Leetcode May Challenge DAY: 31

1. Python

Let `dp[i_1][i_2]` be the edit distance for words `dp[:i_1]` and `dp[:i_2]`. Then there can be 4 options: we can insert symbol on position `i_1` in the first word, insert symbol on position `i_2` in the second word, replace symbol `i_1` in first word with `i_2` if these symbols are different and look into `dp[i_1-1][i_2-1]` if these two symbols are the same.

Complexity: time complexity is `O(mn)`, and space comlexity as well. Space complexity can be improved to `O(m+n)` if we keep only current row or column.

We add stopsymbols in the beggining of words to deal with border cases in simpler way.

```
class Solution:
  def minDistance(self, word1, word2):
    word1, word2 = "!" + word1, "!" + word2
    n_1, n_2 = len(word1), len(word2)
    dp = [[0] * n_2 for _ in range(n_1)]

    for i_1 in range(n_1): dp[i_1][0] = i_1
    for i_2 in range(n_2): dp[0][i_2] = i_2

    for i_1 in range(1, n_1):
      for i_2 in range(1,n_2):
        Cost = (word1[i_1] != word2[i_2])
        dp[i_1][i_2] = min(dp[i_1-1][i_2] + 1, dp[i_1][i_2-1] + 1, dp[i_1-1][i_2-1] + Cost)

    return int(dp[-1][-1])
```

2. C++

```
int minDistance(string word1, string word2) {
// Create a table to store results of subproblems
    int dp[word1.size()+1][word2.size()+1];
// If first string is empty, only option is to
// insert all characters of second string
    for(int k=0; k<=word1.size(); k++)
      dp[k][0] = k;
// If second string is empty, only option is to
// remove all characters of second string
    for(int k=0; k<=word2.size(); k++)
      dp[0][k] = k;
// Fill dp[][] in bottom up manner
    for(int i=1; i<=word1.size(); i++){
      for(int j=1; j<=word2.size(); j++){
//if characters at current position in 2 strings are equal
//there will be no new operation so copy value from previous position
        if(word1[i-1] == word2[j-1])
          dp[i][j] = dp[i-1][j-1];
// If the last character is different, consider all
// possibilities and find the minimum
        else
          dp[i][j] = 1 + min(dp[i][j - 1], // Insert
                  dp[i - 1][j], // Remove
                  dp[i - 1][j - 1]); // Replace
      }
    }
    return dp[word1.size()][word2.size()];
  }
```

3. JAVA

```
public int minDistance(String word1, String word2) {
    int m = word1.length();
    int n = word2.length();
    int[][] dp=new int[m+1][n+1];
    for(int i = m;i >= 0; i--){
       for(int j = n;j >= 0; j--){
          if(i==m || j==n){
             dp[i][j]=(i==m ? n-j : m-i);
          }
          else if(word1.charAt(i)==word2.charAt(j)){
             dp[i][j]=dp[i+1][j+1];
          }
          else{
             dp[i][j]=1+Math.min(dp[i+1][j+1], Math.min(dp[i+1][j], dp[i][j+1]));
          }
       }
    }
    return dp[0][0];
}
```