

21. Factoring Numbers

21.1 Introduction

Many algorithms for factoring numbers have been developed, and in this section we will describe one of them, the function iteration, or the Tortoise and the Hare method. Although it is a very interesting approach, the number of steps it takes is order $N^{1/4}$ on factoring N , which is far worse than the two methods we detail in the next note section. On large numbers, say 10^{50} or greater, the function iteration becomes completely infeasible. The other methods are the quadratic sieve and the elliptic curve, both of which interestingly enough have similar performance to the first method, but are easily parallelizable to multiple computers and thus are superior algorithms to use.

21.2 The Function Iteration or Cycle Method of Factoring

Let's begin with integer N such that $N = PQ$. The central idea of this method is that we begin with a random starting integer, and apply it to a polynomial function. Then, the function is iterated mod N , starting with x , then $f(x)$, then $f(f(x))$, and so on indefinitely.

Since we're doing this iteration in mod N , we are guaranteed to eventually reach a value we have already found before. Once we reach that point, we know that the series of values calculated by this iteration will 'cycle around' in a repeating series of values, as $f(x)$ will always have the same output given an identical input.

We create two separate processes: one of which that iterates the function, and another that iterates two times for one round of iteration performed by the other. These are known as the 'Tortoise' and the 'Hare' respectively. Because they are effectively on a circular loop, eventually the 'Hare' will 'catch up' to the Tortoise. This will happen in any modular system, due to a bounded number of possible values. This will happen both in mod N and in mod P , even if we don't initially know what P happens to be. In fact, by using the Chinese Remainder Theorem, we can use this 'looping' property to solve our problem. When the Tortoise and the Hare are at the same value mod P and not mod N , their difference will be $0 \pmod{P}$ but not $0 \pmod{N}$. In fact, the difference will be a multiple of P ! We can then find its greatest common divisor with N , by using Euclid's Algorithm, and thereby recover P , and therefore Q .

It's impossible to discern when the Hare has caught up to the Tortoise in mod P , simply because we don't yet know what P is! However, if we perform Euclid's algorithm on the difference of every round of iteration, we will be able to find the GCD between our Hare-Tortoise difference and N . On an iteration where the GCD isn't 1, we know we've found a factor of N .

The procedure won't always result in a success. One possible problem is that the difference will be $0 \pmod{P}$ and $0 \pmod{Q}$ on the same iteration, meaning that no conclusions can be drawn. Another problem is that the number of iterations required to

factor N could potentially be quite large with a given combination of N, the starting value of x to feed into f(x), and the polynomial function f(x). The solution to both problems is the same: pick another function and/or another starting value.

21.3 An Example Spreadsheet

Fortunately, this is a very easy problem to do with a spreadsheet, because a given iteration depends only on the previous iteration. Therefore, dragging a row down to copy will produce as much iteration as is needed to find a result. Let's demonstrate with an example.

Let's say we want to factor 2047, and use a starting value of 1 and let $f(x) = x^2 + x + 1$.

We put 2047 into cell A1, and our start value in A2. In B2 we put =A2, and in C2 we put =mod(b2*b2+b2+1,\$a\$1). For A3 we put =mod(a2*a2+a2+1,\$a\$1) and in b3 put =mod(c2*c2+c2+1,\$a\$1). These three columns form our Tortoise and our Hare: the A column is the Hare, and in each new row it takes the previous row's value as the input to f(x). The B and C column each calculate f(x) once, thus 'racing ahead' of the A column.

To perform Euclid's algorithm on the difference between the Tortoise and the Hare, we first copy our number to factor in every row of the D column (i.e. =\$a\$1), and in E2 put =mod(abs(b2-a2),\$a\$1). Now, we write into F2 =mod(d2,e2), and copy that cell to the right as far as is needed. For every row, you'll know how far to copy F2, because whenever a 0 is reached, Euclid's Algorithm is done.

Copy D2 through J2 (or however many columns you copied F2 over) into D3 through J3, and the first two rounds of the problem are complete. Now, to continue, just copy down the third row to as many new rows are needed to until there is a value other than 1 right before the 0 in one of the rows. With our example problem, here is how a spreadsheet might look:

A	B	C	D	E	F	G	H	I	J	K
2047										
1	1	3	2047	0	##	##	##	##	##	##
3	13	183	2047	10	7	3	1	0	##	##
13	921	1705	2047	908	231	215	16	7	2	1
183	1991	1034	2047	1808	239	135	104	31	11	9
921	1657	233	2047	736	575	161	92	69	23	0
1705	1301	1034	2047	404	27	26	1	0	##	##
1991	1657	233	2047	334	43	33	10	3	1	0
1034	1301	1034	2047	267	178	89	0	##	##	##
1657	1657	233	2047	0	##	##	##	##	##	##
233	1301	1034	2047	1068	979	89	0	##	##	##
1301	1657	233	2047	356	267	89	0	##	##	##
1034	1301	1034	2047	267	178	89	0	##	##	##
1657	1657	233	2047	0	##	##	##	##	##	##
233	1301	1034	2047	1068	979	89	0	##	##	##
1301	1657	233	2047	356	267	89	0	##	##	##

1034	1301	1034	2047	267	178	89	0	##	##	##
1657	1657	233	2047	0	##	##	##	##	##	##
233	1301	1034	2047	1068	979	89	0	##	##	##
1301	1657	233	2047	356	267	89	0	##	##	##
1034	1301	1034	2047	267	178	89	0	##	##	##

And on the 8th iteration, we find our factor: $2047 = 89 * 23$. The process completed in 8 iterations, while $2047^{1/4} = 6.72$, so our estimation of effort required to factor the number (i.e. the number of iterations is close to $N^{1/4}$, as we said in the beginning of this section) is fairly accurate.

- Scotty Ostler