Lec15 动态规划、最长公共子序列

Dynamic Programming

"any tabular method for accomplishing something"

a design technique, like Divide&Conquer, a way of solving a class of problems rather than a particular algorithm or something

Example:

Longest Common Subsequence Problem (LCS)

given two sequences $x[1\ldots m]$ and $y[1\ldots n]$, to find a longest sequence which is common to both

x: A B C B D A B

y: B D C A B A

LCS's: BDAB, BCAB, BCBA

brute-force algorithm:

to check every subsequence of $x[1\ldots m]$, to see if it is also a subsequence of $y[1\ldots n]$

there are 2^m subsequences of x, and each check takes $O(n)$, so running time is $O(n\cdot 2^m)$

consider the more simple problem!

Simplification:

1. focus on the length of the longest common subsequence of x and y,
2. extend the algorithm to find the longest common subsequence itself

Notation: $|s|$ denotes the length of sequence s

Strategy: consider prefixes of x and y

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

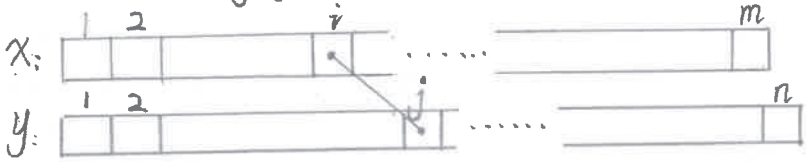
then $c[m,n] = |LCS(x,y)|$

Theorem:

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

Proof:

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 $z[k]=x[i]\ (=y[j])$, or else z could be extended by appending on $x[i]$

thus $z[k-1]$ is a common sequence of $x[1\ldots i-1]$ and $y[1\ldots j-1]$

claim: $z[1\ldots k-1] = LCS(x[1\ldots i-1], y[1\ldots j-1])$

suppose w is a longer common sequence. that is $|w|>k-1$

cut & paste: $w+z[k]$ is a common sequence of $x[1\ldots i]$ and $y[1\ldots j]$, with its length $>k$

contradiction!

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

case $x[i]\neq y[j]$

<similar>

Q.E.D.

Dynamic Programming Hallmarks #1 "特征"

optimal substructure

an optimal solution to a problem (instance) contains optional solutions to subproblems

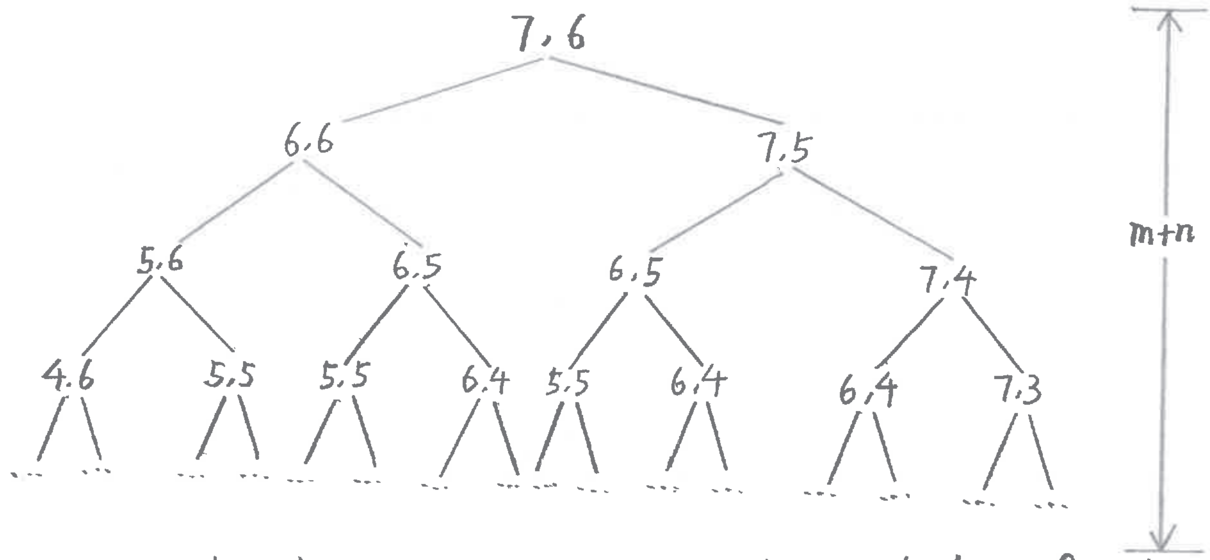
in LCS example, if $z=LCS(x,y)$, then any prefix of z is a longest common subsequence of a prefix of x and a prefix of y

Recursive Algorithm for LCS:

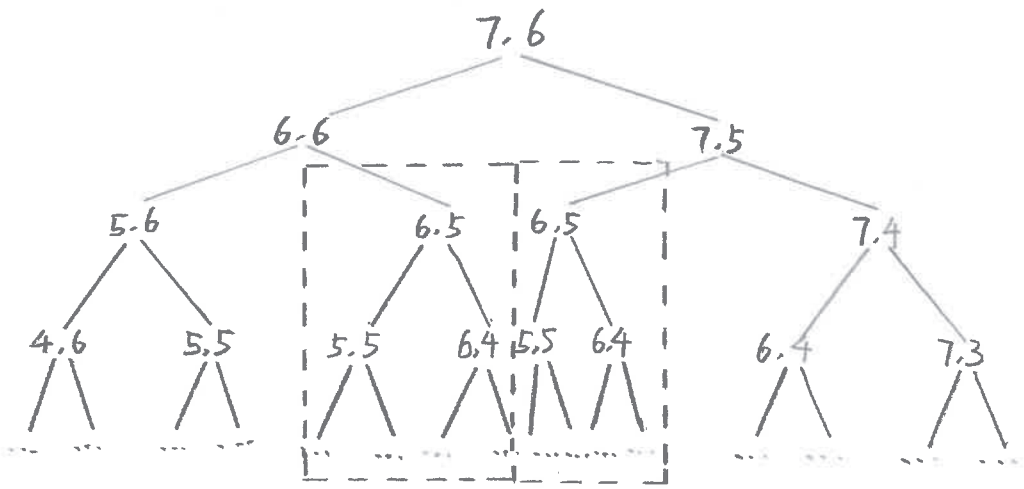
```
LCS(x, y, i, j): // ignoring base cases
    if x[i] = y[j]
        then c[i,j] <- LCS(x, y, i-1, j-1) + 1
    else
        c[i,j] <- max{ LCS(x, y, i-1, j), LCS(x, y, i, j-1) }
    return c[i,j]
```

worst-case: $\forall i,j$, $x[i]\neq y[j]$

Recursion Tree (m=7 , n=6)



height = m+n implies the work is exponential , slow as the brute-force !



same subtree

$\Rightarrow$ same subproblem

$\Rightarrow$ repeated work

Dynamic Programming Hallmarks #2

overlapping subproblems

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

与指数级比起来

in LCS example, subproblem space contains $m \times n$ distinct subproblems

so here is an improved algorithm !

it's an algorithm called "memo-ization algorithm"

"备忘录", 不是 memorization

Memo-ization Algorithm

```
LCS(x, y, i, j):
    if c[i,j] = nil
        then 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]
```

time $= \Theta(mn)$
space $= \Theta(mn)$

there is another strategy for doing exactly the same calculation in a bottom-up way
(真·动态规划)

the idea is to compute the table bottom-up

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0							
D	0							
C	0							
A	0							
B	0							
A	0							

→

		A	B	C	B	D	A	B	
	0	0	0	0	0	0	0	0	
B	0	0	1	1	1	1	1	1	填第一行
D	0								
C	0								
A	0								
B	0								
A	0								

→

		A	B	C	B	D	A	B	
	0	0	0	0	0	0	0	0	
B	0	0	1	1	1	1	1	1	
D	0	0	1	1	1	2	2	2	填第二行
C	0								
A	0								
B	0								
A	0								

→

		A	B	C	B	D	A	B	
	0	0	0	0	0	0	0	0	
B	0	0	1	1	1	1	1	1	
D	0	0	1	1	1	2	2	2	
C	0	0	1	2	2	2	2	2	填第三行
A	0								
B	0								
A	0								

→ ... →

		A	B	C	B	D	A	B	
	0	0	0	0	0	0	0	0	
B	0	0	1	1	1	1	1	1	
D	0	0	1	1	1	2	2	2	
C	0	0	1	2	2	2	2	2	
A	0	1	1	2	2	2	3	3	
B	0	1	2	2	3	3	3	4	
A	0	1	2	2	3	3	4	4	answer

time = $\Theta(mn)$

I can reconstruct the longest common subsequence by tracing backwards

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

"BCBA"

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

"BDAB"

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

"BCAB"

space = $\Theta(mn)$,
actually, we can do $\Theta(\min(m,n))$
实际上,需要保留的只有一行。

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	?	?	?	?
B	0							
A	0							

只有这些会影响?的计算,
之前的行都不参与计算

除了一行一行地填值,也可以一列一列地填值,这取决于m与n谁更小(省空间)