Longest common substring

The longest common substring problem is the problem of finding the longest string (or strings) that is a substring of two strings

eg : s_1 = "A$\boxed{\text{BABC}}$" $S2$ = "$\boxed{\text{BABC}}$A"

method 1

Consider all substrings of 2nd string and find the longest substring that is also a substring of first string

$$O\left((m+n) * m^2\right)$$

method 2

Dynamic Programming

$$lcs[i][j] = \begin{cases} 1 + lcs[i-1][j-1] & \text{if } X[i-1] = Y[j-1] \\ 0 & \text{else} \end{cases}$$

	"	A	B	A	B
	0	0	0	0	0
B	0	0	0+1=1	0	0+1=1
A	0	0+1=1	0	1+1=2	0
B	0	0	1+1=2	0	2+1=3
A	0	0+1=1	0	2+1=3	0

To Backtrack

Find max number in matrix

and diagonally go backward

till we get 0

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

ABA , BAB

These 2 are longest common substring.

```
memo = [m+1] [n+1]                                   boxes
                                       initialize all with 0

        For i in range (1, m+1):
O(m)        for j in range (1, n+1):
                if x[i-1] = Y[j-1]:
    O(n)   O(1)     memo[i][j] = 1 + memo[i-1][j-1]
                else
O(m·n)              memo[i][j] = 0
```

```
      OPTIMAL ANS
    ↓
    temp = max (memo)

    FOR OPTIMAL SOLUTION

    pos = position of temp = (i, j)
            ans = [ ]
            ans.append (memo[i][j])

            For K in range (temp):
                i = i-1
                j = j-1
                ans.append (memo[i-1][j-1])
```

Time complexity $O(m \cdot n)$

space complexity $O(m \cdot n)$