CSE 331: Introduction to Algorithm Analysis and Design Fall 2012

HOMEWORK 4

Due Friday, October 5, 2012 by 1:15pm in class

IMPORTANT: Please submit each problem separately, i.e. each problem should begin on a new page and only the pages for one problem should be stapled together. Failure to do so might result in some problem(s) not being graded.

For general homework policies and our suggestions, please see the policy document.

Section 2.5 of the book *might* be useful for this HW.

---

1. (40 points) This problem is just to get you thinking about graphs and get more practice with proofs.

   A *forest* with c *components* is a graph that is the union of c disjoint trees. Note that a tree is a forest with 1 component. Prove that an n-vertex forest with c components has $n - c$ edges. (Recall we have proved the statement above for $c = 1$ in class.)

2. ($5 + 40 = 45$ points) There is a group of p Brad Pitt fans (or BPFs for short) who have booked an entire movie theater for a screening of Moneyball. Out of the $p(p-1)/2$ possible pairs, f pairs of BPFs are friends. The movie theater is called the Square Theater and has p rows with each row having exactly p chairs in them. (If it helps, you can assume that the p^2 chairs are arranged in a "grid"/"matrix" form where each of the p rows has p chairs each and each of the p columns has p chairs each.)

   The BPF organization committee has come to you to help them efficiently solve the following seating problem for them. A *seating* (as you might expect) is an assignment of the p BPFs to the p^2 chairs in the theater so that each BPF gets his/her own chair. A seating is called *admissible* if (1) for every pair of friends (b_1, b_2), they can *talk* to each other during the movies (2) Some BPF is assigned a seat in the first row and (3) if the seating satisfies the *distance compatible* property.

   The BPFs have been to the Square theater before so they have figured out that two people can *talk* to each other if and only if they are either (i) seated in the same row or (ii) are seated in the rows next to them (i.e. either to the row directly in first or the row directly behind. The first and the last row of course only have one row next to them).

   A seating has the *distance compatible* property if and only if the following holds. For any b_1 who is assigned a seat in the first row and any BPF b_2, such that b_1 and b_2 have a *friendship distance* of d (assuming d is defined) are seated d or more rows apart. (The friendship distance is the natural definition. Let a *friendship path* between b_1 and b_2 be the sequence of BPFs $f_0 = b_1, f_1, \ldots, f_{k-1}, f_k = b_2$ (for some $k \geq 0$) such that for every $0 \leq i \leq k-1$, (f_i, f_{i+1}) are friends– the length of such a path is k. The friendship distance between b_1 and b_2 is the shortest length of any friendship path between them. So if b_1 is b_2's friend, they have a friendship distance of 1 and if b_2 is a friend of a friend, they have a distance of 2 and so on. The friendship distance is not defined if there is no friendship path between b_1 and b_2.)

   Your final task is to design an efficient algorithm that computes an admissible seating, given as input the set of p BPFs and the set of f pairs of BPFs who are friends.

(a) Formalize the problem above in terms of graphs: i.e. write down (i) How you would represent the input as a graph G; (ii) How you would define a seating and (iii) Define what it means for a seating to be admissible in terms of properties of graph G.

(*Note:* This should not be too tricky. Also feel free to skip this part if you want to answer part (b) directly.)

(b) Using part (a) or otherwise design an efficient algorithm to compute an admissible seating. (Recall that your algorithm should work for any set of p people and any possible set of f friendships.) You should prove the correctness of your algorithm. You also need justify the running time of your algorithm.

3. (2 + 13 = 15 points) Consider the following problem where the input are n numbers $a, \ldots, a_n$ and an integer $1 \leq k \leq n$. The goal is to output the k largest numbers in $a_1, \ldots, a_n$. In this problem you need to do the following:

   - Design an algorithm to solve the above problem in time $O(nk)$.
   - Now design an algorithm to solve the above problem in time $O(n \log k)$.

   (*Note:* If you correctly solve the second part then you do not need to solve the first part.)