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 $n$
1 Let $n = 2^r s + 1$, where $s$ is odd
2 Choose $a$ uniformly at random from $\{2, \dots, n-1\}$
3 if $a^s \neq 1$ (mod m) then return composite;
4 for $0 \leq i \leq s-1$ do
5     if $a^{2^i s} = -1$ (mod m) then return prime;
6 end
7 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 $1/2^4 0$ 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 (Lipton-DeMillo-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).