ASSIGNMENT 3: COMPOSITENESS TESTS AND CRYPTOGRAPHY
DUE FRIDAY, NOVEMBER 11, 2011 AT 11:59PM

Contents

1. Compositeness tests

Recall the fast-exponentiation algorithm for computing $a^k \bmod n$ (ie, the remainder of a^k after dividing by n): find the binary expansion of k, and then compute the successive squares of a until you reach the highest power of 2 in the binary expansion of k. Then multiply together appropriate successive squares to get $a^k \bmod n$.
Example. Compute the remainder of 3^{196} when divided by 263. We first write the exponent in binary: $196 = 2^7 + 2^6 + 2^2$. We now compute the 2^ith powers of 3 mod 263, either by hand or with the assistance of a calculator:

$$3^2 \equiv 9 \mod 263, 3^4 \equiv 81 \mod 263, 3^8 \equiv 81^2 \equiv 249 \mod 263, 3^{16} \equiv 249^2 \equiv 196 \mod 263,$$
$$3^{32} \equiv 196^2 \equiv 18 \mod 263, 3^{64} \equiv 18^2 \equiv 61 \mod 263, 3^{128} \equiv 61^2 \equiv 39 \mod 263.$$

We have $3^{196} = 3^{128} \cdot 3^{64} \cdot 3^4$, so $3^{196} \equiv 39 \cdot 61 \cdot 81 \equiv 183 \mod 263$.

A Python tool which might be helpful (although is not necessary) in the following problem is the bin function: bin(n) will return a string representation of the binary expansion of n, prefixed by '0b'. Try it out!

Problem 1 (10 points)**.** Write a function, fastexp(a, k, n), which returns the remainder upon division of a^k by n. The function should implement the fast exponentiation algorithm described above, and in particular should run more or less instantaneously even if k, n have dozens of digits in them.

You should be sure that this function works correctly and quickly, because you are going to use it in many of the other problems in this assignment.

Problem 2 (5 points)**.** Write a function, FCT(n, a), which tests whether n is a composite number or not using the Fermat compositeness test to the base a. Your function should either return the integer 1, if the test shows that n is composite, or the integer 0, if the test is inconclusive. The function should work more or less instantaneously even if n, a have dozens of digits in them.

Problem 3 (5 points)**.** Write a function, MR(n, a), which tests whether n is a composite number or not using the Miller-Rabin test to the base a. Your function should either return the integer 1, if the test shows that n is composite, or the integer 0, if the test is inconclusive. The function should work more or less instantaneously even if n, a have dozens of digits in them.

As a sample testcase of the above two functions, show that the number

$$7403756347956171282804679609742957314259318888923128908493623263897276503402826627689199641962511784399589433050212758537011896809828673317327310893090055250511687706329907239638078671008609696253793465056379635 9$$

is composite. On the other hand, you almost certainly will be unable to find a factor for this number (trial division definitely isn't going to work), and as a matter of fact at one point there was a $30,000 prize offered by RSA Laboratories available for the first person or team to factor this number. As of today no factorization is publicly known, although presumably the creators of the contest know the factorization.

2. Some cryptography

In this section we will write some functions which perform some of the basic cryptographic operations we discussed in class. Before describing the exact problems, we describe some basic string manipulation features of Python which will be helpful.

We will want to use the Caesar cipher, which we think of as acting on elements of the alphabet 'a', 'b', ..., 'z'. The standard way to encode characters in a computer as numerical data is the ASCII character encoding scheme. To obtain the ascii code of a character in Python, use the ord function on a string consisting of a single character. For example,

```
>>> ord('a')
97
>>> ord('b')
98
>>> ord('A')
65
```

To convert from an integer to its corresponding ASCII character, use the chr function on an integer:

```
>>> chr(97)
'a'
>>> chr(97 + 5)
'f'
```

You will also probably want to know how to concatenate a list of strings together for the next problem. Suppose you have a list l consisting of strings, and you want to concatenate them, in order, from left to right. There are two ways to do this. Perhaps the conceptually simplest is to use the + operation on strings; + on strings simply concatenates them. For example,

```
>>> s1 = 'hello'
>>> s2 = 'world'
>>> s1 + s2
'helloworld'
```

Another way, if you are given a list of strings, is to use the .join method on a string. If s is a string, and l a list of strings, then s.join(l) will concatenate all the strings in l together, with the string s separating each pair of consecutive elements of l. For example,

```
>>> l = ['prime', 'numbers']
>>> ''.join(l)
'primenumbers'
>>>' '.join(l)
'prime numbers'
>>>'?'.join(l)
prime?numbers
```

In principle, the use of the .join method should be more efficient than the + operation, at least when applied to many long strings, but certain implementations of Python will optimize the use of + in a loop. For the inputs we will test your functions on, either method will not present any efficiency issues.

Finally, you might want to convert an upper case letter to a lower case letter or vice versa, or test whether a character is a lower case letter, upper case letter, or not a letter at all. For case conversions, you can use the .lower and .upper methods on a string, and to test for whether a character is a lower case letter, you can either use the ¡= applied to characters (which compares their corresponding ASCII values), or import the string module and use the string string.ascii_lowercase:

```
>>> s = 'Hello World!'
>>> s.lower()
'hello world!'
>>> s.upper()
'HELLO WORLD!'
>>> import string
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> 'b' in string.ascii_lowercase
True
>>> 'X' in string.ascii_lowercase
False
>>> '!' in string.ascii_lowercase
False
```

Problem 4 (10 points)**.** Implement a function, caesar(msg, a, b), which takes a string msg, two integers a, b, and applies the Caesar cipher given by $x \mapsto ax + b$ to each character in msg. Caesar should leave non-alphabetic characters unchanged, and should preserve capitalization. The function should return the encrypted message. See the test cases for examples of how this function should operate. This function should run more or less instantaneously on inputs that are hundreds of pages long, although we will only test this function on inputs several sentences long.

Problem 5 (10 points)**.** Implement a function, frequency(msg), which accepts a string msg, and returns a dictionary counting the frequency of each letter of the alphabet in msg. You should ignore non-alphabetic characters, and count lower case and upper case characters as the same. The keys of the dictionary should be lower case letters 'a', 'b', ..., 'z'.

For the following problem, you will probably need a functioning version of the extended gcd algorithm, from Programming Assignment 1.

Problem 6. Implement a function, RSA_decode(cipher, p, q, e), which given an integer cipher representing an encrypted RSA message, the two private primes p, q, and the encryption exponent e, returns the decoded message that the sender wants to transmit. The function should work quickly even if cipher, p, q, e have about 50-100 digits in them.

3. Useful information

3.1. **Problems to hand in.** In your file which contains the code you submit, make sure that you can run the file in IDLE without error and that all functions are loaded appropriately. In other words, if you open the .py file you submit and run it in IDLE (using the Run Module command under the Run submenu), there should be no error messages in the interactive interpreter, and you should be able to use the functions you wrote. Also make sure that you spell the names of the required functions correctly! (On the other hand, the names of the arguments passed into the function can be arbitrary, as long as the function accepts the correct number of arguments.) Finally, please put your name near the top of the file in a comment (use the # symbol to write comments; Python will ignore everything on that line which comes after the #).

- Turn in problems 1, 2, 3, 4, 5, and 6 in a single file named [lastname]3.py (without brackets around your last name.)

3.2. **Test cases.** (This is simply a repeat of the information provided in the previous assignment.) The Python standard library contains a handy module called doctest, which provides a lightweight method for testing test cases. In the assignment page, the template .py file will contain a lengthy *docstring* at the beginning of each function, which contains the test cases we provide for each function (if there are any).

Also, at the end of the file, there will be a short snippet of code, which will only run if you ask IDLE to execute the entire file with the 'Run Module' command. The template we provide will include a 'verbose=True' switch, which will cause your program to output extensive information on the results of each test case. If you would like to suppress the output, and only be informed if cases fail, remove 'verbose=True' from within the parentheses of the doctest.testmod function.

For questions that use floating point numbers, we only test whether your answer is within some degree of accuracy. It is conceivable for your function to fail those testcases even if it is correct because of the way floating point arithmetic is handled on the machine being used.

Usage of the test cases is optional. If you want, you can delete the docstrings, delete the code at the end which calls doctest.testmod, or just not use the template

file provided. However, we strongly recommend you test your functions against some test cases to ensure that your code is at least somewhat functional.