6332 - Advanced algorithms, Spring 2015, CSE, OSU
Homework 2
Instructor: Anastasios Sidiropoulos
Due date: Feb 27, 2014

Problem 1 (Average case analysis vs. expected running time). Let $\mathcal{A}$ be a deterministic integer sorting algorithm. Suppose that the average-case complexity of $\mathcal{A}$ is $T(n)$ when the input consists of a set of integers I, with $|I| = n$, and we assume that the ordering of the input is chosen uniformly at random from the set of all possible such orderings. Notice that there can be some orderings of the input that cause the algorithm be run very slowly. Using this fact, show how to obtain a randomized algorithm for integer sorting, that has expected running time $O(n) + T(n)$, when the input consists of the same set of integers I ordered in an arbitrary way.

Problem 2 (Derandomization). Let us say that a problem Π is a *decision problem* if for any input x to Π, the goal is to output either YES or NO (for example, 2-SAT is such a problem). In other words, the goal is to compute a function $f : I \to \{\text{YES}, \text{NO}\}$, where I denotes the set of all possible inputs to Π.

(a) Suppose that you are given a Las Vegas algorithm $\mathcal{A}$ for Π. Given an input $x \in I$, the algorithm $\mathcal{A}$ runs in expected polynomial time and outputs $\mathcal{A}(x) = f(x)$. Suppose further that $\mathcal{A}$ uses only $O(\log n)$ random bits, where n denotes the length of the description of x. Show how to construct a deterministic polynomial-time algorithm for Π.

(b) Suppose that you are given a Monte Carlo algorithm $\mathcal{A}$ for Π. Given an input $x \in I$, the algorithm $\mathcal{A}$ runs in polynomial time and outputs some answer $\mathcal{A}(x)$ such that

$$\Pr[\mathcal{A}(x) = f(x)] > 2/3.$$

Note that the answer $\mathcal{A}(x)$ is a random variable and its value depends on the values of the random bits that are used by the algorithm. Suppose further that $\mathcal{A}$ uses only $O(\log n)$ random bits, where n denotes the length of the description of x. Show how to construct a deterministic polynomial-time algorithm for Π.

Problem 3 (Karger's algorithm for s-t min-cuts?). Recall that Karger's algorithm succeeds in finding a minimum cut in an n-vertex graph with probability at least $1/n^c$, for some constant $c > 0$. This implies that by running the algorithm polynomially many times, and taking the minimum of all cuts found, we obtain a polynomial-time algorithm that succeeds in finding the minimum cut with high probability.

A natural question is whether Karger's algorithm can be used to compute a minimum s-t cut. Recall that the input to minimum s-t cut problem is a graph $G = (V, E)$ and two vertices $s, t \in V$. The goal is to find some $S \subset V$, with $s \in V$, $t \notin V$, such that the number of edges with exactly one endpoint in S is minimized.

Let us consider the following modification of Karger's algorithm for this problem: While $|V| > 2$, we pick uniformly at random an edge e *that does not have as endpoints both s and t*. That is, we ignore all parallel edges between s and t, if any such edges exist. We contract e and repeat.

Does this algorithm find a minimum s-t cut with probability at least $1/n^{c'}$, for some constant $c' > 0$? Can one find a minimum s-t cut with high probability, by repeating this algorithm polynomially many times and taking the minimum of all cuts found?

Problem 4 (An implication of Karger's algorithm). Show that any n-vertex graph has a most n^c different minimum cuts, for some constant $c > 0$. Hint: Use the analysis of Karger's algorithm.

Problem 5 (Directed s-t connectivity). Recall that an input to the undirected s-t connectivity problem consists of an undirected graph $G = (V, E)$ and two vertices $s, t \in V$. The goal is to decide whether there exists a path between s and t in G. We saw in class that one can obtain an algorithm that correctly solves this problem with high probability by repeating polynomially many random walks starting from s. This algorithm can be implemented using $O(\log n)$ bits of space (that is, the algorithm only needs to remember $O(\log n)$ bits of information).

In the *directed* s-t connectivity problem the input graph G is directed and the goal is to decide whether the exist a directed path from s to t. Suppose that we try to find a directed path from s to t again via a random walk. How much space would such an algorithm require in order to guarantee that the answer is correct with high probability?

Note that we are only interested in bounding the number of bits of memory used by the algorithm and we completely ignore its running time.