Counting Change

Vishal Gupta

Introduction

You're at the CMS and grab a panini in between lectures. You pay with the change in your pocket and as you eat it waiting for the next lecture, you wonder: if I had a lot of coins, how many ways could I have paid for it?

Setting up mathematical notation, say the cost is b and we have coin denominations $a_1, \dots, a_n$ (in pennies). Then we are looking for the number of solutions to the equation

$$a_1 x_1 + \cdots + a_n x_n = b \tag{1}$$

This is known as the Diophantine equation of Frobenius, for positive integers $a_1, \dots, a_n, b$. Let this number be $N(b)$. We aim to learn as much as we can about $N(b)$. We suppose that $\gcd(a_1, \dots, a_n) = 1$ - if it was greater than 1, every b would be a multiple of it so the problem is the same, just scaled up.

Generating function

A useful approach is to consider the generating function f of N:

$$1 + \sum_{b=1}^{\infty} N(b) z^b = \frac{1}{(1 - z^{a_1}) \dots (1 - z^{a_n})}$$

Then if we can find the z^b coefficient of this expression, we'll solve the problem.

One way to approach this is with a contour integral, viewing it as a Taylor series centred at $z = 0$, the x^b coefficient can be obtained by differentiating f b times. Expressing this as a contour integral with Cauchy's formula for repeated differentiation we get

$$N(b) = \frac{f^{(b)}(0)}{b!} = \frac{1}{2\pi i} \oint_{C_r} \frac{f(z)}{z^{b+1}} \mathrm{d}z = \mathrm{Res}_{z=0} \left(\frac{1}{z^{b+1}(1 - z^{a_1}) \dots (1 - z^{a_n})} \right)$$

where C_r is a circle of radius $0 < r < 1$ centred at the origin so that it only contains the pole at $z = 0$ - the other poles all have modulus 1. This is complicated to calculate using the typical derivative formula, instead, notice that for a circle C_R centred at the origin,

$$\lim_{R\to\infty}\left|\oint_{C_R}\frac{f(z)}{z^{b+1}}\mathrm{d}z\right| \leq \lim_{R\to\infty}\left|\int_0^{2\pi}\frac{f(Re^{it})}{R^{b+1}e^{it(b+1)}}Re^{it(b+1)}\mathrm{d}t\right| = \lim_{R\to\infty} O(R^{-(b+a_1+\cdots+a_n)}) = 0$$

Choosing R large enough so that it contains all the poles of f, by the residue theorem we know that the contour integral is the sum of the residues, independent of R. Therefore

$$\oint_{C_R}\frac{f(z)}{z^{b+1}}\mathrm{d}z = 0 \implies N(b) = \mathrm{Res}_{z=0}\left(\frac{f(z)}{z^{b+1}}\right) = -\sum_{z\neq 0}\mathrm{Res}\left(\frac{f(z)}{z^{b+1}}\right)$$

$(1-z^{a_i})$ has only simple roots as its derivative has $z=0$ as its only root. Therefore, the residue at $z=1$ is of order n so we can calculate

$$\begin{aligned}-\mathrm{Res}_{z=1}\left(\frac{f(z)}{z^{b+1}}\right) &= \frac{1}{(n-1)!}\lim_{z\to 1}\frac{\mathrm{d}^{n-1}}{\mathrm{d}z^{n-1}}\left(\frac{(z-1)^n}{z^{b+1}(1-z^{a_1})\dots(1-z^{a_n})}\right)\\ &= \frac{1}{a_1\dots a_n}\left(\frac{b^{n-1}}{(n-1)!}+\frac{\sum_{k=1}^n a_k}{2(n-2)!}b^{n-2}+\frac{3\left(\sum_{k=1}^n a_k\right)^2-\sum_{k=1}^n a_k^2}{24(n-3)!}b^{n-3}+\dots\right)\end{aligned}$$

The remaining terms are complicated expressions. For small enough n, we could write out the full formula. However, supposing that b is significantly larger than a_i for each i, we can use this to help us approximate $N(b)$.

The other roots α of f are of order at most n as we pick up at most one factor of $(z-\alpha)$ for each i. α is a root of both $(1-z^{a_i})$ and $(1-z^{a_j})$ only if a_i and a_j share a common factor.

For a pole of order m, by considering the powers of b in a calculation analogous to the above, the residue is of order b^m. Since we assumed that there is no common factor among the a_i, the leading order term is the first term in our $z=1$ residue. Therefore

$$\lim_{b\to\infty}\frac{N(b)}{b^{n-1}} = \frac{1}{(n-1)!a_1\dots a_n} \tag{2}$$

This is known as Schur's theorem.

A notable case is when all the a_i are pairwise coprime. Then the poles are $z=1$ and $z_{rs}=\exp\left(\frac{2\pi ir}{a_s}\right)$ for $r=1,\dots,a_s-1$ (dependent on s) and $s=1,\dots,n$. Each z_{rs} is a simple pole. Therefore

$$-\mathrm{Res}_{z=z_{rs}}\left(\frac{f(z)}{z^{b+1}}\right) = \lim_{z\to z_{rs}}(z-z_{rs})\frac{f(z)}{z^{b+1}} = C_{rs}$$

where $|C_{rs}|$ is independent of b because $|z_{rs}|=1$. This means for large b, the contribution from the poles at $z=z_{rs}$ is negligible.

Frobenius Coin Problem

Another interesting question is to think about what numbers b we can't express in the form $a_1x_1 + \cdots + a_nx_n$.

For example, if we only have £2 coins and £5 notes, you can't make £3. We can make any larger amount; every even number can be made with £2 coins and for an odd number b greater than 5, $b - 5$ is even. Therefore £3 is the largest number we can't make.

Is this always the case for any collection $a_1, \ldots, a_n$ which don't share a common factor? The answer is yes, by Schur's theorem, $N(b) > 1$ for all sufficiently large b. Therefore there is a maximal number that cannot be expressed in the form $a_1x_1 + \cdots + a_nx_n$, this is known as the Frobenius number $g(a_1, \ldots, a_n)$ of the set $\{a_1, \ldots, a_n\}$. We require that $\gcd(a_1, \ldots, a_n) = 1$ otherwise we could choose any number b not divisible by the common factor. What can we learn about $g(a_1, \ldots, a_n)$?

For $n = 1$, we must have $a_1 = 1$ so every positive integer can be expressed.

For $n = 2$, write $a_1 = r$, $a_2 = s$ - we have $\gcd(r, s) = 1$. By Bézout, any positive integer can be expressed as $p = xr + ys$ for $x, y \in \mathbb{Z}$. This representation can be made unique by enforcing $0 \leq x \leq s - 1$ - then the valid representations are those with $y \geq 0$. This gives the largest non-representable number as $p = (s - 1)r - s = rs - r - s$ when we choose $x = s - 1$ and $y = -1$.

We can go further with generating functions. First, we note that $r, 2r, \ldots, (s - 1)r$ are in distinct residue classes modulo s. Next, consider the following sequences:

$$\begin{aligned}
S_0 &= \{0 + 0,\ 0 + s,\ 0 + 2s,\ \ldots\} \\
S_1 &= \{r + 0,\ r + s,\ r + 2s,\ \ldots\} \\
&\ \vdots \\
S_{s-1} &= \{(s - 1)r + 0,\ (s - 1)r + s,\ (s - 1)r + 2s,\ \ldots\}
\end{aligned}$$

Each sequence is disjoint and we know that their union contains all positive integers with some exceptions. Let $v(x)$ be a function taking the value 1 if $x \in S_i$ for some i and 0 otherwise. Then the generating function of v is

$$V(x) = \frac{1}{1 - x^s}(1 + x^r + \cdots + x^{(s-1)r}) = \frac{1 - x^{rs}}{(1 - x^r)(1 - x^s)}$$

We're now looking for the largest power of x which doesn't appear in the expansion of V. This isn't immediately clear from this expression. Instead, since every coefficient is 1 or 0, we can flip the 1s and 0s by subtracting V from the power series whose coefficients are all 1s, namely $h(x) = \dfrac{1}{1-x}$:

$$h(x) - V(x) = \frac{1}{1-x} - \frac{1-x^{rs}}{(1-x^r)(1-x^s)} = \frac{(1-x^r)(1-x^s) - (1-x^{rs})(1-x)}{(1-x)(1-x^r)(1-x^s)}$$

Because of the existence of the Frobenius number, this expression is a polynomial and its degree can be read off by considering the largest powers in x on the top and bottom. This gives us the Frobenius number $g(r,s) = rs - r - s$ as before.

Moreover, we can find the number of numbers which cannot be expressed in the form $rx_1 + sx_2$; this is the number of powers of x in $h(x) - V(x)$ which have a coefficient of 1. Since every other coefficient is 0, this is just $\lim\limits_{x\to 1} h(x) - V(x)$. To find this, you need to use L'hopital's rule multiple times - you can work out the details if you'd like - the answer is $\dfrac{(r-1)(s-1)}{2}$.

Next, we'd move onto $n = 3$, but there is no known closed formula! Instead, there is a lower bound $p(a_1, a_2, a_3) = g(a_1, a_2, a_3) + a_1 + a_2 + a_3 \geq \sqrt{3a_1a_2a_3}$ where the constant $\sqrt{3}$ is sharp. There are many results on $g(a_1, \ldots, a_n)$ which can be found in [2]. However, there are special cases that have closed form answers such as the case of arithmetic and geometric progressions, as found in [3] and [1].

Sorting Coins

Now imagine we had a lot of coins of different denominations and we wanted to sort them. It turns out that the Frobenius number is still relevant!

The Shellsort is an algorithm by Donald Shell, published in 1959. It works as a generalisation of insertion sort. Suppose we have a list of elements $a_1, \ldots, a_n$ which we aim to sort. Choose some sequence of elements $h_1, \ldots, h_k$, known as the gap sequence. To describe Shellsort, choose some h_i. Then we perform insertion sort on the sequences $a_1, a_{1+h_i}, \ldots$ and again on $a_2, a_{2+h_i}, \ldots$ until $a_{h_i-1}, a_{2h_i-1}, \ldots$. Once these subsequences are all sorted, the list of elements is said to be h_i sorted, that is, when $a_{j-h_i} \leq a_i$ for $j = h_i + 1, \ldots, n$. Shellsort operates by doing this procedure for each $h_1, \ldots, h_k$. If we h_2 sort a h_1 sorted array then it remains h_1 sorted. The final list is guaranteed to be sorted if $h_k = 1$ (so that the final pass is an insertion sort) - if the final pass isn't $h_j = 1$, the list isn't necessarily sorted.

For a very simple example, this can be seen with the sequence $3, 5, 4, 1$ and gap size $h = 2$. The single pass through will give us the sequence $3, 1, 4, 5$ which isn't sorted. This makes Shellsort seem like it's worse than insertion sort as we have to do an insertion sort anyway! However it is typically better - the intuitive reason is that in large arrays, the larger gap sizes will take care of the swaps that need to be made across large distances. This means there is typically less work to be done when working with the smaller gap sizes, making it generally more efficient than insertion sort. For an example of the algorithm that illustrates this, we will use it to sort the array $[5, 3, 1, 6, 2, 4]$ with gap sizes 2,1.

Input data	5	3	1	6	2	4
After 2-sorting	1	3	2	4	5	6
After 1-sorting	1	2	3	4	5	6

Table 1: An example run of Shellsort with gaps 2 and 1

In the first pass with gap size 2, we perform insertion sort on the subarrays $[5, 1, 2]$ and $[3, 6, 4]$, this gives us $[1, 2, 5]$ and $[3, 4, 6]$. With the final pass of gap size 1, we only have to make one swap which is far quicker than if we had run insertion sort in the first place. However, Shellsort isn't always better, such as with the list $1, 3, 2, 4, 5$.

The time complexity depends on the gap sequence chosen - there are many choices but the time complexity for many of them remains an open problem. The relation to the Frobenius number can be seen by noting that any array which is both h_1 and h_2 sorted is $a_1h_1 + a_2h_2$ sorted. In fact, the following result is true (Lemma 8.14 of [2]):

Theorem: The number of steps required to h_i sort an array $a_1, \ldots, a_n$ which is already $h_{i+1}, h_{i+2}, \ldots, h_t$ sorted is

$$O\left(\frac{ng(h_{i+1}, \ldots h_t)}{h_i}\right)$$

However in general, the Shellsort algorithm is slower than algorithms like merge sort and quick sort. Its advantage is that it doesn't require any memory beyond the original array and is fairly simple to implement. There is also flexibility with the choice of gap sequences so that the algorithm can be tweaked for specific scenarios.

Conclusion

Starting with just a problem about change, we've managed to cover quite a lot of different mathematics! There is still yet more mathematics related to this, with more detailed estimates on the Frobenius number and many problems where it finds applications. Lots of these are detailed in *The Diophantine Frobenius Problem*, a great place to read further.

Bibliography

References

[1] Darren C. Ong and Vadim Ponomarenko. The frobenius number of geometric sequences. *Integers*, 8(1):Article A33, 3 p., electronic only–Article A33, 3 p.,

electronic only, 2008. URL http://eudml.org/doc/117381.

[2] Jorge L. Ramírez Alfonsín. *The Diophantine Frobenius Problem.* Oxford University Press, 12 2005.

[3] J. B. Roberts. Note on linear forms. *Proceedings of the American Mathematical Society*, 7(3):465–469, 1956. ISSN 00029939, 10886826.