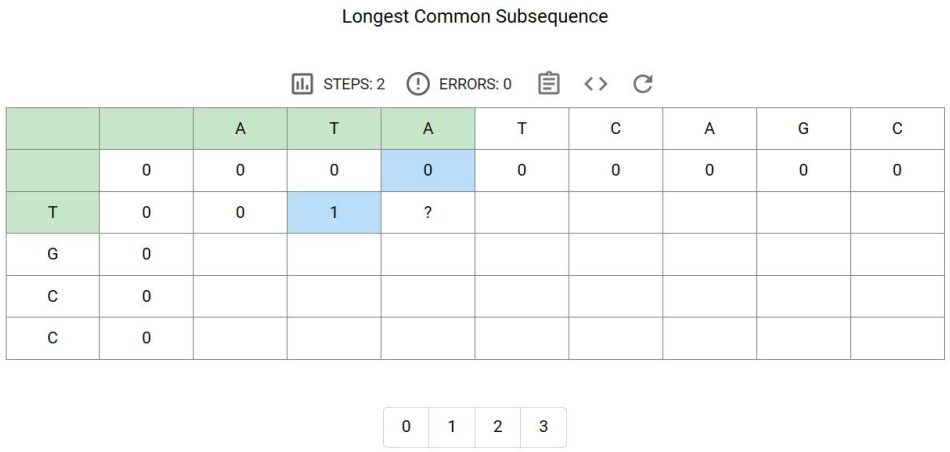
INT102W8_最长公共子序列（动态规划的实例）

定义： 最长公共子序列是一类问题的统称，通常使用动态规划的思想来解决。
达成过程： 我们首先输入一个矩阵，让其横纵坐标分别代表我们的两个字符串，在这之后对矩阵中的元素进行比对，如果元素不相等则在其右上角和前方一个数之间进行比对，选大的更新数组元素，最后矩阵中的最大值就是我们的最长公共子序列的长度，而每一次数值的更新点就是我们的序列。
存储方式： 二维数组
例题： 寻找下图的最长公共子序列



从图中我们可以发现，这里我们首先是对应字符串建立了矩阵并提前空出一列为 0 用来表示最开始没有任何的相同，然后开始遍历，对应相同的地方就 copy 他右上角的值（这里这个值的意思是前一个最长公共子序列的长度，比如这里就 TT 部分就是用 0 到横坐标的 T 的最长子序列长度）添加一，比如这里的 TT，我们就会对应前右上角中的值加一并更新数据，而当数据不相同的时候，我们就会在前一列的数据和上方数据中选取最大值 copy，这里表现就是我们 TA 这里对比 TT 和 0t 的值选大者 copy，重复以上步骤我们就可以得到最后的最长公共子序列长度。如果我们需要路径的话，则需要在执行加一操作的时候记录（推测应该就是加一个 if 判断）。以下是这道题的完整答案，然后我有个网站比较好玩大家可以试一试。
Alchemist (alchemist-al.com)

Longest Common Subsequence

STEPS: 38 ERRORS: 6

		A	T	A	T	C	A	G	C
	0	0	0	0	0	0	0	0	0
T	0	0	1	1	1	1	1	1	1
G	0	0	1	1	1	1	1	2	2
C	0	0	1	1	1	2	2	2	3
C	0	0	1	1	1	2	2	2	3

0 1 2 3

再如下则是我们的伪代码示例

```
PRINT-LCS(b, X, i, j)
1  if i = 0 or j = 0
2     then return
3  if b[i, j] = "↖"
4     then PRINT-LCS(b, X, i − 1, j − 1)
5          print x_i
6  elseif b[i, j] = "↑"
7     then PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

这里我们可以看到他首先是规定了一个方法接受 4 个元素，然后新建二维数组 b，如果 i 或者 j 等于 0（这里代表我们达到序列的起始点）则什么都不做，如果我们的任意元素等于他的左上角，那么我们就会将 i,j 各减少 1,并递归调用该函数并打印该点的值，如果 b【i，j】等于他上方的值，那么我们就会令 i-1 并重复调用，其他情况则是 j-1 然后重复调用。（虽然他教的动态规划但是我看不明白他为什么要用递归，我下面用 JAVA 描述一下）

```
public class LCS {
    public static int findLCS(String a, String b) {
        int m = a.length();
        int n = b.length();
        int[][] table = new int[m + 1][n + 1];

        for (int i = 0; i <= m; i++) {
```

```
            for (int j = 0; j <= n; j++) {
                if (i == 0 || j == 0) {
                    table[i][j] = 0;
                } else if (a.charAt(i - 1) == b.charAt(j - 1)) {
                    table[i][j] = table[i - 1][j - 1] + 1;
                } else {
                    table[i][j] = Math.max(table[i - 1][j], table[i][j - 1]);
                }
            }
        }
        return table[m][n];
    }
```

解析基本就是我之前说的思路，不过用代码具现化一下，值得一提的是或许会有人疑惑为什么矩阵最后的值一定是我的长度，其实很简单，因为我在对应每一行的比对的时候都会扫描他前一个和上方的最大数值，对应每一层都如此操作，自然每一层的最后留下的都是最大的那个值，那我最后的结果必然就是最大的，也就是我们的最长公共子序列。