

CMSC 451: Homework 4, Spring 2004

Due at the beginning of class on March 16.

Warning: some of the problems require thought - do not wait until the last day to start working on them! If you cannot come up with algorithms that run in the required time, then provide (correct) slower algorithms for partial credit. Write your answers using *pseudo-code* in the same style as the textbook. These make the algorithm description precise, and easy to read (as opposed to code in C or some other language).

Important Note: The two problems here are more challenging than usual. For either one (or both) of these problems, you are allowed to “buy a hint”: if you choose to do so, we will give you a hint for the problem, and your final score for that problem will be 70% of the score you obtain for that problem. Please email the instructor or the TA if you want the hint; you can set up a time with them to come and get the hint. Therefore, you are encouraged to think about these problems as soon as possible, to decide if you want to buy the hint.

1. We are given a collection of n books, which must be placed in a library book case. Let $h[1..n]$ and $w[1..n]$ be arrays, where $h[i]$ denotes the height of the i -th book and $w[i]$ denotes the width of the i -th book. Assume that these arrays are sorted according to the books' catalog numbers, and the books *must* be shelved in this order, from top-to-bottom and from left-to-right. (For instance, we may put books 1, 2, 3 from left-to-right in the top shelf, books 4, 5 in the second shelf from the top, books 6, 7, 8, 9 in the third shelf from the top, etc.) Also, the books are stacked in the usual upright manner (i.e., we *cannot* lay a book on its side). All book cases have width W , but you may have any height you like. The books are placed on shelves in the book case. You may use as many shelves as you like, placed wherever you like (up to the height of the chosen book case), and you may put as many books on each shelf as you like (up to the width of the book case). Assume for simplicity that shelves take up no vertical height.

The *book-shelver's problem* is, given $h[1..n]$, $w[1..n]$, and W , what is the minimum height book-case that can shelve all of these books?

- (a) Write down a recurrence for book-shelver's problem.
- (b) Use your recurrence to develop an efficient dynamic programming algorithm for solving the book-shelver's problem.
- (c) Analyze the running time of your algorithm.

Hint: Let $height[i]$ denote the minimum height needed to shelve the first i books, starting from the topmost shelf. Show how to compute $height[i]$ for all i from 1 to n . Reason about the number of books you put in the last shelf, when you need to shelve the first i books.

2. (**For graduate students only**) This problem was inspired by a question asked by a student in class. Consider the recurrence for $m[i, j]$ (related to

matrix-chain multiplication) given at the top of page 335. Recall that we want to compute the value $m[1, n]$, and that the dynamic programming solution in the book computes these values “bottom up” in increasing values of the “length” $j - i + 1$; this takes $O(n^3)$ time.

Suppose we instead take the following recursive approach. We try to directly implement the recurrence for $m[i, j]$ given at the top of page 335 as a recursive procedure, *with the following modification*. We also have variables $s[i, j]$ for all i, j , which are always either 0 or 1; “ $s[i, j] = 0$ ” means that value $m[i, j]$ has not been computed yet, and “ $s[i, j] = 1$ ” means that value $m[i, j]$ has already been computed. (Thus, all the $s[i, j]$ are initialized to zero when we start.) Now, in the recursive procedure to calculate $m[i, j]$ using the above recurrence, we do two things:

- whenever we have (recursively) computed a value such as $m[i, j]$, we go ahead and store the value, and also set $s[i, j] := 1$.
- when considering $m[i, k]$ or $m[k + 1, j]$ in the recurrence relation, we go ahead and recursively find these values *only if* the corresponding “ s ” variable (i.e., $s[i, k]$ or $s[k + 1, j]$) is currently zero; if not, we directly use the already-computed value.

Now suppose we try to compute our required value $m[1, n]$ using the above modified recursive algorithm. Show that the running time is still polynomial in n : i.e., $O(n^c)$ for some constant c .