University of Sydney

MATH 2988

Number Theory and Cryptography

Assignment 2

Author: Keegan Gyoery
SID: 470413467

Lecturer: Dmitry Badziahin
Tutorial: Carslaw Tutorial Room 451
Thursday 12pm

October 26, 2017

1. (a) We are required to find an integer $x \in \{0, \dots, 196\}$, which is a solution to the equation

$$x^{131} \equiv 12 \pmod{197}$$

In order to solve the congruence relation, we need to find some $u \in \mathbb{Z}^+$ such that $131u \equiv 1 \pmod{196}$. By finding such a u, we are able to reduce the congruence relation to a linear realtionship.

$$\begin{aligned} 131u &\equiv 1 \pmod{196} \\ 131^{-1} \cdot 131u &\equiv 131^{-1} \pmod{196} \\ u &\equiv 131^{-1} \pmod{196} \end{aligned}$$

We are now required to find the inverse of $131 \pmod{196}$. In order to do so we use the Extended Euclidean Algorithm.

196	131	65	1
·	·	1	2
0	1	1^-	3
1	0	1	2^-

As a result of the EEA, we have the result that $3 \cdot 131 - 2 \cdot 196 = 1$. It is thus clear that $131^{-1} \equiv 3 \pmod{196}$.

$$\begin{aligned} 131^{-1} &\equiv 3 \pmod{196} \\ \therefore u &\equiv 131^{-1} \pmod{196} \\ \therefore u &\equiv 3 \pmod{196} \end{aligned}$$

Using these results, we are able to reduce the original congruence to a linear relation that is simple to solve.

$$\begin{aligned} x^{131} &\equiv 12 \pmod{197} \\ \therefore x^{131u} &\equiv 12^u \pmod{197} \\ \therefore x^{131 \cdot 3} &\equiv 12^3 \pmod{197} \\ \therefore x^1 &\equiv 12^3 \pmod{197} \\ \therefore x &\equiv 1728 \pmod{197} \\ \therefore x &\equiv 152 \pmod{197} \end{aligned}$$

Thus the integer $x = 152$ solves the congruence relation $x^{131} \equiv 12 \pmod{197}$.

(b) We are now required to show that for any prime $p > 3$ there exists $n \in \{2, 3, \ldots, p-2\}$ such that the polynomial equation

$$x^n \equiv x + 1 \pmod{p}$$

has an integer solution. Let $a \in \mathbb{Z}+$ be a primitve root modulo p. As a consequence of the definiton of a primitive root, we have the result that $a \in \{2, 3, \ldots, p-2\}$. Furthermore, for some i, where $i \in \{0, 1, 2, \ldots, p-2\}$, $a^i \pmod{p}$ generates all residues modulo p. That is, $a^i \in \{2, 3, \ldots, p-1\}$. Furthermore, based on the definition of the set that a resides in, we have the result that $a + 1 \in \{3, 4, \ldots, p-1\}$. Thus, we have the two sets as results.

$$\begin{aligned} a^i \pmod{p} &\in \{2, 3, \ldots, p-1\} \ldots\ldots\ldots (A) \\ a + 1 &\in \{3, 4, \ldots, p-1\} \ldots\ldots\ldots (B) \end{aligned}$$

If $p - 1 \geq 3$, then the sets (A) and (B) overlap. Thus, $p > 3$. As a result, for given a, we can select such an $i \in \{0, 1, 2, \ldots, p-2\}$ such that the following result holds true.

$$a^i \equiv a + 1 \pmod{p}$$

We now must prove that the result does not hold for $i = 0$ and $i = 1$, in order to prove the required result. In the case where $i = 0$, we have the following results.

$$\begin{aligned} a^i &\equiv a^0 \pmod{p} \\ &\equiv 1 \end{aligned}$$

Thus, we must examine the congruence relation $1 \equiv a + 1 \pmod{p}$. For $a + 1$ to be congruent to 1, $a = 0$. If $a = 0$, it does not exist in the set of values that a can take, that is $a \in \{2, 3, \ldots, p-2\}$. Thus, i cannot take the value $i = 0$.

In the case where $i = 1$, we have the following results.

$$\begin{aligned} a^i &\equiv a^1 \pmod{p} \\ &\equiv a \end{aligned}$$

Thus, we must examine the congruence relation $a \equiv a + 1 \pmod{p}$. There are no values of a such that this result holds, as $0 \not\equiv 1 \pmod{p}$. Thus, i cannot take the value $i = 1$.

As a result, we eliminate the values $i = 0$, and $i = 1$ from the set of values that i can take. Thus the set $i \in \{0, 1, 2, \ldots, p-2\}$ is reduced to the set $i \in \{2, 3, \ldots, p-2\}$. Thus, we have an integer solution to the congruence relation

$$a^i \equiv a + 1 \pmod{p}$$

where $a \in \mathbb{Z}+$ is a primitive root, $a \in \{2, 3, \ldots, p-2\}$, and $i \in \{2, 3, \ldots, p-2\}$. Thus by a dummy variable swtich, we have the desired result, that

$$x^n \equiv x + 1 \pmod{p}$$

has an integer solution, where $n \in \{2, 3, \ldots, p-2\}$, and $p > 3$.

2. (a) Let $n \in \mathbb{Z}^+$. We are required to show that for any factorisation of n^2+1 as a product of two positive integers, that is $n^2+1 = a \cdot b$ where $a, b \in \mathbb{Z}^+$, we have the following result.

$$|b-a| \geq \sqrt{4n-3}$$

In order to complete this proof, we represent n^2+1 as a difference of two squares, that is, $n^2+1 = r^2 - s^2$, where $r, s \in \mathbb{Z}$. Thus we have the following results.

$$\begin{aligned}
r^2 - s^2 &= n^2+1 \\
r^2 &= n^2+1+s^2 \\
\therefore r^2 &> n^2+1 \quad \text{as} \quad s^2 > 0 \\
r &> \sqrt{n^2+1} \\
r &> \sqrt{n^2} \quad \text{as} \quad n^2+1 > n^2 \\
\therefore r &> n \,.\,.\,.\,.\,.\,.\,.\,.\, (1)
\end{aligned}$$

We now examine the factorisation of $n^2+1 = r^2 - s^2$ to get the following result.

$$\begin{aligned}
n^2+1 &= r^2 - s^2 \\
&= (r+s)(r-s) \\
\therefore a \cdot b &= (r+s)(r-s) \\
a &= r+s \quad \text{as} \quad a, r, s \in \mathbb{Z} \\
b &= r-s \quad \text{as} \quad b, r, s \in \mathbb{Z} \\
\therefore a+b &= r+s+r-s \\
&= 2r \\
\therefore r &= \frac{a+b}{2} \\
\therefore r &= \frac{k}{2} \quad \text{where} \quad k \in \mathbb{Z} \,.\,.\,.\,.\,.\, (2)
\end{aligned}$$

We now have the two results, (1) and (2), that allow us to arrive at the following result.

$$\begin{aligned}
r &> n \quad \text{from} \quad (1) \\
r &= \frac{k}{2} \quad \text{from} \quad (2) \\
\therefore r &\geq n + \frac{1}{2} \,.\,.\,.\,.\,.\,.\,.\,.\, (3)
\end{aligned}$$

The result (3) arises trivially from the fact that r must be of the form $\frac{k}{2}$, where $k \in \mathbb{Z}^+$.

We now reconsider our representation of $n^2+1=r^2-s^2$. Thus we derive the following results.

$$\begin{aligned}
n^2+1 &= r^2-s^2\\
\therefore n^2+1 &\geq \left(n+\frac{1}{2}\right)^2-s^2\\
\therefore n^2+1 &\geq n^2+n+\frac{1}{4}-s^2\\
\therefore 1 &\geq n+\frac{1}{4}-s^2\\
\therefore s^2 &\geq n+\frac{1}{4}-1\\
s^2 &\geq n-\frac{3}{4}\\
s^2 &\geq \frac{1}{4}(4n-3)\\
\therefore s &\geq \frac{1}{2}\sqrt{4n-3}\ldots\ldots\ldots(4)
\end{aligned}$$

Using result (4), and the absolute value of the difference between a and b, we get the final, desired result.

$$\begin{aligned}
|b-a| &= |r-s-(r+s)|\\
&= |-2s|\\
&= 2s\\
s &\geq \frac{1}{2}\sqrt{4n-3}\\
\therefore 2s &\geq \sqrt{4n-3}\\
\therefore |b-a| &\geq \sqrt{4n-3}
\end{aligned}$$

(b) We are required to find a polynomial time algorithm, which, given $n \in \mathbb{Z}^+$, finds an integer m such that $|m^3 - n|$ is as minimal as possible.

Firstly, we need to explain what the algorithm does and how it finds the integer m. Furthermore, we have the fact that $m \in [0, n]$, as $\sqrt[3]{n} \leq n$. The algorithm works on the principle of a binary search, by halving the interval in which m can reside, and thus narrowing the search for m drastically at each iteration. In order to find m, the algorithm performs the following steps.

1. Compute $k = \lceil \log_2 n \rceil$, by successive multiplications of 2 and tracking the number of such multiplications, in order to set the maximum iterations the alogrithm must cycle through.
2. Establish the initial values of the interval in which m resides. That is, set the upper bound (up) of the interval to n, and the lower bound (low) to 0.
3. Set $m = \dfrac{\text{up} + \text{low}}{2}$.
4. Compute the value (val) $m^3 - n$.
5. Check if val is greater or less than 0. If val is greater than 0, set up equal to the current value of m. If val is less than 0, set low equal to the current value of m.
6. Repeat steps 3 to 5 k times.
7. Set $s = \left\lfloor \dfrac{\text{up} + \text{low}}{2} \right\rfloor$.
8. Compute the value under as $|s^3 - n|$, and the value over as $|(s+1)^3 - n|$. These values s and $s + 1$ are the two possible values for m, such that $|m^3 - n|$ is minimised.
9. Check if over is greater than under, and if so, set $m = s$. If under is greater than over, set $m = s + 1$.

The number of iterations that the algorithm must run through has been stated as k. This result comes from the following proof. Consider a k bit number, n. Thus, in the maximum case, $n = 2^k$. By our method of halving the interval, effectively a binary search, we must figure out the number of divisions of the interval, that is the number of iterations of the loop inside the algorithm, to produce an interval that contains only 1 integer. Thus, in order to satisfy this condition of an interval containing a single integer, we have the result,

$$\frac{n}{2^x} = 1$$

where x is the number of divisions of the interval, that is iterations of the loop, required.

$$\begin{aligned} \frac{n}{2^x} &= 1 \\ \frac{2^k}{2^x} &= 1 \\ \therefore 2^x &= 2^k \\ \therefore x &= k \end{aligned}$$

As a result, the maximum number of iterations that the loop inside the algorithm must perform in order to narrow the interval down to contain a single integer is given by k, the number of bits of n.

Furthermore, $s = \left\lfloor \sqrt[3]{n} \right\rfloor$, and $s + 1 = \left\lceil \sqrt[3]{n} \right\rceil$. Furthermore, we have the results

$$\begin{aligned} \left\lfloor \sqrt[3]{n} \right\rfloor^3 &\leq n \leq \left\lceil \sqrt[3]{n} \right\rceil^3 \\ \therefore s^3 &\leq n \leq (s+1)^3 \end{aligned}$$

As a result, m must be either s or $s + 1$, as they are the closest integers to $\sqrt[3]{n}$. This justifies the final steps of the algorithm to choose between s or $s + 1$. This ensures that the algorithm will always work.

Complexity of Arithmetic Operations

(a). Long multiplication of two numbers, with k and l bits each, requires at most kl bit operations.
(b). Long division of two numbers, with k and l bits each, requires at most kl bit operations.
(c). Addition of two numbers with k and l bits each, where $k > l$, requires at most k bit operations.
(d). Subtraction of two numbers with k and l bits each, where $k > l$, requires at most k bit operations.
(e). Comparisons for k bit numbers, in condition checking statements, require at most k bit operations, due to a maximum of k bits having to be compared.
(f). Squaring a k bit number produces a $2k$ bit number, as $\left(2^k\right)^2 = 2^{2k}$. Squaring requires at most k^2 bit operations by point (a).
(g). Thus, cubing a k bit number requires multiplying the squared number, which has $2k$ bits, by itself, that is $2k \cdot k = 2k^2$ bit operations. Cubing produces a $3k$ bit number, as $\left(2^k\right)^3 = 2^{3k}$.

Computational Complexity
In order to determine the computational complexity of the algorithm, we must first determine the bit operations of each stage of the algorithm, based off the number of bits of the input number, n. Assuming that n is a k bit number, such that $n = (b_{k-1}b_{k-2}\dots b_0)_2$, where $b_i \in \{0, 1\}$, the upper bound on n is $n < 2^k$. Thus taking the worst case scenario, $n = 2^k$. Furthermore, we know that as $m < n$, the number of bits in the number of m is less than k. In order to compute the computational complexity, we need to examine the bit operations at each step of the algorithm.

1. At most k^2 bit operations to compute $2 \cdot 2 \dots$, with k multiplications. At most k bit operations to compare the value of the variable log to the given input n, at each iteration of the loop. With k iterations, it has a total of at most k^2 bit operations for the comaprisons
2. No bit operations required.
3. At most k bit operations for the addition of up and low, and at most k bit operations for the division, as by point (b), the maximum bit operations is kl, where in this case, 2 is a single bit number, and thus $l = 1$.
4. At most $3k^2$ bit operations for the cubing of m, and at most $3k$ bit operations for the subtraction $m^3 - n$.
5. At most $2k$ bit operations for the comparisons in the if statements, for the variable val.
6. k repetitions.
7. At most k bit operations for the addition of up and low, and at most k bit operations for the division, as by point (b), the maximum bit operations is kl, where in this case, 2 is a single bit number, and thus $l = 1$. Magma's inbuilt floor function is polynomial time, and adds no bit operations.
8. At most $3k^2$ bit operations for the cubing of s, and at most $3k$ bit operations for the subtraction $s^3 - n$. At most $3k^2$ bit operations for the cubing of $s + 1$, and at most $3k$ bit operations for the subtraction $(s + 1)^3 - n$.
9. At most $2k$ bit operations for the final comparisons.

From the bit operations calculated at each stage, we have the following function, $f_c(k)$, to model the computational complexity of the algorithm.

$$\begin{aligned}
f_c(k) &= k^2 + k^2 + (k + k + 3k^2 + 3k + 2k) \cdot k + k + k + k + 3k^2 + 3k + 3k^2 + 3k + 2k \\
&= (3k^2 + 7k) \cdot k + 8k^2 + 10k \\
&= 3k^3 + 7k^2 + 8k^2 + 10k \\
&= 3k^3 + 15k^2 + 10k \\
\therefore f_c(k) &= 3k^3 + 15k^2 + 10k
\end{aligned}$$

In order to prove the algorithm is indeed polynomial time, we must show that $f_c(k)$ is $O(k^\alpha)$, for some $\alpha \in \mathbb{Z}$. We first need to define O notation. A function, $f(k)$ is $O(k)$ if

$$f(k) \leq C \cdot g(k) \quad \forall\, k \geq N$$

where $C, N \in \mathbb{Z}^+$. Thus, we need to find such an N, C, and $g(k)$. We thus look for an N, through the following inequalities.

$$\begin{aligned}
3k^3 &\geq 15k^2 \\
3k &\geq 15 \\
\therefore k &\geq 5 \ldots\ldots\ldots (A) \\
3k^3 &\geq 10k \\
3k^2 &\geq 8 \\
k^2 &\geq \frac{8}{3} \\
\therefore k &\geq \frac{2\sqrt{2}}{\sqrt{3}} \\
\therefore k &\geq 2 \ldots\ldots\ldots (B)
\end{aligned}$$

From these two results, (A) and (B), it is clear that $k \geq 5$, and thus we have the existence of an N, where $N = 5$ in this case. Thus we have the following results.

$$\begin{aligned}
f_c(k) &= 3k^3 + 15k^2 + 10k \\
\therefore f_c(k) &\leq 3k^3 + 3k^3 + 3k^3 \quad \forall\, k \geq 4 \\
&= 9k^3 \\
\therefore f_c(k) &\leq 9k^3
\end{aligned}$$

As a result, we have the existence of a C, where $C = 9$, and a $g(k)$, where $g(k) = k^3$. As a result, we have that $f_c(k)$ is $O(k^3)$. Thus the algorithm is polynomial time. Upon inputting large values of n, the algorithm behaves like a polynomial time algorithm is expected to, thus backing up the proof. In the following section, the magma code for the algorithm is provided.

Magma Code

```
cube:=procedure(n)
    log:=1;
    k:=0;
    while log lt n do
        log:=log*2;
        k:=k + 1;
    end while;
    up:=n;
    low:=0;
    for i:=1 to k do
        m:=(up+low) div 2;
        val:=m^3 - n;
        if val gt 0 then
            up:=m;
        end if;
        if val lt 0 then
            low:=m;
        end if;
    end for;
    s:=Floor((up+low) div 2);
    under:=AbsoluteValue(s^3 - n);
    over:=AbsoluteValue((s+1)^3 - n);
    if over - under gt 0 then
        print s;
    end if;
    if over - under lt 0 then
        print(s+1);
    end if;
end procedure;
```