经典DP之编辑距离问题详解

**概念**

字符串的编辑距离，又称为Levenshtein距离，由俄罗斯的数学家Vladimir Levenshtein在1965年提出。是指利用字符操作，把字符串A转换成字符串B所需要的最少操作数。其中，字符操作包括：

* 删除一个字符     a) Insert a character
* 插入一个字符     b) Delete a character
* 修改一个字符     c) Replace a character

例如对于字符串"if"和"iff"，可以通过插入一个'f'或者删除一个'f'来达到目的。

  一般来说，两个字符串的编辑距离越小，则它们越相似。如果两个字符串相等，则它们的编辑距离（为了方便，本文后续出现的“距离”，如果没有特别说明，则默认为“编辑距离”）为0（不需要任何操作）。不难分析出，两个字符串的编辑距离肯定不超过它们的最大长度（可以通过先把短串的每一位都修改成长串对应位置的字符，然后插入长串中的剩下字符）。

这个题目和LCS在思想上有一点类似的地方。

**问题描述**

给定两个字符串A和B，求字符串A至少经过多少步字符操作变成字符串B。  
**问题分析(充分体现了求解难题的思想：从特殊到一般)**

**把问题的三种求解情形全部列出来进行分析。**

1）首先考虑A串的第一个字符

  假设存在两个字符串A和B，他们的长度分别是lenA和lenB。首先考虑第一个字符，如果他们是一样的，所以只需要计算A[2...lenA]和B[2...lenB]之间的距离即可。那么如果两个字符串的第一个字符不一样怎么办？可以考虑把第一个字符变成一样的（这里假设从A串变成B串）：

* 修改A串的第一个字符成B串的第一个字符，之后仅需要计算A[2...lenA]和B[2...lenB]的距离即可；
* 删除A串的第一个字符，之后仅需要计算A[2...lenA]和B[1...lenB]的距离即可；
* 把B串的第一个字符插入到A串的第一个字符之前，之后仅需要计算A[1...lenA]和B[2...lenB]的距离即可。

2）接下来考虑A串的第i个字符和B串的第j个字符。

  我们这个时候不考虑A的前i-1字符和B串的第j-1个字符。如果A串的第i个字符和B串的第j个字符相等，即A[i]=B[j]，则只需要计算A[i...lenA]和B[j...lenB]之间的距离即可。如果不想等，则：

* 修改A串的第i个字符成B串的第j个字符，之后仅需要计算A[i+1...lenA]和B[j+1...lenB]的距离即可；
* 删除A串的第i个字符，之后仅需要计算A[i+1...lenA]和B[j...lenB]的距离即可；
* 把B串的第j个字符插入到A串的第i个字符之前，之后仅需要计算A[i...lenA]和B[j+1...lenB]的距离即可。

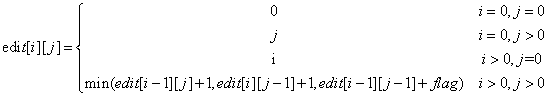
  写到这里，可以看出第一个字符情况和后面任一字符情况处理均一样，自然会想到用递归求解或者动态规划求解，由于用递归会产生很多重复解，所以用动态规划。

**建动态规划方程**

  用edit[i][j]表示A串和B串的编辑距离。edit[i][j]表示A串从第0个字符开始到第i个字符和B串从第0个字符开始到第j个字符，这两个字串的编辑距离。字符串的下标从1开始。

  dis[0][0]表示word1和word2都为空的时候，此时他们的Edit Distance为0。很明显可以得出的，dis[0][j]就是word1为空，word2长度为j的情况，此时他们的Edit Distance为j，也就是从空，添加j个字符转换成word2的最小Edit Distance为j；同理dis[i][0]就是，word1长度为i，word2为空时，word1需要删除i个字符才能转换成空，所以转换成word2的最小Edit Distance为i。

  则从上面的分析，不难推导出动态规划方程：

，其中http://images2015.cnblogs.com/blog/452750/201512/452750-20151206203725956-835672886.gif

上式中的min（）函数中的三个部分，对应三种字符操作方式：

edit[i-1][j]+1相当于给word2的最后插入了word1的最后的字符，插入操作使得edit+1，之后计算edit[i-1][j]；

edit[i][j-1]+1相当于将word2的最后字符删除，删除操作edit+1，之后计算edit[i][j-1];

edit[i-1][j-1]+flag相当于通过将word2的最后一个字符替换为word1的最后一个字符。flag标记替换的有效次数。

[**算法**](http://lib.csdn.net/base/datastructure)**分析：**

  也就是说，就是将一个字符串变成另外一个字符串所用的最少操作数，每次只能增加、删除或者替换一个字符。  
  首先我们令word1和word2分别为：michaelab和michaelxy（为了理解简单，我们假设word1和word2字符长度是一样的），dis[i][j]作为word1和word2之间的Edit Distance，我们要做的就是求出michaelx到michaely的最小steps。

  首先解释下dis[i][j]：它是指word1[i]和word2[j]的Edit Distance。dis[0][0]表示word1和word2都为空的时候，此时他们的Edit Distance为0。很明显可以得出的，dis[0][j]就是word1为空，word2长度为j的情况，此时他们的Edit Distance为j，也就是从空，添加j个字符转换成word2的最小Edit Distance为j；同理dis[i][0]就是，word1长度为i，word2为空时，word1需要删除i个字符才能转换成空，所以转换成word2的最小Edit Distance为i。下面及时初始化代码：

       for (int i = 0; i < row; i++) dis[i][0] = i;  
       for (int j = 0; j < col; j++) dis[0][j] = j;

下面来分析下题目规定的三个操作：添加，删除，替换。

假设word1[i]和word2[j](此处i = j)分别为：michaelab和michaelxy

如果b==y,

那么：dis[i][j] = dis[i-1][j-1]。

如果b!=y，

那么：添加：也就是在michaelab后面添加一个y，那么word1就变成了michaelaby，

此时 dis[i][j] = 1 + dis[i][j-1]；

上式中，1代表刚刚的添加操作，添加操作后，word1变成michaelaby，word2为michaelxy。

dis[i][j-1]代表从word1[i]转换成word2[j-1]的最小Edit Distance，也就是michaelab转换成michaelx的最小

Edit Distance，由于两个字符串尾部的y==y，所以只需要将michaelab变成michaelx就可以了，而他们之间的最

小Edit Distance就是dis[i][j-1]。

删除：也就是将michaelab后面的b删除，那么word1就变成了michaela，此时dis[i][j] = 1 + dis[i-1][j]；

上式中，1代表刚刚的删除操作，删除操作后，word1变成michaela，word2为michaelxy。dis[i-1][j]代表从

word[i-1]转换成word[j]的最小Edit Distance，也就是michaela转换成michaelxy的最小Edit Distance，所以

只需要将michaela变成michaelxy就可以了，而他们之间的最小Edit Distance就是dis[i-1][j]。

替换：也就是将michaelab后面的b替换成y，那么word1就变成了michaelay，此时dis[i][j] = 1 + dis[i-1][j-1]；

上式中，1代表刚刚的替换操作，替换操作后，word1变成michaelay，word2为michaelxy。dis[i-1][j-1]代表从

word[i-1]转换成word[j-1]的最小Edit Distance，也即是michaelay转换成michaelxy的最小Edit Distance，由

于两个字符串尾部的y==y，所以只需要将michaela变成michaelx就可以了，而他们之间的最小Edit Distance就是

dis[i-1][j-1]。

举例：

比如要计算cafe和coffee的编辑距离。cafe→caffe→coffe→coffee

先创建一个6×8的表（cafe长度为4，coffee长度为6，各加2）

（1）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | c | o | f | f | e | e |
|  |  |  |  |  |  |  |  |
| c |  |  |  |  |  |  |  |
| a |  |  |  |  |  |  |  |
| f |  |  |  |  |  |  |  |
| e |  |  |  |  |  | 表 | 1 |

接着，在如下位置填入数字（表2）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | c | o | f | f | e | e |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| c | 1 |  |  |  |  |  |  |
| a | 2 |  |  |  |  |  |  |
| f | 3 |  |  |  |  |  |  |
| e | 4 |  |  |  |  | 表 | 2 |

从3,3格开始，开始计算。取以下三个值的最小值：

* 如果最上方的字符等于最左方的字符，则为左上方的数字。否则为左上方的数字+1。（对于3,3来说为0）
* 左方数字+1（对于3,3格来说为2）
* 上方数字+1（对于3,3格来说为2）

因此为格3,3为0（表3）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | c | o | f | f | e | e |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| c | 1 | 0 |  |  |  |  |  |
| a | 2 |  |  |  |  |  |  |
| f | 3 |  |  |  |  |  |  |
| e | 4 |  |  |  |  | 表 | 3 |

循环操作，推出下表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | c | o | f | f | e | e |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| c | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| a | 2 | 1 | 1 | 2 | 3 | 4 | 5 |
| f | 3 | 2 | 2 | 1 | 2 | 3 | 4 |
| e | 4 | 3 | 3 | 2 | 2 | 2 | 3 |

取右下角，得编辑距离为3

AC代码：

//51nod 1183编辑距离(Levenshtein距离)

#include<bits/stdc++.h>

using namespace std;

int f[1002][1002];//记录a从0到i和b从0到j的编辑距离。

int main()

{

char a[1002];

char b[1002];

int lena,lenb,i,j;

scanf("%s%s",a,b);

lena = strlen(a);

lenb = strlen(b);

memset(f,0,sizeof(f));

for(i=1;i<=lena;i++)//初始化

f[i][0]=i;

for(j=1;j<=lenb;j++)

f[0][j]=j;

for(i=0;i<lena;i++)

{

for(j=0;j<lenb;j++)

{

f[i+1][j+1]=min(min(f[i][j+1]+1,f[i+1][j]+1),f[i][j]+(a[i]==b[j]?0:1));

}

}

printf("%d\n",f[lena][lenb]);

return 0;

}