

Spencer Shimko
CMSC 441
HW 6

1) 15.4-4

Show how to compute the length of an LCS using only $2 * \min(m, n)$ entries in the c table plus $O(1)$ additional space. Then show how to do this using $\min(m, n)$ entries plus $O(1)$ additional space.

$2 * \min(m, n) + O(1)$:

The method of table building described in the text stores $c[i, j]$ entries in a table $c[0..m, 0..n]$ and a second table $b[1..m, 1..n]$. Elements in table b point to the optimal subproblem solution chosen when computing $c[i, j]$. Any given $c[i, j]$ depends only on $c[i - 1, j - 1]$, $c[i - 1, j]$ and $c[i, j - 1]$. That is table entries only need to be stored in 2 rows for $i, j, i - 1, j - 1$ at any given time. This is true since we will not be reconstructing the elements of the LCS. Basically we will construct the table as was done in the book however each row will be overwritten. Note that the length of the LCS is still returned from the lower right corner of the table. We will call these two current rows $R0$ and $R1$:

Given: 2 sequences X, Y (elements in each sequence are represented as lowercase x, y)

```
LCS-Length ( X, Y )
m <- length(X)
n <- length(Y)
for j from 0 to m do
    R0[j] <- 0
R1[0] <- 0
for i from 1 to n do
    for j <- 1 to m do
        if xj = yj then
            R1[j] <- R0[j - 1] + 1
        else if R0[j] > R1[j - 1] then
            R1[j] <- R0[j]
        else
            R1[j] <- R1[j - 1]
    for k from 1 to m do
        R0[k] <- R1[k]
return R1[n]
```

$\min(m, n) + O(1)$:

Building off what was developed above: we see that $R0$'s

elements $0..j$ are only needed until $R1[j]$ is calculated. That is we are only using $0..j$ elements in $R1$ and $j..n$ elements in $R0$. Since this is the case we can now calculate the LCS in $\min(m, n) + O(1)$ by modifying the above psuedocode to store all elements in a single row which we can call R . We must add 2 variables to mark indices (both previous and current) since we are using only a single row. We will call these p and u respectively. Note that although we are returning the "rightmost" element from a single row it is the same as returning the lower right from the original method utilizing a full table although it may appear counter intuitive. Only slight modifications to the above psuedocode are required to use a single row and the new index pointers.

```
LCS-Length ( X, Y )
m <- length(X)
n <- length(Y)
for j from 0 to m do
  R[j] <- 0
for i from 1 to n do
  p <- 0
  for j <- 1 to m do
    u <- R[j]
    if xj = yj then
      R[j] <- p + 1
    else if R[j] < R[j - 1] then
      R[j] <- R[j - 1]
  p <- u
return R[n]
```

2) 15.4-5

Give an $O(n^2)$ -time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

This problem can be treated as a LCS search to achieve this type of runtime. Consider a given sequence X . Sort X and create a second sequence Y . An in-place sort can be used but a copy of the original sequence must be made if this is the case. This sorting can be done in anything from $O(n^2)$ to $O(n \lg n)$ for this algorithm. This runtime will be dropped in calculating the sum total of the runtimes as long as it is less than $O(n^2)$. This sorted sequence may contain duplicates and since the sequence we are looking for is "monotonically increasing" we must remove any duplicates from this sequence. This process can actually

be added to the sorting algorithm without significantly impacting performance. Now we run LCS-Length on X , Y and build the LCS table. We can not use the space optimized algorithms described above however since we must be able to reconstruct the elements of the LCS. If we follow the algorithm described in the book to create the table we will be able to reproduce the LCS that is monotonically increasing in $O(n^2)$.

3) 15-4

Professor Stewart is consulting for the president of a corporation that is planning a company party. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. The personnel office has ranked each employee with a conviviality rating, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.

Professor Stewart is given the tree that describes the structure of the corporation, using the left-child, right-sibling representation described in Section 10.4. Each node of the tree holds, in addition to the pointers, the name of an employee and that employee's conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality ratings of the guests. Analyze the running time of your algorithm.

Red-black tree exhibits some of the properties we need to describe in our tree so we will start by coloring all nodes in black. Any employee that will be coming to the party will have his/her representative node changed to red with the limitation that any red node must have a black node for a parent. The coloring must be done to maximize the sum of the conviviality ratings for the red nodes.

We can use recursion for calculating conviviality by calculating the sum of the conviviality at each subtree. There are two cases to consider here:

The most simple case is when the node the node is a leaf. If it is a leaf and red the conviviality of that subtree (actually only a single node) is marked as the conviviality of the node. If the node is black the conviviality should be marked as 0.

The second case is slightly more complex. If the node is colored red then the conviviality of this particular node is the sum of the conviviality of it's white child (recursively... so it's child's children's vitality etc).

"It's white child" is important here because since the node is red only children of it's white child may also be attending the party. However, if the node is white, the vitality of this node is the recursive sum of the maximum of it's red and white children. Since it's white the color of it's child doesn't matter.

Leaf: $V(x, c) = v$ iff x is red; 0 otherwise

Non-leaf: $V(x, c): v + \text{Sum}(V(\text{child}, \text{white}))$ if node red
 $\text{Sum}(\max(V(\text{child}, \text{white}), V(\text{child}, \text{red})))$

By calling maximizing root $\max(V(\text{root}, \text{red}), V(\text{root}, \text{white}))$ we will optimize this tree recursively. The runtime of this algorithm is $O(n)$.

4) 15-7

Suppose you have one machine and a set of n jobs a_1, a_2, \dots, a_n to process on that machine. Each job a_j has a processing time t_j , a profit p_j , and a deadline d_j . The machine can process only one job at a time, a job a_j must run uninterruptedly for t_j consecutive time units. If job a_j is completed by its deadline d_j , you receive a profit p_j , but if it is completed after its deadline, you receive a profit of 0. Give an algorithm to find the schedule that obtains the maximum amount of profit, assuming that all processing times are integers between 1 and n . What is the running time of your algorithm.