**Ruolin Li**
**31764160**

# CPSC 320 2020S2: Tutorial 9

## 1 Longest Common Subsequence, Completed

Recall the LLCS problem from the worksheet: Given two strings $A[1..n]$ and $B[1..m]$, find the length of the longest string whose letters appear in order (but not necessarily consecutively) in both `A` and `B`.

In class we developed a recurrence for the LLCS:

$$\text{LLCS}(A[1..n], B[1..m]) = \begin{cases} 0, & \text{if } n = 0 \text{ or } m = 0, \\ \text{LLCS}(A[1..n-1], B[1..m-1]) + 1, & \text{if } n, m > 0 \text{ and } A[n] = B[m], \\ \max\{\text{LLCS}(A[1..n-1], B[1..m]), \text{LLCS}(A[1..n], B[1..m-1])\}, & \text{otherwise.} \end{cases}$$

We also developed a memoization algorithm:

**procedure** MEMO-LLCS($A[1..n], B[1..m]$)
  create a 2-dimensional array Soln$[0..n][0..m]$
  initialize all elements of Soln to null
  MEMO-HELPER($A[1..n]B[1..m]$)

**procedure** MEMO-HELPER($A[1..i], B[1..j]$)
  **if** Soln$[i][j]$ is null **then**
    **if** $i == 0$ or $j == 0$ **then**
      Soln$[i][j] \leftarrow 0$
    **else if** $A[i] == B[j]$ **then**
      Soln$[i][j] \leftarrow$ MEMO-HELPER($A[1..i-1], B[1..j-1]$) + 1
    **else**
      Soln$[i][j] \leftarrow$
        max{MEMO-HELPER(A[1..i-1],B[1..j]), MEMO-HELPER(A[1..i],B[1..j-1])}
      **return** Soln$[i][j]$

1. Give a dynamic programming solution that produces the same Soln table as the memoized solution.
2. What is the runtime of MEMO-LLCS and DP-LLCS, as a function of $n$ and $m$? How much memory do they use?
3. If we only want the **length** of the LCS of `A` and `B` with lengths $n$ and $m$, where $n \leq m$, explain how we can "get away" with using only $O(n)$ memory in the dynamic programming solution.

1. Procedure DP-LLCS

```
Initialize Soln[0..n][0..m]   ← memory bound nm
for i from 0 to n do
    Soln[i][0]=0
for j from 0 to m do
    Soln[0][j]=0

for i from 1 to n do              ← Iteration nm times
  for j from 1 to m do
    if A[i]=B[j]
       Soln[i][j]= 1+ Soln[i-1][j-1]
    else
       Soln[i][j]=max{ Soln[i-1][j], Soln[i][j-1] }.
return Soln[n][m].
```

runtime: $O(nm)$

memory: $O(nm)$

## 2 Edit distance

Let $X = x_1 x_2 ... x_n$ and $Y = y_1 y_2 ... y_m$ be strings over some fixed alphabet $\Sigma$.

- An *insertion* of "a" in $X$ at position $i$ yields the string $x_1 x_2 ... x_i a x_{i+1} ... x_n$ (of length $n+1$).
- A *deletion* at position $i$ of $X$ yields $x_1 ... x_{i-1} x_{i+1} ... x_n$ (of length $n-1$).

For example, if $X =$ "tree" then an insertion of "h" at positions 0 or 1 respectively yield "htree" or "three" respectively, and a deletion at position 2 of $X$ yields "tee".

The *edit distance* between $X$ and $Y$ is the minimum number of insertions and deletions to get string $Y$ from string $X$ (or vice versa).

1. What is the edit distance between $X =$ "fence" and $Y =$ "wicked"?
2. Let $\mathrm{ED}(n, m)$ be the edit distance between strings $X[1..n]$ and $Y[1..m]$. Express $\mathrm{ED}(n, m)$ as a recurrence. Make sure to include appropriate base cases. Hint: think about the longest common subsequence problem discussed in class.
3. We already have efficient algorithms to find the LLCS of two strings. Can we use that algorithm to find the edit distance between $X$ and $Y$? (That is, can you find a *reduction* from the Edit Distance problem to the LLCS problem? Recall that in the worksheet on breadth first search, we showed a reduction from Shortest Paths to Breadth First Search.)

2. ① X="fence" Y="wicked"

delete 'f', 'n'
delete 'e'
add 'w', 'i', 'k', 'd'

} 7 edits

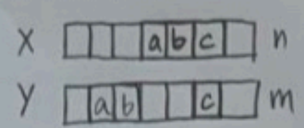
| Step: | | |
|---|---|---|
| 0 | X="fence" | fence |
| 1 del f, i=1 | | ence |
| 2 ins i, i=0 | | ience |
| 3 ins w, i=0 | | wience |
| 4 del i=3 | | wi~~n~~ce |
| 5 del i=3 | | wice |
| 6 ins k | | wicke |
| 7 ins d | | wicked |

② When either string is empty. It's the base case

$$ED(m,n)=\begin{cases} m, & n=0 \\ n, & m=0 \end{cases}$$

$$ED(m,n)=\begin{cases} ED(m-1,n-1), & X[m]=Y[n] \\ \min\begin{Bmatrix} ED(m-1,n) \\ ED(m,n-1) \end{Bmatrix}+1, & \text{else} \end{cases}$$

③ Reductions




ED = n+m−2LLCS

Algo: Given instance X, Y, compute LLCS(X,Y) and subtract from n+m.