DYNAMIC PROGRAMMING 3

DP CATEGORIES

- Matrix type
- Sequence type
    - Single sequence
    - Multiple sequences
- Backpack type

MULTIPLE SEQUENCES DP

- Usually 2 sequences
- State
  - For 2 seq DP, usually represented as 2D list/array state[i][j]
    - i stands for the index of the first sequence
    - j stands for the index of the second sequence
  - Initialization
    - state[i][0] & state[0][j]

NO. 72

EDIT DISTANCE

PROBLEM DESCRIPTION

- Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)
- You have the following 3 operations permitted on a word:
    - a) Insert a character
    - b) Delete a character
    - c) Replace a character

IDEAS

- Insertion, deletion, or replacement

	a	b	c
a	0	1	2
b	1	0	1
c	2	1	0

	a	b	c
a	0	1	1
b	1	0	1

	a	b
a	0	1
b	1	0
c	2	1

	a	b	c
a	0	1	2
d	1	1	2
c	2	1	1

Go diagonal, dis is 0 is char match

Go diagonal, dis is 1 if char not match // replacement

Go right, dis is 1 // deletion

Go down, dis is 1 // insertion

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/72_Edit_Distance

NO. 97

INTERLEAVING STRING

PROBLEM DESCRIPTION

- Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.
- For example, given:
    - s1 = "aabcc"
    - s2 = "dbbca"
- When s3 = "aadbbcbcac", return true.
- When s3 = "aadbbbaccc", return false.

IDEAS

- s1: [a,b,c]
- s2: [1,2,3]
- s3: [a,b,1,2,c,3] *#should return True*
- len(s1) + len(s2) == len(s3)
- Either go right or down
  - dp[i][j] = dp[i][j-1] and s3[i+j-1] == s1[j-1] *# right*
  - dp[i][j] = dp[i-1][j] and s3[i+j-1] == s2[i-1] *# down*

S	a	b	c
1			
2			
3			E

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/97_Interleaving_String

NO. 115

DISTINCT SUBSEQUENCES

PROBLEM DESCRIPTION

- Given a string S and a string T, count the number of distinct subsequences of T in S.
- A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).
- Here is an example: S = "rabbbit", T = "rabbit"
- Return 3.

rabbbit
rabbbit
rabbbit

IDEAS

- It's hard !
- E.g. S=[a,b,c,c], T=[a,b,c]
- Rules:
    - if len(T) < len(S), ans MUST be 0
    - if T[i] == S[j], dp[i][j] = dp[i-1][j-1] + dp[i][j-1]
    - if T[i] != S[j], dp[i][j] = dp[i][j-1]

		a	b	c	c
	1	1	1	1	1
a	0	1	1	1	1
b	0	0	1	1	1
c	0	0	0	1	2

SOLUTION

- https://github.com/Brady31027/leetcode/tree/master/115_Distinct_Subsequences