Dynamic Programming 4: Longest Common Subsequence

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
A string $s$ is a **subsequence** of another string $t$ if either $s = t$ or we can convert $t$ to $s$ by deleting characters.

**Example:** $t = ABCDEF$

The following are subsequences of $t$: ABD, ACDF, and ABCDEF. The following are not: ACB, ACG, and BDFE.
The Longest Common Subsequence Problem

Given two strings $x$ and $y$, find a common subsequence $z$ of $x$ and $y$ with the maximum length.

We will refer to $z$ as a **longest common subsequence** (LCS) of $x$ and $y$.

**Example:** If $x = ABCBDAB$ and $y = BDCABA$, then BCBA is an LCS of $x$ and $y$, so is BCAB.

If $x = \emptyset$ (empty string) and $y = BDCABA$, their (only) LCS is $\emptyset$. 
The key to solving the problem is to identify its underlying recursive structure.

Specifically, how the original problem is related to subproblems.

The recursive structure will then imply a dyn. programming algorithm.
\( n = \) the length of \( x \); \( m = \) the length of \( y \)

**Theorem:** Let \( z \) be any LCS of \( x \) and \( y \), and \( k \) the length of \( z \). Then:

1. If \( x[n] = y[m] \)
   then \( z[k] = x[n] \) (hence, also \( = y[m] \)) and
   \( z[1 : k - 1] \) is an LCS of \( x[1 : n - 1] \) and \( y[1 : m - 1] \).

2. If \( x[n] \neq y[n] \), then \textbf{at least} one of the following holds:
   - \( z \) is an LCS of \( x[1 : n - 1] \) and \( y \)
   - \( z \) is an LCS of \( x \) and \( y[1 : m - 1] \).

This is the recursive structure of the problem.
Example:

- Suppose $x = \text{BCBDA}$ and $y = \text{BDCABA}$, which have an LCS $z = \text{BCBA}$. By Statement 1 (of the theorem), BCB must be an LCS of BCBD and BDCAB.

- Suppose $x = \text{ABCBDAB}$ and $y = \text{BDCABA}$, which have an LCS $z = \text{BCBA}$. By Statement 2, at least one of the following is true:
  - BCBA is an LCS of ABCBDAB and BDCABA;
  - BCBA is an LCS of ABCBDAB and BDCAB.
Proof of Statement 1:

We first prove $z[k] = x[n]$. Suppose that this is not true. Then, $z$ must be a common subsequence of $x[1 : n - 1]$ and $y[1 : m - 1]$. But then $z' \circ x[n]$ is a length-$(k + 1)$ common subsequence of $x$ and $y$, contradicting the fact that $z$ is an LCS of $x$ and $y$.

Next, we prove $z[1 : k - 1]$ is an LCS of $x[1 : n - 1]$ and $y[1 : m - 1]$. Suppose that this is not true. Thus, $x[1 : n - 1]$ and $y[1 : m - 1]$ have an LCS $z'$ with length at least $k$. However, $z' \circ x[n]$ will be a length-$(k + 1)$ common subsequence of $x$ and $y$, contradicting the definition of $z$. □

Remark: $\circ$ means string concatenation. For example, $ABC \circ DEF = ABCDEF$. 
**Proof of Statement 2:**

Because $x[n] \neq y[m]$, at least one of the following is false:

- $z[k] = x[n]$
- $z[k] = y[m]$.

Consider first $z[k] \neq x[n]$. We argue that $z$ must be an LCS of $x[1 : n - 1]$ and $y$. First, $z$ must be a common subsequence of $x[1 : n - 1]$ and $y$ (think: how is this related to $z[k] \neq x[n]$)? Assume, on the contrary, that $z$ is not their LCS. Thus, $x[1 : n - 1]$ and $y$ have an LCS $z'$ of length at least $k + 1$. This means that $x$ and $y$ have a common subsequence of length $k + 1$, contradicting the fact that $z$ is an LCS of $x$ and $y$.

A symmetric argument proves the statement when $z[k] \neq y[m]$. 

\[\Box\]
Define $x[1 : 0] = y[1 : 0] = \emptyset$ (empty string).

For any $i \in [0, n]$ and $j \in [0, m]$, define

$$opt(i, j) = \text{the LCS length of } x[1 : i] \text{ and } y[1 : j].$$

Note that $opt(n, m)$ is the LCS length of $x$ and $y$.

The theorem tells us

$$opt(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
opt(i - 1, j - 1) + 1 & \text{if } i, j > 0 \text{ and } x[i] = y[j] \\
\max\{opt(i, j - 1), opt(i - 1, j)\} & \text{if } i, j > 0 \text{ and } x[i] \neq y[j]
\end{cases}$$

We can compute $opt(n, m)$ in $O(nm)$ time by dynamic programming (last lecture).
Wait! We still need to **generate** an LCS of $x$ and $y$.

This can be done by slightly modifying the dynamic programming algorithm without increasing the time complexity. Details are left as a regular exercise.