ such that $AB\mathbf{r} \neq C\mathbf{r}$ has not reduced. While analyzing such experiments where you are sampling elements one after the other without replacement, the analysis is slightly more complicated due to the conditional probability calculations that you have to do. Moreover, it is easier to implement sampling with replacement than without replacement. In most cases, the bounds that we obtain are useful for the problem at hand.

We will not explicitly write down the sample space and event in every problem that we study if it is clear from the context. We will end this lecture with an important graph problem that arises as a primitive in many scenarios.

1.4 Minimum cut

A *cut-set* in a graph $G(V, E)$ is a collection of edges $E' \subseteq E$ such that the graph gets disconnected. A minimum cut is a cut-set with the minimum cardinality among all cut sets. It is possible that there are more than one minimum cuts in the graph. The simple algorithm that we are going to describe was first given by David Karger in 1993.

The idea of the algorithm is to choose an edge uniformly at random, and contract the end points to obtain a multigraph. In each iteration, the number of vertices reduce by one, and

hence after $n-2$ iterations, there are two vertices remaining. The algorithm outputs the number of edges between these two vertices as the minimum cut of the graph. You can see that the final cut that is obtained is also a cut in the original graph, but it is possible that it is not a minimum cut. We will bound the probability that this procedure does not give a minimum cut.

Algorithm 2: KARGER'S MIN-CUT

Input: Graph $G(V, E)$ with n vertices

```
repeat
    Choose e uniformly at random from G
    G ← G \ e
until |V| = 2;
Return the number of edges in G
```

We have the following lemma about Karger's algorithm.

LEMMA 1.5. *Algorithm 2 returns the minimum cut with probability* $\frac{2}{n(n-1)}$.

Proof. We will argue with respect to a fixed cut C of size k, and compute the probability that this cut remains after the execution of Algorithm 2. A fixed cut C remains after the execution of the algorithm if during every random choice, no cut-edge was chosen.

If the minimum cut size is at most k, every vertex must have degree at least k. Thus there are at least $nk/2$ edges in the graph. Let E_i be the event that the edge contracted in the i^{th} iteration of the algorithm is not an edge in C. Let $F_i = \cap_{j=1}^{i} E_i$ be the event that no edge from C was contracted in the first i iterations. We are interested in computing $\Pr[F_{n-2}]$.

Firstly, $\Pr[E_1] = \Pr[F_1] \geq 1 - \frac{2k}{nk} = 1 - \frac{2}{n}$. If a cut edge was not chosen in the first iteration, we have an $(n-1)$-vertex graph with cut size equal to k. In particular, if a cut edge was not chosen in the first $i-1$ iterations, then we are left with an $(n-i+1)$-vertex graph with cut size equal to k. Thus, we can say that $\Pr[E_i | F_{i-1}] \geq 1 - \frac{2k}{k(n-i+1)} = 1 - \frac{2}{n-i+1}$. Finally, we have

$$\begin{aligned}
\Pr[F_{n-2}] &= \Pr[E_{n-2} \cap F_{n-3}] \\
&= \Pr[E_{n-2} | F_{n-3}] \Pr[F_{n-3}] \\
&= \left(\prod_{i=1}^{n-3} \Pr[E_{i+1} | F_i] \right) \Pr[F_1] \\
&\geq \left(1 - \frac{2}{n}\right) \prod_{i=1}^{n-3} \left(1 - \frac{2}{n-i}\right) = \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)}.
\end{aligned}$$

□

Once again we note that the success probability of the algorithm is small, but the algorithm has one-sided error. So, we do the standard trick of repeating the algorithm independently k

times and taking the minimum value among the results. Analyzing this like in the previous section, the probability that the minimum cut is not output in any of the k iterations is at most

$$\left(1 - \frac{2}{n(n-1)}\right)^k \leq e^{-\frac{2k}{n(n-1)}},$$

where we use the inequality that $1 - x \leq e^{-x}$ (remember this! It is a standard inequality used in a lot of calculations). Thus if we were to repeat this algorithm for $k = n(n-1)\log n$ times, the error probability is at most $1/n^2$.

2 Random variables and their properties

The outcome of a randomized algorithm is a variable that depends on the random choices made by the algorithm during its execution. Analyzing the guarantees of a randomized algorithm amounts to analyzing the properties of this *random variable*. Formally, a random variable is a function $X : \Omega \to \mathbb{R}$. The standard notational convention is to use capital letters to denote random variables. We can associate events with random variables in a very natural way. If a random variable X takes a value a, then the event associated with this is the set $A \subseteq \Omega$ such that $\omega \in A$ iff $X(\omega) = a$. We will also write that

$$\Pr[X = a] = \sum_{\omega \in \Omega, X(\omega)=a} \Pr[\omega].$$

While analyzing randomized algorithm, we will need to define suitable random variables based on the algorithm at hand. In many cases, it will difficult to analyze the random variable corresponding to the output of the algorithm directly, and we may need to express it as a function of other random variables. One important type of random variables that we will see is the *indicator random variable*. For an event $E \subseteq \Omega$, an indicator random variable X_A takes values 1 or 0 depending on the occurrence of the event A. One of the most important parameters associated with a random variable is its *expectation*. The expectation of a random variable is the average value taken by it. Formally, we have

Definition 2.1 (Expectation). *Let Ω be a sample space, and let $X : \Omega \to \mathbb{R}$ be a random variable. The expectation of the random variable $\mathbb{E}[X]$ is given by*

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} \Pr[\omega] X(\omega).$$

Similar to the independence of events, we can define when two random variables are independent. We will say that two random variables X and Y are *independent* if for every x and y, we have

$$\Pr[(X = x) \cap (Y = y)] = \Pr[X = x] \Pr[Y = y].$$

An important property of expectation that will useful in analyzing random variables and algorithms is the following seemingly trivial property. The proof is a simple calculation from the definition, and is left as an exercise.

THEOREM 2.2. *(Linearity of Expectation) Let $X_1, X_2, \ldots, X_n$ be any n random variables, and let $X = \sum_{i=1}^{n} X_i$. Then, we have*

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i].$$

Notice that the statement makes no assumption about the properties of the random variables. With these ideas, we can already say something non-trivial about an important combinatorial problem.

2.1 Maxcut

In the mincut problem that we saw in the last lecture, our aim was to find the cut of smallest size in a graph G. Now, we ask the complement question: Given a graph G, find the cut of largest cardinality. It is a central problem in combinatorial optimization, and no efficient algorithms are known for it. The problem is NP-hard, and hence unlikely to have an efficient algorithm. Let us look at a simple randomized algorithm for this problem.

Algorithm 3: Max-Cut

Input: Graph $G(V, E)$

Set $V_1, V_2 \leftarrow \emptyset$
for $u \in V$ **do**
    Choose b u.a.r from $\{0, 1\}$
    if $b = 0$ **then** $V_1 \leftarrow V_1 \cup \{u\}$ **else** $V_2 \leftarrow V_2 \cup \{u\}$;
end
Output $X = |\{(u, v) \in E | u \in V_1, v \in V_2\}|$.

Firstly, if we analyze this algorithm with respect to a fixed cut C, then the probability that this cut size is output is $1/2^n$. If this probability was any better, then we could have repeated this for sufficient number of times and obtained a fast algorithm. What we will do instead is to look at the random variable X and obtain some non-trivial bound on the size of the maxcuts of a graph. The output X is a random variable that depends on the random choices made for the vertices in the graph. First, we show the following.

LEMMA 2.3. *For X output in Algorithm 3, $\mathbb{E}[X] = |E|/2$.*

Proof. For each edge $e \in E$, define the indicator random variable X_e that denotes the event that e is a cut-edge. We can then write

$$X = \sum_{e \in E} X_e.$$

By the linearity of expectation, $\mathbb{E}[X] = \sum_{e \in E} \mathbb{E}[X_e]$. So, all that remains now is to compute the expectation of X_e. For this we use the observation that for an indicator random variable, the expectation is equal to the probability of the occurrence of the corresponding event. In this case, we have $\mathbb{E}[X_e] = \Pr[e \text{ is a cut edge}]$. Suppose that $e = (u, v)$, then we have

$$\begin{aligned}\Pr[e \text{ is a cut edge}] &= \Pr[(u \in V_1 \text{ and } v \in V_2) \cup (u \in V_2 \text{ and } v \in V_1)] \\ &= \Pr[(u \in V_1 \text{ and } v \in V_2)] + \Pr[(u \in V_2 \text{ and } v \in V_1)] = \frac{1}{2}.\end{aligned}$$

Consequently, $\mathbb{E}[X] = |E|/2$. □

Notice that the expectation here is a weighted mean over all the cut sizes. Consequently, if the weighted mean is greater than a number r, then there must actually exists such a cut of size at least r. In fact, this observation gives us the following highly non-trivial theorem about cuts in graphs.

THEOREM 2.4. *Every graph G with m edges has a cut of size at least $m/2$.*

This method of proving the existence of combinatorial objects using randomization is known as the *probabilistic method*. It is a very powerful tool that can be used in a variety of settings that do not seem amenable to such techniques on first sight.

Let us try to understand the random variables $\{X_e\}_{e \in E}$ a little better. Firstly, for two edges e_1 and e_2, X_{e_1} and X_{e_2} are independent. If e_1 and e_2 don't share a vertex, then it is obvious. Suppose that e_1 and e_2 share a vertex u. Even in this case, if given that X_{e_1} is 0 or 1, the value of X_{e_2} depends on where the other endpoint is placed. Thus $\Pr[X_{e_2} = 1 | X_{e_1} = 1] = \Pr[X_{e_2} = 1 | X_{e_1} = 0] = 1/2$. Now suppose that the edges e_1, e_2, and e_3 for a triangle on vertices u, v, and w. Now give the values of X_{e_1} and X_{e_2}, the positions of all three vertices are fixed, and hence the value of X_{e_3}. Thus, the variables $\{X_e\}_{e \in E}$ are not fully independent. The linearity of expectation gave us a non-trivial bound, even when we had no information about the properties of the random variables involved other than their expectation.

2.1.1 ⋆ Constructing a large cut deterministically

We showed that a random partition of the vertices gives you a cut, that on expectation contains at least half of the edges in the graph. But, can you construct such a cut-set deterministically. Recall the analysis of the expectation. If you look at it carefully, you'll notice that we don't really require that the assignment of vertices to V_1 and V_2 need not be completely independent. For the analysis to go through, all we need is the following:

$$\begin{aligned}\Pr[u \in V_1 \text{ and } v \in V_2] &= \Pr[u \in V_1] \Pr[v \in V_2], \\ \Pr[u \in V_2 \text{ and } v \in V_1] &= \Pr[u \in V_2] \Pr[v \in V_1]\end{aligned}$$

Thus, what we actually need is just pairwise independence, rather than full independence for the analysis to go through. Instead of choosing the part for each vertex uniformly at random, we do the following.

Let $\mathbf{b} = b_1 b_2 \dots b_k$ be a binary string such that $k = \log n$. We can associate subsets of $[k]$ with the n vertices in the graph in a natural way. Let $S_u \subseteq [k]$ denote the set associated with vertex u. For a vertex u, we compute $b_u = \oplus_{i \in S_u} b_i$, and assign u to V_b. Now, for any two u, v, the set S_u and S_v are different, and hence if $\mathbf{b}$ is chosen uniformly at random from $\{0,1\}^{\log n}$, the probability $\Pr[b_u \neq b_v] = 1/2$. Therefore, we can rewrite the maxcut algorithm as follows.

Algorithm 4: Max-Cut Modified

Input: Graph $G(V, E)$

Set $V_1, V_2 \leftarrow \emptyset$
Choose $b_1, b_2, \dots, b_{\log n} \in_r \{0,1\}$
for $u \in V$ **do**
    Compute $b = \oplus_{i \in S_u} b_i$
    Set $V_b \leftarrow V_b \cup \{u\}$.
end
Output $X = |\{(u,v) \in E | u \in V_1, v \in V_2\}|$.

The analysis of this algorithm is identical to the original algorithm. But, now we can remove the randomness in the following way:

Algorithm 5: Max-Cut Deterministic

Input: Graph $G(V, E)$

Set $V_1, V_2 \leftarrow \emptyset$
for $\mathbf{b} = (b_1, b_2, \dots, b_{\log n})$ **do**
    for $u \in V$ **do**
        Compute $b = \oplus_{i \in S_u} b_i$
        $V_b \leftarrow V_b \cup \{u\}$
    end
    Compute $X_{\mathbf{b}} = |\{(u,v) \in E | u \in V_1, v \in V_2\}|$
end
Output $\max\{X_{\mathbf{b}}\}_{\mathbf{b} \in \{0,1\}^{\log n}}$

Since the expected value is at least $m/2$, the largest cut must have size at least $m/2$. Therefore, the output will be a cut of size at least $m/2$. Furthermore, the outer loop runs for n steps, and hence we have a polynomial-time $1/2$-approximation algorithm.

2.2 Quicksort

We use the ideas from the previous section to analyze a randomized version of the Quicksort algorithm. Recall that in the deterministic Quicksort algorithm, we choose a pivot element in

the array (say, the last element) and then partition the array into two parts based on whether the numbers are smaller or larger than the pivot. Then we recursively sort the two parts to obtain the final sorted array. The running time depends on the choice of the pivot since we want the pivot to divide the array into two arrays of similar sizes.

For instance, if the pivot is the largest element of the array each time, then the running time is $O(n^2)$. On the other hand, if we choose the median as the pivot at every step, then the running time is given by the recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$$

Let us look at a randomized variant of Quicksort, where we choose a pivot element uniformly at random from the array. We hope that the algorithm will avoid the case of choosing a bad pivot at every step of the recursion. In this case, the running time of the algorithm, as measured by the number of comparisons that it makes, is a random variable. Let us analyze this random variable now.

For this, let $\mathbf{a} = a_1, a_2, \ldots, a_n$ be the array given and let $\mathbf{b} = b_1, b_2, \ldots, b_n$ be the sorted version of the array. We will argue with respect to this array $\mathbf{b}$. Let X denote the total number of comparisons performed ny the Quicksort algorithm while converting the array $\mathbf{a}$ to $\mathbf{b}$. It will be hard to argue directly about X, so we will express X as a function of random variables that are easier to analyze.

To that end, let X_{ij} denote an indicator random variable corresponding to the event that b_i and b_j are compared during the execution of randomized Quicksort. We can then write X as

$$X = \sum_{1 \le i < j \le n} X_{ij}, \text{ and hence,}$$
$$\mathbb{E}[X] = \sum_{1 \le i < j \le n} \mathbb{E}[X_{ij}].$$

Since X_{ij} is an indicator random variable, $\mathbb{E}[X_{ij}] = \Pr[b_i \text{ and } b_j \text{ are compared}]$.

Let us try to analyse X_{ij}. Consider the elements $b_i \le b_{i+1} \le \ldots \le b_j$. The only way that b_i and b_j are compared is that the first time a pivot is chosen from this interval, it is either b_i or b_j. Since the pivot is chosen uniformly at random from the entire array, $E[X_{ij}] = \frac{2}{j-i+1}$.

Thus we can write

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} = \sum_{k=2}^{n} \sum_{i=1}^{n+1-k} \frac{2}{k} = \sum_{k=2}^{n} (n+1-k)\frac{2}{k} \\
&= 2(n+1) \sum_{k=2}^{n} \frac{1}{k} - 2(n-1) \\
&= 2(n+1) \sum_{k=1}^{n} \frac{1}{k} - 4n
\end{aligned}$$

The sum $\sum_{k=1}^{n} \frac{1}{k}$ is the harmonic sum and is equal to $\ln n + \Theta(1)$. Therefore, we have $\mathbb{E}[X] = 2n \ln n + \Theta(n)$. If you recall analysis of algorithms, this might seem a little unsatisfactory. It is true that on expecation, Quicksort is fast, but how is the random variable actually distributed? Is it the case that the running time can vary wildly, and yet the expectation is $\Theta(n \log n)$? Or is the actual running time concentrated closely around the expectation? That would be a more useful analysis. For now, we will leave it here, but do a more careful analysis once we develop a few more tools.

2.3 Probability Mass Functions (PMF) and random variables

Till now, we have been looking at randomized algorithms, and defined random variables based on the algorithm at hand. Another way to define random variables (without looking at any random experiment) is to define the corresponding probability mass function. We will see come distributions that arise in many different scenarios, and will be useful in the analysis of randomized algorithms. Let us start with the formal definition of a Probability Mass Function (PMF).

Definition 2.5. *Let X be a random variable. The PMF associate with X is a function $p_X : \mathbb{R} \to \mathbb{R}$ defined as follows:*

$$p_X(u) = \Pr[X = u].$$

Notice that the value $p_X(u) = 0$ if u is not in the range of $X : \Omega \to \mathbb{R}$. The events $\{X = u\}$ is a partition of Ω, and hence $\sum_{u \in \text{range}(X)} p_X(u) = 1$. The PMF of a random variable gives you all the information you need to analyze the random variable X. While analyzing a randomized algorithm, we usually don't have access to the PMF directly, but there are some common PMFs that arise in different random process.

2.3.1 Binomial distribution

A common random experiment that arises in many scenarios is the following. Consider an experiment that succeeds with probability p and fails with probability $1-p$. We can associate an indicator random variable with this experiment that is 1 iff the experiment succeeds. Such an indicator random variable is known as a Bernoulli random variable. A natural question that comes up in many situations is to count the number of successes in a run of n iterations of this experiment. This random variable is known as the binomial random variable and the associated PMF is the binomial distribution.

Definition 2.6. *A random variable X is a binomial random variable with parameters n and p if the PMF of the random variable X is given by*

$$p_X(i) = \binom{n}{i} p^i (1-p)^{n-i}$$

for $i \in \{0, 1, \ldots, n\}$ and 0 otherwise.

Alternately, you can think of the binomial random variable being the sum of n Bernoulli random variables, though the PMF defines the random variable completely as far as we are concerned. If we think of X as the sum of n Bernoulli random variables with parameter p, then by the linearity of expectation, we can easily conclude that $\mathbb{E}[X] = np$. If we were think of the binomial random variable purely in terms of its PMF, then we can explicitly compute its expectation as follows:

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=0}^{n} i \binom{n}{i} p^i (1-p)^{n-i} \\
&= \sum_{i=1}^{n} \frac{n!}{(i-1)!(n-i)!} p^i (1-p)^{n-i} = np \sum_{i=1}^{n} \frac{(n-1)!}{(i-1)!(n-i)!} p^{i-1} (1-p)^{n-i} \\
&= np \sum_{j=0}^{n-1} \binom{n-1}{j} p^j (1-p)^{n-1-j} = np.
\end{aligned}$$

2.3.2 Geometric distribution

Consider a Bernoulli trial with parameter p. We are interested in the number of trials that has to be performed before the first success. The random variable that counts this number is known as a geometric random variable.

Definition 2.7. *A random variable X is said to be a geometric random variable with parameter p if the PMF associated with X is given by*

$$p_X(i) = (1-p)^{i-1} p.$$

Given the PMF, we can once again explicitly calculate the expectation from the definition.

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i\geq 1} i(1-p)^{i-1}p \\
&= \sum_{i\geq 1}\sum_{j=1}^{i}(1-p)^{i-1}p = \sum_{j\geq 1}\sum_{i\geq j} p(1-p)^{i-1} \\
&= \sum_{j\geq 1} p(1-p)^{j-1}\sum_{i\geq 0}(1-p)^{i} = \sum_{j\geq 1}(1-p)^{j-1} \\
&= \frac{1}{p}.
\end{aligned}$$

An alternate way to compute this probability uses the definition of the random variable to prove the following property of geometric random variables: the probability that the first success will be after n trials from now is independent of the number of failures that you have encountered. To formally state this, we introduce the notion of *conditional expectation*.

For random variables X and Y, the conditional expectation of X given that $Y = y$ is given by the sum

$$\mathbb{E}[X|Y = y] = \sum_{x} x \Pr[X = x|Y = y].$$

In other words, given that the event $\{Y = y\}$ has occured, what is the new expected value of the random variable X. The following facts about conditional expectation is easy verify.

Fact 2.8. *For any two random variables X and Y,*

$$\mathbb{E}[X] = \sum_{y} \Pr[Y = y]\mathbb{E}[X|Y = y].$$

The linearity property naturally extends to conditional expectation as well.

Fact 2.9. *Let $X_1, X_2, \ldots, X_n$ be n random variables with finite expectation and Y be any random variable. Then,*

$$\mathbb{E}\left[\left(\sum_i X_i\right)|Y = y\right] = \sum_i \mathbb{E}[X_i|Y = y].$$

We now formalize the property about geometric random variables stated above.

Theorem 2.10. *Let X be a geometric random variable with parameter p. Then for any $n > 0$, and k, we have*

$$\Pr[X = n + k|X > k] = \Pr[X = n]$$

Proof.

$$\begin{aligned}\Pr[X = n+k | X > k] &= \frac{\Pr[(X = n+k) \cap (X > k)]}{\Pr[X > k]} \\ &= \frac{\Pr[X = n+k] \Pr[X > k | X = n+k]}{\Pr[X > k]} \\ &= \frac{(1-p)^{n+k-1}p}{\sum_{i \geq k}(1-p)^k p} = (1-p)^{n-1}p.\end{aligned}$$

□

We can now obtain a different derivation for the geometric random variable using conditional expectation and the property proved above. Let X be a geometric random variable with parameter p, and let Y be an indicator random variable that is 1 if the first trial succeeded. Now, we can write $\mathbb{E}[X]$ as follows:

$$\begin{aligned}\mathbb{E}[X] &= \Pr[Y = 0]\mathbb{E}[X|Y = 0] + \Pr[Y = 1]\mathbb{E}[X|Y = 1] \\ &= (1-p)\mathbb{E}[X|Y = 0] + p\mathbb{E}[X|Y = 1]\end{aligned}$$

If $Y = 1$, then $X = 1$ and hence $\mathbb{E}[X|Y = 1] = 1$. But, if $Y = 0$, then we can write $X = Z + 1$ where Z is again a geometric random variable due to Theorem 2.10. Thus we have

$$\begin{aligned}\mathbb{E}[X] &= (1-p)\mathbb{E}[Z+1] + p = (1-p)\mathbb{E}[Z] + 1 = (1-p)\mathbb{E}[X] + 1 \\ &= \frac{1}{p}\end{aligned}$$

The coupon collector's problem

We will briefly study the coupon collector's problem, that is a special case of a more general paradigm, that arises in many situations in the analysis of randomized algorithms. The basic premise of the problem is the following: There are boxes of cereals, each of which contains one among n different coupons. The coupons are assigned to boxes of cereals at random. In this case, how many boxes of cereals should you buy so that you have at least one coupon of each kind. We will calculate the expected number of boxes that you must buy to collect one coupon of each kind.

So, we want to analyze the random variable X which is number of boxes bought until one coupon of each kind is obtained. Instead of directly arguing about X, we try to write X in terms of simpler random variables that can be analyzed easily. Let X_i denote the number of boxes purchased after obtaining $i-1$ different coupons. Clearly, we can write $X = \sum_{i=1}^{n} X_i$, and hence $\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i]$. What we will see is that X_i is a geometric random variable.

If $i-1$ different coupons have been collected, then the probability that in the next step, a new coupon is obtained is $1 - \frac{i-1}{n}$. Since, the coupons are distributed randomly among the boxes, this probability remains the same for the consequent steps as well. Thus, X_i is a

geometric random variable with parameter $p = 1 - \frac{i-1}{n}$. Therefore, $\mathbb{E}[X_i] = \frac{n}{n-i+1}$. Now, we have

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} \\ &= n \sum_{i=1}^{n} \frac{1}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i} \\ &= n \ln n + \Theta(n).\end{aligned}$$

As mentioned earlier, the expectation alone does not provide a satisfactory answer regarding a random variable. In particular, while analyzing randomized algorithms, we will need to measure how closely the random variable associated with the algorithm is concentrated around the expectation. In this particular case of the coupon collector problem, we can do a simpler analysis that shows that the number of boxes that have to be purchased in order to see all the coupons is close to $n \log n + \Theta(n)$ with good probability.

To do that, let E_i be the event that the first coupon did not appear in the k boxes that were bought. The probability for this event is $\Pr[E_i] = \left(1 - \frac{1}{n}\right)^k \leq e^{-k/n}$. When $k = n \log n + 10n$ say, this probability is bounded by $e^{-10} n^{-1}$. We are interested in the even $\cap \overline{E}_i$ which can be written as

$$\begin{aligned}\Pr\left[\bigcap \overline{E}_i\right] &= 1 - \Pr\left[\bigcup E_i\right] \\ &\geq 1 - \sum \Pr[E_i], \text{ by the union bound} \\ &\geq 1 - e^{-10}\end{aligned}$$

3 Moments, deviation and concentration

In this lecture, we will analyze the random variable more closely by looking at its deviation from the mean and concentration around the mean. We will start with a simple bound about how far a random variable can go away from its expectation. We will not assume anything about the random variable, other than it being non-negative, and we will see that the bound we get is not optimal. As we discover more properties about the random variable, we will prove stronger statements about how the random variable is distributed with respect to its expectation.

3.1 Markov's inequality

We start with a simple inequality about a non-negative random variable.

Theorem 3.1 (Markov's Inequality). *Let X be a non-negative random variable, and let $a > 0$. Then,*

$$\Pr[X \geq a] \leq \frac{\mathbb{E}[X]}{a}.$$

Proof. Define an indicator random variable I such that $I = 1$ iff $X \geq a$. Thus we have $\mathbb{E}[I] = \Pr[X \geq a]$. Furthermore, we have $I \leq X/a$ since X is non-negative and $a > 0$, and if $I = 1$, then $X \geq a$. Therefore, we can write

$$\Pr[X \geq a] = \mathbb{E}[I] \leq \mathbb{E}\left[\frac{X}{a}\right] = \frac{\mathbb{E}[X]}{a}.$$

□

While this may not seem much, we can already say something non-trivial about a few algorithms already. Let's try to analyze Quicksort better. We already know that the expected running time is $O(n \log n)$. But, what is the probability that the running time is close to $O(n \log n)$.

3.1.1 Quicksort revisited

Recall that we can draw the recursion tree of Quicksort, where each node is the array under consideration. Initially, the whole array is considered, and depending on the choice of the pivot this array is split into two different arrays. The first observation that we can make is that the total number of comparisons done at every level of the recursion tree is $O(n)$. So what we want to compute is the probability that the depth of the recursion tree is at most $O(\log n)$. We will argue with respect to individual elements in the array, and then take a union bound.

Fix an element a in the array. Let X_i denote the size of the sub-array containing a at the i^{th} level of the recursion. Note that $X_0 = n$ since at the root of the recursion, the entire array is considered. Suppose that $X_{i-1} = t$, we will first compute the conditional expectation $\mathbb{E}[X_i | X_{i-1} = t]$ as follows:

$$\mathbb{E}[X_i | X_{i-1} = t] \leq \frac{1}{2} \cdot \frac{3}{4}t + \frac{1}{2} \cdot t = \frac{7}{8}t.$$

This is because with probability $1/2$, a randomly chosen pivot will partition the array into two parts, each of which is at most $3/4^{th}$ of the original array. Now, we can write

$$\begin{aligned}\mathbb{E}[X_i] &= \sum_t \Pr[X_{i-1} = t]\mathbb{E}[X_i | X_{i-1} = t] \leq \sum_t \frac{7}{8}t \Pr[X_{i-1} = t] \\ &= \frac{7}{8}\mathbb{E}[X_{i-1}].\end{aligned}$$

We can inductively bound the value $\mathbb{E}[X_{i-1}]$, and obtain $\mathbb{E}[X_i] \leq \left(\frac{7}{8}\right)^i n$. For $i = 3\log_{8/7} n$, the value $\mathbb{E}[X_i] \leq 1/n^2$. Hence, we can use Markov's inequality to conclude that for $i = 3\log_{8/7} n$,

$$\Pr[X_i > 1] \leq \mathbb{E}[X_i] \leq \frac{1}{n^2}.$$

In other words, the probability that the part of the array containing a occurs below the $3\log_{8/7} n$-th level is at most $1/n^2$.

For an array $A = (a_1, a_2, \ldots, a_n)$ denote the array we are sorting. Let E_i denote the event that the element a_i is in a subarray of size greater than 1 at a recursion level $3\log_{8/7} n$. We have just computed that for every i, the probability $\Pr[E_i] \leq 1/n^2$. We are interested in the event that none of the E_is occur. Therefore,

$$\begin{aligned}\Pr\left[\bigcap_{i=1}^{n} \overline{E}_i\right] &= 1 - \Pr\left[\bigcup_{i=1}^{n} E_i\right] \\ &\geq 1 - \sum_{i=1}^{n} \Pr[E_i] \text{(due to the union bound)} \\ &\geq 1 - \frac{1}{n}.\end{aligned}$$

Thus, let $T(n)$ denote the running time of the randomized Quicksort algorithm. What we have just shown is that

$$\Pr[T(n) > 3\log_{8/7} n] \leq \frac{1}{n}.$$

If we know more properties of the random variable X, we can naturally say stronger statements about its deviation from the mean. This is what we see next.

3.2 Chebyshev's inequality

To understand how a random variable is distributed around its expectation, we first look at the variance, which tells the expected deviation of the random variable from its expected value. Formally, for a random variable X, the variance $\mathrm{Var}(X)$ is defined as follows:

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2].$$

Equivalently, $\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. The quantity $\mathbb{E}[X^k]$ is known as the k^{th} moment of the random variable X. The standard deviation σ is defined as $\sqrt{\mathrm{Var}(X)}$. We briefly recall some facts about variance.

Fact 3.2. *Let X and Y be two random variables. Then*

$$\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

The quantity $\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$ is known as the covariance of X and Y, denoted as $\mathrm{Cov}(X, Y)$. If X and Y are independent, then $\mathrm{Cov}(X, Y) = 0$, and hence $\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$.

Knowing the variance of a random variable, we can say something stronger about its deviation from the expectation. This is given by Chebyshev's inequality.

Theorem 3.3 (Chebyshev's inequality). *Let X be any random variable. For any $a > 0$,*

$$\Pr\left[|X - \mathbb{E}[X]| \geq a\right] \leq \frac{\mathrm{Var}(X)}{a^2}.$$

Proof. We can write $\Pr[|X - \mathbb{E}[X]| \geq a] = \Pr[(X - \mathbb{E}[X])^2 \geq a^2]$. Now, we can apply Markov's inequality on the random variable $(X - \mathbb{E}[X])^2$, to get

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

□

Sometimes, we use the following form of this inequality as well.

$$\Pr[|X - \mathbb{E}[X]| \geq t\sigma] \leq \frac{1}{t^2}, \text{ where } t > 1 \text{ and } \sigma \text{ is the standard deviation .}$$

3.2.1 Probability amplification using fewer random bits

We have seen how we can amplify the success probability of algorithm with one-sided error by repeating the algorithm k times. If n random bits are required for one run of the algorithm, then we need kn random bits to repeat the algorithm and reduce the error probability to 2^{-k}. Is it possible to use only $O(n)$ random bits and repeat the experiment k times, and yet improve the success probability?

Let us assume that we have an algorithm $\mathcal{A}$ that uses n random bits. If you don't want to bother about an arbitrary algorithm, you can think about the matrix multiplication verification algorithm, say, that we have seen already. To improve the success probability to $1 - 2^{-k}$, we used the fact that independent random bits were used in the different iterations of the algorithm. To reduce the number of random bits used, we will forgo this requirement. We have seen something similar already when removing the randomness in the maxcut algorithm.

We will first start with some technicalities that we will need. Let p be a prime, and let $\mathbb{Z}_p$ denote the set of numbers $\{0, 1, \ldots, p-1\}$. For two numbers a, b in $\mathbb{Z}_p$, we will denote by $a+b$ and $a \cdot b$, the operations of addition and multiplication module the prime p. The structure $\mathbb{Z}_p$ is known as a finite field, and for our purposes, you can assume that it behaves nicely like the rationals or reals.

LEMMA 3.4. *Let p be a prime. Suppose that a and b are chosen uniformly and independently at random from $\mathbb{Z}_p$, then the random variables $Y_1, Y_2, \ldots, Y_{p-1}$ defined as $Y_i = a \cdot i + b$ are*

1. *distributed uniformly over $\mathbb{Z}_p$, and*
2. *are pairwise independent.*

Proof. Let us start with showing that each of the Y_is are distributed uniformly over $\mathbb{Z}_p$. Fix an $i \in \mathbb{Z}_p$. Now for any $c \in \mathbb{Z}_p$, for every choice of b, there is one and only one choice of a such that $c = ai + b$. Therefore $\Pr[Y_i = c] = 1/p$ for every $c \in \mathbb{Z}_p$.

Next, we want to show that for $i \neq j$, for every $c_1, c_2 \in \mathbb{Z}_p$ we have

$$\Pr[Y_i = c_1 \text{ and } Y_j = c_2] = \frac{1}{p^2}.$$

This follows from the observation that the system of linear equations over $\mathbb{Z}_p$ given by

$$\begin{aligned} c_1 &= ai + b, \\ c_2 &= aj + b \end{aligned}$$

has a unique solution over $\mathbb{Z}_p$, where $a = (c_1 - c_2)(i - j)^{-1}$ and $b = (c_1 j - c_2 i)(j - i)^{-1}$. □

Let $\mathcal{A}$ be an algorithm that uses n bits of randomness such that if x is a true input, then $\Pr[\mathcal{A}(x) = 1] = 1$, and if x is a false input, then $\Pr[\mathcal{A}(x) = 1] \leq 1/2$. Let $N = 2^n$, and let $p > N$ be a prime. It is known that there is a prime number between N and $2N$, and we will use numbers in $\mathbb{Z}_p$ as our random strings.

Remark 3.5. *At this point, there is a small subtlety in that we need a way to obtain the prime p efficiently. But, what we actually used was that the structure $\mathbb{Z}_p$ is a finite field. It is well known, from abstract algebra, that for every prime p and integer n, there is one and only one finite field of cardinality p^n up to isomorphism. This is sometimes referred to as the Galois field $\mathbb{GF}(p^n)$. This finite field is not the set $\{0, 1, \ldots, p^{n-1}\}$. Furthermore, given p and n, there is an efficient way to construct this finite field so that we can sample from it. In our case, we will construct the finite field $\mathbb{GF}(2^n)$ and work within this field. The analysis that we will do now, also works exactly the same way in $\mathbb{GF}(2^n)$. We will not get into these technicalities for now.*

Now, instead of sampling k random strings independently to repeat the algorithm $\mathcal{A}$, we choose a, b uniformly at random from $\mathbb{Z}_p$, and then we choose the k random strings to be used as $\{ai + b\}_{1 \leq i \leq k}$. Let $Y_i = ai + b$, and let X_i denote the indicator random variable that is 1 iff the algorithm $\mathcal{A}$ return 0 when Y_i is used as the random string. Since Y_is are pairwise independent, the variables X_is are also pairwise independent. Now, if an input x is a no instance, then $\Pr[X_i = 1] \geq 1/2$. We are interested in the random variable $X = \sum_{i=1}^{k} X_i$ and the probability of the event $X = 0$. We can compute the expecation $\mathbb{E}[X] \geq k/2$.

From the properties of variance that we saw earlier, we can show that if X_is are pairwise independent, then

$$\mathrm{Var}\left(\sum_{i=1}^{k} X_i\right) = \sum_{i=1}^{k} \mathrm{Var}(X_i).$$

Now, the X_i are Bernoulli random variables in this case, with $p \geq 1/2$. We can compute its variance as

$$\mathrm{Var}(X_i) = \mathbb{E}[(X - \mathbb{E}[X])^2] = p(1-p)^2 + (1-p)(-p)^2 = p(1-p).$$

If $1 - p \leq 1/2$, then $\mathrm{Var}(X_i) \leq 1/4$. Therefore, $\mathrm{Var}(X) \leq k/4$. Now, we can compute the error probability of our algorithm as

$$\begin{aligned}
\Pr[X = 0] &\leq \Pr\left[|X - \mathbb{E}[X]| \geq \frac{k}{2}\right], \\
&\leq \frac{4\,\mathrm{Var}(X)}{k^2}, \text{ by Chebyshev's inequality} \\
&\leq \frac{1}{k}.
\end{aligned}$$

Notice, that the number of random bits that we require is at most $2 \log 2N \leq 2(n+1)$, and the error probability is at most $1/k$. If we had used independent random bits for each iteration, then we will require kn random bits and the error probability goes down to 2^{-k}. Probability amplification and reusing randomness is an important topic, and there are more sophisticated ways to do this. We will leave this discussion for now, and move to stronger concentration results. We will see that as we push towards stronger concentration inequalities, we will need to add more restrictions on the type of random variables that we are analyzing.

3.3 Chernoff-Hoeffding inequalities

In this section, we will some inequalities that give very strong concentration bounds for certain types of random variables. This is a tool that is used a lot in the design and analysis of randomized algorithms. In some cases, we might not be able to use the result directly, but Chernoff-Hoeffding type inequalities will still be useful. What this means is that the random variables may not satisfy the conditions to apply the Chernoff-Hoeffding inequalities, but a similar proof can be done (with small modifications) that will be applicable for that particular case. Thus, it is also important to understand the underlying ideas of the proof so that it can be applied in a new situation.

Let us start with n indicator random variables $X_1, X_2, \ldots, X_n$ such that $\Pr[X_i] = p_i$. These random variables are known as *Poisson trials*. Bernoulli trials are the case when all the p_is are equal. We are interested in the random variable $X = \sum_{i=1}^{n} X_i$. For a parameter $t > 0$, the function e^{tX} is known as the *moment generating function*. If we take the formal power series expansion of the expectation of this function, we have

$$\begin{aligned}\mathbb{E}[e^{tX}] &= \mathbb{E}[\sum_{i\geq 0} \frac{t^i X^i}{i!}] \\ &= \sum_{i\geq 0} \frac{t^i}{i!}\mathbb{E}[X^i].\end{aligned}$$

Now, we can say that for any $t > 0$,

$$\begin{aligned}\Pr[X > m] &= \Pr[e^{tX} > e^{tm}] \\ &\leq \frac{\mathbb{E}[e^{tX}]}{e^{tm}}, \text{ by Markov's inequality.}\end{aligned}$$

We will now compute the moment generating function as follows.

$$\begin{aligned}\mathbb{E}[e^{tX}] &= \mathbb{E}[e^{t\sum_{i=1}^{n} X_i}] \\ &= \mathbb{E}\left[\prod_{i=1}^{n} e^{tX_i}\right] \\ &= \prod_{i=1}^{n} \mathbb{E}[e^{tX_i}], \text{ since the } X_i\text{s are independent} \\ &= \prod_{i=1}^{n} \left(p_i e^t + 1 - p_i\right) = \prod_{i=1}^{n} \left(p_i(e^t - 1) + 1\right)\end{aligned}$$

Notice that $\sum_{i=1}^{n} p_i = \mathbb{E}[X]$ which we will denote as μ. We can use the AM-GM inequality to say that

$$\begin{aligned}\mathbb{E}[e^{tX}] &\leq \left(\sum_{i=1}^{n} \frac{p_i(e^t - 1) + 1}{n}\right)^n \\ &= (pe^t - q)^n,\end{aligned}$$

where $p = \sum_{i=1}^{n} p_i/n$ and $q = 1 - p$. This inequality is tight when all the p_is are equal, which corresponds to the case of sum of binomial random variables. Now, we can compute the probability

$$\Pr[X > (p+r)n] \leq \frac{\mathbb{E}[e^t X]}{e^{tn(p+r)}} \leq \left(\frac{pe^t + q}{e^{t(p+r)}}\right)^n$$

Since t is a parameter that we can choose, we can minimize the right-hand side to obtain the crude form (and the tightest) Chernoff bounds as

$$\begin{aligned}\Pr[X > (p+r)n] &\leq \left(\left(\frac{p}{p+t}\right)^{p+t}\left(\frac{q}{q-t}\right)^{q-t}\right)^n \\ &= \exp\left(-n\left((p+t)\ln\frac{p+t}{p} + (q-t)\ln\frac{q-t}{q}\right)\right).\end{aligned}$$

While this bound has a nice interpretation in terms of a certain notion of distance between probability distributions, we will now write a version of the bound that is easy to use in the analysis of the algorithms that we will see in this course.

THEOREM 3.6 (Chernoff bounds). *Let $X_1, X_2, \ldots X_n$ be independent indicator random variables with $\Pr[X_i = 1] = p_i$. Let $X = \sum_{i=1}^{n} X_i$ be the sum of the random variables. Then the following holds.*

1. *For every $t > 0$,*

$$\Pr[|X - \mathbb{E}[X]| > t] \leq e^{-2t^2/n}.$$

2. *For $\epsilon > 0$*

$$\begin{aligned}\Pr[X > (1+\epsilon)\mathbb{E}[X]] &\leq e^{-\epsilon^2\mathbb{E}[X]/3}, \\ \Pr[X < (1-\epsilon)\mathbb{E}[X]] &\leq e^{-\epsilon^2\mathbb{E}[X]/2}\end{aligned}$$

Hoeffding extended the bounds using a similar proof of bounding the moment generating function to obtain the following strengthening of the bound. This is also useful in many situations that we will encounter in the analysis of randomized algorithms.

THEOREM 3.7 (Hoeffding's extension). *Let $X_1, X_2, \ldots, X_n$ be independent random variables that take values in the interval $[a, b]$ such that $\mathbb{E}[X_i] = \mu$. Then,*

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right] \leq 2e^{-2n\epsilon^2/(b-a)^2}.$$

3.3.1 Probability amplification

The randomized algorithms that we have seen so far had one-sided error. For some problems, we may also have two sided error. We will look at problems that have a yes/no answer for now. In a two-sided error algorithm $\mathcal{A}$, if the input x is a yes instance then $\Pr[\mathcal{A}(x) = 1] \geq 2/3$, and if x is a no instance, then $\Pr[\mathcal{A}(x) = 0] \geq 2/3$. In other words, the algorithm can err on both sides (yes and no), but the error probability is at most $2/3$. The following is simple way to amplify the success probability of this algorithm. For now, we will not worry about reducing the number of random bits.

We will choose k random strings independently and uniformly at random, and run the algorithm $\mathcal{A}$ with these k random strings. We will then output the majority answer. If the original algorithm $\mathcal{A}$ had a running time $T(n)$, then this new algorithm has a running time of $O(kT(n))$. Let us calculate the error probability of this new algorithm.

Let us define k indicator random variables $X_1, X_2, \ldots, X_k$ where $X_i = 1$ iff the i^{th} iteration of $\mathcal{A}$ (with the k^th random string) outputs the correct answer. Notice that the X_is are Bernoulli random variables with parameter $p \geq 2/3$. Let $X = \sum_{i=1}^{k} X_i$ denote the number of times the algorithm answered correctly. Now, we have $\mathbb{E}[X] \geq 2k/3$. We want to compute the probability $\Pr[X \geq k/2]$. We can use a version of the Chernoff bounds here since

$$\Pr\left[X < \frac{k}{2}\right] = \Pr\left[X < \left(1 - \frac{1}{4}\right)\frac{2k}{3}\right]$$

Notice that here we don't know the exact value of the expectation of X, but rather a lower bound of this value. Therefore, we have

$$\begin{aligned}\Pr\left[X < \left(1 - \frac{1}{4}\right)\frac{2k}{3}\right] &\leq \Pr\left[X < \left(1 - \frac{1}{4}\right)\mathbb{E}[X]\right] \\ &\leq e^{-\mathbb{E}[X]/32} \leq e^{-k/48}.\end{aligned}$$

Thus if we repeat algorithm $\mathcal{A}$ k times, there is an exponential fall in the error probability.

3.3.2 Load balancing

We will now look at the problem of load balancing, where we have a set of processors and jobs arrive for scheduling on the processors. Our job is to distribute the jobs to the processors so that no processor is overloaded. This system works in a distributed environment, and we are interested in an efficient decentralized solution. Obviously, if we have the resources to check the load of each machine and assign jobs to machines, then we will achieve the optimal load. This is a case of the paradigm of *balls and bins* which is used to model various random processes.

Let us assume that there are k servers and n jobs where k is much smaller than n. We will look at the performance of a random job allocation algorithm where each job is assigned to a processor with probability $1/k$. Under this randomized strategy, each server has an expected load of n/k. This can happen in the worst case also under other cleverer strategies, but what

we will see is that the randomized strategy will not be too far with high probability. The advantage being that we need not remember any state information while allocating jobs to servers.

First, let's start with a fix server, and compute how many jobs will be allotted to it. Let X_i be the indicator random variables for the event that the i^{th} job is allocated to the server. We are interested in the random variable $X = \sum_{i=1}^{n} X_i$. We know that $\mathbb{E}[X] = n/k$. Since the random variables X_i are all independent, the variance $\mathrm{Var}(X) = n\frac{1}{k}\left(1 - \frac{1}{k}\right)$, which is approximately n/k. We can use Chernoff bounds to obtain the following.

$$\begin{aligned}\Pr\left[X > \frac{n}{k} + 3\sqrt{\frac{n\log k}{k}}\right] &= \Pr\left[X > \frac{n}{k}\left(1 + 3\sqrt{\frac{k\log k}{n}}\right)\right] \\ &\leq e^{-3\log k} \leq \frac{1}{k^3}.\end{aligned}$$

Let E_i be the event that server i has a load of at most $n/k + 3\sqrt{n\log k/k}$. Then we are interested in

$$\begin{aligned}\Pr\left[\bigcap_{i=1}^{k} E_i\right] &= 1 - \Pr\left[\bigcup_{i=1}^{k} \overline{E}_i\right] \\ &\geq 1 - \sum_{i=1}^{k} \Pr[\overline{E}_i], \text{ by the union bound} \\ &\geq 1 - \frac{1}{k^2}.\end{aligned}$$

We will now see a more detailed analysis of the balls into bins process and its applications.

4 Balls and Bins

In this part of the course, we will study the basic balls into bins process, and explore its applications. We will be interested mainly in the questions of the distribution of balls in bins when n balls are thrown uniformly and independently at random into n bins. We will also see that the bounds on the maximum load is related to the running time of hashing using chaining.

4.1 Warm-up: Birthday problem

Suppose that we throw m balls into n bins. By the pigeonhole principle, we know that if $m > n$, then there surely must exist a bin that has more than one ball. But what should be the value of m so that the probability of there existing a bin with more than one ball is at least $1/2$. It turns out that for this m needs to be only $\Theta(\sqrt{n})$.

To analyze this problem, notice that for the second ball to land in a bin on its own, the probability is $(1 - 1/n)$. Following this argument further, if the first i balls have all fallen in different bins, the probability of the $(i+1)^{st}$ ball landing in a bin of its own is $(1 - i/n)$. Thus, for all balls to fall into a bin of their own, the probability is given by the expression

$$\left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\ldots\left(1 - \frac{m-1}{n}\right)$$