**[Levenshtein Distance (编辑距离) 算法详解](http://www.cnblogs.com/pandora/archive/2009/12/20/Levenshtein_distance.html)**

今天看到[万仓一黍](http://www.cnblogs.com/grenet/)发的[计算字符串的相似度（VB2005）](http://www.cnblogs.com/grenet/archive/2009/12/17/1626649.html)，感兴趣研究了一下，看了半天也没搞懂（惭愧惭愧），上网google了一下，大多也都是实现，没什么解释，几经波折还是在wikipedia找到了详细的算法解释，首先还是C#实现代码：

http://www.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gifusing System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Diagnostics;

namespace WordCompare

{

class Program

{

static void Main(string[] args)

{

var fromString = "English is a West Germanic language that developed in England during the Anglo-Saxon era. As a result of the military, economic, scientific, political, and cultural influence of the British Empire during the 18th, 19th, and early 20th centuries, and of the United States since the mid 20th century,[7][8][9][10] it has become the lingua franca in many parts of the world.[11][12] It is used extensively as a second language and as an official language in Commonwealth countries and many international organisations.English is a West Germanic language that developed in England during the Anglo-Saxon era. As a result of the military, economic, scientific, political, and cultural influence of the British Empire during the 18th, 19th, and early 20th centuries, and of the United States since the mid 20th century,[7][8][9][10] it has become the lingua franca in many parts of the world.[11][12] It is used extensively as a second language and as an official language in Commonwealth countries and many international organisations.English is a West Germanic language that developed in England during the Anglo-Saxon era. As a result of the military, economic, scientific, political, and cultural influence of the British Empire during the 18th, 19th, and early 20th centuries, and of the United States since the mid 20th century,[7][8][9][10] it has become the lingua franca in many parts of the world.[11][12] It is used extensively as a second language and as an official language in Commonwealth countries and many international organisations.";

var toString = "English is West Germanic language that developed in England during the Anglo-Saxon era. As a result of the military, economic, scientific, political, and cultural influence of the British Empire during the 18th, 19th, and early 20th centuries, and of the United States since the mid 20th century,[7][8][9][10] it has become the lingua franca in many parts of the world.[11][12] It is used extensively as a second language and as an official language in Commonwealth countries and many international organisations.English is a West Germanic language that developed in England during the Anglo-Saxon era. As a result of the military, economic, scientific, political, and cultural influence of the British Empire during the 18th, 19th, and early 20th centuries, and of the United States since the mid 20th century,[7][8][9][10] it has become the lingua franca in many parts of the world.[11][12] It is used extensively as a second language and as an official language in Commonwealth countries and many international organisations.English is a West Germanic language that developed in England during the Anglo-Saxon era. As a result of the military, economic, scientific, political, and cultural influence of the British Empire during the 18th, 19th, and early 20th centuries, and of the United States since the mid 20th century,[7][8][9][10] it has become the lingua franca in many parts of the world.[11][12] It is used extensively as a second language and as an official language in Commonwealth countries and many international organisations.a";

Stopwatch watch = new Stopwatch();

watch.Start();

var result = CompareStrings(fromString, toString);

watch.Stop();

Console.WriteLine("The result is {0}, spent {1} milliseconds.", result, watch.ElapsedMilliseconds);

}

private static int CompareStrings(string fromString, string toString)

{

var fLength = fromString.Length;

var tLength = toString.Length;

// pre verify the simplest condition

if (fLength == 0)

{

return tLength;

}

if (tLength == 0)

{

return fLength;

}

// prepare the martix

var martix = new int[fLength + 1, tLength + 1];

for (int i = 0; i <= fLength; i++)

{

martix[i, 0] = i;

}

for (int j = 0; j <= tLength; j++)

{

martix[0, j] = j;

}

// compare the chars

for (int i = 1; i <= fLength; i++)

{

var tempF = fromString[i - 1];

var cost = 0;

for (int j = 1; j <= tLength; j++)

{

var tempT = toString[j - 1];

if (tempT == tempF)

{

cost = 0;

}

else

{

cost = 1;

}

var valueAbove = martix[i - 1, j] + 1;

var valueLeft = martix[i, j - 1] + 1;

// left corner

var valueDiag = martix[i - 1, j - 1] + cost;

// find the minimum from the three vars above

var cellValue = valueAbove < valueLeft ? (valueDiag < valueAbove ? valueDiag : valueAbove) : (valueDiag < valueLeft ? valueDiag : valueLeft);

martix[i, j] = cellValue;

}

}

var result = martix[fLength, tLength];

return result;

}

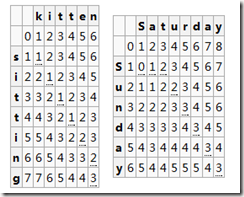
}

}

简单的说就是从C++的代码抄来的，哈哈~~

速度真的不错，正确性应该是毋庸质疑，但我就是不能理解为什么这个程序可以得到正确的结果。

算法如下表：

[](http://images.cnblogs.com/cnblogs_com/pandora/WindowsLiveWriter/LevenshteinDistance_11B1A/image_2.png)

注意，有下划线的就是每个循环得到的结果。

**算法证明**

这个算法计算的是将s[1…i]转换为t[1…j]（例如将kitten转换为sitting）所需最少的操作数（也就是所谓的编辑距离），这个操作数被保存在d[i,j]（d代表的就是上图所示的二维数组）中。

* 在第一行与第一列肯定是正确的，这也很好理解，例如我们将kitten转换为空字符串，我们需要进行的操作数为kitten的长度（所进行的操作为将kitten所有的字符丢弃）。
* 我们对字符可能进行的操作有三种：
  + 如果我们可以使用k个操作数把s[1…i]转换为t[1…j-1]，我们只需要把t[j]加在最后面就能将s[1…i]转换为t[1…j]，操作数为k+1
  + 如果我们可以使用k个操作数把s[1…i-1]转换为t[1…j]，我们只需要把s[i]从最后删除就可以完成转换，操作数为k+1
  + 如果我们可以使用k个操作数把s[1…i-1]转换为t[1…j-1]，我们只需要在需要的情况下（s[i] != t[j]）把s[i]替换为t[j]，所需的操作数为k+cost（cost代表是否需要转换，如果s[i]==t[j]，则cost为0，否则为1）。
* 将s[1…n]转换为t[1…m]当然需要将所有的s转换为所有的t，所以，d[n,m]（表格的右下角）就是我们所需的结果。

这个证明过程只能证明我们可以得到结果，但并没有证明结果是最小的（即我们得到的是最少的转换步骤）。所以我们引进了另外一个算法，即d[i,j]保存的是上述三种操作中操作数最小的一种。这就保证了我们获得的结果是最小的操作数（可使用[argument by contradiction](http://en.wikipedia.org/wiki/Reductio_ad_absurdum)进行证明，离题太远，忽略。。）

**可能进行的改进**

* 现在的算法复杂度为O(mn)，可以将其改进为O(m)。因为这个算法只需要上一行和当前行被存储下来就可以了。
* 如果需要重现转换步骤，我们可以把每一步的位置和所进行的操作保存下来，进行重现。
* 如果我们只需要比较转换步骤是否小于一个特定常数k，那么只计算高宽宽为2k+1的矩形就可以了，这样的话，算法复杂度可简化为*O*(*kl*)，*l*代表参加对比的最短string的长度。
* 我们可以对三种操作（添加，删除，替换）给予不同的权值（当前算法均假设为1，我们可以设添加为1，删除为0，替换为2之类的），来细化我们的对比。
* 如果我们将第一行的所有cell初始化为0，则此算法可以用作模糊字符查询。我们可以得到最匹配此字符串的字符串的最后一个字符的位置（index number），如果我们需要此字符串的起始位置，我们则需要存储各个操作的步骤，然后通过算法计算出字符串的起始位置。
* 这个算法不支持并行计算，在处理超大字符串的时候会无法利用到并行计算的好处。但我们也可以并行的计算cost values（两个相同位置的字符是否相等），然后通过此算法来进行整体计算。
* 如果只检查对角线而不是检查整行，并且使用延迟验证（lazy evaluation），此算法的时间复杂度可优化为O(m(1+d))（d代表结果）。这在两个字符串非常相似的情况下可以使对比速度速度大为增加。

**感言**

没什么特别想说的，就是觉得自己太笨了。这么简单的问题搞了半天才弄明白。。。惭愧。。惭愧。。。

不过觉得这个算法还是很有用处的，特别是在模糊查询方面的应用。总算是大概知道模糊查询是怎么处理的了哈。。