The generating function of primes mod 2

June 16, 2010

1 Introduction

Fix a polynomial $f(x) \in \mathbb{F}_2[x]$ of degree d. We wish to compute

$$\sum_{p \leq N} x^p \pmod{2, f(x)}$$

quickly. We will explain how to do this (actually, a related problem – we confine to short intervals) using at most $N^{1/2-\varepsilon}$ or so bit operations.

Using the obvious generalization of our work for computing the parity of $\pi(x)$, it suffices to show that we can compute

$$\sum_{\substack{2 \leq n \leq N \\ n \text{ not a } \square}} (\tau(n)/2) x^n \pmod{2, f(x)}$$

in time $N^{1/2-\delta}$, for some $\delta > 0$. The larger $\delta > 0$ is in our algorithm, the larger $\varepsilon > 0$ will be.

Now what's the idea? Well, let's first think about the idea for the parity of $\pi(x)$: Recall that what we essentially had to do was compute

$$\sum_{n \leq \sqrt{N}} \lfloor N/n \rfloor \tag{1}$$

substantially faster than the trivial algorithm, which takes time $N^{1/2+o(1)}$. The key observation was that we can compute the sum of fractional parts

$$\sum_{x \leq n \leq x+q} \{N/x\},$$

to within an error of $1/2$, for certain x and q, very quickly – essentially time $O(1)$, which is faster by a factor of q over the trivial algorithm. What allows us to do this is the fact that N/n can be "linearized", in the sense that

$$|\{N/n\} - \{N/x - at/q\}| \ < \ 1/2q,$$

where $n = x + t$, and where $N/x^2 \sim a/q$.

So really all we need to compute is

$$\sum_{x \le n \le x+q} \{N/x - at/q\}.$$

(And, we need to fuss over certain exceptional cases.) And this is easily handled.

The analogue of the sum (1) in the polynomial context is

$$\sum_{d \le \sqrt{N}} \sum_{m \le N/d} x^{dm} \pmod{2, f(x)}. \tag{2}$$

Now of course we can use the geometric series formula (or other identities) to compute this in time $N^{1/2+o(1)}$, so as in non-polynomial case that is the bound to beat.

Of course evaluating *that* sum quickly would be overkill as far as locating primes quickly. Remember that upon using Odlyzko's algorithm, all we *really* need to be able to do is to locate a prime quickly inside an interval $[N - N^{0.51}, N]$, since we can always localize to some interval like that containing primes, using only $N^{0.49}$ operations. The sort of sum we would need to evaluate in order to compute this "short interval" generating function is

$$\sum_{d \le \sqrt{N}} \sum_{(N-N^{0.51})/d \le m \le N/d} x^{dm} \pmod{2, f(x)}, \tag{3}$$

since it's not hard to imagine that computing sums such as this can be used to compute

$$\sum_{\substack{N-N^{0.51} < n \le N \\ n \text{ not a } \square}} (\tau(n)/2) x^n \pmod{2, f(x)}$$

efficiently, which can be used to compute

$$\sum_{\substack{N-N^{0.51} < p \le N \\ p \text{ prime}}} x^p \pmod{2, f(x)}$$

efficiently.

And as a bonus, it turns out that upon confining ourselves to such short intervals (width $N^{0.51}$) we can beat the trivial algorithm!

But now how do we do this – how do we beat the trivial bound of $N^{1/2+o(1)}$ for computing (3) efficiently (using the geometric series formula)? Well, the first thing to do is to split off the small d's from the rest; that is, we just compute

$$\sum_{d \le N^{1/2-\delta}} \sum_{(N-N^{0.51})/d \le m \le N/d} x^{dm} \pmod{2, f(x)}$$

using the geometric series formula. So far, we have only consumed $N^{1/2-\delta+o(1)}$ operations (OK, we lost a $N^{o(1)}$ – not important... we can just change δ to compensate). What remains, then, is

$$\sum_{N^{1/2-\delta} < d \le N^{1/2}} \sum_{(N-N^{0.51})/d \le m \le N/d} x^{dm} \pmod{2, f(x)}. \tag{4}$$

Now of course we don't have a function like $\lfloor N/n \rfloor$ to play with that we can linearize. But we *can* linearize *something* here: The first thing to note is that even if we just added every term together, and didn't use the geometric series formula, we would get an upper bound of

$$N^{1/2+o(1)} \cdot (N^{0.51}/N^{1/2-\delta}) \;=\; N^{0.51+\delta+o(1)}$$

for the running time. So, all we have to do is to improve this by a factor $N^{0.01+\delta+o(1)}$, and we are in business.

Now we need an observation: Notice that if

$$m/d \;=\; a/q + O(1/qQ),\; q \le Q,$$

then

$$|dm - (d - tq)(m + ta)| \;\ll\; t(d/Q) + t^2 aq,$$

which is quite a bit smaller than dm, provided q is much smaller than d, provided Q is not too small, and provided that t isn't "too big". So, if dm were not too near the endpoints of the interval $[N - N^{0.51}, N]$, then we would expect that for "t not too big", $(d - tq)(m + ta)$ is also in that interval.

What this allows us to do is to take the sum in (4), and to decompose it into a bunch of sums that look like

$$\sum_{t_0 \le t \le t_1} x^{(d-tq)(m+ta)} \pmod{2, f(x)}. \tag{5}$$

Okay.... but how does this help? Well, that's where another trick comes in: It turns out that there is a way to convert the problem of evaluating sums such as this one into a multipoint polynomial evaluation problem, and then one can apply Strassen's algorithm. Once the dust has settled, one arrives at an algorithm to evaluate (5) in time at most $(t_1 - t_0)^{0.9+o(1)}$ – so, there is a saving by a factor $(t_1 - t_0)^{0.1}$, which is all we need.

What about this multipoint polynomial evaluation algorithm? Well, you can find it by going to the following note:

`http://www.math.gatech.edu/~ecroot/fast_strassen.pdf`

The vast majority of what makes my algorithm "technical and complex" is that there are lots of little cases to consider. e.g. What happens if dm really is "near the endpoints"... how do we show that can't happen too often to hurt us? And, how do we choose the forms $(d - tq)(m + ta)$ so that they don't cover the same numbers twice, and yet cover all the terms we need?