# Levenshtein Distance

Levenshtein Distance is the edit distance between two strings. We can add, delete, replace characters from string A to string B. We call the minimum operation times transferring from A to B as the edit distance. For example, string1 = “ABC”, string2 = “BCD”. For string1, we delete A, then add D at end, it will be same as string2. So the edit distance is 2.

For this issue, we use DP to solve it. For string1 and string2, we define dp[i][j] as edit distance ends at string1[i] and string2[j]. Give an example, string1 = “abcde”, string2 = ‘ghacee’

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Empty | g | h | a | c | e | e |
| Empty | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| a | 1 | 1 | 2 | 2 | 3 | 4 | 5 |
| b | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| c | 3 | 3 | 3 | 3 | 3 | 4 | 5 |
| d | 4 | 4 | 4 | 4 | 4 | 4 | 5 |
| e | 5 | 5 | 5 | 5 | 5 | 4 | 4 |

Dp[i][j]= min{

1) Dp[i][j-1] + 1,

2) Dp[i-1][j] + 1,

3) Dp[i-1][j-1] + (string1[i] != string2[j])

}

Explanation：字符串A("xyzab")和字符串B("axyzc")，问至少经过多少步操作可以把A变成B。

我们还是从两个字符串的最后一个字符来考察即'b'和'c'。显然二者不相同，那么我们有以下三种处理办法：  
(1)增加：在A末尾增加一个'c'，那么A变成了"xyzabc"，B仍然是"axyzc"，由于此时末尾字符相同了，那么就变成了比较"xyzab"和"axyz"的距离，即d(xyzab,axyzc) = d(xyzab,axyz) + 1。可以写成d(i,j) = d(i,j - 1) + 1。表示下次比较的字符串B的长度减少了1，而加1表示当前进行了一次字符的操作。

(2)删除：删除A末尾的字符'b'，考察A剩下的部分与B的距离。即d(xyzab,axyzc) = d(xyza,axyzc) + 1。可以写成d(i,j) = d(i - 1,j) + 1。表示下次比较的字符串A的长度减少了1。

(3)替换：把A末尾的字符替换成'c'，这样就与B的末尾字符一样了，那么接下来就要考察出了末尾'c'部分的字符，即d(xyzab,axyzc) = d(xyza,axyz) + 1。写成d(i,j) = d(i -1,j-1) + 1表示字符串A和B的长度均减少了1。