CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2019**

Lecture 10

October 3, 2019

Instructor: Sepehr Assadi

1 The Longest Increasing Subsequence Problem

Recall the longest increasing subsequence problem from the last lecture.

Problem 1. The longest increasing subsequence problem (LIS for short) is defined as follows:

- **Input:** An input array $A[1:n]$ of distinct integers of length n.
- **Output:** The length of the longest *increasing* subsequence of array A, i.e., the length of longest B such that (1) B is a subsequence of A, and (2) B is increasing, i.e., $B[1] < B[2] < B[3] < \cdots$.

So far, we designed the following recursive formula for this problem.

- *Specification (in plain English):* For every $0 \leq i \leq n$, we define:
  - $LIS(i)$: the length of the longest increasing subsequence of the array $A[1:i]$ $\underline{\text{that ends in } A[i]}$ (we will shortly see why we need this extra part). We define $A[1:i]$ for $i = 0$ to be the 'empty' array as a convention and thus $LIS(0) = 0$.

  The solution to our original problem is then $\max_j \{LIS(j) \mid 1 \leq j \leq n\}$: the actual longest increasing sequence should end in some entry of the array, say $A[k]$, and thus $LIS(k)$ would be equal to its length; by taking maximum over all choices of j, we ensure that we get the correct answer then.

- *Solution:* We write a recursive formula for $LIS(i)$ as follows:

$$LIS(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max \begin{cases} \max\{1 + LIS(j) \mid 1 \leq j < i \text{ and } A[j] < A[i]\} \\ 1 \end{cases} & \text{otherwise} \end{cases}$$

  We proved the correctness of this formula in the previous lecture notes.

Before we move on from this part, it is helpful to see an example of how this recursive formula works.

Example: Consider using the formula above to solve the problem on an array $A = [1, 3, 7, 2, 4, 5, 0]$ ($n = 7$):

- $LIS(7)$: computes the length of the LIS ending in $A[7] = 0$:

$$LIS(7) = 1,$$

  by definition of the formula since $A[7]$ is smaller than all the other entries.

- $LIS(6)$: computes the length of the LIS ending in $A[6] = 5$:

$$LIS(6) = \max\{1 + LIS(5), 1 + LIS(4), 1 + LIS(2), 1 + LIS(1)\},$$

  by definition of the formula since $A[6]$ is larger than $A[5], A[4], A[2], A[1]$, but not $A[3]$.

- $LIS(5)$: computes the length of the LIS ending in $A[5] = 4$:

  $$LIS(5) = \max\{1 + LIS(4), 1 + LIS(2), 1 + LIS(1)\},$$

  by definition of the formula since $A[5]$ is larger than $A[4], A[2], A[1]$, but not $A[3]$.

- $LIS(4)$: computes the length of the LIS ending in $A[4] = 2$:

  $$LIS(4) = \max\{1 + LIS(1)\},$$

  by definition of the formula since $A[4]$ is larger than $A[1]$, but not $A[2], A[3]$.

- $LIS(3)$: computes the length of the LIS ending in $A[3] = 7$:

  $$LIS(3) = \max\{1 + LIS(2), 1 + LIS(1)\},$$

  by definition of the formula since $A[3]$ is larger than $A[2], A[1]$.

- $LIS(2)$: computes the length of the LIS ending in $A[3] = 3$:

  $$LIS(2) = 1 + LIS(1),$$

  by definition of the formula since $A[2]$ is larger than $A[1]$.

- $LIS(1)$: computes the length of the LIS ending in $A[1] = 1$:

  $$LIS(1) = 1,$$

  by definition of the formula since $A[1]$ is not larger that any (zero) entries before it.

We now need to go back and evaluate each of these subproblems:

- $LIS(1) = 1$ (the base case).
- $LIS(2) = 1 + LIS(1) = 2$.
- $LIS(3) = \max\{1 + LIS(2), 1 + LIS(1)\} = 1 + LIS(2) = 3$.
- $LIS(4) = 1 + LIS(1) = 2$.
- $LIS(5) = \max\{1 + LIS(4), 1 + LIS(2), 1 + LIS(1)\} = 1 + LIS(4) = 3$
  (note that we could have also updated $LIS(5)$ from $LIS(2)$ instead of $LIS(4)$).
- $LIS(6) = \max\{1 + LIS(5), 1 + LIS(4), 1 + LIS(2), 1 + LIS(1)\} = 1 + LIS(5) = 4$.
- $LIS(7) = 1$.

The formula then returns $\max_j \{LIS(j) \mid j \in \{1, \ldots, 7\}\} = LIS(6) = 4$ which is the correct answer. Let us also see what is the corresponding subsequence for each value of the formula:

- $LIS(1) = 1$: $A[1] = [1]$.
- $LIS(2) = 1 + LIS(1)$: $A[1], A[2] = [1, 3]$.
- $LIS(3) = 1 + LIS(2)$: $A[1], A[2], A[3] = [1, 3, 7]$.
- $LIS(4) = 1 + LIS(1)$: $A[1], A[4] = [1, 2]$.
- $LIS(5) = 1 + LIS(4)$: $A[1], A[4], A[5] = [1, 2, 4]$.
- $LIS(6) = 1 + LIS(5)$: $A[1], A[4], A[5], A[6] = [1, 2, 4, 5]$.
- $LIS(7) = 1$: $A[7] = [0]$.

Algorithm: We now turn this recursive formula into a memoization and a bottom-up dynamic programming solution and analyze their runtime.

Memoization: We pick an array $T[0:n]$ of length $(n+1)$ initialized to be all 'undefined'. We then have the following memoization algorithm.

```
MemLIS($i$):
  1. If $T[i] \neq$ 'undefined' return $T[i]$.
  2. If $i = 0$ let $T[i] = 0$.
  3. Else, let $T[i] = 1$ and for $j = 1$ to $i - 1$:
      (a) If $A[j] < A[i]$, let $T[i] = \max\{T[i], 1 + \texttt{MemLIS}(j)\}$.
  4. Return $T[i]$.
```

The answer to our problem is the obtained by computing $\texttt{MemLIS}(1), \texttt{MemLIS}(2), \ldots, \texttt{MemLIS}(n)$ and then returning the maximum value – in other words, using the following algorithm:

```
  1. Let $m \leftarrow 0$. For $j = 1$ to $n$:
          Let $m \leftarrow \max\{m, \texttt{MemLIS}(j)\}$.
  2. Return $m$.
```

Again, the correctness of this algorithm follows directly from that of the recursive formula $LIS(\cdot)$. As for the runtime, there are $n+1$ subproblems and subproblem $\texttt{MemLIS}(i)$ takes $O(i)$ time (for doing a for-loop). Note that unlike the previous dynamic programming algorithms we saw, now the runtime of each subproblem is different. However, the principle is exactly as before (similar to the case of recurrences): we simply need to add up all the time spent for all subproblems. As such, the total time spent by the algorithm is (we replace $O(i)$ by $c \cdot i$ for some constant c):

$$\sum_{i=0}^{n} c \cdot i = c \cdot \sum_{i=1}^{n} i = c \cdot \frac{n \cdot (n+1)}{2} = O(n^2).$$

So, the runtime of this algorithm is $O(n^2)$ (there is an additional $O(n)$ time at the end to compute the maximum value but that is negligible compared to the $O(n^2)$ term).

Bottom-up dynamic programming: We have to first determine the evaluation order. It is quite easy here because each $LIS(i)$ is updated only from $LIS(i')$ for $i' < i$ and so we simply need to evaluate the subproblems in the increasing order of i.

```
DynamicLIS($A[1:n]$):
  1. Let $T[0:n]$ be an array initialized by 0 originally (to take care of the base case).
  2. For $i = 1$ to $n$:
        Let $T[i] = 1$ and for $j = 1$ to $i - 1$:
            If $A[j] < A[i]$, let $T[i] = \max\{T[i], T[j] + 1\}$.
  3. Let $m \leftarrow 0$. For $j = 1$ to $n$:
            Let $m \leftarrow \max\{m, T[j]\}$.
  4. Return $m$.
```

Again the correctness follows from the correctness of the recursive formula $LIS(\cdot)$ and the in the evaluation order we picked, each entry is updated from the cells which are already computed correctly. It is easy to see that the runtime of this algorithm is $O(n^2)$.