CLASS BPP IN PROBABILISTIC ALGORITHMS

JOSHUA KOO

Abstract. In this article, we introduce the concept of probabilistic algorithms and the amplification lemma. Furthermore, we will introduce the class BPP and prove that the algorithm $PRIMES$ is in it.

1. Introduction

For centuries, mathematicians have been trying to find an algorithm that would successfully determine the primality of a number. They continued to focus on utilizing deterministic algorithms, until in 1997 Robert M. Solovay, a professor at the University of Berkeley, and Volker Strassen, a professor at the University of Konstanz, discovered a working method by using probabilistic algorithms. So what exactly are probabilistic algorithms, and how can we use them?

A probabilistic algorithm works just like how it sounds. It's an algorithm where its behavior is determined by randomness. We can better think of it as coin flips, where the result of the flip, either heads or tails, will decide whether the algorithm will do one thing or another. So how can this ever be better than pure calculation? This is because calculating the answer may take too much time or the polynomial solution is just too complicated.

We now define probabilistic Turing machines. The general idea of probabilistic Turing machines is that they are pretty much identical to nondeterministic Turing machines but with just one difference: while in a nondeterministic Turing machine we ask does there exists a sequence of steps that leads the machine to accept, in a probabilistic Turing machine we ask how big is the fraction of selections that lead the machine to accept. We will not be going into detail about the definition, but if interested here is Michael Sipser's definition of a probabilistic Turing machine.

Definition 1.1. A probabilistic Turing machine M is a type of nondeterministic Turing machine in which each nondeterministic step is called a coin-flip step and has two legal next moves. We assign a probability to each branch b of M's computation on input w as follows. Define the probability of branch b to be

$$\Pr[b] = 2^{-k}$$

where k is the number of coin-flip steps that occur on branch b. Define the probability that M accepts w to be

$$\Pr[M \text{ accepts } w] = \sum_{b \text{ is an accepting branch}} \Pr[b] = 1$$

In this paper, we will be focusing on the class BPP, defined as follows.

Definition 1.2. BPP is the class of languages that can be determined by Turing machines that run in probabilistic polynomial time, with an error probability of $\frac{1}{3}$.

Note that the error probability of $\frac{1}{3}$ does not actually matter as long as the constant is strictly between 0 and $\frac{1}{2}$. This is called the amplification lemma, which is proved later in the paper. Finally, this paper will ultimately prove that the algorithm $PRIMES$ in the class BPP.

2. Amplification Lemma

As stated in the introduction, we will first go over the amplification lemma, which is defined as follows:

Lemma 1. *There exists a probabilistic polynomial time Turing machine M_2 with an error probability of $2^{-p(n)}$, where $p(n)$ is a polynomial and ϵ is a fixed constant strictly between 0 and $\frac{1}{2}$, such that M_2 is equivalent to a probabilistic polynomial time Turing machine M_1 that operates with error probability ϵ.*

Proof. We start by considering the behavior of M_2 on any input x. M_2 first calculates a value for k, which is determined by the desired error probability and the error probability of M_1. M_2 then runs $2k$ independent simulations of M_1 on input x. If the majority of these simulations accept, M_2 will accept; otherwise, M_2 will reject.

Next, we consider the probability that M_2 gives the wrong answer on an input x. Let S be any sequence of results that M_2 might obtain in stage 2. Let P_S be the probability that M_2 obtains S. If S has c correct results and w wrong results, where $c + w = 2k$, then if $c \leq w$ and M_2 obtains S, M_2 will output incorrectly. We call such an S a bad sequence.

We can then bound the probability of obtaining a bad sequence S by $P_S \leq \epsilon^w \cdot (1-\epsilon)^c$ This is at most $\epsilon^w \cdot (1-\epsilon)^c$ because $\epsilon^x \cdot (1-\epsilon)^{(2k-x)}$ is maximized when $x = k$. Furthermore, $\epsilon^k \cdot (1-\epsilon)^k$ is at most $\epsilon^k \cdot (1-\epsilon)^k$ because $k \leq w$ and $\epsilon < \frac{1}{2}$.

Summing P_S for all bad sequences S gives us the probability that M_2 outputs incorrectly on input x. There are at most $2^(2k)$ bad sequences because 2^{2k} is the number of all sequences. Therefore, the probability that M_2 outputs incorrectly on input x is at most $2^{2k} \cdot \epsilon^k \cdot (1-\epsilon)^k$.

We can then choose a specific value for k such that M_2's error probability is bounded by $2^{-p(n)}$ for any polynomial $p(n)$. To do this, we let $\alpha = -log_2(4\epsilon(1-\epsilon))$ and choose $k \geq \frac{t}{\alpha}$. This gives us an error probability of $2^{-p(n)}$ within polynomial time, as desired. □

Thus, the $\frac{1}{3}$ does not actually matter. As stated in the introduction, as we will ultimately be proving that the algorithm $PRIMES$ is in the class BPP, we'll now first go over a few necessary definitions and theorems.

3. *PRIMES* and Class BPP

Definition 3.1. A prime number is a positive integer greater than 1 that has no positive integer divisors other than 1 and itself.

Definition 3.2. A composite number is a positive integer greater than 1 that has at least one positive integer divisor other than 1 and itself. In other words, a composite number is a positive integer that is not a prime number.

Definition 3.3. Two integers a and b are said to be equivalent modulo p, denoted $a \equiv b \pmod{p}$, if they leave the same remainder when divided by p. In other words, if p is a divisor of $a - b$, then a and b are equivalent modulo p. For example, consider the integers 8 and 11. If $p = 3$, then $8 \equiv 11 \pmod{3}$ because both 8 and 11 leave a remainder of 2 when divided by 3. On the other hand, if $p = 4$, then $8 \equiv 0 \pmod 4$ and $11 \equiv 3 \pmod 4$, so 8 and 11 are not equivalent modulo 4.

Note that as of now a polynomial-time solution for testing whether a number is prime or composite, called the AKS primality test, does exist, but it is far too complicated which is why we are going over the probabilistic approach. If you'd like to read more about the AKS primality test, follow this link: AKS Primality Test (Linked). Exponential solutions have been around for a while, such as checking if any integer less than the number is a factor. An even more optimized solution is to check integers up to $\sqrt{n}$. The time complexity of this may seem to be $O(n)$, but note that the size of a number is exponential as it grows. With the utter importance of primality testing, the first proof of the probabilistic algorithm was discovered by the 1970s. To go over the proof, we will first define Fermat's Little Theorem, which is the very base of the probabilistic primality test.

Definition 3.4. Fermat's Little Theorem states that for any prime number p and any integer a such that $gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod p$.

We will not be going over a detailed proof of the theorem, but here is an official proof from Art of Problem Solving using induction:

Proof. The most straightforward way to prove this theorem is by applying the induction principle. We fix p as a prime number. The base case, $1^p \equiv 1 \pmod p$, is obviously true. Suppose the statement $a^p \equiv a \pmod p$ is true. Then, by the Binomial Theorem,

$$(a+1)^p = a^p + \binom{p}{1}a^{p-1} + \binom{p}{2}a^{p-2} + \cdots + \binom{p}{p-1}a + 1.$$

Note that p divides into any binomial coefficient of the form $\binom{p}{k}$ for $1 \leq k \leq p-1$. This follows by the definition of the binomial coefficient as $\binom{p}{k} = \frac{p!}{k!(p-k)!}$; since p is prime, then p divides the numerator, but not the denominator.

Taken $\mod p$, all of the middle terms disappear, and we end up with $(a+1)^p \equiv a^p + 1 \pmod p$. Since we also know that $a^p \equiv a \pmod p$, then $(a+1)^p \equiv a+1 \pmod p$, as desired. □

We will also need to know the Chinese Remainder Theorem and the Binomial Theorem, which are defined as follows.

Definition 3.5. The Chinese Remainder Theorem states that given a system of simultaneous congruences of the form:

$$\begin{aligned} x &\equiv a_1 \pmod{p_1} \\ x &\equiv a_2 \pmod{p_2} \\ &\vdots \\ x &\equiv a_n \pmod{p_n} \end{aligned}$$

where $m_1, m_2, ..., m_n$ are pairwise relatively prime (meaning they have no common factors other than 1), then there exists a unique solution $x \pmod M$ for this system, where M is the product of all the moduli $m_1, m_2, ..., m_n$.

The following is a proof of the theorem by Stanford University (note that the proofs of most of these theorems are from basic number theory, which is why we are not going through them in detail and are instead using references) for two congruences, $x \equiv a \pmod{p}$ and $x \equiv b \pmod{q}$. Note that this can trivially be expanded to more congruences.

Proof. Let $p_1 \equiv p^{-1} \pmod{q}$ and $q_1 \equiv q^{-1} \pmod{p}$. These must exist since p, q are coprime. Then we claim that if y is an integer such that

$$y \equiv aqq_1 + bpp_1 \pmod{pq}$$

then y satisfies both equations:

Modulo p, we have $y \equiv aqq_1 \equiv a \pmod{p}$ since $qq_1 \equiv 1 \pmod{p}$. Similarly, $y \equiv b \pmod{q}$. Thus, y is a solution for x. It remains to show that no other solutions exist modulo pq. If $z \equiv a \pmod{p}$ then $z - y$ is a multiple of p. If $z \equiv b \pmod{q}$ as well, then $z - y$ is also a multiple of q. Since p and q are coprime, this implies $z - y$ is a multiple of pq, hence $z \equiv y \pmod{pq}$ □

Theorem 2. *The Binomial Theorem states that for numbers x, y and non-negative integer n,*

$$(x + y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

Proof. The easiest way to go about proving this theorem is to induct on n. The base case, where $n = 0$, is obviously true. Now, suppose the theorem holds for some non-negative integer n. We will now show that it holds for $n + 1$. The LHS of the equation becomes $(x + y)^{n+1} = (x + y)(x + y)^n$, and the RHS becomes $\sum_{k=0}^{n+1} \binom{n+1}{k} x^{(n+1)-k} y^k$. Expanding the LHS, we get: $(x + y)(x + y)^n = x(x + y)^n + y(x + y)^n$. Using the induction hypothesis, we can rewrite this as:

$$x \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k + y \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

This can be simplified as:

$$\sum_{k=0}^{n} \binom{n}{k} x^{n-k+1} y^k + \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^{k+1}$$

Combining the two sums, we get:

$$\sum_{k=0}^{n+1} \left[\binom{n}{k} + \binom{n}{k-1} \right] x^{n-k+1} y^k$$

Using the identity $\binom{n}{k} + \binom{n}{k-1} = \binom{n+1}{k}$, we can rewrite this as: $\sum_{k=0}^{n+1} \binom{n+1}{k} x^{n-k+1} y^k$. This is exactly the right side of the equation, so we have shown that the theorem holds for $n + 1$. By induction, the theorem holds for all non-negative integers n, as desired. □

Using Fermat's Little Theorem to test whether a number is prime is called the Fermat Test. That is, for some integer p, it will pass the Fermat Test for value a if $a^{p-1} \equiv 1 \pmod{p}$. Thus, note that a prime number will always pass this test for all values of a such that $a \in \mathbb{Z}, a > 0$, and $a < p$. Is this enough to work as a correct primality test? Not quite. This is due to the existence of Carmichael numbers, numbers that are composite but look prime. By "look prime", this means that it passes the Fermat Test for all values of a. Though, note that Carmichael numbers are very rare, with the first few of them being very large (and growing fast):

$$561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, \cdots$$

Thus, we will first introduce the algorithm $PSEUDOPRIME$, which is a probabilistic algorithm that properly classifies between prime and composite with the exception of Carmichael numbers (note that a pseudoprime number is a number that passes all Fermat Tests). For a number p, the time complexity of checking whether it passes through all possible Fermat Tests would clearly be exponential. Therefore, we will use the fact that, for some number p, if it is not pseudoprime, it will always fail on at least half of its Fermat tests. Here is a basic rundown of how the algorithm / Turing machine works: For input p and random numbers $a_1, \ldots, a_k$, where $a_i > 0, a_i \in \mathbb{Z}$, and $a_i < p$, compute the values of $a_i^{p-1} \pmod{p}$. If every single one of the values is 1, then accept. If not, then reject.

And so note that in the case where p is not a pseudoprime, the probability that it will pass all k a_is is 2^{-k}. Now, to change this into a probabilistic algorithm that takes care of the Carmichael numbers, we will use the following principle: For any prime number p, $\sqrt{1}$ $\pmod{p}$ is always equivalent to 1 or -1, while for a composite number c, $\sqrt{1} \pmod{c}$ always has at least four solutions. The proof of this is quite trivial by using the Chinese Remainder Theorem. Using this, we define the Turing machine $PRIMES$ as follows (from Michael Sipser but edited):

```
PRIMES = "On input p:
(1) If $p = 2$, accept. Otherwise, if $p$ is even, then reject.
(2) Randomly choose $a_1, \ldots a_k$ such that $a_i > 0, a_i \in \mathbb{Z}$, and $a_i < p$.
(3) For each $i$ from 1 to $k$:
(4)         Compute $a_i^{p-1} \pmod{p}$. Reject if not equal to 1.
(5)         Let $p - 1 = s \cdot 2^l$, where $s$ is odd.
(6)         Compute the sequence $a_i^{s \cdot 2^0}, \ldots a_i^{s \cdot 2^l} \pmod{p}$.
(7)         If at least one element of the sequence is not equal to 1, find the last such
            element and reject if the element is not equal to -1.
(8) Accept.
```

Note that k sets the maximum error probability to 2^{-k}. From here, we will prove the following two lemmas to show that the algorithm works. We will also only be looking at odd numbers from now on, as even numbers will clearly be accounted for in the very first step of the algorithm.

Lemma 3. *If p is an odd prime number, Pr[PRIMES accepts p] = 1.*

Proof. Looking at the algorithm, all we have to show is that p will never be rejected if it is prime. The only two steps that have the possibility of rejection are steps 4 and 7. First, if we look at step 4, if p, a prime number, were to be rejected then for some a_i, $a_i^{p-1} \pmod{p}$ would have to not equal 1. Referring back to Fermat's Little Theorem, this would mean that p is composite, which is a contradiction. Thus, p cannot be rejected at step 4. Second, if p

were to be rejected at step 7 then there would have to exist an integer x such that $x > 0$, and $x < p$, where $x^2 \equiv 1 \pmod{p}$ and $x \not\equiv \pm 1 \pmod{p}$. With some simple algebra:

$$b^2 - 1 = (b-1)(b+1) \equiv 0 \pmod{p}$$

This means that $(b-1)(b+1)$ is a multiple of p. Since $x \not\equiv \pm 1 \pmod{p}$, $0 < b-1, b+1 < p$. This is a contradiction since p must be composite if some multiple of it can be the product of numbers smaller p. Thus, a prime p cannot be rejected at step 7, as desired. ☐

Lemma 4. *If p is an odd composite number, Pr[PRIMES accepts p] $\leq 2^{-k}$.*

Proof. This proof has two parts, and we will be going more over the second one. To go into more detail about the first part, read page 402 in Michael Sipser's Introduction to the Theory of Computation. To show that the lemma is true, we must prove that for every randomly selected a, such that p is not rejected, there must exist a unique randomly selected a_{reject} such that p is rejected. There are two cases to look at. The first one is when p is the product of two relatively prime numbers, while the second one is when p is the power of a prime. Though we will not be going over the first case, the key idea is that we can use the Chinese Remainder Theorem to find an a such that p gets rejected. For the second case, let $p = q^e$, such that q is prime and e is greater than 1. To find a possible a such that p gets rejected, let us set $n = 1 + q^{e-1}$. Using the Binomial Theorem, if we expand n^p we get:

$$1 + p \cdot q^{e-1} + \ldots \equiv 1 \pmod{p}$$

This shows us that n is a possible a_{reject} that causes p to be rejected in step 4 since if it were not, then by Fermat's Little Theorem, $n^{p-1} \equiv 1 \pmod{()p}$ leads to $n^p \equiv n \not\equiv 1 \pmod{p}$, which is a contradiction. Thus, let m be a possible a such that p does not get rejected. Then, we claim that $nm \pmod{p}$ is a unique a_{reject} such that p gets rejected. To do so, let m_1 and m_2 be distinct possible as. If $nm_1 \pmod{p} = nm_2 \pmod{p}$, then

$$m_1 = m_1 \cdot n \cdot n^{p-1} \pmod{p} = m_2 \cdot n \cdot n^{p-1} \pmod{p} = m_2,$$

which is a contradiction since we assumed m_1 and m_2 to be distinct. Thus, there will always be at least as many possible a_{reject}s as there are as, as desired. ☐

Combining the two lemmas is clearly enough to show that $PRIMES \in$BPP.

4. Additional Resources

Note that there are now many different kinds of proofs for showing $PRIMES \in$BPP. For instance, here is a much more advanced proof from the book Computational Complexity: A Modern Approach written by Sanjeev Arora and Boaz Barak (with a few grammar edits):

Proof. For every number N, and $A \in [N-1]$, define:

$$QR_N(A) = \begin{cases} 0 & gcd(A,N) \neq 1 \\ +1 & A \text{ is a } \textit{quadratic residue} \text{ modulo } N \\ -1 & \text{otherwise} \end{cases}$$

We use the following facts that can be proven using elementary number theory:

(1) For every odd prime N and $A \in [N-1], QR_N(A) = A^{(N-1)/2} \pmod{N}$.

(2) For every odd N, A, define the *Jacobi symbol* $(\frac{N}{A})$ as $\sum_{i=1}^{k} QR_{P_i}(A)$ where $P_1 \ldots, P_k$ are all the (not necessarily distinct) prime factors of N (i.e., $N = \sum_{i=1}^{k} P_i$). Then, the Jacobi symbol is computable in time $O(\log A \cdot \log N)$.

(3) For every odd composite N, $|\{A \in [N-1] : gcd(N, A) = 1 \text{ and } (\frac{N}{A}) = A^{(N-1)/2}\}| \leq \frac{1}{2}|\{A \in [N-1] : gcd(N, A) = 1\}|$

Together these facts imply a simple algorithm for testing the primality of N (which we can assume without loss of generality is odd): choose a random $1 \leq A < N$, if $gcd(N, A) > 1$ or $(\frac{N}{A}) \neq A^{(N-1)/2} \pmod N$ then output "composite", otherwise output "prime". This algorithm will always output "prime" if N is prime, but if N is composite it will output "composite" with a probability of at least $\frac{1}{2}$ (Of course, this probability can be amplified by repeating the test a constant number of times). $\square$

In terms of probabilistic algorithms and classes, note that there are a lot more findings in the field than just the algorithm $PRIMES$ and class BPP. For instance, you could explore the classes $RL, RP,$ and ZPP, as well as the algorithm EQ_{ROBP}.

References

[1] Arora, Sanjeev, and Boaz Barak. Computational Complexity: A Modern Approach. Cambridge, UK: Cambridge University Press, 2009.

[2] Sipser, Michael. Introduction to the Theory of Computation. 3rd ed. Boston, MA: Thomson Course Technology, 2013.

[3] Ta-Shma, Amnon. "Lecture 7: AKS." Tel Aviv University, Tel Aviv, Israel, 2019.

[4] "Chinese Remainder Theorem." Stanford University, crypto.stanford.edu/pbc/notes/numbertheory/crt.html.

[5] "Fermat's Little Theorem." AoPS, artofproblemsolving.com/wiki/index.php/Fermat%27s_Little_Theorem.