Longest common subsequence

Given two sequences $x = [1, \dots, m]$ and $y = [1, \dots, n]$ determine (a) longest common subsequence

$LCS(x, y)$ ⤳ functional notation even though it's not a function

Ex.

x: A B C B D A B
y: B D C A B A
→
BDAB
BCAB
BCBA

Brute force algorithm

1. Check every subsequence of $x[1 \dots m]$ to see if it is also a sequence of $y[1 \dots n]$

Analysis: 1. to check if a sequence is a subsequence of y
↳ $O(n)$ → you have to get the first char matching and then you scan

2. How many subsequences in x?
↳ 2^m

$\Rightarrow O(n \cdot 2^m)$

Simplification Step

1. Look at the length of $LCS(x, y)$ ⤳ $c[x, y]$
2. Extend 1. to return the actual LCS

Strategy: Consider only prefixes of x and y and we are going to show how we can express the length of the lcs of prefixes in terms of each other.

Define: $c[i, j] = |LCS(x[1 \dots i], y[1 \dots j])|$

↳ we need to calculate $c[i, j]$ (or c_{ij}) $\forall i, j \in n, m$

How do we solve the problem of LCS(x,y) if we have c_{ij} $\forall i,j$?

↳ $c[m,n] = |LCS(x,y)|$

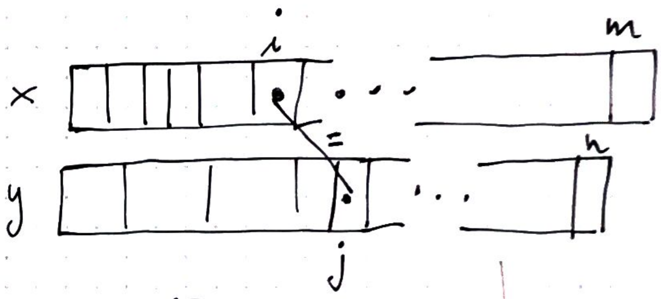
We want to express the general $c[m,n]$ in terms of other $c[i,j]$

<u>Theorem</u>

$$c[i,j] = \begin{cases} 0 & \text{, if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{, if } x[i]=y[j] \\ \max\{c[i,j-1], c[i-1,j]\} & \text{, otherwise} \end{cases}$$

<u>Proof.</u>

case $x[i]=y[j]$



Let $z[1\ldots k] = LCS(x[1,\ldots,i], y[1,\ldots,j])$ where $c[i,j]=k$

Then, $z[k] = x[i] = y[j]$ because if z does not include $x[i]$ or $y[j]$ then we can add this char to z and make it longer because $x[i]=y[j]$ → if I can add an element to $z' = z + x[i]$ then z could not be a LCS...

Thus $z[1,\ldots,k-1]$ is a CS of $x[1\ldots i-1]$ and $y[1\ldots j-1]$

CLAIM: $z[1,\ldots,k-1]$ is a LCS of $x[1\ldots i-1]$ and $y[1\ldots j-1]$

Proof of the claim: Ab absurdo suppose that w is a longer CS than z ⇒ $|w| > k-1$

We use the CUT & PASTE argument

$w \| z[k]$ → is certainly a CS of $x[1\ldots i]$ and $y[1\ldots j]$ and has length greater than k

↳ string concatenation

↳ CONTRADICTION

Thus, $c[i-1, j-1] = k-1$, which implies that

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

The other case can be proven in the same way. $\blacksquare$

We have an <u>OPTIMAL SUBSTRUCTURE</u> because the optimal solution of the subproblems compose the solution of the whole problem.

↳ If $z = LCS(x, y)$ then any prefix of z is a LCS of a prefix of x and a prefix of y.

— o —

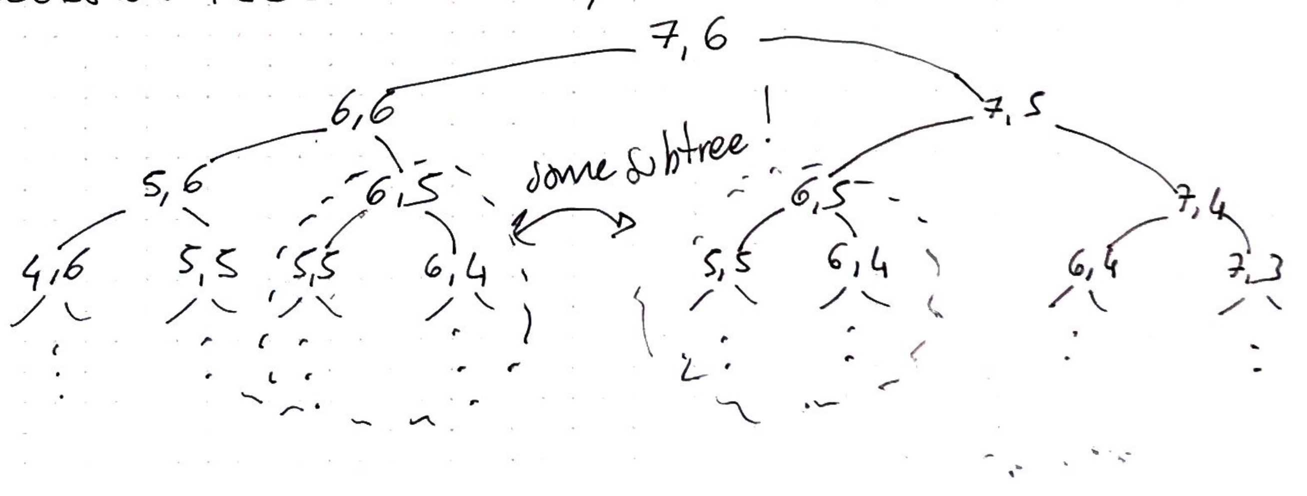
Now we have a strategy to compute LCS

<u>RECURSIVE algorithm</u>

```
LCS(x, y, i, j)
// we ignore the base cases i=0, j=0
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))
return c[i,j]
```

<u>worst-case analysis</u>: the second clause is problematic because there are 2 LCS calls

RECURSION TREE for $n = 7$, $m = 6$



The height of the tree is $m+n$ so being a binary tree each level brings 2^i work $\rightsquigarrow O(2^{m+n})$

$\hookrightarrow$ i.e. LCS contains $m \cdot n$ distinct subproblems.

$$T(n,m) = \begin{cases} T(n-1, m-1) + O(1) & \text{if } x[n] = y[m] \\ \underbrace{T(n-1, m) + T(n, m-1) + O(1)}_{\text{worst-case}}, & \text{otherwise} \end{cases}$$

$T(n,m) = 2^{n-1} T(0,m) + \dots + \dots$ if $m < n$ (longest branch of height n)

$T(n,m) = 2^{m-1} T(0,n) + \dots$ if $n < m$

BOTTOM-UP. MEMOIZED ALGORITHM

```
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 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, b
```

Example

i \ j		0	1	2	3	4	5	6 = n
		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↖
m = 7	B	0	1↖	2↑	2↑	3↑	4↖	4↑

⇒ BCBA

```
PRINT-LCS(b, x, i, j)
if i == 0 or j == 0
    return null
if b[i,j] == "↖" then
    PRINT-LCS(b, x, i-1, j-1)
    print x_i
elseif(b[i,j] = "↑") then
    PRINT-LCS(b, x, i-1, j) // we go up
else PRINT-LCS(b, x, i, j-1) // we go left
```