CS3000: Algorithms & Data — Summer '23 — Laney Strange

Longest Common Subsequence

For a given sequence S, we can say that valid *subsequence* is just S with 0 or more elements removed. If $S =< N, O, R, T, H, E, A, S, T, E, R, N >$, then valid subsequences include $< N, T, H >$, $< T, H, E, E >$, $< O, R, T, H, E, A, S, T >$, and $< O, E, S, E >$. A *common subsequence* of two sequences X and Y is a subsequence of X and a subsequence of Y.

The Longest Common Subsequence (LCS) problem is an optimization problem. Given two sequences X and Y, our goal is to find a commmon subsequence of both X and Y, maxmimizing its length. LCS is a good fit for dynamic programming, because:

- It exhibits optimal substructure, because an LCS of two sequences contains within it an LCS of prefixes of the two sequences.
- A recursive solution would work, but re-solves smaller subproblems.

Below is pseudocode for solving the LCS problem with bottom-up dynamic programming. We'll go over them together in class.

The function LCS-Length generates two tables, b and c, which build a bottom-up DP solution for the LCS problem. The input to the function is our two sequences X and Y, and their respective lengths, m and n.

```
LCS-Length(X, Y, m, n)
let b[1 : m, 1 : n] and c[0 : m, 0 : n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else
            c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

The c and b tables give us solutions to the smaller subproblems, but to reconstruct the actual LCS, we need a helper function:

```
Print-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    Print-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    Print-LCS(b, X, i − 1, j)
else
    Print-LCS(b, X, i, j − 1)
```

LCS Tables

Using the code above, we make a c table, which stores the lengths of all the LCS subproblems. We also make a b table, which stores left, up, and "northwest" arrows so that we can reconstruct the value of the LCS.

Here's what the c and b tables would look like if we have $X =< A, B, C, D >$ and $Y =< A, E, B, D, H >$.

c table

	y_j	A	E	B	D	H
x_i	0	0	0	0	0	0
A	0	1	1	1	1	1
B	0	1	1	2	2	2
C	0	1	1	2	2	2
D	0	1	1	2	3	3

Here are a few things we can see from reading the c table:

- The length of the LCS of AEBDH, ABCD is 3.
- The length of the LCS of A, A is 1.
- The length of the LCS of A, AEBDH is 1.
- The length of the LCS of ABC,AEB is 2.

b table

	A	E	B	D	H
A	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
B	$\uparrow$	$\uparrow$	$\nwarrow$	$\leftarrow$	$\leftarrow$
C	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
D	$\uparrow$	$\uparrow$	$\uparrow$	$\nwarrow$	$\leftarrow$

Here are a few things we can see from reading the b table:

- D is included in the LCS (because we see a northwest arrow where both X and Y have element D)
- B is included in the LCS (because we see a northwest arrow where both X and Y have element B)
- A is included in the LCS (because we see a northwest arrow where both X and Y have element A)

We typeset the two LCS pseudocode procedures with the following LaTeX:

```
\begin{codebox}
\Procname{$\proc{LCS-Length}(X, Y, m, n)$}
\li let $b[1:m, 1:n]$ and $c[0:m, 0:n]$ be new tables
\li \For $i \gets 1 \To m$
\Do
\li $c[i, 0] = 0$
\End
\li \For $j \gets 0 \To n$
\Do
\li $c[0, j] \gets 0$
\End
\li \For $i \gets 1 \To m$
\Do
\li \For $j \gets 1 \To n$
\Do
\li \If $x_i == y_j$
\Do
\li $c[i, j] = c[i-1, j-1] + 1$
\li $b[i, j] = "\nwarrow"$
\li \ElseIf $c[i-1, j] \ge c[i, j-1]$
\Do
\li $c[i, j] = c[i-1, j]$
\li $b[i, j] = "\uparrow"$
\li \Else
\li $c[i, j] = c[i, j-1]$
\li $b[i, j] = "\leftarrow"$
\End
\End
\End
\li \Return $c$ and $b$
\end{codebox}

\begin{codebox}
\Procname{$\proc{Print-LCS}(b, X, i, j)$}
\li \If $i == 0$ or $j == 0$
\Do
\li \Return
\End
\li \If $b[i, j] == "\nwarrow"$
\Do
\li $\proc{Print-LCS}(b, X, i-1, j-1)$
\li print $x_i$
\li \ElseIf $b[i, j] == "\uparrow"$
\Do
\li $\proc{Print-LCS}(b, X, i-1, j)$
\li \Else
```

```
\li $\proc{Print-LCS}(b, X, i, j-1)$
\End
\end{codebox}
```