6.856 — Randomized Algorithms

Spring Term, 2011

Handout #2, Feb. 2, 2011 — Homework 1, Due 2/9

1. Consider the problem of using a source of unbiased random bits to generate a sample from the set $S = \{1, \ldots, n\}$ such that element i is chosen with probability p_i.

   (a) Suppose $n = 2$ (so $p_2 = 1 - p_1$). Give a scheme that uses $O(1)$ bits in expectation to choose one of the two items. **Hint:** start with an easy scheme that uses a possibly infinite number of random bits to choose the item. Then make it lazy, generating/examining only enough bits to uniquely identify the item that would be selected by the infinite sequence. What is the probability looking at a new bit lets you stop? Analyze the expected number of bits you will actually examine.

   (b) Generalize this scheme to sample from n elements using $O(\log n)$ random bits in expectation per sample, regardless of the values of the p_i.

   (c) Prove that for certain p_i and n (for example, a uniform sample from $\{1, 2, 3\}$), if you only have unbiased random bits, it is *impossible* to generate an appropriately distributed sample using a finite number of bits *in the worst case.*

2. Consider the following algorithm $\textsc{Find}(S, k)$ to find the k^{th} smallest item in set S. Pick a random element $x \in S$, and (as in quicksort) use it to partition the set into S_1, the items smaller than x, and S_2, the items larger than x. Let s be the size of S_1. If $s \geq k$, execute $\textsc{Find}(S_1, k)$; else execute $\textsc{Find}(S_2, k - s - 1)$.

   (a) Suppose S has n elements. Prove that the expected size of the set in the recursive call is bn for some constant $b < 1$.

   (b) Pick an appropriate c, and argue by induction that the expected runtime of $\textsc{Find}$ on an n-element set is at most cn.

   (c) Explain why your induction in the previous step depended on the fact that you were proving a *linear* running time.

   (d) **Optional:** Can you determine the probability that the *ith* item is compared to the j^{th}, and use that for an analysis similar to our quicksort analysis? This will be messier, since it depends on i, j, *and* k.

3. Consider the following algorithm for finding a minimum cut. Assign a random score to each edge, and compute a minimum spanning tree. Removing the heaviest edge in the tree breaks it into two pieces.

   (a) Argue that with probability $\Omega(1/n^2)$, those pieces will be the two sides of a minimum cut. **Hint:** relate this algorithm to the contraction algorithm.

   (b) Conclude that there is a simple implementation of the basic contraction algorithm taking $O(m \log n)$ time.

   (c) (optional) Refine your implementation to take $O(m)$ time.

4. MR 1.8. Consider adapting the min-cut algorithm of Section 1.1 to the problem of finding an s-t *min-cut* in an undirected graph. In this problem, we are given an undirected graph G together with two distinguished vertices s and t. An s-t min-cut is a set of edges whose removal disconnects s from t; we seek an edge set of minimum cardinality. As the algorithm proceeds, the vertex s may get amalgamated into a new vertex as the result of an edge being contracted; we call this vertex the s-vertex (initially s itself). Similarly, we have a t-vertex. As we run the contraction algorithm, we ensure that we never contract an edge between the s-vertex and the t-vertex.

   (a) Show that there are graphs (*not* multi-graphs) in which the probability that this algorithm finds an s-t min-cut is exponentially small.

   (b) How large can the number of s-t min-cuts in an instance be?

5. **This problem should be done without collaboration.** Consider the problem of finding the *second smallest cut* in a graph. This cut might equal the min-cut, if there are two min-cuts. Alternatively, this cut may be much larger than the minimum cut (can you think of an example?). Argue that nonetheless, a small modification to the randomized contraction algorithm has an $\Omega(1/n^2)$ chance of finding the second smallest cut.