Refer to the second page of the Course Survey for instructions on submitting written work on Gradescope, and to the instructions on Problem Set 1 for developing and submitting programming problems.

Written problems

1. Textbook exercise 1.10, parts (a) and (b) (use a calculator/computer for the arithmetic, but show the steps). (Extended Euclidean algorithm examples)

2. Textbook exercise 1.15 (basic properties of congruence modulo m).

3. Prove the following basic facts about congruence, asserted in class.

   (a) For any integer $a \in \mathbb{Z}$ and positive integer m, $a \equiv (a\%m) \pmod{m}$.

   (b) With a, m as above, the number $a\%m$ is the *unique* element of $\{0, 1, \cdots, m-1\}$ that is congruent to a modulo m (that is, no other element of this set is congruent to a modulo m).

   (c) For any two integers $a, b \in \mathbb{Z}$ and any positive integer m, $a \equiv b \pmod{m}$ *if and only if* $a\%m = b\%m$.

4. The previous problem shows that congruence modulo m is "compatible with" addition and multiplication, in a suitable sense. In this problem, you'll see that this is **not** true of other arithmetic operations, so you have to be careful.

   (a) (Congruence is not compatible with powers) Find integers a_1, a_2, b_1, b_2 such that $a_1 \equiv a_2 \pmod{3}$ and $b_1 \equiv b_2 \pmod{3}$, but $a_1^{b_1} \not\equiv a_2^{b_2} \pmod{3}$.

   (b) (Congruence is not compatible with division) Find integers a_1, a_2, b_1, b_2 such that $a_1 b_1 \equiv a_2 b_2 \pmod{6}$ and $a_1 \equiv a_2 \not\equiv 0 \pmod{6}$, but $b_1 \not\equiv b_2 \pmod{6}$.

   Note: we'll see later that we can recover a sort of compatibility with both powers and division, but the details are subtle.

5. Textbook exercise 1.16, parts (a), (b), and (c). (Multiplication tables in modular arithmetic)

6. Textbook exercise 1.19. (deducing $g^{\gcd(a,b)} \equiv 1 \pmod{m}$)

Programming problems

1. Write a function `bezout(a,b)` that takes two positive integers a, b and returns three integers g, u, v, where $g = \gcd(a, b)$ and $au + bv = g$. The numbers in the test bank will range up to 256 bits in size, but there will also be smaller case that can be solved by a naive approach. I recommend that you implement the extended Euclidean algorithm (either the way we outlined it in class, or following one of the methods in the text), but other methods may also work.

2. Write a function `disclog(g,h,p)` that solves the discrete logarithm problem in a naive way (quickly enough to work in less than 1 second if p is a 20-bit prime). Multiple answers are possible (we'll discuss this later); an answer n will be marked correct as long as $g^n \equiv h \pmod{p}$. To allow you to see exactly where the naive approach becomes too slow (or, if you're up for it, to allow you to try to implement better methods), the test bank will include cases where p ranges up to 40 bits, but **you will receive full credit as long as your code solves the test cases up to 20-bit primes**. (Later, we'll discuss and implement an algorithm that can solve the entire test bank).