ADA 2023 Tutorial 3

Diptapriyo, Debarka

February 2023

This tutorial is more warmup on DPs. What we would like you to do for each of the problems is:-

1. Define the subproblems clearly
2. Write a recursion using the above definition and argue properly about why the recursion is correct (this is the optimal substructure property)
3. Implement using tables and argue runtime.

1 Maximizing Revenue by choosing jobs in different weeks

Suppose that there are n weeks, a plan is specified by a 'low stress job' and a 'high stress job' or 'none' in each of the weeks. For any $i \geq 2$, if a high stress job is chosen for the week i, then no job can be chosen in the week $i-1$. However, if a low stress job is chosen for the week i, then both high stress and low stress job can be chosen for the week $i-1$. In the week 1, a high stress job and low stress job both can be chosen. A low stress job gives revenue a_i and a high stress job gives a revenue of b_i. Give an algorithm that provides a plan for n consecutive weeks that maximizes the revenue.

Solution: If $n = 0$, then the scenario is obvious that the total revenue is 0. Suppose that $n = 1$. Then, both low stress job and high stress job can be chosen or no job can be chosen at all. So, $\max(a_1, b_1)$ is the solution when $n = 1$. For $n \geq 2$, the main trick here is to consider two cases. First case is that at a *low stress job* is chosen at the week i, and the second case is that a *high stress job* is chosen in week i. The first case gives a_i revenue in the i'th week and the second case gives b_i revenue in the i'th week. For the first case, the $i-1$'th week can have any job. On the other hand, for the second case, the $i-1$'th week cannot have any job but the $i-2$'th week can have any job.

So, we consider the subproblem REVENUE(i) as the optimal (maximum) revenue for the weeks $\{1, 2, \ldots, i\}$.

- **Base Cases:** REVENUE$(0) = 0$ and REVENUE$(1) = \max(a_1, b_1)$.
- **Recurrence:** REVENUE$(i) = \max\left(\text{REVENUE}(i-1) + a_i, \text{REVENUE}(i-2) + b_i\right)$.
- **Final solution:** REVENUE(n) gives the final solution.

The rest is converting this recurrence relation into an iterative algorithm by constructing the table.

- Initialize REVENUE$(0) \leftarrow 0$ and REVENUE$(1) \leftarrow \max(a_1, b_1)$.
- For $i = 2, 3, \ldots, n$ in this order set
  REVENUE$(i) \leftarrow \max\Big($REVENUE$(i-1) + a_i$, REVENUE$(i-2) + b_i\Big)$.
- Output REVENUE(n).

Correctness Proof: By induction on the number of weeks, i.e. n.

2 Maximum Switching Subarray

Suppose you are given an array of distinct real numbers, $A[1:n]$. Define a switching subsequence of A as a sequence $A[k_1], A[k_2], \cdots A[k_\ell]$ such that ,

$$A[k_i] < A[k_{i+1}], \text{for odd i}$$
$$A[k_i] > A[k_{i+1}], \text{for even i}$$

In plain English, the sequence switches between increasing and decreasing, starting with increasing. The goal is to find the longest switching sub-sequence of A. Deisgn a linear time algorithm for this.

Solution. The main trick here is to realize that you actually need to define two different subproblems for each $i = 1, 2, \cdots n$. The rest is very similar to the above problem with a minor twist.

Suppose $longestInc(i)$ denote the longest switching subsequence ending at i where the last entry is bigger than the preceding one while $longestDec(i)$ denote the longest switching subsequence ending at i where the last entry is smaller than the preceding one. Then the following is true :

$$longestInc(i) = max_{j<i, A[j]<A[i]} longestDec(i) + 1$$
$$longestDec(i) = max_{j<i, A[j]>A[i]} longestInc(i) + 1$$

Now the rest is again converting this to an iterative solution using table.

(Note: There is a simpler $\mathcal{O}(n^2)$-time DP solution to solve this problem. Can you figure that out?)