Computing Roots in Finite Fields and Groups, with a Jaunt through Sums of Digits

By

Scott Charles Lindhurst

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
(Mathematics)

at the
UNIVERSITY OF WISCONSIN—MADISON
1997

Acknowledgements

I thank my advisor, Eric Bach, for suggesting these problems to me and for many helpful discussions and references.

This work was supported financially in part by the U.S. National Science Foundation grant CCR-9510244.

Notation

We use some notation that is becoming common among computer scientists, but may still be unfamiliar to some mathematicians.

Notation	Description
$\Theta(f)$	Similar to $O(f)$, but denotes an exact order of growth instead of an upper bound. Specifically, $g = \Theta(f)$ means $g = O(f)$ and $f = O(g)$.
$\lfloor x \rfloor$	The floor of x is sometimes also written $[x]$, and is the greatest integer less than or equal to x. For example, $\lfloor -3.1 \rfloor = -4$ and $\lfloor 7 \rfloor = 7$.
$\lceil x \rceil$	The ceiling of x is the least integer greater than or equal to x. For example, $\lceil -3.1 \rceil = -3$ and $\lceil 7 \rceil = 7$.
$m \bmod n$	Remainder operation—the least nonnegative residue of m modulo n. Thus $0 \le m \bmod n \le n - 1$.
$\lg x$	Base 2 logarithm; $\lg x = \log_2 x$.

Contents

Chapter 1

Introduction

Every schoolchild learns to compute square roots of integers. But what about square roots in the integers modulo a prime number p (that is, solving the equation $x^2 \equiv a \bmod p$)? In 1891, Alberto Tonelli [35] published what was probably the first efficient algorithm to compute square roots mod p. In succeeding years, many more algorithms were published; Bach and Shallit [2, pp. 194–195] give references to several. Daniel Shanks [29, section 5] improved on Tonelli's algorithm in 1972. Shanks called his algorithm RESSOL, for "residue solver," but today it's often called "Shanks's algorithm." Many other authors [2, 5] have described either Tonelli's or Shanks's square root algorithm. The most detailed description I have seen of the group theory behind the algorithm is in Niven, Zuckerman, and Montgomery's elementary number theory textbook [23, pp. 110–114].

Why are square roots important? Of course, we may simply be interested in the problem because it's there, but there are also applications to cryptography. Rabin's public-key encryption scheme [20, section 8.3] was the first provably secure public-key encryption scheme. In Rabin's scheme, decryption requires computing two square roots modulo two different prime numbers, so an implementor needs to know an efficient method of computing square roots. There are also other public-key encryption schemes and digital signature schemes [10, 18] that require computing square roots.

Now that we know we want to compute square roots mod p, how long will it take? We begin in chapter 2 by calculating the average work (average number of operations mod p) used by Shanks's algorithm. We obtain exact average results for the expected number of operations for a fixed prime and also averaged over all prime numbers. We compute the second moment and worst case to demonstrate that the average case is also typical. That is, no specific input uses more than twice the average number of operations. As a cap to the analysis, we show that the limiting distribution is essentially normal.

Shanks stated his algorithm only for *square* roots mod p. The algorithm of Adleman, Manders, and Miller [1] is very similar to Shanks's and Tonelli's algorithms. They extended their algorithm to compute roots of higher degree, and Shanks's algorithm also extends to higher roots. Adleman, Manders, and Miller restricted themselves to roots of prime degree, for example, computing a sixth root by computing a square root then a cube root. When we examine generalizations of Shanks's algorithm in chapter 3, we shall see that it is sometimes possible to compute the sixth root directly.

It turns out that both the square root algorithm and its higher-degree generalizations use only the multiplicative structure of $\mathbf{Z}/p\mathbf{Z}$—all of the calculations occur in the multiplicative group $(\mathbf{Z}/p\mathbf{Z})^\times$. A natural question to ask is: In which other groups does the algorithm work? The class of such groups includes not only the multiplicative groups of finite fields, but also some non-abelian groups. For the square root case, we characterize such groups.

One of the steps of Shanks's algorithm is computing a power mod p. When the exponentiation is done using the binary power method [16], the number of modular multiplications used depends upon the number of ones in p's binary representation. This leads us to consider numbers of ones in binary integers, or more generally, sums of digits of numbers in other bases. Katai [15] found asymptotic results for digit sums of the prime numbers up to some bound. In chapter 4, we consider a similar problem that has also been studied, but with fewer results: sums of digits of numbers in arithmetic progressions.

How many numbers up to x in a given arithmetic progression have digits that sum to k? We consider generating functions for this quantity. Our main result is a generating function recurrence that can be used with many arithmetic progressions and bases. The recurrence lets us easily find the generating function when x is a power of the base. We obtain exact values for excesses of even sums of digits over odd sums among multiples of 3 and 5. Our technique applies to computing similar quantities with many different bases and arithmetic sequences.

Chapter 2

Exact Analysis of Shanks's Square Root Algorithm

2.1 Introduction and Notation

As we noted in the introduction, Shanks's algorithm computes square roots in the finite field $\mathbf{Z}/p\mathbf{Z}$, where p is a prime number. In other words, if a is a square mod p, the algorithm solves the equation $x^2 \equiv a \bmod p$.

The algorithm has two parts, the *initialization* and the *body*. We will see that the number of multiplications mod p used by the initialization is almost completely independent of the input, so initially we concentrate on counting the number of multiplications mod p in the body, then later remark briefly on the initialization.

The number of multiplications used depends strongly on which powers of 2 divide $p-1$. Because computing square roots in $\mathbf{Z}/2\mathbf{Z}$ is trivial, assume that p is odd. Factor out as many factors of 2 as possible from $p-1$ to express it as $p-1 = 2^n q$, where q is odd (we call n the *order* of p). Cohen's description of the algorithm [5, p. 33] mentions "It is easy to show that, on average," the body requires $n^2/4$ multiplications mod p. Among other things, we prove he is asymptotically correct by giving an exact value for the average number of multiplications.

If the input a is a uniformly chosen random quadratic residue, we show that (excluding the initialization) Shanks's algorithm uses on average $\frac{1}{4}(n^2+7n-12)+1/2^{n-1}$ multiplications in $\mathbf{Z}/p\mathbf{Z}$ for a fixed prime of order n, and $8/3$ multiplications averaged over all primes. The second moment of the distribution (again excluding the initialization) is $\frac{1}{48}(3n^4+46n^3+93n^2-478n+864)-(4n+18)/2^n$ for a fixed prime of order n and $36\frac{2}{9}$ averaged over all primes.

Next, we show that as the order n goes to infinity, the limiting distribution of the number of multiplications is normal with mean asymptotic to $n^2/4$ and variance $\Theta(n^{3/2})$. As with the other results, this excludes the initialization. In this case, it also excludes an asymptotically negligible term whose expected value is less than 4.

Now recall a few facts about the finite field $\mathbf{Z}/p\mathbf{Z}$, which is where the algorithm's calculations all take place. The multiplicative group $(\mathbf{Z}/p\mathbf{Z})^\times$ of $\mathbf{Z}/p\mathbf{Z}$ is a cyclic group of order $2^n q$, hence its Sylow 2-subgroup S_n is cyclic of order 2^n. There is a descending chain of subgroups of S_n,

$$S_n \supset S_{n-1} \supset \cdots \supset S_2 \supset S_1 \supset S_0 = \{1\}.$$

If we have a quadratic non-residue u, then S_n is generated by $z = u^q$ and has order 2^n, S_{n-1} is generated by z^2 and has order 2^{n-1}, and in general, S_{n-i} is generated by z^{2^i} and has order 2^{n-i} for $i = 0, 1, 2, \ldots, n$.

2.2 Shanks's Algorithm

The algorithm takes as input an odd prime, p, which we write as $p = 2^n q + 1$, where q is odd. The other two inputs are a quadratic residue a and a nonresidue u, both mod p; that is, the Jacobi symbols are $(a/p) = +1$ and $(u/p) = -1$. Shanks's algorithm finds an element x satisfying $x^2 \equiv a \bmod p$. All of the calculations take place in the multiplicative group $(\mathbf{Z}/p\mathbf{Z})^\times$.

The algorithm first finds an index k and group elements b, z, and x satisfying $b \in S_{k-1}$, z generates S_k, and $x^2 = ab$. Then at each step it modifies k, b, z, and x so that these relations still hold and the index k decreases. Since initially $k = n$ and later $k \geq 1$, after no more than $n - 1$ modifications we have $k = 1$ and $b \in S_0 = \{1\}$. Then $x^2 = a \cdot 1 = a$, and we have our solution.

Shanks's Algorithm for square roots mod p

Given $p = 2^n q + 1$, $\left(\frac{a}{p}\right) = +1$, $\left(\frac{u}{p}\right) = -1$

Initialization:

Let $k = n$, $z = u^q$, $x = a^{(q+1)/2}$, $b = a^q$

Body (loop invariants: $x^2 = ab$, z generates S_k, $b \in S_{k-1}$):

while $b \neq 1$ do

    Find least positive integer m such that $b^{2^m} = 1$ {*}

    Update $t = z^{2^{k-m-1}}$, {**}

        $z = t^2$

        $b = bz$

        $x = xt$

        $k = m$

After the initialization, the loop invariants are satisfied: because a is a quadratic residue, b is a square in S_n and so is in S_{n-1}. As we remarked in the previous section, $z = u^q$ generates S_n because u is a quadratic nonresidue. The initialization can be completed with two exponentiations and two multiplications by successively computing $z := u^q$, $t := a^{(q-1)/2}$, $x := at$, $b := xt$.

The first line in the loop (labeled *) figures out exactly where b is in the chain of 2-subgroups, i.e., what value of m satisfies $b \in S_m \setminus S_{m-1}$. This may be done in m multiplications by successively squaring b until we get 1 because if $y \in S_l \setminus S_{l-1}$, then $y^2 \in S_{l-1} \setminus S_{l-2}$.

The rest of the loop (labeled **) pushes b down into a strictly smaller subgroup $S_l \subset S_m$ and preserves the invariants. The element t generates S_{m+1} and z generates

S_m. The new value of b is a nonsquare of S_m multiplied by another nonsquare of S_m (in fact, by a generator) so it's a square in S_m and hence $b \in S_{m-1}$. This part of the loop uses $k-m-1+3 = k-m+2$ multiplications, so one iteration of the whole loop uses $k+2$ multiplications. Notice that the time for one iteration depends only on the value of k at the beginning of the iteration.

Before loop		After loop
z generates	S_k	
	$\cup$	
	$\vdots$	
	S_{m+1}	t generates
	$\cup$	
$b \in$	S_m	z generates
	$\cup$	
$b \notin$	S_{m-1}	$\ni b$

2.3 Examples of Shanks's Algorithm

Example 1. *Find the square root of* $2 \bmod p = 95231 = 2^1 \cdot 47615 + 1$, *using the nonresidue 11.* We have $p = 95231$, $n = 1$, $q = 47615$, $a = 2$, and $u = 11$. In the initialization, we compute

$$\begin{aligned} z &:= u^q = 11^{47615} = -1 \\ t &:= a^{(q-1)/2} = 2^{23807} = 16734 \\ x &:= at = 2 \cdot 16734 = 33468 \\ b &:= xt = 33468 \cdot 16734 = 1 \\ k &:= n = 1. \end{aligned}$$

Since $b = 1$, the solution is $x = 33468$. The other solution to $x^2 \equiv 2 \bmod 95231$ is $-x = 61763$.

Example 2. *Find the square root of* $2 \bmod p = 95233 = 2^{10} \cdot 93 + 1$, *using the nonresidue 5.* We have $p = 95233$, $n = 10$, $q = 93$, $a = 2$, and $u = 5$ and compute the initialization:

$$\begin{aligned} z &:= u^q = 5^{93} = 75817, \\ t &:= a^{(q-1)/2} = 2^{46} = 59279, \\ x &:= at = 2 \cdot 59279 = 23325, \\ b &:= xt = 23325 \cdot 59279 = 89981, \\ k &:= n = 10. \end{aligned}$$

In the first iteration of the loop, we must square b nine times before getting 1, so $m = 9$ and $b \in S_9 \setminus S_8$.

$$\begin{array}{lll} b^{2^1} = 89981^2 = 61167 & b^{2^2} = 61167^2 = 78251 & b^{2^3} = 78251^2 = 22800 \\ b^{2^4} = 22800^2 = 58286 & b^{2^5} = 58286^2 = 10957 & b^{2^6} = 10957^2 = 53958 \\ b^{2^7} = 53958^2 = 2488 & b^{2^8} = 2488^2 = 95232 = -1 & b^{2^9} = 95232^2 = 1. \end{array}$$

$$
\begin{aligned}
t &:= z^{2^{k-m-1}} = 75817^{2^0} = 75817, \\
z &:= t^2 = 75817^2 = 48842, \\
b &:= bz = 89981 \cdot 48842 = 39518, \\
x &:= xt = 23325 \cdot 75817 = 49948, \\
k &:= m = 9.
\end{aligned}
$$

After the first iteration, z generates $S_k = S_9$ and we can verify directly that $x^2 = ab$. In the second iteration of the loop, we square b six times ($m = 6$) to see that $b \in S_6 \setminus S_5$, then compute:

$$
\begin{aligned}
t &:= z^{2^{9-6-1}} = 48842^4 = 84168, \\
z &:= t^2 = 59820, \\
b &:= bz = 93234, \\
x &:= xt = 49948 \cdot 84168 = 57712, \\
k &:= m = 6.
\end{aligned}
$$

During the third iteration of the loop, we compute $b^{2^5} = 1$, so $m = 5$. The updates for the other variables are

$$
\begin{aligned}
t &:= z^1 = 59820, \\
z &:= t^2 = 52425, \\
b &:= bz = 93234 \cdot 52425 = 53958, \\
x &:= xt = 57712 \cdot 59820 = 40357, \\
k &:= m = 5.
\end{aligned}
$$

In the fourth and last iteration, we find $b^8 = 1$, so $m = 3$.

$$
\begin{aligned}
t &:= z^2 = 51478, \\
z &:= t^2 = 31026, \\
b &:= bz = 53958 \cdot 31026 = 1, \\
x &:= xt = 40357 \cdot 51478 = 84984, \\
k &:= m = 3.
\end{aligned}
$$

Now $b = 1$, so we are done. One square root of 2 mod 95233 is 84984.

2.4 Analysis of the Algorithm

If we assume that a and u are random (that is, uniformly distributed) elements of $(\mathbf{Z}/p\mathbf{Z})^\times$ subject to $(a/p) = +1$ and $(u/p) = -1$, then z is a random generator of S_n

and $b = a^q$ is a random square in S_n, hence a random element of S_{n-1}. We compute that

$$\Pr(b \in S_m \setminus S_{m-1}) = \frac{|S_m| - |S_{m-1}|}{|S_{n-1}|} = \frac{2^m - 2^{m-1}}{2^{n-1}} = \frac{1}{2^{n-m}}$$

$$\Pr(b \in S_0) = \frac{|S_0|}{|S_{n-1}|} = \frac{1}{2^{n-1}}.$$

The quantities b and z are random at the start of one loop iteration, so at the end of the iteration the new values of b and z are a random square in S_m and a random generator of S_m, respectively.

Now consider the sequence of values that the index k takes in the algorithm (which are also the subgroups that b is actually in, $b \in S_k \setminus S_{k-1}$, except for the first value, $k = n$). We begin with $k_0 = n$, then continue with values $k_1, k_2, \ldots, k_r$. Because the values of b are random, the probability of a particular sequence is

$$\frac{1}{2^{n-k_1}} \frac{1}{2^{k_1-k_2}} \cdots \frac{1}{2^{k_{r-1}-k_r}} \frac{1}{2^{k_r-1}} = \frac{1}{2^{n-1}}.$$

Therefore every sequence has the same probability!

That suggests our strategy for analyzing the algorithm. We count the total number of multiplications for all sequences, C_n, then divide by the number of sequences, 2^{n-1}, to get the average. To find the total, we need to know how many multiplications a particular sequence $k_0, k_1, k_2, \ldots, k_r$ requires. The last value k_r contributes no multiplications, but the others contribute $k_i + 2$ each. So altogether, the body of the algorithm uses $(n+2) + (k_1 + 2) + \cdots + (k_{r-1} + 2)$ multiplications.

Here are the possible sequences for the first few orders n.

Sequence of indices	Multiplications	Total
Order 1		
1	0	$C_1 = 0$
Order 2		
2	0	
$2, 1$	$2 + 2 = 4$	$C_2 = 4$
Order 3		
3	0	
$3, 1$	$3 + 2 = 5$	
$3, 2$	$3 + 2 = 5$	
$3, 2, 1$	$(3 + 2) + (2 + 2) = 9$	$C_3 = 19$
Order 4		
4	0	
$4, 1$	$4 + 2 = 6$	
$4, 2$	$4 + 2 = 6$	
$4, 2, 1$	$(4 + 2) + (2 + 2) = 10$	
$4, 3$	$4 + 2 = 6$	
$4, 3, 1$	$(4 + 2) + (3 + 2) = 11$	
$4, 3, 2$	$(4 + 2) + (3 + 2) = 11$	
$4, 3, 2, 1$	$(4 + 2) + (3 + 2) + (2 + 2) = 15$	$C_4 = 65$

For order n, there are 2^{n-1} sequences altogether, taking a total of C_n multiplications:

Sequence	Multiplications	Total
n	0	
$n, 1$	$n + 2$	
$n, 2$	$n + 2$	
$\vdots$	$\vdots$	
$n, n - 1, \ldots, 3, 2, 1$	$(n + 2) + (n + 1) + \cdots + 5 + 4$	C_n

The 2^n sequences for order $n + 1$ can be divided into two types: those in which k skips the value n and those in which k takes the value n. The 2^{n-1} sequences of the first type are obtained by copying the sequences from order n, changing n to $n + 1$. All except the first take one more multiplication than before, for a subtotal of $C_n + 2^{n-1} - 1$:

Sequence	Multiplications
$n + 1$	0
$n + 1, 1$	$n + 3$
$n + 1, 2$	$n + 3$
$\vdots$	$1 + \text{old value}$
$n + 1, n - 1, \ldots, 3, 2, 1$	$(n + 3) + (n + 1) + \cdots + 5 + 4$

The 2^{n-1} sequences of the second type are obtained by prefixing the sequences from order n by $n+1$. Each new sequence takes $n+3$ more multiplications than the corresponding old sequence, for a subtotal of $C_n + 2^{n-1}(n+3)$ multiplications:

Sequence	Multiplications
$n+1, n$	$(n+3)+0$
$n+1, n, 1$	$(n+3)+(n+2)$
$n+1, n, 2$	$(n+3)+(n+2)$
$\vdots$	$(n+3)+\text{old value}$
$n+1, n, \ldots, 3, 2, 1$	$(n+3)+(n+2)+\cdots+5+4$

Adding the subtotals gives the value for C_{n+1} and the recurrence

$$C_{n+1} = 2C_n + 2^{n-1}(n+4) - 1, \quad C_1 = 0,$$

which can be (somewhat tediously) solved by standard difference equation methods [3, section 7.3] to get

$$C_n = 2^{n-3}(n^2 + 7n - 12) + 1.$$

Of course, once the solution has been found, it is easily checked by induction.

Since each of the 2^{n-1} sequences is equally likely, the average number of multiplications required after the initialization is

$$M_n = \frac{C_n}{2^{n-1}} = \frac{2^{n-3}(n^2+7n-12)+1}{2^{n-1}} = \tfrac{1}{4}(n^2+7n-12) + \frac{1}{2^{n-1}}.$$

This proves our first lemma:

Lemma 1. *Averaged over all quadratic residue and non-residue inputs, and ignoring the initialization stage, the average number of modular multiplications required by Shanks's algorithm to compute a random square root mod* $p = 2^n q + 1$ *is* $\frac{1}{4}(n^2 + 7n - 12) + \frac{1}{2^{n-1}}$.

Now let's extend this to averaging over all primes as well. First write down the average over all primes less than or equal to N, then let $N \to \infty$. The average number of multiplications over all primes up to N is

$$\sum_{n=1}^{\infty} (\text{Fraction of primes up to } N \text{ with order } n) \cdot M_n.$$

Note that the sum is actually finite because the primes less than or equal to N have order $n < \log_2 N$. The primes of order n are those whose binary expansion ends in $10\ldots01$, where there are $n-1$ zeros, so these primes are the ones congruent to $1 + 2^n \bmod 2^{n+1}$. The number of them up to N is $\pi(N; 2^{n+1}, 1+2^n)$, where $\pi(x; k, l)$ is the number of prime numbers less than or equal to x and congruent to l mod k, so we can write the general term in the sum as

$$f_N(n) = \frac{\pi(N; 2^{n+1}, 1+2^n)}{\pi(N)} M_n.$$

To average over *all* primes, we need to compute the limit $\lim_{N\to\infty} \sum_{n=1}^{\infty} f_N(n)$. The natural way to evaluate this is to interchange the sum and the limit, use Dirichlet's theorem about primes in arithmetic progressions to evaluate

$$\lim_{N\to\infty} \frac{\pi(N; 2^{n+1}, 1+2^n)}{\pi(N)} = \frac{1}{\phi(2^{n+1})} = \frac{1}{2^n},$$

and sum the resulting series using formulas in section 2.5 below to get

$$\sum_{n=1}^{\infty} \frac{M_n}{2^n} = \sum_{n=1}^{\infty} \left(\frac{1}{4}\frac{n^2}{2^n} + \frac{7}{4}\frac{n}{2^n} - \frac{3}{2^n} + \frac{2}{2^{2n}} \right) = \frac{8}{3}.$$

Of course, we cannot exchange limiting processes willy-nilly; we must prove that the exchange doesn't affect the value or convergence. A similar exchange of limits in [36] was done by laboriously proving that the double limit exists, but we use an easier method.

We prove that $f_N(n)$ is bounded by a summable function $g(n)$ independent of N. Then Lebesgue's Dominated Convergence Theorem [26, Theorem 1.34] implies that the limit of the sum is the sum of the limit. To bound $f_N(n)$, we use the following proposition.

Proposition 2. *For all integers $N \geq 1$ and $n \geq 1$,*

$$\frac{\pi(N; 2^{n+1}, 1+2^n)}{\pi(N)} \leq \frac{6n}{2^n}.$$

Proof. Using estimates (10.6) and (11.7b) from [31], namely $\pi(N) > \frac{N}{\ln N}$ and $\pi(N; k, l) < \frac{3N}{\phi(k)\ln(N/k)}$ for all $N \geq 17$, we see that for $N > 2^{2n+2}$, we have $N/2^{n+1} > N^{1/2}$ so

$$\frac{\pi(N; 2^{n+1}, 1+2^n)}{\pi(N)} < \frac{3N}{\phi(2^{n+1})\ln(N/2^{n+1})}\frac{\ln N}{N} < \frac{3\ln N}{2^n \ln(N^{1/2})} = \frac{6}{2^n} \leq \frac{6n}{2^n}$$

for all $n \geq 1$. There are fewer than $N/2^n$ integers less than N of order n, much less primes, so for $N \leq 2^{2n+2}$ and all $n \geq 1$,

$$\frac{\pi(N; 2^{n+1}, 1+2^n)}{\pi(N)} < \frac{N}{2^n}\frac{\ln N}{N} \leq \frac{\ln(2^{2n+2})}{2^n} = \frac{(2n+2)\ln 2}{2^n} < \frac{6n}{2^n}.$$

∎

By the proposition, $f_N(n)$ is uniformly bounded by $g(n) = \frac{6n}{2^n} M_n$. Since g is summable (that is, $\sum_{n=1}^{\infty} g(n)$ converges), and thinking of a sum as an integral with respect to a counting measure, Lebesgue's Dominated Convergence Theorem applies to prove that $\lim \sum f_N = \sum \lim f_N$. Thus we have:

Theorem 3. *If we average over all primes less than x and all quadratic residues and non-residues, then let $x \to \infty$ (intuitively, this is the average over all possible inputs), Shanks's algorithm requires $8/3$ modular multiplications after the initialization step.*

2.5 Some Sums

The following formulas are easily derived by differentiating the power series expansion $\sum_{n=1}^{\infty} x^n = 1/(1-x)$ and are used to get closed forms for summations in this chapter (above for the expected value, below for the second moment).

$$\sum_{n=1}^{\infty} n x^n = \frac{x}{(1-x)^2} \qquad \sum_{n=1}^{\infty} \frac{n}{2^n} = \frac{1/2}{(1-1/2)^2} = 2$$

$$\sum_{n=1}^{\infty} n^2 x^n = \frac{x(x+1)}{(1-x)^3} \qquad \sum_{n=1}^{\infty} \frac{n^2}{2^n} = \frac{(1/2)(3/2)}{(1-1/2)^3} = 6$$

$$\sum_{n=1}^{\infty} n^3 x^n = \frac{x(x^2+4x+1)}{(1-x)^4} \qquad \sum_{n=1}^{\infty} \frac{n^3}{2^n} = 26$$

$$\sum_{n=1}^{\infty} n^4 x^n = \frac{x(x^3+11x^2+11x+1)}{(1-x)^5} \qquad \sum_{n=1}^{\infty} \frac{n^4}{2^n} = 150$$

2.6 What about the Initialization?

The initialization step of Shanks's algorithm requires two multiplications and two exponentiations, computing u^q and $a^{(q-1)/2}$. Using the binary method for exponentiation, the first exponentiation requires $\lfloor \lg q \rfloor - 1 + s(q)$ multiplications while the second requires $\left\lfloor \lg \frac{q-1}{2} \right\rfloor - 1 + s(\frac{q-1}{2})$. Here $s(x)$ is the number of ones in x's binary representation (sometimes called the *Hamming weight* of x) and $\lg x$ is an abbreviation for $\log_2 x$, the binary logarithm. Because $(q-1)/2$ is obtained by removing the binary 1 from the end of q, $s((q-1)/2) = s(q) - 1$ and $\left\lfloor \lg \frac{q-1}{2} \right\rfloor = \lfloor \lg q \rfloor - 1$. Putting these together with the other two multiplications, the number of multiplications required for the initialization is exactly $2 \lfloor \lg q \rfloor + 2s(q) - 2$.

We can further simplify the number of multiplications in the initialization by noting that $p = 2^n q + 1$, so $\lfloor \lg q \rfloor = \left\lfloor \lg \frac{p-1}{2^n} \right\rfloor = \lfloor \lg p \rfloor - n$ and $s(q) = s(p) - 1$. Therefore, the number of multiplications used in the initialization can be written solely in terms of p as $2 \lfloor \lg p \rfloor - 2n + 2s(p) - 4 = 2\big(\lg p - n + s(p)\big) + O(1)$. If we average this expression over primes less than N, we prove the following theorem.

Theorem 4. *Let $I(p)$ be the number of modular multiplications used by the initialization stage of Shanks's algorithm for the prime p. Then*

$$\frac{1}{\pi(N)}\sum_{p\le N} I(p) \sim 3\lg N.$$

In other words, averaged over primes less than or equal to N, the initialization of Shanks's algorithm requires about $3\lg N$ modular multiplications.

Proof. We observed above that $I(p) = 2\big(\lg p - n + s(p)\big) + O(1)$. Therefore, the average value is

$$\frac{1}{\pi(N)}\sum_{p\le N} I(p) = \frac{2}{\pi(N)}\sum_{p\le N}\big(\lg p - n + s(p) + O(1)\big).$$

Consider each term of the sum in turn. The first term is related to the quantity often called $\vartheta(N)$, which is estimated by the prime number theorem:

$$\vartheta(N) = \sum_{p\le N}\log p \sim N.$$

Therefore,

$$\frac{2}{\pi(N)}\sum_{p\le N}\frac{\log p}{\log 2} \sim \frac{2\log N}{N}\cdot\frac{N}{\log 2} = 2\lg N.$$

The easy way to evaluate the second term is to count how many primes up to N have order $n = n(p)$, then sum over n.

$$\sum_{p\le N} n(p) = \sum_{n=1}^{\infty} n\frac{\pi(N; 2^{n+1}, 1+2^n)}{\pi(N)}.$$

By Proposition 2, each term of this sum is less than or equal to $6n^2/2^n$, so this sum is $O(1)$.

Next, we have the sum of $s(p)$. Use Shiokawa's result [30],

$$\sum_{p\le x} s(p) \sim \frac{x}{2\log 2},$$

to evaluate the term

$$\frac{2}{\pi(x)}\sum_{p\le N} s(p) \sim \frac{2\log N}{N}\cdot\frac{N}{2\log 2} = \lg N.$$

Finally, the last term is $O(1)$. Adding everything together gives

$$\frac{1}{\pi(N)}\sum_{p\le N} I(p) \sim 3\lg N.$$

∎

2.7 Second Moment

The distribution of post-initialization multiplications is far from uniform. This is reflected in the large second moment of the distribution. The first moment, m_1, of the distribution is the expected number of multiplications. The second moment, m_2, is the expectation of the square of the number of multiplications. As we did for the first moment, we find a recurrence to count D_n, the square of the number of multiplications, summed over all the sequences of subgroups (values of k), then divide by the number of sequences. For primes of low orders, we directly calculate D_n: $D_1 = 0$, $D_2 = 4^2 = 16$, $D_3 = 5^2+5^2+9^2 = 131$, $D_4 = 675$, and $D_5 = 2789$. To find a recurrence for D_n, suppose that the 2^{n-1} sequences with order n required $a_1, a_2, a_3, \ldots, a_{2^{n-1}}$ multiplications, with $C_n = \sum_{i=1}^{2^{n-1}} a_i$ as before.

Sequence	Multiplications	Multiplications^2
n	$a_1 = 0$	a_1^2
$n, 1$	a_2	a_2^2
$n, 2$	a_3	a_3^2
$\vdots$	$\vdots$	$\vdots$
$n, n-1, n-2, \ldots, 3, 2, 1$	$a_{2^{n-1}}$	$a_{2^{n-1}}^2$
Total	C_n	D_n

The 2^n sequences possible for a prime of order $n+1$ can again be divided into two types: the sequences that skip the value $k = n$ (or equivalently, b is never an element of $S_n \setminus S_{n-1}$) and those where k takes the value n. As in the derivation of the average, the sequences of the first type are all obtained by copying sequences from order n and changing n to $n+1$. All sequences but the first require one more multiplication than the corresponding order n sequence.

Sequence	Multiplications	Multiplications^2
$n+1$	$a_1 = 0$	a_1^2
$n+1, 1$	$a_2 + 1$	$a_2^2 + 2a_2 + 1$
$n+1, 2$	$a_3 + 1$	$a_3^2 + 2a_3 + 1$
$\vdots$	$\vdots$	$\vdots$
$n+1, n-1, n-2, \ldots, 3, 2, 1$	$a_{2^{n-1}} + 1$	$a_{2^{n-1}}^2 + 2a_{2^{n-1}} + 1$
	Total	$D_n + 2C_n + 2^{n-1} - 1$

The sequences of the second type are obtained by prefixing the sequences from order n by $n+1$, and each takes $n+3$ more multiplications than the corresponding order n

sequence.

Sequence	Multiplications	Multiplications^2
$n+1, n$	$a_1 + (n+3)$	$a_1^2 + 2(n+3)a_1 + (n+3)^2$
$n+1, n, 1$	$a_2 + (n+3)$	$a_2^2 + 2(n+3)a_2 + (n+3)^2$
$n+1, n, 2$	$a_3 + (n+3)$	$a_3^2 + 2(n+3)a_3 + (n+3)^2$
$\vdots$	$\vdots$	$\vdots$
$n+1, n, n-1, \ldots, 3, 2, 1$	$a_{2^{n-1}} + (n+3)$	$a_{2^{n-1}}^2 + 2(n+3)a_{2^{n-1}} + (n+3)^2$
	Total	$D_n + 2(n+3)C_n + 2^{n-1}(n+3)^2$

Adding these two types and expanding C_n we get the recurrence

$$\begin{aligned} D_{n+1} &= 2D_n + (2n+8)C_n + (n^2+6n+10)2^{n-1} - 1 \\ &= 2D_n + 2^{n-2}(n^3 + 13n^2 + 28n - 28) + 2n + 7, \qquad D_1 = 0, \end{aligned}$$

which can again be somewhat tediously solved (but routinely checked by induction) to obtain the closed form

$$D_n = 2^{n-5}(n^4 + \tfrac{46}{3}n^3 + 31n^2 - \tfrac{478}{3}n + 288) - 2n - 9.$$

Since each of the 2^{n-1} sequences is equally probable, as before we calculate the expectation $E\,[\#\text{multiplications}^2]$ by dividing D_n by 2^{n-1} to get $L_n = D_n/2^{n-1}$. The result is:

Lemma 5. *Averaged over all quadratic residue and non-residue inputs, and ignoring the initialization stage, the average of the square of the number of multiplications required by Shanks's algorithm to compute a random square root mod* $p = 2^n q + 1$ *is* $L_n = \frac{n^4}{16} + \frac{23}{24}n^3 + \frac{31}{16}n^2 - \frac{239}{24}n + 18 - \frac{2n+9}{2^{n-1}}$.

As before, we extend this by averaging over all primes. Intuitively, $1/2^n$ of all primes have order n, so the average of the second moment over all primes is

$$\begin{aligned} \sum_{n=1}^{\infty} \frac{L_n}{2^n} &= \sum_{n=1}^{\infty} \left(\frac{1}{16}\frac{n^4}{2^n} + \frac{23}{24}\frac{n^3}{2^n} + \frac{31}{16}\frac{n^2}{2^n} - \frac{239}{24}\frac{n}{2^n} + \frac{18}{2^n} - 4\frac{n}{4^n} - \frac{18}{4^n} \right) \\ &= \tfrac{1}{16} \cdot 150 + \tfrac{23}{24} \cdot 26 + \tfrac{31}{16} \cdot 6 - \tfrac{239}{24} \cdot 2 + 18 \cdot 1 - 4 \cdot \tfrac{4}{9} - 18 \cdot \tfrac{1}{3} \\ &= 36\tfrac{2}{9}. \end{aligned}$$

However, as before this exchange of limiting processes must be made rigorous by starting with

$$\lim_{N\to\infty} \sum_{n=1}^{\infty} \frac{\pi(N; 2^{n+1}, 1+2^n)}{\pi(N)} L_n.$$

The justification is similar to that for the first moment, so is omitted.

2.7.1 Variance and Standard Deviation

For a prime of order n, we compute the variance of X_n, the run time, using the above results:

$$\begin{aligned}\sigma^2(X_n) &= E\left[X_n^2\right] - E\left[X_n\right]^2 \\ &= \frac{n^4}{16} + \frac{23}{24}n^3 + \frac{31}{16}n^2 - \frac{239}{24}n + 18 - \frac{2n+9}{2^{n-1}} - \left(\frac{1}{4}(n^2+7n-12) + \frac{1}{2^{n-1}}\right)^2 \\ &= \frac{n^3}{12} + \frac{3n^2}{8} + \frac{13n}{24} + 9 - \frac{n^2+11n+6}{2^n} - \frac{1}{4^{n-1}}.\end{aligned}$$

Averaged over all primes, the moments are $m_1 = 2\frac{2}{3}$ and $m_2 = 36\frac{2}{9}$, so $\sigma^2 = m_2 - m_1^2 = 29\frac{1}{9}$ and the standard deviation is $\sigma = 5.395471\ldots$. On average, Shanks's algorithm requires fewer than 3 multiplications in the body. However, this average is dominated by the 1/2 of all primes with order 1 that require no additional multiplications. The relatively high standard deviation represents the fact that some primes require a lot of multiplications, $\Theta(n^2)$ multiplications where n is nearly $\lg p$. In practice, it is probably better to use a different algorithm for those primes with large orders. Peralta's algorithm [24], for example, runs slightly faster when the order is large, but is about three times slower than Shanks's when the order is small.

2.8 Running Time Distribution

The distribution of running times of the body is (very nearly) asymptotically normal as the order n goes to infinity. Precisely, let X_n be the random variable that counts the number of multiplications used by Shanks's algorithm (excluding the initialization) on inputs (a, u, p), where p is a prime with order n. The distribution of X_n depends only on n because the $1/2^{n-1}$ different sequences of k-values are equally probable and are the same for all primes of the same order.

Think of the algorithm as proceeding in steps, where one step is one iteration of the loop. The last step is cut short when we find that $b = 1$. Define random variables Y_{nl} for each possible step, including the one that's cut short. Define one more random variable Z_n to account for the early exit. It's the number of multiplications saved by not executing the loop when $b = 1$.

$$Y_{nl} := \begin{cases} l+2 & \text{if } k \text{ is ever assigned the value } l \text{ in the algorithm} \\ 0 & \text{otherwise} \end{cases}$$

$$Z_n := 2 + \text{the last value } k \text{ takes in the algorithm.}$$

In particular, $Y_{nn} = n+2$ and

$$Y_{nl} := \begin{cases} l+2 & \text{with probability } 1/2 \\ 0 & \text{with probability } 1/2 \end{cases} \qquad l = 1, 2, \ldots, n-1.$$

Because all sequences of k-values are equally likely, $\{Y_{nl} : l = 1, 2, \ldots, n\}$ is independent. However, Z_n is not independent of the Y_{nl}. For example, if $Z_n = 6$ then we know that k's last value in the algorithm is 4 and therefore $Y_{n4} = 6$ while Y_{n1}, Y_{n2}, and Y_{n3} are all zero. Z_n's distribution is

$$Z_n = \begin{cases} 3 & \text{with probability } 1/2 \\ 4 & \text{with probability } 1/2^2 \\ 5 & \text{with probability } 1/2^3 \\ & \qquad \vdots \\ n & \text{with probability } 1/2^{n-2} \\ n+1 & \text{with probability } 1/2^{n-1} \\ n+2 & \text{with probability } 1/2^{n-1} \end{cases}$$

Let $W_n = \sum_l Y_{nl}$. Shanks's algorithm uses $X_n = W_n - Z_n$ multiplications in the body. A version of the Central Limit Theorem applies to prove that W_n tends to a normal distribution and direct computation shows that Z_n is small.

Corollary [4, p. 196]. *Assume* $\sum_l \sigma^2(X_{nl}) = 1$, $\gamma_{nl} = E\left[|X_{nl}|^3\right]$ *is finite,* $\Gamma_n = \sum_l \gamma_{nl} \to 0$, *and* $|X_{nl}| \le M_{nl}$ *with* $\max_l M_{nl} \to 0$. *Let* $S_n = \sum_l X_{nl}$. *Then* $S_n - E[S_n]$ *converges in distribution to the standard normal distribution.*

Theorem 6. *Let X_n be the random variable counting the number of modular multiplications used by the body of Shanks's algorithm computing a square root modulo a prime of order n. Then $X_n = W_n - Z_n$, where Z_n is positive with mean less than 4 and variance less than 2 and*

$$\frac{W_n - \mu_n}{\sigma_n} \to \textit{standard normal}$$

where

$$\mu_n = \tfrac{1}{4}(n^2 + 7n + 4)$$

$$\sigma_n^2 = \frac{n^3}{12} + \frac{3n^2}{8} + \frac{13n}{24} - 1.$$

Proof. To apply the above corollary, we let X_{nl} be an appropriately scaled version of Y_{nl} then verify the rest of the hypotheses. Let $b_n^2 = \sigma^2(W_n) = \sum \sigma^2(Y_{nl})$ by independence and let $X_{nl} = Y_{nl}/b_n$. A couple of short calculations gives an exact value for b_n^2:

$$\begin{aligned} b_n^2 &= \sum_l \sigma^2(Y_{nl}) = \sum_l (E\left[Y_{nl}^2\right] - E\left[Y_{nl}\right]^2) = \sum_{l=1}^{n-1} \left(\tfrac{1}{2}(l+2)^2 - \tfrac{1}{4}(l+2)^2\right) + 0 \\ &= \frac{1}{4}\sum_{l=1}^{n-1}(l+2)^2 = \frac{n^3}{12} + \frac{3n^2}{8} + \frac{13n}{24} - 1 \sim \frac{n^3}{12}. \end{aligned}$$

Then $X_{nl} = Y_{nl}/b_n$ satisfies $\sum_l \sigma^2(X_{nl}) = 1$.

Since $b_n = \Theta(n^{3/2})$ and $Y_{nl} = O(n)$, $X_{nl} = \frac{Y_{nl}}{b_n} = O(n^{-1/2}) \to 0$. The third moment $\gamma_{nl} = E\left[X_n^3\right] = O(n^{-3/2})$, and $\Gamma_n = O(n \cdot n^{-3/2}) = O(n^{-1/2}) \to 0$. Thus the corollary applies to show that $S_n - E\left[S_n\right] \to N(0,1)$, the standard normal distribution.

Since $S_n = \sum_l X_{nl} = \sum_l Y_{nl}/b_n = W_n/b_n$ and

$$E\left[W_n\right] = \sum_{l=1}^{n-1} \tfrac{1}{2}(l+2) + (n+2) = \tfrac{1}{4}(n^2+7n+4),$$

we compute $S_n - E\left[S_n\right] = \frac{1}{b_n}(W_n - E\left[W_n\right]) = \frac{1}{b_n}(W_n - \frac{1}{4}(n^2+7n+4)) \to N(0,1)$. So $W_n \to N(\frac{n^2+7n+4}{4}, b_n)$. The time for Shanks's algorithm is $W_n - Z_n$ and

$$E\left[Z_n\right] = \frac{3}{2} + \frac{4}{4} + \frac{5}{8} + \cdots + \frac{n+1}{2^{n-1}} + \frac{n+2}{2^{n-1}} = 4 - \frac{1}{2^{n-1}}.$$

(We obtained a closed form for $1x^1 + 2x^2 + 3x^3 + \cdots + nx^n$ by differentiating the sum of a finite geometric series and multiplying by x.) Similarly, we calculate $E\left[Z_n^2\right] = 18 - \frac{2n+7}{2^{n-1}}$. The variance of Z_n is $E\left[Z_n^2\right] - E\left[Z_n\right]^2 = 2 - \frac{2n-1}{2^{n-1}} - \frac{1}{4^{n-1}}$.

In summary, then, the time X_n for the body of Shanks's algorithm operating on a prime of order n can be written as the difference $W_n - Z_n$, where W_n's limiting distribution is normal with mean $\frac{n^2}{4} + \frac{7n}{4} + 1$ and standard deviation $\sqrt{\frac{n^3}{12} + \frac{3n^2}{8} + \frac{13n}{24} - 1}$. Z_n is positive; its expectation is less than 4 and its variance is less than 2. ∎

2.9 Worst Case of the Algorithm

The worst case of Shanks's algorithm for a specific prime is the last line of its order in one of the above tables. Then the body of the algorithm requires $(n+2)+(n+1)+\cdots+5+4 = \frac{1}{2}(n^2+5n-6)$ multiplications, about twice as many as on average. Therefore, the distribution of running times for various quadratic residues isn't too spread out: no input requires more than twice as many modular multiplications as the average.

Chapter 3

Extensions of Shanks's Algorithm

3.1 Introduction

So far, we have looked at Shanks's algorithm in the finite field $\mathbf{Z}/p\mathbf{Z}$. All of the algorithm's operations are multiplications mod p, not additions, so we can consider the algorithm to be working in the multiplicative group $(\mathbf{Z}/p\mathbf{Z})^\times$. An obvious question to ask is: In which other groups can Shanks's algorithm be used to compute square roots? The algorithm actually works in any cyclic group of even order, so it can be used in any finite field with odd characteristic.

Throughout this chapter, we consider Shanks's algorithm in some group G. Write the group operation as multiplication and denote the group identity by 1. We tacitly assume that we know the order of G, know how to multiply any two elements of G, and can tell when two elements are equal or when an element is the group's identity.

First, we consider computing cube or higher roots in cyclic groups. The degree of the root need not be prime, though there are some coprimality conditions.

Second, we investigate using Shanks's algorithm (both for square roots and for r^{th} roots) in groups other than cyclic groups. When r is a prime number, the generalization to r^{th} roots works in any group that is a direct product $S \times H$ where S is a cyclic r-group and H has order relatively prime to r. In fact, we will see that H does not even need to be abelian! We also show that such groups are the only groups that mimic finite fields in that every element with at least one r^{th} root has exactly r such roots.

Last, we characterize the most general groups in which Shanks's algorithm works to compute square roots. Such groups need not be direct products and may have many Sylow 2-subgroups. A square in one of these groups may have more than two square roots; Shanks's algorithm will compute one of them.

Computing r^{th} roots is easy when the degree r is relatively prime to the order of the group. The following proposition is well-known and will be useful later in this chapter.

Proposition 7. *Suppose a group element y has order relatively prime to r. Then y has an r^{th} root, and furthermore, the root can be computed easily.*

Proof. Let $n = o(y)$. Choose a positive integer s with $sn + 1 \equiv 0 \bmod r$. Such an integer exists because n and r are coprime, and can be found easily by the Euclidean algorithm. The number $(sn+1)/r$ is an integer, so we can compute $x = y^{(sn+1)/r}$. We see that $x^r = y^{sn+1} = y$, so x is an r^{th} root of y. ∎

3.2 Extension of Shanks's Algorithm to r^{th} Roots in Cyclic Groups

Suppose r is an integer at least 2. How does Shanks's algorithm extend to computing r^{th} roots? We already know what to do when $r = 2$, but in that special case our methods below reduce to Shanks's original square root algorithm. Adleman, Manders, and Miller [1] considered the case when r is prime; this case is also described in Bach and Shallit [2, section 7.3]. We go beyond this to certain composite values of r and to operations in any cyclic group.

For this section, let G be a cyclic group. Factor the order of G as $|G| = r^n q$ with $r \nmid q$. An element $x \in G$ is an r^{th} power if and only if $x^{|G|/r} = 1$. This criterion generalizes Euler's criterion for being a quadratic residue modulo a prime.

The cyclic group G has a subgroup S_n of order r^n. This subgroup contains a descending chain of subgroups

$$S_n \supset S_{n-1} \supset \cdots \supset S_2 \supset S_1 \supset S_0 = \{1\}.$$

The subgroup S_{n-i} has order r^{n-i}, and if the element z generates S_n, then z^{r^i} generates S_{n-i} for $i = 0, 1, 2, \ldots, n$. As in the square root case, we will find the generator z by taking the q^{th} power of an appropriate input element. If the degree r is composite, it is not enough to find a group element u that is not an r^{th} power. For example, in the cyclic group of order $6^2 \cdot 5$, even though an element u of order $4 \cdot 5$ is not a sixth power, $z = u^5$ does not generate the cyclic subgroup of order 6^2. We shall require that for every $p \mid r$, the element u not be a p^{th} power. Then it's not hard to show that u^q generates the subgroup S_n.

Since r is not necessarily prime, we have to worry about whether r and q have any factors in common. Let $d = \gcd(r, q)$. If $d > 1$, there may not exist $x = a^k$ and $b = a^l$ so that b is in S_n and $x^r = ab$. (This is a generalization of the loop invariant $x^2 = ab$ in Shanks's square root algorithm.) In general, the best our extension of Shanks's algorithm can do is solve the equation $x^r = a^d$ when an r^{th} power a is given. If r is prime, there is no problem because the gcd d is necessarily 1. Otherwise, we need to consider r's prime factorization and take different roots in succession to get the final answer.

3.2.1 Shanks's Algorithm for r^{th} Roots

In the cyclic group G with order $|G| = r^n q$, where $r \nmid q$, suppose we are given an r^{th} power a and another element u that is a non-p^{th} power for all primes $p \mid r$. Let $d = \gcd(r, q)$. The algorithm finds an element $x \in G$ with $x^r = a^d$. Usually, we will have $d = 1$, but the added generality costs very little. To find all of the r^{th} roots of x, we need to know a primitive r^{th} root of unity in G. Fortunately, $u^{|G|/r}$ is a primitive r^{th} root of unity.

As with the square root case, the algorithm maintains an index k, a generator z of the subgroup S_k, and elements $b \in S_{k-1}$ and $x \in G$. The only difference is that now $x^r = a^d b$ instead of $x^2 = ab$. The algorithm updates the four quantities, maintaining the relationships but decreasing the index k. Eventually, we must have $b = 1$; then the algorithm has computed x so that $x^r = a^d$.

Shanks's Algorithm for r^{th} roots
Initialization:
Find an integer m with $mq + d \equiv 0 \bmod r$ and $1 \le m \le r - 1$
(using the Euclidean algorithm, for example).
Let $k = n$, $z = u^q$, $x = a^{(mq+d)/r}$, $b = a^{mq}$
Body (loop invariants: $x^r = a^d b$, z generates S_k, $b \in S_{k-1}$)
while $b \neq 1$ do
  Find least positive integer m such that $b^{r^m} = 1$
  Update $t = z^{r^{k-m-1}}$
  $z = t^r$
  Find l from $1, 2, \ldots, r-1$ so that $bz^l \in S_{m-1}$
  Update $b = bz^l$
  $x = xt^l$
  $k = m$

3.2.2 Correctness of the Algorithm

After the initialization, $z = u^q$ has order dividing r^n. For every prime $p \mid r$, the power $z^{r^n/p} \neq 1$. Therefore, z has order exactly r^n and generates the subgroup S_n. Because the input a is an r^{th} power, so is $b = a^q$. Therefore $b \in S_{n-1}$. We chose m so that $mq + d$ is a multiple of r, so we can compute the integer power $x = a^{(mq+d)/r}$. A simple calculation verifies that $x^r = a^d b$. Thus all the invariants are satisfied after the initialization.

In the loop, we begin with the element b and repeatedly raise to the r^{th} power to compute b^{r^1}, b^{r^2}, and so on, until $b^{r^m} = 1$. The order of b divides r^{k-1} so this computation terminates after fewer than k exponentiations. Then we have found m so $b \in S_m \setminus S_{m-1}$. The updated value of z generates S_m, so there is an integer $l \in \{1, 2, \ldots, r-1\}$ with $bz^l \in S_{m-1}$. In the square root case, we know that l has to be 1, so no computation is necessary to find it. Finally, it's easy to check that $x^r = a^d b$ is still true with the new values of x and b if it was true for the old values.

After each iteration of the loop, the element b is in a smaller finite group. Hence eventually $b = 1$, at which point $x^r = a^d b$ reduces to $x^r = a^d$, as desired.

3.2.3 Speeding Things Up

Some of the calculations in the algorithm can be done more quickly using less obvious methods. First, the initialization can be computed a bit faster by using an intermediate value $t = a^{(mq+d)/r-1}$. We still need to compute $z = u^q$, but now the calculations of x and b become $x = at$ and $b = a^{r-d}t^r$.

Next, we don't need to use trial and error to compute the power l required in the loop. The condition on l is that $(bz^l)^{r^{m-1}} = 1$. Expand this a bit to get $b^{r^{m-1}} \cdot (z^{r^{m-1}})^l = 1$. Both $\bar{b} = b^{r^{m-1}}$ and $\bar{z} = z^{r^{m-1}}$ are r^{th} roots of unity, so we really need to solve $\bar{z}^l = \bar{b}^{-1}$ for l in a cyclic group of order r. This is an example of a discrete logarithm problem—in this case, computing the logarithm of $\bar{b}^{-1}$ to the base $\bar{z}$. For further discussion of the discrete logarithm problem and several algorithms for solving it, see [20, section 3.6].

In general, discrete logs are very difficult to compute: only in very special cases are polynomial-time algorithms known. Fortunately, this isn't quite as bad as it seems. First of all, the value of $\bar{z}$ is the same every time through the loop because successive values of z are r^{th} powers of the old values. This lets us do some precomputation once (for example, making a lookup table) and use the results many times. Second, we are often interested in computing only low-degree roots, so r is small enough that a precomputed lookup table or exponential-time algorithm is fine. An easy-to-use discrete logarithm algorithm is the baby-step giant-step algorithm, also due to Shanks [28]. This algorithm uses $O(\sqrt{r}\log r)$ group operations and element comparisons to compute the discrete logarithm.

3.2.4 Timing

Suppose we need c multiplications to raise a number to the r^{th} power. Typically c is $\lfloor \log_2 r \rfloor + s(r) - 1$, where $s(r)$ is the number of ones in r's binary representation, although it is possible to do a little bit better [9, 17]. We just assume $c = O(\log r)$. Let's ignore the initialization for a moment and concentrate on the body of the algorithm.

One iteration of the loop uses $m + (k - m - 1) = k - 1$ raisings to the r^{th} power, plus the time to compute the integer l and raise to the l^{th} power twice. As discussed above, l can be found using $O(\sqrt{r}\log r)$ group operations and comparisons each time through the loop (or by doing r operations once as a precomputation). Since $k = O(n)$, each iteration of the loop uses $O(n \log r + \sqrt{r}\log r)$ (or $O(n\log r)$) group operations, and the entire body uses $O(n^2 \log r + n\sqrt{r}\log r)$ (or $O(n^2\log r + r)$) group operations.

The initialization uses two or three exponentiations to powers near $|G|$, which requires $O(\log |G|)$ group operations. The grand total is $O(n^2 \log r + n\sqrt{r}\log r + \log |G|)$ operations if we compute a discrete logarithm each time through the loop, or $O(n^2 \log r + r + \log |G|)$ if we precompute a table of r^{th} roots of unity and use that for the discrete logarithms.

Proposition 8. *Given a cyclic group G and a degree $r = O(\log^2 |G|)$, we can compute r^{th} roots in G deterministically (assuming we are given an r^{th} power nonresidue) using $O(\log^2 |G| \log\log |G|)$ group operations.*

Proof. Use the extension of Shanks's algorithm discussed above. Since $|G| = r^n q$ and $r \geq 2$, we have $n = O(\log |G|)$. Because $r = O(\log^2 |G|)$, we can precompute a table of r^{th} roots of unity and compute a general r^{th} root using $O(n^2 \log r + r + \log |G|) = O(\log^2 |G| \log\log |G|)$ group operations. ∎

If we assume tighter bounds on n and r, we can prove that the extension of Shanks's algorithm uses even fewer group operations.

Proposition 9. *Given an integer r and a cyclic group G, with $|G| = r^n q$, $r \nmid q$, if $r = O(\log|G|)$ and $n = O(\sqrt{\log |G|})$, we can compute r^{th} roots in G deterministically (again assuming we are given an appropriate r^{th} power nonresidue) in $O(\log |G| \log\log |G|)$ group operations.*

Proof. Again, use the extension of Shanks's algorithm discussed above. Precomputing the necessary table of r^{th} roots of unity and computing the actual r^{th} root use $O(n^2 \log r + r + \log |G|) = O(\log |G| \log\log |G|)$ group operations. ∎

3.2.5 Precise Timing

Define a random variable X_n to be the number of group operations used in an instance of Shanks's algorithm for r^{th} roots in the cyclic group G, where $|G| = r^n q$. In an analysis similar to that in section 2.8, we write X_n as a sum of random variables, one for each iteration of the loop, then compute the expected value.

Let c be the number of group operations used to raise to the r^{th} power, let R be the average number of group operations to find the value of l in the loop, and let L be the average number of group operations used to raise to the l^{th} power for $l < r$. An unenlightening and tedious computation shows that

$$E[X_n] = \frac{cr^2(n-2) + r(r-2)(cn^2+4n-4) + (n-1)\left(2(R+L)(r-1)^2 + cn + 4\right)}{2r(r-1)} + \frac{c}{r^{n-1}(r-1)}$$

This expression has so many variables that it's nearly impossible to get a feeling for what it means in general. In the case of square roots, we have $r = 2$, $c = 1$, $R = 0$, and $L = 0$. Then the expected value reduces to $\frac{1}{4}(n^2 + 7n - 12) + 1/2^{n-1}$, which agrees with the value found in chapter 2.

3.3 Computing Roots in Noncyclic Groups

Next, we limit ourselves to the case where r, the degree of the root, is prime. The benefit is that we move beyond just cyclic groups—even into non-abelian groups!

Suppose S is a cyclic group of order r^n and H is any finite group with order relatively prime to r. Let G be the direct product $S \times H$. We prove that Shanks's algorithm as described above computes r^{th} roots correctly in such a group; in fact, this part does not depend on r being prime.

On the other hand, suppose that r is prime and the equation $x^r = a$ has exactly 0 or exactly r solutions for every $a \in G$. (This is the case in the multiplicative group of a field, for example). We then show that G is the direct product of a cyclic r-group with a group of order relatively prime to r, so the extension of Shanks's algorithm computes r^{th} roots in G.

3.3.1 Shanks's Algorithm Works in This Group

We begin by verifying that the criterion for being an r^{th} power still works in this more general setting. Then we prove that the r^{th} root generalization of Shanks's algorithm works.

Lemma 10. *Suppose the group G is a direct product $G = S \times H$, where S is a cyclic group of order r^n (for some positive integers r and n) and H is any finite group with order relatively prime to r. Under these restrictions, an element $g \in G$ is an r^{th} power iff $g^{|G|/r} = 1$.*

Proof. If g is an r^{th} power, $g = x^r$, then $g^{|G|/r} = x^{|G|} = 1$. Conversely, suppose $g^{|G|/r} = 1$. Since G is the direct product $S \times H$, we can represent g as an ordered pair $g = (s, h)$ with $s \in S$ and $h \in H$. The group multiplication operates on each element separately, so we need only show that each of s and h is a r^{th} power. Because h's order is coprime to r, we know immediately that h is an r^{th} power and just need to show that s is. The subgroup S is cyclic; let t be a generator. Then we can write $s = t^k$ for some integer k, and s is an r^{th} power iff k is a multiple of r. Let $q = |H|$. Because $h^q = 1$,

$$(1,1) = 1 = g^{|G|/r} = (s^{r^{n-1}q}, h^{r^{n-1}q}) = (s^{r^{n-1}q}, 1),$$

so $s^{r^{n-1}q} = 1$. We have $1 = s^{r^{n-1}q} = t^{r^{n-1}qk}$, hence $r^n \mid r^{n-1}qk$ and $r \mid qk$. The integers r and q are relatively prime, so $r \mid k$ and s is an r^{th} power. ∎

Theorem 11. *If the group G is a direct product as in Lemma 10, the extension to Shanks's algorithm in section 3.2.1 computes r^{th} roots in G.*

Proof. To verify that the algorithm works, we check that the invariants are maintained and that the index k decreases with each iteration of the loop. Let $q = |H|$. Just as in the case of a cyclic group, $z = u^q$ has order r^n so it generates S, while $b = a^q$ has order dividing r^n so is in S_{n-1}. Since $d = \gcd(r, q) = 1$ and the initial values of the square b and the element x are powers of the residue a, we initially have $x^r = ab$. Therefore the invariants are correctly initialized.

The generator z and r^{th} power b are in a cyclic group of order r^n, so just as before, their updates maintain their invariants and push them into smaller subgroups (i.e., k decreases). All that remains to check is whether $x^r = ab$ after the updates. The new values of x, b, and z are $x_1 = xt^l$, $b_1 = bz^l$, and $z_1 = t^l$. The subgroup S is an abelian direct factor of G, so S is in the center of G. The element t is in S, so it commutes with x. We compute $x_1^r = x^r t^{lr} = ab \cdot z^l = ab_1$. The loop invariants are all satisfied after each trip through the loop so the algorithm terminates with an r^{th} root of the input a. ∎

3.3.2 Groups with r Distinct r^{th} Roots

One property of a cyclic group whose order is a multiple of r is that every element that has any r^{th} root has exactly r such roots. Suppose our group G has the some property. In other words, the equation $x^r = a$ has either 0 or r solutions in G. What can we say about the group and about computing those r^{th} roots? If r is composite, we cannot say much in general: in every finite group G, the identity is the only element with an $|G|^{\text{th}}$ root, and it has exactly $|G|$ such roots. However, if we restrict r to be prime, then we can prove that the group is a direct product as in Lemma 10, so the extension of Shanks's algorithm computes r^{th} roots.

Theorem 12. *Let r be a prime number, G be a finite group with order $r^n q$, $r \nmid q$, and assume that every element of G that has any r^{th} roots has exactly r of them. Then G is the direct product $G = S \times H$, where $|H| = q$ and S is cyclic of order r^n.*

We need to prove a couple of lemmas before we attack the theorem. Along the way, we'll also build up some notation.

Lemma 13. *Let S be a Sylow r-subgroup of G. S is normal and cyclic of order r^n.*

Proof. First, we show by induction that there are no more than r^k elements of G with order dividing r^k, that is, the equation $x^{r^k} = 1$ has at most r^k solutions in G. If $k = 0$, the inductive hypothesis merely states that there is only one element of order 1, which we happen to know is the identity. For $k = 1$, we know that there are exactly r solutions to $x^r = 1$ (since there is one, there must be r of them). For general k, assume the inductive hypothesis is true for k and prove it for $k+1$. We need to show that the equation $x^{r^{k+1}} = 1$ has no more than r^{k+1} solutions. Rewrite the equation as $(x^r)^{r^k} = 1$. By the inductive hypothesis, there are no more than r^k values for x^r; the r^{th} roots condition on G tells us that each value of x^r gives us at most r values for x. Thus there are no more than $r \cdot r^k = r^{k+1}$ solutions to $x^{r^{k+1}} = 1$.

Now take $s \in G$ of maximal r-power order r^k, that is, $o(s) = r^k$ and no element of G has order r^m with $m > k$. Let $T = \langle s \rangle$. This subgroup of G contains r^k elements with orders dividing r^k. But there are no more than r^k such elements, so T contains all of them. In fact, T contains *all* elements with power-of-r order: It contains all the elements with order dividing r^k, and by the construction of s there are no elements of G with power-of-r order higher than r^k. Since $|T| = r^n$, T is a Sylow r-subgroup of G. By its construction, T is the unique Sylow r-subgroup of G, hence $T = S$. Because S is the unique Sylow r-subgroup, it is normal. ∎

From this point, I know of two ways to prove the theorem. The first is short and group-theoretic[1], but requires one lemma that is often taught only in advanced group theory courses. The second proof is mainly computational and, although somewhat tedious, is more elementary.

The advanced lemma used is a corollary to Fitting's Lemma. Fitting's Lemma can be found in gory generality in many advanced undergraduate or graduate-level group theory books—for example, in Rotman's classic book [25, p. 78]. The following corollary was proved in my group theory course [14].

Corollary to Fitting's Lemma. *Suppose r is a prime number, S is an abelian r-group, H acts on S, and H fixes every element of S with order r. If $r \nmid |H|$, the action is trivial.*

Proof of theorem, group-theoretic version. We show that every element with order coprime to r centralizes the Sylow subgroup S. Once we have done that, it follows that the subgroup H generated by such elements commutes with S and intersects S trivially. The subgroups S and H together generate G so $G = S \times H$.

Let $x \in G$ have order m relatively prime to r and let C be the centralizer of x. Because $\gcd(m, r) = 1$, the element x has an r^{th} root by Lemma 10. The r distinct r^{th} roots property of G ensures that x has exactly r roots. These roots all commute with x so they are in the centralizer C, and they generate a subgroup whose order is a multiple of r. Therefore $r \mid |C|$. Since r is prime, C has an element of order r. Ah, but we know what all of the elements of order r are: they're the r^{th} roots of 1. The centralizer C must contain all of them, so the conjugation action of x fixes all elements of order r. This holds for every element $x \in G$, so it holds for the subgroup H generated by them. By the corollary to Fitting's Lemma, the action of H is trivial. Therefore H commutes with S and $G = S \times H$. ∎

The more computational proof requires some preliminary lemmas.

Lemma 14. *G is a semidirect product of S by another subgroup H, $G = S \rtimes H$. This means that $S \cap H = \{1\}$ and there is a homomorphism $\Phi : H \to \mathrm{Aut}(S)$ defined so that $\Phi(h) \in \mathrm{Aut}(S)$ is conjugation by h: $\Phi(h) : s \mapsto hsh^{-1}$. Finally, any element of G is uniquely of the form sh for some $s \in S$ and $h \in H$.*

Proof. $S \triangleleft G$ so by Schur's splitting theorem [27, Theorem 9.3.7] (often called the Schur-Zassenhaus theorem), S has a complement H. Therefore G is the semidirect product $S \rtimes H$. ∎

All we need to do now is prove that the homomorphism Φ is trivial. That means $hsh^{-1} = s$, so H commutes with S and therefore $G = S \times H$.

Proposition 15. *If $h \in H$ and $s \in S$, then $h^{r-1}s = sh^{r-1}$.*

[1]Thanks to Professor Martin Isaacs for suggesting this proof.

Proof. We use the notation introduced in the previous lemma. S is cyclic of order r^n so its automorphism group has $|\operatorname{Aut}(S)| = \phi(r^n) = r^{n-1}(r-1)$, where ϕ is Euler's totient function. The image of H under Φ, $\Phi(H)$, is a subgroup of S isomorphic to $H/\ker(\Phi)$. Therefore $|\Phi(H)|$ divides $|\operatorname{Aut}(S)| = r^{n-1}(r-1)$. Because $\Phi(H)$ is isomorphic to a quotient of H, its order is not divisible by r. Hence $|\Phi(H)|$ divides $r-1$ and $\Phi(h)^{r-1}$ is the identity automorphism. So $h^{r-1}s(h^{-1})^{r-1} = s$ or $h^{r-1}s = sh^{r-1}$. ∎

For the rest of this section, fix a generator s of S. For any $h \in H$, the element $hsh^{-1} \in S$ so $hsh^{-1} = s^i$ for some integer i depending on h, $1 \leq i \leq r^n - 1$. Strictly speaking, i is a function of h so we should write i_h to indicate which element of H we are considering. However, we suppress the h when doing so improves readability. We prove that in fact $i = 1$ so the homomorphism Φ is trivial. Note that for any integers a and b, $hs^ah^{-1} = s^{ia}$ and $h^bs(h^{-1})^b = s^{i^b}$.

Corollary 16. *The integer $i = i_h$ satisfies $i^r \equiv i \bmod r^n$, independent of h.*

Proof. Observe that $s = h^{r-1}s(h^{-1})^{r-1} = s^{i^{r-1}}$. Therefore $s^{i^{r-1}-1} = 1$. Since the order of s is r^n, we have that $i^{r-1} - 1 \equiv 0 \bmod r^n$ or $i^r \equiv i \bmod r^n$. ∎

Lemma 17. *If $hsh^{-1} = s^i$, then for any integer a*

$$(s^ah)^r = \begin{cases} s^{ar}h^r & \text{if } i = 1 \\ s^ah^r & \text{if } i \neq 1 \end{cases}$$

Proof. If $i = 1$ then h and s commute and $(s^ah)^r = s^{ar}h^r$. Now assume that $i \neq 1$. We prove by induction on k that $(s^ah)^k = s^{a\left(1+i+\cdots+i^{k-1}\right)}h^k$. If $k = 1$, the result is obviously true. Now assume the induction hypothesis for $k-1$ and compute $(s^ah)^k = (s^ah)(s^ah)^{k-1} = s^ahs^{a\left(1+i+\cdots+i^{k-2}\right)}h^{k-1} = s^as^{ai\left(1+i+\cdots+i^{k-2}\right)}hh^{k-1} = s^{a\left(1+i+\cdots+i^{k-1}\right)}h^k$. Therefore, $(s^ah)^r = s^{a\left(1+i+\cdots+i^{r-1}\right)}h^r$. Since $i \neq 1$, we can write $1 + i + \cdots + i^{r-1} = (i^r - 1)/(i-1)$. This quantity is congruent to $(i-1)/(i-1) = 1 \bmod r^n$ by Corollary 16. Since $o(s) = r^n$, we have $(s^ah)^r = s^ah^r$. ∎

Proof of theorem. Once again we get to use the hypothesis that elements of G have exactly r distinct r^{th} roots. At most $1/r$ of the elements of G are r^{th} powers because the r^{th} roots of that many elements fill up G completely. Let's count the r^{th} powers of G. We can collect all of them by computing $(s^ah)^r$ for all integers a, $1 \leq a \leq r^n$, and elements $h \in H$.

First, for a fixed h, how many distinct values of $(s^ah)^r$ are there? If $i_h = 1$, Lemma 17 says that $(s^ah)^r = s^{ar}h^r$. Two such values $s^{ar}h^r$ and $s^{br}h^r$ are the same if and only if $ar \equiv br \bmod r^n$, or $a \equiv b \bmod r^{n-1}$. Of the r^n possible choices for a, r^{n-1} of them yield distinct r^{th} powers. On the other hand, if $i_h \neq 1$, then $(s^ah)^r = s^ah^r$ so all of the r^n possible choices for a yield different r^{th} powers.

Next, note that because $r \nmid |H|$ the r^{th} power map in H is one-to-one. Therefore distinct values $h \neq \bar{h}$ give distinct values $h^r \neq \bar{h}^r$ and distinct values $(s^ah)^r \neq (s^a\bar{h})^r$.

Let $q = |H|$ and $n =$ number of $h \in H$ with $i_h = 1$. Then $q - n$ is the number of $h \in H$ with $i_h \neq 1$, and the total number of r^{th} powers in G is $r^{n-1}(n + r(q-n))$. This number is at most $|G|/r = r^{n-1}q$. Therefore $n + r(q-n) \leq q$. Since $r \geq 2$, this inequality implies $n \geq q = |H|$. But we also know $n \leq |H|$, so $n = |H|$. Therefore $i_h = 1$ for all $h \in H$ so the conjugation action is trivial and $G = S \times H$. ∎

3.4 When Does Shanks's Algorithm Work?

We now restrict the degree of the root even further—back to computing only square roots—and characterize groups in which Shanks's algorithm works.

What does it mean to say that "Shanks's algorithm works" in some group G? To begin with, square roots in groups of odd order are easily computed (Proposition 7). So we consider only groups with even order. Two essential ingredients for Shanks's algorithm in a finite field or cyclic group are a quadratic residue and a nonresidue, so we shall need some analogs in general groups. If we know nothing about the structure of G except its order, it is difficult to tell whether a given group element is a square or not. In a cyclic group of even order, the element x is a square iff $x^{|G|/2} = 1$. This test is know as Euler's criterion, and we shall adopt it as a stand-in for a sure-fire method of residuacity testing. The group element x is said to satisfy Euler's criterion if $x^{|G|/2} = 1$. One of the consequences of Shanks's algorithm working in a group is that this criterion actually does test whether an element is a square. The criterion does not work in all groups. For example, in the Klein 4-group (the direct product of two cyclic groups of order 2), every element has order 2, so all 4 elements satisfy Euler's criterion. But the only square in this group is the identity.

The discussion motivates the following definition.

Definition 18. *For a group of even order, we say that* Shanks's algorithm works in the group G *if*

1. *There exists an element $u \in G$ with $u^{|G|/2} \neq 1$.*
2. *For every $a \in G$ with $a^{|G|/2} = 1$ and every $u \in G$ with $u^{|G|/2} \neq 1$, Shanks's algorithm with inputs a and u terminates with an element $x \in G$ satisfying $x^2 = a$.*

Since the group G may not be abelian, we must be careful in which order we multiply. We will multiply elements in the order given in the description of the algorithm in Chapter 2.

We have already seen that Shanks's algorithm works in cyclic groups with even order. It's easy to show that the algorithm works in any group whose order is twice an odd number. We have also shown that the algorithm works in any group that is the direct product of a cyclic 2-group with a group of odd order. What other groups does the algorithm work in? What characterizes such groups?

Theorem 19. *If Shanks's algorithm works in the group G of order $2^n q$ with q odd, then*

1. *G's Sylow 2-subgroups are cyclic.*
2. *The intersection of all of G's Sylow 2-subgroups is a group T of order 2^{n-1}.*
3. *For all squares $a \in G^2$ with $a^q \neq 1$ and $t \in G$ with $o(t) \mid 2o(a^q)$, the elements $a^{(q+1)/2}$ and t commute.*

Proof. 1. Since Shanks's algorithm works, there is an element $u \in G$ with $u^{2^{n-1}q} \neq 1$. We know $z = u^q$ has order dividing 2^n because $|G| = 2^n q$. But since $z^{2^{n-1}} \neq 1$, the order of z is exactly 2^n.

The cyclic subgroup generated by z has order 2^n hence must be a Sylow 2-subgroup of the group G. All the Sylow 2-subgroups of a group have the same structure since they're conjugates of each other, so all of G's Sylow 2-subgroups are cyclic of order 2^n.

2. We show that for any elements b, $y \in G$ having order 2^{n-1}, the cyclic subgroup generated by y contains b. This implies that G has a unique cyclic subgroup T of order 2^{n-1}. Every Sylow 2-subgroup of G has some cyclic subgroup of order 2^{n-1}, which then must be T.

Let b and y be elements of G with order 2^{n-1}, and let z be a square root of y. The square root exists because $o(y) = 2^{n-1}$ and y is in a cyclic subgroup of order 2^n. If an instance of Shanks's algorithm encounters b and z in the initialization, then the updates to b are always of the form $b := bz^{2r}$ for some $r \geq 1$, hence of the form $b := by^r$. The final result of all these updates is the value of b when the algorithm terminates, namely, 1. Therefore $1 = by^r$ for some $r \geq 1$ and hence b is in the cyclic group generated by y.

Finally, we need to check that the values of b and z in the previous paragraph actually do result from an instance of Shanks's algorithm. These elements both have order dividing 2^n, which is coprime to q, so by Proposition 7, there is an a with $a^q = b$; check that $a^{2^{n-1}q} = 1$. Similarly, there is a u with $u^q = z$. Because $o(z) = 2^n$, we have $u^{2^{n-1}q} \neq 1$.

3. For a fixed a, note that $o(a^q)$ is a power of 2 because q is the odd part of the order of the group G. Thus we can write $o(a^q) = 2^m$ for some integer $0 \leq m \leq n$. But we also know that $a^q \neq 1$, so m must be at least 1, and a is a square so $m < n$. We begin by proving this part of the theorem for a specific value of t (depending on a), then prove it for all t with the correct order.

Run Shanks's algorithm with square a and an element u to be specified later not satisfying Euler's criterion. The initialization computes $z := u^q$, $b := a^q$, and $x := a^{(q+1)/2}$. After the initialization, $x^2 = ab$. We have assumed that $a^q \neq 1$, so we must enter the loop of the algorithm.

The order $o(b) = 2^m$, so the algorithm computes $t := z^{2^{n-m-1}}$, $z_1 := t^2 = z^{2^{n-m}}$, $x_1 := xt$, and $b_1 := bz_1 = bt^2$. Suppose we chose the input u so $b_1 = 1$ (we will see in a moment that this is possible). The algorithm then terminates with $(xt)^2 = x_1^2 = a$. Because $bz_1 = b_1 = 1$, we have $xtxt = a = abz_1 = x^2t^2$. Cancel the factors x on the left and t on the right to give $tx = xt$; in other words, $x = a^{(q+1)/2}$ commutes with this specific value of t.

How should we choose the input u? We want $t^2 = b^{-1}$. Since $b = a^q$ has order 2^m and is in a cyclic subgroup of order $2^n > 2^m$, the inverse b^{-1} has the same order as b and has a square root; let t be one of the square roots. This element t has order 2^{m+1} and is in a cyclic subgroup of order 2^n, so it has a $2^{n-m-1\text{th}}$ root; let z be one of these roots. We can finally define u: it is a q^{th} root of z. Such a root exists because q is odd and the order of z is a power of 2.

The element t generates a cyclic subgroup of order 2^{m+1}. If $m+1 < n$, the group G has a unique subgroup of order 2^{m+1} (it's some subgroup of the cyclic subgroup T defined by the theorem). Every element with order dividing 2^{m+1} must then be a power of t. Since the specific value t commutes with $a^{(q+1)/2}$, every power of t also commutes with $a^{(q+1)/2}$, proving this part of the theorem for elements a with $o(a^q)$ dividing 2^{n-2}.

On the other hand, if $m = n-1$ (so $o(a^q) = 2^{n-1}$), we need to be a bit more careful: not all elements of order $2 \cdot 2^{n-1}$ are necessarily powers of the specific element t resulting from the above choice of the input u, so we may need to choose a different element not satisfying Euler's criterion. Let s be an arbitrary element with order 2^n. Since $m = n-1$, the computations in the loop of Shanks's algorithm compute $t := z$ and $z_1 := t^2 = z^2$. We need to show there is some element u such that $t = u^q$ has order 2^n, is a square root of b^{-1}, and generates the same Sylow 2-subgroup as s. Let S be the subgroup generated by s. Because $m = n-1$, the element b^{-1} has order 2^{n-1} and therefore must generate T, the unique subgroup with order 2^{n-1}. Conclusion 2 says that T is a proper subgroup of S. Working in S, we observe that b^{-1} has a square root $t \in S$, and that t has a q^{th} root u. For this u, Shanks's algorithm works by hypothesis. We have seen that $t = u^q$ commutes with $a^{(q+1)/2}$ and generates the subgroup S containing s. Therefore, s, being a power of t, also commutes with the element $a^{(q+1)/2}$. Since this holds for all s with order 2^n, we have proved this part of the theorem. ∎

We now know some necessary conditions for Shanks's algorithm to work. In fact, the three conditions are also sufficient. They are also independent: there are groups satisfying the first two conditions but not the third.

Theorem 20. *If the group G has even order $|G| = 2^n q$, where q is odd, and the following three conditions hold, then Shanks's algorithm works in G.*

1. *G's Sylow 2-subgroups are cyclic.*

2. *The intersection of all of G's Sylow 2-subgroups is a group T of order 2^{n-1}.*

3. *For all squares $a \in G^2$ with $a^q \neq 1$ and $t \in G$ with $o(t) \mid 2o(a^q)$, the elements $a^{(q+1)/2}$ and t commute.*

Proof. First of all, we have assumed that $|G|$ is even. Next, we check that G includes an element not satisfying Euler's criterion. Since G's Sylow 2-subgroups are cyclic with order 2^n and q is odd, any generator u of a Sylow 2-subgroup has $u^{2^{n-1}q} \neq 1$.

Next, suppose that a satisfies Euler's criterion and u does not. We show that Shanks's algorithm computes $x \in G$ with $x^2 = a$ by checking that the loop invariants are always satisfied. They are obviously satisfied after the initialization. Suppose the square b is not 1 and we iterate the loop. Initially, we know that z generates some 2-subgroup, S, of order 2^k and that b is a square in this subgroup. In particular, $b \in T$, the intersection of the Sylow 2-subgroups, so the computation of its order 2^m proceeds normally. The algorithm computes t, which may be above the intersection T, and $z_1 := t^2 \in T$. Thus $b_1 = bz_1$ has order strictly smaller than $o(b)$ and is a square in $\langle z_1 \rangle$.

It only remains to check that $x_1^2 = ab_1$ after we compute $x_1 := xt$. We need to verify that $xtxt = abt^2 = x^2t^2$, i.e., that the elements x and t commute. The first time through the loop of Shanks's algorithm, we compute an element t with $o(t) = 2o(b)$. We need to know that x and t commute, but since $x = a^{(q+1)/2}$ and $o(t) \mid 2o(a^q)$, the third hypothesis implies that x and t do commute.

In the second iteration of the loop, we have a new value t_1, which is a power of t, and $x_1 = xt$. In order that the invariant be maintained, we must have $x_1t_1 = t_1x_1$, or $xtt_1 = t_1xt$. Because t_1 is a power of t, it commutes with both x and t. Therefore $xtt_1 = t_1xt$, as desired.

In later iterations of the loop, we have an element x_k which is the product of the original element x with various powers of t. This element must commute with t_k, another product of powers of t. Because x and t commute, so do the elements x_k and t_k. Therefore, the invariant $x_k^2 = ab_k$ still holds.

Since $x^2 = ab$ at each stage of the algorithm, and since b is in a smaller finite group each time, Shanks's algorithm terminates with a square root of the input a. ∎

Chapter 4

Sums of Digits of Numbers in Arithmetic Progressions

4.1 Introduction

We can write any positive integer n in base b uniquely as $n = n_0 + n_1 b + n_2 b^2 + \cdots + n_r b^r$, where $0 \le n_i < b$ and $1 \le n_r < b$. Define the sum of digits function $s(n)$ by

$$s(n) = \sum_{i=0}^{r} n_i$$

(some authors write $s_b(n)$ to emphasize the dependence on the base). The sum of digits in base 2 is sometimes called the Hamming weight; we will generally talk about the *weight* of a number, meaning its digital sum (the sum of its digits). Many people have studied the sum of weights of various subsets of the integers.

The simplest case is the total weight of all positive integers up to some bound, $S(x) = \sum_{n<x} s(n)$. In a short paper in 1949, Mirsky [21] proved that

$$S(x) = \frac{b-1}{2}\, x \log_b x + O(x).$$

Other researchers have found tight upper and lower bounds on $S(x)$ in the base 2 case [19] and studied the fractal properties of the error term [6, 12]. Stolarsky [34] briefly surveys much early work on the problem including many rediscoveries and misattributions of Mirsky's result.

Summing the weights of primes is more complicated and generally uses non-elementary techniques. Shiokawa [30] proved

$$\sum_{p \le x} s(p) \sim \frac{b-1}{2} \frac{x}{\log b}$$

in 1974 and gave an explicit error term. In 1977, Katai [15] improved the error term to $O(x/(\log x)^{1/2})$ and considered other moments of the distribution.

A third possibility, and the one we consider in this chapter, is summing weights of numbers in arithmetic sequences. One early analysis of this problem was done by Donald Newman in 1969 [22], who showed that in base 2, more multiples of 3 have even weight than odd. Newman also gave the same explicit formulas for the excesses that we derive in section 4.7.1, although his proof was more analytic and less general than ours. J. Coquet [6] wrote about fractal behavior of the excess of multiples of 3 with even over odd binary weight; in fact, explicit expressions for the Fourier coefficients of the fractal part of the function are known [12].

Dumont [8] also restricted himself to binary numbers, but worked on answering the question "which arithmetic progression mod m has the greatest excess of even weight over odd?" He noted that the answer depends on the order of $2 \bmod m$, specifically, whether 2 is a primitive root mod m.

Fine [11] considered integers $n < x$ in the arithmetic sequence $n \equiv l \bmod p$ and with $s(n) \equiv k \bmod p$. He showed that as long as p is prime and $p \nmid (b-1)$, the density of such numbers is asymptotically what we might expect, namely $1/p^2$. Gelfond [13] generalized this result to two different moduli:

Theorem (Gelfond, 1968). *The number of positive integers n, $n \leq x$, satisfying the conditions $n \equiv l \bmod m$ and $s(n) \equiv k \bmod p$ with $p > 1$, $m > 1$, and $\gcd(p, b-1) = 1$ is given by the formula*

$$T(x) = \frac{x}{mp} + O(x^{\lambda})$$

where $\lambda < 1$ does not depend on x, m, l, or k.

Dekking [7] worked on getting estimates for the exponent λ in Gelfond's theorem. He obtained explicit values for base $b = 2$ when 2 is a primitive root mod m and also considered weights of even integers in bases other than 2.

Not much work has been done with weights of integers expressed in bases other than 2. Although we begin with base 2, we later generalize our results to arbitrary bases.

Our chief result is the explicit generating function recurrence in Theorem 31 that makes it easy to write down the sum of the digits of all numbers in some arithmetic progression mod m as long as the base b is a primitive root for the prime number m, and we only consider the numbers up to a power of the base.

4.2 Weights of Binary Integers

We begin with base 2; the weight (sum of digits) of an integer is now the number of ones in its binary representation.

Define $f_m(x, k, l)$ to be the number of nonnegative integers up to x with a given weight, k, in a particular congruence class, $l \bmod m$, and define the generating function or exponential sum $F_m(x, l)$:

$$f_m(x, k, l) = \#\{0 \leq n < x : s(n) = k \text{ and } n \equiv l \bmod m\}$$

$$F_m(x, l) = \sum_{\substack{n<x \\ n \equiv l \bmod m}} q^{s(n)} = \sum_{k=0}^{\infty} f_m(x, k, l) q^k.$$

Note that the value of the last argument, l, only matters up to congruence modulo m. Also, we will occasionally suppress the subscript m for simplicity when there is no attendant loss of clarity.

Alan Stein [32] studied $F_1(x,0)$, the generating function for the arithmetic sequence that contains all the nonnegative integers, and worked out its order of growth. In a later paper [33], he extended the work to arbitrary bases, but still not to numbers restricted to arithmetic sequences.

4.2.1 Basic Recurrences (Fixed Modulus)

By the definitions of $f(x,k,l)$ and $F(x,l)$, we observe immediately that $f(x,k,l)=0$ if $x \le 0$ or if $k<0$ and also that

$$f(1,k,l) = \begin{cases} 1 & k=0 \text{ and } l \equiv 0 \\ 0 & \text{otherwise} \end{cases}$$

because $s(0)=0$.

Lemma 21 (Recurrences for f and F in base 2). *For integers $n \ge 0$, $k > 0$, $m > 0$, l, and $x \le 2^n$,*

$$f_m(2^n + x, k, l) = f_m(2^n, k, l) + f_m(x, k-1, l-2^n) \tag{4.1}$$

$$F_m(2^n + x, l) = F_m(2^n, l) + qF_m(x, l-2^n).$$

In particular,

$$f_m(2^{n+1}, k, l) = f_m(2^n, k, l) + f_m(2^n, k-1, l-2^n)$$

$$F_m(2^{n+1}, l) = F_m(2^n, l) + qF_m(2^n, l-2^n). \tag{4.2}$$

Proof. The integers up to $2^n + x$ are either less than 2^n or are 2^n plus a number less than x. Use the definition of $f_m(x,k,l)$ to compute

$$\begin{aligned} f_m(2^n + x, k, l) = \#\{r < 2^n : s(r) = k \text{ and } r \equiv l \bmod m\} \\ + \#\{r < x : s(r + 2^n) = k \text{ and } r + 2^n \equiv l \bmod m\}. \end{aligned}$$

The first term is simply $f_m(2^n,k,l)$. In the second term, since $r < x \le 2^n$, we have that $s(r+2^n) = 1+s(r)$ so $\#\{r < x : s(r) = k-1 \text{ and } r \equiv l-2^n \bmod m\} = f_m(x, k-1, l-2^n)$.

The F_m recurrences may be derived from the f_m recurrences directly by writing F_m as a generating function for f_m, or we can prove them similarly to the f_m recurrences by using the definition. ∎

How long does it take to compute $f_m(y,k,l)$ using these recurrences? Let $n = \lfloor \log y \rfloor$; then intermediate and final results will have length $O(n)$. First write down the base cases $f(1,k,l)$ for all $0 \le k \le n$, $0 \le l < m$. Then compute $f(2^1,k,l) = f(2^0,k,l) + f(2^0,k-1,l-2^0)$ for $0 \le k \le n$, $0 \le l < m$. In general, we compute $f(2^i,k,l)$ using values of $f(2^{i-1},.,.)$. For each of these computations, we need to compute $l - 2^i \bmod m$ and add two numbers of length $O(n)$, requiring $O(n + \log m)$ bit operations. We do this mn^2 times to get $f(2^i,k,l)$ for all $i \le n$, $k \le n$, $l < m$; this requires $O(mn^3 + mn^2 \log m)$ bit operations. Finally, we use recurrence (4.1) no more than n times to get the final value of $f(y,k,l)$. Since $n = O(\log y)$, the total computation uses $O(m(\log y)^2(\log y + \log m))$ bit operations.

The polynomial F_m is a generating function for f_m, so the computations just described suffice to find the coefficients, which gives us the polynomial. Alternatively, we could compute using the recurrence for F_m; that requires about the same number of arithmetic operations.

4.2.2 Basic Recurrences (Variable Modulus)

Notice that in base 2, the numbers congruent to 4 mod 10 are exactly twice those congruent to 2 mod 5, or in general, those congruent to $2l \bmod 2m$ are twice the numbers congruent to $l \bmod m$. Similarly, the numbers congruent to $2l+1 \bmod 2m$ are one more than twice the numbers congruent to $l \bmod m$. This leads us to formulate the following proposition that reduces the computation of f_{2m} to that of f_m.

Proposition 22. *$f_{2m}(x,k,2l) = f_m(x/2,k,l)$ and $f_{2m}(x,k,2l+1) = f_m((x-1)/2,k-1,l)$.*

Proof. If $n < x$, $n \equiv 2l \bmod 2m$, and $s(n) = k$ then n must be even and $n/2 < x/2$, $n/2 \equiv l \bmod m$, and $s(n/2) = k$. Conversely, if $n/2$ satisfies the second set of properties, then n satisfies the first. Therefore, division by 2 defines a bijection from the set counted by $f_{2m}(x,k,2l)$ to the set counted by $f_m(x/2,k,l)$.

Similarly, if $n < x$ and $s(n) = k$, but now $n \equiv 2l+1 \bmod 2m$, then n is odd and $(n-1)/2 < (x-1)/2$, $s((n-1)/2) = k-1$, and $(n-1)/2 \equiv l \bmod m$. Thus $n \mapsto (n-1)/2$ defines a bijection between the sets counted by $f_{2m}(x,k,2l+1)$ and $f_m((x-1)/2,k-1,l)$. ∎

4.2.3 Other Properties of These Functions

If we sum f_m over all possible weights k, we count all of the integers up to x which are congruent to $l \bmod m$. Thus $\sum_k f_m(x,k,l) = \lfloor (x-l)/m \rfloor + 1 \approx x/m$. By the definition of F_m, this is also $F_m(x,l)$ evaluated at $q = 1$.

Summing over l to get all numbers with a given weight is easy in the binary case, but will be more difficult for general bases. When x is a power of 2, $x = 2^n$, we have

$$\sum_{l=0}^{m-1} f_m(2^n, k, l) = \binom{n}{k}$$

because the sum includes all binary numbers of length less than or equal to n having exactly k ones. The corresponding generating function is

$$\sum_l F_m(2^n, k, l) = \sum_k \binom{n}{k} q^k = (1+q)^n.$$

It turns out to be easier to study the deviations of f and F from their average values, so we define

$$\hat{f}_m(x,k,l) = f_m(x,k,l) - \frac{1}{m}\sum_{l=0}^{m-1} f_m(x,k,l) \qquad \text{and} \qquad \hat{F}_m(x,l) = \sum_k \hat{f}_m(x,k,l) q^k.$$

We shall generally use these only when x is a power of 2; in that case

$$\hat{f}_m(2^n,k,l) = f_m(2^n,k,l) - \frac{1}{m}\binom{n}{k} \qquad \text{and} \qquad \hat{F}_m(2^n,l) = F_m(2^n,l) - \tfrac{1}{m}(1+q)^n.$$

With these new definitions, the sums over congruence classes become

$$\sum_{l=0}^{m-1} \hat{f}_m(x,k,l) = 0 \qquad \text{and} \qquad \sum_{l=0}^{m-1} \hat{F}_m(x,l) = 0. \tag{4.3}$$

The old recurrences give exactly analogous recurrences for $\hat{f}$ and $\hat{F}$ because the recurrences are additive: Compute

$$\begin{aligned}\hat{f}(2^n + x, k, l_0) &= f(2^n + x, k, l_0) - \frac{1}{m}\sum_l f(2^n + x, k, l)\\ &= f(2^n, k, l_0) + f(x, k-1, l_0 - 2^n) - \frac{1}{m}\sum_l \left(f(2^n, k, l) - f(x, k-1, l-2^n)\right)\\ &= \hat{f}(2^n, k, l_0) + \hat{f}(x, k-1, l_0 - 2^n),\end{aligned}$$

and similarly for $\hat{F}$.

The quantities $\hat{F}$ and $\hat{f}$ are easy to compute for general x. To compute $\hat{F}_m(x, l_0)$, for example, compute $F_m(x,l)$ for $l = 0, 1, \ldots, m-1$ as discussed in section 4.2.1, then use them to calculate the average to subtract from $F_m(x, l_0)$.

4.3 When 2 Is a Primitive Root

In some cases, the two-dimensional basic recurrences in Lemma 21 can be applied several times so that only a one-dimensional recurrence is left. The simplest case of this is when the modulus m is a prime that has 2 as a primitive root (i.e., 2 generates the multiplicative group of $\mathbf{Z}/m\mathbf{Z}$; by Artin's conjecture, this happens for 37% of all primes). First, let's examine what happens in general if we apply recurrence (4.2) several times starting with a power of 2.

Theorem 23. *For any integers m, n, i, l satisfying $m, n \geq 1$ and $0 \leq i \leq n$, writing F for F_m,*

$$F(2^n, l) = \sum q^{a_1+a_2+\cdots+a_i} F(2^{n-i}, l - a_1 2^{n-1} - a_2 2^{n-2} - \cdots - a_i 2^{n-i})$$

where the sum is taken over the 2^i distinct i-tuples $(a_1, \ldots, a_i)$ consisting of zeroes and ones. The same result holds when we replace F by $\hat{F}$.

Proof. Proceed by induction. When $i = 0$, the result is trivial: it says that $F(2^n, l) = F(2^n, l)$. For $i = 1$, we obtain

$$F(2^n, l) = \sum_{a_1=0}^{1} q^{a_1} F(2^{n-1}, l - a_1 2^{n-1}),$$

which is a restatement of recurrence (4.2) above.

Now assume the result for $i - 1$ and prove it for i. Using the inductive hypothesis, compute

$$F(2^n, l) = \sum q^{a_1+\cdots+a_{i-1}} F(2^{n-(i-1)}, l - a_1 2^{n-1} - \cdots - a_{i-1} 2^{n-(i-1)})$$

and expand each term of the sum using recurrence (4.2), which says that

$$F(2^{n-(i-1)}, l - a_1 2^{n-1} - \cdots - a_{i-1} 2^{n-(i-1)}) = \sum_{a_i=0}^{1} q^{a_i} F(2^{n-i}, l - a_1 2^{n-1} - \cdots - a_i 2^{n-i}).$$

This gives exactly the desired result for F.

Because $\hat{F}$ obeys the same recurrence as F, the same result holds for $\hat{F}$. ∎

In a moment, we shall see that this formula simplifies considerably when m is a prime with primitive root 2 and $i = m - 1$. We begin by studying a polynomial that can be used to find the coefficients of $F(2^{n-i}, l)$ in the expansion above.

Let

$$G(q, y) = (1 + qy)(1 + qy^2)(1 + qy^3) \cdots (1 + qy^{m-1}).$$

The coefficient α_{kj} of $q^k y^j$ in $G(q, y)$ is the number of ways to choose k distinct integers from $\{1, 2, \ldots, m-1\}$ that sum to j. We can also think of this coefficient as the number of ways to select a zero-one vector $(a_1, a_2, \ldots, a_{m-1})$ with $\sum a_i = k$ and $\sum a_i i = j$. Define a similar quantity β_{kj} to be the number of ways to select a zero-one vector $(a_1, a_2, \ldots, a_{m-1})$ with $\sum a_i = k$ as before, but now with $\sum a_i i \equiv j \bmod m$. Since

$$\beta_{kj} = \sum_{r \equiv j \bmod m} \alpha_{kr},$$

we will be able to find a closed form for its generating function

$$G_j(q) = \sum_k \beta_{kj} q^k.$$

Lemma 24. *In the notation of the previous paragraph, if m is an odd prime, then*

$$G_j(q) = \begin{cases} \frac{1}{m}\left(G(q,1) + (m-1)\frac{q^m+1}{q+1}\right) & j = 0 \\ \frac{1}{m}\left(G(q,1) - \frac{q^m+1}{q+1}\right) & j \neq 0 \end{cases}$$

and for i, $j \neq 0$ we have $\beta_{ki} = \beta_{kj}$.

Proof. We need to collect the terms of $G(q,y)$ with exponents in certain congruence classes. In general, to collect the terms of a polynomial $h(q) = \sum a_j q^j$ with exponents congruent to $-l \bmod m$, let ω be a primitive m^{th} root of unity and compute

$$\sum_{j \equiv -l \bmod m} a_j = \frac{1}{m} \sum_{r=0}^{m-1} h(\omega^r)\omega^{rl}.$$

Verify this formula by noting that

$$\sum_{r=0}^{m-1} h(\omega^r)\omega^{rl} = \sum_{r=0}^{m-1} \sum_j a_j \omega^{rj}\omega^{rl} = \sum_j a_j \sum_{r=0}^{m-1} (\omega^r)^{j+l}.$$

If $j + l \equiv 0 \bmod m$, then $\sum (\omega^r)^{j+l}$ is simply m; if $j + l \not\equiv 0 \bmod m$, then this sum is a sum of all n^{th} roots of unity for some $n \mid m$, hence is 0.

Apply this to our polynomial $G(q,y)$. First compute $G(q,\omega) = (1 + q\omega)(1 + q\omega^2)\cdots(1 + q\omega^{m-1})$. Since $(1-z)(1-\omega z)(1-\omega^2 z)\cdots(1-\omega^{m-1}z) = 1 - z^m$ and m is odd, we see that

$$G(q,\omega) = \frac{1-(-q)^m}{1+q} = \frac{q^m+1}{q+1}.$$

This holds when we replace ω by any other primitive m^{th} root of unity. Since m is prime, all of the nontrivial roots of unity ω^r, $r = 1, 2, \ldots, m-1$, are primitive. Therefore $G(q,\omega^r) = G(q,\omega)$ for $r = 1, 2, \ldots, m-1$.

Collect terms with exponents $j \equiv 0$ to get

$$\begin{aligned} G_0(q) &= \frac{1}{m}\sum_{r=0}^{m-1} G(q,\omega^r)\omega^{r0} = \frac{1}{m}\sum_{r=0}^{m-1} G(q,\omega^r) \\ &= \tfrac{1}{m}\big(G(q,1)+(m-1)G(q,\omega)\big) = \frac{1}{m}\Big(G(q,1)+(m-1)\frac{q^m+1}{q+1}\Big). \end{aligned}$$

When $j \neq 0$, note that when m is prime, the sum of all primitive m^{th} roots of unity is -1 and compute

$$G_j(q) = \frac{1}{m}\sum_{r=0}^{m-1} G(q,\omega^r)\omega^{rj} = \frac{1}{m}\Big(G(q,1)+G(q,\omega)\sum_r \omega^{rj}\Big) = \frac{1}{m}\Big(G(q,1)-\frac{q^m+1}{q+1}\Big).$$

Since $G_j(q) = G_i(q)$ when $i, j \neq 0$, it follows that their coefficients β_{kj} satisfy $\beta_{kj} = \beta_{ki}$. ∎

Now we can prove the simple one-dimensional generating function recurrence mentioned at the beginning of this section.

Theorem 25. *If m is a prime number and 2 is a primitive root for m, then*

$$\hat{F}_m(2^{n+m-1}, l) = \hat{F}_m(2^n, l)\frac{q^m+1}{q+1}.$$

Proof. According to Theorem 23,

$$\hat{F}(2^n, l) = \sum q^{a_1+\cdots+a_{m-1}} \hat{F}(2^{n-m+1}, l - a_1 2^{n-1} - a_2 2^{n-2} - \cdots - a_{m-1}2^{n-m+1})$$

where the sum is over all zero-one vectors $(a_1, \ldots, a_{m-1})$. Since 2 is a primitive root for the prime m, the $m-1$ values $\{2^{n-1}, 2^{n-2}, \ldots, 2^{n-m+1}\}$ mod m are a permutation of $\{1, 2, \ldots, m-1\}$. The sum is invariant under permutations of the a_i, so

$$\hat{F}(2^n, l) = \sum q^{a_1+a_2+\cdots+a_{m-1}} \hat{F}(2^{n-m+1}, l - a_1 1 - a_2 2 - \cdots - a_{m-1}(m-1)).$$

The number of times the term $q^k \hat{F}(2^{n-m+1}, l-j)$ appears is the quantity β_{kj} defined above, so the coefficient of q^k is $\sum_j \beta_{kj}\hat{F}(2^{n-m+1}, l-j)$. We know $\beta_{kj} = \beta_{ki}$ for $i, j \neq 0$ and also that $\sum_j \hat{F}(x, j) = 0$, so

$$\begin{aligned} \sum_{j=0}^{m-1} \beta_{kj}\hat{F}(2^{n-m+1}, l-j) &= \beta_{k0}\hat{F}(2^{n-m+1}, l) + \beta_{k1}\sum_{j=1}^{m-1}\hat{F}(2^{n-m+1}, l-j) \\ &= (\beta_{k0} - \beta_{k1})\,\hat{F}(2^{n-m+1}, l). \end{aligned}$$

Put all of these together to get

$$\begin{aligned}\hat{F}(2^n, l) &= \sum_k q^k(\beta_{k0} - \beta_{k1})\hat{F}(2^{n-m+1}, l) \\ &= \hat{F}(2^{n-m+1}, l)\sum_k(\beta_{k0}q^k - \beta_{k1}q^k) \\ &= \hat{F}(2^{n-m+1}, l)\left(G_0(q) - G_1(q)\right).\end{aligned}$$

Now by Lemma 24, $G_0(q) - G_1(q) = \frac{q^m+1}{q+1}$. Therefore $\hat{F}(2^{n-m+1}, l) = \hat{F}(2^n, l)\frac{q^m+1}{q+1}$. ∎

4.4 Arbitrary Integer Bases

Now we extend the work above to arbitrary (positive integer) bases b. The reader may wish to verify that all of the statements in this section specialize to the previously-proved base 2 observations. We still have $f_m(x, k, l) = \#\{n < x : s(n) = k \text{ and } n \equiv l \bmod m\}$ and $F_m(x) = \sum_k f_m(x, k, l)q^k$, but now $s(n)$ is the sum of the digits in base b. For simplicity, we do not indicate the base in the notation.

As before, define $\hat{f}_m(x, k, l)$ and $\hat{F}_m(x, k, l)$ to be the deviation of f or F from the average:

$$\begin{aligned}\hat{f}_m(x, k, l) &= f_m(x, k, l) - \frac{1}{m}\sum_{l=0}^{m-1} f_m(x, k, l) \\ \hat{F}_m(x, l) &= F_m(x, l) - \frac{1}{m}\sum_{l=0}^{m-1} F_m(x, l).\end{aligned}$$

Just as before, observe that

$$\sum_l \hat{f}_m(x, k, l) = 0 \qquad \text{and} \qquad \sum_l \hat{F}_m(x, l) = 0.$$

Lemma 26 (Recurrence for f and F in base b). *For $1 \le a \le b$, n any positive integer, $x < ab^n$, and integers l and m,*

$$\begin{aligned}f_m(ab^n + x, k, l) &= f_m(ab^n, k, l) + f_m(x, k - a, l - ab^n) \\ f_m(ab^n, k, l) &= \sum_{i=0}^{a-1} f_m(b^n, k - i, l - ib^n) \\ F_m(ab^n + x, l) &= F_m(ab^n, l) + q^a F_m(x, l - ab^n) \\ F_m(ab^n, l) &= \sum_{i=0}^{a-1} q^i F_m(b^n, l - ib^n).\end{aligned}$$

In particular,

$$f_m(b^{n+1},k,l) = \sum_{i=0}^{b-1} f_m(b^n, k-i, l-ib^n)$$
$$F_m(b^{n+1},l) = \sum_{i=0}^{b-1} q^i F_m(b^n, l-ib^n). \tag{4.4}$$

The identical recurrences also hold when $\hat{f}$ and $\hat{F}$ are substituted for f and F, respectively.

Proof. Partition the set of integers $\{r < ab^n + x\}$ into two subsets: $\{r < ab^n\}$ and $\{ab^n \le r < x\}$. Then using the definition,

$$\begin{aligned} f(ab^n + x, k, l) =& \#\{r < ab^n : s(r) = k \text{ and } r \equiv l \bmod m\} \\ &+ \#\{r < x : s(ab^n + r) = k \text{ and } ab^n + r \equiv l \bmod m\} \\ =& f(ab^n, k, l) + f(x, k-a, l-ab^n). \end{aligned}$$

Similarly, partition the set of integers $\{r < ab^n\}$ into a subsets, $\{r < b^n\}, \{b^n \le r < 2b^n\}, \ldots, \{(a-1)b^n \le r < ab^n\}$, and compute

$$\begin{aligned} f(ab^n, k, l) &= \sum_{i=0}^{a-1} \#\{r < b^n : s(ib^n + r) = k \text{ and } ib^n + r \equiv l \bmod m\} \\ &= \sum_{i=0}^{a-1} f(b^n, k-i, l-ib^n). \end{aligned}$$

Since F is just the generating function of f, the recurrences for F follow by computing the generating functions of both sides of the recurrences for f. The analogous statements for $\hat{f}$ and $\hat{F}$ follow easily from their definitions. ∎

Theorem 27. *For any integer $r \ge 0$,*

$$\hat{F}(b^{n+r}, l) = \sum q^{a_1+a_2+\cdots+a_r} \hat{F}(b^n, l - a_1 b^{n+r-1} - a_2 b^{n+r-2} - \cdots - a_r b^n)$$

where the sum is taken over all b^r distinct r-tuples $(a_1, a_2, \ldots, a_r)$ with $0 \le a_i < b$.

Proof. Prove this by induction on r. For $r = 0$, the result is trivial: it merely says that $\hat{F}(b^n, l) = \hat{F}(b^n, l)$. Also, for $r = 1$, the result is exactly the $\hat{F}$ version of recurrence (4.4) above.

Now assume the result for $r-1$ and prove it for r. Using the inductive hypothesis, compute

$$\hat{F}(b^{n+r}, l) = \sum q^{a_1+\cdots+a_{r-1}} \hat{F}(b^{n+1}, l - a_1 b^{n+r-1} - a_2 b^{n+r-2} - \cdots - a_{r-1} b^{n-1})$$

and expand each term with the basic recurrence

$$\hat{F}(b^{n+1}, l - a_1 b^{n+r-1} - \cdots - a_{r-1} b^{n-1}) = \sum_{a_r=0}^{b-1} q^{a_r} \hat{F}(b^n, l - a_1 b^{n+r-1} - \cdots - a_{r-1} b^{n-1} - a_r b^n).$$

This gives the stated r-fold recurrence. ∎

4.5 Example

Consider base $b = 3$, modulus $m = 5$, and $r = 4$. We have

$$\hat{F}(3^{n+4}, l) = \sum_{0 \le a_1, a_2, a_3, a_4 \le 2} q^{a_1+a_2+a_3+a_4} \hat{F}(3^n, l - a_1 3^{n+3} - a_2 3^{n+2} - a_3 3^{n+1} - a_4 3^n).$$

Recall that $\hat{F}(x,0) + \hat{F}(x,1) + \hat{F}(x,2) + \hat{F}(x,3) + \hat{F}(x,4) = 0$. We merely have to maintain our sanity while calculating 81 terms and simplifying, and we get some very nice cancellation.

Begin by computing the coefficient of q^0. The only way to choose (a_1, a_2, a_3, a_4) that sum to 0 is $(a_1, a_2, a_3, a_4) = (0,0,0,0)$, which gives $\hat{F}(3^n, l)$.

To get a sum of 1 (for the coefficient of q^1), we have 4 possibilities: $(0,0,0,1)$, $(0,0,1,0)$, $(0,1,0,0)$, and $(1,0,0,0)$ which respectively give $\hat{F}(3^n, l - 3^n)$, $\hat{F}(3^n, l - 3 \cdot 3^n)$, $\hat{F}(3^n, l - 9 \cdot 3^n) = \hat{F}(3^n, l - 4 \cdot 3^n)$, and $\hat{F}(3^n, l - 2 \cdot 3^n)$. Using the identity, these sum to $-\hat{F}(3^n, l - 0 \cdot 3^n)$ so the q term is $-q\hat{F}(n, l)$.

Similarly, we compute the other terms up to $q^8 \hat{F}(3^n, l)$ and collapse using the sum identity; the final result is

$$\hat{F}(3^{n+4}, l) = \hat{F}(3^n, l)(1 - q + q^3 - q^4 + q^5 - q^7 + q^8).$$

4.6 Generating Function

Analogous to the base 2 case, the multistep recurrence in Theorem 27 simplifies considerably when m is a prime with primitive root b and we go back $r = m - 1$ steps. We again find a polynomial that is the generating function for the coefficients we need information about.

Define

$$G(q, y) = \prod_{i=1}^{m-1} \sum_{a=0}^{b-1} \left(qy^i\right)^a = \prod_{i=1}^{m-1} \left(1 + qy^i + q^2 y^{2i} + \cdots + q^{b-1} y^{(b-1)i}\right).$$

The coefficient α_{kj} of $q^k y^j$ is the number of ways to select integers $a_1, a_2, \ldots, a_{m-1}$ with $0 \le a_i \le b-1$, $\sum a_i = k$, and $\sum a_i i = j$. In other words, it is the number of ways to select an $(m-1)$-tuple $(a_1, \ldots, a_{m-1})$ whose entries are base b digits that sum to k and satisfy $\sum a_i i = j$. Also define the similar quantity β_{kj} that is the number of ways to select such a tuple, except with $\sum a_i i \equiv j \bmod m$. We will use the fact that

$$\beta_{kj} = \sum_{r \equiv j \bmod m} \alpha_{kr}$$

to find the generating function $G_j(q) = \sum_k \beta_{kj} q^k$. The first step is a technical lemma about factoring certain polynomials related to $G(q, y)$.

Lemma 28. *Let U_m be the set of m^{th} roots of unity. If k and m are positive integers with* $\gcd(k, m) = 1$, *then*

$$\frac{q^{mk} - 1}{q^m - 1} = \prod_{\omega \in U_m} \left(1 + (q\omega) + (q\omega)^2 + \cdots + (q\omega)^{k-1}\right).$$

Proof. Note that

$$x^m - 1 = \prod_{\omega \in U_m} (x\omega - 1).$$

Substitute $x = q^k$ to get $q^{mk} - 1 = \prod_\omega (q^k \omega - 1)$. Since $\gcd(k, m) = 1$, the sets of roots of unity $\{\omega : \omega \in U_m\}$ and $\{\omega^k : \omega \in U_m\}$ are the same. Therefore,

$$\prod_\omega (q^k \omega - 1) = \prod_\omega (q^k \omega^k - 1) = \prod_\omega \left((q\omega)^k - 1\right).$$

Now divide by $q^m - 1 = \prod_\omega (q\omega - 1)$ to get

$$\frac{q^{mk} - 1}{q^m - 1} = \prod_\omega \frac{(q\omega)^k - 1}{q\omega - 1} = \prod_\omega \left(1 + (q\omega) + (q\omega)^2 + \cdots + (q\omega)^{k-1}\right).$$

∎

Corollary 29. *For any primitive m^{th} root of unity ω and base b satisfying* $\gcd(m, b) = 1$,

$$G(q, \omega) = \frac{q^{mb} - 1}{q^m - 1} \frac{q - 1}{q^b - 1}.$$

Proof. Since ω is primitive, each power ω^i, $i = 1, 2, \ldots, m-1$ is a different element of $U_m \setminus \{1\}$. The lemma gives

$$\frac{q^{mb} - 1}{q^m - 1} = (1 + q + q^2 + \cdots + q^{b-1}) G(q, \omega),$$

which we solve for $G(q, \omega)$. ∎

Lemma 30. *If m is an odd prime and ω is a primitive m^{th} root of unity, then*

$$G_j(q) = \begin{cases} \frac{1}{m}\big(G(q,1) + (m-1)G(q,\omega)\big) & j = 0 \\ \frac{1}{m}\big(G(q,1) - G(q,\omega)\big) & j \neq 0 \end{cases}$$

and for i, $j \neq 0$ the coefficients of G_i and G_j satisfy $\beta_{ki} = \beta_{kj}$.

Proof. As in the base 2 case, collect terms of $G(q,y)$ with exponent $j \equiv l \bmod m$ by evaluating at roots of unity and summing.

$$G_0(q) = \frac{1}{m}\sum_{r=0}^{m-1} G(q,\omega^r)\omega^{r0} = \frac{1}{m}\sum_{r=0}^{m-1} G(q,\omega^r) = \tfrac{1}{m}\big(G(q,1) + (m-1)G(q,\omega)\big).$$

For $j \neq 0$, note that ω^j is another primitive m^{th} root of unity so $G(q,\omega) = G(q,\omega^j)$ and

$$G_j(q) = \frac{1}{m}\sum_{r=0}^{m-1} G(q,\omega^r)\omega^{rj} = \frac{1}{m}\Big(G(q,1) + G(q,\omega)\sum_r \omega^{rj}\Big) = \tfrac{1}{m}\big(G(q,1) - G(q,\omega)\big).$$

Since $G_j(q) = G_i(q)$ when $i, j \neq 0$, it follows that their coefficients β_{kj} satisfy $\beta_{kj} = \beta_{ki}$. ∎

Theorem 31. *If m is prime and b is a primitive root for m, then*

$$\hat{F}(b^{n+m-1}, l) = \hat{F}(b^n, l)\,\frac{q^{mb}-1}{q^m-1}\cdot\frac{q-1}{q^b-1}.$$

Proof. The proof is exactly analogous to the base $b = 2$ case. Recall that

$$\hat{F}(b^n, l) = \sum q^{a_1+\cdots+a_{m-1}}\hat{F}\big(b^{n-m+1}, l - \textstyle\sum a_i b^{n-i}\big).$$

Since b is a primitive root for m and since we may permute the a_i without altering the sum, we have

$$\begin{aligned}\hat{F}(b^n, l) &= \sum q^{a_1+\cdots+a_{m-1}}\hat{F}\big(b^{n-m+1}, l - \textstyle\sum a_i i\big) \\ &= \sum_k q^k \sum_j \beta_{kj}\hat{F}(b^{n-m+1}, l - j).\end{aligned}$$

As before, we find that

$$\sum_j \beta_{kj}\hat{F}(b^{n-m+1}, l-j) = (\beta_{k0} - \beta_{k1})\hat{F}(b^{n-m+1}, l)$$

hence

$$\hat{F}(b^n, l) = \hat{F}(b^{n-m+1}, l)\big(G_0(q) - G_1(q)\big).$$

Finally, substitute in the values of $G_i(q)$ we found above to prove the theorem. ∎

4.7 Applications and Actual Values

We now use the general theorems above in some specific cases with different moduli. The examples are for base 2, but general bases can be treated similarly.

If m is an odd prime number with 2 as a primitive root, then by repeated application of Theorem 25, we get

$$\hat{F}_m(2^n, l) = (q^{m-1} - q^{m-2} + \cdots + q^2 - q + 1)^{\lfloor n/(m-1) \rfloor} \hat{F}_m(2^{n \bmod m}, l).$$

Evaluate this at $q = -1$ to get the excess of even weights over odd of

$$m^{\lfloor n/(m-1) \rfloor} \hat{F}_m(2^{n \bmod m}, l) \Big|_{q=-1}.$$

So we see that the excess of even weights in congruence classes mod m is determined by the base cases $\hat{F}(2^n, l)$ for $n < m$.

We can also differentiate the generating function to get information about the moments of the weight, taken over numbers with even or odd weights.

4.7.1 Modulus 3

Since 2 is a primitive root for the prime 3 (that is, 2 generates the multiplicative group of integers mod 3), Theorem 25 applies to give $\hat{F}_3(2^n, l) = (q^2 - q + 1)\hat{F}_3(2^{n-2}, l)$ and therefore

$$\hat{F}_3(2^n, l) = (q^2 - q + 1)^{\lfloor n/2 \rfloor} \hat{F}_3(2^{n \bmod 2}, l).$$

Compute the six base cases $\hat{F}(2^n, l)$ for $n = 0, 1$ and $l = 0, 1, 2$, either using the recurrences or simply by writing down the numbers 0 and 1 and their residues mod 3 and weights. Then compute the values of f, F, and finally $\hat{F}$. The result is

$$\begin{array}{lll} \hat{F}(1,0) = 2/3 & \hat{F}(1,1) = -1/3 & \hat{F}(1,2) = -1/3 \\ \hat{F}(2,0) = (-q+2)/2 & \hat{F}(2,1) = (2q-1)/3 & \hat{F}(2,2) = (-q-1)/3. \end{array}$$

Let's examine the excess of even weights over odd in multiples of 3 up to 2^n using $\hat{F}(2^n, 0)$. The computations for numbers congruent to 1 or 2 mod 3 using $\hat{F}(2^n, 1)$ or $\hat{F}(2^n, 2)$ are similar. The recurrence and base cases above evaluated at $q = -1$ tell us that there are

$$\hat{F}(2^n, 0)\Big|_{q=-1} = \begin{cases} 2 \cdot 3^{n/2-1} & n \text{ even} \\ 3^{(n-1)/2} & n \text{ odd} \end{cases}$$

more multiples of 3 with even weight than odd among the numbers less than 2^n. These particular results were published by Newman [22] in 1969. Also note that the fractional deviation of even over odd weight among multiples of 3 up to 4^r is $2 \cdot 3^{r-2}/4^r = \frac{2}{3}\left(\frac{3}{4}\right)^r$, which decays to zero as r increases.

Also observe: Since $\hat{F}(2 \cdot 4^r, 2)\Big|_{q=-1} = 0$, there are exactly as many numbers up to $2 \cdot 4^r$ congruent to 2 mod 3 with even weight as with odd.

4.7.2 Modulus 5

The base 2 is a primitive root for the prime 5, so we can use Theorem 25 in this case as well to obtain

$$\hat{F}_5(2^n, l) = (q^4 - q^3 + q^2 - q + 1)^{\lfloor n/4 \rfloor} \hat{F}_5(2^{n \bmod 4}, l).$$

Again, we only consider $l = 0$, that is, multiples of 5. Other congruence classes modulo 5 can be handled similarly. First compute

$$\begin{aligned}
\hat{F}(2^0, 0) &= 4/5 & \qquad \hat{F}(2^0, 0)\Big|_{q=-1} &= 4/5 \\
\hat{F}(2^1, 0) &= (-q + 4)/5 & \qquad \hat{F}(2^1, 0)\Big|_{q=-1} &= 1 \\
\hat{F}(2^2, 0) &= (-q^2 - 2q + 4)/5 & \qquad \hat{F}(2^2, 0)\Big|_{q=-1} &= 1 \\
\hat{F}(2^3, 0) &= (-q^3 + 2q^2 - 3q + 4)/5 & \qquad \hat{F}(2^3, 0)\Big|_{q=-1} &= 2.
\end{aligned}$$

Since all of the evaluations at $q = -1$ are positive, we know that up to 2^n there are always more multiples of 5 with even weight than odd. For example, of the approximately $\frac{1}{5} 8 \cdot 16^n$ multiples of 5 below $8 \cdot 16^n$, there are $2 \cdot 5^n$ more with even weight than odd.

4.7.3 Moduli 7 and 17

Because 2 isn't a primitive root mod 7, we can't use Theorem 25, but the basic recurrences in Lemma 21 are still valid. I searched for a linear recurrence relation of the form $\hat{F}(2^n, 0) = \alpha_1 \hat{F}(2^{n-1}, 0) + \alpha_2 \hat{F}(2^{n-2}, 0) + \cdots + \alpha_i \hat{F}(2^{n-i}, 0)$ experimentally by computing values of $\hat{F}_7(2^n, 0)$ for $0 \le n \le 20$ and solving the system of linear equations

$$\hat{F}(2^n, 0) = \sum_{i=1}^{r} \alpha_i \hat{F}(2^{n-i}, 0) \qquad n = i, i+1, \ldots, 20,$$

using the Gröbner basis capabilities of Maple. The smallest value of i for which the system has a solution is $i = 6$. Thus we can be sure there is no linear recurrence going back fewer than 6 steps.

For $i = 6$, the recurrence suggested by experiment is

$$\hat{F}(2^n, l) = (q + 1)(2q^2 - 3q + 2)\hat{F}(2^{n-3}, l) - \frac{q^7 + 1}{q + 1} \hat{F}(2^{n-6}, l);$$

it is proved by using the basic recurrences in Lemma 21. There is no obvious pattern to the coefficients other than the symmetry in $2q^2 - 3q + 2$, and the complex zeros of the polynomial $2q^2 - 3q + 2$ don't even have modulus 1.

The next prime that doesn't have 2 as a primitive root is 17. Again, we can search for a recurrence experimentally and then verify it using the basic recurrences. We find

$$\hat{F}_{17}(2^n, l) = (2q^8 - q^7 + 5q^6 - 7q^5 + 4q^4 - 7q^3 + 5q^2 - q + 2)\hat{F}_{17}(2^{n-8}, l) + \frac{q^{17}+1}{q+1}\hat{F}_{17}(2^{n-16}, l),$$

which again doesn't appear to have a simple form, although the unusual polynomial's coefficients again are symmetric.

There are a couple of similarities between these recurrences for moduli $m = 7$ and $m = 17$. In both cases, the recurrence involves three evenly spaced terms, $\hat{F}(2^n, l)$, $\hat{F}(2^{n-(m-1)/2}, l)$, and $\hat{F}(2^{n-(m-1)}, l)$, and in both cases 2 generates a subgroup of index 2 in the multiplicative group modulo m. Finally, the last polynomial coefficient is cyclotomic. However, the other polynomial appearing doesn't appear to be anything special.

4.7.4 Composite Moduli

We can search for recurrences among composite moduli m as well, and some turn up. Unfortunately, they don't exhibit any nice patterns that I can determine.

For modulus 9, the lowest-order linear recurrence is

$$\begin{aligned}\hat{F}_9(2^n, l) = {} & (q^2 - q + 1)\hat{F}(2^{n-2}, l) + (q^6 - q^3 + 1)\hat{F}(2^{n-6}, l) \\ & - (q^8 - q^7 + q^6 - q^5 + q^4 - q^3 + q^2 - q + 1)\hat{F}(2^{n-8}, l).\end{aligned}$$

Another recurrence is

$$\begin{aligned}\hat{F}(2^n, l) = {} & (2q^6 - 3q^5 + 6q^4 - 8q^3 + 6q^2 - 3q + 2)\hat{F}(2^{n-6}, l) \\ & - (q^6 - q^3 + 1)(q^2 - q + 1)^3\hat{F}(2^{n-12}, l).\end{aligned}$$

Modulus 15 is even worse. The minimal recurrence expresses $\hat{F}(2^n, l)$ in terms of seven other $\hat{F}$ down to $\hat{F}(2^{n-14}, l)$, and I haven't found anything much nicer than it.

$$\begin{aligned}\hat{F}(2^n, l) = {} & (q^2 - q + 1)\hat{F}(2^{n-2}, l) \\ & + 3(q^4 - q^2 + 1)\hat{F}(2^{n-4}, l) \\ & - 3(q^6 - q^5 + q^3 - q + 3)\hat{F}(2^{n-6}, l) \\ & - 3(q^8 - q^6 + q^5 - q^4 + q^3 - q^2 + 1)\hat{F}(2^{n-8}, l) \\ & + 3(q^{10} - q^9 + 2q^7 - 3q^6 + 3q^5 - 3q^4 + 2q^3 - q + 1)\hat{F}(2^{n-10}, l) \\ & + (q^{12} - q^9 + q^6 - q^3 + 1)\hat{F}(2^{n-12}, l) \\ & - (q^{14} - q^{13} + q^{12} - q^{11} + q^{10} - q^9 + q^8 - q^7 + q^6 - q^5 + q^4 - q^3 + q^2 - q + 1)\hat{F}(2^{n-14}, l)\end{aligned}$$

4.8 Further Directions

It would be nice to know exact or asymptotic values of what might be called $\pi_k(x)$, the number of primes less than x with digit sum k, and also to know something about its generalization to primes in arithmetic progressions. Unfortunately, these questions are quite hard. In fact, simply deciding the analog of Euclid's theorem about the infinitude of primes (are there infinitely many primes with weight k?) can be difficult.

Restricting ourselves to base 2 only, we at least know that $\pi_1(x) = 1$ for $x \geq 3$ because the only prime with exactly one binary one is the prime 2. Once we get to $\pi_2(x)$, things already become quite difficult. Any odd prime with weight 2 must be one more than a power of 2, so $\pi_2(x) = \#\{p = 2^n + 1 : p < x \text{ and } p \text{ prime}\}$. The only primes of such a form are the Fermat primes $2^{2^n}+1$, and it is an open question whether there are infinitely many such primes. About the only thing I can say about $\pi_k(x)$ is trivial: $\pi(x) = \pi_1(x) + \pi_2(x) + \pi_3(x) + \cdots + \pi_{\lceil \lg x \rceil}(x)$.

4.9 Conclusions

We have found a general and useful recurrence for $\hat{F}_m(b^n, l)$, the exponential sum of base-b weights of integers in the arithmetic sequence $n \equiv l \bmod m$, for $n < b^n$, in the case when m is prime and b is a primitive root for m. Values of $\hat{F}$ can be used to compute how many more numbers in some arithmetic progression have even weight than odd, how many have a particular weight, or anything else you can compute with a generating function in hand.

Bibliography

[1] Leonard Adleman, Kenneth Manders, and Gary Miller. On taking roots in finite fields. In *18th Annual Symposium on Foundations of Computer Science (Providence, R.I., 1977)*, pages 175–178. IEEE Computer Society, Long Beach, Calif., 1977.

[2] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory. Vol. 1: Efficient Algorithms.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1996.

[3] Richard A. Brualdi. *Introductory Combinatorics.* North-Holland Publishing Co., New York, second edition, 1992.

[4] Kai Lai Chung. *A Course in Probability Theory.* Harcourt, Brace & World, Inc., New York, 1968.

[5] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.

[6] Jean Coquet. A summation formula related to the binary digits. *Inventiones Mathematicae*, 73(1):107–115, 1983.

[7] F. Michel Dekking. On the distribution of digits in arithmetic sequences. In *Seminar on Number Theory, 1982–1983 (Talence, 1982/1983)*, chapter 32, 12 pp. Univ. Bordeaux I, Talence, 1983.

[8] Jean-Marie Dumont. Discrépance des progressions arithmétiques dans la suite de Morse. *Comptes Rendus des Séances de l'Académie des Sciences. Série I. Mathématique*, 297(3):145–148, 1983.

[9] Ömer Eğecioğlu and Çetin Kaya Koç. Exponentiation using canonical recoding. *Theoretical Computer Science*, 129(2):407–417, 1994.

[10] Chun-I Fan and Chin-Laung Lei. Low-computation blind signature schemes based on quadratic residues. *Electronics Letters*, 32(17):1569–1570, August 1996.

[11] Nathan J. Fine. The distribution of the sum of digits (mod p). *Bulletin of the American Mathematical Society*, 71:651–652, 1965.

[12] Philippe Flajolet, Peter Grabner, Peter Kirschenhofer, Helmut Prodinger, and Robert F. Tichy. Mellin transforms and asymptotics: Digital sums. *Theoretical Computer Science*, 123(2):291–314, 1994.

[13] Aleksandr O. Gelfond. Sur les nombres qui ont des propriétés additives et multiplicatives données. *Acta Arithmetica*, 13:259–265, 1967/1968.

[14] Martin Isaacs. Math 745 (group theory) class at the University of Wisconsin—Madison, Fall 1993. Nov 17, 1993.

[15] Imre Kátai. On the sum of digits of primes. *Acta Mathematica Academiae Scientiarum Hungaricae*, 30(1–2):169–173, 1977.

[16] Donald E. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical algorithms.* Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont, 1969.

[17] Çetin Kaya Koç. High-radix and bit recoding techniques for modular exponentiation. *Intern. J. Computer Math.*, 40:139–156, 1991.

[18] Kaoru Kurosawa, Toshiya Ito, and Masashi Takeuchi. Public key cryptosystem using a reciprocal number with the same intractability as factoring a large number. *Cryptologia*, 12(4):225–233, 1988.

[19] M. Douglas McIlroy. The number of 1's in binary integers: Bounds and extremal properties. *SIAM Journal on Computing*, 3:255–261, 1974.

[20] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, 1997. With a foreword by Ronald L. Rivest.

[21] Leonid Mirsky. A theorem on representations of integers in the scale of r. *Scripta Math.*, 15:11–12, 1949.

[22] Donald J. Newman. On the number of binary digits in a multiple of three. *Proceedings of the American Mathematical Society*, 21:719–721, 1969.

[23] Ivan Niven, Herbert S. Zuckerman, and Hugh L. Montgomery. *An Introduction to the Theory of Numbers.* John Wiley & Sons Inc., New York, fifth edition, 1991.

[24] René C. Peralta. A simple and fast probabilistic algorithm for computing square roots modulo a prime number. *Institute of Electrical and Electronics Engineers. Transactions on Information Theory*, 32(6):846–847, November 1986.

[25] Joseph J. Rotman. *The Theory of Groups. An Introduction.* Allyn and Bacon Inc., Boston, Mass., second edition, 1973. Allyn and Bacon Series in Advanced Mathematics.

[26] Walter Rudin. *Real and Complex Analysis.* McGraw-Hill Book Co., New York, third edition, 1987.

[27] William Raymond Scott. *Group Theory.* Dover Publications Inc., New York, second edition, 1987.

[28] Daniel Shanks. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969)*, pages 415–440, Providence, R.I., 1971. Amer. Math. Soc.

[29] Daniel Shanks. Five number-theoretic algorithms. In Robert S. D. Thomas and Hugh Cowie Williams, editors, *Proceedings of the Second Manitoba Conference on Numerical Mathematics (Univ. Manitoba, Winnipeg, Man., 1972)*, pages 51–70. Congressus Numerantium, No. VII, Winnipeg, Man., 1973. Utilitas Math.

[30] Iekata Shiokawa. On the sum of digits of prime numbers. *Proceedings of the Japan Academy*, 50:551–554, 1974.

[31] Blair Spearman and Kenneth S. Williams. *Handbook of Estimates in the Theory of Numbers*. Department of Mathematics, Carleton University, Ottawa, Ont., 1975. Carleton Mathematical Lecture Note, No. 14.

[32] Alan H. Stein. Exponential sums related to binary coefficient parity. *Proceedings of the American Mathematical Society*, 80(3):526–530, 1980.

[33] Alan H. Stein. Exponential sums of sum-of-digit functions. *Illinois J. Math.*, 30(4):660–675, 1986.

[34] Kenneth B. Stolarsky. Power and exponential sums of digital sums related to binomial coefficient parity. *SIAM Journal on Applied Mathematics*, 32(4):717–730, 1977.

[35] Alberto Tonelli. Bemerkung über die Auflösung quadratischer Congruenzen. *Göttinger Nachrichten*, pages 344–346, 1891.

[36] Stephen M. Turner. Square roots mod p. *The American Mathematical Monthly*, 101(5):443–449, May 1994.