DP Template #2. Longest Common Subsequence (LCS)

↳ Subset but with order.

Q. |m| s_1 = aggtab $\underline{s_1}$

|n| s_2 = gctaayb scramble ↓

s_2 and s_2' - gctabay s_2'

CS : {a, a, b} → length = 3

Some subsequence that is available in both strings.

O/p. : length of LCS.

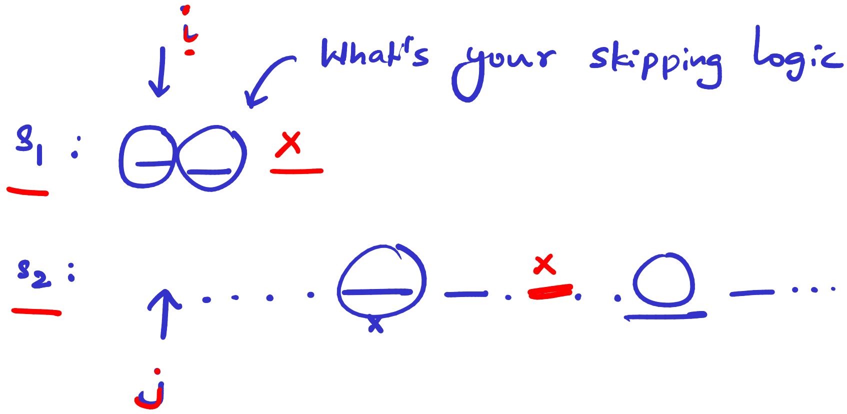
LCS → {g t a b} → length = 4.

4. Can I say this:

★ Feel this foresight!!

I can't tell you if I should pick/skip $s_1[0]$ unless I have seen rest of string!!

What's your skipping logic ?



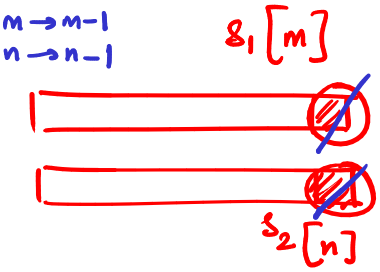
Knapsack 23 20 4
11 10 2
c = 12. ? ×

Recursion:

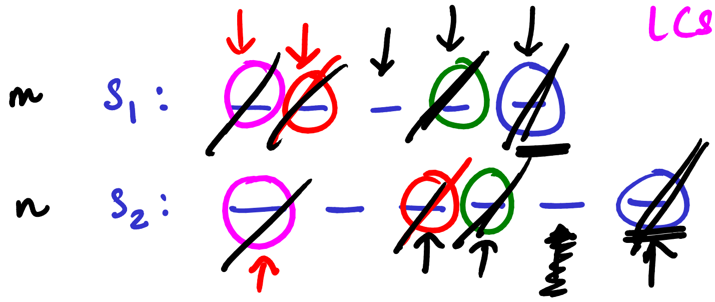
int lcs (s_1, s_2, m, n) :

length of LCS.

pick. m → m-1, n → n-1



LCS = 4.



Don't know. → m → m-1 OR n → n-1

	A	B
blue :	✓	✓
green :	✗	✓
red :	✓	✗

ans →

2 possibilities.

c: $s_1[m] == s_2[n]$ → May be part of LCS → pick. / May be not. → skip

$s_1[m] != s_2[n]$ → surely not. → skip.

Final logic :

Match : m → m-1, n → n-1 both | AND

mismatch : ans → m-1, n OR m, n-1

```
int   lcs (s1, s2, m, n) :          ← if either of string finishes
    if  m == -1  ||  n == -1  :  return 0
→   if  memo[m][n] != -1 :  return memo[m][n]
    if  s1[m] == s2[n] :
        memo[m][n] =
        ~~return~~  1 + lcs(s1, s2, m-1, n-1)

    else:
        memo[m][n] =
        ~~return~~   max { lcs (s1, s2, m, n-1) ,
                           lcs (s1, s2, m-1, n) }
                 ~~lcs (s1, s2, m-1, n-1)~~*   ← needful. → ✓ ✗
```

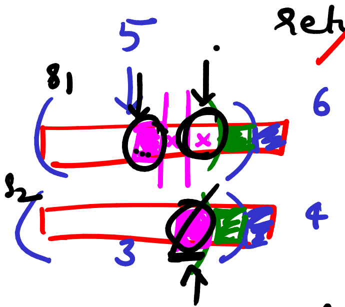
s_1: "abc"

s_2: ""

int memo[m][n] = {-1}

LCS

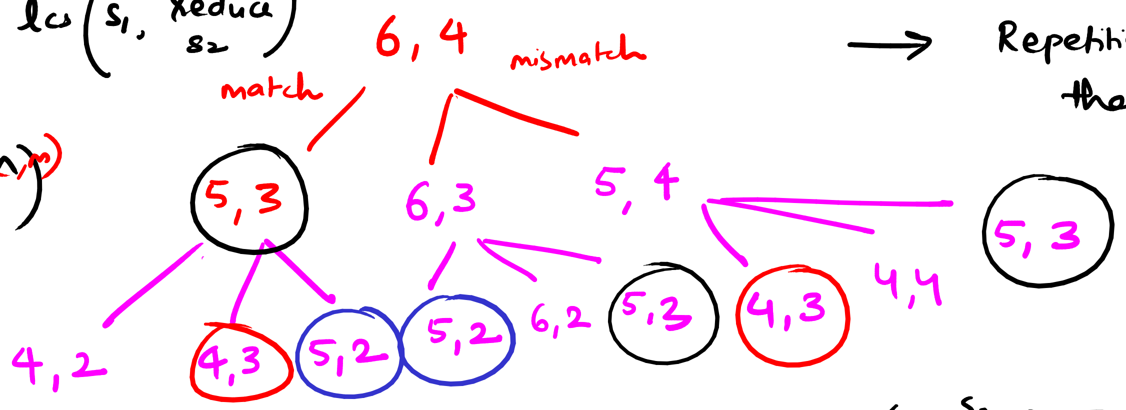
Code DONE!



6 ← return memo[m][n]

4 ← ✗ →

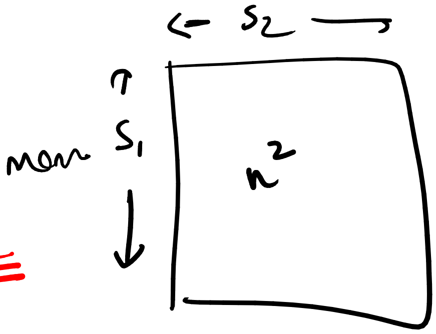
lcs (reduce s1, s2)   lcs(s1, reduce s2)



→ Repetition is there !!

T.C. ~ $O(3^{(n,m)})$

T.C. $O(m.n)$



Variations.

Longest common substring → LC

① s_1 = abcdef | n

s_2 = dbcbcdea | m

o/p : 4.

```
int  lc substring (s1, s2, m, n):
    if  m == -1  || n == -1  :  return 0
    if  s1[m] == s2[n] :
        max_len = max (maxlen, 1+ lcssubstring(s1, s2,
                                     m-1, n-1))
        return max_len
    else:
        lcsubstring (s1, s2, m-1, n)   } ✗
        lcsubstring (s1, s2, m, n-1)   }
```

O(m.n) ↓ after memo.

LCS logic.

```
if string finishes :
    return 0
if match happens:
    return 1+lcs(m-1, n-1)
else:
    return max { lcs (m, n-1)
                 lcs (m-1, n) }
```

return 0

② s_1 = abacd . } LCS. → LC: | +.

s_2 = bcde .

s_1 : abce

s_2 : bdcf

s = abacde | shortest supersequence that contains s_1 and s_2 both as subsequences.

length.

logic: abac de → SCS → shortest common supersequence.

length = lcs + all chars in s_1 that are not in lcs + all chars in s_2 that are not in LCS.

$$= lcs + (m - lcs) + (n - lcs)$$

lcs occurs in both.

$$= |m + n - lcs|$$

⇒ takes s_1 } take s_2 } subtract once ∴ chars need to be considered once

③ 2 kind of Operation : insert a char
delete a char
~~update a char~~ *

s_1 : heap s_1

s_2 : pea s_2

$s_1 \to s_2$ # min operations required to do so.

$$\text{heap} \xrightarrow{-p} \text{hea} \xrightarrow{-h} \overset{LCS}{\boxed{ea}} \xrightarrow{+p} \text{pea}$$

o/p : 3.

Remove extra chars from s_1 (∴ const deleting)

add extra chars in s_2 (∴ const insertion)

$$\#\text{min ops} = \underline{(m - lcs)} + (n - lcs)$$

$$= |m + n - 2 * lcs| \longrightarrow \text{ans.}$$

④ $s = \overset{\downarrow}{a}\, g\, \overset{\downarrow}{b}\, \overset{\downarrow}{c}\, \overset{\downarrow}{b}\, \overset{\downarrow}{a}$

$\underline{abcba}$

longest subsequence that is a palindrome

O/p = 5

LPS ⇒ longest palindromic subsequence.

$s_1 = s.$

$s_2 = s.\text{reverse}()$

$$\left.\begin{array}{l} a\,g\,b\,c\,b\,a \\ a\,b\,c\,b\,g\,a \end{array}\right\}\text{lcs} \rightarrow \underleftarrow{abcba}$$

⑤ s = agbcba → # min ops (insertion/deletion) to make it palindrome.

M

+

Either insert: agbcbga → 1 insertion

↑

OR delete: a ~~g~~ bcba

abcba → 1 deletion.

$$|\, m - \text{lcs}(s, s.\text{reverse}())\,| \rightarrow \underline{\underline{\text{ans.}}}$$

$$6 - 5 \Rightarrow \text{ans} = \boxed{1}$$