Data Structures and Algorithms

Spring 2014

Cesar Agustin Garcia Vazquez

March 22, 2014

1 Problem Set 06

1. Suppose that Harvard ID numbers were issued randomly, with replacement. That is, your Harvard ID would consist of just 8 randomly generated digits, and no check was made to ensure that the same number was not issued twice. You might use the last four digits of your ID number as a password. Try go give exact numerical answers for the following questions:
   a) How many people would you need to have in a room before it was more likely than not that two had the same last four digits?
   b) How many numbers could be issued before it would be more likely than not that there is a duplicate Harvard ID number?
   c) What would the answers for the above questions be if there were 12 digit ID numbers?

2. Suppose each person gets a random hash value from the range $[1, \ldots, n]$. (For the case of birthdays, n would be 365.) Show that for some constant c_1, where there are at least $c_1\sqrt{n}$ people in a room, the probability that no two have the same hash value is at most $1/e$. Similarly, show that for some constant c_2 (and sufficiently large n), when there are at most $c_2\sqrt{n}$ people in the room, the probability that no two have the same hash value is at least $1/2$. Make these constants as close to optimal as possible.
   Hint: you may use the fact in equation (1)

$$e^{-x} \geq 1 - x \tag{1}$$

   and in equation (2).

$$e^{-x-x^2} \leq 1 - x \text{ for } x \leq \frac{1}{2} \tag{2}$$

   You may feel free to find and use better bounds.

3. For the document similarity scheme described in class, it would be better to store fewer bytes per document. Here is one way to do this, that uses just 48 bytes per document: take an original sketch of a document, using 84 different permutations. Divide the 84 permutations into 6 groups of 14. Re-hash each group of 14 values to get 6 new 64 bit values. Call this the *super-sketch*. Note that for each of the 6 values in the super-sketch, two documents will agree on a value when they agree on all 14 of the corresponding values in the sketch. Why does it make sense to simply assume that this is the only time a match will occur?
   Consider the probability that two documents with resemblance r agree on two or more of the six sketches. Write equations that give this probability and graph the probability of a function of r. Explain and discuss your results.
   What happens if instead of using a 64 bit hash value for each group in the super-sketch, we only use a 16 bit hash? An 8 bit hash?

4. Prove that 636,127 is composite by finding an appropriate witness. Be sure to give ample evidence showing that your witness is in fact witnesses. (Note: do not use a factor as a witness!) Sure, these numbers are small enough that you can exhaustively find a factor; that is not the point. A factor is not a witness, according to our definition.) Hint: you will want to write some code. You will preferably use a package that deals with big integers appropriately, as you may want to use some of this code for the next problem (RSA). We don't need a code listing for this problem, a short summary of the output should suffice.

   The number 294, 409 is a Carmichael number. Prove that it is composite by finding a witness. Briefly explain why Fermat's little theorem won't help.

   Answer: For the number 636,127, we first generate a random number w, between $(1, 636, 127)$ and then we just applied Fermat's little Theorem. The trick here is to correctly compute $w^{636,126}$ mod $636, 127$, which can be easily done using the java class *java.math.BigInteger*. If the reminder is different than 1, that means that the number is composite and then we stop and log the witness and the reminder obtained.
   The number 360,850 acts as a witness that 646,127 composite, since it is $360, 850 \equiv 36, 085 \bmod 636, 127$.

   To find a witness for the Carmichael number, Fermat's Little Theorem does not work directly with any number, since it will return the reminder as 1 if the possible witness share a factor with $294, 409$. This might lead to computing a lot of unnecessary computations if we don't consider this condition.
   So the tweak is that after getting randomly the possible witness, we compute the greater common divisor, and if it is 1, then we drop that possible witness and we retrieve another one.
   After adding the dropping condition, the program finds the witness $88, 987$ which returned the reminder $16, 133$, so we have that $88, 987 \equiv 16, 133 \bmod 294, 409$.

5. My RSA public key is (469478487497204305296287390 81, 372674862636792350620645369 73). Convert the message

Give me an A

into a number, using ASCII in the natural way. (So for "A b": in ASCII, A = 65, space = 32, and b = 98; translating each number into 8 bits gives "A b" = 010000010010000001100010 in binary.) Encode the message as though you were sending it to me using my RSA key, and write for me the corresponding encoded message in decimal.

Answer: First we convert every character into its corresponding ASCII number and then into 8 bits. This is shown in Table 1.

Table 1: Values of each character in the message

Character	ASCII	Binary
G	71	0100 0111
i	105	0110 1001
v	118	0111 0110
e	101	0110 0101
	32	0010 0000
m	109	0110 1101
e	101	0110 0101
	32	0010 0000
a	97	0110 0001
n	110	0110 1110
	32	0010 0000
A	65	0100 0001

If we append each binary string after another, we get the string in equation (3).

$$\begin{aligned} &010001110110100101110110011001010010000001101101 \\ &011001010010000001100001011011100010000001000001 \end{aligned} \tag{3}$$

The binary number in equation (3) of 96 bits (8 bits per 12 characters), represents the number $x = 22{,}100{,}932{,}013{,}077{,}800{,}977{,}026{,}654{,}273$, which can be obtained by using the java class *java.math.BigInteger*. If we consider the public key (n, e) given, then message to be sent is in the range $[1, \ldots, n]$. So we just need to compute $e(x) = x^e \bmod n$, which yields $e(x) = 27{,}016{,}764{,}340{,}118{,}192{,}395{,}712{,}492{,}378$. The above result was computed using *java.math.BigInteger#modPow* which internally breaks the operations and then it uses the Chinese Remainder Theorem to combine results.