CSCI3160 Design and Analysis of Algorithms (2025 Fall)

Dynamic Programming: Longest Increasing Subsequences

Instructor: Xiao Liang

Department of Computer Science and Engineering
The Chinese University of Hong Kong

Problem Statement

Longest Increasing Subsequence (LIS)

Given an array of integers $A[1 \dots n]$, find the length of the longest strictly increasing subsequence LIS.

Example:

$$A = [10,\ 9,\ 2,\ 5,\ 3,\ 7,\ 101,\ 18]$$

LIS $= [2,\ 3,\ 7,\ 101]$
Length $= 4$

Note: The elements in LIS do not need to be contiguous in the original array A; there may be gaps between them. The only requirement is that they are in strictly increasing order.

Brute Force Idea

- Try all possible subsequences.
- Check each one for strict increasing order.
- Keep track of the longest one.

Time Complexity: $O(2^n)$ — Not feasible for large n.

A Dynamic Programming Algorithm for the Longest Increasing Subsequence Problem

Subproblems! Subproblems! Subproblems!

The core trick of dynamic programming is to break the problem into subproblems that can be solved in a nice order that later subproblems can reuse the solutions of earlier subproblems.

With that in mind, when solving a problem by dynamic programming, the most crucial question is:

> What are the proper subproblems?

Unfortunately, this step requires both experience and creativity. There is no universal method that guarantees you will always find the appropriate subproblems.

For our problem in hand: what are the proper subproblems for **Longest Increasing Subsequence**?

Dynamic Programming Idea

Key Insight:
Define subproblems to build the solution incrementally.

Let $\text{dp}[i]$ = length of the LIS ending at index i.

Recurrence Relation:

$$\text{dp}[i] = 1 + \max\left(\text{dp}[j]\right) \quad \text{for all } j < i \text{ and } A[j] < A[i]$$

If no such j exists, then $\text{dp}[i] = 1$

An Example: experience plus creativity

This is an opportunity for you to see how "experience and creativity" could help!

Recall the earlier example:

$$A = [10,\ 9,\ 2,\ 5,\ 3,\ 7,\ 101,\ 18]$$

Again, let's ask ourselves: why is this problem so hard?

- we don't even know how to choose **the first element** to put in LIS.

Wait! Isn't this reminiscent of the Rod Cutting Problem we just solved?

The “experience” part I

For **Rod Cutting**, we encountered a similar challenge: we don’t know where to make the first cut!

Our solution was to try every possible position for the first cut and choose the one that yields the maximum revenue.

Crucially, what prevents us from falling into exponential-time brute-force search is a key observation:

- Once we decide to make the first cut at position i, the optimal revenue (conditioned on this choice) is exactly $P[i] + opt(n-i)$.

This insight allows us to write the following recursive formulation of subproblems.

$$opt(n) \quad = \quad \max_{i=1}^{n}(P[i] + opt(n-i))$$

The “experience” part II

Can we migrate this idea to our current problem of **Longest Increasing Subsequence**?

$$A = [10,\ 9,\ 2,\ 5,\ 3,\ 7,\ 101,\ 18]$$

First Attempt: Try every possible element $A[i]$ as the first element to put in LIS, and see if we can reduce the problem to one of smaller size, i.e., **a LIS problem** for $A[i+1, i+2, \ldots, n]$.

However, you will soon realize that this attempt does not really work:

- It is unclear how $A[i]$ together with the LIS solution to $A[i+1, i+2, \ldots, n]$ could contribute the the LIS problem for $A[i, \ldots, n]$. That is, we cannot get a clean recursive formula as the following one for **rod cutting**:

$$opt(n) \quad = \quad \max_{i=1}^{n}(P[i] + opt(n-i))$$

The "creativity" part I

If you spend enough time on this problem, the following idea may strike you:
/* this is the creative (or magical) step */

- What if, instead of focusing on the first element of the LIS, we shift our attention to the last element?

Another Attempt: Try every possible element $A[i]$ as the last element to put in LIS, and see if we can reduce the problem to one of smaller size, i.e., **a LIS problem** for $A[1, 2, \ldots, i-1]$.

Emm... not really working yet

- we don't know if the last element in the LIS for $A[1, 2, \ldots, i-1]$ is really smaller than $A[i]$. Without this information, it is still unclear how to combine the solution of the subproblem (i.e., LIS for $A[1, 2, \ldots, i-1]$) and the value $A[i]$ to build what we want: LIS for $A[1, 2, \ldots, i]$.

The "creativity" part II

However, the previous discussion reveals an important insight: we must clearly identify the element at which the LIS of $A[1, 2, \ldots, i]$ ends.

This naturally leads us to adopt a more fine-grained approach: Instead of considering the LIS of $A[1, 2, \ldots, i]$ as a whole, let us focus specifically on the LIS that **ends at** $A[i]$.[1]

Final Attempt:

- Let $len_{\text{LIS}}(i)$ denote the length of the LIS that **ends at** $A[i]$.
- This leads to the following clean recursive formula among subproblems:

$$len_{\text{LIS}}(i) = 1 + \max_{\{j \mid j<i \text{ and } A[j]<A[i]\}} \left\{ len_{\text{LIS}}(j) \right\}$$

Intuitively: append $A[i]$ to the LIS ends at $A[j]$ only if $A[j] < A[i]$ (and of course this LIS ends before $A[i]$)

The “creativity” part III

The recursive formula for **LIS problem**:

$$len_{\mathrm{LIS}}(i) = 1 + \max_{\{j \mid j<i \text{ and } A[j]<A[i]\}} \left\{ len_{\mathrm{LIS}}(j) \right\}$$

If we are able to compute $len_{\mathrm{LIS}}(i)$ for each $i \in \{1, 2, \ldots, n\}$, then the length of LIS for the original array A is just

$$\max_{i \in \{1,2,\ldots,n\}} \left\{ len_{\mathrm{LIS}}(i) \right\}$$

Think: Do you believe the following recursive formula is equivalent to the one above? Why or why not?

$$len_{\mathrm{LIS}}(i) = \max_{j \in \{1,2,\ldots,i\}} \left\{ len_{\mathrm{LIS}}(j) + \delta(A[j] < A[i]) \right\}, \text{ where } \delta(A[j] < A[i]) := \begin{cases} 1 & A[j] < A[i] \\ 0 & \text{otherwise} \end{cases}$$

[1]Note that this is different from length of the LIS for $A[1, 2, \ldots, i]$, which may not necessary end at $A[i]$.

Pseudocode

LIS Algorithm

- Input: array $A[1..n]$
- Output: length of the Longest Increasing Subsequence

```
• Initialize: dp[1...n] = 1        /* dp[i] will store our value $len_{\mathrm{LIS}}(i)$ */
• For $i = 1$ to $n$:
    • For $j = 1$ to $i - 1$:
        • If $A[j] < A[i]$, set
          $\mathrm{dp}[i] = \max(\mathrm{dp}[i], \mathrm{dp}[j] + 1)$
• Return $\max(\mathrm{dp}[1...n])$
```

- Time complexity $O(n^2)$.
- As usually, piggybacking will help you recover the actual longest increasing sequence.

Example Walkthrough

Given: $A = [10, 9, 2, 5, 3, 7, 101, 18]$

Step-by-step dp:

i	1	2	3	4	5	6	7	8
$A[i]$	10	9	2	5	3	7	101	18
$dp[i]$	1	1	1	2	2	3	4	4

Answer: 4

Closing Remarks

This LIS problem nicely illustrates the value of past experience. The key lesson is:

The more dynamic programming problems you've encountered, the more intuition and tools you build up, increasing your chances of solving the next DP problem you face.

So, do enough exercise before taking the exam lol.

Also, now that we've solved this problem, we can see in retrospect that our **First Attempt** (on page 9) could have worked as well if we were to try harder—we just need to define $len_{\mathrm{LIS}}(i)$ as the length of the LIS **starting from** $A[i]$; Then, we can derive a similar recursive formula. (This is left to you as an exercise.)

That said, it's also natural that we solved the problem from the "last element" perspective, rather than the "first element" perspective—when you encounter a problem for the first time, certain viewpoints may seem more intuitive and guide you toward the solution. But once you've solved it, looking back, you'll often discover other valid approaches from different angles, even ones you initially thought were unhelpful or too difficult