CS174 Lecture 2 John Canny

Monte-Carlo vs. Las Vegas

A random algorithm is **Las Vegas** if it always produces the correct answer. The running time depends on the random choices made in the algorithm. Random Quicksort (where pivot elements are chosen at random) is a Las Vegas algorithm. It always sorts the elements correctly, but its running time depends on the choices of pivot.

A random algorithm is **Monte Carlo** if it can give the wrong answer sometimes. The running time usually doesn't depend on the random choices, but for some Monte Carlo algorithms it does.

A useful mnemonic that someone proposed is "Las Vegas will never cheat you". Why they thought that Las Vegas is more trustworthy than Monte Carlo is a mystery.

From the definitions, you can see that a Las Vegas algorithm is also a Monte Carlo algorithm (Monte Carlo is a broader definition because nothing is guaranteed). To turn a Monte-Carlo algorithm into a Las Vegas algorithm, you need to add a means for checking if the answer is correct and retrying if it isn't.

Random Permutations
Recall that a permutation π is one-to-one and onto function

$$\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$$

A permutation defines a reordering of the elements $1, \ldots, n$. It can be specified by the new values. e.g. $(2\ 4\ 3\ 5\ 1)$ specifies a permutation where $\pi(1) = 2$, $\pi(2) = 4$, $\pi(3) = 3$, $\pi(4) = 5$ and $\pi(5) = 1$.

There are $n!$ permutations of n elements. That's too many to generate them by numbering them all and choosing one at random. Random permutations are quite useful in randomized algorithms. So its helpful to have efficient algorithms for generating them. But more than this, certain algorithms give us good characterizations of random permutations for use in proofs. An algorithm breaks a calculation down into small steps. Those steps can be convenient to use in proofs of correctness for randomized algorithms. We'll see a concrete example of this shortly.

Generating a random permutation is like "unsorting". A sort program can make an arbitrary permutation of elements to place them in ascending order. If the swaps in a sorting program are replaced by random swaps in a consistent way, we obtain a random permutation generator. If done carefully, we can generate random permutations with the uniform distribution.

One such algorithm which gives a uniform distribution on permutations, we can use the following method. Assume an array A with n elements, initialized to the identity permutation so that $A[i] = i$ for all i.

```
RandomPerm(A, n)
for i = 1 to n-1
   choose j uniformly at random from [i,...,n]
   swap A[i] and A[j]
```

Claim 1: This algorithm can produce any permutation of $\{1, \dots, n\}$.

Suppose The array A is initialized with elements that are the representation of a permutation, that is $A[i] = \pi(i)$. If we sort the array A with selection sort, let the final contents of array be A'. The contents of A' are just $1, \dots, n$ sorted, so $A'[i] = i$. The permutation we have just made on the elements of A is exactly π, because we have moved the element in position i to $\pi(i)$. To put it another way, $A[i] = A'[\pi(i)]$.

Now observe that whatever the initial order of the elements in the array A, the random permutation algorithm might sort them in increasing order. It will do this if every random choice of j happens to pick the smallest element in the sequence $A[i], \dots, A[n]$. In that case, the permutation algorithm is actually implementing selection sort. The permutation algorithm doesnt look at the contents of A when it runs. The contents of A might represent the permutation π (in other words $A = A'$ as defined earlier). If it is, the random permutation algorithm might sort the array in increasing order. As we saw above, this corresponds to applying the permutation π to the elements of A. In summary, for any permutation , the random permutation algorithm has non-zero probability of applying the permutation π to the elements of A. QED

Claim 2: Every permutation of $\{1, \dots, n\}$ is generated with equal probability $1/n!$ by algorithm RandomPerm.

Proof:
There is exactly one set of choices of the random j values that will produce any given permutation. To see that, notice first that there are

$$n(n-1)(n-2)\cdots = n!$$

possible combinations of random j values. Since there are $n!$ permutations of n elements, there is a one-to-one correspondence between the algorithm's permutations and all the permutations of $1, \dots, n$.

The probability of a particular choice of j on the first iteration is $1/n$, then $1/(n-1)$ on the second iteration. So the probability of particular tuple of j values which defines a unique permutation is:

$$1/n \times 1/(n-1) \times 1/(n-2) \cdots = 1/n!$$

That's the uniform distribution on permutations of $1, \dots, n$. QED

Fixed Points in a Permutation
Here's the first application of "the probabilistic method" that's a theme of the course. We first define a random variable X which counts the quantity we are interested in:

$X =$ number of fixed points in the permutation π , which is the cardinality of the set $\{i | \pi(i) = i\}$

Since π is a random permutation, we are interested in the expected value of X. We could try to compute $\mathrm{E}[X]$ directly, but there is a simpler way. Define

$$X_i = \begin{cases} 1 & \text{if } \pi(i) = i \\ 0 & \text{otherwise} \end{cases}$$

Then notice that

$$X = \sum_{i=1}^{n} X_i$$

Now we can apply the linearity of expected value that we proved last time to show that

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

There is obvious symmetry between permutations that fix particular elements, so all the $\mathrm{E}[X_i]$ are equal. Therefore

$$\mathrm{E}[X] = n\mathrm{E}[X_i]$$

and computing $\mathrm{E}[X_i]$ is easy because its an indicator (0-1-valued) random variable:

$$\begin{aligned} E[X_i] &= 0 \times \Pr[X_i = 0] + 1 \times \Pr[X_i = 1] \\ &= \Pr[X_i = 1] \end{aligned}$$

which is just the probability that $\pi(i) = i$. This is easy to figure by counting permutations that fix i and dividing by all permutations:

$$\Pr[X_i = 1] = (n-1)!/n! = 1/n$$

so

$$\mathrm{E}[X] = nE[X_i] = n \times 1/n = 1$$

So we get the rather surprising fact that the expected number of fixed points in a random permutation is 1, independent of how many elements are being permuted.

Counting cycles in a random permutation

A harder and more interesting question is the following: What is the expected number of cycles for a random permutation?

As before, we use a random variable:

$$Y = \text{number of cycles in the permutation } \pi$$

Defining indicator random variables is trickier this time. We can't count 1 for each point in each cycle, or we will just get a total of n. We would be counting points rather than cycles.

Since a cycle of length k is shared among k points, the logical thing to do is to have each point contribute $1/k$. Then the total contribution from all points in the cycle of length k is 1. Hence we define:

$$Y_i = 1/k \text{ where } k \text{ is the length of the cycle containing point } i$$

Check for yourself that the Y_i are not independent. Luckily we wont need independence in this analysis.

Now verify that

$$Y = \sum_{i=1}^{n} Y_i$$

and using linearity of expected value, we have that

$$\mathrm{E}[Y] = \sum_{i=1}^{n} \mathrm{E}[Y_i]$$

Now all the $\mathrm{E}[Y_i]$ are equal, therefore

$$\mathrm{E}[Y] = n\mathrm{E}[Y_i]$$

and

$$\mathrm{E}[Y_i] = \sum_{j=1}^{n} 1/j \mathrm{Pr}[i \text{ is in a cycle of length } j]$$

We can now use our random permutation algorithm. wlog assume i is 1. Then i is in a cycle of length 1 iff RandomPerm chooses $j = 1$ on the first iteration. The probability of this choice is just $1/n$, so:

$$\mathrm{Pr}[i \text{ is in a cycle of length 1}] = 1/n$$

We can use the same idea for cycles of length two. The permutation algorithm must map i to some point different from itself, (prob. $= (n-1)/n$) and then map that point back to i, (prob. $= 1/(n-1)$). Therefore:

$$\mathrm{Pr}[i \text{ is in a cycle of length 2}] = (n-1)/n \times 1/(n-1) = 1/n$$

similarly

$$\mathrm{Pr}[i \text{ is in a cycle of length 3}] = (n-1)/n \times (n-2)/(n-1) \times 1/(n-2) = 1/n$$

and in general

$$\mathrm{Pr}[i \text{ is in a cycle of length } j] = 1/n \qquad \text{for any } j$$

This is a surprising result: the length of the cycle containing the point has the uniform distribution! i.e. any point is equally likely to be in a cycle of any length up to n.

Going back to the expected number of cycles and substituting for this probability, we get:

$$\mathrm{E}[Y_i] = \sum_{j=1}^{n} 1/j \times 1/n = (1/n)H_n$$

where $H_n = (1 + 1/2 + 1/3 + ... + 1/n)$ is the n^{th} harmonic number. So

$$\mathrm{E}[Y] = n\mathrm{E}[Y_i] = H_n$$

Now H_n is approximately the natural log of n, $\ln n$. So the expected number of cycles in a permutation grows as the natural log of the number of elements.