

CSE 101 Homework 4
Dynamic Programming
Due Thursday, June 2
100 points total = 10 %

Grade maximization Remember the grade maximization problem from the midterm:

We are taking a class with k projects. We have H hours to divide among the projects, and will spend an integer number of hours on each project. For every project i , we are given an array $G_i[1..H]$ so that, if we spend h hours on project i , our grade for that project will be $Grade_i = \sum_{j \leq h} G_i[j]$. If we spend no time on a project, we will get a 0.

We need to allocate H hours among projects $1..n$, i.e., find non-negative integers $h_1, ..h_k$ with $\sum h_i = H$ in order to maximize $\sum_i Grade_i$.

On the midterm, you showed that the special case of diminishing returns could be solved with a greedy algorithm, but not the general case. Here, we will devise a dynamic programming algorithm for the general case.

The following (corrected) recursive algorithm for the general grade maximization problem is based on the possible answers to the question: How many hours do I devote to project 1? The possible answers are $0..H$. The recursion just returns the best achievable total grade, not the assignment that achieves it.

BTGM($H, G_1[0..H], ..G_k[0..H]$);

1. IF $n = 1$ return $\sum_{i=1}^{i=H} G_1(i)$.
2. $BestGrade \leftarrow 0, Grade_1 \leftarrow 0$.
3. FOR $h = 0$ to H do:
4. $ThisGrade \leftarrow Grade_1 + BTGM(H - h, G_2, ..G_k)$
5. IF $ThisGrade > BestGrade$ THEN $BestGrade \leftarrow ThisGrade$.
6. IF $h < H$ THEN $Grade_1 \leftarrow Grade_1 + G_1[h + 1]$.
7. Return $BestGrade$.

Part1: 2 points Illustrate the above algorithm on the following input (which does not have decreasing returns) : $k = 3, H = 3, G_1[1..3] = [70, 10, 20], G_2[1..3] = [30, 70, 0], G_3[1..3] = [35, 40, 25]$. (as a tree of recursive calls and answers.)

Part2: 3 points Give an upper bound on the number of recursive calls the above algorithm makes, in the worst-case. (Be sure to explain your answer.)

Part 3: 10 points Give a dynamic programming version of the recurrence.

Part 4: 3 points Give a time analysis of this dynamic programming algorithm.

Part 5: 2 points Show the array or matrix that your dynamic programming algorithm produces on the above example.

For each of the following three problems, describe the fastest dynamic programming algorithm you can find, and give a time analysis (in terms on any of the given parameters).

Gizmos (20 points) You wish to purchase (at least) n identical gizmos. Gizmos come in packages of different sizes and different prices. You can buy any number of packages of each size, as long as the total number is at least n . You wish to find the minimum total price of such a set of packages.

The input is given as n and an array $Packages[1..m]$, where each $Package[i]$ has a positive integer field $Package[i].size$ and a positive real field $Package[i].price$ giving the number of gizmos in the package and the price of the package.

Coffee shops Smallville is the last city on Earth not saturated by Big Bucks coffee shops. Smallville has one business street with n blocks. The profit associated with putting a coffee shop on block i is given in an array $Profit$ as $Profit[i]$. However, we cannot put coffee shops within $d \geq 1$ blocks from each other, i.e., if we put a shop in block i then we cannot put one in block $i - d, i - d + 1, i - 1$ or $i + 1, i + 2, \dots, i + d$. We wish to place stores on a subset of the n blocks meeting this constraint in order to maximize the total profit.

Library storage-20pts A library has n books that must be stored in alphabetical order on adjustable height shelves. Each book has a height and a thickness. The width of the shelf is fixed at W , and the sum of the thicknesses of books on a single shelf must be at most W . The next shelf will be placed on top, at a height equal to the maximum height of a book in the shelf. Give an algorithm that minimizes the total height of shelves used to store all the books. You are given the list of books in alphabetical order, $b_i = (h_i, t_i)$, where h_i is the height and t_i is the thickness, and must organize the books in that order.

Implementation: Implement both the memoized recursive and dynamic programming version of algorithms for the longest increasing subsequence problem. Time them on randomly permuted arrays of length n with a wide variety of lengths n . (Plot time and n on a log-log scale.) Compare their two performances, and give an explanation for any differences.