1) Longest common Sequence

S_1: M A Z C A (0 1 2 3 4)
S_2: K A Z Y Y

Order: Fill the dp table

		K	A	Z	Y	Y
	0	0	0	0	0	0
M	0	0	0	0	1	1
A	0	1	1	1	1	1
Z	0	1	1	2	2	2
C	0	1	1	2	2	2
A	0	1	2	2	2	2

dp[5][5] contains

$S_1[0-5]$, $S_2[0-5]$
Ans: Max

Final length: (4,4) → (3,4) → (2,3) → (1,2) → (0,1)
$\Rightarrow$ max(0, 1, 2, 2, 2)

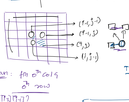
2) Longest common Subsequence

By state: dp[i][j] = longest common subsequence between string $S_1[0-i]$ & $S_2[0-j]$

By expression for LCS

1) dp[i][j] = { if ($S_1[i] == S_2[j]$): dp[i][j] = 1 + dp[i-1][j-1]
else: dp[i][j] = max(dp[i-1][j], dp[i][j-1]) }

3) dp[i][j] → dp[i-1][j-1], dp[i-1][j], dp[i][j-1]
depend:



Time: for n^2 cells
$n^2 \rightarrow n$

4) When will dp[i][j] come from dp[i-1][j-1]?

→ if ($S_1[i] == S_2[j]$): dp[i][j] = 1 + dp[i-1][j-1]
if we have to pick, 0/1 to our final ans
↑ ... J ...

[Ex: $C_{(2,2)}$* → max($C_{(1,2)}$, $C_{(2,1)}$)]

Print longest common subsequence
Note: use get-in recursion method

→ Return S_1, S_2, dp[N][M]
N M
Take i = N-1, M-1, lcs = "" (empty)

while (i >= 0 && j >= 0)
{ if ($S_1[i] == S_2[j]$): lcs = add($S_1[i]$), i--, j--
else { if dp[i][j] == dp[i-1][j] : i--
else { if (i > 0 && dp[i][j] == dp[i-1][j]) i--
else { j--
}
}
}
→ return lcs & print

TC: $O(n+m)$

3) Longest common Substring

S_1: [0 1 2 3 4 5 6] { g a h y u t f }
S_2: [0 1 2 3 4 5 6] { f l a h y t }

dp[i][j] = { if ($S_1[i] == S_2[j]$): dp[i][j] = 1 + dp[i-1][j-1]
else: dp[i][j] = 0 }

dp[i][j] = 1+ dp[3][4] } → may not be continuous
↑ ↑
i+1 j+1

dp[i][j] = longest common substring between $S_1[0-i]$, $S_2[0-j]$, such that ending at i & ending at j

Dp expression

dp[i][j] = { if ($S_1[i] == S_2[j]$): dp[i][j] = 1 + dp[i-1][j-1]
else: dp[i][j] = 0 } → There is no common substring starting in ending at i & j

Note: Where is your ans?
→ max in dp[i][j]
→ Return is forever
→ Tricks ?
Return dp[N][M] is wrong

Tricks
TC: $O(N*M)$
SC: $O(N*M)$
prev: 1 (substring) ↓
→ Use previous values to get
Recall

{ if ($S_1[i] == S_2[j]$): dp[i][j] = 1 + dp[i-1][j-1]
else: dp[i][j] = 0 }

5) Longest palindromic subsequence

S_1: [0 1 2 3 4 5 6 7 8 9 10] c b a g b c h i b a g b c
Same
→ find the longest palindromic subsequence
{ Longest palindromic subsequence: will also be same }

S_2: b a g b h c i b a g b c

lcs(S_1, S_2) = b a b a b
↳ lcs(S_1, rev(S_1))

6) Longest palindromic substring:

S_1: d f [c b g a g b c] f d
S_2: d f [c b g a g b c] f d

→ LCS(substring(S_1), rev(S_1)) ; We can do it $O(N)$ space
TC: $O(N^2)$

7) Min character to be added to make string palindrome
{ Only palindrome → }

0 1 2 3 4 5 6 7 8
S_1: [a c m c i g c h] → Max palindrome subseq we can get = c
ans = N - Longest Palindrome Subsequence (S_1)

8) Min characters to be added to make string palindrome

S_1: [a c m c i g c h] → N - lcs(S_1, rev(S_1))
(1 to n-1)

{ We need to form a 5 ans after reversing
{ We have given a string ans
{ All characters which are not in longest palindromic subsequence, for them we need to create, & mark copy }

9) Given S_1 & S_2 find Smallest Super sequence
length ↑

S_1: [a c e f d h]
S_2: [g c m f d]

{ A sequence is said to be a super sequence, if it contains both S_1 & S_2 as a sequence

Ex: a c e f d h g c m f d : super sequence ✓

Ex: (Ans: 8)
a g c m e f d h → common as a subsequence

→ [lcs(S_1, S_2)] + { S_1 - lcs(S_1, S_2) } + { S_2 - lcs(S_1, S_2) }

→ S_1 + S_2 - lcs(S_1, S_2) → [S_1.length() + S_2.length() - lcs(S_1, S_2)]

Ex: a c e a c
Ex: a n c c
a a c a c
→ a n c c (6)

* If + ? - lcs(S_1, S_2)
↑ + ↑ - j ↑

10) Printing Smallest Super Sequence

S_1: a c e f d h
S_2: g c m f d j
lcs(S_1, S_2): c f d
Dp(S_1, S_2): a g c m f

↑ ↑ ↑
0 0 0

Take i = 0, j = 0, k = 0
String lcs = lcs(S_1, S_2)
while(k < lcs.length()){
if ($S_1[i]$ == lcs[k] && $S_2[j]$ == lcs[k]) {
print($S_1[i]$), i++, j++, k++
}
else if ($S_1[i]$ == lcs[k]){
print ($S_2[j]$), j++
}
else if { print ($S_1[i]$), i++
while(i < S_1.length()) { print ($S_1[i]$), i++ }
while(j < S_2.length()) { print ($S_2[j]$), j++ }

// After this we can DFS take from top left character in S_1 & S_2

11) Check if A is a subsequence in B or not? [S.O.]

→ Note: Take B as a single character array

12) Given a string S_1, find length of longest repeating subsequence [Note]

Rules form
Note: Substr: Character of a subsequence should not have same index value in S_1
Note: If we use $S[i]$ in subseq S_1 we cannot use $S[i]$ in subseq S_2

Ex:
S → A A B : A
S → A A B B : AB
S → A A B B C C D : AB
S → A B A B A : A
S → X X X X : XX
S: d A A A A d: AA

Ex: [LCS (S, S)] → Same character is present in both subsequence

dp[i, j] = { $S[i-1]$ == $S[j-1]$ }

dp expression

dp[i, j] = { if (i == j): dp[i, j] = 0
else if ($S[i-1]$ == $S[j-1]$) { dp[i, j] = max(dp[i-1, j-1], dp[i-1, j]) }
else { dp[i, j] = max(dp[i-1, j], dp[i-1, j]) }

Since both are longest common subsequence, then same number character, B not picked in both B Team.

$S[i]$ should only be consider 1 subsequence

dp[i, j] → { if (i == j): 0
else { if ($S[i-1]$ == $S[j-1]$) { dp[i-1, j-1] + 1 }
else { max(dp[i-1, j], dp[i-1, j]) } }

The same character is not considered both subsequence

Note: LCS: LCS excluding character at same index
dp[i][j] = LCS between $S[0-i]$ & $S[0-j]$, such that character at same index not considered

Dp expression:
dp[i][j] =

Ex: Find Dp Table for
S: AABBC

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