CSCI B03 Lecture Notes 03/09/2010

LCS - Least Common Subsequence Problem

We are given two strings: ① $x = (x_1, \ldots x_n)$
② $y = (y_1, \ldots y_m)$

$Q(x,y) \triangleq$ Length of LCS of x and y

$$Q(x,y) = \begin{cases} 1 \text{ if } x_1 = y_1 \\ 0 \text{ otherwise} \end{cases} \quad m = n = 1,\ |x| = |y| = 1 \quad \text{Basis}$$

$$\begin{cases} 1 \text{ if } y_1 \in (x_1 \cdots x_n) \\ 0 \text{ otherwise} \end{cases} \quad \text{if } m = 1$$

$$\begin{cases} 1 \text{ if } x_1 \in (x_1 \cdots y_m) \\ 0 \text{ otherwise} \end{cases} \quad \text{if } n = 1$$

Dynamic Programming

$$Q([x_2 \cdots x_n], [y_2 \cdots y_m]) + 1 \quad \text{if } x_1 = y_1,\ m > 1,\ n > 1$$

$$\max\left(\begin{matrix} Q[x_1 \cdots x_n], [y_2 \cdots y_m] \\ Q[x_2 \cdots x_n], [y_1 \cdots y_m] \end{matrix}\right) \quad \text{if } x_1 \neq y_1$$

Substructure optimality

Dynamic Programming Q Table (is backwards from how book does it)

Q	$Q[x_1 \cdots x_n], [y_1 \cdots y_m]$	$[1n][2m]$	$[1n][3m]$	$\cdots$	$[1n][nm]$
	$[2n][1m]$	$[2n][2m]$			✓
	$[3n][1m]$	$[3n][2m]$	$[3n][3m]$		✓
					✓
					✓
					✓
					✓
					✓
					✓
	$[n,n][1,m]$	✓ ✓ ✓ ✓ ✓ ✓			$[nn][nm]$

✓ means it's trivial to Comput

03/09/2010

Example

		A	B	B	A	B	B	A	A
A	4	3	3	3	3	2	2	1	
A	4	3	3	3	2	2	2	1	
A	4	3	3	3	2	2	2	1	
A	3	3	3	3	2	1	1	1	
B	2	2	2	2	2	1	0	0	
B	1	1	1	1	1	1	0	0	

[8,8] [1,6] AAAABB
[2,6] AAABB
AABB
ABB
BB
[8,8] [6,6] B
A

[AA] = max([AA] [B] / [A] [BB])

Q[(ABBABBAA), (AAAABB)] = 4 ✓

Rules: (A) - If you go up diagonally, you add one
(B) - If you go vertically and horizontally, you don't add one

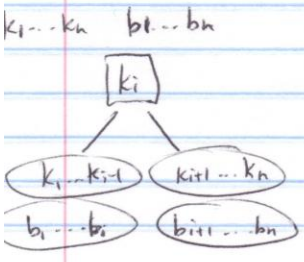
(A) - If you're starting with the same letter, go up diagonally
(B) - If you're starting with different letters, take the max of vertical and horizontal neighbors

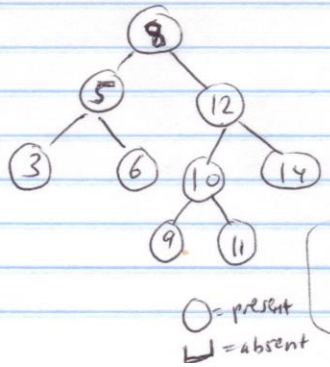
- To determine LCS, look at every time you went diagonal and incremented by 1. Each of those is part of the LCS when you trace it back.

So LCS = AABB

BINARY SEARCH TREES

→ If bigger, go right, If smaller, go left
→ Then recurse





- Optimizing is like Huffman Code
- Both can be solved by dynamic programming

1 2 3 4 5 6 7 8 9 10 11 12 13 14