Final Exam for CS 25

Prof. Drysdale

December 1, 2003

Ground Rules

1. This exam is due at noon on Monday, December 8, 2003.
2. You may not discuss this exam with anyone except Prof. Drysdale until after you have turned it in.
3. The only written materials you may consult are your copy of the course textbook *Introduction to Algorithms*, your lecture notes, your homeworks, and the solutions distributed this term.
4. In all questions that ask you to give an algorithm, you must argue that your algorithm is correct and analyze its running time.
5. You may buy a hint for any problem. (If you are stuck it is probably better to buy a hint.) Tell Professor Drysdale what you have figured out about the problem. He will offer one or more hints and tell you how much (in points) each hint will cost. You then decide which hint (if any) to buy. You will be given the hint and the points will be deducted from your eventual total to the problem (but you cannot go negative for the problem). You will not be charged for a hint unless you agree to buy it.
6. Use the standard directions for turning in homework (one problem per sheet, etc.)
7. Any changes or clarifications will be sent via email, so check regularly.

1. **[15 points]**
   A *mixed* graph is graph in which some of the edges are directed and some of the edges are undirected. If a given mixed graph G has no directed cycle, then it is always possible to orient the remaining undirected edges so that the resulting graph has no directed cycle. Give an efficient algorithm for obtaining such an orientation if one exists. Prove the correctness of your algorithm and analyze its complexity.

2. **[25 points]**
   Given n vertices $v_1, \ldots, v_n$, a random graph is formed as follows: for each pair of vertices v_i, v_j with $i \neq j$ we flip a fair coin, and include directed edge (v_i, v_j) in the graph if and only if the coin comes up heads. Two labeled graphs are *identical* if they have the same set of edges. Assume that the graphs are represented as adjacency matrices.

   (a) What is the probability that two random graphs G_1 and G_2 formed on the same n vertices $v_1, \ldots, v_n$ are identical? Justify your answer.

   (b) Given two random graphs G_1 and G_2 formed on the same n vertices $v_1, \ldots, v_n$, we wish to determine if the two graphs are identical. Give an algorithm with the fastest expected running time that you can that solves this problem. What are the expected and worst case running times of your algorithm?

   (c) Given m random graphs $G_1, \ldots G_m$ formed on the same n vertices $v_1, \ldots, v_n$, we wish to determine all pairs of identical graphs. Give an algorithm with the fastest expected running time that you can that solves this problem. What are the expected and worst case running times of your algorithm? *Hint:* You can do better than trying all pairs.

3. **[20 points]**
   An *independent set* of a graph G is a set of vertices, no two of which are adjacent. Finding an independent set is NP-complete, so we are unlikely to be able find independent sets quickly. Therefore we consider an approximation algorithm for a special class of graphs.

   (a) Suppose the undirected graph G is regular of degree d. That is, every vertex of G has exactly d vertices adjacent to it. For any fixed d, give an algorithm to find a independent set of size $c|V|$ for as large a value of the constant c as you can. Express c as a function of d.

   (b) For what ρ is your algorithm a ρ-approximation algorithm? To solve this come up with a good estimate of the maximum-sized independent set possible in a regular graph of degree d. Prove that your estimate is tight.

4. **[20 points]**
   You are given a weighted directed graph $G = (V, E)$, with a source s and edge weights $w(u, v)$ for each edge $(u, v) \in E$. Furthermore, you know that the *only* edges with negative weights leave the source vertex s. Give the most efficient algorithm that you can to compute the single-source shortest paths from s to all other vertices (or report that G has a negative cycle, if one exists). Why does your algorithm work, and what is its run time?

5. **[15 points]**
   A graph G has edges which are colored either red or blue. Give the fastest algorithm that you can to compute a spanning tree with as few blue edges as possible. What is the run time of your algorithm?

6. **[20 points]**
   You are given a directed network $G = (V, E)$ in which each edge (u, v) has a positive integral capacity $c(u, v)$. G has a source s and a sink t. An edge is called *sensitive* if it crosses *some* minimum cut (S, T) of the network, going from S to T.

   (a) Show that there is a graph with an exponential number of different minimum cuts.

   (b) Give as efficient a polynomial time algorithm as you can to find all sensitive edges. Prove its correctness and analyze its runtime.

7. **[25 points]**
   You are to design a data structure that supports the following operations:

   CreateSet($x_1, x_2, \ldots, x_k$) - Creates a k-element set containing $x_1, x_2, \ldots, x_k$. It returns a reference to the set.

   DeleteSet(r) - Removes the set that r refers to.

   Union(r_1, r_2) - Forms the union of the two sets referred to by r_1 and r_2 and returns a reference to the merged set.

   FindLargest() - Returns a reference to one of the sets that has the largest number of elements. Thus if the sets are $\{a, b\}$, $\{c, d, e\}$, and $\{x, y, z\}$ it should return a reference to either $\{c, d, e\}$ or $\{x, y, z\}$.

   The amortized time for CreateSet should be $O(k)$ and the amortized time for all other operations should be $O(1)$.

   Your structure should include an array of references, one for each possible set size, and you should bucket the sets by size.

   (a) Assume that no set will ever contain more than M items. Describe your data structure in detail and explain how to perform each operation. (Note that you do not have time to initialize the array of M elements. You can allocate it in constant time, but cannot initalize it.)

   (b) Justify that the amortized running time of each operation is as stated above. (Don't worry about time taken to free up memory.)

   (c) If you had no upper limit M how could you change things so that you could still achieve the same amortized running times? Outline the difficulty and how you would get around it.