Longest Increasing Subsequence

Dragan Marković

The Problem

Given a sequence of integers $a_0, a_1, ..., a_{n-1}$. Find the contiguous sub-array with maximum sum. More often than not, just the sum is required, however, it is easy to modify the following algorithms to find the sub-array itself.

Divide and Conquer

Let's examine a sub-array whose left point is index l and right point is at index r. Let's define it's middle point as $m = \lfloor \frac{l+r}{2} \rfloor$. Them we can recursively define our solution as $solve(l, r) = max(solve(l, m), solve(m+1, r), midCrossing(l, m, r))$. That is to say solution for (l, r) is either the two of the largest solutions for (l, m) and $(m + 1, r)$, or some sub-array that crosses both sides (contains the middle element)."Mid-crossing" array must contain the element a_m, There are $m-l+1$ arrays that begin with some index $i \leq m$ and end at m. Let's denote the starting point of one with maximum sum as $imax$. Similarly, there are $r-m+1$ such arrays on the right sides, maximal will be represented with index $jmax$. Then the array with maximum sum that contains the middle element is exactly $imax, imax+1, ..., m, m+1, ..., jmax$. This step can be done in $\mathcal{O}(n)$ and there are $\mathcal{O}(\log n)$ recursive calls to $solve()$ function so the entire complexity of the solution is $\mathcal{O}(n \log n)$.It also uses $\mathcal{O}(\log n)$ stack memory for recursion.

Dynamic Programming

Let dp_i hold the maximum sum of some array that ends at index i. It is easy to see that:

$$dp_i = \begin{cases} max(dp_{i-1} + a_i,\ dp_i) & \text{if } i > 0 \\ dp_i = a_i & \text{if } i = 0 \end{cases}$$

Either we add a_i to the "best" array ending with $i - 1$ or we start a new array, only containing the element a_i. Answer is, simply, the largest dp_i.
To calculate dp_i we only need dp_{i-1} so this algorithm takes $\mathcal{O}(1)$ extra space, and $\mathcal{O}(n)$ time.