15-859(D) Randomized Algorithms

Homework # 2 **due: October 8, 1998**

Consider a database that users can query for information. It would be nice if this database could quickly provide the user with an estimate of the amount of time a search is going to take, before beginning the search. In this assignment, we will work through a fast randomized algorithm for a problem based on this issue: given a directed graph G, quickly estimate the number of nodes reachable from each vertex in G (think of G as a database where a user queries some vertex v, and then the search engine performs some operation on all nodes reachable from v.) The best exact algorithms for this problem take time $O(\min(mn, n^{2.38}))$. The randomized approximation will take time only $O(m \log n)$.

The basic idea of the algorithm is as follows. We give a random ranking to the nodes of G (that is, we label the nodes according to a random permutation of $\{1, \ldots, n\}$). Then, for each vertex v in G, we find the node of *lowest rank* reachable from v. Let's think about what we expect this rank to be: if v can reach only one node (i.e., itself), then the expected value of this rank is $(n+1)/2$. If it can reach two nodes, then the expected value of the smaller of the two ranks is $(n+1)/3$. More generally, say v can reach R_v nodes, then the expected value of the smallest rank is $(n+1)/(R_v+1)$. So, this suggests that we can use these ranks to guess at the R_v's. If we want better guesses, we run this procedure several times and combine the information received. Actually, for technical reasons (as you will see below) we will deviate from this outline slightly, but this is the basic idea.

Exercises:

1. Describe a method for using a fair coin (a source of unbiased random bits) to generate a random permutation of $\{1, 2, \ldots, n\}$, using a "quicksort" paradigm. Your method should use an expected $O(n \log n)$ random bits and take expected time $O(n \log n)$.

2. Describe a method for using a source of $O(\log n)$-bit random numbers to generate a random permutation of $\{1, 2, \ldots, n\}$, that uses an expected $O(n)$ random numbers and takes expected time $O(n)$, assuming we can perform word operations (picking a random number, indirect addressing, etc.) in unit time.

   (Note: if the notion of a "quicksort paradigm" doesn't make sense, or if you would rather try something else, then simply give two different types of algorithms that meet these bounds.)

Problems:

3. Describe a deterministic $O(m+n)$ time algorithm that given a directed graph with n nodes and m edges, will find, for every vertex, the node of least index reachable from that vertex. (So, we've now shown that each iteration of our algorithm takes only linear time.)

4. Suppose we have a subset S of a universe U of size n, and we assign a random permutation of $\{1, 2, \ldots, n\}$ to the elements of U. Prove that the expected value of the minimum element in S is $(n+1)/(|S|+1)$.

Actually, we're not going to use the result you proved in Problem 4 in the algorithm, because we're going to change the algorithm a bit for simplicity. Let's assume that the way that we generated our random ranking was that for each node in our graph G, we picked a random real number in the range $[0, 1]$ and then we sorted the nodes accordingly. (This is along the lines of Exercise 2; for additional simplicity, we will not worry about the fact that these real numbers are only given up to $O(\log n)$ bits of precision (unless you want to).) We will now use the "minimum number reachable from v" rather than the "minimum rank reachable from v" to estimate R_v. (So, the only purpose of creating the ranking is for efficiently finding this minimum, using the algorithm of Problem 3.)

Here is the algorithm:

1. We give each vertex in G, independently, a random number in $[0, 1]$. We then calculate for each vertex v, the *minimum* number m_v reachable from v.
2. We repeat Step 1 for t iterations. Let's say that for a given vertex v, this results in $m_{v,1}, m_{v,2}, \ldots, m_{v,t}$ as the t minima. Let $\tilde{m}_v$ be the median element in this list. That is, half of the m_i are larger than $\tilde{m}_v$ and half of the m_i are smaller.
3. For each vertex v, output $\tilde{R}_v = 1/\tilde{m}_v$ as our estimate of the number of vertices reachable from v. (Actually, a better estimate is $1/\tilde{m}_v - 1$, but let's just say $1/\tilde{m}_v$ for simplicity.)

5. Prove there exist constants c_1, c_2 such that for any vertex v, with probability $e^{-\Omega(t)}$,

$$R_v/c_1 < \tilde{R}_v < c_2 R_v.$$

This means that after not too many iterations, with high probability *all* our estimates are within a constant factor of their true values.[1]

Hint: For the lower bound on $\tilde{R}_v$, think of each iteration of Step 1 as flipping a coin, with the event that $m_v > c_1/R_v$ as a "heads". Calculate a value of c_1 such that the probability of heads is small enough that it is unlikely we would have seen more than $t/2$ heads.

[1] If you want, you are free to modify the algorithm to prove this bound, though you should not need to. If you want to prove a stronger statement about the approximations being off by only $(1+\epsilon)$, then you may need to alter the algorithm slightly.