CS 127/CSCI E-127: Introduction to Cryptography

Problem Set 1

Assigned: Sep. 6, 2013 Due: Sep. 13, 2013 (5:00 PM)

Justify all of your answers. See the syllabus for collaboration and lateness policies. Submit solutions by email to `mbun@seas` (and please put the string "CS127PS1" somewhere in your subject line).

Problem 1. (Expectations)

a) Let X be a random variable that takes non-negative integer values. Prove that $\mathrm{E}[X] = \sum_{i=1}^{\infty} \Pr[X \geq i]$. (Hint: define $\{0,1\}$-valued random variables X_i, where $X_i = 1$ iff $X \geq i$.)

b) Suppose we have a random experiment that "succeeds" with probability p, and we repeat independent trials of the experiment until we obtain the first success. Show that the expected number of trials is $1/p$.

Solution.

a) Define indicator random variables X_i, as described if $X = x$, then X_i is 1 for $1 \leq i \leq x$ and 0 for $i > x$. So

$$E[X] = \sum_{i=1}^{\infty} X_i$$

and by applying linearity of expectations:

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{\infty} X_i\right] = \sum_{i=1}^{\infty} \mathrm{E}[X_i] = \Pr[X \geq i].$$

The last equality follows because the expectation of the indicator of an event is just the probability of the event, i.e. $\mathrm{E}[I_A] = \Pr[A]$

b) We apply part a) to the random variable X counting the number of trials until the first success. For any i, $\Pr[X \geq i]$ is the probability that the first $i-1$ trials are failures, which by independence of the trials is $(1-p)^{i-1}$. Thus, $\mathrm{E}[X]$ is the geometric sum

$$\mathrm{E}[X] = \sum_{i=1}^{\infty} (1-p)^{i-1} = \frac{1}{p}$$

.

Problem 2. (Arbitrary Random Choices from Coin Flips) Often we describe randomized algorithms as making random choices from arbitrary sets, but sometimes it will be convenient to assume that we only make use of fair coin tosses (i.e. random bits).

Consider the following methods for generating a random number in the interval $\{0, \dots, N-1\}$. In each, we let $n = \lfloor \log_2(N-1) \rfloor + 1$ be the bit-length of $N-1$ and let $b_{n-1}b_{n-2}\cdots b_0$ be the binary representation of $N-1$ (so $b_{n-1} = 1$).

1. Use n coin tosses to generate a random number M between 0 and $2^n - 1$. If $M < N$, output M. Otherwise repeat.

2. For $i = n-1$ down to 0, do the following:

   - If $b_i = 1$ or there is a $j > i$ such that $c_j < b_j$, then use a coin toss to generate $c_i \overset{R}{\leftarrow} \{0,1\}$.
   - Otherwise set $c_i = 0$.

   Output $c_{n-1}c_{n-2}\cdots c_0$ (interpreted as a binary number).

3. Use $n+10$ coin tosses to generate a random number M between 0 and $2^{n+10} - 1$. If $M < N \cdot \lfloor 2^{n+10}/N \rfloor$, output $(M \bmod N)$. Otherwise, repeat.

For each of the above methods, (a) say whether its output is uniformly distributed in $\{0, \dots, N-1\}$, and (b) compute the expected number of coin tosses used. Which method would you prefer if N is a "typical" 128-bit number? Why?

(Extra Credit!) Show that for every k that there is no procedure that samples uniformly from $\{0, 1, 2\}$ while *always* using at most k coin flips.

Solution.

1. The output is uniform. In any given trial the probability that $M < N$ is 2^-n, so each number in N-1 is equally likely to be generated. Since the output is uniform conditioned on any number of trials being required, the output of the whole algorithm is uniform. The probability success is any trial $N/2^n$, and n coin flips are needed to in each trial, so by the result of Problem 1b) the expected number of coin flips is $n2^n/N$.

2. The output is not uniform (unless $N = 2^n$).) As a simple counterexample, let $N = 3$, so the bit representation $N-1$ is 10. If the first coin toss c_1 comes up 0, then this method generates 0, so the probability of generating 0 is $\frac{1}{2} \neq \frac{1}{3}$.

   The expected number of coin flips depends rather delicately on the bit representation of $N-1$. The trick is just to find the right recurrence that relates the expected number of flips at iteration i to be the expected number of flips at iteration $i-1$ (note that the iterations decrease from $n-1$ to 0 as the algorithm progresses).

   For $0 \leq i \leq n$, define a function $f : 0, 1^i - > \int$ where $f(b_{i-1}, ..., b_0)$ as the expected number of coin tosses needed during iterations $i-1, ...0$ conditioned on $c_j \geq b_j$ for all $j > i$. We will show that f is well-defined (i.e. does not depend on the bits $b_{n-1}, ..., b_i$) and compute its value

$$f(b_i - 1, ..., b_0) = \sum_{j : b_j = 1} 2^{j-i}(j+2)$$

by induction on i. Clearly $f(0) = 0$ and $f(1) = 1$. We now compute $f(b_{i-1}, ..., b_0)$ in terms on $f(b_{i-2}, ...b_0)$. If $b_{i-1} = 0$, then c_i is a coin toss. If the coin comes up 0, then $c_{i-1} < b_{i-1}$ and every subsequent iteration requires a coin toss. On the other hand, if the coin toss comes up 1, then the expected number of coin tosses from iterations $i-2, ...0$ is $f(b_{n-2,...,0})$. Thus

$$\begin{aligned} &f(1, b_{i-2}, ..., b_0) \\ &= \frac{1}{2}i + \frac{1}{2}(1 + f(b_{i-2}, ..., b_0)) \\ &= 2^{-1}((i-1)+2) + 2^{-1} \sum_{\substack{0 \le j \le i-2 \\ b_j = 1}} 2^{j-i+1}(j+2) \\ &= \sum_{j : b_j = 1} 2^{j-i}(j+2) \end{aligned}$$

If $b_{i-1} = 0$, then by the definition of f, c_{i-1} is set to 0. So $f(0, b_{i-2}, ..., b_0)$ as we wanted to show.
Plugging in $i = n$ yields the formula

$$\sum_{j : b_j = 1} 2^{j-n}(j+2)$$

for the expected number of coin tosses. The sum is maximized when $b_j = 1$ for each j, and takes the value n. It is minimized when $b_j = 0$ for every $j \neq n-1$, where it takes the value $(n+1)/2$. Note also that this method always uses at most n coin tosses. Generalizing the extra credit problem shows that unless $N = 2^n$, no method with this property can actually sample a uniformly random number.

3. The output is uniform. In any trial, any number $M' < N$ is generated whenever $M = M', M'+N, ..., M'+N(\lfloor 2^{n+10}/N \rfloor - 1)$, and thus generated with probability $2^{n+10}/N\rfloor/2^{n+10}$.. So as in part a), each such number is generated with equal probability. The probability of success in any round is $N \times \lfloor 2^{n+10}/N \rfloor / 2^{n+10}$, and $n+10$ coins are tossed in each round, so by problem 1b) the expected number of coin tosses is $(n+10)2^{n+10}/(N\lfloor 2^{n+10}/N \rfloor)$.

Problem 3. (Vigenère Cryptanalysis) 19th century writer and mathematician Charles L. Dodgson (Lewis Carroll) famously asserted that the Vigenère cipher was "unbreakable." Ironically, it was some of his contemporaries who devised generic methods of cracking the cipher. Just how confident should Carroll have felt about passing around, say, encrypted preprints of *Alice in Wonderland*?

a) Write a fully automated program for breaking **shift ciphers** of plaintexts written in English. The table of frequencies of English words from page 13 of Katz-Lindell (1st ed.) has been reproduced in the provided file `english_frequencies.txt` for your convenience. You may use the strategy suggested on pages 13–14 of Katz-Lindell (1st ed.) and/or your own ideas. In particular, we encourage you to experiment with different measures of distance between letter distributions.

   In prose, describe what your program does and what the main ideas are, and justify your major design choices. Attach the source code in an appendix to your submission.

b) Run your program on the challenge ciphertexts contained in `shift_ciphers.txt`. It's ok if your program can't decrypt all of them on its own. Submit the output of your program.

c) Building on your program for shift ciphers, write a program for breaking the **Vigenère cipher** for known key length. (As an extra challenge, you can try to decrypt under unknown key lengths.)

   In `alice.txt`, we have provided you with the text of *Alice in Wonderland*, divided into blocks of 20 characters. Pick a random Vigenère key k of length 3. Find the minimum number of blocks that, when encrypted under k, can be successfully decrypted by your program. Repeat this exercise for some different key lengths ranging from 1 to 1000, and draw a plot of key length against the number of blocks required.

   The full text of *Alice in Wonderland* is 107,720 characters long (without spaces or punctuation). About how long an encryption key would Carroll have needed to secure his book from your program?