1. Edit Distance

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

**Example 1:**

Input: word1 = "horse", word2 = "ros"  
Output: 3  
Explanation:   
horse -> rorse (replace 'h' with 'r')  
rorse -> rose (remove 'r')  
rose -> ros (remove 'e')

**Example 2:**

Input: word1 = "intention", word2 = "execution"  
Output: 5  
Explanation:   
intention -> inention (remove 't')  
inention -> enention (replace 'i' with 'e')  
enention -> exention (replace 'n' with 'x')  
exention -> exection (replace 'n' with 'c')  
exection -> execution (insert 'u')

**解**

动态规划。假设 dp[i][j]表示word1前i个字符变为word2前j个字符的最小编辑距离

初始条件：

dp[0][j] : 空串变为长度为 j 的字符串，插入j个字符

dp[i][0] : 长度为i的字符串变为空串，删除i个字符

递归方程：

dp[i][j] :

* dp[i-1][j] : 增加字符word1[i]
* dp[i][j-1] : 删除字符word2[j]
* dp[i-1][j-1] :
  + word1[i] == word2[j] : 结果为dp[i-1][j-1]
  + 否则，结果为dp[i-1][j-1]+1

根据dp数组更新规则，dp[i][j]仅与dp[i-1][j]、dp[i][j-1]、dp[i-1][j-1]有关， 可以将二维数组压缩为一维数组

class Solution {  
public:  
 int minDistance(string word1, string word2) {  
 int len1 = word1.size(), len2 = word2.size();  
 int dp[len2+1];  
 for(int i = 0; i <= len2; ++i)dp[i] = i;  
 for(int i = 1; i <= len1; ++i){  
 int tmp1 = dp[0], tmp2;  
 dp[0] = i;  
 for(int j = 1; j <= len2; ++j){  
 tmp2 = dp[j];  
 dp[j] = min(min(dp[j]+1, dp[j-1]+1), word1[i-1] == word2[j-1] ? tmp1 : tmp1 + 1);  
 tmp1 = tmp2;  
 }  
 }  
 return dp[len2];  
 }  
};