CSE 521: Design and Analysis of Algorithms | Winter 2006

Problem Set #1 | Instructor: Venkatesan Guruswami

Due on **January 19, 2006** in class.

Reminder: If you haven't done so already, subscribe to CSE 521 email group ASAP by following the link from the course webpage `http://www.cs.washington.edu/521`.

Instructions: You are allowed to collaborate with fellow students taking the class in solving problem sets, but you must write up your solutions entirely on your own. If you do collaborate in solving problems, you must acknowledge for each problem the people you worked with on that problem.

The problems have been carefully chosen for their pedagogical value and hence might be similar or identical to those given out in past offerings of this course at UW, or similar courses at other schools. Using any pre-existing solutions from these sources, from the Web or other algorithms textbooks constitutes a violation of the academic integrity expected of you and is strictly prohibited.

Most of the problems require only one or two key ideas for their solution – spelling out these ideas should give you most of the credit for the problem even if you err in some finer details. So, make sure you clearly write down the main idea(s) behind your solution even if you could not figure out a complete solution.

A final piece of advice: Begin work on the problem set early and don't wait till the deadline is only a few days away.

Readings: Kleinberg and Tardos: Chapter 1; Chapter 4 (till 4.5); Chapter 5 (till 5.4).

Problems: Each problem is worth 10 points unless noted otherwise. All problem numbers refer to the Kleinberg-Tardos textbook.

1. Chapter 1, Problem 4 (Stable matching between hospitals and residents)

   Gale and Shapley published their paper on the stable marriage problem in 1962; but a version of their algorithm had already been in use for ten years by the National Resident Matching Program, for the problem of assigning medical residents to hospitals.

   Basically, the situation was the following. There were m hospitals, each with a certain number of available positions for hiring residents. There were n medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than there were slots available in the m hospitals.

   The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.)

   We say that an assignment of students to hospitals is *stable* if neither of the following situations arises.

   - First type of instability: There are students s and s', and a hospital h, so that

  - s is assigned to h, and
  - s' is assigned to no hospital, and
  - h prefers s' to s.
- Second type of instability: There are students s and s', and hospitals h and h', so that
  - s is assigned to h, and
  - s' is assigned to h', and
  - h prefers s' to s, and
  - s' prefers h to h'.

So we basically have the stable marriage problem from the section, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students.

Show that there is always a stable assignment of students to hospitals, and give an efficient algorithm to find one.

2. Chapter 1, Problem 8 (Truthfulness in Gale-Shapley algorithm)
   Suggestion: Try to work out some small examples to get started.

   For this problem, we will explore the issue of *truthfulness* in the Stable Matching Problem and specifically in the Gale-Shapley algorithm. The basic question is: Can a man or a woman end up better off by lying about his or her preferences? More precisely, we suppose each participant has a true preference order. Now consider a woman w. Suppose w prefers m to m', but both m and m' are low on her list of preferences. Can it be the case that by switching the order of m and m' on her list of preferences (i.e., by falsely claiming that she prefers m' to m) and running the algorithm with this false preference list, w will end up with a man m'' that she truly prefers to both m and m'? (We can ask the same question for men, but will focus on the case of women for purposes of this question.)

   Resolve this question by doing one of the following two things:

   (a) Give a proof that, for any set of preference lists, switching the order of a pair on the list cannot improve a woman's partner in the Gale-Shapley algorithm; or

   (b) Give an example of a set of preference lists for which there is a switch that would improve the partner of a woman who switched preferences.

3. Consider the problem of making change for n cents using the least number of coins.

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins are in the denominations $b^0, b^1, \ldots, b^k$ for some integers $b > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

   (c) Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution.

4. Chapter 4, Problem 24 (Zero skew trees)

   Timing circuits are a crucial component of VLSI chips. Here's a simple model of such a timing circuit. Consider a complete balanced binary tree with n leaves, where n is a power of two.

Each edge e of the tree has an associated length l_e, which is a positive number. The distance from the root to a given leaf is the sum of the lengths of all the edges on the path from the root to the leaf.

The root generates a *clock signal* which is propagated along the edges to the leaves. We will assume that the time it takes for the signal to reach a given leaf is proportional to the distance from the root to the leaf.

Now, if all leaves do not have the same distance from the root, then the signal will not reach the leaves at the same time, and this is a big problem. We want the leaves to be completely synchronized, and all to receive the signal at the same time. To make this happen, we will have to *increase* the lengths of certain edges, so that all root-to-leaf paths have the same length (we are not able to shrink edge lengths). If we achieve this, then the tree (with its new edge lengths) will be said to have *zero skew*. Our goal is to achieve zero skew in a way that keeps the sum of all the edge lengths as small as possible.

Given an algorithm that increases the lengths of certain edges so that the resulting tree has zero skew and the total edge length is as small as possible.

(Example omitted.)

5. Chapter 4, Problem 26 (MST with time varying edge costs)

   One of the first things you learn in calculus is how to minimize a differentiable function such as $y = ax^2 + bx + c$, where $a > 0$. The Minimum Spanning Tree Problem, on the other hand, is a minimization problem of a very different flavor: There are now just a finite number of possibilities for how the minimum might be achieved - rather than a continuum of possibilities - and we are interested in how to perform the computation without having to exhaust this (huge) finite number of possibilities.

   One can ask what happens when these two minimization issues are brought together, and the following question is an example of this. Suppose we have connected graph $G = (V, E)$. Each edge e now has a *time-varying edge cost* given by a function $f_e : R \rightarrow R$. Thus, at time t, it has cost $f_e(t)$. We will assume that all these functions are positive over their entire range. Observe that the set of edges constituting the minimum spanning tree of G may change over time. Also, of course, the cost of the minimum spanning tree of G becomes a function of the time t; we will denote this function $c_G(t)$. A natural problem then becomes: Find a value of t at which $c_G(t)$ is minimized.

   Suppose each function f_e is a polynomial of degree 2: $f_e(t) = a_e t^2 + b_e t + c_e$, where $a_e > 0$. Give an algorithm that takes the graph G and the values $\{(a_e, b_e, c_e) : e \in E\}$ and returns a value of the time t at which the minimum spanning tree has minimum cost. Your algorithm should run in time polynomial in the number of nodes and edges of the graph G. You may assume that arithmetic operations on the numbers $\{(a_e, b_e, c_e)\}$ can be done in constant time per operation.

6. Chapter 4, Problem 29 (Feasibility of degree sequence)
   Hint: Consider the vertices adjacent to the lowest degree vertex.

   Given a list of n natural numbers $d_1, d_2, \ldots, d_n$, show how to decide in polynomial time whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, \ldots, d_n$ (That is, if $V = \{v_1, v_2, \ldots, v_n\}$, then the degree of v_i should be exactly

d_i.) G should not contain multiple edges between the same pair of nodes, or "loop" edges with both endpoints equal to the same node.

7. Chapter 5, Problem 5 (Hidden surface removal)

   Hidden surface removal is a problem in computer graphics that scarcely needs an introduction — when Woody is standing in front of Buzz you should be able to see Woody but not Buzz; when Buzz is standing in front of Woody, ...well, you get the idea.

   The magic of hidden surface removal is that you can often compute things faster than your intuition suggests. Here's a clean geometric example to illustrate a basic speed-up that can be achieved. You are given n non-vertical lines in the plane, labeled $L_1, \ldots, L_n$, with the i-th line specified by the equation $y = a_i x + b_i$. We will make the assumption that no three of the lines all meet at a single point. We say line L_i is *uppermost* at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0: $a_i x_0 + b_i > a_j x_0 + b_j$ for all $j \neq i$. We say line L_i is *visible* if there is some x-coordinate at which it is uppermost – intuitively, some portion of it can be seen if you look down from "$y = \infty$".

   Give an algorithm that takes n lines as input, and in $O(n \log n)$ time returns all of the ones that are visible.

   (Example omitted.)