Partitions in the quintillions
or
Billions of congruences

Fredrik Johansson

November 2011

The partition function

$p(n)$ counts the number of ways n can be written as the sum of positive integers without regard to order.

Example: $p(4) = 5$ since
$(4) = (3+1) = (2+2) = (2+1+1) = (1+1+1+1)$

OEIS A000041: $1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42 \ldots$

Euler (1748): $\displaystyle\sum_{n=0}^{\infty} p(n)x^n = \prod_{k=1}^{\infty} \frac{1}{1-x^k}$

Computation of $p(n)$

$p(n) \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{2n/3}}$, so $p(n)$ has approximately $n^{1/2}$ digits.
Example: $p(1000) \approx 2.4 \times 10^{31}$

Lehmer (1938): $p(599)$, $p(721)$

Calkin et al (2007): computation of $p(n)$ mod m for all $n \leq 10^9$ and primes $m \leq 103$

Various (circa 2009): $p(n)$, $n \approx 10^9$ (approximately 1 minute in Sage or Mathematica; implementation in Sage by Jonathan Bober)

This talk (2011): new implementation in FLINT

Euler's pentagonal number theorem

$$\sum_{n=0}^{\infty} p(n)x^n = \prod_{k=1}^{\infty} \frac{1}{1-x^k} = \left(\sum_{k=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2} \right)^{-1}$$

Recursive formula:

$$p(n) = \sum_{k=1}^{n} (-1)^{k+1} \left(p\left(n - \frac{k(3k-1)}{2}\right) + p\left(n - \frac{k(3k+1)}{2}\right) \right)$$

$O(n^{3/2})$ integer operations, $O(n^2)$ bit operations
$O(n^{3/2})$ bit operations modulo a fixed prime m

Asymptotically fast vector computation

Use fast power series arithmetic (FFT multiplication + Newton iteration) to compute $1/f(x) = p_0 + p_1 x + \ldots + p_n x^n$

$O(n^{3/2+o(1)})$ bit operations over $\mathbb{Z}$

$O(n^{1+o(1)})$ bit operations over $\mathbb{Z}/m\mathbb{Z}$ for fixed m

The complexity in both cases is essentially optimal

How quickly can we compute a single coefficient?

For some combinatorial sequences

$$f(x) = \sum_{k=0}^{\infty} c_k x^k$$

we know how to compute c_n in essentially $O(\log c_n)$ time.

Example: Fibonacci numbers, binomial coefficients, holonomic sequences

For others, no way is known to compute c_n that is essentially faster than computing all values $c_0, c_1, \ldots, c_n$ ($O(n \log c_n)$ time, for appropriate $f(x)$)

Example: Bernoulli numbers, Stirling numbers, Bell numbers ...

The Hardy-Ramanujan-Rademacher formula

Hardy and Ramanujan (1917), Rademacher (1936)

$$p(n) = \frac{1}{\pi\sqrt{2}} \sum_{k=1}^{\infty} \sqrt{k}\, A_k(n)\, \frac{d}{dn} \left(\frac{1}{\sqrt{n - \frac{1}{24}}} \sinh \left[\frac{\pi}{k} \sqrt{\frac{2}{3}\left(n - \frac{1}{24}\right)} \right] \right)$$

$$A_k(n) = \sum_{0 \le h < k;\ \gcd(h,k)=1} e^{\pi i \left[s(h,k) - \frac{1}{k} 2nh \right]}$$

$$s(h,k) = \sum_{i=1}^{k-1} \frac{i}{k} \left(\frac{hi}{k} - \left\lfloor \frac{hi}{k} \right\rfloor - \frac{1}{2} \right)$$

Rademacher's error bound

The remainder after N terms satisfies $|R(n,N)| < M(n,N)$ where

$$M(n,N) = \frac{44\pi^2}{225\sqrt{3}} N^{-1/2} + \frac{\pi\sqrt{2}}{75}\left(\frac{N}{n-1}\right)^{1/2} \sinh\left(\frac{\pi}{N}\sqrt{\frac{2n}{3}}\right)$$

$M(n, cn^{1/2}) \sim n^{-1/4}$ for every positive c

Algorithm: take $N = O(n^{1/2})$ such that $|R(n,N)| + |\varepsilon| < 0.5$ where ε is the numerical error, then round the sum to the nearest integer.

Complexity of numerical evaluation

Odlyzko (1995): the HRR formula "gives an algorithm for calculating $p(n)$ that is close to optimal, since the number of bit operations is not much larger than the number of bits of $p(n)$".

But no further details (let alone concrete algorithms) are to be found in the literature.

How well can we do in practice? There are many implementations (Mathematica, Sage, Pari/GP, ...), but their performance differs by orders of magnitude.

Checking Odlyzko's claim

We need $O(n^{1/2})$ terms calculated to a precision of $b = O(n^{1/2}/k + \log n)$ bits to determine $p(n)$.

Assume that we can evaluate each term with bit complexity $O(b \log^c b)$. Then the total cost is

$$\sum_{k=1}^{n^{1/2}} \left(\frac{n^{1/2}}{k}\right) \log^c \left(\frac{n^{1/2}}{k}\right) \sim n^{1/2} \log^{c+1} n$$

Multiplication: $M(b) = O(b \log b \log\log b)$

Elementary functions (binary splitting): $O(M(b) \log^2 b)$

Elementary functions (arithmetic-geometric mean): $O(M(b) \log b)$

Dedekind sums

The "inner loop" consists of the Dedekind sum

$$s(h,k) = \sum_{i=1}^{k-1} \frac{i}{k}\left(\frac{hi}{k} - \left\lfloor \frac{hi}{k} \right\rfloor - \frac{1}{2}\right)$$

Naive algorithm: $O(k)$ integer or rational operations

$O(k^2)$ integer operations to evaluate $A_k(n)$

$O(n^{3/2})$ integer operations to evaluate $p(n)$

To compute $p(n)$ in optimal time (up to log factors), we must get $A_k(n)$ down to $O(\log^c k)$ function evaluations and integer operations

Fast computation of Dedekind sums

Let $0 < h < k$ and let $k = r_0, r_1, \ldots, r_{m+1} = 1$ be the sequence of remainders in the Euclidean algorithm for $\gcd(h, k)$. Then

$$s(h,k) = \frac{(-1)^{m+1} - 1}{8} + \frac{1}{12} \sum_{j=1}^{m+1} (-1)^{j+1} \frac{r_j^2 + r_{j-1}^2 + 1}{r_j r_{j-1}}.$$

Can be implemented directly or with the fraction-free algorithm of Knuth (1975).

$O(\log k)$ integer or rational operations to evaluate $s(h, k)$

$O(k \log k)$ integer operations to evaluate $A_k(n)$

$O(n \log n)$ integer operations to evaluate $p(n)$

Evaluating $A_k(n)$ without Dedekind sums

A formula due to Selberg gives a faster (and remarkably **simple**) algorithm.

$$A_k(n) = \left(\frac{k}{3}\right)^{1/2} \sum_{(3\ell^2+\ell)/2 \equiv -n \bmod k} (-1)^\ell \cos\left(\frac{6\ell+1}{6k}\pi\right)$$

The index ranges over $0 \le \ell < 2k$ ($O(k^{1/2})$ terms are nonzero)

Brute force is efficient: testing whether ℓ solves the quadratic equation is much cheaper than evaluating a Dedekind sum

The cost for $p(n)$ is still $O(n)$ integer operations and $O(n^{3/4})$ cosines

Evaluating $A_k(n)$ using prime factorization

Whiteman (1956): if $k = p^e$, then

$$A_k(n) = \sqrt{\frac{s}{t}} \cos\left(\frac{\pi r}{24k}\right)$$

where $r, s, t \in \mathbb{Z}$ are determined by certain quadratic modular equations, GCD and Jacobi symbol computations.

If $k = k_1 k_2$ where $\gcd(k_1, k_2) = 1$, then $A_k(n) = A_{k_1}(n_1) A_{k_2}(n_2)$ where n_1 and n_2 are solutions of modular equations.

Algorithm: factor k into prime powers to write $A_k(n)$ as a product of $O(\log k)$ cosines.

Cost of modular arithmetic

We can factor all k in time $O(n^{1/2} \log n)$.

A single k has at most $\log k$ prime factors

Jacobi symbol multiplication, inverse, etc. mod k: $O(\log^{1+o(1)} k)$

Square roots mod p: $O(\log^{3+o(1)} p)$ using the Shanks-Tonelli algorithm, $O(\log^{2+o(1)} p)$ using Cipolla's algorithm (assuming the Extended Riemann Hypothesis!)

Cost of modular arithmetic for each $A_k(n)$: $O(\log^{3+o(1)} k)$ (assuming ERH)

Practical implementation of modular arithmetic

FLINT provides highly optimized routines for operations on integers up to 32/64 bits

Division, modular inverse, GCD, primality testing, Jacobi symbol, square root, etc

Integer factorization (using trial division with precomputation plus Hart's new "One Line Factor" algorithm) accounts for less than 1% of the total time, so there is no need for sieving

Numerical evaluation

We use arbitrary-precision floating-point arithmetic to evaluate

$$t_k = \frac{\alpha}{\beta}\frac{\sqrt{a}}{\sqrt{b}}\, U\left(\frac{C}{k}\right) \prod_{i=1}^{m} \cos\left(\frac{p_i \pi}{q_i}\right)$$

where $U(x) = \cosh x - \dfrac{\sinh x}{x}$

Choose N such that $R(n, N) < 0.25$, compute $\hat{t}_k$ such that $|\hat{t}_k - t_k| \leq 0.125/N$ and with errors in the summation bounded by $0.125/N$. Then $|\hat{p}(n) - p(n)| < 0.5$.

Implementation: MPFR (+MPIR), custom code

Faster high-precision numerical evaluation

For small k, compute $\exp(C/k)$ as $(\exp(C))^{1/k}$

For small q, compute $\cos(p\pi/q)$ using numerical Newton iteration to solve $P(\cos(2\pi/n)) = 0$ where $P(x)$ is an integer polynomial of degree $\phi(n)/2$.

P can be generated numerically (balanced product) or symbolically (repeated decomposition into Chebyshev polynomials)

Alternative: use $x^n - 1$ (complex arithmetic)

Faster low-precision numerical evaluation

Most of the terms in the HRR sum are small (close to unit magnitude)

Use hardware double precision arithmetic when the needed precision falls below 53 bits

We need to make assumptions about the accuracy of system transcendental functions. Range reducing $x = p\pi/q$ to $(0, \pi/4)$ avoids potential catastrophic error in trigonometric functions.

Reliability of numerical evaluation

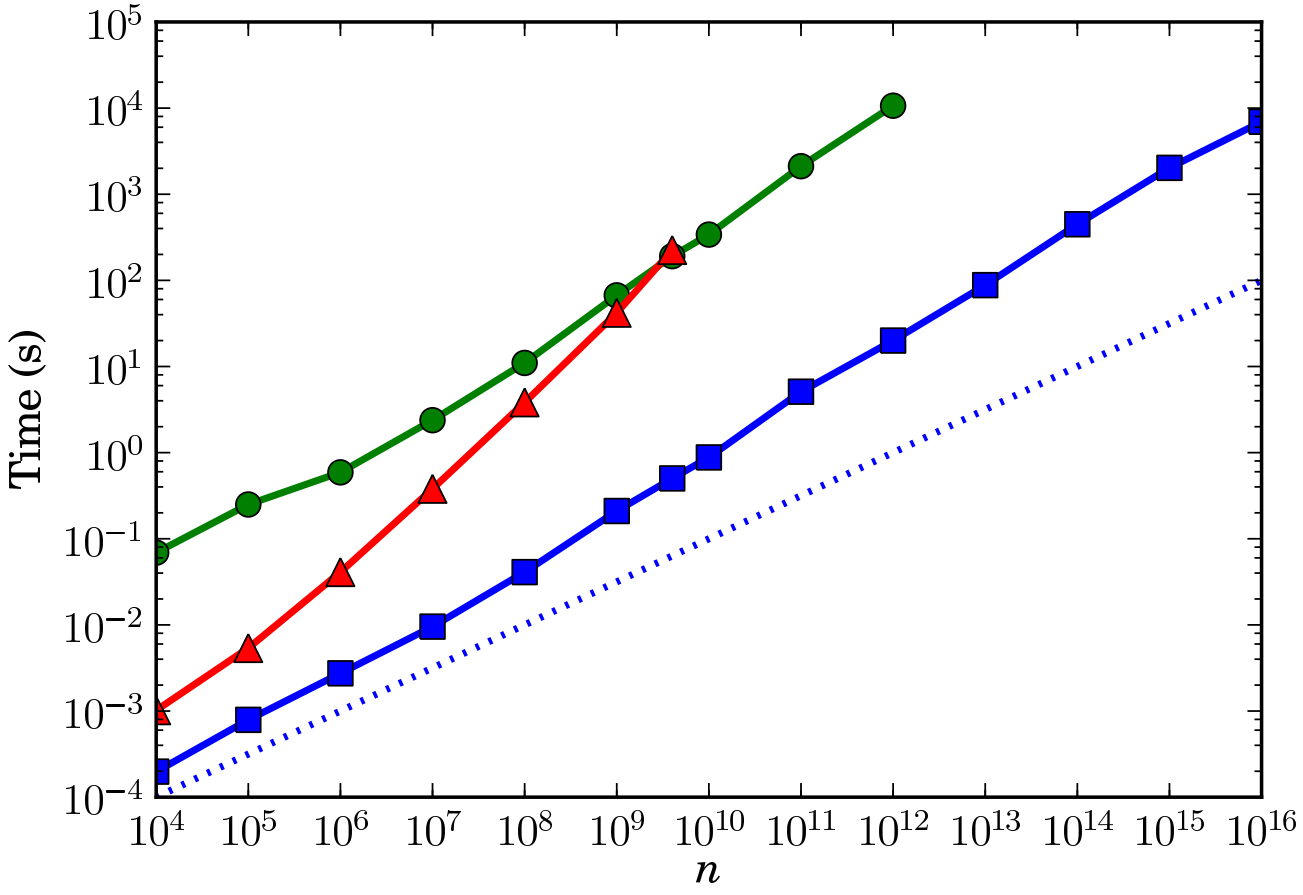
Bugs have been found in the HRR implementations in both Pari/GP and Sage (due to using insufficient precision)

Each term in our implementation is evaluated as a product, so it is numerically well-behaved

Rigorous error bounds proved in paper

MPFR provides guaranteed high-precision numerical evaluation (but at present, we also rely on heuristic custom functions)

Computing $p(n)$ with FLINT, Mathematica, Sage



FLINT (blue squares), Mathematica 7 (green circles), Sage 4.7 (red triangles). Dotted line: $t = 10^{-6} n^{1/2}$.

Timings

n	Mathematica 7	Sage 4.7	FLINT	Initial
10^4	69 ms	1 ms	0.20 ms	
10^5	250 ms	5.4 ms	0.80 ms	
10^6	590 ms	41 ms	2.74 ms	
10^7	2.4 s	0.38 s	0.010 s	
10^8	11 s	3.8 s	0.041 s	
10^9	67 s	42 s	0.21 s	43%
10^{10}	340 s		0.88 s	53%
10^{11}	2,116 s		5.1 s	48%
10^{12}	10,660 s		20 s	49%
10^{13}			88 s	48%
10^{14}			448 s	47%
10^{15}			2,024 s	39%
10^{16}			6,941 s	45%
10^{17}			27,196* s	33%
10^{18}			87,223* s	38%
10^{19}			350,172* s	39%

Near-optimality (in practice)

Computing the first term (basically π and a single $\exp(x)$) takes nearly half the total running time

$\Rightarrow$ Improving the tail of the HRR sum further can give at most a twofold speedup

$\Rightarrow$ Parallelization of the HRR sum can give at most a twofold speedup

For a larger speedup, we would need faster high-precision transcendental functions (for example, using parallelization at the level of computing exp, or even in single multiplications)

Large values of $p(n)$

n	Decimal expansion	Num. digits	Terms	Error
10^{12}	6129000962...6867626906	1,113,996	264,526	10^{-7}
10^{13}	5714414687...4630811575	3,522,791	787,010	10^{-8}
10^{14}	2750960597...5564896497	11,140,072	2,350,465	10^{-8}
10^{15}	1365537729...3764670692	35,228,031	7,043,140	10^{-9}
10^{16}	9129131390...3100706231	111,400,846	21,166,305	10^{-9}
10^{17}	8291300791...3197824756	352,280,442	63,775,038	10^{-9}
10^{18}	1478700310...1701612189	1,114,008,610	192,605,341	10^{-10}
10^{19}	5646928403...3674631046	3,522,804,578	582,909,398	10^{-11}

The number of partitions of ten quintillion:
$p(10^{19}) = p(10000000000000000000) \approx 5.65 \times 10^{3,522,804,577}$

3.5 GB output, 97 CPU hours, $\sim$ 150 GB memory

Combinatorial interpretations

$p(10^{15})$: the number of different ways the United States national debt ($\approx \$10^{13}$) can be paid off in bags of cents

$p(10^{19})$: the number of ways to form a collection of sticks whose lengths are multiples of 1 m, such that they add up to the thickness of the Milky Way galaxy ($\approx 10^{19}$ m) when placed end to end

Finding congruences

Ramanujan (1919):

$$p(5k+4) \equiv 0 \pmod 5$$
$$p(7k+5) \equiv 0 \pmod 7$$
$$p(11k+6) \equiv 0 \pmod{11}$$

Ono (2000): for every prime $m \geq 5$, there exist infinitely many congruences of the type

$$p(Ak+B) \equiv 0 \bmod m$$

A constructive, computational procedure for finding such (A, B, m) with $13 \leq m \leq 31$ was discovered by Weaver (2001)

Theorem (Weaver)

Let $m \in \{13, 17, 19, 23, 29, 31\}$, $\ell \geq 5$ a prime, $\varepsilon \in \{-1, 0, 1\}$. If (m, ℓ, ε) satisfies a certain property, then (A, B, m) is a partition function congruence where

$$A = m\ell^{4-|\varepsilon|}$$

$$B = \frac{m\ell^{3-|\varepsilon|}\alpha + 1}{24} + m\ell^{3-|\varepsilon|}\delta,$$

where α is the unique solution of $m\ell^{3-|\varepsilon|}\alpha \equiv -1 \bmod 24$ with $1 \leq \alpha < 24$, and where $0 \leq \delta < \ell$ is any solution of

$$\begin{cases} 24\delta \not\equiv -\alpha \bmod \ell & \text{if } \varepsilon = 0 \\ (24\delta + \alpha \mid \ell) = \varepsilon & \text{if } \varepsilon = \pm 1. \end{cases}$$

The free choice of δ gives $\ell - 1$ distinct congruences for a given tuple (m, ℓ, ε) if $\varepsilon = 0$, and $(\ell - 1)/2$ congruences if $\varepsilon = \pm 1$.

Weaver's algorithm

Input: A pair of prime numbers $13 \leq m \leq 31$ and $\ell \geq 5$, $m \neq \ell$
Output: (m, ℓ, ε) defining a congruence, or Not-a-congruence

$\delta_m \leftarrow 24^{-1} \bmod m$ {Reduced to $0 \leq \delta_m < m$}
$r_m \leftarrow (-m) \bmod 24$ {Reduced to $0 \leq m < 24$}
$v \leftarrow \frac{m-3}{2}$
$x \leftarrow p(\delta_m)$ {We have $x \not\equiv 0 \bmod m$}
$y \leftarrow p\left(m\left(\frac{r_m(\ell^2-1)}{24}\right) + \delta_m\right)$
$f \leftarrow (3 \mid \ell)\,((-1)^v r_m \mid \ell)$ {Jacobi symbols}
$t \leftarrow y + fx\ell^{v-1}$
if $t \equiv \omega \bmod m$ where $\omega \in \{-1, 0, 1\}$ **then**
  return $(m, \ell, \omega\,(3(-1)^v \mid \ell))$
else
  return Not-a-congruence
end if

Weaver's table

Weaver gives 76,065 congruences (167 tuples), obtained from a table of all $p(n)$ with $n < 7.5 \times 10^6$ (computed using the recursive Euler algorithm).

Limit on $\ell \approx 10^3$

Example: $m = 31$

$\varepsilon = 0$: $\ell = 107, 229, 283, 383, 463$

$\varepsilon \neq 0$: $(\ell, \varepsilon) = (101, 1), (179, 1), (181, 1), (193, 1), (239, 1), (271, 1)$

New table

Testing all $\ell < 10^6$ resulted in 22 billion new congruences (70,359 tuples).

This involved evaluating $p(n)$ for $6(\pi(10^6) - 3) = 470,970$ distinct n, in parallel on ≈ 40 cores (hardware at University of Warwick, courtesy of Bill Hart)

m	$\varepsilon = 0$	$\varepsilon = +1$	$\varepsilon = -1$	Congruences	CPU	Max n
13	6,189	6,000	6,132	5,857,728,831	448 h	5.9×10^{12}
17	4,611	4,611	4,615	4,443,031,844	391 h	4.9×10^{12}
19	4,114	4,153	4,152	3,966,125,921	370 h	3.9×10^{12}
23	3,354	3,342	3,461	3,241,703,585	125 h	9.5×10^{11}
29	2,680	2,777	2,734	2,629,279,740	1,155 h	2.2×10^{13}
31	2,428	2,484	2,522	2,336,738,093	972 h	2.1×10^{13}
All	23,376	23,367	23,616	22,474,608,014	3,461 h	

Examples of new congruences

Example 1: $(13, 3797, -1)$ with $\delta = 2588$ gives

$$p(711647853449k + 485138482133) \equiv 0 \bmod 13$$

which we easily evaluate for all $k \leq 100$.

Example 2: $(29, 999959, 0)$ with $\delta = 999958$ gives

$$p(28995244292486005245947069k + 28995221336976431135321047) \equiv 0 \bmod 29$$

This is out of reach for explicit evaluation ($n \approx 10^{25}$)

Download the data

`http://www.risc.jku.at/people/fjohanss/partitions/`

or

`http://sage.math.washington.edu/home/fredrik/partitions/`

Comparison of algorithms for vector computation

n	Series ($\mathbb{Z}/13\mathbb{Z}$)	Series ($\mathbb{Z}$)	HRR (all)	HRR (sparse)
10^4	0.01 s	0.1 s	1.4 s	0.001 s
10^5	0.13 s	4.1 s	41 s	0.008 s
10^6	1.4 s	183 s	1430 s	0.08 s
10^7	14 s			0.7 s
10^8	173 s			8 s
10^9	2507 s			85 s

HRR competitive over $\mathbb{Z}$: when n/c values are needed (our improvement: $c \approx 10$ vs $c \approx 1000$)

HRR competitive over $\mathbb{Z}/m\mathbb{Z}$: when $O(n^{1/2})$ values are needed (speedup for Weaver's algorithm: 1-2 orders of magnitude).

Most important advantages: little memory, parallel, resumable

Conclusions

The HRR formula allows performing computations that are impractical with power series methods

We can compute $p(n)$ with nearly optimal complexity in both theory and practice

This requires careful attention to asymptotics (otherwise "cheap" operations might start to dominate when n is larger than perhaps 10^9) as well as implementation details

Possible generalization to other HRR-type series for special types of partitions (into distinct parts, etc)