Lecture 17 - Pollard's factoring algorithms

This lecture (actually 2 in class) concern two clever factoring algorithms introduced by J. Pollard. The first, Pollard's ρ algorithm for integer factorization (as distinct from his ρ algorithm for discrete logarithms) factors an integer n in time $O\left(n^{1/4}\right)$, as opposed to the naive $O\left(n^{1/2}\right)$ trial division algorithm. The second is a special purpose algorithm, and only works well when $p-1$ is smooth for one of the factors p of the integer to be factored.

The idea of both algorithms is to hypothetically consider a factor p of n, and derive properties it satisfies from some experimentation. Enough clues are found to obtain the value of p.

Pollard's ρ algorithm for integer factorization

Let p be the smallest factor of an integer n (p is unknown, of course). We consider a random-looking function that maps integers mod n to other integers mod n. The classic example is to take the polynomial $f(x) = x^2 + 5$ and iterate it.

Let $x_0 = 1$, $x_1 = f(x_0)$, $x_2 = f(x_1)$, ..., $x_{k+1} = f(x_k)$, ...

The Birthday Paradox tells us that it is unlikely two of the values will overlap until we have iterated this function about $n^{1/2}$ times. However, the same Birthday Paradox tells us we do expect an overlap of values modulo the prime p within about $p^{1/2} \le n^{1/4}$ steps. Suppose that $x_i = x_j$ is such a collision (mod p). Of course we don't know which i and j these are, but if we took the GCD $\left(x_i - x_j, n\right)$ it would very likely be p. After all, it has to be a divisor of n, and p divides both. It's not likely to include a bigger factor, such as n itself, because it would mean a collision mod n (which we don't expect to happen until much later on).

The algorithm thus tests for such collisions. However, storing them all and comparing them all would be prohibitive. Instead, it uses the fact that once there is a collision of values $x_i = x_j$, then there are collisions every successive step:

$x_{i+1} = x_{j+1}$

$x_{i+2} = x_{j+2}$

$x_{i+3} = x_{j+3}$

etc ...

We don't need to see the first collision, just *some* collision. In general, $x_r = x_{r+|i-j|}$ for any sufficiently large r. In particular,

for sufficiently large multiples k of $|i-j|$ we have $x_k = x_{2k}$. We thus only look at differences of these (which requires little storage), and take GCD $(x_k - x_{2k}, n)$.

Here is an example:

```
f[x_] = x^2 + 5
```

$5 + x^2$

```
p = Prime[4]; q = Prime[6]; n = p q
```

91

```
(* This is the sequence *)
tab = {1}; Do[AppendTo[tab, Mod[f[tab[[-1]]], n]], {100}]; tab
```

{1, 6, 41, 48, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34,
69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34,
69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34,
69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34,
69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34, 69, 34}

```
(* These are the GCDs *)
Table[GCD[%[[2 i]] - %[[i]], n], {i, 50}]
```

{1, 7, 7, 7, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91,
7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91, 7, 91}

We see 7 is pulled out .

Here is a much bigger number, but with a moderately - sized smallest prime factor. This determines how long the algorithm takes.

```
p = Prime[100]; q = Prime[10 000 000 000]; n = p q
```

136 384 910 137 043

```
tab = {1}; Do[AppendTo[tab, Mod[f[tab[[-1]]], n]], {100}]; tab
```

```
{1, 6, 41, 1686, 2 842 601, 8 080 380 445 206, 42 348 633 296 828,
 66 248 037 349 051, 85 439 130 607 169, 27 959 685 793 612, 85 026 439 508 660,
 70 981 221 085 653, 25 524 420 934 229, 19 246 918 738 220, 90 481 766 497 303,
 55 236 837 149 244, 26 061 652 572 533, 58 603 176 830 026, 69 777 371 705 792,
 111 063 517 850 050, 27 436 601 737 489, 115 863 518 811 119,
 39 320 633 105 332, 30 350 209 272 546, 60 742 946 525 903, 8 338 376 143 609,
 68 868 770 337 758, 65 368 801 576 721, 129 304 919 759 788, 40 093 108 851 508,
 68 551 016 435 065, 9 278 382 097 111, 33 583 975 588 429, 66 329 131 852 855,
 123 282 519 212 217, 46 416 095 006 884, 28 571 041 176 555, 125 494 569 812 595,
 38 657 266 017 549, 116 713 226 908 005, 57 354 902 292 247, 28 986 968 258 024,
 120 202 048 535 362, 73 571 275 933 896, 41 164 018 495 673, 35 883 548 822 288,
 82 141 815 369 081, 13 948 745 441 816, 62 572 216 834 662, 66 473 367 275 961,
 32 483 287 007 437, 65 090 741 700 067, 81 711 829 010 537, 73 924 464 261 211,
 12 706 662 628 149, 108 806 730 890 289, 107 927 571 373 432,
 23 507 560 373 191, 91 081 374 158 003, 43 661 686 269 449, 131 234 796 041 487,
 12 826 885 039 212, 75 613 877 347 894, 68 737 434 530 529, 130 703 700 790 660,
 98 148 592 223 992, 43 027 020 119 298, 3 266 052 257 507, 98 200 479 133 911,
 92 149 788 878 773, 108 437 721 462 279, 68 854 698 137 098, 123 775 102 042 856,
 30 552 599 847 333, 75 411 812 062 594, 33 949 652 622 247, 60 134 367 248 840,
 132 710 641 621 512, 85 697 078 013 141, 37 335 613 180 209, 43 074 645 668 974,
 81 297 779 073 005, 49 598 489 091 086, 42 368 491 378 683, 68 164 047 918 741,
 1 286 738 735 799, 128 839 597 493 040, 10 510 468 426 379, 49 813 875 613 605,
 70 606 956 332 173, 45 233 580 029 363, 135 440 475 770 683, 493 274 602 480,
 21 303 642 235 374, 28 082 906 449 539, 8 777 858 413 067, 115 812 916 894 354,
 11 607 742 488 667, 80 861 309 465 793, 72 941 905 430 238, 57 948 894 377 265}
```

```
Table[GCD[%[[2 i]] - %[[i]], n], {i, 50}]
```

```
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 541, 1, 1, 1, 1, 1, 541, 1, 1, 1, 1, 1, 541, 1, 1, 1, 1, 1, 541, 1, 1}
```

Sure enough, 541 is a prime factor!

Pollard' s p - 1 algorithm

As we mentioned above, this algorithm only works well when p - 1 is *smooth*, meaning that it is the product of small primes (exactly how small can be quantified in terms of its running time). It uses the same idea of trying to find two numbers which are equal mod p, but very unlikely to be equal mod n. (As before, we don' t know p, so this method doesn' t work very well when the numbers are random.)

Let us recall the generalization of Fermat' s Little Theorem to composite numbers, $a^{\phi(m)}$ is congruent to 1 modulo n, if a and n are relatively prime. In practice,

we can assume a and n are relative prime,
for if they had a common factor, we'd've already factored n.

For simplicity, let's look at n = p q,
where p and q are both primes. It's easiest to motivate this with an example,
with p = 37 (here $p-1 = 2^2\ 3^2$). Let's think of a number a mod p as g^x,
where x is the exponent of a generator g (mod p). Then a^2 is $g^{2\,x}$ mod p,
and repeated squaring gives us $a^4 = g^{2^2\,x}$ (mod p). Since x is determined mod p − 1 = 36,
this tells us that $a^4 = g^y$, where y is a multiple of 2^2. Now by repeated cubing, we get $a^{12} = g^{3\,y}$ and $a^{36} = g^{3^2\,y}$. So we'd get a multiple of 36 by some repeated squarings and cubings.

Of course, here we *knew* the value of p − 1 to begin with. However, even if we didn't,
we could do some repeated squarings and cubings to get it quickly -- simply because p − 1 is smooth, and hence a product of small powers of 2 and small powers of 3. It doesn't take too many tries to find all of them: if $p-1 = 2^x\ 3^y$,
you never need to try x bigger than the logarithm of p to the base 2,
or try y bigger than the logarithm of p to the base 3. If you threw in more small primes (5, 7, 11, ...) this wouldn't change much -- as long as you didn't throw in too many more,
it still wouldn't take too many combinations to hit it.

Of course, in practice, we don't know p − 1,
or even its factorization. But if it is indeed smooth,
then we could take some random number a and keep taking small powers of it,
hoping to get back to the identity. Then by looking at the GCD of this power − 1,
we'd expect to get p.

In this example, I picked the 1076 th prime because it, minus 1,
doesn't have a factor bigger than 5. Again, in practice you would not know this!

```
n = Prime[1076] Prime[351 321]
```

```
43 581 116 807
```

```
a = 2; Table[a = PowerMod[a, k, n]; a, {k, 1, 10}]
```

```
{2, 4, 64, 16 777 216, 25 378 515 287, 17 949 771 412,
 36 281 609 707, 8 408 577 954, 23 269 452 593, 10 078 534 043}
```

```
a = 2; Table[a = PowerMod[a, k, n]; GCD[a - 1, n], {k, 1, 10}]
```

```
{1, 1, 1, 1, 1, 1, 1, 1, 8641, 8641}
```

```
FactorInteger[n]
```

```
{{8641, 1}, {5 043 527, 1}}
```

```
n = Prime[30] Prime[10]
```

```
3277
```

```
a = 2; Table[a = PowerMod[a, k, n]; a, {k, 1, 10}]
```

```
{2, 4, 64, 2253, 256, 3213, 1, 1, 1, 1}
```

```
a = 2; Table[a = PowerMod[a, k, n]; GCD[a - 1, n], {k, 1, 10}]
```

```
{1, 1, 1, 1, 1, 1, 3277, 3277, 3277, 3277}
```

```
FactorInteger[n]
```

```
{{29, 1}, {113, 1}}
```