**实验3-最长公共子序列**

**问题分析**

* **最长公共子序列长度（即动态规划方程）**

**最后一步**

序列和序列的最后一个元素是否相等。

若相等，两序列的最长公共子序列（后用LCS代替）即为和的LCS长度+1即可。

若不等，则又变成两种情况LCS出现在和或或中，因为这时两个序列的最后一个元素不会同时出现在LCS中。

**子问题**

根据上述情况分为,和

结合最后一步和子问题，最终状态转移方程如下：

* **反证法证明最优子结构**

若 ，则是X 和Y 的长度为 k +1的公共子序列。这与 Z 是 X 和 Y 的最长公共子序列矛盾，因此必有  =  =  。由此可知，是 和的长度为 k -1的公共子序列。若 X 和 Y 有长度大于 k -1的公共子序列 W ，则将  加在其尾部产生 X 和 Y 的长度大于 k 的公共子序列。此为矛盾。所以， Z 是 X 和 Y ，的 长公共子序列。  
由于, Z 是 X 和 Y 的公共子序列，若 X 和 Y 有长度大于 k 的公共子序列 W ,则 W 也是 X 和 Y 的长度大于 k 的公共子序列。这与 Z 是 X 和 Y 的最长公共子序列矛盾。由此可知， Z 是 X -1和 Y 的最长公共子序列。

* **输出一个最长公共子序列**

输出一个最长公共子序列的思想是上述求LCS长度的逆向操作。根据上述LCS的数组中的值和两串字符是否相等去还原公共子序列。具体为：若两序列字符相同则直接答应，并转移到LCS数组的左上角，若不同再判断该LCS值是由还是得到。

具体采用递归调用的方法，每次递归调用只打印一个字符并采用先进后出的顺序，类似于一个栈。

1. **void** display(**int** m,**int** n,**char**\* s1,**char**\*s2)
2. {
3. **if** (m < 1 || n < 1) **return**;
4. **if** (s1[m] == s2[n]) cout << s1[m];
5. **else**
6. {
7. **if** (dp[m - 1][n] >= dp[m][n - 1])display(m - 1, n, s1, s2);
8. **if** (dp[m - 1][n] < dp[m][n - 1])display(m , n - 1, s1, s2);
9. }
10. }

* **输出全部的最长公共子序列**

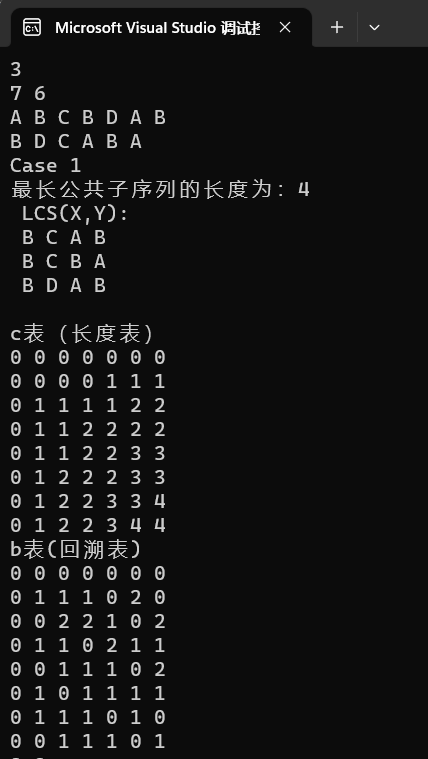
之所以上述只能输出单个最长公共子序列，是因为在面对时当二者相等的情况下，仅选了其中一个进行递归调用，因此要想输出全部首先要覆盖所有情况。

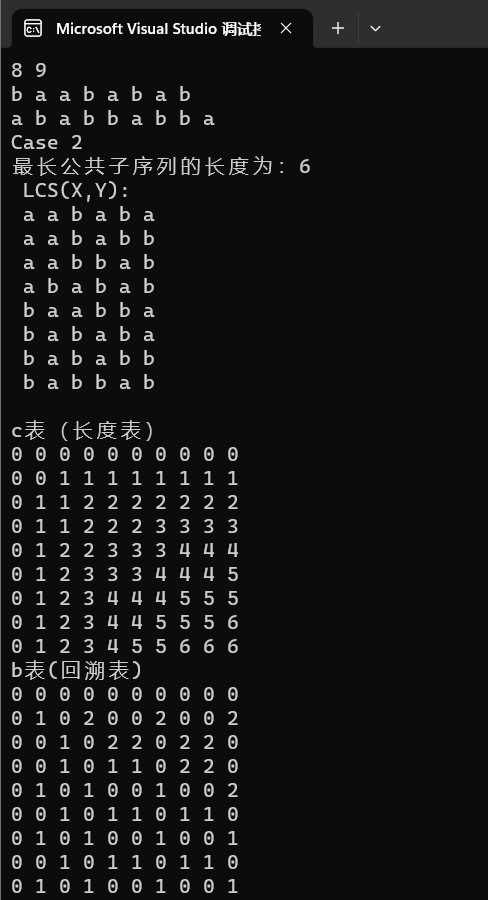
1. **if** (dp[m - 1][n] >= dp[m][n - 1])display(m - 1, n, s1, s2,ans);
2. **if** (dp[m - 1][n] <= dp[m][n - 1])display(m , n - 1, s1, s2,ans);

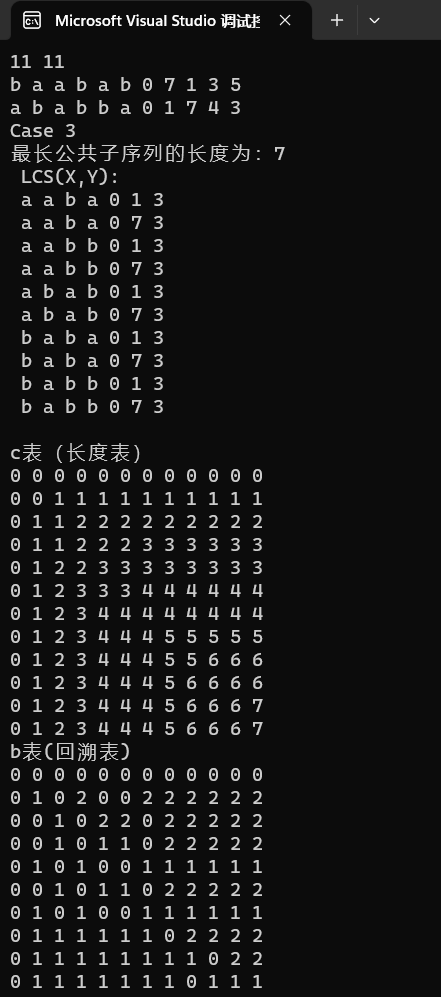
除此之外不能采取简单的递归调用逐个打印字符，采用先依次保存后全部输出的方法。使用STL中的set容器既能保存数据也能去重。同时需要在递归函数中新增一个传递字符的参数，使得每次递归能够生成一个此参数的副本，从而得到所有答案。

1. set<string> a;
2. **void** display(**int** m,**int** n,**char**\* s1,**char**\*s2,string ans)
3. {
4. **if** (m < 1 || n < 1)
5. {
6. a.emplace(ans);
7. **return**;
8. }
9. **if** (s1[m] == s2[n])display(m - 1, n - 1, s1, s2, ans + s1[m]+' ');
10. **else**
11. {
12. **if** (dp[m - 1][n] >= dp[m][n - 1])display(m - 1, n, s1, s2,ans);
13. **if** (dp[m - 1][n] <= dp[m][n - 1])display(m , n - 1, s1, s2,ans);
14. }
15. }

**运行结果截图**







**设计调试中的问题**

* 在输出所有最长公共子序列时，发现输出的序列都是相反的。这是因为根据回溯表进行构造最优子结构是从后向前的，因此在递归调用时也是先存储最后一个字符。

解决方法：在递归调用的最后，逆置字符串即可。

1. **if** (m < 1 || n < 1)
2. {
3. reverse(ans.begin(), ans.end());
4. a.emplace(ans);
5. **return**;
6. }

**实验体会**

此次实验相较于前两个实验难度有所增加，同时也是我第二次应用动态规划的思想。一开始我想沿用第二次实验矩阵连乘的方法，但是经过思考实验发现尽管都属于动态规划类型但并不可以生搬硬套。而此次实验让我了解到动态规划的要点可分为求取动态方程、构造表格和填表，方程能让我理解题目的总体思想，表格却能给我清晰的过程演示，这是在以往题目中未曾使用到的。 在最开始的时候，我只能找到一个最长的公共子序列，不知道如何简便地找出所有的最长公共子序列，在同学的讨论下，我更改了最初的直接递归输出函数，将所有输出先保存起来最后一并输出。本次实验加深了我对动态规划的理解，也巩固了一些关于字符串操作的相关知识，让我受益匪浅。

**程序源码**

1. #include<iostream>
2. #include<bits/stdc++.h>
3. **using** **namespace** std;
4. **int** dp[51][51];
5. **int** path[51][51];//0是斜上，2是左，1是上
6. set<string> a;
7. **void** display(**int** m,**int** n,**char**\* s1,**char**\*s2,string ans)
8. {
9. **if** (m < 1 || n < 1)
10. {
11. reverse(ans.begin(), ans.end());
12. a.emplace(ans);
13. **return**;
14. }
15. **if** (s1[m] == s2[n])
16. {
17. display(m - 1, n - 1, s1, s2, ans + s1[m]+' ');
18. }
19. **else**
20. {
21. **if** (dp[m - 1][n] >= dp[m][n - 1])display(m - 1, n, s1, s2,ans);
22. **if** (dp[m - 1][n] <= dp[m][n - 1])display(m , n - 1, s1, s2,ans);
23. }
24. }
26. **int** main()
27. {
28. **int** T;
29. cin >> T;
30. **for**(**int** count = 1;count <=T;count++)
31. {
32. **int** m, n;
33. **char** s1[50], s2[50];
34. memset(dp, 0, **sizeof**(dp));//初始化dp数组
35. memset(path, 0, **sizeof**(path));
36. cin >> m >> n;
37. **for** (**int** i = 1; i <= m; i++)
38. {
39. cin >> s1[i];
40. }
41. **for** (**int** i = 1; i <= n; i++)
42. {
43. cin >> s2[i];
44. }
45. **for** (**int** i = 1; i <= m; i++)
46. {
47. **for** (**int** j = 1; j <= n; j++)
48. {
49. **if** (s1[i] == s2[j])
50. {
51. dp[i][j] = dp[i - 1][j - 1] + 1;
52. path[i][j] = 0;
53. }
54. **else** **if** (dp[i - 1][j] >= dp[i][j - 1])
55. {
56. dp[i][j] = dp[i - 1][j];
57. path[i][j] = 1;
59. }
60. **else** **if** (dp[i - 1][j] < dp[i][j - 1])
61. {
62. dp[i][j] = dp[i ][j - 1];
63. path[i][j] = 2;
64. }
65. }
66. }
67. cout << "Case " << count << endl;
68. cout << "最长公共子序列的长度为：";
69. cout << dp[m][n] << endl << " LCS(X,Y): " << endl;
70. string ans;
71. display(m, n, s1, s2,ans);
72. **for** (set<string>::iterator it = a.begin(); it != a.end(); it++)
73. {
74. cout << \*it << endl;
75. }
76. a.clear();
77. cout << endl ;
78. cout << "c表（长度表）" << endl;
79. **for** (**int** i = 0; i <= m; i++)
80. {
81. **for** (**int** j = 0; j <= n; j++)
82. {
83. cout << dp[i][j] << ' ';
84. }
85. cout << endl;
86. }
87. cout << "b表(回溯表)" << endl;
88. **for** (**int** i = 0; i <= m; i++)
89. {
90. **for** (**int** j = 0; j <= n; j++)
91. {
92. cout << path[i][j] << ' ';
93. }
94. cout << endl;
95. }
96. }
97. }