CS 70 Spring 2005 — Discrete Mathematics for CS — Clancy/Wagner — HW 7

Due Saturday, March 19

Coverage: This assignment involves topics from the lectures of March 1, 8, and 10, and from Rosen section 2.6.

Administrative reminders: We will accept only unformatted text files or PDF files for homework submission. Include your name, login name, section number, and partner list in your submission. Give the command `submit hw7` to submit your solution to this assignment.

We advise you to work with a partner. If you haven't yet switched partners, you should do so for this assignment.

Question 1 is a programming assignment. You should work on your own for the programming assignment (but you may work with a partner for the rest of the questions, as usual).

Homework exercises:

1. **(24 pts.) Implementing RSA**
   You are to complete a program (either in Scheme or in Java) to generate a public/private key pair and encrypt and decrypt messages. In Scheme, this involves completing the functions `key-component-list`, `inverse`, and `mod-power` in the framework file `rsa.scm`. The analogous task in Java is to complete the methods `createKeys`, `inverse`, and `modPower` in the framework file `RSA.java`. Don't change any of the code provided in the file other than what's specified.

   The Scheme functions will be tested in `stk`. The Java code will be tested both with a `RSAtester.java` class that has access to the methods you are to provide, and with a shell script that calls the `main` method. Command-line arguments to the `RSA` program are as follows.

   - The first argument is either `-e`, specifying that an ASCII string is to be encrypted, or `-d`, specifying that a big integer is to be decrypted.
   - The second argument is the string (enclosed in double quotes) or the big integer. If the second argument is `-`, i.e. a single hyphen, the string or big integer is read from standard input; this allows easy chaining of an encryption with a decryption or vice versa.
   - The third argument is only relevant for an encryption. If provided, it specifies the name of a file that contains a public key. If the third argument is not provided, the file `public-key` in your working directory is used.

   Initialization code provided in both programs reads or create files in your working directory named `private-key` and `public-key` if they don't yet exist; each should contain a single line containing two numbers.

Scheme and Java both have built-in support for large integers. You are not allowed to use Scheme's `expt-mod` function. Java provides in the class `java.math.BigInteger` enough support to complete this assignment with hardly any code. Thus, you may not use the following methods in `java.math.BigInteger`:

- `gcd (BigInteger)`
- `isProbablePrime (int)`
- `modInverse (BigInteger)`
- `modPow (BigInteger, BigInteger)`
- `pow (int)`
- `probablePrime (int, Random)`

You are also forbidden to use any other RSA or bignum library, whether part of Java or external.

The class `java.util.Random` provides a random number generator that you may find useful when generating keys. The Scheme counterpart is the `random` function; given an integer argument n, it returns a random integer between 0 and $n-1$, inclusive.

Framework files are in the directory `~cs70/code`.

The directory `~cs70/public-keys` will contain files named `aa`, `ab`, etc.; file *xy* will contain the public key for user `cs70-xy` in the format described above. Each file initially will contain the pair

```
3 6682189
```

(the corresponding private key is `4451347 6682189`). These files are writable; we encourage you to update your file with more reasonable values for the public key, to allow other students to test their code by sending you messages.

2. (6 pts.) Thinking about n

Suppose someone chose a value for n that was not a product of two primes, i.e., $n = pq$ with $p > 1$, $q > 1$, and q is composite. It would obviously be easier to factor, thus posing a security risk. But would the encrypting and decrypting operations still work with this n? Defend your answer.

3. (6 pts.) Euclid's argument

Consider the following result, first proved many centuries ago.

Theorem 1 (Euclid) *There exist infinitely many primes.*

Proof: Assume to the contrary that there exist finitely many primes. Let these primes (in increasing order) be $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $\ldots$, p_k. Let $q_k = p_1 p_2 p_3 \cdots p_k + 1$. Note that q_k is a new number not in the list of primes $p_1, \ldots, p_k$. At the same time, it is not divisible by p_i for any i, since $q_k \equiv p_1 p_2 p_3 \cdots p_k + 1 \equiv 1 \pmod{p_i}$, which would mean that q_k is a new prime different from $p_1, \ldots, p_k$, which is a contradiction. This completes the proof. □

Let $p_1, \ldots, p_k$ represent the first k primes. Are we guaranteed that $p_1 p_2 p_3 \cdots p_k + 1$ is always prime for all $k \geq 1$? Is the above proof valid? Explain.

4. (24 pts.) String matching

(a) Consider again the fingerprint $F_p(w) = w \bmod p$, where we identify the integer $w = 2^{n-1}w_{n-1} + \cdots + 2w_1 + w_0 \in \mathbb{N}$ with the n-bit bitstring $w = \langle w_0, w_1, \ldots, w_{n-1} \rangle$.

Fix $n, m \in \mathbb{N}$ with $n < m$. Let X be a m-bit string $\langle x_0, x_1, \ldots, x_{m-1} \rangle$. Let $X_{(i)}$ denote the n-bit substring $\langle x_i, x_{i+1}, \ldots, x_{i+n-1} \rangle$ of X that starts at position i, or equivalently, the integer $X_{(i)} = 2^{n-1}x_{i+n-1} + \cdots + 2x_{i+1} + x_i \in \mathbb{N}$. For example, if $m = 5$, $n = 4$, and $X = \langle 0, 1, 1, 0, 1 \rangle$, then $X_{(0)} = \langle 0, 1, 1, 0 \rangle = 6$ and $X_{(1)} = \langle 1, 1, 0, 1 \rangle = 11$.

Show how to efficiently compute $F_p(X_{(i)})$ from $F_p(X_{(i+1)})$.

(b) Suppose we are given a m-bit string X and a n-bit string Y, and we want to test whether Y appears as a substring of X. Consider the following naive algorithm:

NAIVESTRINGMATCH(X, Y):
1. For $i = m - n$ down to 0, do:
2. If $X_{(i)} = Y$, return i.
3. Return "No match."

Argue that this naive algorithm has a $O(mn)$ worst-case running time, if we count each bit operation as one unit of time.

(c) Give a faster algorithm. (Hint: make step #2 more efficient.)

(d) Suppose your algorithm is allowed to have a small chance (say, $< 1/m$) of returning an erroneous answer. Describe an algorithm with a $O(m \lg m)$ worst-case running time. Your analysis of the running time can be somewhat informal, but be sure to show all parameter choices.

(If you cannot find an algorithm with such a provable bound on the running time, reduce the asymptotic running time as much as you are able. Ignore constant factors.)