计算字符串的相似度

许多程序会大量使用字符串。对于不同的字符串，我们希望能够有办法判断其相似程度。我们定义了一套操作方法来把两个不相同的字符串变得相同，具体的操作方法为：

1. 修改一个字符（如把“*a*”替换为“*b*”）。

2. 增加一个字符（如把“*abdd*”变为“*aebdd*”）。

3. 删除一个字符（如把“*travelling*”变为“*traveling*”）。

比如，对于“*abcdefg*”和“*abcdef*”两个字符串来说，我们认为可以通过增加/减少一个“*g*”的方式来达到目的。上面的两种方案，都仅需要一次操作。把这个操作所需要的次数定义为两个字符串的距离，而相似度等于“距离+1”的倒数。也就是说，“*abcdefg*”和“*abcdef*”的距离为 1，相似度为 1 / 2 = 0.5。

给定任意两个字符串，你是否能写出一个算法来计算出它们的相似度呢？

分析与解法

不难看出，两个字符串的距离肯定不超过它们的长度之和（我们可以通过删除操作把两个串都转化为空串）。虽然这个结论对结果没有帮助，但至少可以知道，任意两个字符串的距离都是有限的。

我们还是应该集中考虑如何才能把这个问题转化成规模较小的同样的问题。如果有两个串 *A=xabcdae* 和 *B=xfdfa* ,它们的第一个字符是相同的 ,只要计算 *A*[2, …, 7] = *abcdae* 和 *B*[2, …, 5] = *fdfa* 的距离就可以了。但是如果两个串的第一个字符不相同，那么可以进行如下的操作（len*A* 和 len*B* 分别是 *A* 串和 *B* 串的长度）：

1．删除*A*串的第一个字符，然后计算*A*[2, …, len*A*]和*B*[1, …, len*B*]的距离。

2．删除*B*串的第一个字符，然后计算*A*[1, …, len*A*]和*B*[2, …, len*B*]的距离。

3．修改*A*串的第一个字符为*B*串的第一个字符，然后计算*A*[2, …, len*A*]和*B*[2, …, len*B*]的距离。

4．修改*B*串的第一个字符为*A*串的第一个字符，然后计算*A*[2, …, len*A*]和*B*[2, …, len*B*]的距离。

5．增加*B*串的第一个字符到*A*串的第一个字符之前，然后计算*A*[1, …, len*A*]和*B*[2, …, len*B*]的距离。

6．增加*A*串的第一个字符到*B*串的第一个字符之前，然后计算*A*[2, …, len*A*]和*B*[1, …, len*B*]的距离。

在这个题目中，我们并不在乎两个字符串变得相等之后的字符串是怎样的。所以，可以将上面 6 个操作合并为：

1. 一步操作之后，再将$A[2, \cdots, \text{len}A]$和$B[1, \cdots, \text{len}B]$变成相同字符串。

2. 一步操作之后，再将$A[1, \cdots, \text{len}A]$和$B[2, \cdots, \text{len}B]$变成相同字符串。

3. 一步操作之后，再将$A[2, \cdots, \text{len}A]$和$B[2, \cdots, \text{len}B]$变成相同字符串。

这样，很快就可以完成一个递归程序：

代码清单 3-6

```
Int CalculateStringDistance(string strA, int pABegin, int pAEnd, string strB,
  int pBBegin, int pBEnd)
{
    if(pABegin > pAEnd)
    {
        if(pBBegin > pBEnd)
            return 0;
        else
            return pBEnd – pBBegin + 1;
    }

    if(pBBegin > pBEnd)
    {
        if(pABegin > pAEnd)
            return 0;
        else
            return pAEnd – pABegin + 1;
    }

    if(strA[pABegin] == strB[pBBegin])
    {
        return CalculateStringDistance(strA, pABegin + 1, pAEnd, strB,
         pBBegin + 1, pBEnd);
    }
    else
    {
        int t1 = CalculateStringDistance(strA, pABegin + 1, pAEnd, strB,
         pBBegin + 2, pBEnd);
        int t2 = CalculateStringDistance(strA, pABegin + 2, pAEnd, strB,
         pBBegin + 1, pBEnd);
        int t3 = CalculateStringDistance(strA, pABegin + 2, pAEnd, strB,
         pBBegin + 2, pBEnd);
        return minValue(t1,t2,t3) + 1;
    }
}
```

上面的递归程序，有什么地方需要改进呢？在递归的过程中，有些数据被重复计算了。比如，如果开始我们调用 `CalculateStringDistance(strA, 1, 2, strB, 1, 2)`，图 3-4 是部分展开的递归调用：

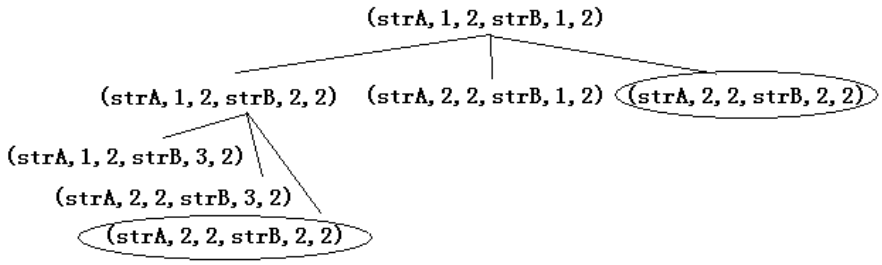

图 3-4

可以看到，圈中的两个子问题被重复计算了。为了避免这种不必要的重复计算，可以把子问题计算后的解存储起来。如何修改递归程序呢？这个问题就留给读者自己完成吧！