Introduction to Algorithms	**Problem Set 10**
CS 482 Spring 2004	**Due May 7, 2004**

Please hand in each problem on a separate sheet with your name and NetID on each.

Reading: Sections 11.1-11.5.

(1) *(This is essentially Problem 8 at the end of Chapter 11, but expressed in terms of the 3-Coloring problem.) 3-Coloring* is a yes/no question, but we can phrase it as an optimization problem as follows.

Suppose we are given a graph $G = (V, E)$, and we want to color each node with one of three colors, even if we aren't necessarily able to give different colors to every pair of adjacent nodes. Rather, we say that an edge (u, v) is *satisfied* if the colors assigned to u and v are different.

Consider a 3-coloring that maximizes the number of satisfied edges, and let c^* denote this number. Give a polynomial-time algorithm that produces a 3-coloring that satisfies at least $\frac{2}{3}c^*$ constraints. If you want, your algorithm can be randomized; in this case, the *expected* number of constraints it satisfies should be at least $\frac{2}{3}c^*$.

(2) *(This is an expanded version of Problem 6 at the end of Chapter 11.)* A number of *peer-to-peer systems* on the Internet are based on *overlay networks*: rather than using the physical Internet topology as the network on which to perform computation, these systems run protocols by which nodes choose collections of virtual "neighbors" so as to define a higher-level graph whose structure may bear little or no relation to the undelying physical network. Such an overlay network is then used for sharing data and services, and it can be extremely flexible compared with a physical network, which is hard to modify in real-time to adapt to changing conditions.

Peer-to-peer networks tend to grow through the arrival of new participants, who join by linking into the existing structure. This growth process has an intrinsic effect on the characteristics of the overall network. Recently, people have investigated simple abstract models for network growth that might provide insight into the way such processes behave, at a qualitative level, in real networks.

Here's a simple example of such a model. The system begins with a single node v_1. Nodes then join one at a time; as each node joins, it executes in a protocol whereby it forms a directed link to a single other node chosen uniformly at random from those already in the system. More concretely, if the system already contains nodes $v_1, v_2, \ldots, v_{k-1}$ and node v_k wishes to join, it randomly selects one of $v_1, v_2, \ldots, v_{k-1}$ and links to this node.

Suppose we run this process until we have a system consisting of nodes $v_1, v_2, \ldots, v_n$; the random process described above will produce a directed network in which each node other than v_1 has exactly one out-going edge. On the other hand, a node may have multiple in-coming links, or none at all. The in-coming links to a node v_j reflect all the other nodes whose access into the system is via v_j; so if v_j has many in-coming links, this can place a large load on it. To keep the system load-balanced, then, we'd like all nodes to have a

roughly comparable number of in-coming links, but that's unlike to happen here, since nodes that join earlier in the process are likely to have more in-coming links than nodes that join later. Let's try to quantify this imbalance as follows.

(a) Given the random process described above, what is the expected number of in-coming links to node v_j in the resulting network? Give an exact formula in terms of n and j, and also try to express this quantity asymptotically (via an expression without large summations) using $\Theta(\cdot)$ notation.

(b) Part (a) makes precise a sense in which the nodes that arrive early carry an "unfair" share of the connections in the network. Another way to quantify the imbalance is to observe that, in a run of this random process, we expect many nodes to end up with no in-coming links.

Give a formula for the expected number of nodes with no in-coming links in a network grown randomly according to this model.

(3) *(This is Problem 2 at the end of Chapter 11.)* In class, we designed an approximation algorithm to within a factor of 7/8 for the *MAX 3-SAT* problem, where we assumed that each clause has terms associated with 3 different variables. In this problem we will consider the analogous *MAX SAT* problem: given a set of clauses $C_1, \ldots, C_k$ over a set of variables $X = \{x_1, \ldots, x_n\}$, find a truth assignment satisfying as many of the clauses as possible. Each clause has at least one term in it, but otherwise we do not make any assumptions on the length of the clauses: there may be clauses that have a lot of variables, and others may have just a single variable.

(a) First consider the randomized approximation algorithm we used for *MAX 3-SAT*, setting each variable independently to *true* or *false* with probability 1/2 each. Show that the expected number of clauses satisfied by this random assignment is at least $k/2$, i.e., half of the clauses is satisfied in expectation. Give an example to show that there are *MAX SAT* instances such that no assignment satisfies more than half of the clauses.

(b) If we have a clause that consists just of a single term (e.g. a clause consisting just of x_1, or just of $\overline{x_2}$), then there is only a single way to satisfy it: we need to set the corresponding variable in the appropriate way. If we have two clauses such that one consists of just the term x_i, and the other consists of just the negated term $\overline{x_i}$, then this is a pretty direct contradiction.

Assume that our instance has no such pair of "conflicting clauses"; that is, for no variable x_i do we have both a clause $C = \{x_i\}$ and a clause $C' = \{\overline{x_i}\}$. Modify the above randomized procedure to improve the approximation factor from 1/2 to at least a .6 approximation, that is, change the algorithm so that the expected number of clauses satisfied by the process is at least $.6k$.

(c) Give a randomized polynomial time algorithm for the general *MAX SAT* problem, so that the expected number of clauses satisfied by the algorithm is at least a .6 fraction of the maximum possible.

(Note that by the example in part (a), there are instances where one cannot satisfy more than $k/2$ clauses; the point here is that we'd still like an efficient algorithm that, in expectation, can satisfy a .6 fraction *of the maximum that can be satisfied by an optimal assignment.*)