Notes on Generating Random Permutations

January 24, 2008

(Adapted partially from Cormen *et al*'s *Introduction to Algorithms*.

Problem Statement: We want to generate permutations of $1 \dots n$ uniformly at random, meaning each permutation has probability $1/n!$ of occurring. Note that this allows us a general means to permute any n elements, say of an array, by permuting the indices of the array.

The first algorithm we consider (Algorithm 1) is obviously correct. Any output is a legal permutation, and the probability of any given permutation is $1/n!$ because the elements are selected completely at random at each step. The running time is more complicated to analyze. We'll consider it *on average*. First let's consider the question of how many random draws we would expect to have to make in order to generate a new number for some j.

j of the n numbers would be duplicates, so the probability of success (generating a new number) is $\frac{n-j}{n}$. Then the expectation of the number of trials until the first success is $\frac{n}{n-j}$ (this is a standard result – when you have Bernoulli trials with probability p of sucess the expected number of trials until first success is $1/p$ – look up the geometric distribution if you are interested in the derivation.

So now, for the running time we get:

$$\sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{j=1}^{n} \frac{1}{j} = O(n \log n)$$

Keep in mind that there is always some chance that the algorithm will not terminate by any specified finite time T.

Input: n
for i in 1 **to** n **do**
  used[i] = false
end for
for j in 0 **to** $n-1$ **do**
  repeat
    temp = randInt(1,n)
    if not(used[temp]) **then**
      a[j] = temp
      used[temp] = true
    end if
  until a[j] is filled
end for

Algorithm 1: Not so great algorithm for permutation

Input: n
for i in 0 **to** $n-1$ **do**
  a[i] = i+1
end for
for j in 0 **to** $n-1$ **do**
  swap(a[j], a[randInt(j,n-1)])
end for

Algorithm 2: Better algorithm for permutation

Algorithm 2 is a better algorithm. The running time is obvious: $O(n)$. Proving correctness is not so easy. First, define a k-permutation of a set of n elements as a sequence containing k of those n elements. There are $\frac{n!}{(n-k)!}$ such permutations (how?).

We now introduce the following *loop invariant:*

Prior to the jth iteration, $a[0 \ldots j-1]$ contains each j-permutation with probability $\frac{(n-j)!}{n!}$.

Convince yourself that the loop invariant holds for $j = 0, 1$. Now that we have the base case done, we have to show that the loop invariant is maintained at each iteration. Let's say iteration j ends with: $\langle x_0, x_1, \ldots, x_j \rangle$ Define the following two events

Event E_1 is that the first $j-1$ iterations have yielded $\langle x_0, \ldots, x_{j-1} \rangle$

Event E_2 is that the jth iteration puts x_j in position j

Applying Bayes rule,

$$\Pr(E_2 \wedge E_1) = \Pr(E_2 | E_1) \Pr(E_1)$$

$$= \frac{1}{n-j} \frac{(n-j)!}{n!}$$

$$= \frac{(n-j-1)!}{n!}$$

This proves that the loop invariant is maintained.

Finally, at termination, $j = n$, so a contains each n-permutation with probability $(n-n)!/n! = 1/n!$.