Advanced Algorithms

Signal Data Science

Here's a collection of problems about advanced algorithms. These are designed to reinforce your R programming skills while teaching you useful and applicable material.

Fast primality testing

Prime numbers are mathematically important because they form the "backbone" of the natural numbers, and hence have many interesting and mysterious properties which, if understood, give us deep insights into questions involving integers. Number theory was considered in the past to be the purest of the fields of mathematics, with no possible application to practical problems; however, in the modern era, primality testing (checking if a number is prime or composite) is of crucial importance because of its role in cryptography (which keeps your bank account, medical information, etc. safe).

Primality testing is in fact a classic algorithmic task, stretching all the way back to 200 BC with the Sieve of Erastosthenes developed by Erastosthenes of Cyrene. We will first implement the Sieve and then work toward writing an implementation of the Miller–Rabin primality test, a modern test for primality known to be very fast in practice for reasonably small numbers.

The Miller–Rabin primality test

In the Miller–Rabin primailty test, we test the primality of a number $n > 2$ as follows: Since n is odd, $n - 1$ must be even, so we can write $n - 1 = 2^s \cdot d$, where d is odd. (For example, if $n = 13$, then $n - 1 = 12 = 2^2 \cdot 3$ with $s = 2$ and $d = 3$.) The Miller–Rabin primality test is based on the observation that if we can find a number a such that $a^d \not\equiv 1 (\text{mod } n)$ *and* $a^{2^r d} \not\equiv -1 (\text{mod } n)$ for all integers r in the range $0 \leq r \leq s - 1$, then n is not prime. Otherwise, n is likely to be prime.

Note that the Miller–Rabin primality test, as formulated here for a specific value of a, is *probabilistic* rather than *deterministic* – it cannot definitively establish that n is prime. It can be made detetermınistic by checking all $a \leq 2(\ln n)^2$. Better yet, when n is sufficiently small, it has been found that we only need to consider a couple different values of a; for example, for $n < 4,759,123,141$, we only have to check $a \in \{2, 7, 61\}$.

We have one more utility function to write:

- Write a function `decompose_even(n)` which takes as input an *even* integer `n` and returns a vector of two integers `c(s, d)` such that `n` is equal to `2^s * d` and `d` is odd.

With `decompose()`, `decompose_even()`, and `pow3()` (your fastest modular exponentiation implementation from *Basic Algorithms*, we are now ready to implement the entire primality test.

- Following the above description, implement the deterministic Miller–Rabin test as `miller_rabin(n)` for $n < 4,759,123,141$, returning `TRUE` for a prime number and `FALSE` otherwise. Keep the following points in mind:
  - Checking if $x \equiv -1 (\text{mod } n)$ is equivalent to checking if $x \equiv n - 1 (\text{mod } n - 1)$.
  - The values of a used are themselves prime but will not be evaluated as such by the algorithm, so they must be handled specially if passed in as input.
  - Make sure that the special case of 1 (which is a composite number) is handled properly.
  - You can verify that your implementation works correctly by combining it with `Filter()` to find the prime numbers from 1 to 100 and checking the output against that of the Sieve of Erastosthenes.
- Write a function `simple_check(n)` that checks if `n` is a prime by checking if `n` is divisible by any integers from 2 up to `floor(sqrt(b))`. Verify that `miller_rabin()` and `simple_check()` produce the same output for the first 10,000 integers.

A small primality problem

We can apply the Miller–Rabin primality test to solve a simple problem in computational number theory.

- Write a function `variations(n)` which takes in an integer `n` and returns a vector containing every number which can be obtained by changing a single digit of `n`.
- With `variations()` and the Miller–Rabin primality test, find a counterexample to the following statement: By changing at most a single digit of any positive integer, we can obtain a prime number.