CSE 521: Design and Analysis of Algorithms
Assignment #3
April 19, 2002
Due: Friday, April 26

Reading Assignment: Kleinberg and Tardos, Chapters 5 and 6

Problems:

1. Suppose it's nearing the end of the quarter and you're taking n courses, each with a final project that still has to be done. Each project will be graded on the following scale: it will be assigned an integer number on a scale of 1 to $g > 1$, higher numbers being better grades. Your goal, of course, is to maximize your average grade on the n projects.

   Now, you have a total of $H > n$ hours in which to work on the n projects cumulatively, and you want to decide how to divide up this time. For simplicity, assume H is a positive integer, and you'll spend an integer number of hours on each project. So as to figure out how best to divide up your time, you've come up with a set of functions $\{f_i : i = 1, 2, \ldots, n\}$ (rough estimates, of course) for each of your n courses; if you spend $h \leq H$ hours on the project for course i, you'll get a grade of $f_i(h)$. (You may assume that the functions f_i are *non-decreasing*: if $h < h'$ then $f_i(h) \leq f_i(h')$.)

   So the problem is: given these functions $\{f_i\}$, decide how many hours to spend on each project (in integer values only) so that your average grade, as computed according to the f_i, is as large as possible. In order to be efficient, the running time of your algorithm should be polynomial in n, g, and H; none of these quantities should appear as an exponent in your running time.

2. Consider the Bellman-Ford minimum-cost path algorithm from the text, assuming that the graph has no negative cost cycles. This algorithm is both fairly slow and also memory-intensive. In many applications of dynamic programming, the large memory requirements can become a bigger problem than the running time. The goal of this problem is to decrease the memory requirement. The pseudo-code Shortest-Path(G, s, t) in the text maintains an array $M[0...n-1; V]$ of size n^2, where $n = |V|$ is the number of nodes on the graph.

   Notice that the values of $M[i, v]$ are computed only using $M[i-1, w]$ for some nodes $w \in V$. This suggests he following idea: can we decrease the memory needs of the algorithm to $O(n)$ by maintaining only two columns of the M matrix at any time? Thus we will "collapse" the array M to an $2 \times n$ array B: as the algorithm iterates through values of i, $B[0, v]$ will hold the "previous" column's value $M[i-1, v]$, and $B[1, v]$ will hold the "current" column's value $M[i, v]$.

```
Space-Efficient-Shortest-Path(G, s, t)
```

```
n = number of nodes in G
Array B[0...1, V]
For v ∈ V in any order
  B[0, v] = ∞
Endfor
B[0, s] = 0
For i = 1, ..., n − 1
  For v ∈ V in any order
    M = B[0, v]
    M' = min_{w∈V:(w,v)∈E} (B[0, w] + c_wv)
    B[1, v] = min(M, M')
  Endfor
  For v ∈ V in any order
    B[0, v] = B[1, v]
  Endfor
Endfor
Return B[1, t]
```

It is easy to verify that when this algorithm completes, the array entry $B[1, v]$ holds the value of $OPT(n-1, v)$, the minimum-cost of a path from s to v using at most $n-1$ edges, for all $v \in V$. Moreover, it uses $O(n^3)$ time and only $O(n)$ space. You do not need to prove these facts.

The problem is: where is the shortest path? The usual way to find the path involves tracing back through the $M[i, v]$ values, using the whole matrix M, and we no longer have that. The goal of this problem is to show that if the graph has no negative cycles, then there is enough information saved in the last column of the matrix M, to recover the shortest path in $O(n^2)$ time.

Assume G has no negative or even zero length cycles. Give an algorithm Find-Path(t, G, B) that uses only the array B (and the graph G) to find the the minimum-cost path from s to t in $O(n^2)$ time.

3. As most of us don't know, there are many sunny days in Ithaca, NY; but this year, as it happens, the spring ROTC picnic at Cornell has fallen on rainy day. The ranking officer decides to postpone the picnic, and must notify everyone by phone. Here is the mechanism she uses to do this.

   Each ROTC person on campus except the ranking officer reports to a unique *superior officer*. Thus, the reporting hierarchy can be described by a tree T, rooted at the ranking officer, in which each other node v has as a parent node u equal to his or her superior officer. Conversely, we will call v a *direct subordinate* of u. See Figure 1, in which A is the ranking officer, B and D are the direct subordinates of A, and C is the direct subordinate of B.

   To notify everyone of the postponement, the ranking officer first calls each of her direct subordinates, one at a time. As soon as each subordinate gets the phone call, he or she must notify each of his or her direct subordinates one at a time. The process continues this way, until everyone has been notified. Note that each person in this process can only call direct subordinates on the phone; for example, in Figure 1, A would not be allowed to call C.

Now, we can picture this process as being divided into *rounds*: In one *round*, each person who has already learned of the postponement can call one of his or her direct subordinates on the phone. The number of rounds it takes for everyone to be notified depends on the sequence in which each person calls their direct subordinates. For example, in Figure 1, it will take only two rounds if A starts by calling B, but it will take three rounds if A starts by calling D.

Give an efficient algorithm that determines the minimum number of rounds needed for everyone to be notified, and outputs a sequence of phone calls that achieves this minimum number of rounds.

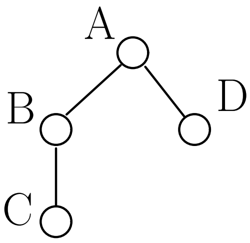

Figure 1: A hierarchy with four people. The fastest broadcast scheme is for A to call B in the first round. In the second round, A calls D and B calls C. If A were to call D first, then C could not learn the news until the third round.

4. Let $G = (V, E)$ be a graph with n nodes in which each pair of nodes is joined by an edge. There is a positive weight w_{ij} on each edge (i, j); and we will assume these weights satisfy the *triangle inequality* $w_{ik} \leq w_{ij} + w_{jk}$. For a subset $V' \subseteq V$, we will use $G[V']$ to denote the subgraph (with edge weights) induced on the nodes in V'.

   We are given a set $X \subseteq V$ of k *terminals* that must be connected by edges. We say that a *Steiner tree* on X is a set Z so that $X \subseteq Z \subseteq V$, together with a sub-tree T of $G[Z]$. The *weight* of the Steiner tree is the weight of the tree T.

   Show that there is function $f(\cdot)$ and a *polynomial function* $p(\cdot)$ so that the problem of finding a minimum-weight Steiner tree on X can be solved in time $O(f(k) \cdot p(n))$. The function $f(\cdot)$ can be exponential in k.

   Hint: Define Y_i to be $X \cup i$ for every vertex i such that i is not in X. Consider the problem of finding a minimum-weight Steiner tree on every subset of the Y_i's (for each of the $n - k$ Y_i's).

5. **Extra Credit:** Give feedback on chapters 5 and 6 of book. Send this by email to Anna and Gideon.