

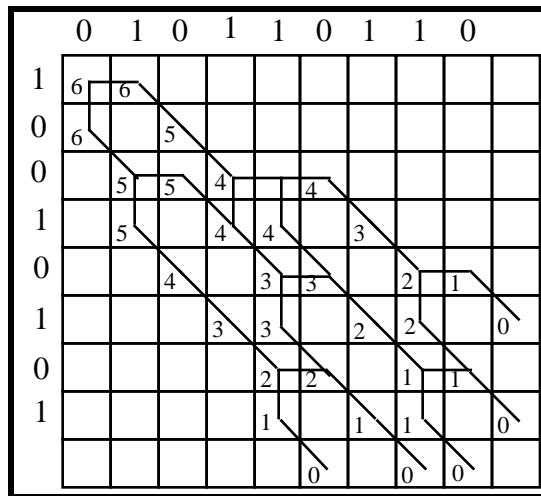
PROBLEM SET 7
Due: Thursday, April 19

Reading: CLR Chapter 16 (Dynamic Programming); CLR Chapter 17, Sections 17.1--17.3 (Greedy Algorithms)

Suggested Problems: 16.1-1; 16.3-1; 16.3-6; 16-2; 16-3; 16-4; 17.1-1, 17.1-3, 17.2-2, 17.2-3, 17.2-4, 17.2-5, 17.2-6; 17.3-1, 17.3-8; 17-1

Problem 1 [15] Use the memoized longest common subsequence (LCS) algorithm presented in class to determine *all* of the longest common subsequences of the sequences BACCABABC and ACBABCCAB.

As a concrete example of a concise way to present your results, here is a solution to CLR 16.3-1, which asks for the LCS of 10010101 and 010110110:



In the table, each diagonal line corresponds to a "prepend" step, and each pair of a vertical and horizontal line corresponds to a "max" step. The numbers in each box indicate the length of an LCS starting from that point. Note that only those slots of the table actually visited by the memoized algorithm are actually filled in.

From the table it is possible to read of the following length-6 LCSs of the two sequences:

- 010101
- 101101
- 101011
- 101010
- 100110

Problem 2 [25] Solve CLR 16.3-5 (p. 319) using the two strategies described below. Assume that the input is stored in an array $A[1..n]$. For each strategy:

- describe your algorithm (in English is fine)
- briefly argue why it is correct
- briefly argue why it takes $O(n^2)$ time.

a[10] Develop a solution that uses the (mn) LCS algorithm as a black box as part of the solution. Note: you can express a solution of this form in just a few lines!

b[15] Develop a solution that does *not* use the LCS algorithm as a black box, but instead uses an auxiliary array $M[1..n]$, where each $M[i]$ stores the longest monotonically increasing sequence in $A[1..n]$ that begins with $A[i]$.

Problem 3 [30]

A binary tree is either (1) a leaf or (2) a node with left and right subtrees. (In this definition, nodes do not carry values.) The following function counts the number of distinct binary trees with n nodes:

```
Count-Trees (n) =  
  if n = 0 then  
    return 1 {There is one leaf}  
  else  
    count <- 0  
    for i = 0 to n-1 do  
      count <- count + (Count-Trees(i) * Count-Trees((n-1)-i))  
    return count
```

For example,

```
Count-Trees(0) = 1  
Count-Trees(1) = 1  
Count-Trees(2) = 2  
Count-Trees(3) = 5
```

$\text{Count-Trees}(n)$ is also known as the n th Catalan number. It can be shown that $\text{Count-Trees}(n)$ takes time exponential in n (CLR 13-4, p. 262).

a[10] Write pseudocode for a memoized version of Count-Trees .

b[10] Write pseudocode for a dynamic programming solution to Count-Trees .

c[5] Use the solution to part (b) to calculate $\text{Count-Trees}(8)$.

d[5] What is the running time of your solutions to parts (a) and (b). (They should be the same!) Justify your answer.

Problem 4[10] CLR 17.1-2 (p. 333) Informally justify that your algorithm is optimal.

Problem 5[20] CLR 17.2-4 (p. 337) Formally prove that your algorithm is optimal. Sections 17.1--17.2 of the book (especially the proof on p. 332) will help you in proving that your strategy yields an optimal solution. As indicated by the proof on p. 332, there are two aspects to a proof that a greedy algorithm is optimal:

1. Show that the algorithm exhibits the **greedy choice property**: an optimal solution can begin with the greedy choice. The strategy for proving this is as follows. Assume you are given an optimal solution S . Then show that (1) S begins with the greedy choice or (2) if S does not begin with the greedy choice, then replacing whatever it begins with by the greedy choice leads to another optimal solution. (Note: there may be many optimal solutions.)

2. Show that the problem exhibits the **optimal substructure property**: the optimal solution for the whole problem can be derived from the optimal solution for the parts. As shown on p. 332, this is usually shown by the following strategy. Assume that you are given an optimal solution S for the whole problem and show that you can extract the optimal solution S_i for the subproblems from the optimal solution to the whole problem. The proof proceeds by contradiction: assume that there is a better solution S'_i to one of the subproblems and show that you could construct a better whole solution S' using S'_i . But that contradicts your original assumption about the optimality of S . So it must be the case that your other assumption is incorrect: S'_i cannot be better than S_i . That is, each S_i is optimal.

Extra Credit Problem 1 [20] CLR 16.3-6 (p. 319)

Extra Credit Problem 2 [25] CLR 16.2 (p. 325)

Problem Set Header Page
Please make this the first page of your hardcopy submission.

CS231 Problem Set 7
Due Thursday, April 19, 2001

Name:

Date & Time Submitted (*only if late*):

Collaborators (*anyone you collaborated with in the process of doing the problem set*):

*In the **Time** column, please estimate the time you spent on the parts of this problem set. Please try to be as accurate as possible; this information will help me to design future problem sets. I will fill out the **Score** column when grading your problem set.*

Part	Time	Score
General Reading		
Problem 1 [15]		
Problem 2 [25]		
Problem 3 [30]		
Problem 4 [10]		
Problem 5 [20]		
Extra Credit 1 [25]		
Extra Credit 2 [25]		
Total		