Question: https://leetcode.com/problems/edit-distance/

with two string there can be two cases

1.s1=""s2="abc", then n number of insertions required

2.s1="abc"s2="", then n number of deletions required

Now while comparing charecters there can be two cases

1.c1==c2, no operation required to be perfomed

2.c1!=c2, there can be 3 possibilities

1.Insert: we imagine a charecter is added at the front and thus now the pointer for string 1 remains at same place because if we consider te new charecter then the pointer actually increamented by 1 and the pointer for string 2 increaments by 1

2.Replace: the pointer for both the string increases

3.Delete: the pointer for string 1 increases cause we consider that the previous pointed char got deleted,and the pointer at string 2 remains unchanged because we havn't yet found the matching char

DP Approach best explained at this link: <https://www.youtube.com/watch?v=b6AGUjqIPsA>

Code:  
class Solution {

public int min(int a, int b, int c){

if(a<b){

if(a<c){

return a;

}else{

return c;

}

}else{

if(b<c){

return b;

}else{

return c;

}

}

}

public int minDistance(String word1, String word2) {

int dp[][] = new int[word2.length()+1][word1.length()+1];

for(int i=0; i<=word2.length(); i++){

for(int j=0; j<=word1.length(); j++){

if(i==0){

dp[i][j]=j;

}else if(j==0){

dp[i][j]=i;

}else{

if(word2.charAt(i-1)==word1.charAt(j-1)){

dp[i][j]=dp[i-1][j-1];

}else{

dp[i][j]=min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1]+1);

}

}

}

}

return dp[word2.length()][word1.length()];

}

}

Github Link :<https://lnkd.in/ecwtJeaz>