COSC 240, Fall 2018: Problem Set $\#4$

Assigned: Wednesday, 10/10.
Due: Monday 10/22, at the beginning of class (hand in hard copy).
Lectures Covered: 10 to 13
Academic Integrity: You can work with other people in the class but you must write up your own answers in your own words. You can also use the textbook and talk to the professor. You may not use any other resources (e.g., material found online) or talk to people outside the class about these problems. See the syllabus for details on the academic integrity policy for problem sets.

Problems

1. In class we studied the LCS problem and provided the following recurrence for the LCS length table:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ \max\{c[i,j-1], c[i-1,j]\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases}$$

   Using this recurrence, design a recursive *divide-and-conquer* algorithm that takes as input two sequences X and Y (stored as arrays), and two positions i and j, with $1 \leq i \leq length(X)$ and $1 \leq j \leq length(Y)$, and then returns the *length* of the LCS of the sequences $X[1...i]$ and $Y[1...j]$.

2. Motivate the need for dynamic programming by proving that the worst case step complexity of your algorithm from the above problem is at least exponential in the length of the shorter input sequence. (Hint: discuss the resulting recursion tree for a bad pair of input sequences.)

3. Consider the following problem. You run ticket sales for a baseball stadium and are trying to sell the valuable seats in the row right behind the home team dugout. There are n total seats in the row.

   For each $i \in \{1, 2, ..., n\}$, let p_i be the price for a *contiguous block* of i seats (i.e., a block containing seats j, $j+1$, ... , $j+i-1$, for some $j \in \{1, 2, ..., n-i+1\}$). These prices were set by a complicated pricing algorithm so they do not necessarily increase with group size (it might be possible, for example, that $p_7 < p_4$, as groups of 4 are more common than groups of 7, and so on). With this in mind, you should not assume anything about these p_i values other than the fact that they are all integers greater than 0.

   Your goal, given a set of p_i values, is to figure out how to break up the dugout row into contiguous blocks so as to maximize the money you make selling the blocks for their corresponding block prices. (For example, if $n = 6$ and you break the row into one block of size 3, and three blocks of size 1, then you would earn $p_3 + p_1 + p_1 + p_1$ by selling the row in blocks of those sizes.)

   Because there are an exponential number of different ways to break up the row into blocks, standard brute-force algorithms are too slow. The three-part problem that follows asks you to develop a more efficient dynamic programming solution.

   (a) For each $i \in \{1, 2, ..., n\}$, let $q[i]$ be the maximum amount of money you can make breaking up the first i seats of the row into contiguous blocks of size i or less. It is clear to see that $q[0] = 0$, $q[1] = p_1$, and $q[n]$ is the final answer you are trying to calculate.

   Define $q[i]$ with a recurrence by filling in the blank line in the following:

$$q[i] = \begin{cases} 0 & \text{if } i = 0 \\ p_1 & \text{if } i = 1 \\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ & \text{if } i > 1 \end{cases}$$

(Hint: the answer I have in mind includes a max statement containing n different values.)

(b) Using your answer from part (a) of this problem, design a dynamic programming algorithm that calculates $q[n]$.

(c) Show how to update the algorithm from part (b) so that it in addition to returning $q[n]$, it also prints the sizes of the blocks whose prices add up to $q[n]$. You can either write a new algorithm from scratch, or just describe the new lines required for this printing and specify where they should be added to your part (b) solution.

4. Fix some hash range $m > 1$ and universe $U = \{1, 2, ..., 2^m\}$. Consider the following collection of hash functions defined for this range and universe:

   $\mathcal{H}_m = \{h_1, h_2, ..., h_m\}$, where $h_i(x) = (x + i) \mod m$.

   Prove that for every $m > 1$, the collection $\mathcal{H}_m$ is *not* universal.

5. In the verbose lecture notes on hashing, I discussed implementing hash tables with open addressing. Open addressing requires an *extended hash function* that maps each key to a permutation of the values $0, 1, ..., m - 1$. Consider the following simple definition of an extended hash function h:

$$\forall k \in U : h(k) = \langle 0, 1, 2, ..., m - 1 \rangle.$$

   Given an open address hash table implemented with h, how many probes are required for an unsuccessful SEARCH, assuming that $\alpha < 1$? Express your answer with respect to n.