Homework 1

Randomized Algorithms

Due Wednesday, August 30

1. [MR 1.8]. Consider adapting the min-cut algorithm of the first class 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 different s–t min-cut solutions in an instance be?

   (c) Can you derive a very different bound for the number of different global min-cuts, as a consequence of the algorithm presented in class?

2. Suppose we have access to a source of unbiased random bits. This problem looks at constructing biased coins or dice from this source.

   (a) Show how to construct a biased coin, which is 1 with probability p and 0 otherwise, using $O(1)$ random bits in expectation. [Hint: First show how to construct a biased coin using an arbitrary number of random bits. Then show that the expected number of bits examined is small.]

   (b) Show how to sample from $[n]$, with probabilities $p_1, \ldots, p_n$, using $O(\log n)$ random bits in expectation.

   (c) Show that the "in expectation" caveat is necessary: for example, one cannot sample uniformly over $\{1, 2, 3\}$ using $O(1)$ bits in the worst case.

   (d) [Optional.] Give a *fast* algorithm to sample from $[n]$ with probabilities $p_1, \ldots, p_n$. That is, give an algorithm that uses in expectation $O(\log n)$ bits and $O(1)$ time per sample (in the word RAM model, so manipulating/indexing with $O(\log n)$-bit words takes $O(1)$ time.). Your algorithm may preprocess the input, using $O(n)$ time and space. [Hints: (a) if all the p_i came in pairs that summed to $2/n$, could you solve the problem? (b) can you break up any set of p_i into $2n$ total pieces, so the pieces come in pairs that sum to $1/n$?]