Algorithms Analysis and Design

Chapter 7

Dynamic Programming
Part 3

Longest Common Subsequence

Longest Common Subsequence (LCS)

- In biological applications we often want to , we often want to compare the DNA of two (or more) different organisms.
- A strand of DNA consists of a string of molecules called bases, where the possible bases are adenine quinine cytosine and , quanine, cytosine, and thymine (A, C, G, T).

Longest Common Subsequence (LCS)

Similarity can be defined in different ways:

- Two DNA strands are similar if one is a substring of the other.
- Two strands are similar if the number of changes need to turn one into the other is small.
- There is a thirds strand S3 in which the bases in S3 appear in each of S1 and S2; these bases must appear in the same order, but not necessarily consecutively. The longer the strand S3 we can find, the more similar S1 and S2 are.

Example of LCS

LCS for S_1 and S_2:

S_1=ACC**G**G**TCG**AG**T**G**CG**C**G**G**AAGCCG**G**CCGAA**

S_2=**GTCGT**T**CGGAA**T**GCCG**TTG**C**T**C**T**G**T**AA**A

S_3=**GTCGTCGGAAGCCGCCGAA**

Example

- Longest Common Subsequence
  - Given two sequences x[1..m] and y[1..n], find a longest subsequence common to them both.

X = ⟨A, B, C, B, D, A, B⟩ X = ⟨A, B, C, B, D, A, B⟩

Y = ⟨B, D, C, A, B, A⟩ Y = ⟨B, D, C, A, B, A⟩

- ⟨B, C, B, A⟩ and ⟨B, D, A, B⟩ are longest common subsequences of X and Y (length = 4)

- ⟨B, C, A⟩, however is not a LCS of X and Y

LCS Brute Force Solution

- Enumerate all subsequences of X[1 ….. m] and check each subsequence to see if it is also a subsequence of Y[1….n], keeping track of the longest subsequence found.
- Analysis:
- Each subsequence of X corresponds to a subset of the indices {1, 2, …, m} of X. ($X = X_1, X_2, X_3, \ldots\ldots X_m$)
- There are 2^m subsequences of X. (each bit-vector of length m determines a distinct subsequence of X)
- Checking = O(n) time per subsequence
- Running time = O(n 2^m) → Exponential Time!

A recursive Solution

Case 1: $x_i = y_j$

e.g.: $X = \langle A, B, D, E \rangle$

$Y = \langle Z, B, E \rangle$

$$c[i, j] = c[i - 1, j - 1] + 1$$

- Append $x_i = y_j$ to the LCS of X_{i-1} and Y_{j-1}
- Must find a LCS of X_{i-1} and Y_{j-1} $\Rightarrow$ optimal solution to a problem includes optimal solutions to sub problems

Define $\boldsymbol{C[i,j] = LCS\ (\ X[1 \ldots \ldots i)\ , Y[1 \ldots .\,.\, j]\ )}$

Then $\boldsymbol{c[m,n] = LCS\ (X, Y)}$

A recursive Solution

Case 2: $x_i \neq y_j$

e.g.: $X = \langle A, B, D, G \rangle$

$Y = \langle Z, B, D \rangle$

$$c[i, j] = \max \{ c[i - 1, j], c[i, j-1] \}$$

- Must solve two problems
  - find a LCS of X_{i-1} and Y_j: $X_{i-1} = \langle A, B, D \rangle$ and $Y_j = \langle Z, B, D \rangle$
  - find a LCS of X_i and Y_{j-1}: $X_i = \langle A, B, D, G \rangle$ and $Y_j = \langle Z, B \rangle$
- Optimal solution to a problem includes optimal solutions to subproblems

Optimal substructure

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

If $z = \mathrm{LCS}(x, y)$, then any prefix of z is an LCS of a prefix of x and a prefix of y.

Recursive algorithm for LCS

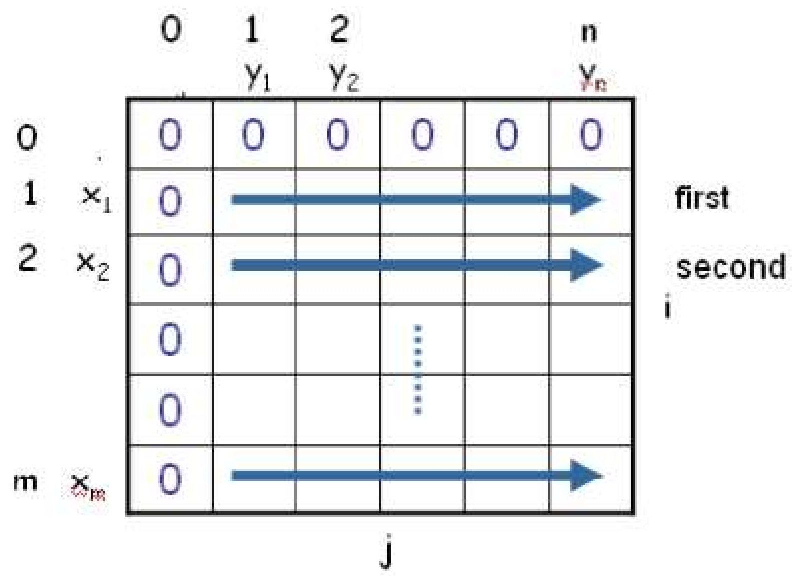
RECURSIVE for LCS

```
LCS(x,y,i,j)
  if x[i] = y[j]
     then c[i,j] ← LCS(x,y,i-1,j-1)+1
     else c[i,j] ← max{LCS(x,y,i-1,j),
                        LCS(x,y,i,j-1)}
```

- **Worst-case:** $x[i] \neq y[j]$, in which case the algorithm evaluates two subproblems, each with only one parameter decremented.

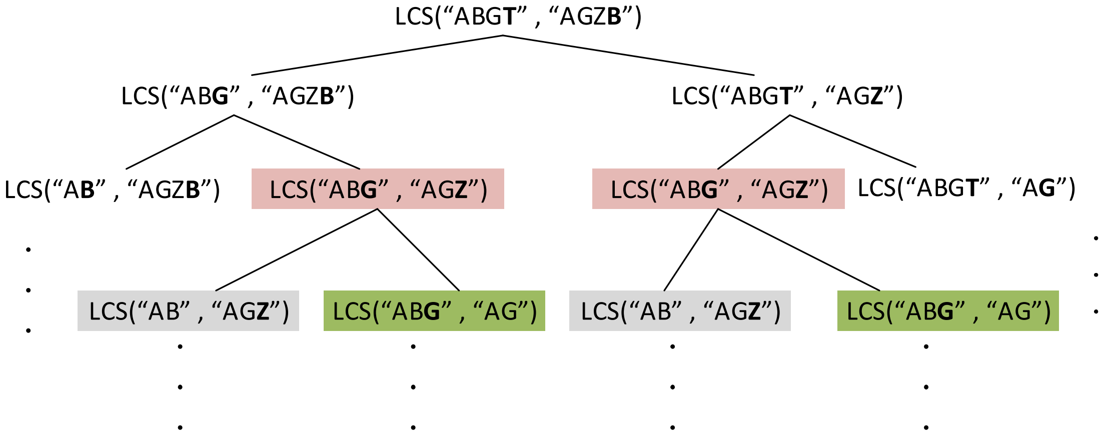
Recursive Formulation

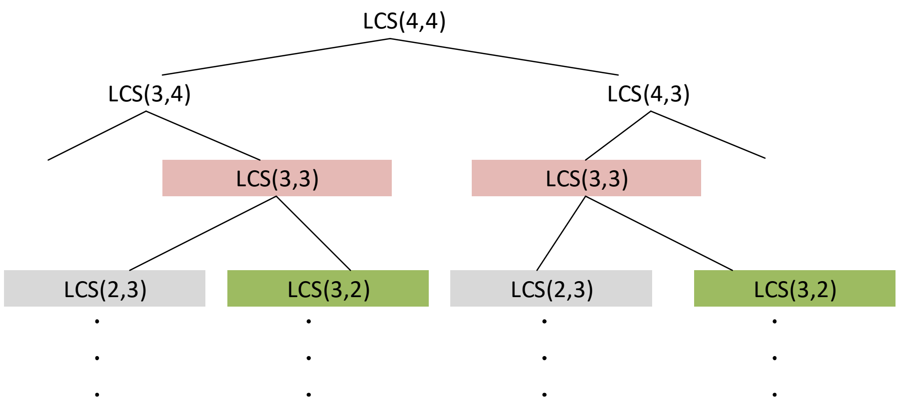
$$
\begin{cases}
0 & if\ i = 0\ or\ j = 0 \\
c[i-1\ , j-1] + \mathbf{1} & if\ xi = yj \\
\max(\ c[i, j-1], c[i-1\ , j] & if\ \ \ xi \neq y_j
\end{cases}
$$



Recursive Tree example

- X = "ABGT" Y="AGZB" m = 4 , n = 4 , Find LCS (X,Y)
- LCS ("ABGT" , AGZB) → $LCS\ (4\ ,4)$





- Lots of repeated subproblems
- Instead of recomputing, store in a table.

Overlapping Subproblems

Overlapping subproblems

A recursive solution contains a "small" number of distinct subproblems repeated many times.

The number of distinct LCS subproblems for two strings m and n is only mn.

Bottom-Up DP algorithms

COMPUTING the LENGTH of LCS

```
LCS-LENGTH(X,Y)
  m ← length[X]
  n ← length[Y]
  for i ← 1 to m
      do c[i,0] ← 0
  for j ← 0 to n
      do c[0,j] ← 0
  for i ← 1 to m
      do for j ← 1 to n
             do if x[i] = y[j]
                    then c[i,j] ← c[i-1,j-1]+1
                         b[i,j] ←"↖"
                    else if c[i-1,j] ≥ c[i,j-1]
                            then 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
```

For constructing LCS from the table

Example

Complete the DP table to compute the length of the Longest Common Subsequence (LCS) between two strings **X = ABCBDAB** and **Y = BDCABA**

	j	0	1	2	3	4	5	6
i		y_j	***B***	***D***	***C***	***A***	***B***	***A***
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	***A***	**0**						
2	***B***	**0**						
3	***C***	**0**						
4	***B***	**0**						
5	***D***	**0**						
6	***A***	**0**						
7	***B***	**0**						

First Optimal-LCS initializes row 0 and column 0

Example

Complete the DP table to compute the length of the Longest Common Subsequence (LCS) between two strings **X = ABCBDAB** and **Y = BDCABA**

	j	0	1	2	3	4	5	6
i		y_j	*B*	*D*	*C*	*A*	*B*	*A*
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	*A*	**0**	**^ 0**	**^ 0**	**^ 0**	**↖ 1**	**< 1**	**1**
2	*B*	**0**	**↖ 1**	**< 1**	**< 1**	**^ 1**	**↖ 2**	**< 2**
3	*C*	**0**	**^ 1**	**^ 1**	**↖ 2**			
4	*B*	**0**						
5	*D*	**0**						
6	*A*	**0**						
7	*B*	**0**						

Next each c[i, j] is computed, row by row, starting at c[1,1].
If $x_i == y_j$ then c[i, j] = c[i-1, j-1]+1 and b[i, j] = ↖

Example

Complete the DP table to compute the length of the Longest Common Subsequence (LCS) between two strings **X = ABCBDAB** and **Y = BDCABA**

	j	0	1	2	3	4	5	6
i		y_j	*B*	*D*	*C*	*A*	*B*	*A*
0	x_i	0	0	0	0	0	0	0
1	*A*	0	^ 0	^ 0	^ 0	↖ 1	< 1	1
2	*B*	0	↖ 1	< 1	< 1	^ 1	↖ 2	< 2
3	*C*	0	^ 1	^ 1	↖ 2	< 2		
4	*B*	0						
5	*D*	0						
6	*A*	0						
7	*B*	0						

If x_i <> y_j then c[i, j] = max(c[i-1, j], c[i, j-1]) and b[i, j] points to the larger value

Example

Complete the DP table to compute the length of the Longest Common Subsequence (LCS) between two strings **X = ABCBDAB** and **Y = BDCABA**

	j	0	1	2	3	4	5	6
i		y_j	*B*	*D*	*C*	*A*	*B*	*A*
0	x_i	0	0	0	0	0	0	0
1	*A*	0	^ 0	^ 0	^ 0	↖ 1	< 1	1
2	*B*	0	↖ 1	< 1	< 1	^ 1	↖ 2	< 2
3	*C*	0	^ 1	^ 1	↖ 2	< 2	^ 2	
4	*B*	0						
5	*D*	0						
6	*A*	0						
7	*B*	0						

if c[i-1, j] == c[i, j-1]
then b[i,j] points up

Example

Complete the DP table to compute the length of the Longest Common Subsequence (LCS) between two strings **X = ABCBDAB** and **Y = BDCABA**

	j	0	1	2	3	4	5	6
i		y_j	*B*	*D*	*C*	*A*	*B*	*A*
0	x_i	0	0	0	0	0	0	0
1	*A*	0	^ 0	^ 0	^ 0	↖ 1	< 1	↖ 1
2	*B*	0	↖ 1	< 1	< 1	^ 1	↖ 2	< 2
3	*C*	0	^ 1	^ 1	↖ 2	< 2	^ 2	^ 2
4	*B*	0	↖ 1	^ 1	^ 2	^ 2	↖ 3	< 3
5	*D*	0	^ 1	↖ 2	^ 2	^ 2	^ 3	^ 3
6	*A*	0	^ 1	^ 2	^ 2	↖ 3	^ 3	↖ 4
7	*B*	0	↖ 1	^ 2	^ 2	^ 3	↖ 4	^ 4

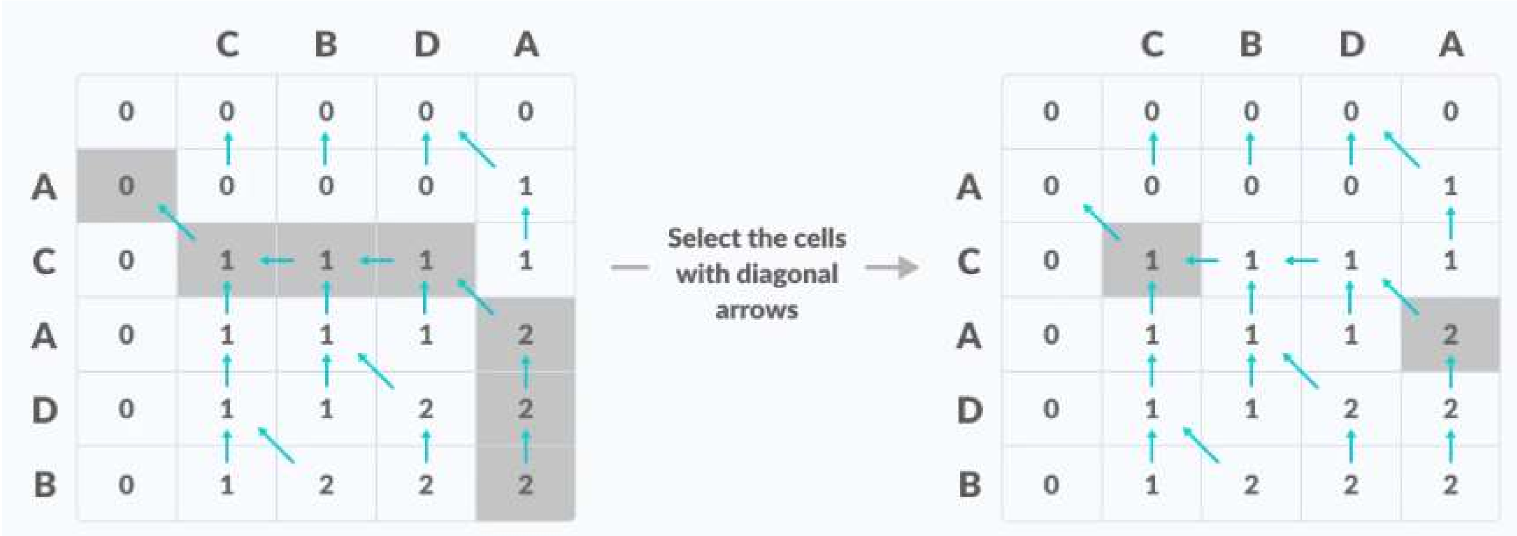
Time = $\Theta(mn)$.

LCS: B C B A

To construct the LCS, start in the bottom right-hand corner and follow the arrows. A ↖ indicates a matching character.

Exercise

Use DP to find the LCS of the two sequences X = ACADP and Y = CBDA



LCS = CA

Exercise

Write a top-down memoization algorithm for LCS.

Hint: You need to modify the recursive version for LCS.