# 实验一

## 关于LCS算法的个人思想

LCS最长公共子序列，就是任意取2个X,Y序列（也可以考虑为向量或字符串），将X,Y中两个序列中都相同的元素取出来构成一个新的序列，但是最长公共子序列不唯一，最长公共子序列长度是一定的，所以我们要先把问题简化，求出两个序列的最长公共子序列长度。

一个已知序列都有Xi=<x1,x2...,xi>(1≤i≤m)，且我们将每个序列的最后一个字符设为xm，同理Yj=<y1,y2...,yj>(1≤j≤n),最后一个字符设为yn，这2个序列将构成一个新的子序列Zk=<z1,z2…zk>。zk则为ZK序列的最后一个字符，Zk∈LCS（X,Y）。

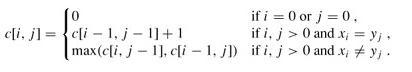
一定要特别注意：Xi,Yj,Zk是表示整个序列,xm,yn,zk表示序列中的每个元素字符，ZK和zk都是未知的！！！！

若xm=yn（最后一个字符相同），我们用反证法证明：该字符必是X与Y的任一最长公共子序列Z（设长度为k）的最后一个字符，即有zk = xm = yn 且显然有Zk-1∈LCS(Xm-1 , Yn-1)即Z的前缀Zk-1是Xm-1与Yn-1的最长公共子序列。此时，问题化归成求Xm-1与Yn-1的LCS（LCS(X , Y)的长度等于LCS(Xm-1 , Yn-1)的长度加1）。

若xm≠yn，用反证法证明：要么Z∈LCS(Xm-1, Y)，要么Z∈LCS(X , Yn-1)。由于zk≠xm与zk≠yn其中至少有一个必成立，若zk≠xm则有Z∈LCS(Xm-1 , Y)，同理，若zk≠yn 则有Z∈LCS(X , Yn-1)。其实在这里之前我看到的时候也不是特别懂，后面思考了一下，可以这么理解，若xm≠yn两个序列的最后一个字符都不相等，那Zk序列的最后一个字符是如何得到的呢？那只有两种可能，要么zk≠xm，要么zk≠yn，如果zk≠xm就说明zk跟X序列的最后一个字符就没关系了，就要从X序列中Xm-1（为xm的前一个字符xm-1，因为xm跟zk没有关系，所以要从它前面一个字符找起）和Y序列的最后一个字符开始找起，所以就有了Z∈LCS(Xm-1 , Y)，同理这样的解释也可以用在Y序列中得出，若zk≠yn 则有Z∈LCS(X , Yn-1)。

此时，问题化归成求Xm-1与Y的LCS及X与Yn-1的LCS。LCS(X , Y)的长度为：max{LCS(Xm-1 , Y)的长度, LCS(X , Yn-1)的长度}（就是求最大值，只要求出两个长度的最大值就是LCS的长度）。

这样就可以由此得出下面的推导：

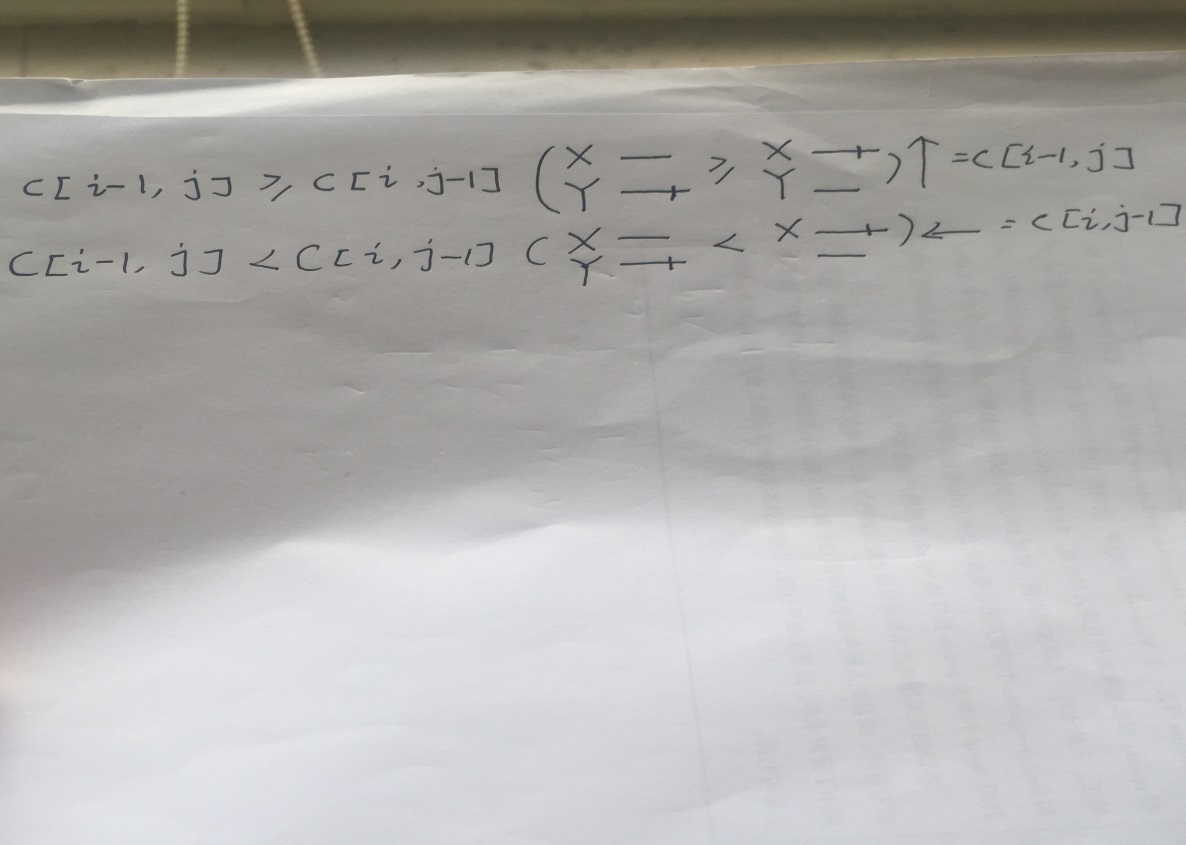


设一个C[i,j] 记录序列Xi和Yj的最长公共子序列的长度，当i=0或j=0时，空序列是Xi和Yj的最长公共子序列，故c[i,j]=0。如果xi=yj那么就可以得出c[i,j]=c[i-1,j-1]+1。

这种情况就是最复杂的第三种情况

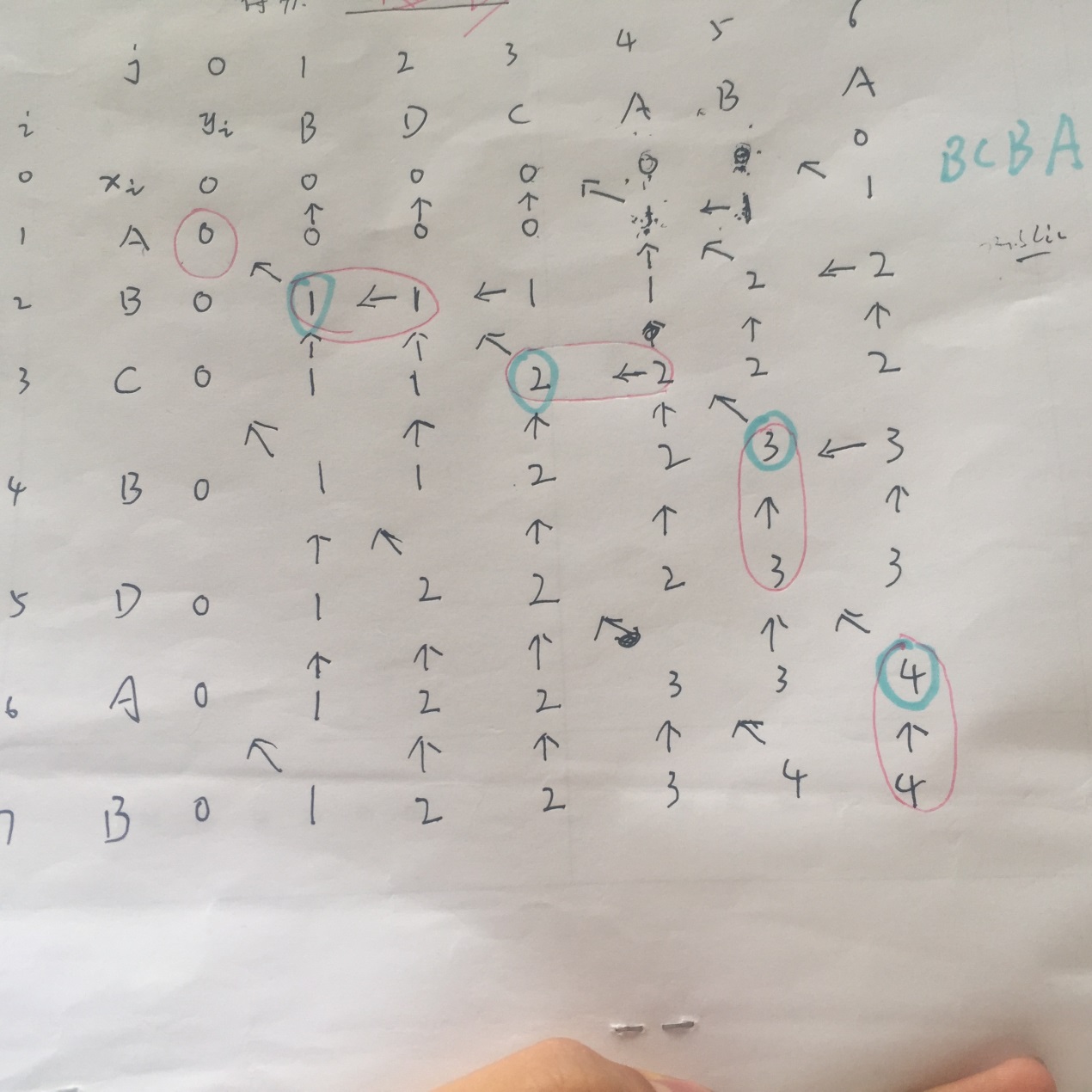
zk≠xmZ∈LCS(Xm-1 , Y)

zk≠ynZ∈LCS(X , Yn-1)



c[i,j]可以表示一个矩阵，根据第三种max（c[i-1,j],c[i,j-1]）这种情况来判断里面的值哪个比较大，如果大的则为LCS。可以得出上图的推导。如果c[i-1,j]≥c[i,j-1]=c[i-1,j]则在矩阵中向上回溯↑（i和j可以在矩阵中表示行和列，向上回溯就是表示j不变,i-1）。如果c[i-1,j]＜c[i,j-1]=c[i,j-1]则在矩阵中向左回溯←（同理上面向左回溯就是表示i不变，j-1）.

后面根据自己的推导画出矩阵图：



如果xi=yj =＞得出c[i-1]=c[j-1],则往左上回溯。优先级是先左上>上>左，至于为什么这样定义的优先级是跟这个算法的作者自己设置定义的优先级有关，如果自己在代码中实现，可以改变上和左的优先级，但是左上的优先级必须的最优先的。如图将所有矩阵中的元素都用上面三种回溯箭头标出，则不难找出向左上回溯并且箭头连续的一条线性回溯路，得出LCS为BCBA。

以上就是我对LCS算法的理解和观点，至于改进的算法只是在这算法的基础上节省了O（m\*n）和O（m+n）的时间复杂度。

代码实现：

Procedure LCS\_LENGTH(X, Y);

begin

m:=length[X];

  n:=length[Y];

  for i:=1 to m do c[i, 0]:=0;   //因为i=0或j=0 c[i,j]数组长度为0，之所以for循环中从1开始到m

  for j:=1 to n do c[0, j]:=0;

  for i:=1 to m do   //用双重for循环遍历X和Y的序列的元素，先执行内层循环Y序列

    for j:=1 to n do

      if x[i]=y[j] then //如果X序列和Y序列的最后一个元素相等

begin

c[i, j]:=c[i - 1, j - 1]+1;   //LCS长度+1

          b[i, j]:="↖";   //根据左上角元素自动+1

        end

      else if c[i - 1, j]≥c[i, j - 1] then //如果X和Y序列最后一个元素不相等

begin

c[i, j]:=c[i - 1, j];   //LCS（Xm-1,Y）

          b[i, j]:="↑";  //根据上方元素表示相等

        end

      else

        begin

c[i, j]:=c[i, j - 1];   //LCS(X,Ym-1)

          b[i, j]:="←"  //根据左方元素表示相等

        end;

  return(c, b);

end;

根据这个算法代码还需要进行代码的修改和编写，所以以上就是我对算法的个人感悟和思想。