Edit - Distance

↳ Minimum steps to convert word 1 to word 2

Recursive ⇒ DP

Base Case :

(i) word1 = "abc" → Delete all characters
word2 = " " → return len(word1)

(ii) word1 = " "
word2 = "abc" ⇒ insert all elements of word2
return len(word2)

Hypothesis : Operations → Insert, Delete, Replace } minimum

word1 = a b d (i)
word2 = a c d (j)

not equal
word1[i] ≠ word2[j]
⇒ Calculate with which operation steps will be minimum.

equal
word1[i] == word2[j]
⇒ Don't need any operation
i++ , j++

---

word1 = a b d (n)
word2 = a c d (m)
→ equal (n-1, m-1)

---

word1 = a c d (n)
word2 = a c d (m)
Replace ⇒ (n-1, m-1)
both became equal by replace so decrease both pointers.

---

word1 = a ~~b~~ d (n)
word2 = a c d (m)
Delete ⇒ a d (n), a c d (m) → (n-1, m)
As 'd' is already checked so check for 'a' in word1 and (n-1)

word1 = a b d (b: n)
word2 = a c d (c: m)

Insert ⟹ a b c d (n)
a c d (m)

As for 'c' of word2 we added 'c' in word1 so c is checked. so now we have to check for remaining ie. left side elements of word2 ∴ (m-1)

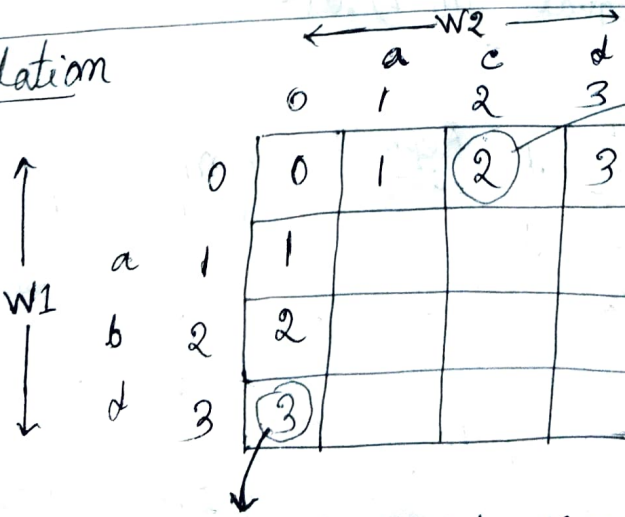
(n, m-1)

```
def solve(w1, w2, n, m):
    if n==0 or m==0 : return m or n

    elif w1[n-1] == w2[m-1]:
        return solve(w1, w2, n-1, m-1)

    else:
        return 1 + min(
        solve(w1, w2, n-1, m-1) # replace
        solve(w1, w2, n-1, m) # Delete
        solve(w1, w2, n, m-1) # Insert
        )
```

Tabulation



if w1 is empty and w2 is of length 2 i.e. w2 = ac minimum 2 steps needed

if w2 = "" and w1 = abd minimum 3 steps that is why this cell contain 3

Top-down

```
dp = [[0]*(m+1) for i in range(n+1)]

for j in range(m+1):        # Base case
    dp[0][j] = j                0th row
                                w1 = " "
for i in range(n+1):        # Base case
    dp[i][0] = i                0th column
                                w2 = " "

for i in range(1, n+1):
    for j in range(1, m+1):
        if w1[i-1] == w2[j-1]
            dp[i][j] = dp[i-1][j-1]

        else:
            dp[i][j] = 1 + min(
                        dp[i-1][j-1]   # Replace
                        dp[i-1][j]     # Delete
                        dp[i][j-1]     # Insert
                        )

return dp[-1][-1]
```