Nathan McKenzie
11-23-11
nathan@icecreambreakfast.com

Summary – this is an overview of my approach to counting prime sums. It should run, I think, in O(n^2/3 log n) time and O(n^1/3 log n) space.

This algorithm is quite similar to the Mertens function counting function found here http://projecteuclid.org/DPubS?verb=Display&version=1.0&service=UI&handle=euclid.em/1047565447 ; familiarity with that algorithm might make the mechanics of the algorithm more clear.

A C implementation of this algorithm can be found at

http://www.icecreambreakfast.com/primecount/primesumcount.cpp

Overview

Okay, here's a gloss of my prime sum counting algorithm.

The high level summary version:

1) We can express sums of the (primes powers)^A up to n in terms of a number of sum of divisor-style terms that are ideally easier to reason about. This is basically a generalized (summed) version of a identity of Linnik.

2) By a kind of moebius inversion, we can extract the sum of primes^A from that.

So then the problem becomes finding a way to compute those sum of divisor-style terms. This is actually a really broad and interesting problem, and one that I think might be susceptible to better ideas than the ones I'm providing. But here's my best approach:

3) There is a way to express these sum-of-divisor style terms involving only versions of those same terms less than or equal to n^2/3. Better still, if we imagine we live in a world where we can look up our sum of divisor terms in constant time as long as they are less than n^2/3, we can compute the larger sums in O(n^2/3) time. It's a bit complicated looking, but it works.

4) By using a segmented sieve, we can compute these divisor-style sums in O(n^2/3 log n) time and O(n^1/3 log n) space.

5) If we rearrange our identities from 3), and interleave their computation with the segmented sieving mentioned in 4), we can compute our larger sums in O(n^2/3) time and O(n^1/3) space, which we can then use to compute our sum of primes^A using the identities in 1) and 2)

If you happen to have Mathematica handy, if you copy and paste in the following bits of mathematica code, you can try out many of the identities and functions this document will refer to.

```
TestPrimePowerCount[ A_, n_] := FullSimplify[ Sum[ MangoldtLambda[ j]/Log[j] j^A, {j,2,n}]]
ReferenceSumPrimes[A_, n_ ] := Sum[ 1/(j ) MoebiusMu[ j ]TestPrimePowerCount[ j A, n^(1/j)], {j,1,
Log[2,n]} ]
```

```
StrictDivisors[A_, k_, n_ ]:= Sum[ j^A StrictDivisors[A, k-1, n/j], {j, 2, n}]
StrictDivisors[A_, 1, n_ ] := Sum[ j^A, {j,2,n}]
SumPrimes[A_, n_ ] := Sum[ (-1)^(k+1)/(j k) MoebiusMu[ j ]StrictDivisors[ j A, k, n^(1/j)], {j,1,
Log[2,n]}, {k,1,Log[2,(n^(1/j))]} ]

RecurseCount[ A_, k_, n_  ] := Sum[ j^A( 1/k - RecurseCount[ A, k+1, n/j]), {j,2,n}]
SumPrimesRecurse[A_, n_] := Sum[ 1/j MoebiusMu[ j ]RecurseCount[ j A, 1, n^(1/j)], {j,1, Log[2,n]} ]

StrictDivisorsHyperbola[ A_, k_, n_, s_] := Sum[ ((m^A)^(k-j)) Binomial[ k, j] StrictDivisorsHyperbola[
A, j, n/(m^(k-j)), m+1], {m,s,n^(1/k)}, {j,0,k-1}]
StrictDivisorsHyperbola[A_,1, n_,s_] := Sum[ j^A, { j, s, n}]
StrictDivisorsHyperbola[0,1, n_,s_] := Floor[n] - s + 1
StrictDivisorsHyperbola[1,1, n_,s_] := Floor[n](Floor[n]+1)/2 - s (s-1)/2
StrictDivisorsHyperbola[2,1, n_,s_] := Floor[n](Floor[n]+1)(2Floor[n]+1)/6 - (s-1)s (2s-1)/6
StrictDivisorsHyperbola[3,1, n_,s_] := Floor[n]^2(Floor[n]+1)^2/4 - s^2(s-1)^2/4
StrictDivisorsHyperbola[A_,0, n_,s_] := 1
SumPrimesHyperbola[A_, n_] := Sum[ (-1)^(k+1)/(j k) MoebiusMu[ j ]StrictDivisorsHyperbola[ j A,k,
n^(1/j), 2], {j,1, Log[2,n]}, {k,1,Log[2,(n^(1/j))]} ]

Smalld[ A_, k_, n_] := StrictDivisorsHyperbola[ A, k, n,2] - StrictDivisorsHyperbola[ A, k, n-1,2]
StrictDivisorsReduced[ a_, A_, k_, n_ ]:= Sum[ Smalld[ A, 1, j] StrictDivisors[ A, k-1, n/j], {j, a+1,
n}]+
  Sum[ Smalld[ A, k-1, j] StrictDivisors[ A, 1, n/j], {j, 2, a}]+
  Sum[ Smalld[ A, 1, s] Smalld[ A, m, j] StrictDivisors[ A, k-m-1, n/( j s )], {j,2,a}, {s, Floor[a/j]
+ 1, n/j}, {m,1,k-2}]
StrictDivisorsReduced[ a_, A_, 1, n_ ] := Sum[ j^A, {j,2,n}]
SumPrimesReduced[A_, n_] := Sum[ (-1)^(k+1)/(j k) MoebiusMu[ j ]StrictDivisorsReduced[ Floor[ n^(1/3)],
j A,k, n^(1/j)], {j,1, Log[2,n]}, {k,1,Log[2,(n^(1/j))]} ]

StrictDivisorsFullReduced[ A_, k_, n_]:=Sum[ j^A StrictDivisorsHyperbola[ A, k-1, n/j,2], {j,
Floor[ n^(1/3)]+1, n^(1/2)}]+
  Sum[ Sum[ m^A, {m, Floor[ n/(j+1)]+1, n/j}]StrictDivisorsHyperbola[ A, k-1, j,2], {j,1,n /
Floor[n^(1/2)] - 1}]+
  Sum[ Smalld[ A, k-1, j] Sum[ m^A, {m,2,n/j}], {j,2,n^(1/3)}]+
  Sum[ s^A Smalld[ A,m,j] StrictDivisorsHyperbola[ A, k-m-1, n/(j s),2], {j, 2, n^(1/3)}, {s,
Floor[ Floor[ n^(1/3)]/j]+1, Floor[ n/j]^(1/2)}, {m,1,k-2}]+
  Sum[ (Sum[ m^A, {m,  Floor[ n/(j(s+1))]+1, n/(j s)}] )(Sum[ Smalld[ A, m, j] StrictDivisorsHyperbola[
A, k-m-1, s,2], {m,1,k-2}]), {j,2,n^(1/3)}, {s,1, Floor[ n/j] / Floor[Floor[ n/j ]^(1/2)] - 1}]
StrictDivisorsFullReduced[ A_, 1, n_ ] := Sum[ j^A, {j,2,n}]
SumPrimesFullReduced[A_, n_] := Sum[ (-1)^(k+1)/(j k) MoebiusMu[ j ]StrictDivisorsFullReduced[ j A,k,
n^(1/j)], {j,1, Log[2,n]}, {k,1,Log[2,(n^(1/j))]} ]
```

At any rate, here we go.

The Main Identity

First, it's handy to go check up on Linnik's identity. This link will give an overview:

http://books.google.com/books?id=8i7wpzjSWrIC&pg=PA342&lpg=PA342&dq=linnik
%27s+identity#v=onepage&q=linnik%27s%20identity&f=false

It might not be obvious, but if we sum Linnik's identity from 2 to n, we have

$$\pi(n)+\frac{1}{2}\pi(n^{\frac{1}{2}})+\frac{1}{3}\pi(n^{\frac{1}{3}})+\ldots=\sum_{j=2}^{n}1-\frac{1}{2}\sum_{j=2}^{n}\sum_{k=2}^{\frac{n}{j}}1+\frac{1}{3}\sum_{j=2}^{n}\sum_{k=2}^{\frac{n}{j}}\sum_{l=2}^{\frac{n}{jk}}1+\frac{1}{4}\ldots$$

or, which is the same,

$$\sum_{j=2}^{n} \frac{\Lambda(j)}{\log j} = \sum_{j=2}^{n} 1 - \frac{1}{2} \sum_{j=2}^{n} \sum_{k=2}^{\frac{n}{j}} 1 + \frac{1}{3} \sum_{j=2}^{n} \sum_{k=2}^{\frac{n}{j}} \sum_{l=2}^{\frac{n}{jk}} 1 + \frac{1}{4} \ldots$$

where $\Lambda(n)$ is the Mangoldt function, as usual. This is actually a specialization of

$$\sum_{j=2}^{n} \frac{\Lambda(j)}{\log j} \cdot j^A = \sum_{j=2}^{n} j^A - \frac{1}{2} \sum_{j=2}^{n} \sum_{k=2}^{\frac{n}{j}} j^A \cdot k^A + \frac{1}{3} \sum_{j=2}^{n} \sum_{k=2}^{\frac{n}{j}} \sum_{l=2}^{\frac{n}{jk}} j^A \cdot k^A \cdot l^A + \frac{1}{4} \ldots \tag{1}$$

This is the identity we really want to use (you can take this even further and swap out j^A with any $f(j)$ that is completely multiplicative, but we don't need to go there)

So, this turns the problem of counting prime power sums at different powers into a problem similar to (but not identical to) computing the Dirichlet Divisor problem for divisors with different indices. In fact, let's make that connection more clear. Let's introduce the following divisor summatory-style function:

$$D_k^{(A)}(n) = \sum_{j=2}^{n} j^A D_{k-1}^{(A)}\left(\frac{n}{j}\right)$$
$$D_1^{(A)}(n) = \sum_{j=2}^{n} j^A$$
$$D_0^{(A)}(n) = 1 \tag{2}$$

Mathematica: `StrictDivisors[A_, k_, n_ ]`

So, given our new sums from (2), we can rewrite (1) as

$$\sum_{j=2}^{n} \frac{\Lambda(j)}{\log j} \cdot j^A = \sum_{j=1}^{\log_2 n} \frac{-1^{j+1}}{j} D_j^{(A)}(n) \tag{3}$$

Two important notes here for later computations. One is that, if you look closely at the mechanics of (2), it should be clear that if $n < 2^k$, $D_k^{(A)}(n) = 0$. Thus, in (1) there will only be $\log_2 n$ nested sums we have to calculate.

There other note is that the term $D_1^{(A)}(n) = \sum_{j=2}^{n} j^A$ can often be computed in constant time, depending on the value of A. So, for example,

$$D_1^{(0)}(n) = \sum_{j=2}^{n} j^0 = \lfloor n \rfloor - 1$$
$$D_1^{(1)}(n) = \sum_{j=2}^{n} j^1 = \frac{(\lfloor n \rfloor)(\lfloor n \rfloor + 1)}{2} - 1$$

$$D_1^{(2)}(n)=\sum_{j=2}^{n} j^2=\frac{(\lfloor n\rfloor)(\lfloor n\rfloor+1)(2\lfloor n\rfloor+1)}{6}-1$$
$$D_1^{(3)}(n)=\sum_{j=2}^{n} j^3=\frac{\lfloor n\rfloor^2(\lfloor n\rfloor+1)^2}{4}-1$$
(2a)

and so on. This is just the general "summing natural numbers from 2 through n" set of formulas, of course. It's an open question, for me, how successfully this approach would work for non-integer or negative values of *A*.

(Also, as an interesting aside, (1) can also be rewritten recursively as

$$P_k^{(A)}(n)=\sum_{j=2}^{n} j^A\left(\frac{1}{k}-P_{k+1}^{(A)}\left(\frac{n}{j}\right)\right)$$

Mathematica: **`RecurseCount[ A_, k_, n_  ]`**

In this notation,

$$\sum_{j=2}^{n}\frac{\Lambda(j)}{\log j}\cdot j^A=P_1^{(A)}(n)$$

Mathematica: **`RecurseCount[ A_, 1, n_  ]`**

Also compare the resulting **`SumPrimesRecurse[A_, n_]`**
to **`ReferenceSumPrimes[A_, n_]`**

It's not at all quick, but it is tidy.)

But, our final goal is to count primes, not prime powers. I'm going to gloss where the following statement comes from, but it's not too tricky to show that

$$\sum_{p\le n} p^A=\sum_{m=1}^{\log_2 n}\frac{\mu(m)}{m}\sum_{j=2}^{n^{\frac{1}{m}}}\frac{\Lambda(j)}{\log j}\cdot j^{m\cdot A}$$

(the $\log_2 n$ ceiling is because once $n^{\frac{1}{m}}<2$ the inner sum becomes 0, of course)

And so, by (3),

$$\sum_{p\le n} p^A=\sum_{j=1}^{\log_2 n}\sum_{k=1}^{\log_2 n^{\frac{1}{j}}}\frac{-1^{k+1}}{jk}\mu(j)D_k^{(A\cdot j)}(n^{\frac{1}{j}})$$
(4)

Mathematica: **`SumPrimes[A_, n_ ]`**
Identical output to **`ReferenceSumPrimes[A_, n_ ]`**

As a quick analogy, this is basically a slight generalization of the typical relationship for prime counting

$$\Pi(n)=\pi(n)+\frac{1}{2}\pi(n^{\frac{1}{2}})+\frac{1}{3}\pi(n^{\frac{1}{3}})+\ldots$$

being inverted to

$$\pi(n)=\Pi(n)+\frac{1}{2}\mu(2)\Pi(n^{\frac{1}{2}})+\frac{1}{3}\mu(3)\Pi(n^{\frac{1}{3}})+...$$

So that's the general overview of this approach, embodied in (4) – to count the sum of primes to the power of *A* up to some number *n*, we need to find that fastest way to compute those divisor summatory style functions from (2). As written in (2), this is painfully slow.

The sums in (2) are interesting, I think, in that they open the door to a really wide range of possible techniques to compute. I've explored quite a few myself – the best I've come up with is what follows.

An Aside

This is the first interesting way to compute those sums. It results in something like an O(n) time, O(epsilon) space algorithm, with good constant time factors, but I keep hoping I might be overlooking something clever to get more mileage out of it.

Basically, it's a generalization of the Dirichlet Hyperbola method to higher degrees.

$$D_k^{(A)}(n,s)=\sum_{m=s}^{n^{\frac{1}{k}}}\sum_{j=0}^{k-1} m^{A(k-j)}\binom{k}{j} D_j^{(A)}\left(\frac{n}{m^{k-j}},m+1\right)$$

$$D_1^{(A)}(n,s)=\sum_{m=s}^{n} m^A$$

$$D_0^{(A)}(n,s)=1$$

(5)

Mathematica: `StrictDivisorsHyperbola[ A_, k_, n_, s_]`

(And remember, $D_1^{(A)}(n,s)=\sum_{j=s}^{n} j^A$ can be computed in constant or log time for many values of *A,* as per (2a))

The important thing to know here is that (2) is connected to (5) by

$$D_k^{(A)}(n)=D_k^{(A)}(n,2)$$

Mathematica: `StrictDivisorsHyperbola[ A_, k_, n_, 2]`
Identical output to `StrictDivisors[A_, k_, n_ ]`

So our expression for prime sum counting with (5) is

$$\sum_{p\le n} p^A=\sum_{j=1}^{\log_2 n}\sum_{k=1}^{\log_2 n^{\frac{1}{j}}} \frac{-1^{k+1}}{jk}\mu(j) D_k^{(A\cdot j)}(n^{\frac{1}{j}},2)$$

Mathematica: `SumPrimesHyperbola[A_, n_]`
Identical output to `ReferenceSumPrimes[A_, n_ ]`

Incidentally, if we set A to 0 (so we're just doing prime counting), and we use a suitably large wheel, (so we're using a version of (5) that only allows *m* to take on values that aren't divisible by, say, primes < 23) then (5) provides for a surprisingly fast no memory prime counting algorithm, at least for numbers < 10^12 or so.

The Main Sum

The idea in the aside is interesting, but it won't get you to O(n^2/3) time in computation, or at least it won't with any ideas I have. So let's put it aside. Instead, we're going to turn to an approach that has a lot in common with the one found in

Deleglise, Marc and Rivat, Joel, Computing the summation of the Mobius function. Experiment. Math. 5 (1996), no. 4, 291-295.

So, let's take a look at the identities from (2) we want to compute. The basic identity is

$$D_k^{(A)}(n)=\sum_{j=2}^{n} j^A D_{k-1}^{(A)}(\frac{n}{j}) \tag{6}$$

This identity works, but it has the nasty quality of relying on large values like $D_{k-1}^{(A)}(\frac{n}{2})$, $D_{k-1}^{(A)}(\frac{n}{3})$, and so on. We'd really like to reduce this to a problem relying on sums involving smaller functions.

So that's the goal here.

First, let's define the difference between two consecutive instances of (6) as

$$d_k^{(A)}(n)=D_k^{(A)}(n)-D_k^{(A)}(n-1) \tag{6a}$$

This is a rough analog to the divisor function. It can be computed as

$$d_k^{(A)}(n)=d_k^{(0)}(n)\cdot n^A \tag{6b}$$

In turn, $d_k^{(0)}(n)$ (which is essentially the count of divisor function for different indices of k, but excluding any entries with values of 1 – this is the function Iwaniec refers to as the strict count of divisors in the first link), can be expressed in terms of the normal count of divisor function by

$$d_k^{(0)}(n)=\sum_{j=0}^{k} -1^{k-j}\binom{k}{j} d_j(n) \tag{6c}$$

The normal count of divisors, in turn, can be calculated as

$$d_k(n)=\binom{a_1+k-1}{a_1}\cdot\binom{a_2+k-1}{a_2}\cdot\binom{a_3+k-1}{a_3}\cdot...$$

(6d)

if we have our number n in prime factored form as

$$n = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$$

Later, we will be factoring numbers in bulk by sieving, which means we will be able to calculate $d_k(n)$, and thus $d_k^{(0)}(n)$, and thus $d_k^{(A)}(n)$, in something like log time by the steps above.

So, given our identities for $d_k^{(A)}(n)$, we can express (6) as follows. I'm going to skip the steps involved here, but by a certain clever process, we have

$$\begin{aligned} D_k^{(A)}(n) = & \\ & \sum_{j=a+1}^{n} d_1^{(A)}(j) \cdot D_{k-1}^{(A)}\left(\frac{n}{j}\right) \\ & + \sum_{j=2}^{a} d_{k-1}^{(A)}(j) \cdot D_1^{(A)}\left(\frac{n}{j}\right) \\ & + \sum_{j=2}^{a} \sum_{s=\frac{a}{j}+1}^{\frac{n}{j}} \sum_{m=1}^{k-2} d_1^{(A)}(s) \cdot d_m^{(A)}(j) \cdot D_{k-m-1}^{(A)}\left(\frac{n}{js}\right) \end{aligned}$$

(7)

Mathematica: `StrictDivisorsReduced[ a_, A_, k_, n_ ]`
Identical output to `StrictDivisors[A_, k_, n_ ]`

Also compare the resulting `SumPrimesReduced[A_, n_]`
to `ReferenceSumPrimes[A_, n_]`

for some value $1 < a < n$. The most important aspect of this identity, which admittedly looks a bit complicated, is that it expresses $D_k^{(A)}(n)$ in terms of values of $D_k^{(A)}$ no greater than $\frac{n}{a}$ and $d_k^{(A)}$ no greater than a. If we select as our value of a $n^{1/3}$, then we can compute $D_k^{(A)}(n)$ if we can compute values up to $D_1^{(A)}(n^{2/3})$, $D_2^{(A)}(n^{2/3})$, ... $D_{k-1}^{(A)}(n^{2/3})$, and values up to $d_1^{(A)}(n^{1/3})$, $d_2^{(A)}(n^{1/3})$, … $d_{k-1}^{(A)}(n^{1/3})$ within our time bound. The one exception to this is for values of $D_1^{(A)}(n)$ and $d_1^{(A)}(n)$, which should be computable in near constant time for any sensible values of n based on (2a) and then (5a). Particularly, $d_1^{(A)}(n) = n^A$. It's worth looking through the sums to verify that they do, in fact, have these properties.

So, to recap, we want to compute the sum on the left of

$$\sum_{p \le n} p^A = \sum_{j=1}^{\log_2 n} \sum_{k=1}^{\log_2 n^{\frac{1}{j}}} \frac{-1^{k+1}}{jk} \mu(j) D_k^{(A \cdot j)}(n^{\frac{1}{j}})$$

for some value of A. Our equation turns the problem into one of computing $D_2^{(A)}(n)$, $D_3^{(A)}(n)$, $D_4^{(A)}(n)$, and so on, given in (2). (It should be pointed out that this sum also has us computing $D_2^{(2A)}(n^{1/2})$ and further values, but they are relatively trivially to compute).

Then, the equation in (7) asserts that we can turn the computation of $D_k^{(A)}(n)$ into a problem that requires, at most, calculation of values up around $D_k^{(A)}(n^{2/3})$, if we set a to be n^1/3. This is good. On the other hand, the triple sum on the last line of (7) looks like it runs through too many values to perform well, and the first sum is too large as well.

So, here will be our next step.

First, we're going to take (7) and replace values of $D_1^{(A)}(n)$ and $d_1^{(A)}(n)$ with their actual values, $\sum_{j=2}^{n} j^A$ and n^A .

Second, we have specified the variable $a=n^{\frac{1}{3}}$.

Last, there are symmetries in (7) that can be expressed more compactly than they are in (7). This is analogous to, though a bit different from, the fact that for some function f,

$$\sum_{j=2}^{n} f(\lfloor\frac{n}{j}\rfloor)=\sum_{j=2}^{n^{\frac{1}{2}}} f(\lfloor\frac{n}{j}\rfloor)+\sum_{j=1}^{\frac{n}{n^{\frac{1}{2}}}-1} (\lfloor\frac{n}{j}\rfloor-\lfloor\frac{n}{j+1}\rfloor) f(j)$$

I'm going to hand-wave over that process, but it is accounted for in our next equation.

$$\begin{aligned}
D_k^{(A)}(n)= & \\
& \sum_{j=\lfloor n^{\frac{1}{3}}\rfloor+1}^{n^{\frac{1}{2}}} j^A\cdot D_{k-1}^{(A)}(\frac{n}{j}) \\
& +\sum_{j=1}^{\frac{n}{\lfloor n^{\frac{1}{2}}\rfloor}-1} (\sum_{m=\lfloor\frac{n}{j+1}\rfloor+1}^{\frac{n}{j}} m^A) D_{k-1}^{(A)}(j) \\
& +\sum_{j=2}^{n^{\frac{1}{3}}} d_{k-1}^{(A)}(j)\cdot(\sum_{m=2}^{\frac{n}{j}} m^A) \\
& +\sum_{j=2}^{n^{\frac{1}{3}}} \sum_{s=\frac{\lfloor n^{\frac{1}{3}}\rfloor}{j}+1}^{\lfloor\frac{n}{j}\rfloor^{\frac{1}{2}}} \sum_{m=1}^{k-2} s^A\cdot d_m^{(A)}(j)\cdot D_{k-m-1}^{(A)}(\frac{n}{js}) \\
& +\sum_{j=2}^{n^{\frac{1}{3}}} \sum_{s=1}^{\frac{\lfloor\frac{n}{j}\rfloor}{\lfloor\lfloor\frac{n}{j}\rfloor^{\frac{1}{2}}\rfloor}-1} (\sum_{m=\lfloor\frac{n}{j(s+1)}\rfloor+1}^{\frac{n}{js}} m^A)\cdot \sum_{m=1}^{k-2} d_m^{(A)}(j)\cdot D_{k-m-1}^{(A)}(s)
\end{aligned}$$

(8)

Mathematica: **`StrictDivisorsFullReduced[ A_, k_, n_]`**
Identical output to **`StrictDivisors[ A_, k_, n_]`**

Also compare the resulting **`SumPrimesFullReduced[A_, n_]`**
to **`ReferenceSumPrimes[A_, n_]`**

So this is our main equation.

The terms in red should generally be replaceable with smaller terms as described in (2a), so it's worth remembering that those can be calculated generally in constant time.

Also, looking at this equation, remember the claim made back at (7) – I'm asserting that we have some process for looking up values up to $D_k^{(A)}(n^{2/3})$ and values up to $d_k^{(A)}(n^{1/3})$ in our time and space bound.

Given those caveats, we can compute this sum in our time and space bounds.

The Sieve

So, the previous section claimed to provide (as (8)) a method for computing the function $D_k^{(A)}(n)$ in under O(n^2/3 log n) time assuming that there was a way to get access to values up to $D_k^{(A)}(n^{2/3})$ and values up to $d_k^{(A)}(n^{1/3})$ within our time bound. That computation, in turn, will let us compute

$$\sum_{p\le n} p^A = \sum_{j=1}^{\log_2 n} \sum_{k=1}^{\log_2 n^{\frac{1}{j}}} \frac{-1^{k+1}}{jk} \mu(j) D_k^{(A\cdot j)}(n^{\frac{1}{j}})$$

which is our ultimate goal.

This section will describe how we can compute values up to $D_k^{(A)}(n^{2/3})$ and values up to $d_k^{(A)}(n^{1/3})$ within our time bound.

First, we will keep a list of all primes up to $n^{\frac{1}{3}}$. We will also keep a buffer of size $n^{\frac{1}{3}}$ with log n entries. This makes our memory boundary. Then, we will repeat the following steps $n^{\frac{1}{3}}$ times.

First, we will sieve all of the numbers in our current range to get them in their fully prime factorized forms. To do this up to $n^{\frac{2}{3}}$ obviously only requires primes up to $n^{\frac{1}{3}}$. This kind of sieving will be doable in something like O(n^2/3 log n) time.

For any given value *v* in our current segment of the sieve, if we have the number in fully prime factored form (particularly its power signature) at $n=p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$, we can use

$$d_k(n)=\binom{a_1+k-1}{a_1}\cdot\binom{a_2+k-1}{a_2}\cdot\binom{a_3+k-1}{a_3}\cdot\ldots$$

and then

$$d_k^{(0)}(n)=\sum_{j=0}^{k}-1^{k-j}\binom{k}{j}d_j(n)$$

and finally

$$d_k^{(A)}(n)=d_k^{(0)}(n)\cdot n^A$$

(which were (6d), then (6c), then (6b)) to compute $d_1^A(v)$, $d_2^A(v)$, through $d_{\log_2 n}^A(v)$.

Then, clearly, if we rearrange by (6a), we have

$$D_k^{(A)}(n-1)+d_k^{(A)}(n)=D_k^{(A)}(n)$$

What this means is that if we are calculating all the values of $d_k^{(A)}(n)$ in sequence from 1 up to $n^{\frac{2}{3}}$, we can clearly keep a running tally of all the values of $D_k^{(A)}(n)$ as well up to $n^{\frac{2}{3}}$ as well.

All this can be done in something like O(n^2/3 log n) time and O(n^1/3 log n) space.

Interleaving / Executing

To preserve our memory bounds, we segment the sieve. This means that, though we will have access to all values of $D_k^{(A)}(n)$ and $d_k^{(A)}(n)$ less than $n^{\frac{2}{3}}$ over the course of computation, we won't have access to them all at the same time – rather, we will have access to them sequentially from smallest to greatest.

Therefore, we need to take our identity from (8) and reorder the order that we compute our sums, from smallest values of $D_k^{(A)}(n)$ and $d_k^{(A)}(n)$ to the greatest values.

In practice, for (8), this means we will interleave computation of

$$\sum_{j=1}^{\frac{n}{n^{\frac{1}{2}}}-1}\Big(\sum_{m=\lfloor\frac{n}{j+1}\rfloor+1}^{\frac{n}{j}}m^A\Big)D_{k-1}^{(A)}(j)$$

with our sieving; once that sum has been computed, we compute

$$\sum_{j=\lfloor n^{\frac{1}{3}}\rfloor+1}^{n^{\frac{1}{2}}}j^A\cdot D_{k-1}^{(A)}\Big(\frac{n}{j}\Big)$$

but we will do so in reverse order, from $j=\lfloor n^{\frac{1}{2}}\rfloor$ through $j=\lfloor n^{\frac{1}{3}}\rfloor+1$, all interleaved with the sieving process.

A similar approach is taken with all the sums of the form

$$\sum_{j=2}^{n^{\frac{1}{3}}} \sum_{s=1}^{\frac{\lfloor \frac{n}{j} \rfloor}{\lfloor \lfloor \frac{n}{j} \rfloor^{\frac{1}{2}} \rfloor} - 1} \left(\sum_{m=\lfloor \frac{n}{j(s+1)} \rfloor + 1}^{\frac{n}{js}} m^A \right) \cdot \sum_{m=1}^{k-2} d_m^{(A)}(j) \cdot D_{k-m-1}^{(A)}(s)$$

and then

$$\sum_{j=2}^{n^{\frac{1}{3}}} \sum_{s=\frac{\lfloor n^{\frac{1}{3}} \rfloor}{j}+1}^{\lfloor \frac{n}{j} \rfloor^{\frac{1}{2}}} \sum_{m=1}^{k-2} s^A \cdot d_m^{(A)}(j) \cdot D_{k-m-1}^{(A)}\left(\frac{n}{js}\right) .$$

Rough Conclusion

This seems to work.

My original prime counting implementation could be sped up a bit by integrating a wheel into its computation. I haven't done that yet. I'm not sure how a wheel would interact with the sums from (2a), if at all. That could be a significant issue.

I'm highly curious if there are other interesting ways to calculate the sums in (2).

This is a pretty rough write up, I know. To read some clearer arguments about where some of these identities come from (in particular, (7)), and to see the source code for my implementation of this general approach for when A=0 (the prime counting version), see
http://www.icecreambreakfast.com/primecount/PrimeCounting_NathanMcKenzie.pdf

Once again, a current C implementation of the above can be found at

http://www.icecreambreakfast.com/primecount/primesumcount.cpp