EECS 70 Discrete Mathematics and Probability Theory
Fall 2014 Anant Sahai

Homework 5

This homework is due October 6, 2014, at 12:00 noon.

1. **Modular Arithmetic Lab (continue)**

Oystein Ore described a puzzle with a dramatic element from Brahma-Sphuta-Siddhanta (Brahma's Correct System) by Brahmagupta (born 598 AD) as follows.

An old woman goes to market and a horse steps on her basket and crushes the eggs. The rider offers to pay for the damages and asks her how many eggs she had brought. She does not remember the exact number, but when she had taken them out two at a time, there was one egg left. The same happened when she picked them out three, four, five, and six at a time, but when she took them seven at a time they came out even. What is the smallest number of eggs she could have had?

In the first part of this week's Virtual Lab, we will implement the functions to solve the above puzzle based on the Chinese Remainder Theorem. In the second part, we will explore two very useful theorems in modular arithmetic: Fermat's Little Theorem and Euler's Theorem.

Please download the IPython starter code from Piazza or the course webpage, and answer the following questions.

We will reuse a lot of functions implemented in last week's lab. You will be given sample implementations of these functions; there is no need to copy and paste your work from last week.

(a) Implement the function `chinese_remainder` using the function `inverse(x, m)`, which computes the positive inverse of x modulo m. Given a list of tuples (a_i, n_i) where the n_i's are pairwise coprime, this function solves for x such that $x \equiv a_i \pmod{n_i}$ and $0 \le x < \prod(n_i)$.

(b) Write the system of congruences that describes the above puzzle, and solve it using `chinese_remainder`. Remember that the n_i's are pairwise coprime, and make sure to double check if your solution is in fact the smallest number of eggs that satisfies the congruences.

Solutions: The number of broken eggs, x, must satisfy:

$x \equiv 1 \bmod 2$
$x \equiv 1 \bmod 3$
$x \equiv 1 \bmod 4$
$x \equiv 1 \bmod 5$
$x \equiv 1 \bmod 6$
$x \equiv 0 \bmod 7$

Unfortunately, the moduli of the congruences are currently not pairwise coprime. As seen in Discussion 5M's problem 3, we can eliminate some redundant information.

First, $x \equiv 1 \bmod 4$ can be written as $x = 4k + 1$. This implies that x is odd, which means the first congruence, $x \equiv 1 \bmod 2$, is redundant.

Then, recall from Discussion 4M and 5M that we can break $x \equiv 1 \bmod 6$ into two congruences, mod 2 and mod 3, respectively, because $6 = 2 \cdot 3$. From $x \equiv 1 \bmod 6$, we have $x = 6k + 1$. This implies that x is odd (which we already know is redundant), and x modulo 3 will give us 1 (which we already have above). Hence, $x \equiv 1 \bmod 6$ is also redundant, and we can also eliminate this congruence before running `chinese_remainder`.

In summary, to use the Chinese Remainder Theorem, we will omit the congruences $x \equiv 1 \bmod 2$ and $x \equiv 1 \bmod 6$ so that the moduli of the remaining congruences (3, 4, 5 and 7) are relatively prime in pairs.

Using the function `chinese_remainder`, we can solve for $x = 301$ as follows:

```
chinese_remainder([(1, 3), (1, 4), (1, 5), (0, 7)])
```

As this value of x is odd and satisfies $x \equiv 1 \bmod 6$, it is the smallest solution of the broken eggs puzzle.

(c) In lecture, we studied Fermat's Little Theorem, which states that if p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p.

In part (d) of last week's Virtual Lab, you have implemented a naive way of testing whether a positive integer p is prime. Based on Fermat's Little Theorem, we can come up with a new primality test as follows.

For a randomly-chosen a, where $1 \le a \le p-1$, test if $a^{(p-1)} \equiv 1$ (mod p). If the equality does not hold, then p is composite. If the equality does hold, then we can say that p is a probable prime.

Your task is to implement the function `is_prime_fermat`, which basically conveys the idea described above. We will give you the starter code to choose a randomly.

Test your implementation and compare it with the one from last week. For example, what do you see when you run `is_prime(561)` v.s. `is_prime_fermat(561)`?

For those who are interested, 561 is a Carmichael Number, which is a composite number n that passes the Fermat Primality Test for all choices of a. Most composite numbers pass the test for very few choices of a, so by picking a few random a, you can detect them with high probability. The existence of Carmichael Numbers means that even though the test works (with high probability) for almost all numbers, it does not work for all numbers.

Solutions: `is_prime(561)` always returns False, whereas `is_prime_fermat(561)` occassionally returns True for some choices of a. (It returns True for every choice of a satisfying $1 < a < n$ and $gcd(a,n) = 1$, i.e. a and n are relatively prime).

(d) Consider the problem of trying to find the value of a tower of iterated exponents modulo some prime number p. We know from the running time of the modular exponentiation algorithm that this can be done quickly for expressions of the form

$$x^x \bmod p$$

Moreover, because of Fermat's Little Theorem (FLT), we know that we can even quickly calculate expressions of the form

$$x^{x^x} \bmod p$$

Here we use FLT to change the exponent to $x^x \bmod (p-1)$, which we can then calculate quickly using the modular exponentiation algorithm.

Great! But now what if we want to take this even higher? If our exponent was instead $x^{x^x} \bmod (p-1)$, what can we do? We can't use Fermat's Little Theorem here because there is no guarantee that $p-1$ is prime (in fact, unless $p = 3$, it's not). We have to calculate the exponent x^x without a modulus. If x is sufficiently large, we'll be working with numbers so big they'll fry your computer (try asking Python for `1000000**1000000`).

In order for calculations of this form to be tractable, we need some way to bubble the modulus up through the exponents so we can continue using the modular exponentiation algorithm.

In 1763, Euler generalized Fermat's Little Theorem to not require the modulus to be prime. Before we introduce Euler's Theorem though, we need to know about Euler's totient function.

Euler's totient function is defined as follows:

$$\phi(n) = |\{i : 1 \leq i \leq n, \texttt{gcd}(n,i) = 1\}|$$

In other words, $\phi(n)$ is the total number of positive integers less than n which are relatively prime to it, where 1 is counted as being relatively prime to all numbers. Since a number less than or equal to and relatively prime to a given number is called a *totative*, the totient function $\phi(n)$ can be simply defined as the number of totatives of n. For example, there are eight totatives of 24 $(1, 5, 7, 11, 13, 17, 19,$ and $23)$, so $\phi(24) = 8$.

Implement the function `totient`, which returns the number of positive integers less than n which are relatively prime to it. What can you conclude about the value of $\phi(n)$ when n is prime? Why does this make sense?

Solutions: When n is prime, $\phi(n) = n - 1$. This makes sense because all the numbers from 1 to $n-1$ are relatively prime to n.

(e) Euler's theorem states that for coprime a, n

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

This is very similar to Fermat's Little Theorem, but instead of requiring n to be prime, we only require that it is coprime to a, a looser condition. As a sanity check, verify that this works for prime n.

As with Fermat's Little Theorem, this theorem can be massaged into a more useful form with some algebra.

$$a^b \bmod n \equiv a^{b \bmod \phi(n)} \bmod n$$

Implement the function `tower` which, given x and some prime p, calculates the value of

$$x^{x^{x^x}} \bmod p$$

Hint: break the chain of exponents into three parts in a bottom-up approach using the formula above. Then, calculate each part in a top-down approach. Use the functions `mod_exp` *and* `totient` *defined in previous parts.*

Solutions: We first break the chain of exponents into three parts.

$$\begin{aligned} & x^{x^{x^x}} \bmod p \\ \equiv\ & x^{(x^{x^x} \bmod \phi(p))} \bmod p \\ \equiv\ & x^{(x^{x^x} \bmod (p-1))} \bmod p \\ \equiv\ & x^{(x^{(x^x \bmod \phi(p-1))} \bmod (p-1))} \bmod p \end{aligned}$$

We now calculate each individual part using `mod_exp`,

```
first = mod_exp(x, x, totient(p-1))
second = mod_exp(x, first, p-1)
```

, and simply return `mod_exp(x, second, p)` as the final answer.

Side note: It's perfectly fine if you went one step further and applied the formula to the innermost layer.

$$x^{(x^{(x^x \bmod \phi(p-1))} \bmod (p-1))} \bmod p$$

$$\equiv x^{(x^{(x^{(x \bmod \phi(\phi(p-1)))} \bmod \phi(p-1))} \bmod (p-1))} \bmod p$$

In that case, the code above would be modified as follows.

```
first = x % totient(totient(p-1))
second = mod_exp(x, first, totient(p-1))
third = mod_exp(x, second, p-1)
```

We would then return `mod_exp(x, third, p)` as the final answer.

We assumed for the implementation of `tower` *that x is coprime with* `totient(p) = p - 1`*. This doesn't hold generally, but the example given in the skeleton works because* 21 *is coprime with both* 101 *and* 100*.*

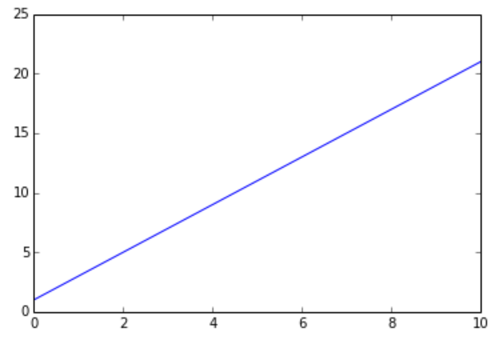
(f) This question aims to introduce you to basic plotting in Python with Matplotlib. First, we will generate 50 evenly spaced points from 0 to 10.

```
x = np.linspace(0, 10, 50)
```

Now let's say we want to plot the line $y = 2x + 1$. What should we do?

It's exactly how you would guess it...

```
y = 2*x + 1
plt.plot(x, y)
```

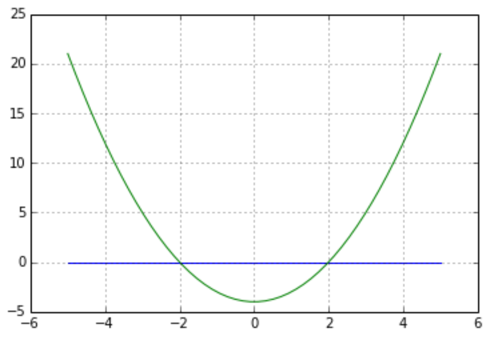


Now it's your turn. Generate 50 points between -5 and 5, then plot the curve $y = x^2 - 4$ as seen in Note 7. Add the x-axis ($y = 0$) to your plot. Can you spot the two roots of $y = x^2 - 4$?

In future labs, we will explore more powerful features of Matplotlib, as well as do more fun plotting with polynomials. Stay tuned!

Solutions:

```
x = np.linspace(-5, 5, 50)
y = x**2 - 4
x_axis = x * 0
plt.plot(x, x_axis)
plt.plot(x, y)
```



You can easily spot the two roots -2 and 2 of $y = x^2 - 4$ from the above graph.

Reminder: When you finish, don't forget to convert the notebook to pdf and merge it with your written homework. Please also zip the ipynb file and submit it as `hw5.zip`.

2. Counting by remainder

Over 1000 students walked out of class and marched to protest the war. To count the exact number of students protesting, the chief organizer lined the students up in columns of different length. If the students are arranged in columns of 2, 3, or 4, 1 person is left out. Also, we know the number students is a multiple of 5. What is the minimum number of students present?

Solutions: Since we know the number of students is a multiple of 5, let the number of students be $5x$. The problem statement allows us to write the system of congruences

$$\begin{aligned} 5x &\equiv 1 (\bmod 2) \\ 5x &\equiv 1 (\bmod 3) \\ 5x &\equiv 1 (\bmod 4). \end{aligned} \tag{1}$$

Similarly to 1(b), we know that $5x \equiv 1(\bmod 2)$ is redundant information given $5x \equiv 1(\bmod 4)$, since $5x = 4k+1$ implies $5x$ is odd. Omitting the relation $5x \equiv 1(\bmod 2)$, we know that the above equations result in $5x \equiv 1(\bmod 12)$, where we have used $12 = 3 \times 4$. Now, we can use the extended Euclidean algorithm to solve for the multiplicative inverse of 5 modulo 12 which we know exists, since $\gcd(12,5) = 1$. Running $\text{egcd}(12,5)$ gives us the values $d = 1$, $a = -2$, $b = 5$. (Note that to check your work, you would verify that $1 = -2(12) + 5(5)$). Therefore $x \equiv 5 \pmod{12}$.

Now, we know that the number of students is $5x > 1000$ where $x \equiv 5 \pmod{12}$. Thus, we we are looking for the smallest number that satisfies $x = 12t + 5$ where $x \geq 200$. Solving for t, we see $t > \lceil 195/12 \rceil = 17$. Thus, minimum number of students is $5 \times (12 \times 17 + 5) = 1045$.

3. Party Tricks

You are at a party celebrating your completion of the CS70 midterm. Show off your modular arithmetic skills and impress your friends by quickly figuring out the last digit(s) of each of the following numbers:

(a) Find the last digit of 11^{3142}.

Solutions: First, we notice that $11 \equiv 1 \pmod{10}$. So $11^{3142} \equiv 1^{3142} \equiv 1 \pmod{10}$, so the last digit is a 1.

(b) Find the last digit of 9^{9999}.

Solutions: 9 is its own multiplicative inverse mod 10, so $9^2 \equiv 1 \pmod{10}$. Then

$$9^{9999} = 9^{2(4999)} \cdot 9 \equiv 1^{4999} \cdot 9 \equiv 9 \pmod{10},$$

so the last digit is a 9.

Another solution: We know $9 \equiv -1 \pmod{10}$, so

$$9^{9999} \equiv (-1)^{9999} \equiv -1 \equiv 9 \pmod{10}.$$

You could have also used this to say

$$9^{9999} \equiv (-1)^{9998} \cdot 9 \equiv 9 \pmod{10}.$$

(c) Find the last 5 digits of the binary representation of 3^{1203}.

Solutions: We've previously learned that any integer n can be written as

$$n = c_k \cdot 2^k + c_{k-1} \cdot 2^{k-1} + \ldots + c_5 \cdot 2^5 + c_4 \cdot 2^4 + c_3 \cdot 2^3 + c_2 \cdot 2^2 + c_1 \cdot 2^1 + c_0 \cdot 2^0,$$

where the c_k's give the binary representation of n. So since we want to find the last 5 digits, that is, c_4, c_3, c_2, c_1, and c_0, we will look at the number $3^{1203} \mod (2^5 = 32)$ to find the remaining part $c_4 \cdot 2^4 + c_3 \cdot 2^3 + c_2 \cdot 2^2 + c_1 \cdot 2^1 + c_0 \cdot 2^0$.

Using Euler's Totient Function,

$$3^{16} \equiv 1 \pmod{32},$$

and since $1203 = 16(75) + 3$, we have

$$3^{1203} \equiv 3^{16(75)} \cdot 3^3 \equiv 1^{75} \cdot 3^3 \equiv 27 \pmod{32}.$$

So the answer will be the number 27 represented in binary. If we didn't know what the representation of 27 in binary was, we could do the following conversion:

$$\begin{aligned} 27/2 &= 13 \cdot 2 + \textcircled{1} \\ 13/2 &= 6 \cdot 2 + \textcircled{1} \\ 6/2 &= 3 \cdot 2 + \textcircled{0} \\ 3/2 &= 1 \cdot 1 + \textcircled{1} \\ 1/2 &= 0 \cdot 1 + \textcircled{1}, \end{aligned}$$

so looking at the remainders above, we see the last five digits of 3^{1203} represented in binary are 11011.

4. Fermat's Little Theorem

Fermat's Little Theorem in Lecture Note 6 [Theorem 6.1] states that for any prime p and any $a \in \{1, 2, ..p-1\}$, we have $a^{p-1} \equiv 1 (\bmod p)$. Without using induction, prove that $\forall n \in \mathbb{N}$, $n^7 - n$ is divisible by 42.

Solutions: We begin by breaking down 42 into prime factors: $42 = 7 \times 3 \times 2$. Since 7,3, and 2 are prime, we can apply Fermat's Little Theorem, which says that $a^p \equiv a \pmod{p}$, to get the congruences

$$n^7 \equiv n (\bmod 7), \tag{2}$$

$$n^3 \equiv n (\bmod 3), \text{ and} \tag{3}$$

$$n^2 \equiv n (\bmod 2). \tag{4}$$

Now, let's take (3) and multiply it by $n^3 \cdot n$. This gives us

$$n^7 \equiv n^3 \cdot n^3 \cdot n \equiv n \cdot n \cdot n \equiv n^3 \pmod 3,$$

and since by (3), $n^3 \equiv n \pmod 3$, this gives

$$n^7 \equiv n \pmod 3.$$

Similarly, we take (4) and multiply by $n^2 \cdot n^2 \cdot n$ to get

$$n^7 \equiv n^2 \cdot n^2 \cdot n^2 \cdot n \equiv n^4 \pmod 2.$$

Notice that $n^4 \equiv n^2 \cdot n^2 \equiv n \cdot n \equiv n^2 \pmod 2$, and by (4) $n^2 \equiv n \pmod 2$, so we have

$$n^7 \equiv n \pmod 2.$$

Now, since

$$\begin{aligned} n^7 &\equiv n (\bmod 7), \\ n^7 &\equiv n (\bmod 3), \text{ and} \\ n^7 &\equiv n (\bmod 2), \end{aligned}$$

we have that $n^7 \equiv n \pmod{7 \times 3 \times 2}$, so $n^7 \equiv n \pmod{42}$. Subtracting n from both sides of the congruence gives $n^7 - n \equiv 0 \pmod{42}$, which means $n^7 - n$ is divisible by 42.

5. GCD and Divisions

(a) Let $F(n)$ denote the nth Fibonacci number. Show that $\forall n \in \mathbb{N}$, $\gcd(F(n+1), F(n)) = 1$.

Solutions: We will prove this by induction.

Base case: For $n = 0$, we have $F(1) = 1$ and $F(0) = 1$; obviously $\gcd(1,1) = 1$.

Inductive hypothesis: Assume $\gcd(F(n+1), F(n)) = 1$ for some $n \geq 1$.

Inductive step: For $n+1$, since $F(n+2) = F(n+1) + F(n)$, we have

$$\begin{aligned} \gcd(F(n+2), F(n+1)) &= \gcd(F(n+1) + F(n), F(n+1)) \\ &= \gcd(F(n+1), F(n)) \\ &= 1 \quad \text{(by the inductive hypothesis).} \end{aligned}$$

(b) Prove that for $n \geq 1$, if 935 divides $n^{80} - 1$, then $5, 11$ and 17 do not divide n.

Solutions:

Note that $935 = 5 \times 11 \times 17$. We wish to prove that if $n^{80} \equiv 1 (\bmod 935)$ then $5, 11, 17 \nmid n$.

Since $5, 11, 17$ are pairwise prime, by the Chinese Remainder Theorem we know $n^{80} \equiv 1 (\bmod 5)$, $n^{80} \equiv 1 (\bmod 11)$, and $n^{80} \equiv 1 (\bmod 17)$.

We will now prove the statement by contradiction. Let us now assume the contrary; i.e., that $n^{80} \equiv 1 (\bmod 935)$ and either $5 \mid n$ or $11 \mid n$ or $17 \mid n$. Then we have 3 possible cases:

- If $5 \mid n$ then, $n = 5k$, which implies, $n \equiv 0 (\bmod 5)$ which in turn implies $n^{80} \equiv 0 (\bmod 5)$
- If $11 \mid n$ then, $n = 11k$, which implies, $n \equiv 0 (\bmod 11)$ which in turn implies $n^{80} \equiv 0 (\bmod 11)$
- If $17 \mid n$ then, $n = 17k$, which implies, $n \equiv 0 (\bmod 17)$ which in turn implies $n^{80} \equiv 0 (\bmod 17)$

which are all false as under the assumptions that $n^{80} \equiv 1 \pmod{935}$, since this implies $n^{80} \equiv 1 \pmod{5}$, $n^{80} \equiv 1 \pmod{11}$, and $n^{80} \equiv 1 \pmod{17}$. Thus we have reached a contradiction, and we must have that $5, 11, 17 \nmid n$.

6. Write Your Own Problem

Write your own problem related to this week's material and solve it. You may still work in groups to brainstorm problems, but each student should submit a unique problem. What is the problem? How to formulate it? How to solve it? What is the solution?