Calculating exact values of $N(x,m)$ without using recurrence relations

This note describes an algorithm for calculating exact values of $N(x,m)$, the number of partitions of $\lfloor m(x+1)/2 \rfloor$ into m distinct positive integers each less than or equal to x, without using recurrence relations. For example, with $x = m^2$, $N(x,m)$ is equal to the number of magic series of order m, see OEIS sequence A052456.

This algorithm does not suffer from the quite heavy memory constraints imposed by algorithms based on recurrence relations, and can be used successfully to calculate $N(x,m)$ for combinations of x and m that are out of scope for other programs. Also, it is very well suited for parallel computation.

On the downside, this algorithm has to restart from scratch every time $N(x,m)$ has to be calculated for new values of m and/or x, which makes it in principle less suitable for generating tables (unless of course no other options are available). Even so, our straightforward implementation, without other optimizations than the ones described in this note, appears to be faster than the C program from Robert Gerbicz that was used to generate the first 150 terms of OEIS sequence A052456.

Let $p_K(m,n)$ denote the number of partitions of K with at most m parts each less than or equal to n. By comparing the definitions of $N(x,m)$ and $p_K(m,n)$ it is clear that

$$N(x,m) = p_{\lfloor m(x-m)/2 \rfloor}(m, x-m)$$

In order to calculate exact values of $N(x,m)$ we will use the following known property of the $p_K(m,n)$:

$$\sum_{K=0}^{mn} p_K(m,n)\, q^K = \binom{m+n}{m}_q$$

where the expression of the right-hand side is the q-binomial coefficient

$$\binom{m+n}{m}_q = \prod_{i=1}^{m} \frac{1-q^{n+i}}{1-q^i}$$

An elegant proof of this property can be found in http://ocw.mit.edu/courses/mathematics/18-318-topics-in-algebraic-combinatorics-spring-2006/lecture-notes/young.pdf, theorem 6.6. Because the summation on the left-hand side of the previous equation is a polynomial of degree mn in q, the q-binomial coefficient on the right-hand side of the equation is also guaranteed to be a polynomial in q (of course also of degree mn), which is not obvious from the product form definition of the q-binomial coefficient.

So, if we use the common notation $[q^K]f(q)$ to represent the coefficient of q^K in $f(q)$, where $f(q)$ can be any polynomial (or a formal power series) in q, we have

$$p_K(m,n) = [q^K]\binom{m+n}{m}_q$$

and substituting $K = \lfloor m(x-m)/2 \rfloor$ and $n = x - m$ ($x \geq m > 0$), we have

$$N(x,m) = p_{\lfloor m(x-m)/2 \rfloor}(m, x-m) = \left[q^{\lfloor m(x-m)/2 \rfloor}\right]\binom{x}{m}_q$$

Now, at least in principle, starting from the known initial values (constant polynomials)

$$\binom{k}{0}_q = 0 \ (k > 0) \text{ and } \binom{k}{k}_q = 1 \ (k \geq 0)$$

all the coefficients of the polynomial $\binom{x}{m}_q$ of degree $m(x-m)$ in q can be computed using a recurrence formula such as

$$\binom{x}{m}_q = q^m\binom{x-1}{m}_q + \binom{x-1}{m-1}_q$$

Such an approach requires that all the coefficients needed by the recurrence are stored. As x and/or m increase substantially, this will become problematic.

The same polynomial could also be computed by other (more complicated) methods, e.g. as a product of certain cyclotomic polynomials. However, in order to find $N(x,m)$ we only need to determine *one particular* coefficient of the polynomial. It is possible to determine any coefficient by evaluating the polynomial for some well-chosen values of q (specifically: roots of unity). Moreover, such evaluations can be performed quite efficiently using the above equation

$$\binom{m+n}{m}_q = \prod_{i=1}^{m}\frac{1-q^{n+i}}{1-q^i}$$

which, after the substitution $n = x - m$ ($x \geq m > 0$), becomes

$$\binom{x}{m}_q = \prod_{i=1}^{m}\frac{1-q^{x-m+i}}{1-q^i}$$

So we are looking for a general method which allows us to determine any coefficient c_k of any polynomial

$$h(q) = c_0 + c_1 q + \cdots + c_d q^d$$

where the coefficients c_k are unknown, but where nevertheless $h(q)$ is given in some form that allows us to *evaluate* $h(\alpha)$ for certain values of α. An example of such a polynomial is

$$h(q) = \binom{x}{m}_q$$

because indeed, although we do not know the coefficients c_k in advance, we can still *evaluate* this polynomial using the simple equation

$$h(q) = \prod_{i=1}^{m} \frac{1 - q^{x-m+i}}{1 - q^i}$$

If the right-hand side contains zeroes in one or more of the denominators, the point in which the function is being evaluated will always be a *removable* discontinuity. Anyway, we want to avoid this complication, and select values of q such that none of the denominators are equal to 0. We do allow one exception though, $q = 1$, because we know that

$$h(1) = \binom{x}{m}$$

Now suppose we want to calculate the coefficient c_K ($0 \leq K \leq d$). Select any integer r such that $r > \max(K, d - K)$ (we will put further restrictions on r later in order to avoid unwanted zero denominators in the above products). For the moment, let us assume that the polynomial is evaluated in $\mathbb{C}$, the field of complex numbers. Let $\{\alpha_0, \ldots, \alpha_{r-1}\}$ be the set of all r-th roots of unity. This means that all α_j are of the form $\alpha_j = \omega^j$, where ω is a *primitive* r-th root of unity (so r has to be the *smallest* integer ≥ 0 such that $\omega^r = 1$). In $\mathbb{C}$ we can take $\omega = e^{2\pi i/r}$. Then we have

$$\sum_{j=0}^{r-1} \frac{h(\omega^j)}{\omega^{jK}} = \sum_{j=0}^{r-1} \frac{1}{\omega^{jK}} \sum_{k=0}^{d} c_k \omega^{jk} = \sum_{k=0}^{d} c_k \sum_{j=0}^{r-1} \frac{\omega^{jk}}{\omega^{jK}} = \sum_{k=0}^{d} c_k \sum_{j=0}^{r-1} \omega^{j(k-K)}$$

In the outer summation on the right-hand side we have

$$0 \leq k \leq d \Longrightarrow -K \leq k - K \leq d - K \Longrightarrow -r < k - K < r \Longrightarrow |k - K| < r$$

The inner summation on the right-hand side is of the form

$$\sum_{j=0}^{r-1} z^j$$

with $z = \omega^{k-K}$. Now, for all $z \neq 1$, the following identity holds:

$$\sum_{j=0}^{r-1} z^j = \frac{1 - z^r}{1 - z}$$

For the terms in the outer summation with $k \neq K$, we have $0 < |k - K| < r$, and $z = \omega^{k-K} \neq 1$ because ω is a *primitive* r-th root of unity. So the value of the inner summation is equal to

$$\sum_{j=0}^{r-1} z^j = \frac{1 - z^r}{1 - z} = \frac{1 - (\omega^r)^{k-K}}{1 - z} = \frac{1 - 1^{k-K}}{1 - z} = 0$$

So the only term left in the outer summation is the one with $k = K$, and we have

$$\sum_{j=0}^{r-1} \frac{h(\omega^j)}{\omega^{jK}} = c_K \sum_{j=0}^{r-1} \omega^{j(K-K)} = c_K \sum_{j=0}^{r-1} 1 = r c_K$$

Consequently,

$$c_K = \frac{1}{r}\sum_{j=0}^{r-1}\frac{h(\omega^j)}{\omega^{jK}}$$

It should now be clear that the value c_K obtained using the above formula will be equal to $N(x,m)$ if we perform the following substitutions:

$$h(q) = \binom{x}{m}_q;\ d = m(x-m);\ K = \left\lfloor\frac{d}{2}\right\rfloor;\ r \in \mathbb{Z} : r > \max(K, d-K) = \left\lceil\frac{d}{2}\right\rceil;\ \omega = e^{\frac{2\pi i}{r}}$$

Note that the coefficient extraction method described so far can easily be extended for multivariate polynomials. Other applications can be found in the literature, for example in this article from 1977, http://www.sciencedirect.com/science/article/pii/0012365X77901170, where essentially the same method is applied to homogeneous multivariate polynomials.

Optimizations are possible. As long as $r > \lceil d/2 \rceil$, with $d = m(x-m)$, we can take any $r \in \mathbb{N}$ we want. Let us assume that r is odd, so $r = 2s+1$ for some $s \in \mathbb{N}$.

$$rc_K = \sum_{j=0}^{2s}\frac{h(\omega^j)}{\omega^{jK}} = h(1) + \sum_{j=1}^{s}\frac{h(\omega^j)}{\omega^{jK}} + \sum_{j=s+1}^{2s}\frac{h(\omega^j)}{\omega^{jK}}$$

Using the fact that

$$\binom{x}{m}_{1/q} = q^{-m(x-m)}\binom{x}{m}_q = q^{-d}\binom{x}{m}_q$$

$$h(\omega^{-j}) = \omega^{-jd}h(\omega^j)$$

the last summation can be rewritten as

$$\sum_{j=s+1}^{2s}\frac{\omega^{jd}h(\omega^{-j})}{\omega^{jK}} = \sum_{j=s+1}^{2s}\omega^{j(d-K)}h(\omega^{-j})$$

In terms of a new free variable $i = r - j$ this can be further rewritten as

$$\sum_{i=1}^{s}\omega^{-i(d-K)}h(\omega^i)$$

Finally, after renaming the free variable i and substituting the last term in the above sum for rc_K we obtain

$$rc_K = h(1) + \sum_{j=1}^{s}\omega^{-jK}h(\omega^j) + \sum_{j=1}^{s}\omega^{-j(d-K)}h(\omega^j)$$

If d is even, we have $2K = d$, so $d - K = K$, and

$$rc_K = h(1) + \sum_{j=1}^{s}\omega^{-jK}h(\omega^j) + \sum_{j=1}^{s}\omega^{-jK}h(\omega^j) = h(1) + 2\sum_{j=1}^{s}\omega^{-jK}h(\omega^j)$$

If d is odd, we have $2K = d - 1$, so $d - K = K + 1$, and

$$rc_K = h(1) + \sum_{j=1}^{s} \omega^{-jK} h\left(\omega^j\right) + \sum_{j=1}^{s} \omega^{-j(K+1)} h\left(\omega^j\right) = h(1) + \sum_{j=1}^{s} \left(\omega^{-jK} + \omega^{-j(K+1)}\right) h\left(\omega^j\right)$$

In any case, we only need the values $h\left(\omega^j\right)$ for $j = 0, \dots, s$, so we can (almost) halve the execution time.

Next, let us see if and how we can avoid unwanted zero denominators in the evaluation of

$$h(q) = \prod_{i=1}^{m} \frac{1 - q^{x-m+i}}{1 - q^i}$$

We just saw that we only have to evaluate $h(q)$ for $q = \omega^j$ and $j = 0, \dots, r-1$ (or up to s if we apply the optimization). So, a zero denominator will only appear if $q^i = 1$, that is if $\omega^{ij} = 1$. Obviously, for $j = 0$ we have $q = \omega^0 = 1$, which is not a problem because we know that

$$h(1) = \binom{x}{m}$$

and we do not have to evaluate the product at all. So we only want to guarantee that $\omega^{ij} \neq 1$ for $j = 1, \dots, r-1$ (or up to s if we apply the optimization) and $i = 1, \dots, m$, which means that none of the products ij should be a multiple of r. A sufficient condition is that the smallest prime factor of r is larger than m, so we will add this as a condition when selecting $r > \lceil d/2 \rceil$. Note that if we restrict the algorithm to the case that $m \geq 2$ (we do not need an algorithm for $m = 1$ anyway), this additional condition implies that r is odd.

Now, in principle we could indeed evaluate the summation in $\mathbb{C}$, as explained. However, since we want to calculate *exact* values, and irrational numbers are involved, this may lead to numerical problems. So we will use another well-known numerical technique: we select an appropriate prime number p, and perform all the calculations in the finite field $GF(p)$, where all additions and multiplications are performed modulo p. So the result will not immediately be the value $N(x, m)$ we are looking for, unless $p > N(x, m)$, but the value of

$$N(x, m) \bmod p$$

If we repeat the whole process for various prime numbers p, we can derive the value of $N(x, m)$ using the Chinese remainder theorem, provided that the product of all the selected prime numbers is larger than $N(x, m)$.

Now the prime numbers p must be chosen carefully, because the method will fail unless there exists an element ω of multiplicative order r in $GF(p)$, meaning that r is the smallest integer ≥ 0 such that $\omega^r = 1$. Remember that in $GF(p)$ all additions and multiplications are performed modulo p. We know from abstract algebra that the multiplicative order of any element in the multiplicative group of $GF(p)$ must be a divisor of the order of the multiplicative group, which is $p - 1$ (the element 0 does not belong to the multiplicative group).

So assuming we have already selected r as described earlier, we now have to select prime numbers p such that $r | (p - 1)$. So p has to be of the form $jr + 1$, where j is an even positive integer (j must be even, because, as we explained, if $m \geq 2$, then r must be odd; but then, since p must be odd as well,

jr and thus j must be even). In practice we will use the smallest possible primes, so we start with $j = 2$, increment j by 2 each time, and drop those j for which $jr + 1$ is not prime.

Once a proper prime number has been selected, we still have to find an element ω of multiplicative order r in $GF(p)$. For this we use a rather brute force method: first we find an element z with multiplicative order (say r_z) such that $r|r_z$. We do this by trying $z = 2,3, \ldots$ successively, each time checking whether indeed $r|r_z$. We know from algebra that such values z (and r_z) can always be found. Because we also know that $r_z|(p-1)$, we do not necessarily have to completely determine r_z in order to decide that a particular value of z must be rejected. Finally, we take $\omega = z^{r_z/r}$.

There are in fact more efficient (and more complicated) ways to find ω, but our brute force method is good enough because the time needed to compute ω is very small compared to the time needed to compute the summation.

This completes the general description of the algorithm, and one possible optimization. It should now be clear that the algorithm is well suited for parallel computation, since the sums for the various prime numbers can be calculated in parallel, and independently.

I have implemented this algorithm in C++, using only standard 64 bit unsigned integers. The result is a list of pairs $(p, N(x,m) \bmod p)$. From this list it is easy to calculate the value of $N(x,m)$ (I wrote a small `dc` script for this, because it is easy, and `dc` supports large integers). I did not use discrete logarithms in order to speed up the calculations of the terms in the summations. This would be counterproductive, since the required additional table(s) would grow proportionally with p, reintroducing the memory constraints we were trying to avoid. Nevertheless, I still decided to use a very small table (of size proportional to m) in order to speed up the evaluation of the $h(\omega^j)$.

Dirk Kinnaes

2013-04-19