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 $0 \leq x < \prod(n_i)$.

(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.

(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 $1 \leq a \leq p-1$, 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.

(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?

(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.*

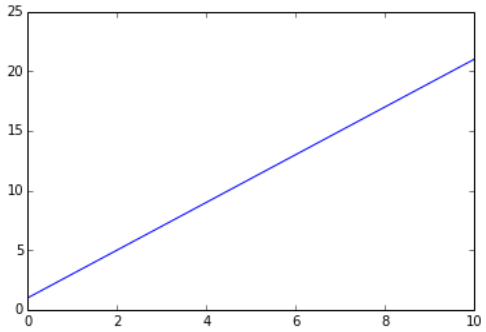
(f) This question aims to introduce you to basic plotting in Python with Matplotlib. First, we will generate 50 evenly spaced points from 0 to 10.

```
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!

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?

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}.

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

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

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.

5. GCD and Divisions

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

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