6.856 — Randomized Algorithms

David Karger

Handout #4, September 17, 2002 — Homework 1 Solutions

M.R. refers to this text:

Motwani, Rajeez, and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge: Cambridge University Press, 1995.

Problem 1 MR 1.1.

(a) We rely on the fact that flips are independent, even if biased. To try to generate one bit, flip the coin twice. If you get heads followed by tails (HT) then output heads. If tails followed by heads (TH) output tails. If HH or TT, report failure.

Conditioned on having succeeded (gotten HT or TH) the probability that you got HT is equal to the probability you got TH, so the output you produce is unbiased.

If you fail, you try again, repeating until you succeed. The probability that you succeed in one trial is just $2p(1-p)$ (probability of one head and one tail). So the number of trials you perform before succeeding has a geometric distribution; its expectation is $1/[2p(1-p)]$. Since we toss the coin twice at each trial, the expected number of total coin tosses is $1/[p(1-p)]$.

(b) The following solution owes to [Elias72]. This one is tricky. Any reasonable effort is sufficient for full credit.

Before we tackle the problem itself, let us consider the following scenario: suppose you are given a number r drawn uniformly at random from $\{1, \ldots, N\}$. How many unbiased bits can you output from such a sample? The following scheme will turn out to be asymptotically optimal:

Suppose that the binary representation of N is as follows:

$$N = \alpha_m 2^m + \alpha_{m-1} 2^{m-1} + \cdots + \alpha_1 2^1 + \alpha_0 2^0,$$

where $\alpha_i \in \{0, 1\}$, $\alpha_m = 1$, and $m = \lfloor \log N \rfloor$.

We assign output strings to the N numbers, so that for every $i > 0$ with $\alpha_i = 1$, we map 2^i of the numbers to all the binary strings of length i. The exact assignment does not matter, but since every input number is equally likely, we are assured that the output bits are unbiased and independent random bits. Note that if N is odd, i.e. $\alpha_0 = 1$, one of the input numbers will not be assigned to any output – if we encounter it, we output nothing. Luckily, this case will become increasingly unlikely as N grows.

It remains to be seen how many bits we expect to output using this scheme. Due to the above construction, if $\alpha_i = 1$ for some $i > 0$, then we output a length i bit string with

probability $2^i/N$. So the expected output length is equal to

$$\begin{aligned} E_N := E[\ \#\text{ bits output }] &= \sum_{i=0}^{m} i \cdot \alpha_i \cdot \frac{2^i}{N} \\ &= \frac{1}{N}\left(\sum_{i=0}^{m} m \cdot \alpha_i \cdot 2^i - \sum_{i=0}^{m}(m-i)\cdot\alpha_i\cdot 2^i\right) \\ &= m - \frac{1}{N}\sum_{i=0}^{m}(m-i)\cdot\alpha_i\cdot 2^i \end{aligned}$$

We have $N \geq 2^m$, and $\sum(m-i)\alpha_i 2^i \leq \sum(m-i)2^i = 2^{m+1} - m \leq 2^{m+1}$. This implies

$$\log N \geq m \geq E_N \geq m - \frac{2^{m+1}}{2^m} = m - 2 \geq \log N - 3$$

Let us now apply this observation to the problem we really want to solve, extracting bits from n biased coin flips. Suppose we flip the coin n times and see k heads. There are $\binom{n}{k}$ different different positions those k heads can take among the n flips (that is, which of the k flips came up heads), and each is equally likely. So, using the above scheme, we can map them to bit sequences with an expected length of about $\log\binom{n}{k}$. Since the probability that we see k heads is $p^k(1-p)^{n-k}\binom{n}{k}$, we can conclude that the expected number E of output bits is

$$\sum_{k=0}^{n} p^k(1-p)^{n-k}\binom{n}{k}\log\binom{n}{k} - 3 \leq E \leq \sum_{k=0}^{n} p^k(1-p)^{n-k}\binom{n}{k}\log\binom{n}{k} \qquad (1)$$

To bound this quantity will use the following two facts:

Fact 1 (Weak law for binomial distributions) *For all $p \in [0,1]$, $\varepsilon > 0$ there exists $n_0 \in \mathbb{N}$, such that for all $n \geq n_0$, a $(1-\varepsilon)$ fraction of all length n sequences of p-biased coin flips contain between $n(p-\varepsilon)$ and $n(p+\varepsilon)$ heads.* □

Fact 2

$$\lim_{n\to\infty} \frac{1}{n}\log\binom{n}{n\cdot p} = H(p) := p\cdot\log\frac{1}{p} + (1-p)\cdot\log\frac{1}{1-p} \quad \square$$

This second fact follows easily from $\log n! \approx n\log n$ – a consequence of Stirling's Formula; see Proposition B.1 in the MR book.

This first fact implies that (1) asymptotically grows like $\log\binom{n}{np}$,[1] which according to

[1]More precisely, that

$$(1-\varepsilon)\cdot\min_{|\alpha|\leq\varepsilon}\log\binom{n}{n(p+\alpha)} - 3 \leq E \leq (1+\varepsilon)\cdot\max_{|\alpha|\leq\varepsilon}\log\binom{n}{n(p+\alpha)}.$$

the second fact converges to $nH(p)$,[2] which proves that our scheme obtains the optimum expected output length.

(c) This is a (the) classic result from information theory [Shannon48]. The basic idea behind the proof is the following. By Fact 1 from part (b), we know that as n becomes large, almost all sequences have about np heads in them. So the input can be approximated by just $\binom{n}{np}$ sequences of equal probability. The 'best' we can do to get unbiased bits is to map all of these sequences to different output strings from some $\{0,1\}^k$. Since we only have $\binom{n}{np}$ sequences to map from, we have that $k \leq \log \binom{n}{np}$, which by Fact 2 from above implies that $k \leq nH(p)$ in the limit.

Problem 2 MR 1.5.

(a) Let's partition the unit interval into n intervals I_i of length p_i. If we could generate a number U chosen uniformly at random from the unit interval, then it would fall into interval I_i with probability p_i, so we could output item i in that case. Since we don't want to generate the infinitely many bits needed to specify a number in the unit interval, we perform a lazy evaluation, generating only as long a prefix of U as we need to be certain that U falls in a particular interval I_i (this can be thought of as an application of the principle of deferred decision: we generate all the bits of U, but the only examine as many as we need to make a decision).

To decide how many bits of U we will generate/look at, let us consider the probability that we will need to generate more than k bits. Suppose we have generated k bits of U, call them U_k. Based on U_k, we know that U lies in a particular one of 2^k equal-length subintervals of the unit interval, call it V. If this subinterval V is contained in one of the I_i, then we know that U is contained in I_i and we can stop generating bits. This can fail to happen only if an endpoint of some I_i is contained in V. Now note that with n intervals I_i, there are only $n-1$ interval endpoints. Thus, at most $n-1$ of the equal-length subintervals V can contain an endpoint. Since V is chosen uniformly at random from among the possibilities, the probability this happens is at most $(n-1)/2^k$. Thus the probability that we need more than k bits is at most 1 if $k \leq \log n$, and at most $(n-1)/2^k$ if $k > \log n$. It follows that if R is the number of bits needed, then

$$\begin{aligned} E[R] &= \sum_r \Pr[R \geq r] \\ &\leq \log n + \sum_{r \geq \log n} (n-1)/2^r \\ &= O(\log n) \end{aligned}$$

[2] Again, more precisely, we get

$$(1-\varepsilon) \cdot \min_{|\alpha| \leq \varepsilon} nH(p+\alpha) - 3 \leq E \leq (1+\varepsilon) \cdot \max_{|\alpha| \leq \varepsilon} nH(p+\alpha)$$

for sufficiently large n. Letting ε tend to 0 gives $\lim_{n \to \infty} E/n = H(p)$.

We can actually get a somewhat more sophisticated bound as follows: let us condition on the fact that $U \in I_j$. Under this assumption, how many bits of U do we need to reveal (in expectation) to know that $U \in I_j$? Arguing much as above, note that k bits of U specify an interval of length $1/2^k$. Conditioned on the fact that $U \in I_j$, U_k basically specifies an interval V that intersects I_j. There are $2^k p_j$ such intervals, so the probability that we have to generate more bits, which is just the probability that V contains an endpoint of I_j, which is roughly $2/(2^k p_j)$. So arguing as above, the expected number of bits needed (given that we are in I_j) is $\log(1/p_j)$. It follows that

$$\begin{aligned} E[R] &= \sum_j \Pr[I_j] E[R \mid I_j] \\ &= \sum_j p_j \log(1/p_j) \end{aligned}$$

which is just the entropy of the distribution. This is simply taking MR 1.1 in the opposite direction (from unbiased to biased bits).

(b) Suppose that we have an algorithm using k bits in the worst case to choose uniformly from $\{1, 2, 3\}$. We can think of this algorithm as a function that, for any sequence of random bits of size k, maps to an output — 1, 2, or 3 (the algorithm may not pay attention to all k random bits, but we assume it pays attention to no more than these bits). Suppose that r of the k-bit sequences lead the algorithm to output "1." Then the probability that the algorithm outputs "1" with probability $a/2^k$. This number has a finite-length base-2 representation (length k to be precise). However, "1" is actually supposed to be output with probability 1/3. a number that does *not* have a finite-length base 2 representation. Thus, the algorithm cannot be outputting "1" with the correct probability.

Many students did not give a complete explanation as to *why* numbers that don't have a finite base two representation would cause problems. Alot of arguments said the right things, but failed to indicate in some way that the argument works independent of the scheme.

(c) Consider the following algorithm:

1. Set interval $[a, b] \leftarrow [0, 1]$
2. Repeat for each coin toss:

2a. If coin comes up heads, $[a, b] \leftarrow [a, a + p(b - a)]$

2b. If coin comes up tails, $[a, b] \leftarrow [a + p(b - a), b]$

2c. Output the bits in which the binary expansions of a and b agree (and that have not been output before).

This algorithm recursively subdivides the unit interval $[0, 1]$ based on the coin flips we encounter. For each coin flip, the current interval is subdivided in the ratio $p : (1 - p)$; if we see heads, we go the left interval, otherwise to the right side.

If we read an infinite sequence of coin flips, our interval will eventually converge to a single point. We can view the sequence as encoding this particular point x. Our output is the binary representation of x: as our interval $[a, b]$ becomes smaller, we output the digits of x as we get to know them (since the lower bound a and the upper bound b agree on them).

During the execution of this algorithm, the size of each interval $[a, b]$ is equal to the probability of actually ending up in it. Therefore, the sequence is equally likely to converge to any point in $[0, 1]$. Thus, the bits that are output are unbiased and independent.

While this scheme is conceptually simpler than the one from problem 1b, its analysis involves a combination of the techniques we used for problems 1b and 2a.

Suppose we read n biased coin flips, where n is sufficiently large.[3] What is the expected number of bits that were output? Let $\varepsilon > 0$. Using Fact 1 from problem 1b, we know that with probability at least $(1 - \varepsilon)$, the input sequence has between $n(p - \varepsilon)$ and $n(p + \varepsilon)$ heads.

By choosing n sufficiently large, we can assume that any $\binom{n}{np}$ of these sequences contribute only an insignificant weight, i.e. (assuming $p \geq 1/2$)

$$\binom{n}{np} p^{np+\varepsilon}(1-p)^{n(1-p)-\varepsilon} \leq \varepsilon \tag{2}$$

Now we can determine the probability that we will *not* output at least $n \cdot H(p)$ bits after these n steps. Given some interval $[a, b]$, we will obviously output less than $nH(p)$ bits iff the binary representations of a and b differ in one or more of their first $nH(p)$ digits. This is the case if there is some number of the form

$$\frac{m}{2^{\lfloor nH(p) \rfloor}}, \quad m \in \left\{0, 1, \ldots, 2^{\lfloor nH(p) \rfloor} - 1\right\}$$

within the interval $[a, b]$. Since all of our intervals are disjoint, at most $2^{nH(p)}$ of them can contain any of these 'bad' numbers. By Fact 2 from problem 1b, we have $2^{nH(p)} \approx \binom{n}{np}$ for large enough n – but we already know from (2) that this many intervals only contribute an insignificant weight. Thus, the probability that we actually output at least $nH(p)$ bits, is at least $(1 - \varepsilon) - \varepsilon = 1 - 2\varepsilon$. By letting ε tend to 0, this shows the claim.

Problem 3 Packets in a stream.

We use the following algorithm:

- Save the first k packets you see.
- After this, for the n-th packet, with probability k/n save this packet at a random position in our storage (jettisoning the packet that was already there).

[3] Note that if we stopped our algorithm after n steps, the output bits would *not* be unbiased. In this problem we are only interested in the *rate* with which the algorithm (that necessarily runs forever) produces output bits.

We claim that at any point our storage will contain a random sample from the already seen packets. We give a proof by induction, the base case $n \leq k$ being trivial. So assume that after $n-1 \geq k$ steps we have a random sample among the first $n-1$ packets. We have to show that after seeing (and possibly storing) the n-th packet, we now have a random sample among the first n packets. We will identify the n packets with the numbers $\{1, \ldots, n\}$ for the following.

First, we note that $\binom{n-1}{k-1}/\binom{n}{k} = k/n$ of k-element subsets of $\{1, \ldots, n\}$ contain the number n. This implies that we can choose a random k-element subset from $\{1, \ldots, n\}$ as follows:

$$\text{Subset} = \begin{cases} \{n\} \cup \text{“random } k-1\text{-element subset of } \{1, \ldots, n-1\}\text{”}, & \text{with probability } \frac{k}{n} \\ \text{“random } k\text{-element subset of } \{1, \ldots, n-1\}\text{”}, & \text{with probability } 1 - \frac{k}{n} \end{cases}$$

But this is exactly what our algorithm does: with probability k/n it includes n in the sample. The other elements are chosen by omitting a random element from (by induction hypothesis) a random k-element subset of $\{1, \ldots, n-1\}$, which yields a random $(k-1)$-element subset of $\{1, \ldots, n-1\}$. And with probability $1 - k/n$, we just leave the sample like it is, i.e. a random k-element subset of $\{1, \ldots, n-1\}$. This implies that after n steps we are still left with a random sample.

Note that to show that we have a random sample, it is *not* enough to prove that each packet is in the sample with probability k/n. One also has to show that these probabilities are *independent*. This was a very common mistake.

References

[Elias72] Peter Elias, The efficient construction of an unbiased random sequence, *The Annals of Mathematical Statistics*, 43(3):865–870, 1972.

[Shannon48] Claude E. Shannon, A Mathematical Theory of Communication, 1948.