#### [583. 两个字符串的删除操作](https://leetcode-cn.com/problems/delete-operation-for-two-strings/)

给定两个单词 word1 和 word2，找到使得 word1 和 word2 相同所需的最小步数，每步可以删除任意一个字符串中的一个字符。

示例 1:

输入: "sea", "eat"

输出: 2

解释: 第一步将"sea"变为"ea"，第二步将"eat"变为"ea"

说明:

给定单词的长度不超过500。

给定单词中的字符只含有小写字母。

来源：力扣（LeetCode）

链接：https://leetcode-cn.com/problems/delete-operation-for-two-strings

著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

解法一：动态规划

Dp[i][j] 表示将word1的前 i 个字符串转变为word2的前 j 个字符串所需要的最少次数

有如下三种情况  
1.如果word1[i]=word2[j]

那么只需要讲word1 的前 i-1个字符串转变为 word2 的前j-1个字符串即可匹配完成

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

2.如果word1[i]!=word2[j]

那么可以有两种情况

1. 将 word1 的前i-1个字符串转变为word2的前j个字符串 然后删除掉word1[i]

Dp[i][j] = dp[i-1][j]+1

1. 将word1的前i个字符串转变为word2的前j-1个字符串 然后删除掉word2[j]

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

最后结果取两者中的较小值即可

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

代码如下：

二维动态规划

class Solution {

public int minDistance(String word1, String word2) {

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

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

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

{

if(i==0)

dp[0][j] = j;

if(j==0)

dp[i][0] = i;

if(i!=0 && j!=0)

{

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

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

else

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

}

}

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

}

}

一维动态规划

public class Solution {

public int minDistance(String s1, String s2) {

int[] dp = new int[s2.length() + 1];

for (int i = 0; i <= s1.length(); i++) {

int[] temp=new int[s2.length()+1];

for (int j = 0; j <= s2.length(); j++) {

if (i == 0 || j == 0)

temp[j] = i + j;

else if (s1.charAt(i - 1) == s2.charAt(j - 1))

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

else

temp[j] = 1 + Math.min(dp[j], temp[j - 1]);

}

dp=temp;

}

return dp[s2.length()];

}

}

方法二：求两个字符串的最大相同子序列长度 n ,结果为 word1.length+word2.length-2\*n

1. 递归求法（超时）

LCS（s1,i,s2.j）表示 s1 的前i个字符串和s2的前j个字符串最大相同子序列长度

class Solution {

public int LCS(String s1,int i,String s2,int j)

{

if(i==0 || j==0)

return 0;

if(s1.charAt(i-1)==s2.charAt(j-1))

return 1+LCS(s1,i-1,s2,j-1);

else

return Math.max(LCS(s1,i-1,s2,j),LCS(s1,i,s2,j-1));

}

public int minDistance(String word1, String word2) {

int k = LCS(word1,word1.length(),word2,word2.length());

return word1.length()+word2.length()-2\*k;

}

}

1. 带记忆化的递归

public class Solution {

public int[][] memo;

public int minDistance(String s1, String s2) {

memo = new int[s1.length()+1][s2.length()+1];

return s1.length() + s2.length() - 2 \* lcs(s1, s2, s1.length(), s2.length());

}

public int lcs(String s1, String s2, int m, int n) {

if (m == 0 || n == 0)

return 0;

if (memo[m][n] > 0)

return memo[m][n];

if (s1.charAt(m - 1) == s2.charAt(n - 1))

memo[m][n] = 1 + lcs(s1, s2, m - 1, n - 1);

else

memo[m][n] = Math.max(lcs(s1, s2, m, n - 1), lcs(s1, s2, m - 1, n));

return memo[m][n];

}

}