Massachusetts Institute of Technology | Handout 17
6.046J/18.410J: Introduction to Algorithms | April 5, 2001
Professors Piotr Indyk and Madhu Sudan

Problem Set 5

This problem set is due in class on *Thursday, April 12th* at the end of class.

There are four problems. **Each problem is to be done on a separate sheet (or sheets) of paper. Separate problems should *not* be stapled together.** Mark the top of **each and every sheet** with your name, 6.046J/18.410J, the **problem number**, your **recitation section, your TA's name**, and the date. It is very important to hand in problems separately and to indicate your recitation section **and** TA name on every sheet. Failure to do so will result in delays in returning your work and possible confusions with entering your grade. All problem sets throughout the semester should be handed in this way.

Reading: Chapters 16, 22, 24 of CLR.

Problem 5-1. Computing Logs.

You bring an L-foot log of wood to your favorite sawmill. You want the log cut at k specific places, $L_1, L_2, \ldots, L_k$ feet from the left end, where $L_1 < L_2 < \ldots < L_k < L$. The sawmill charges x dollars to cut an x-foot log once, in any place you want. Give an efficient algorithm to minimize the total cost.

Problem 5-2. Making Change

Consider the problem of making change for n cents using the least number of coins. Ben Bitdiddle, a cashier at the local Star Market, claims that a greedy algorithm is correct: "It's best to first use as many quarters as possible, because using smaller denominations would require more coins for the same amount of money. Then use as many dimes as possible, by the same reasoning. The same goes for nickels. Then use pennies to finish the job."

(a) Ben's manager questions his reasoning. Is this a valid argument that the greedy algorithm is optimal? Explain why or why not.

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

(c) Give a set of coin denominations for which the greedy algorithm does not always yield an optimal solution. (Your set must include pennies, so that exact change can always be made.)

(d) Give an $O(nk)$ dynamic programming algorithm which works for any set of k different coin denominations. You may assume that the set includes pennies.

Problem 5-3. Greedy Graphs

In this problem, assume that graphs are given by an adjacency-list representation.

(a) An **independent set** of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that for any two vertices $u, v \in V'$, $(u, v) \notin E$ (i.e., there is no edge between any two vertices in an independent set). A **maximal independent set** is an independent set V' such that for all vertices $v \in V - V'$, the set $V' \cup \{v\}$ is not independent (i.e., every vertex not in V' is adjacent to some vertex in V'). Give an $O(|V| + |E|)$ algorithm for finding a maximal independent set for any graph $G = (V, E)$.

(b) A **k-coloring** of an undirected graph $G = (V, E)$ is a function

$$\chi : V \to \{1, 2, 3, \ldots k\}$$

such that $\chi(u) \neq \chi(v)$ for every edge $(u, v) \in E$. In other words, each vertex is given a color (a number in $\{1, 2, 3, \ldots, k\}$), and adjacent vertices have different colors. Give an $O(|V| + |E|)$ algorithm that finds a $(d + 1)$-coloring of a graph $G = (V, E)$, where d is the largest degree of the vertices of G.

Problem 5-4. The knapsack problem

In this problem, we explore variants on the famous "knapsack" problem. We have a knapsack of a given size M, and n objects with integer sizes $m_1, m_2, \ldots, m_n$. The general idea of the problem is to find a subset of objects that exactly fit into the knapsack, where by "fit," we mean that the sum of the sizes of objects in the subset equals the size of the knapsack (i.e., the knapsack is one-dimensional).

(a) Give a recursive algorithm to determine whether there exists a subset of the n items that fits exactly into a knapsack of size M. How many distinct subproblems actually occur during the execution of your algorithm?

(b) Give an $O(nM)$-time dynamic programming algorithm to solve the problem.

(c) Improve your algorithm to use $O(M)$ space.

(d) Improve your algorithm to print out the subset if it exists. Maintain the $O(M)$ space bound if you can.