Midterm Examination – Errata

Introduction

There are currently two important corrections regarding midterm problems.

- Throughout Problem 4, regarding optimal binary search trees, you should assume that all keys are unique. (Priorities may still be arbitrary)
- On Problem 3, regarding longest monotone subsequences, part d has been changed. For completeness, the entire revised problem is given below.

Problem 3. Longest Monotone Sequence (25 points)

Given a sequence of numbers $X = \langle x_1, x_2, \ldots, x_n \rangle$, a *monotonically increasing subsequence* is a sequence $X' = \langle x_{a_1}, x_{a_2}, \ldots, x_{a_k} \rangle$ such that $1 \leq a_1 < a_2 < \ldots < a_k \leq n$ (i.e., the sequence is truly a subsequence), and that $x_{a_1} \leq x_{a_2} \leq \ldots \leq x_{a_k}$ (i.e., the sequence is monotonically increasing). Our goal is to find the *longest* monotonically increasing sequence. Actually, to make things easier, let's assume that we are only interested in knowing the *length* of the longest such sequence (although all of these techniques can be extended to build the actual sequence).

For example if $X = \langle 5, 1, 4, 2, 3, 8, 6, 7 \rangle$, then the longest monotonically increasing subsequence is $\langle 1, 2, 3, 6, 7 \rangle$, with length 5.

To design a dynamic programming algorithm we will consider the following subproblem for each $k \in \{1, \ldots, n\}$, namely, what is the longest monotonically increasing subsequence which ends with x_k. If we let $L(k)$ equal the *length* of the longest such sequence which ends with x_k, we claim the following recursive formula exists,

$$L(k) = 1 + \max_{\substack{j \in \{1, \ldots, k-1\} \\ \text{with } x_j \leq x_k}} L(j) \tag{1}$$

a. Prove Equation (1).
(Make sure you prove both the "$\geq$" and the "$\leq$" implied by equality.)

b. Show that the *length* of the longest monotonically increasing sequence can be computed using dynamic programming based directly on Equation (1). Show that this can be done in $O(n^2)$ time.

Additional algorithmic techniques can often be used to speed up the running time of a dynamic program. This is the case with this problem, as we will see how to improve our algorithm so that it runs in $O(n \lg n)$ time. To do so, we make the following observation,

Lemma 1. *Let $S_k \subseteq \{1, \ldots, k-1\}$ be a subset containing both a and b. If $L(a) = L(b)$ and $x_a \leq x_b$, then*

$$\max_{\substack{j \in S_k \\ \text{with } x_j \leq x_k}} L(j) = \max_{\substack{j \in S_k - \{b\} \\ \text{with } x_j \leq x_k}} L(j)$$

c. Prove Lemma 1.

In Equation 1, the value of j ranges over the entire set $\{1, \ldots, k-1\}$. For the remainder of the problem, we will consider a set $C_k \subseteq \{1, \ldots, k-1\}$, which we will call "candidates", defined as follows.

We will say $s \in C_k$ if the following three conditions are met:

- $s \in \{1, \ldots, k-1\}$
- $\nexists t \in \{1, \ldots, k-1\}$ such that $L(t) = L(s)$ and $x_t < x_s$.
- $\nexists t \in \{1, \ldots, s-1\}$ such that $L(t) = L(s)$ and $x_t = x_s$.

d. Use Lemma 1 to show that

$$L(k) = 1 + \max_{\substack{j \in C_k \\ \text{with } x_j \leq x_k}} L(j) \tag{2}$$

Furthermore, argue that if $a \in C_k$ and $b \in C_k$ then $L(a) \neq L(b)$.

Since part d states that set C_k contains at most one candidate for each possible length, we can maintain an array $candidate[1..n]$, relative to set C_k, with entries

$$candidate[l] = \begin{cases} x_a & \text{if } a \in C_k \text{ with } L(a) = l \\ \infty & \text{if no such } a \text{ exists} \end{cases}$$

e. Prove that the array $candidate[1..n]$ is sorted in non-decreasing order, that is, show that $candidate[r] \leq candidate[s]$ if $r < s$ for $r, s \in \{1, \ldots, n\}$

f. Show how we can use the array $candidate[1..n]$, together with Equation (2), in order to compute $L(k)$ in $O(\lg n)$ time.

g. What does set C_1 look like? How should we initialize the array $candidate[1..n]$ to correspond to set C_1?

h. How does set C_k compare to set C_{k+1}? After computing $L(k)$, how quickly can we update the array $candidate[1..n]$ to correspond to set C_{k+1} rather than set C_k?

i. Explain why we are now able to calculate the length of the longest monotonically increasing subsequence in $O(n \lg n)$ time overall.