01204211 Discrete Mathematics
Lecture 16: Primality testing (1)

Jittat Fakcharoenphol

October 5, 2015

Integers

This is the second major area that we shall study in this course: integers and their properties. This area is called **number theory.** We will see many properties of integers and their applications that include data encoding techniques and data encryption.
Most techniques depend on having large primes. So, we start this area by focus on the primality testing algorithm. This is also the first algorithm that we considered in this class as well.

Brute-force division

FUNCTION CheckPrime2(n)
1. $k \leftarrow 2$
2. WHILE $k \leq \sqrt{n}$ DO
3. IF k divides n THEN
4. RETURN **false**
5. ENDIF
6. $k \leftarrow k + 1$
7. ENDWHILE
8. RETURN **true**

The WHILE loop in CheckPrime2 runs for at most $\sqrt{n}$ times. This improves over the original algorithm that uses roughly n rounds. While this is a big improvement, it is usually not good enough when we consider a typical usage of the algorithm that we need, where we need to check if a 1000-digit number is a prime.

Running time analysis: the O-notation

- We shall study the running time of various algorithms in this section. Since we will not be very precise, keeping tracks of all details, we will use the O-notation when we talk about the running time.
- We will informally use the notation. When we say that function $f(n)$ is $O(g(n))$, we means that the growth of $f(n)$ is at most that of $g(n)$. This means that the largest terms in $f(n)$ is not larger than that of $g(n)$.
- **Examples:**
  - $n^2 + 100n = O(n^2)$
  - $\sqrt{n} + n = O(n)$
  - $2^n + 10000 + 100000n^{10} = O(2^n)$
  - $5 \cdot n^3 + n^2 = O(n^3)$
  - $10n^2 \times 7n + 12n^3 \times 75n^2 = O(n^3) + O(n^5) = O(n^5)$
- Note that it is also true that $n^2 = O(n^3)$.

Polynomial running times

Let's discuss CheckPrime2's running time.
We usually think about the running time as a function on the size of the input. When we want to sort n numbers, the size of the input is n. However, when dealing with big numbers (i.e., those much larger than directly manipulatable in the CPU), you cannot manipulate them in constant time. In this case, we usually count the number of bits as the size of the input.
Since n is the value of the input, to keep this integer in computer memory, you need at least $\log_2 n$ bits[1]. This will be the size of the input that we shall consider.

Let $m = O \log n$ be the number of bits of n. The algorithm that runs in time $O(\sqrt{n})$, actually runs in time $O(\sqrt{2^m}) = O(2^{m/2})$, an exponential running time. This means that the algorithm does not scale very well, as the size of the input increases.
In this case, we want a more efficient algorithm, i.e., it runs in time in the polynomial of m.

[1] We shall use logarithm base 2 in this part of the course.

This lecture covers

- basic definitions related to division and modulo operation
- prime factorization
- a fundamental theorem stating a fact related to prime numbers called the Fermat's Little Theorem

Definitions: divisibilty

Let's start with basic definitions.

- We say that "a divides b", "b is divisible by a", or "b is a multiple of a" if there exists an integer k such that $b = ka$. In this case, we write
$$a|b.$$
- If it's not the case, we write $a \not| b$.
- When a does not divides b, there is a remainder. We say that r is a remainder of dividing b by a if $0 \leq r < a$ and there exists integer k such that $b = ka + r$. We also write
$$r = a \mod b.$$

  - $10 \mod 3 = 1$, $10 \mod 2 = 0$, $10 \mod 15 = 10$
  - $-10 \mod 3 = 2$, $-10 \mod 15 = 5$

The modulo operation

For integers a and b and positive integer q, we have

- $(a+b) \mod q = ((a \mod q) + (b \mod q)) \mod q$
- $(a-b) \mod q = ((a \mod q) - (b \mod q)) \mod q$
- $(ab) \mod q = ((a \mod q) \times (b \mod q)) \mod q$

Examples:

- $(14+7) \mod 5 = ((14 \mod 5) + (7 \mod 5)) \mod 5 = (4+2) \mod 5 = 1$
- $(14 \cdot 7 \cdot 13 \cdot 19) \mod 5 = ((14 \mod 5) \cdot (7 \mod 5) \cdot (13 \mod 5) \cdot (19 \mod 5)) \mod 5 = (4 \cdot 2 \cdot 3 \cdot 4) \mod 5 = 96 \mod 5 = 1$

These facts are really helpful when you try to compute $x \mod y$ when x is very large compared to y and x is a result of many operations of small numbers. In this case, we can keep moduloing intermediate results to keep them under y.

You will prove these properties in your homework.

Definitions: primes

An integer p is a **prime** if $p > 1$ and p has only 4 factors: $1, -1, p$, and $-p$. If a number larger than 1 is not a prime, we say that it is a **composite**.

Prime numbers are very fascinating. There are many facts that have proved about them. E.g., we looked at Euclid's proof that there are infinitely many primes. Here's another one by Euclid:

Theorem: For any positive integer n, there are n consecutive composites.
Proof: Let $m = n + 1$. Consider

$$(m! + 2), (m! + 3), \ldots, (m! + m).$$

Note that these $m - 1 = n$ numbers are composite because for any $1 \leq i \leq m$, $i|m!$, $i|i$, and thus, $i|(m! + i)$. ■

Prime factorization

It is known since the Greeks that if you have a composite n, you can factor it as a product of prime numbers. For example, you can write

$$140 = 2 \times 2 \times 5 \times 7.$$

Theorem: A prime factorization of a positive number larger than 1 is unique.

Lemma: If p is a prime for any integer $1 \leq k < p$, we have that

$$p \Big| \binom{p}{k}.$$

Fermat's Little Theorem

Theorem: If p is a prime and for any integer a, we have that $p|a^p - a$.