Intro to Algorithms HW 3

Greg Stewart

February 9, 2018

Q1

Problem. Show whether $4^{1536} \equiv 9^{4824} \mod 35$

Well first off, for any prime p, we know $x^{p-1} \equiv 1 \mod p$. 35 is not prime, but its factorization is $5 \cdot 7$, and we can split up the statement so

$$x^{5-1} \equiv 1 \mod 5 \text{ and } x^{7-1} \equiv 1 \mod 7$$

From this we can see that

$$(x^{5-1})^{7-1} = x^{24} \equiv 1 \mod (5 \cdot 7) = 1 \mod 35$$

for all x such that $1 \leq x < 35$. Now all we need to do is factor the exponents a bit.

$$4^{1536} = 4^{24 \cdot 64} \equiv 1 \mod 35$$

and

$$9^{4824} = 9^{24 \cdot 201} \equiv 1 \mod 35$$

So finally we can see that

$$4^{1536} \equiv 9^{4824} \mod 35$$

Q2

Problem. Solve $x^{86} \equiv 6 \mod 29$

From Fermat's Little Theorem,

$$x^{28} \equiv 1 \mod 29$$

so we have

$$x^{86} \equiv x^2 \mod 29$$

and using the original problem,

$$x^2 \equiv 6 \mod 29$$

which is fortunately similar to

$$x^2 \equiv 64 \mod 29$$

and means that we can write

$$x^2 - 64 \equiv 0 \mod 29$$

$$(x-8)(x+8) = 0 \mod 29$$

which has solutions $x = 8$ and $x = 21$ so

$$x \equiv 8, 21 \mod 29$$

Q3

Prove that $\gcd(F_{n+1}, F_n) = 1$ for $n \geq 1$ where F_n is the n-th Fibonacci element.

The $n = 1$ case is clearly true, as $\gcd(1, 1) = 1$. Now suppose that the statement is true for some $n \geq 1$, so $\gcd(f_{n+1}, f_n) = 1$. Then for $n + 1$ we have

$$\gcd(f_{n+2}, f_{n+1}) = \gcd(f_{n+1}, f_{n+2} - f_{n+1})$$

which, by the definition of a Fibonacci number, is just

$$\gcd(F_{n+1}, F_n) = 1$$

So we have shown that $\gcd(F_{n+1}, F_n) = 1$.

Q4

Multiplying n-bit by an m-bit integer is $O(nm)$. Given x and y, give an efficient algorithm to compute x^y.

The algorithm is as follows:

```
function power(x, y):
  input: base x and exponent y
  output: x^y

  if y = 0:
    return 1
  if y = 1:
    return x
  if y is even:
    return power(x*x, y/2)
  else
    return x*power(x*x, (n-1)/2)
```

The problem of x^y can be broken down into products of x^2, essentially. For example, $x^8 = (x^2)^{8/2} = ((x^2)^2)^2$. In general, if y is even, then

$$x^y = (x^2)^{y/2}$$

and if y is odd, then

$$x^y = x(x^2)^{(n-1)/2}$$

Unless $y = 0$ or $y = 1$, this is exactly what the above algorithm does, recursively dividing y by 2 for each call. So this is a sort of divide-and-conquer algorithm. There are $\log y$ squarings of x and in the worst case $\log y$ multiplications. In this case y is an m-bit number, so $\log y = m$. Squaring the n-bit x is $O(n^2)$, done m times, so we're at $O(mn^2)$, and since there are m multiplications we finally end up with

$$O((mn)^2)$$