

15-451 Algorithms, Fall 2017

Recitation #7 Worksheet

Dynamic Programming

Off-Line Stock Market Problem You're given a sequence of stock prices $[p_1, p_2, \dots, p_n]$. You want to find the maximum profit that you could have made on the stock in hindsight. In other words, you want to find i and j with $1 \leq i \leq j \leq n$ such that $p_j - p_i$ is maximal. Your algorithm should run in $O(n)$ time.

Solution: Scan the sequence from first to last. After processing p_i keep two things: (1) m_i the minimum of p_1, \dots, p_i , and (2) g_i the maximum profit achievable so far. It's easy to update these when processing the next stock price p_{i+1} .

$$m_{i+1} = \min(m_i, p_{i+1})$$

$$g_{i+1} = \max(g_i, p_{i+1} - m_{i+1})$$

The final answer is g_n . (For completeness, note that $m_0 = \infty$ and $g_0 = -\infty$.)

Longest Increasing Subsequence: Given an array A of n integers like [7 2 5 3 4 6 9], find the longest subsequence that's in increasing order (in this case, it would be 2 3 4 6 9). Give a dynamic-programming algorithm that runs in time $O(n^2)$ to solve this problem.

1. To keep things simple, first let's say you just need to output the *length* of the longest-increasing subsequence. E.g., in the above case, the length is 5.

Solution: Suppose that for each $i' < i$ you have computed the length of the LIS of $A_{0..i'}$ that ends with $A[i']$. How would you use this to solve the corresponding problem for i ? $L[i] = \max\{L[i'] + 1 : i' < i, A[i'] < A[i]\}$, or $L[i] = 1$ if there are no such i' .

2. Now extend your solution to actually find the LIS.

Solution: One approach is when computing the max above, to also have a separate array that stores the argmax, that is, the index i' such that $L[i] = L[i'] + 1$. One can then read off the sequence by going backwards from the end.

Making Change: You are given denominations v_1, v_2, \dots, v_n (all integers) of the various kinds of currency you have. (Say $v_1 = 1$, so you can make change for any integer amount $C \geq 1$.) Given C , give a dynamic programming solution which makes change for C with the fewest bills possible.

(Again, as a first stab, compute the number of bills required, and then extend the solution to output the number of bills of each denomination needed.)

Solution: Create an array B where $B[C']$ represents the fewest bills needed to make change for C' . We can fill this in using the formula $B[C'] = \min\{B[C' - v_i] + 1 : v_i \leq C'\}$, where we begin with $B[0] = 0$ and then work upward from $C' = 1$ to C . The total time taken is $O(Cn)$.

Making Change (Part II): Now suppose you have only one bill of each denomination i . Given C , give a dynamic programming solution which makes change for C using the fewest bills, using no more than one bill of each denomination i (or says this is not possible).

Solution: One approach is to create a 2-dimensional array B where $B[C', i]$ represents the fewest bills needed to make change for C' using denominations $1, 2, \dots, i$ only (or infinity if it is not possible). Base case $B[0, 0] = 0$ and $B[C', 0] = \infty$ for $C' > 0$. For general values of i we have $B[C', i] = \min(B[C', i - 1], B[C' - v_i, i - 1] + 1)$ if $C' - v_i \geq 0$ or else $B[C', i] = B[C', i - 1]$ if $C' - v_i < 0$.

Making Change (Part III): Can you solve the problem if you have ℓ_i bills of denomination i ?

Solution: We can just modify the formula for B above to:

$$B[C', i] = \min\{B[C' - jv_i, i - 1] + j : 0 \leq j \leq \ell_i, C' - jv_i \geq 0\}.$$

Balanced Partition. You have a set of n integers each in the range $0, \dots, K$. In time $O(n^2K)$, partition these integers into two subsets such that you minimize $|S_1 - S_2|$, where S_1 and S_2 denote the sums of the elements in each of the two subsets.

Solution: Let S be the sum of all the integers. Then $S \leq nK$. To minimize $|S_1 - S_2|$ it suffices to find a set A_1 whose numbers sum to $S_1 \leq \lfloor S/2 \rfloor$, that is as close to $S/2$ as possible. And this can be done by a dynamic program like for knapsack, in time $O(nS) = O(n^2K)$.