15-451 Algorithms, Fall 2000

Homework # 5 **due: November 7, 2000**

Groundrules: You should try to do the assignment as much as possible by yourself. If you get stuck you may ask for help from friends or TAs or the instructors. In any event, you should do your writeup by yourself, in your own words. If you did get help from others, please list their names at the top of the first page of your assignment. Your assignment is due at the beginning of class on the due date.

1. Fair carpooling (20 points)

There are n people, $p_1, p_2, \ldots, p_n$. Every day, some subset of these people carpool together. Let S_1 be the set that carpool together on day 1. There are m such sets, $S_1, \ldots, S_m$. For each set, one of the people in the set must be chosen to be the driver that day. Driving is not desirable, so the people want the work of driving to be divided as equitably as possible. Your task in this problem is to give an algorithm to do this efficiently and fairly.

The fairness criterion is the following: A person p is in some of the sets. Say the sizes of the sets that she's in are $a_1, a_2, \ldots, a_k$. Person p should really have to drive $\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_k}$ times, because this is the amount of resource that this person effectively uses. Of course this number may not be an integer, so let's round it up to an integer. The fairness criterion is simply that she should drive no more times that this many times.

For example, say that on day 1, Alice and Bob carpool together, and on day 2, Alice and Carl carpool together. Alice's fair cost would be $1/2 + 1/2 = 1$. So Alice driving both days would not be fair. Any solution except that one is fair.

(a) Prove that there always exists a fair solution.

(b) Give a polynomial-time algorithm for computing a fair solution.

Hint: One approach to the problem is to model it using network flow. Part (a) then follows from the integrality theorem for network flow, and part (b) just follows from the fact that we can solve max flow in polynomial time. Your job is to come up with the right flow graph to model the problem.

2. Pseudo Path Compression (20 points)

The find algorithm in the union/find problem (using trees with path compression) has to make two passes along the path from the starting node to the root: One pass to find the root, and another pass to change all the pointers of these nodes to point to the root. (In the recursive code fragment given in class, this was done implicitly, but it is still two passes when you unravel it.) A clever programmer has the brilliant idea of pointing all the nodes on this path to the STARTING node (the node on which find is being applied). This way, only one pass is needed. If you do it this way, the canonical element of the set changes, but let's assume this is not critical in this application.

Prove that this algorithm could take $\Omega(\log(n))$ per find operation.

Specifically, show how to construct a tree with n nodes, such that for some node x in this tree that is at $\Omega(\log(n))$ depth, doing find(x) reproduces a tree with exactly the same structure. (Hint: It might be helpful to study the lecture on Binomial heaps/Fibonacci heaps.)

Extra Credit: Give an $O(\log n)$ upper bound. Specifically, prove that if a sequence of finds are applied to a single tree of n nodes, the amortized cost of a find is $O(\log n)$.

3. Practice problem: Matching in Trees (5 points)

Problem 7.95 in Manber: design a linear time algorithm for finding a maximum matching in a tree. (I.e., find the largest set of edges such that no two of them have an endpoint in common.) We are calling this a "practice problem" because the solution is in the book, and yes, you are allowed to look at it. But we want you to solve and hand it in anyway.