# 72.编辑距离

<https://leetcode-cn.com/problems/edit-distance/>

**import static** java.lang.Math.*max*;  
**import static** java.lang.Math.*min*;  
  
**public class** Solution {  
 **public int** minDistance(String word1, String word2) {  
 **return this**.MyLeetcodeTry(word1, word2);  
 }  
  
 **public int** MyLeetcodeTry(String word1, String word2) {  
 **int** len1 = word1.length();  
 **int** len2 = word2.length();  
 **int**[][] dp = **new int**[len1 + 1][len2 + 1];  
 **for** (**int** i = 0; i <= len1; i++) {  
 dp[i][0] = i;  
 }  
 **for** (**int** i = 0; i <= len2; i++) {  
 dp[0][i] = i;  
 }  
 **int** a, b, c;  
 **for** (**int** i = 1; i <= len1; i++) {  
 **for** (**int** j = 1; j <= len2; j++) {  
 a = dp[i - 1][j - 1];  
 b = dp[i][j - 1];  
 c = dp[i - 1][j];  
 **if** (word1.charAt(i - 1) != word2.charAt(j - 1))  
 dp[i][j] = Math.*min*(a, Math.*min*(b, c)) + 1;  
 **else** dp[i][j] = a;  
 }  
 }  
 **return** dp[len1][len2];  
 }  
  
 **public int** LeetcodeMinDis(String word1, String word2) {  
 **int** n = word1.length();  
 **int** m = word2.length();  
  
 *// 有一个字符串为空串* **if** (n \* m == 0)  
 **return** n + m;  
  
 *// DP 数组* **int**[][] D = **new int**[n + 1][m + 1];  
  
 *// 边界状态初始化* **for** (**int** i = 0; i < n + 1; i++) {  
 D[i][0] = i;  
 }  
 **for** (**int** j = 0; j < m + 1; j++) {  
 D[0][j] = j;  
 }  
  
 *// 计算所有 DP 值* **for** (**int** i = 1; i < n + 1; i++) {  
 **for** (**int** j = 1; j < m + 1; j++) {  
 **int** left = D[i - 1][j] + 1;  
 **int** down = D[i][j - 1] + 1;  
 **int** left\_down = D[i - 1][j - 1];  
 **if** (word1.charAt(i - 1) != word2.charAt(j - 1))  
 left\_down += 1;  
 D[i][j] = Math.*min*(left, Math.*min*(down, left\_down));  
  
 }  
 }  
 **return** D[n][m];  
 }  
  
 **public int** MyMinDis(String word1, String word2) {  
 **if** (word1.length() == 0)  
 **return** word2.length();  
 **if** (word2.length() == 0)  
 **return** word1.length();  
 **if** (word1.equals(word2))  
 **return** 0;  
  
 **int** maxLength = Math.*max*(word1.length(), word2.length());  
 **int** lengthOfSubS = word1.length();  
 **int** start, end, indexContain;  
 **int** distance = 0, minDis = maxLength;  
  
 **while** (lengthOfSubS > 0) {  
 start = 0;  
 end = lengthOfSubS;  
 **for** (**int** i = 0; i < word1.length() - lengthOfSubS + 1; i++) {  
 String tempSubString = word1.substring(start, end);  
  
 indexContain = word2.indexOf(tempSubString, 0);  
 *//有某一位相同，进入子串分支求和* **if** (indexContain >= 0) {  
 distance = 0;*//清空改动计数器  
 //前半部分* String tempWord1 = word1.substring(0, start);  
 String tempWord2 = word2.substring(0, indexContain);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//后半部分* tempWord1 = word1.substring(end);  
 tempWord2 = word2.substring(indexContain + lengthOfSubS);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//比较是不是最小改动* minDis = Math.*min*(minDis, distance);  
 }  
  
 **while** (indexContain > 0) {  
 *//查看剩余位是否相同* indexContain = word2.indexOf(tempSubString, indexContain + 1);  
 *//有某一位相同，进入子串分支求和* **if** (indexContain >= 0) {  
 distance = 0;*//清空改动计数器  
 //前半部分* String tempWord1 = word1.substring(0, start);  
 String tempWord2 = word2.substring(0, indexContain);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//后半部分* tempWord1 = word1.substring(end);  
 tempWord2 = word2.substring(indexContain + lengthOfSubS);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//比较是不是最小改动* minDis = Math.*min*(minDis, distance);  
 }  
 }  
  
 *//word1换一个子串查* start++;  
 end++;  
 }  
 lengthOfSubS--;  
  
 **if** (distance > 0 && lengthOfSubS > 3) {  
 *//如果在这个位数下面已经有结果了，还继续嘛？  
 //试一下少一位，最多也就是差一位吧？？  
 //嗯 个位就不用试了 没意义* start = 0;  
 end = lengthOfSubS;  
 **for** (**int** i = 0; i < word1.length() - lengthOfSubS + 1; i++) {  
 String tempSubString = word1.substring(start, end);  
  
 indexContain = word2.indexOf(tempSubString, 0);  
 *//有某一位相同，进入子串分支求和* **if** (indexContain >= 0) {  
 distance = 0;*//清空改动计数器  
 //前半部分* String tempWord1 = word1.substring(0, start);  
 String tempWord2 = word2.substring(0, indexContain);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//后半部分* tempWord1 = word1.substring(end);  
 tempWord2 = word2.substring(indexContain + lengthOfSubS);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//比较是不是最小改动* minDis = Math.*min*(minDis, distance);  
 }  
  
 *//查看剩余位是否相同* **while** (indexContain > 0) {  
 indexContain = word2.indexOf(tempSubString, indexContain + 1);  
 *//有某一位相同，进入子串分支求和* **if** (indexContain >= 0) {  
 distance = 0;*//清空改动计数器  
 //前半部分* String tempWord1 = word1.substring(0, start);  
 String tempWord2 = word2.substring(0, indexContain);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//后半部分* tempWord1 = word1.substring(end);  
 tempWord2 = word2.substring(indexContain + lengthOfSubS);  
 distance += **this**.MyMinDis(tempWord1, tempWord2);  
 *//比较是不是最小改动* minDis = Math.*min*(minDis, distance);  
 }  
 }  
  
 *//word1换一个子串查* start++;  
 end++;  
 }  
 **break**;  
 }  
  
 }  
  
 **if** (distance == 0) {  
 *//说明没有任意一位相同* **return** maxLength;  
 }  
 **return** minDis;  
 }  
  
 **public int** minDisWrong(String word1, String word2) {  
 **if** (word1.length() == 0)  
 **return** word2.length();  
 **if** (word2.length() == 0)  
 **return** word1.length();  
 **char** chCompare;  
 **int** index1 = 0, index2 = 0;  
 **int** indexSame = 0, indexStart2 = 0;  
 **int** dis2 = 0, dis1 = 0;  
 **int** distance = 0;*//按照本次长度比较得到的改动数* **int** minDis = Math.*max*(word1.length(), word2.length());*//最终* **while** (indexStart2 < word2.length()) {  
 *//从头开始比较* index1 = 0;  
 index2 = indexStart2;  
 dis2 = indexStart2;*//从第几位开始比较，之前的所有都算2的改动* distance = 0;*//清空之前的dis* **while** (index1 < word1.length() && index2 < word2.length()) {  
 chCompare = word2.charAt(index2);  
 indexSame = word1.indexOf(chCompare, index1);  
 **if** (indexSame == -1) {  
 dis2++;*//如果没有相同字符，计入字符串2的改动* } **else** {  
 *//如果有相同，标记改动，计数* dis1 = indexSame - index1;*//字符串1的改动数，是相同的字符之间的间隔* distance += Math.*max*(dis1, dis2);*//计算总改动* dis1 = 0;*//清空计数器* dis2 = 0;*//清空计数器* index1 = indexSame + 1;*//字符串1的指针移到字符相同的位置，的下一个位置* }  
 index2++;  
 **if** (index2 >= word2.length()) {  
 *//如果2比较结束，就结束整段比较* distance += Math.*max*(dis2, word1.length() - index1);  
 }  
 **if** (index1 >= word1.length()) {  
 *//如果1比较结束，就结束整段比较* distance += word2.length() - index2;  
 }  
 }  
 minDis = Math.*min*(distance, minDis);*//验证是最小的方案* indexStart2++;*//每一轮比较都向后找一位word2* }  
 **return** minDis;  
 }  
  
 **public static void** main(String[] args) {  
 Solution s = **new** Solution();  
 **int** n;  
 n = s.minDistance(**"horse"**, **"ros"**);*//3* System.***out***.println(n);  
 n = s.minDistance(**"mart"**, **"karma"**);*//3* System.***out***.println(n);  
 n = s.minDistance(**"intention"**, **"execution"**);*//5* System.***out***.println(n);  
 n = s.minDistance(**"execution"**, **"intention"**);*//5* System.***out***.println(n);  
 n = s.minDistance(**""**, **"a"**);*//1* System.***out***.println(n);  
 n = s.minDistance(**"a"**, **"ab"**);*//1* System.***out***.println(n);  
 n = s.minDistance(**"ab"**, **"bc"**);*//2* System.***out***.println(n);  
 n = s.minDistance(**"sea"**, **"ate"**);*//3* System.***out***.println(n);  
 n = s.minDistance(**"industry"**, **"interest"**);*//6* System.***out***.println(n);  
 n = s.minDistance(**"prosperity"**, **"properties"**);*//4* System.***out***.println(n);  
 n = s.minDistance(**"a"**, **"a"**);*//0* System.***out***.println(n);  
 n = s.minDistance(**"abcdxabcde"**, **"abcdeabcdx"**);*//2* System.***out***.println(n);  
 n = s.minDistance(**"dinitrophenylhydrazine"**, **"acetylphenylhydrazine"**);*//6* System.***out***.println(n);  
  
 }  
}