**Written problems**

1. Textbook exercise 1.15 (1.14 in first edition).

   **Solution.**

   Since $a_1 \equiv a_2 \pmod{m}$, there exists an integer $k$ such that $a_1 = a_2 + km$. Similarly, there exists an integer $h$ such that $b_1 = b_2 + hm$.

   It follows that

$$\begin{aligned} a_1 \pm b_1 &= a_2 + km \pm (b_2 + hm) \\ &= a_2 \pm b_2 + (k \pm h)m \end{aligned}$$

   and therefore $a_1 \pm b_1 \equiv a_2 \pm b_2 \pmod{m}$, since the two sides differ by a multiple of $m$.

   Similarly,

$$\begin{aligned} a_1 b_1 &= (a_2 + km)(b_2 + hm) \\ &= a_2 b_2 + a_2 hm + b_2 km + hkm^2 \\ &= a_2 b_2 + (a_2 h + b_2 k + hkm)m \end{aligned}$$

   which shows that $a_1 b_1$ and $a_2 b_2$ differ by a multiple of $m$, i.e. $a_1 b_1 \equiv a_2 b_2 \pmod{m}$.

2. Textbook exercise 1.20 (1.19 in first edition).

   Let $u_1, u_2$ be inverses modulo $m$ of $a_1, a_2$. Then:

$$\begin{aligned} a_1 u_1 &\equiv 1 \pmod{m} \\ a_2 u_2 &\equiv 1 \pmod{m} \\ \Rightarrow (a_1 u_1)(a_2 u_2) &\equiv 1 \cdot 1 \pmod{m} \\ \Rightarrow (a_1 a_2)(u_1 u_2) &\equiv 1 \pmod{m} \end{aligned}$$

   so $u_1 u_2$ is an inverse of $a_1 a_2$ modulo $m$. In particular, $a_1 a_2$ has an inverse modulo $m$, so it is a unit.

3. Textbook exercise 1.22 (1.21 in first edition).

   **Solution.**

   (a) Since $m$ is odd, $m = 2k - 1$ for some integer $k$. Thus $2k = m + 1$, which means that $2k \equiv 1 \pmod{m}$, hence this integer $k$ is the inverse of 2 modulo $m$. Explictly, $k$ equals $\frac{m+1}{2}$, which is therefore the inverse of 2 modulo $m$.

(b) Since $m \equiv 1 \pmod{b}$, $m = 1 + kb$ for some integer $k$. Explicitly, this integer $k$ is $\frac{m-1}{b}$. Then $kb = m - 1$, so $kb \equiv -1 \pmod{m}$; multiplying by $(-1)$ gives $(-k)b \equiv 1 \pmod{m}$. Therefore $-k$ is an inverse of $b$ modulo $m$. To obtain a "reduced" inverse, we must simply reduce $-k$ modulo $m$. Since $0 < k < m$, it follows that $0 < m - k < m$, i.e. $1 \leq m - k \leq m - 1$. So $m - k$ is the value we seek. Expressing this in terms of $m$ and $b$, we obtain:

$$\begin{aligned} m - k &= m - \frac{m-1}{b} \\ &= \frac{(b-1)m + 1}{b}. \end{aligned}$$

Notice that when $b = 2$, this formula specializes to $\frac{m+1}{2}$, as found in part (a).

4. Consider the linear congruence $ax \equiv b \pmod{M}$. Prove that this congruence has a solution if and only if $\gcd(a, M)$ divides $b$.

**Solution.**

First, suppose that the congruence has a solution. Then there is an integer $k$ such that $ax = b + kM$. Let $g = \gcd(a, M)$. Then $b = ax - kM = g \cdot (\frac{a}{g}x - k\frac{M}{g})$, so $b$ is equal to $g$ times an integer, i.e. $g$ divides $b$.

Conversely, suppose that $g = \gcd(a, M)$ divides $b$. From the extended Euclidean algorithm, there exist integers $u, v$ such that $au + Mv = g$. Multiplying both sides of this equation by the integer $\frac{b}{g}$ gives $a(u\frac{b}{g}) + (v\frac{b}{g})M = b$. Since $v\frac{b}{g}$ is an integer, it follows that $au\frac{b}{g} \equiv b \pmod{M}$. Therefore $x = u\frac{b}{g}$ is a solution to the congruence $ax \equiv b \pmod{M}$; in particular, the congruence has a solution.

5. Textbook exercise 1.34 (1.32 in first edition).

**Solution.**

(a) The following table shows the sequence of powers of 2 modulo the four primes listed (shown until it repeats).

$$\begin{aligned} p = 7 &\quad 2, 4, 1, \cdots \text{(period 3)} \\ p = 13 &\quad 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1, \cdots \text{(period 12)} \\ p = 19 &\quad 2, 4, 8, 16, 13, 7, 14, 9, 18, 17, 15, 11, 3, 6, 12, 5, 10, 1, \cdots \text{(period 18)} \\ p = 23 &\quad 2, 4, 8, 16, 9, 18, 13, 3, 6, 12, 1, \cdots \text{(period 11)} \end{aligned}$$

For $p = 7$ and $p = 23$, these lists are incomplete (both include exactly half of all units modulo $p$). For $p = 13$ and $p = 19$, these lists include all units, so 2 is a primitive root.

(b) The following table shows the sequence of powers of 3 modulo the four primes listed.

$$\begin{aligned} p = 5 &\quad 3, 4, 2, 1, \cdots \text{(period 4)} \\ p = 7 &\quad 3, 2, 6, 4, 5, 1, \cdots \text{(period 6)} \\ p = 11 &\quad 3, 9, 5, 4, 1 \cdots \text{(period 5)} \\ p = 17 &\quad 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1, \cdots \text{(period 16)} \end{aligned}$$

Therefore 3 is a primitive root modulo $p = 5, 7$, and 17, but not 11.

(c) We can check whether any given number is a primitive root by listing its powers (mod $p$) and seeing if there are $p - 1$ of them. Here is a quick function definition that does this.

```
>>> def is_pr(g,p):
...     if g%p == 0: return False
...     pows = set()
...     for e in range(p-1):
...             pows.add( (g**e)%p )
...     return len(pows) == p-1
...
```

Here, `g**e` is Python's syntax for $g^e$ (if you type `g^e`, you will end up with the "bitwise xor" of $g$ and $e$, which is a very different thing). Of course, we ought to replace this with a "fast-power" algorithm that takes advantage of the fact that we're planning to reduce the result modulo $p$; for numbers of the size we're working with, though, performance is not an issue.

Using this function we can start trying potential values of $g$ until we find a primitive root.

```
>>> is_pr(2,23)
False
>>> is_pr(3,23)
False
>>> is_pr(4,23)
False
>>> is_pr(5,23)
True
```

So 5 is a primitive root modulo 23. Similarly, we can find a primitive root for each of the other primes mentioned this way.

```
>>> is_pr(2,29)
True
>>> is_pr(2,41)
False
>>> is_pr(3,41)
False
>>> is_pr(4,41)
False
>>> is_pr(5,41)
False
>>> is_pr(6,41)
True
>>> is_pr(2,43)
False
>>> is_pr(3,43)
True
```

so 2 is a primitive root modulo 29, 6 is a primitive root modulo 41, and 3 is a primitive root modulo 43.

For reference, here are lists of all primate roots for each of these primes.

$$
\begin{aligned}
p = 23 \quad & 5, 7, 10, 11, 14, 15, 17, 19, 20, 21 \\
p = 29 \quad & 2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27 \\
p = 41 \quad & 6, 7, 11, 12, 13, 15, 17, 19, 22, 24, 26, 28, 29, 30, 34, 35 \\
p = 43 \quad & 3, 5, 12, 18, 19, 20, 26, 28, 29, 30, 33, 34
\end{aligned}
$$

(d) Using the `is_pr` function written above, we can quickly produce this list as follows.

```
>>> for g in range(1,11):
...     if is_pr(g,11): print g,
...
2 6 7 8
```

Indeed, there are 4 of them, and $\phi(10) = 4$ (the numbers relatively prime to 10 are $1, 3, 7, 9$).

(e) We can define the following function to return all the primitive roots of a given prime as a list.

```
>>> def all_prs(p):
...     res = []
...     for g in range(1,p):
...             if is_pr(g,p): res += [g]
...     return res
...
```

Calling this function on $p = 229$ gives the full list.

```
>>> all_prs(229)
[6, 7, 10, 23, 24, 28, 29, 31, 35, 38, 39, 40, 41, 47, 50, 59, 63, 65, 66,
67, 69, 72, 73, 74, 77, 79, 87, 90, 92, 96, 98, 102, 105, 110, 112, 113,
116, 117, 119, 124, 127, 131, 133, 137, 139, 142, 150, 152, 155, 156,
157, 160, 162, 163, 164, 166, 170, 179, 182, 188, 189, 190, 191, 194,
198, 200, 201, 205, 206, 219, 222, 223]
```

We can quickly determine the length of this list as follows.

```
>>> len(all_prs(229))
72
```

So there are 72 primitive roots modulo 229. To check that this is equal to $\phi(228)$, we can quickly put together a naive (but efficient enough for primes of this size) algorithm to commute $\phi$ directly from the definition as follows.

```
>>> def gcd(a,b):
...     while b != 0: a,b = b,a%b
...     return a
...
```

```
>>> def phi(m):
...     res = 0
...     for a in range(m):
...             if gcd(a,m) == 1: res += 1
...     return res
...
>>> phi(228)
72
```

So indeed the number of primitive roots modulo 229 is equal to $\phi(228)$.

(f) This can be accomplished in a few lines in the Python terminal as follows. Note that it is easy enough in this case to just type in the primes less than 100, but various algorithms are available to do this automatically for a bound of your choice.

```
>>> primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,
 59, 61, 67, 71, 73, 79, 83, 89, 97]
>>> for p in primes:
...     if is_pr(2,p): print p,
...
3 5 11 13 19 29 37 53 59 61 67 83
```

So 2 is a primitive root for 12 of the first 25 primes, or about half of them. Despite this seemingly strong evidence, it is actually unknown whether 2 is even a primitive root for infinitely many primes or not (it is believed that it is; this claim is called Artin's conjecture).

(g) We can use the same code as before.

```
>>> for p in primes:
...     if is_pr(3,p): print p,
...
2 5 7 17 19 29 31 43 53 79 89
>>> for p in primes:
...     if is_pr(4,p): print p,
...
>>>
```

Note that 4 is not a primitive root for *any* primes less than 100. In fact, 4 is not a primitive root for any primes at all. This is because no perfect square can be a primitive root for any primes (with the exception of $p = 2$, for which any odd number, hence any odd square, is a primitive root).

6. (a) Suppose that $a$ and $b$ are two integers such that $g^a \equiv 1 \pmod{m}$ and $g^b \equiv 1 \pmod{m}$. Prove that $g^{\gcd(a,b)} \equiv 1 \pmod{m}$.

   (b) Suppose $g \in (\mathbf{Z}/p\mathbf{Z})^\times$, where $p$ is a prime. Let $d$ be the smallest positive integer such that $g^d \equiv 1 \pmod{p}$ (called the *multiplicative order* of $g \pmod{p}$). Prove that $d$ divides $p - 1$.

**Solution.**

(a) Since $g^a \equiv 1 \pmod{m}$, it follows that $g^{au} \equiv (g^a)^u \equiv 1 \pmod{m}$ for any nonnegative integer $m$. Furthermore, defining $g^{-n} \pmod{m}$ to be the inverse of $g^n$ modulo $m$, it also follows that $g^{au} \equiv 1 \pmod{m}$ for all *negative* integers $u$ as well, since the inverse of 1 is 1. Similarly, $g^{vb} \equiv 1 \pmod{m}$ for all integers $v$.

From the extended Euclidean algorithm, we know that there exist integers $u, v$ such that $\gcd a, b = au + bv$. It follows that

$$g^{\gcd(a,b)} = (g^a)^u (g^b)^v \equiv 1^u 1^v \equiv 1 \pmod{m}.$$

(b) From Fermat's little theorem, $g^{p-1} \equiv 1 \pmod{p}$. Since $g^d \equiv 1 \pmod{p}$, part (a) implies that $g^{\gcd(d,p-1)} \equiv 1 \pmod{p}$. Since $d$ is the *smallest* such exponent, and since $\gcd(d, p-1)|d$, it follows that $\gcd(d, p-1)$ must equal $d$ exactly (if it were a proper factor, it would be smaller than $d$). So $d = \gcd(d, p-1)$, which implies in particular that $d$ divides $p-1$.

7. Textbook exercise 2.3 (same in first edition).

**Solution**

(a) If $a$ and $b$ are two solutions to $g^x \equiv h \pmod{p}$, then $g^a \equiv g^b \pmod{p}$, hence $g^{a-b} \equiv 1 \pmod{p}$ (by multiplying by $g^{-1} \pmod{p}$ $b$ times on both sides). The powers of $g$ modulo $p$ form a periodic sequence of period $p-1$, so in particular the number 1 occurs (modulo $p$) as a power of $g$ precisely when the exponent is a multiple of $(p-1)$. Therefore $a - b$ is a multiple of $p-1$, i.e. $a \equiv b \pmod{p-1}$.

This means that the various possible integer values of the discrete logarithm $\log_g(h)$ are all congruent modulo $p-1$. In particular, there is only one possible element of $\mathbf{Z}/(p-1)$ that arises by reducing one of these exponents modulo $p$. Hence, $\log_g$ gives an unambiguously defined function from $(\mathbf{Z}/p)^\times$ to $\mathbf{Z}/(p-1)$.

(b)

$$\begin{aligned} g^{\log_g h_1 + \log_g h_2} &= g^{\log_g h_1} g^{\log_g h_2} \\ &\equiv h_1 h_2 \pmod{p} \end{aligned}$$

Thus the number $\log_g h_1 + \log_g h_2$ (or rather, the sum of any possible integer values for each of these two discrete logarithms) satisfies the defining property of $\log_g(h_1 h_2)$, so it is a discrete logarithm of $h_1 h_2$, as desired.

(c)

$$\begin{aligned} g^{n \log_g h} &\equiv \left(g^{\log_g h}\right)^n \pmod{p} \\ &\equiv h^n \pmod{p} \end{aligned}$$

Thus $n \log_g h$ satisfies the defining property of $\log_g(h^n)$.

8. Textbook exercise 2.4 (same in first edition).

**Solution.**

(a) The powers of 2 modulo 23 are $2, 4, 8, 16, 9, 18, \mathbf{13}, 3, 6, 12, 1, \cdots$ (period 11). Therefore $\log_2 13 = 7$ is one solution. More generally, since the order of 2 modulo 23 is 11 (as seen above), this discrete logarithm can be described by the class 7 (mod 11).

(b) The powers of 10 modulo 47 are $10, 6, 13, 36, 31, 28, 45, 27, 35, 21, \mathbf{22}, \cdots$ (this sequence turns out to have period 46, though I won't copy it all out), so $\log_{10}(22) = 11$ (or more generally, 11 (mod 46)).

(c) $\log_{627}(608) = 18$, since table 2.1 shows that $627^{18} \equiv 608 \pmod{941}$. The text mentions that 627 is a primitive root modulo 941, so the most complete answer would be 18 (mod 940).

**Programming problems**

9. You play Bob in this problem. Write a program to perform Diffie-Hellman key exchange. Specifically, you will receive three integers from Alice: $p$, $g$, and $A$ (as described in Table 2.2 of the textbook), where $p$ is a 1024-bit prime. You must generate a number $B$ to send to Alice, and also compute the shared secret $S$. You should look up how to generate random numbers. The autograder will check to make sure that your program is not deterministic.

**Solution.**

As described in the book, the values $B$ and $S$ are computed by choosing an ephemeral key $b \in \mathbf{Z}/(p-1)$ and then computing $B \equiv g^b \pmod{p}$ and $S \equiv A^b \pmod{p}$. Once we have a fast-powering algorithm, this can be done in a few lines as follows.

```
import random
random.seed()

# Fast powering algorithm
# Returns (a^e)%m
def pow_mod(a,e,m):
        res = 1
        while e>0:
                if e%2: res = (res*a)%m
                a = (a*a)%m
                e /= 2
        return res

# Read the input
[p,g,A] = map(int,raw_input().split())

# Generate the ephemeral key
b = random.randrange(1,p-1)

# Compute the public number B and shared secret S
print pow_mod(g,b,p), pow_mod(A,b,p)
```

*Note.* I've implemented a fast modular powers function here, in order to show how it works. However, this is actually built into Python as simply `pow(a,e,m)`, so you can just use that rather than implementing it yourself.

10. Write a problem to solve linear congruences of the form $ax \equiv b \pmod{M}$. If the congruence has solutions, your program should give a single congruence $x \equiv c \pmod{N}$ that describes them all. If the congruence has no solutions, your program must detect this.

**Solution**

We know from problem 4 that a solution exists if any only if $\gcd(a, M)|b$. So we can compute this gcd (using an efficient method, like the Euclidean algorithm) and return "None" if it does not divide $b$.

In the special case $\gcd(a, M) = 1$ (as in the first ten test cases), a solution is guaranteed to exist, and it can be found in the same way as in usual algebra: multiplying be an inverse of $a$. The congruence $ax \equiv b \pmod{M}$ is true if and only if $x \equiv ba^{-1} \pmod{M}$. The value $a^{-1} \pmod{M}$ can be found quickly as the integer $u$ in a solution to $au + Mv = 1$.

In the general case, where $\gcd(a, M) \neq 1$, we can reduce to this special case by observing that $g$ divides all three of $a, b, M$. Therefore

$$\begin{aligned} ax &\equiv b \pmod{M} \\ \Leftrightarrow \exists k \in \mathbf{Z}:\ ax - b &= kM \\ \Leftrightarrow \exists k \in \mathbf{Z}:\ \frac{a}{g}x - \frac{b}{g} &= k\frac{M}{g} \\ \Leftrightarrow \frac{a}{g}x &\equiv \frac{b}{g} \pmod{M/g}. \end{aligned}$$

Since $\frac{a}{g}u + \frac{M}{g}v = 1$ (for the same $u$ and $v$ that give $au + Mv = g$), it is now the case that $\frac{a}{g}$ is a unit modulo $\frac{M}{g}$; its inverse is $u$. Therefore the original congruence is equivalent to

$$x \equiv u\frac{b}{g} \pmod{M/g}.$$

Hence we need to solve the equation $au + bM = g$, which can be done at the same time that $g$ is computed (using the extended Euclidean algorithm); then the desired answer will be $N = \frac{M}{g}$, $c = \left(u\frac{b}{g}\right) \% N$. Here is an implementation. Note that the function implementing the extended Euclidean algorithm is streamlined somewhat, so that is computes $g$ and $u$ but does not bother to compute $v$ (which is not needed for any later computation).

```
# A version of the extended Euclidean Algorithm.
# Returns a pair (g,u), where g = gcd(a,m) and au = g mod m.
def ext_euclid(a,m):
        pre = (m,0)
        cur = (a,1)
        while cur[0] > 0:
```

```
                k = pre[0]/cur[0]
                nex = (pre[0]-k*cur[0],pre[1]-k*cur[1])
                pre = cur
                cur = nex
        return pre

# Returns None if ax = b mod M has no solutions.
# Otherwise returns a pair (c,N) where the solution is x = c mod N.
def solve(a,b,M):
        g,u = ext_euclid(a,M)
        if b%g != 0: return None
        return ( ((b/g)*u)%(M/g), M/g )

# Read the input
[a,b,M] = map(int,raw_input().split())

# Compute and print the output
soln = solve(a,b,M)
if not(soln): print 'None'
else: print soln[0],soln[1]
```

11. You play Eve in this problem. You have intercepted six encrypted messages sent from Alice to Bob. The cryptosystem they are using converts a string (the plaintext) into an integer (the ciphertext); so the data you have intercepted consists of six integers. You have also obtained the source code Alice and Bob are using for their encryption; it is reproduced below on the last page (you can also find it in the Python starter code on hackerrank). Alice and Bob have a secret key $k$, which is a 1024-bit integer.

    Write a program to break Alice and Bob's encryption, and print the original six plaintext messages.

    **Solution.**

    The encryption can be broken by extracting the secret key from the ciphertexts. The key observation is that each cipher text is of the form `key` $\cdot$ `encode(text)`, hence they all share a common factor of `key`. It is possible that they share an even larger common factor: specifically, the gcd of all six ciphertexts will be equal to `key` times the gcd of the six encoded plaintexts. However, it turns out that it is extremely unlikely that six reasonably well-distributed numbers have a common factor (if the integers are large enough, it is a neat exercise in probability to show that the probability that they are relatively prime very close to is $\prod_{p \text{ prime}}(1-\frac{1}{p^6})$, which can be reexpressed with a bit of algebraic effort as $\left(\sum_{n=1}^{\infty}\frac{1}{n^6}\right)^{-1}$, which is about 0.98. You can find an exact value with a bit more work, but it's not really needed for our current purpose).

    Therefore, with a small chance of error, we will extract the key and hence break the encryption by simply taking the gcd of the six ciphertexts. You can use your code from a problem on the previous problem set to do this. Here is an example implementation (you should also include Alice and Bob's original source; I have omitted here for clarity).

```
def gcd(a,b):
        while b != 0:
                a,b = b,a%b
        return a

# Extract the key and decipher a list of ciphertexts
def analyze(ciph):
        key = reduce(gcd,ciph,0) # Concise syntax to take g=0 and repeatedly
                                 # replace g with gcd(g,ciph[i]).
        return [decipher(n,key) for n in ciph]

# Read the input
ciph = map(int,raw_input().split())

# Print the output
for plain in analyze(ciph): print plain
```

*Note.* If you want a stronger algorithm that could work with fewer than 6 plaintexts, and have a reduced probability of error in any case, you could use compute the gcd and then successively try integer multiples of it. This would require some algorithm to tell whether the resulting deciphered texts "look like" valid plaintext or not, which would depend somewhat on what sort of data they are (e.g. if they are English text, frequency analysis would likely be effective).