CSE 521: Design and Analysis of Algorithms — Autumn 2006

Take Home Midterm — Instructor: Anna Karlin

Due on **November 16, 2006** in class.

Instructions: This is a take home exam with the following rules and instructions:

- The work you turn in should be entirely your own. You are not allowed to collaborate or discuss the problems, solutions, or any aspect relating to this exam with your classmates, or anyone else. If you have some difficulty following any of the questions or think something is ambiguous, send an email to the course instructor/TA.
- You are ONLY permitted to refer to the Kleinberg-Tardos text, your class notes and solutions to previous problem sets. Reference to any other source will be considered an act of academic dishonesty.
- On all the problems, give the running time of your algorithm. Also, be sure to prove that your algorithm is correct.
- Devote sufficient time for writing your solutions in a clear manner. We are looking for correctness, precision and clarity of exposition.
- If for some reason you cannot make it to class on Thursday, Nov 16, make **prior** arrangements with us to turn in your exam before class that day.

Questions:

1. (20 points) Alice wants to throw a party and is deciding who to call. She has n people to choose from, and she has made up a list of which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints: at the party, each person should have at least five other people that they know *and* five other people that they don't know.

   Give an efficient algorithm that takes as input the list of n people and the list of pairs who know each other and outputs the best choice of party invitees.

2. (30 points) An array $A[1..n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However, you *can* answer questions of the form: "is $A[i] = A[j]$?" in constant time.

   (a) (15 points) Show how to solve this problem in $O(n \log n)$ time. (*Hint:* Split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A?. If so, you can use a divide-and-conquer approach.)

   (b) (15 points) Give a linear time algorithm. (*Hint:* Try a divide-and-conquer approach based on pairing up the elements and using that pairing to eliminate elements.)

3. (25 points) Consider the following game: A “dealer” produces a sequence $s_1, \ldots, s_n$ of “cards”, face up, where each card s_i has value v_i. Then two players take turns picking a card from the sequence, but can only pick the first or the last card of the (remaining) sequence. The goal is to collect cards of largest total value. (For example, you can think of the cards as bills of different denominations.) Assume n is even.

   (a) (5 points) Show a sequence of cards such that it is not optimal for the first player to start by picking up the available card of larger value. That is, the natural *greedy* strategy is suboptimal.

   (b) (20 points) Give an $O(n^2)$ algorithm to compute an optimal strategy for the first player. Given the initial sequence, your algorithm should precompute in $O(n^2)$ time some information, and then the first player should be able to make each move optimally in $O(1)$ time by looking up the precomputed information.

4. (25 points) A variation on change-making:

   Given an unlimited supply of coins of denominations $x_1, x_2, \ldots, x_n$, we wish to make change for a value v using at most k coins. (All the x_i's are integers and v is an integer as well.) That is, we wish to find a set of at most k coins whose total value is v. This might not be possible: for instance, if the denominations are 5 and 10 and $k = 6$, then we can make change for 55 but not for 65. Give an efficient dynamic programming algorithm (as a function of n, k and v) for the following problem:

   Given $x_1, \ldots, x_n$, k and v: is it possible to make change for v using at most k coins, of denominations $x_1, \ldots, x_n$?

   What is the running time of your algorithm?