PRIME GENERATION DRAFT

JASON MEDCOFF

Abstract. The generation of large primes has been of great importance to modern cryptography. In this paper we review a widely known and deceptively efficient method for generating prime numbers: the Sieve of Eratosthenes. We build up to the algorithm with a review of some basic properties of primes and composites, and develop some results that make the sieve more efficient in practice. Emphasis is placed on the mathematics behind the algorithm, with the intention of educating the student of computer science student about the significance of its time complexity.

1. Introduction

Today, the generation of prime numbers is a task generally associated with hashing algorithms, public key cryptography, and searching for factors of large numbers. Most modern algorithms designed to generate primes are derived from a process devised by Eratosthenes of Cyrene in Ancient Greece. In this paper, we explore the motivating ideas about prime numbers that lead to the intuitive derivation of the Sieve of Eratosthenes, beginning with some review of elementary number theory concerning primes and composites.

The algorithm is decidedly easy to understand, and similarly easy to implement in one's favorite programming language. In fact, the careful eye may notice potential areas of improvement in the interest of time and space. It is perhaps not surprising, then, that most modern prime generation algorithms are derived from the fundamental prime sieve due to Eratosthenes.

We begin by reviewing the essential ideas of primes and composites. The Fundamental Theorem of Arithmetic describes that all numbers have a factorization of primes only, unique up to the ordering of the factors. In addition, it can be shown that at least one of a number n's factors must lie in the range $[2, \sqrt{n}]$. Combining these results, we can show that for any number n, a prime factor in this range implies that n is composite. From this, it is easily seen that to generate the primes up to n, we need only cross out the multiples of numbers less than $\sqrt{n}$.

Following this important result, we have all we need to implement a simple version of the Sieve of Eratosthenes, at six lines of code. It is shown by proof of correctness that the algorithm produces all primes up to n. By applying Mertens's Second Theorem, we can derive the sieve's asymptotic running time. Next, a time test is performed, demonstrating the running time in practice, and we see that the asymptotic running time bounds the experimental trials.

Next, we observe areas of potential improvement of the algorithm, verify the ideas for improvement, and modify the algorithm.

2. Some Basics of Primes and Composites

Understanding of the prime number sieve depends upon an understanding of some important properties of prime numbers. The ideas of checking up to the square root of a number for factors and unique factorizations of composites are developed in this section. We begin with familiar

Date: November 15, 2017.

definitions, and then move into groundwork that will allow the correctness of the sieve to be shown.

Recall the following definition.

Definition 2.1. A natural number $p > 1$ is said to be prime if its only positive divisors are 1 and p. If p is not prime, it is said to be composite.

One of the most important theorems with regards to the primes and integers follows.

Theorem 1 (Fundamental Theorem of Arithmetic)**.** *If $n > 1$ is a natural number, it can be written as a product of powers of primes unique up to the order of the factors.*

Proof. Suppose there exists an integer $n > 1$ such that n cannot be written as a product of primes. By the well-ordering principle, we know that there must be a smallest n satisfying this. Write n as a product, $n = xy$, with $x > 1$ and $y < n$. Since n is the smallest positive integer that cannot be written as a product of primes, and x and y must be less than n, we can write x and y as products of prime factors. Then n can be written as a product of primes, which contradicts the above assumption. So, every integer $n > 1$ can be written as a product of primes.

Next, suppose an integer $n > 1$ has two prime factorizations, such that

$$n = j_1 \cdots j_s = k_1 \cdots k_t$$

where all j_i and k_i are prime. By the well-ordering principle, we know that there must be a smallest n satisfying this as well. Then $j_1 | n = k_1 \cdots k_t$ and since j_1 is prime, it must divide k_i for some i. But we know that k_i is prime too, so it must be that $j_1 = k_i$. Then we relabel k_i as k_1 for simplicity, and we now cancel j_1 to obtain

$$j_2 \cdots j_s = k_2 \cdots k_t.$$

Since n is the smallest positive integer that has a non-unique prime factorization, and $j_2 \cdots j_s = k_2 \cdots k_t < n$ is a product of primes, we arrive at a contradiction. Then there cannot exist an integer $n > 1$ with two different factorizations. □

An immediate question resulting from this theorem is how we can best obtain the prime factors of arbitrary numbers. This question is still open in number theory and theoretical computer science; no classical algorithm is known that can factor a number in polynomial time on the number of digits. The difficulty of factoring very large numbers (on the order of thousands of digits) led to the creation of the RSA cryptosystem.

Given some number n, we may want to find out whether n is prime or composite. Naively, we can simply enumerate the natural numbers starting at 2, and check for divisibility, stopping when we reach n. However, in the interest of saving time, we can apply a result that will cut down the time spent checking divisors.

Lemma 1. *If n is composite, at least one of its factors lies in $[2, \sqrt{n}]$.*

Proof. Let $m = \sqrt{n}$. Then we can write n as

$$n = m \cdot m = a \cdot b.$$

Consider a. One of the following must be true: $a > m$, $a < m$, or $a = m$. In the first case, b must be less than m; in the second case, b will be greater than m. If $a = m$, clearly $b = m$. Therefore, if we search the integers up to m, we will find at least one factor of n. □

We can combine this result with the fundamental theorem of arithmetic, and easily show the following.

Corollary 1. *Let $n > 1$ be an integer. If n does not have a prime factor in the range $[2, \sqrt{n}]$, it is prime.*

This is a combination of the contrapositive of lemma 1 with theorem 1. So now, we have a way to check for primality of small numbers. For some n, we need only check divisibility by primes in $[2, \sqrt{n}]$. This algorithm, called trial division, is shown to be very slow on larger numbers.

3. Generating Primes with the Sieve of Eratosthenes

The Sieve of Eratosthenes is a process for generating primes up to some positive integer n. Roughly speaking, the process is to cross off the multiples of the numbers up to the square root of n. Those numbers that remain are the primes up to n.

Consider the following algorithm.

Figure 1. Sieve of Eratosthenes

```
def Eratosthenes(n):
    composites = set()
    for i in range(2, n+1):
        if i not in composites:
            yield i
            composites.update(range(i**2, n+1, i))
```

The python code above utilizes the "set" data structure; this is suitable for two reasons. First, since we are only considering one list of numbers 2 to n, every element we are considering is unique, suggesting use of a set. Second, the code performs many membership checks. For lists, checking membership in a list of length m is $O(m)$ in the average case, while for sets this check is $O(1)$.

Theorem 2. *The algorithm* `Eratosthenes` *produces the prime numbers in the range* $2, n$.

Proof. The algorithm's correctness is easy to see based on the use of `composites`. The loop invariant here is that the set `composites` contains all composite numbers between 2 and i; this holds for every iteration, where we yield i if it is not composite, then add multiples of i from i^2 to n to the `composites` set. At the end of the last loop, every prime has been yielded in the interval $[2, n]$. Note that in addition, the algorithm is totally correct; it terminates after a finite number of steps because its loop bounds are finite. □

Informally speaking, we are using the `composites` set to "mark" composite numbers in the range, then returning along the way those numbers which are not marked. We know that we can start at i^2, because all of the multiples of i up to i^2 have already been marked off.

Lemma 2. *For every i as in the algorithm above, every multiple of i up to i^2 has already been collected into* `composites`.

Proof. We will prove this by induction. Consider first the case of $i = 2$; add it to the set `composites`. Every multiple of 2 less than $2^2 = 4$ has already been added to the set; 2 is the only such number. Then we add every multiple of 2 greater than 4 to the set, and the set contains all multiples of 2. Now suppose the conjecture holds for some $i = k$. Then for $i = k + 1$, we consider the multiples $a(k + 1)$ for some positive integer $a < k + 1$. For $a = 2$, we have $2k + 2$, which is a multiple of 2. For $a = 3$, we have a multiple of 3, and so on. For $a = k$, we have a multiple of k, which is already taken care of by the induction hypothesis. So every $a(k+1)$ gives a multiple of a. Therefore, every multiple of i up to i^2 has been collected into `composites`. □

This result saves us from a lot of unnecessary set membership checking. In fact, we can recognize that this lemma follows from the earlier corollary 1. It is clear that if we start considering multiples of i at i^2, when $i = \sqrt{n}$ we have reached the upper bound of sieving and the remaining numbers are the primes. We will now give the running time a rigorous treatment.

Theorem 3. *The asymptotic running time of* `Eratosthenes` *is $O(n \log\log n)$, where n is the upper limit on the sieving range.*

Proof. The noteworthy work done by the algorithm is identifying the composites; we will count these operations. For each of the n iterations, we are trimming the multiples of the primes up to n from the list. In particular, for $i = 2$, we perform roughly $n/2$ trims, for $i = 3$ we perform $n/3$, etc. We can write this formulation as

$$\sum_{p \leq n} \frac{n}{p} = n \sum_{p \leq n} \frac{1}{p}.$$

Mertens's second theorem shows that

$$\lim_{n \to \infty} \left[\sum_{p \leq n} \frac{1}{p} - \log\log n - M \right] = 0$$

where M is the Mertens constant, and rearranging terms, we observe

$$\lim_{n \to \infty} \sum_{p \leq n} \frac{1}{p} = \lim_{n \to \infty} \left[\log\log n - M \right].$$

We multiply by n to obtain

$$\lim_{n \to \infty} n \sum_{p \leq n} \frac{1}{p} = \lim_{n \to \infty} n \left[\log\log n - M \right]$$

and so the asymptotic running time as written above is

$$n \log\log n - nM = O(n \log\log n).$$

□

We can concretely demonstrate the running time by writing a test. We run the algorithm some number of times, and calculate the average time taken. After performing this test for several different input sizes, we can construct a plot of running time against input size. Figure 2 displays the running time experiment with an average of 50 trials, on sieve sizes from 1000 to 100000. The function $f(x)$ is the result of theorem 3, multiplied by a constant. Namely, we use

$$f(x) = \frac{x \log\log x}{2443470.5}.$$

It is evident from figure 2 that the function derived from the running time analysis bounds the experimental running time.

FIGURE 2. Asymptotic Running Time of `Eratosthenes`