1. 부분 수열 : 부분 수열이란 서로 겹치는 수열(문자열)을 의미함

₩중간에 건너 뛰어도 되지만 순서대로 지나가면서 겹치는 문자열들의 모임₩

2. LCS : 부분 수열 중에서 가장 긴 것

ACAYKP 와 CAPCAK -> ACAK

3. Brute-Force로 풀기(무식하게) : X(X_1~X_m)의 모든 부분 서열의 개수 -> O(2^m)

4.

0) X = x_1~x_m

Y = y_1~y_n

Z = z_1~z_k

이렇게 있을 때 (Z는 최장 부분수열이라면)

1) z_k = x_m = y_n

₩둘이 같다면 z_k가 정해짐₩

z_k-1 = LCS(X_m-1, Y_n-1)

₩나머지 에서 z_k-1를 다시 구하면 됨₩

2) x_m != y_n

₩둘이 공통 부분 수열이 아니라면 - > 경우 2가지₩

ㄱ) z_k != x_m : Z = LCS(X_m-1, Y_n)

₩z_k가 x_m이 아닐때₩

ㄴ) z_k != y_n : Z = LCS(X_m, Y_n-1)

₩z_k가 y_n이 아닐때₩

3) 재귀적으로 정의하기

ㄱ) c(i, j) : 수열 X_i와 Y_j의 최장공통부분수열의 길이

ㄴ) 베이스케이스 : i = 0 또는 j = 0 이면 c(i, j) = 0

ㄷ) 재귀조건(점화식)

a. x_i = y_j 이면 c(i, j) = c(i-1, j-1) + 1

₩이미 구한 공통부분수열 1개는 제외하고 나머지 X_i-1과 Y_j-1에서 공통부분수열 찾기₩

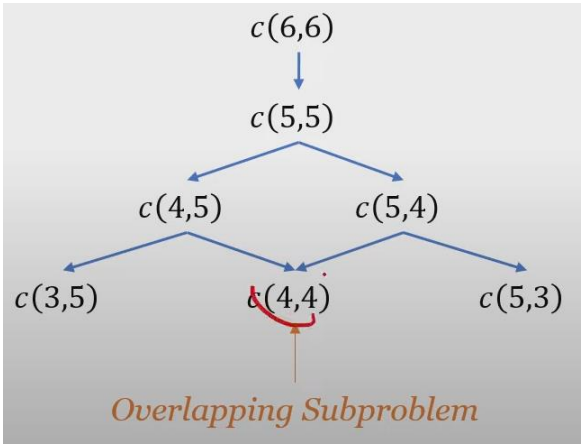
b. x_i != y_j 이면 c(I, j) = max{c(i, j-1), c(i-1, j)}

₩각자 하나씩 빼서 공통부분수열의 길이를 찾아보고, 더 긴쪽을 선택하면 됨₩

```
def lcs(x, y):
    m, n = len(x), len(y)
    if m == 0 or n == 0:
        return 0
    else:
        if x[-1] == y[-1]:
            return lcs(x[:(m-1)], y[:(n-1)]) + 1
        else:
            return max(lcs(x[:m], y[:(n-1)]),
                       lcs(x[:(m-1)], y[:n]))
```

* 중복 부분 문제

1) 'c(i, j) = 0' 이거나 'c(i, j) = c(i – 1, j – 1) + 1' 이거나 'c(i, j) = max{c(i, j-1), c(i-1, j)}

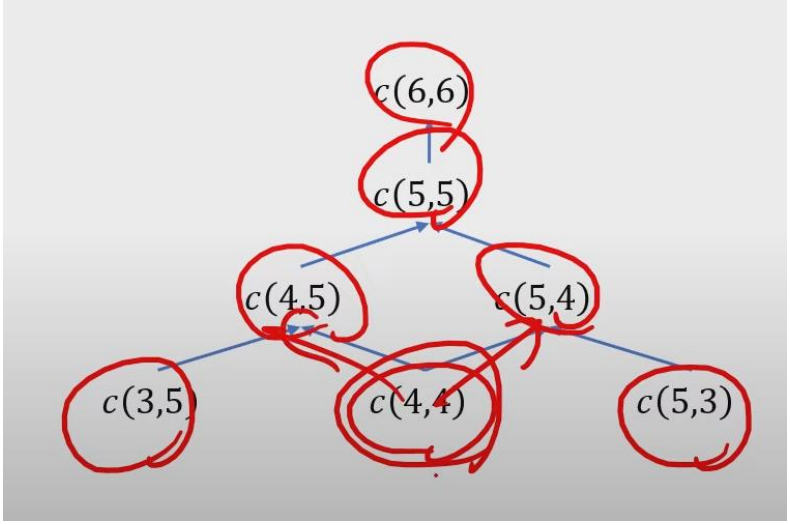


c(4,4)는 중복 부분 문제가 됨 - > 시간복잡도가 O(2^n)

₩재귀적으로 갈라지다보면 c(4,4)처럼 중복되는 부분문제가 생김₩

* 동적계획법 풀이

** 저런 중복 부분 문제가 있으면 해결해줄 수 있는 방법 -> 메모이제이션



₩상향식으로 생각해보면 밑에서 c(4,4)를 테이블이 저장해둬서 중복될 때마다 메모이제이션 해둔 걸 사용하면 됨₩

```
# 9251 LCS
S1 = list(input())
S2 = list(input())
# 문자열을 입력받아서 각 문자별로 리스트에 저장

len1 = len(S1)
len2 = len(S2)
# 각 길이를 저장

dp = [[0]*(len2 + 1) for _ in range(len1+1)]
# 메모이제이션할 공간 만들기(하나씩 더 공간을 만드는 이유 : 처음 시작하는 0000 으로
시작하는 세로와 가로를 세팅하기 위해서)
```

```
# print(dp)


for i in range(1,len1 + 1) :
    for j in range(1,len2 + 1) :
        if S1[i-1] == S2[j-1] :
            dp[i][j] = dp[i-1][j-1] + 1
        else :
            dp[i][j] = max(dp[i-1][j],dp[i][j-1])
# 표로 설명


# print(dp)


print(dp[len1][len2])
```