Problem 216

1 Problem

How many numbers $t(n) = 2n^2 - 1$ are prime for $1 < n \leq 50,000,000$?

2 Solutions

Throughout this section, $L = 50,000,000$.

2.1 Brute Force

A simple brute force approach would be to check which numbers $2n^2 - 1$ are prime using a suitable primality test (such as Miller-Rabin or Lucas Test). It would be something like:
`For every` $1 < n \leq L$:

- `If` $2n^2 - 1$ `is prime, then add 1 to the result.`

Unfortunately this is too slow to beat the one-minute rule in most languages, so we try to find faster solutions.

2.2 Sieve

We implement a factorization sieve that computes prime factors of numbers of the form $2n^2 - 1$. The idea is that if a prime p divides $2n^2 - 1$, then it also divides $2m^2 - 1$, where $m = kp + n$ or $m = kp - n$ for some k. This can be stated more compactly as $m \equiv \pm n \pmod{p}$.

So begin by building a list T, of size L, initially containing the values $(t(1), t(2), ..., t(L))$. Then, iterate for all $1 < n \leq L$:

- `If the value of` $T(n)$ `is 1, skip to next` n.

- `Otherwise, the value at` $T(n)$ `is prime. Let` $p = T(n)$.

  `If` $p = 2n^2 - 1$ `(i.e.` $T(n)$ `is unchanged), then` $t(n)$ `is prime, so increment the result by 1.`

  `Next, for all` $m = n, p \pm n, 2p \pm n, 3p \pm n, ...,$ `divide` $T(m)$ `by` p `as much as possible.`

One might ask, is the value at $T(n)$ really prime when we reach it?

Let p be a prime dividing some $t(n) = 2n^2 - 1$, and let $t(n_p) = 2n_p^2 - 1$ be the *smallest* such value. What can we say about n_p? First, $n_p < p$, because if $n_p > p$, then p divides the smaller number $t(n_p - p)$ (note that $n_p \neq p$, otherwise p cannot divide $2n_p^2 - 1$). Further, $n_p < p/2$, because if $n_p > p/2$, then p divides the smaller number $t(p - n_p)$ (note also that $n_p \neq p/2$ because p is an odd prime).

So at the nth iteration, every factor $p \leq 2n$ of $t(n)$ has already appeared in a previous iteration (because p would have already appeared at iteration n_p, and $n_p < p/2 \leq n$), so no prime factors less than or equal to $2n$ would be present in $T(n)$ at this time. Now, if the value remaining at $T(n)$ is not a prime, then it is divisible by at least 2 primes $p, q > 2n$, but $pq > 4n^2 > 2n^2 - 1 = t(n)$, which is impossible, so the value at $T(n)$ is indeed prime when we reach it.

2.3 Modular Square Roots

We can use modular root finding to calculate the smallest index n_p for a given p.

Let p be an odd prime dividing $2n^2 - 1$. Then:

$$\begin{aligned} 2n^2 - 1 &\equiv 0 \pmod{p} \\ 2n^2 &\equiv 1 \pmod{p} \\ n^2 &\equiv 2^{-1} \pmod{p} \end{aligned}$$

The inverse of $2 \mod p$ can be found with Fermat's little theorem as $2^{p-2} \mod p$, or more simply with the extended Euclidean algorithm as $(p+1)/2$. Hence we have:

$$n^2 \equiv \frac{p+1}{2} \pmod{p} \tag{1}$$

Thus, n_p is the smallest possible n satisfying (1), i.e., the smallest *square root* of $(p+1)/2 \mod p$.

Now, we can brute force (1) until we find the smallest solution n_p, or we can use the Tonelli-Shanks Algorithm which computes *square roots modulo* p much quicker (a description is given in (3.1)).

When is (1) solvable? The Legendre symbol tells us that 1 is solvable if and only if:

$$((p+1)/2)^{(p-1)/2} \equiv 1 \pmod{p} \tag{2}$$

Now we can check this equation for our p quickly using modular exponentiation, or we can go further. Note that $(p+1)/2 \equiv 2^{-1} \mod p$, so (2) is equivalent to:

$$2^{-(p-1)/2} \equiv 1 \pmod{p}$$

Inverting both sides (alternatively, multiplying $2^{(p-1)/2}$ to both sides),

$$2^{(p-1)/2} \equiv 1 \pmod{p}$$

The Legendre symbol also tells us that this is true if and only if $p \equiv \pm 1 \pmod{8}$, so we actually only need to check primes $p = 8k \pm 1$.

So we find all such primes $p \leq \sqrt{2L^2 - 1}$ using a normal sieve (or a list of primes if you have one), compute the *modular square root* n_p for a given p, and then continue with the sieve. It now looks like this:

- `Mark all` $t(n)$ `initially as prime, for` $1 < n \leq L$.
- `For all primes` $p \leq \sqrt{2L^2 - 1}$ `and` $p \equiv \pm 1 \pmod{8}$:

  `Compute` $n_p =$ `(square root of` $(p+1)/2 \mod p$`) using the Tonelli-Shanks algorithm.`

  `If` $n_p > p/2$, `set` $n_p := p - n_p$ `Now,` n_p `is the smallest square root of` $(p+1)/2 \mod p$.

  `For all` m `such that` $m \equiv \pm n_p \pmod{p}$ `(but also` $m \neq n_p$ `if` $p = 2n_p^2 - 1$`), mark` $t(m)$ `as composite.`
- `Then, we count all values` $t(n)$ `which are still marked as prime (these values are indeed prime), and we are done.`

For the original sieve method, you need an array of L 64-bit integers, however for this algorithm only L boolean values are needed, and that can be reduced to L bits of you use a bit sieve, so the memory requirements for this algorithm is significantly less.

Also, if you used a normal sieve to compute the primes $p = 8k \pm 1 \ \leq \sqrt{2L^2 - 1}$ instead of a prime number list, an unoptimized implementation would require about $1.4L$ boolean values, so you would need about $2.4L$ bits in total, which is still very low compared to $64L$ bits. A simple optimization of that sieve would be to store boolean values only for the numbers $p = 8k \pm 1$, and this reduces the memory usage of the sieve to just $0.35L$ bits and the overall memory to about $1.35L$ bits.

3 Appendix

3.1 Tonelli-Shanks Algorithm

This is a description of Tonelli-Shanks Algorithm taken from Wikipedia. It computes square roots modulo a prime number, i.e., it solves the congruence $x^2 \equiv n \pmod{p}$.

Note that ":=" means assignment and ":≡" means assignment modulo p.

Input: n, p where p is an odd prime and n has a square root modulo p.

Output: R satisfying $R^2 \equiv n \pmod{p}$.

- `Compute` Q, S `from` $p - 1 = Q2^S$ `with` Q `odd.`
- `Find a number` z `that has no square root modulo` p`.`
- `Set` $c :\equiv z^Q, R :\equiv n^{(Q+1)/2}, t :\equiv n^Q \pmod{p}$ `and` $M := S$`.`
- `Repeat until` $t \equiv 1 \pmod{p}$`:`

  `Find the lowest` $0 < i < M$ `such that` $t^{2^i} \equiv 1 \pmod{p}$`, e.g. via repeated squaring.`

  `Let` $b :\equiv c^{2^{M-i-1}} \pmod{p}$`.`

  `Set` $R :\equiv Rb, t :\equiv tb^2, c :\equiv b^2 \pmod{p}$ `and` $M := i$`.`
- `Return the value` R`. The other solution is simply` $p - R$`.`

Note that the smallest solution is the smaller number between R and $p - R$. Also, the algorithm doesn't work when p is composite; computing square roots modulo composite numbers is a computational problem equivalent to integer factorization.

In step 2, how do we find a number z that has no square root modulo p, i.e. is a *quadratic nonresidue* modulo p? Wikipedia says that there is no deterministic algorithm that runs in polynomial time for finding such a number. Luckily, with the Legendre symbol, we know that half of the numbers $1, 2, .., p-1$ are quadratic residues and the other half are quadratic nonresidues, so *on average* we only have to check two random numbers $1 \leq z < p$ to find a valid z.

A simple way to determine whether a number z is a quadratic residue or not is to use Euler's criterion. It simply says that a number z, where p does not divide z, is a quadratic residue modulo p if and only if

$$z^{(p-1)/2} \equiv 1 \pmod{p}$$

This is easily computed using modular exponentiation.

Lastly, if $p \equiv 3 \pmod{4}$, then you do not have to use Tonelli-Shanks to find the answer. You can compute the solutions more simply as

$$R \equiv \pm n^{(p+1)/4} \pmod{p}$$

To incorporate this into the pseudocode, note that $p \equiv 3 \pmod{4}$ if and only if $S = 1$ in the first step, so if you find $S = 1$ after computing Q and S, then stop there and return the value $R = n^{(p+1)/4} \pmod{p}$ using modular exponentiation.