Problem Session #1 Hints

Prepared By: Zachary Friggstad

This document contains hints and solutions for the problems covered in the earlier problem sessions.

Power of a String

Let x be a string and k be a positive integer. Let x^k denote the concatenation of x with itself k times. For example, $(abc)^3 = abcabcabc$. Finally, for a string z let $\rho(z)$ denote the largest integer k such that z can be written as x^k for some x. For example, $\rho(ababababab) = 5$.

The problem: given a string z, compute $\rho(z)$ in $O(|z|)$ time.

UVa Number: 10298

Hint: Consider the prefix function π as used in the KMP algorithm for string matching. What is $\pi(|z|)$? For $z = ababababab$ we have $\pi(10) = 8$.

Directed and Acyclic Graphs

Let $G = (V, E)$ be a directed and acyclic graph.

The problem: compute the length of a longest path in $O(|V| + |E|)$ time.

UVa Number: 103, 10000

Hint: For a given vertex $v \in V$, let $f[v]$ denote the length of the longest path in G starting at v. How does this length relate to $f[u]$ for $u \in V$ with $vu \in E$? Use dynamic programming.

Maximum Sum Submatrix

Given a square array M of dimension n, define the sum of a rectangular subarray M' as the sum of all numbers in M'.

The problem: find a rectangular subarray of M with maximum possible sum in $O(n^3)$ time.

UVa Number: 108

Hint: Let $f[r, c]$ denote the sum of rectangular subarray of M with one corner at entry $[1, 1]$ and the other at $[r, c]$. This can be computed in $O(n^2)$ time. Observe that the sum of any rectangular subarray of M can be computed in constant time given the table f. This yields an $O(n^4)$ algorithm; simply guess the lower-left and the upper-right corner of the rectangular subarray and use the f table to compute its sum.

Here is how we can get $O(n^3)$ time. Recall that the maximum sum substring of a 1 dimensional array algorithm simply walks through the array one entry at a time and keeps a running total of the entries. If this total ever becomes negative then set it to 0. Output the largest of these totals. Use this as an auxiliary function to solve the two dimensional problem in the following way. Guess the top and bottom row r_1 and r_2 of the subarray. Using the table f computed previously and the maximum sum substring algorithm for 1 dimensional arrays we can determine the maximum sum rectangular subarray with top row r_1 and bottom row r_2 in linear time.

Maximum All-1 Submatrix

Given a square array M of dimension n with entries only being 0 or 1, an all-1 subarray is a rectangular subarray with all entries being 1.

The problem: find the largest all-1 rectangular subarray of M in $O(n^3)$ time.

UVa Number: 836

Hint: How can use the algorithm to the previous problem to solve this? It is quite easy.

Covering With Intervals

Given a set of closed intervals I on the real-number line and another closed interval z, I is to cover z if z is contained in the union of all intervals in I.

The problem: select the least number of intervals of I to cover z (if possible) in $O(|I| \log |I|)$ time.

Hint: Cover the left-most point of z with the interval a that covers this point and has the farthest right endpoint of all such intervals. First, convince yourself that this can be assumed in an optimal solution.

Then, let z' be the interval starting at the right endpoint of a and ending at the right endpoint of interval z. Repeat this procedure on the resulting interval z'. Why is this correct? How can this be implemented to run in $O(|I| \log |I|)$ time? What if the intervals are open or half-open?

Colouring Intersection Points

Given a set L of n line segments in the plane of which no three meet at a common point, let V denote the set of intersection points. We say two intersection points u and v in V are adjacent if they lie on a common line of L and no other point of V lies between u and v.

The problem: in $O(n^2 \log n$ time, assign one of three numbers to the intersection points in V such that no two adjacent intersection points receive the same number. It is actually possible in $O(n^2)$ time.

Hint: Determine all intersection points in $O(n^2)$ time. Sort these intersection points by increasing x value (breaking ties by increasing y value). Assign the numbers to these points one at a time in their sorted order. When assigning a number to a point $v \in V$, there can be at most two adjacent points previous to it in the sorted ordering of V that have already been numbered. Assign v one of the numbers that was not used in a point adjacent to v that occurs before V in this ordering.

Minimizing Delay Penalties

We are given a set of n tasks where each has a job length t_i and a penalty p_i. Task i takes t_i minutes to complete and for every minute spent not working on an unfinished job i, a penalty of p_i is incurred.

The problem: scedule the jobs to minimize the total accumulated penalty in $O(n \log n)$ time.

UVa Number: 10026

Hint: Sort the jobs in increasing order of the value $\frac{t_i}{p_i}$. This is an optimal schedule. To see this, assume you have a schedule that is not sorted in this way. That is, there are two consecutive jobs i and j such that $\frac{t_i}{p_i} > \frac{t_j}{p_j}$. Can we reduce the overall penalty by swapping the order these jobs are completed?

Longest Increasing Subsequence

Given a sequence of numbers $\langle x_1, x_2, \ldots, x_n \rangle$, an increasing subsequence is a sequence $x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_k}$ where $i_1 < i_2 < \ldots, < i_x$.

The problem: find the longest increasing subsequence in $O(n^2)$ time. It is possible in $O(n \log n)$ time.

UVa Number: 481

Let A be the original sequence and B be the sequence with the same numbers as A, but in sorted order. A longest increasing subsequence of A is a longest common subsequence between A and B. Recall that this can be found in $O(|A| \cdot |B|) = O(n^2)$ time.

See notes at *http://www.algorithmist.com/index.php/Longest_Increasing_Subsequence* for the idea behind the $O(n \log n)$ algorithm. It isn't too difficult.