数据结构与算法题解（9）：最长公共子序列和最长公共子串

一、最长公共子序列（LCS）

求最长公共子序列的数目，注意这里的子序列可以不是连续序列，务必问清楚题意。求『最长』类的题目往往与动态规划有点关系，这里是两个字符串，故应为双序列动态规划。

这道题的状态很容易找，不妨先试试以$f[i][j]$表示字符串 A 的前 i 位和字符串 B 的前 j 位的最长公共子序列数目，那么接下来试试寻找其状态转移方程。

从实际例子ABCD和EDCA出发，首先初始化f的长度为字符串长度加1，那么有$f[0][0] = 0, f[0][*] = 0, f[*][0] = 0$,最后应该返回$f[lenA][lenB]$. 即 f 中索引与字符串索引对应(字符串索引从1开始算起)，那么在A 的第一个字符与 B 的第一个字符相等时，$f[1][1] = 1 + f[0][0]$, 否则$f[1][1] = max(f[0][1], f[1][0])$。

推而广之，也就意味着若$A[i] == B[j]$, 则分别去掉这两个字符后，原 LCS 数目减一，那为什么一定是1而不是0或者2呢？因为不管公共子序列是以哪个字符结尾，在$A[i] == B[j]$时 LCS 最多只能增加1. 而在$A[i] != B[j]$时，由于A[i] 或者 B[j] 不可能同时出现在最终的 LCS 中，故这个问题可进一步缩小，$f[i][j] = max(f[i-1][j], f[i][j-1])$. 需要注意的是这种状态转移方程只依赖最终的 LCS 数目，而不依赖于公共子序列到底是以第几个索引结束。

$$c(i,j)=\begin{cases} 0 & 当i=0或者j=0 \\ c(i-1,j-1)+1 & 当i>0,j>0,且x_i=y_j \\ \max\{c(i-1,j),c(i,j-1)\} & 当i>0,j>0,且x_i\neq y_j \end{cases}$$


```
public class Demo {
    public static int longestCommonSubsequence(String A, String B) {
        if (A == null || A.length() == 0) return 0;
        if (B == null || B.length() == 0) return 0;

        int lenA = A.length();
        int lenB = B.length();
        int[][] lcs = new int[1 + lenA][1 + lenB];
```

```
        for (int i = 1; i < 1 + lenA; i++) {
            for (int j = 1; j < 1 + lenB; j++) {
                if (A.charAt(i - 1) == B.charAt(j - 1)) {
                    lcs[i][j] = 1 + lcs[i - 1][j - 1];
                } else {
                    lcs[i][j] = Math.max(lcs[i - 1][j], lcs[i][j - 1]);
                }
            }
        }
        return lcs[lenA][lenB];
    }

    public static void main(String[] args) {
        String source = "zhanghua";
        String target = "zhanghau";
        System.out.println("longestCommonSubsequence=" + longestCommonSubsequence(s
ource, target));
    }
}
```

二、最长公共子串

2.1 简单考虑

可以使用两根指针索引分别指向两个字符串的当前遍历位置，若遇到相等的字符时则同时向后移动一位。

```
public class Demo {
    public static int longestCommonSubstring(String A, String B) {
        if (A == null || A.length() == 0) return 0;
        if (B == null || B.length() == 0) return 0;
        int lenA = A.length();
        int lenB = B.length();
        int lcs = 0, lcs_temp = 0;
        for (int i = 0; i < lenA; ++i) {
            for (int j = 0; j < lenB; ++j) {
                lcs_temp = 0;
                while ((i + lcs_temp < lenA) &&
                       (j + lcs_temp < lenB) &&
                       (A.charAt(i + lcs_temp) == B.charAt(j + lcs_temp)))
                {
                    ++lcs_temp;
                }
```

```
                if (lcs_temp > lcs) {
                    lcs = lcs_temp;
                }
            }
        }
        return lcs;
    }
    public static void main(String[] args) {
        String source = "zhanghua";
        String target = "zhanghau";
        System.out.println("longestCommonString=" + longestCommonSubstring(source,
target));
    }
}
```

2.2 动态规划

把$D[i][j]$定义为：两个string的前i个和前j个字符串，尾部连到最后的最长子串。

然后$D[i][j] =$

1. $i = 0 || j = 0 : 0$
2. $s1.char[i-1] = s2.char[j-1] ? D[i-1][j-1] + 1 : 0;$

另外，创建一个max的缓存，不段更新即可。

```
public class Demo {
    public static int longestCommonSubstring(String A, String B) {
        if (A == null || A.length() == 0) return 0;
        if (B == null || B.length() == 0) return 0;

        int lenA = A.length();
        int lenB = B.length();
        int[][] D = new int[lenA + 1][lenB + 1];
        int max = 0;

        for (int i = 0; i <= lenA; i++) {
            for (int j = 0; j <= lenB; j++) {
                if (i == 0 || j == 0) {
                    D[i][j] = 0;
                } else {
                    if (A.charAt(i - 1) == B.charAt(j - 1)) {
                        D[i][j] = D[i - 1][j - 1] + 1;
                    } else {
                        D[i][j] = 0;
                    }
```

```
                }
                max = Math.max(max, D[i][j]);
            }
        }
        return max;
    }
    public static void main(String[] args) {
        String source = "zhanghua";
        String target = "zhanghau";
        System.out.println("longestCommonString=" + longestCommonSubstring(source, 
target));
    }
}
```