An Experimental Assignment on Random Processes

Michael Mitzenmacher*

December 27, 2000

1 Introduction

Although I still generally use standard problem sets in my theory classes, I find myself experimenting more and more with alternatives, including more open-ended assignments. One big motivation for this change is to put some *science* back in my computer science class, by making assignments more like laboratory exercises in chemistry and biology: students have to experiment to find an answer. One of the great things about studying randomness is that the results sometimes lend themselves quite naturally to experimental assignments, as is the case for the assignment I describe here. Before going on, I will present the assignment.

2 The assignment

Consider a complete binary tree with $N = 2^n - 1$ nodes. Here n is the depth of the tree. Initially, all nodes are *unmarked*. Over time, through processes we shall describe, nodes becomes *marked*.

All of the processes share the same basic form. Each unit of time, I *send* you a node. When you receive a sent node, you mark it, if it is not already marked. Also, you invoke the following *marking rule*, which takes effect before I send out the next node.

- If a node and its sibling are marked, its parent is marked.
- If a node and its parent are marked, the other sibling is marked.

The marking rule is applied recursivly as much as possible before the next node arrives. For example, in the picture below, the marked nodes are filled in. The arrival of the node labelled by an X will allow you to mark the remainder of the nodes, as you apply the marking rule first up and then down the tree. Keep in mind you always apply the marking rule as much as possible.

Now let us consider different ways that I might be sending you the nodes:

Process 1: Each unit of time, I send a node chosen *independently and uniformly at random from all of the N nodes*. Note that I might send you a node that is already marked, and in fact I may send a useless node that I have already sent. Almost surely some nodes will arrive many times!

*Harvard University, Computer Science Department. 33 Oxford St., Cambridge, MA 02138. Supported in part by NSF CAREER Grant CCR-9983832 and an Alfred P. Sloan Research Fellowship. E-mail: michaelm@eecs.harvard.edu.

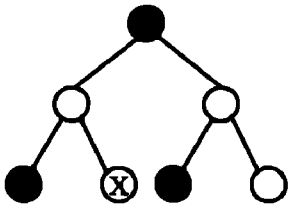

Figure 1: The arrival of X causes all other nodes to be marked.

Process 2: Each unit of time I send a node chosen uniformly at random from those nodes *that I have not yet sent.* Again, a node that has already been marked might arrive, but each node will be sent at most once.
Process 3: Each unit of time I send a node chosen uniformly at random from those nodes *that you have not yet marked.* Keep in mind that the marking rule is applied as much as possible before I send the next node.

To begin, we want to determine formulae for the growth rate of the expected number of time steps as a function of n (or N) until all the nodes are marked for each of these processes. You should begin by writing programs to emulate the sending processes and the marking rule. A reasonable set of data to begin with would be to run each process 10 times for each value of n in the range $[10,20]$. Note that when $n = 20$, your tree will have over one million nodes. You should therefore think first about making your implementation efficient. Using this data, try to develop reasonable guesses for the expected times until all nodes are marked as functions of n.

There are also more open-ended questions we can ask. Based on your experimental results, can you offer formal or informal explanations for the behavior of the different processes? Your goal here should be to take whatever insight you have obtained from your experiments and use it to develop a mathematical proof. Try to prove at least one statement about one of the processes. A tip: you may find it helpful to have your program divulge the last node that was sent before the tree became completely marked; that may be useful information. You may find printing out other information helpful as well.

3 Discussion and partial solutions

Partial solutions follow. You may want to code up the problem yourself, just so you can see what happens before reading on.

I think the basic assignment, which just involves programming, is suitable for an introductory computer science programming class. Depending on how much you expect from the students, however, it is also suitable for more advanced classes. For example, you could use it in an advanced discrete mathematics class or a probability class, and then give a follow-up assignment with mathematical problems based on the computation.

I gave the assignment to my graduate seminar on random processes, and as part of the assignment they were expected to develop conjectures and proofs based on their experiments. One interesting thing I found was that despite multiple warnings that should strive for basic statements like "For Process 1, the expected number of nodes sent is $O(N^{2/3})$ and $\Omega(N^{1/2})$", students wasted a great deal of time looking for simple and elegant closed form solutions. Beginning graduate students need to learn that simple answers are not always so forthcoming!

I emphasize again that the goal of this assignment is to make it experimental; the students are expected

Nodes	Average steps (10 runs)	$Avg/(N\ln N)$
1023	3171.9	0.44
2047	7358.8	0.47
4095	15345.7	0.45
8191	33498.8	0.45
16383	75637.0	0.48
32767	146096.6	0.43
65535	344182.4	0.47
131071	717546.4	0.46
262143	1530538.9	0.47
524287	3238208.3	0.47
1048575	6901244.4	0.47

Table 1: Sample data, Process 1

to discover something, and perhaps be surprised. My personal motivation for writing this up is that I believe that this is approach in underutilized in computer science currently.

If anyone wants to use this assignment or some variation in a class, you have my permission and encouragement. I would ask that you let me know how it goes and if you have any suggestions for how to improve it.

3.1 Process 1

For Process 1, each time step the node that arrives is chosen uniformly at random from all N nodes. Experiments reveal that the last node that arrives is (almost) always a leaf node. This suggests the following insight: the bottleneck to marking all of the nodes is that for each pair of sibling leaves at the bottom of the tree, at least one of the pair must be hit. The expected time to hit each pair of sibling leaves is a variation on the standard coupon collector's problem. (See Chapter 3 of Motwani and Raghavan's text **Randomized Algorithms** for further information on the coupon collector's problem.) It is easily verified that asymptotically the expected time to hit each leaf sibling pair at least once is $(N\ln N)/2$. Moreover, it is easy to verify that after this many steps, with high probability all of the nodes one level above the leaves have all been marked – either directly, or by the two leaves below, or by its parent and other sibling. Once all nodes one level above the leaves have been marked, by the marking rule all nodes higher up in the tree are marked, so the bottleneck is truly the leaves.

The analysis actually yields that the expected number of time steps until all nodes are marked is approximately $(N\ln N)/2 - cN$ for some small constant c. Hence, students may have some trouble guessing the asymptotics, and a hint is reasonable.

3.2 Process 2

For Process 2, each time step the node that arrives is chosen uniformly at random from all nodes that have not yet arrived. This is easily programmed by modelling the arrival process as a random permutation of the N nodes.

Nodes	Average steps (10 runs)	$(N-Avg)/\sqrt{N}$
1023	966	1.782
2047	1993.9	1.174
4095	3977.8	1.831
8191	8037	1.702
16383	16121.2	2.045
32767	32459.6	1.698
65535	65087	1.750
131071	130371.9	1.931
262143	261385.4	1.480
524287	522849.1	1.986
1048575	1046563.2	1.965

Table 2: Sample data, Process 2

Again, experiments reveal the last node that arrives is (almost) always a leaf node, and in this case almost all the nodes must arrive before the process terminates. To explain why, note that there are $(N+1)/4$ pairs of sibling leaf nodes at the bottom the tree. Consider the last $2\sqrt{N}$ elements of the permutation. The probability that both nodes of a pair of sibling leaf nodes lie in these last $2\sqrt{N}$ elements is approximately $\left(\frac{2}{\sqrt{N}}\right)^2 = 4/N$. Hence, the expected number of pairs of sibling leaf nodes that have both nodes in the last $2\sqrt{N}$ elements of the permutation is approximately 1. We would therefore roughly expect to have to obtain at least $N-2\sqrt{N}$ nodes before all would be marked. More careful and exact arguments can obviously be made. Again, it may be useful to provide the students a hint of the asymptotics, especially as there is a reasonable amount of variance.

3.3 Process 3

For Process 3, each time step the node that arrives is chosen uniformly at random from all nodes that have not yet been marked. Interestingly, for this variation of the process, the number of time steps *is always the same*! Experiments reveal it is $(N+1)/2 = 2^{n-1}$.

This fact can be proven by induction on n, the number of levels. We assume it is true for n levels, and prove it true for $n+1$ levels. There are two cases. First, suppose the root does not arrive during the course of the process. Then each side of the tree required 2^{n-1} arrivals, for a total of 2^n. In the second case, suppose that the root does arrive during the course of the process. Without loss of generality, assume that the left child of the root is marked first. When this happens, the right child of the root is marked, and note that this happens before the node arrives. It is therefore clear that the left side of the tree requires 2^{n-1} arrivals, and the right side requires $2^{n-1}-1$ arrivals, as the arrival of the right child of the root is like a "free arrival" for the right hand side. Again, a total of 2^n arrivals are required.

Another interesting interpretation that makes the deterministic behavior of this process more comprehensible is the following. Let the leaves of tree each have a data value. The value of the parent of two nodes will just be the XOR of the values of its children. Marking a node corresponds to obtaining its value; note that the marking rule encodes the fact that when we have the values of two children, we can derive the value of the parent, and similarly for the other part of the marking rule. Now the question of when the

process finishes becomes the question of when we have enough information to derive all the data values in the tree. One may think of the N nodes as representing N equations in the $(N+1)/2$ unknown values determined by the original leaves. In this third process, at each step we get a single independent equation that cannot be derived from previous equations, since the marking rule derives as much as possible at each step. To compute the $(N+1)/2$ unknown values it is therefore necessary and sufficient to get $(N+1)/2$ equations, which takes $(N+1)/2$ time steps.

4 Acknowledgments

This assignment arose out of discussions with several people who gave insight and help, including John Byers, Michael Luby, Alan Frieze, Claire Kenyon, Micah Adler, Berhold Voecking, and Andrei Broder.