Massachusetts Institute of Technology
6.046J/18.410J: Introduction to Algorithms
Professors Piotr Indyk and Bruce Tidor

Handout 20
April 25, 2003

Problem Set 7

This problem set is due **at the beginning of** class on *Thursday, May 8, 2003*.

Each problem is to be done on a separate sheet (or sheets) of paper. Mark the top of each sheet with your name, 6.046J/18.410J, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

Problem 7-1. Integral Flows

An important issue with network flow problems is that of integrality. A flow through a graph G is said to be *integral* if the flow on every edge of G is an integer. If the flow on any edge is not an integer, we say the flow is *fractional*.

If the capacity of every edge in G is an integer, then it is not difficult to see that the Ford-Fulkerson algorithm and its variants (e.g. Edmonds-Karp) will always compute an integral flow. This is easy to prove by induction on the number of iterations performed by the algorithm, by noting that we always augment flow in integer amounts, thereby maintaining as an invariant the fact that all edges in the residual network will have integral residual capacities. Integrality is an important property that lets us solve many types of problems, for example bipartite matching problems, using network flow techniques (see CLRS, section 26.3, for more detail).

Let us now take an instance of the maximum flow problem with integral capacities. By the argument above, there must exist some integral max flow for this problem, and we could find it by running the Ford-Fulkerson algorithm. Suppose, however, that we only have at our disposal a max flow algorithm that does not guarantee integrality of its output, so it may generate a fractional max flow (note that the *value* of this flow will still be an integer). Devise an algorithm that runs in $O(VE)$ time that converts a fractional max flow into an integral max flow. Hint: consider the subgraph of edges on which there is fractional flow.

Problem 7-2. Intersection Counting

Suppose you are given a collection of n line segments in the plane, each of which is either horizontal or vertical. For simplicity, assume that no two horizontal segments overlap, and no two vertical segments overlap. In lecture, we discuss an algorithm that computes and outputs all of the intersection points between segments in our set. The running time of this algorithm is *output-sensitive* — it runs in $O(n \log n + P)$ time, where P is the number of points output by the algorithm. In the worst case, its running time could be quadratic.

Give an algorithm that *counts* the total number of intersection points in only $O(n \log n)$ time.

Problem 7-3. Painting Rectangles

When not solving algorithms problems, TA Mihai Bădoiu enjoys painting pictures of overlapping rectangles. For his latest project, he has chosen a set of n rectangles in the plane, each of which is specified by three numbers, given to you in arrays $x_1[1 \ldots n]$, $x_2[1 \ldots n]$, and $y[1 \ldots n]$. The corners of the i'th rectangle will be $(x_1[i], 0)$, $(x_1[i], y[i])$, $(x_2[i], y[i])$, and $(x_2[i], 0)$. The rectangles may overlap each-other.

Mihai is curious how much paint he will need to buy to color in all of the rectangles. Help him out by devising an algorithm that computes, in $O(n \log n)$ time, the area of the union of all n rectangles.

Problem 7-4. Graduate student party

GSC organizes a party. Since this is a *graduate* student party, GSC must ensure that none of the students in the party has too much fun. In particular, it must ensure that no participating student is too *sociable*. We say that a student s is sociable, if there are 5 or more other people within a two-meter radius from s.

To solve this issue, GSC decides to attach a GPS transmitter to every entering student. The (x, y)-coordinates of all students are measured every few seconds. Assuming there are n students, their coordinates are represented by a sequence $(x_1, y_1) \ldots (x_n, y_n)$. GSC now just needs a fast algorithm that, given the student coordinates, determines if there is any sociable student in the party (if so, the party is declared a failure and closed immediately).

Give an efficient algorithm for checking if any of the students is sociable (the algorithm does *not* have to *find* such student if one exists). Your algorithm can be randomized and should run in expected $O(n)$ time.