重庆师范大学

实验报告

实验课程名称	算法设计与分析
实验序号	6
实验内容	最长公共子序列（动态规划）
班级	23 计科 6 班
姓名	周子依
学号	2023051603162

2024 年 4 月 25 日

实验目的与要求

学习目标：

1. 掌握动态规划的思想。
2. 理解寻找两个序列的最长公共子序列的思路。
3. 掌握如何将动态规划的思想用到求最长公共子序列中。
4. 对以下的实际例子求解连乘的顺序使得乘法次数最少。

输入：

	1	2	3	4	5	6	7
X_i	A	B	C	B	D	A	B
Y_j	B	D	C	A	B	A	

输出：

B	C	B	A

实验内容

1、理解问题

假设有两个序列，寻找其最长公共子序列，注意，与求最长公共子串不同，求最长公共子序列不要求字母连续。

- A B C B D A B
- B D C A B A

首先想到是暴力求解，找出每个序列各自的所有的子串，然后逐一比较，然而这样的时间复杂度是指数级的，仔细想想，这里的求解存在最优子结构和子重叠问题，那么能否利用动态规划的思想？

分析：

设序列 $X=\{x_1, x_2, \cdots, x_m\}$ 和 $Y=\{y_1, y_2, \cdots, y_n\}$ 的最长公共子序列为 $Z=\{z_1, z_2, \cdots, z_k\}$ ，则：

(1)若 $x_m=y_n$，则 $z_k=x_m=y_n$，且 z_{k-1} 是 x_{m-1} 和 y_{n-1} 的最长公共子序列。

(2)若 $x_m \neq y_n$ 且 $z_k \neq x_m$，则 Z 是 x_{m-1} 和 Y 的最长公共子序列。

(3)若 $x_m \neq y_n$ 且 $z_k \neq y_n$，则 Z 是 X 和 y_{n-1} 的最长公共子序列。

根据上面的思路，我们可以构造一个递推式，如下：

$$C[i,j]=\begin{cases}0 & i=0或j=0\\ C[i-1,j-1]+1 & i,j>0;x_i=y_i\\ max\{C[i-1,j],C[i,j-1]\} & i,j>0;x_i\neq y_i\end{cases}$$

2、手动推算整个求解过程

如图 1

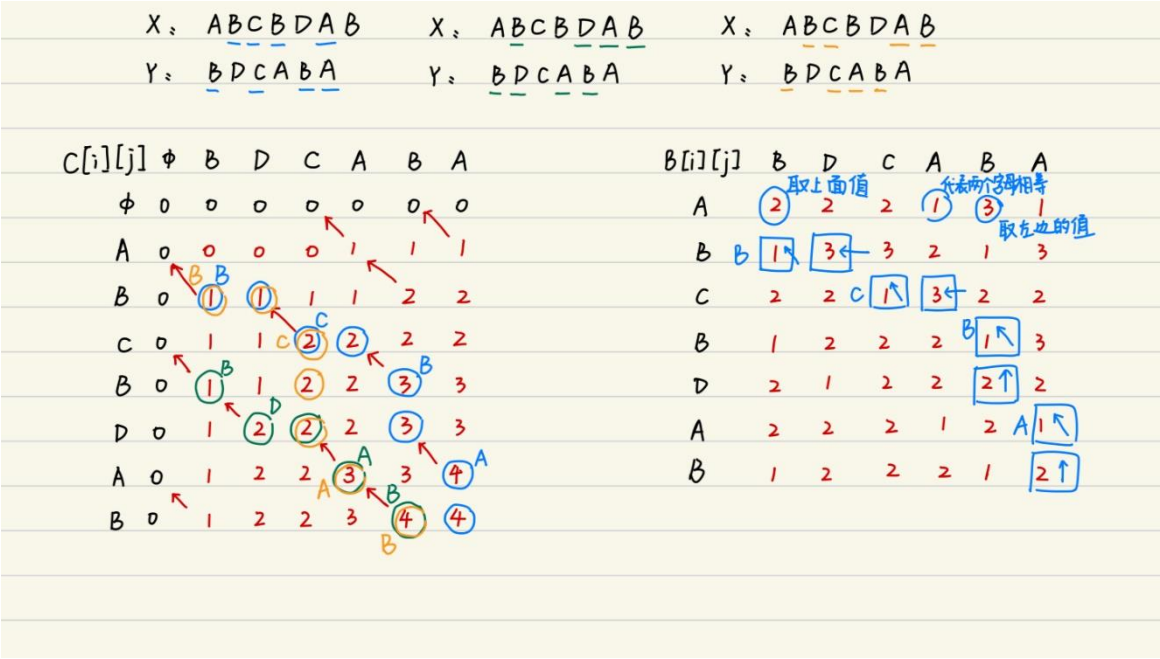

图 1 手动演算

3、核心代码分析：

首先，将 c 数组的首行和首列初始化为 0，这是因为当其中一个字符串为空时，最长公共子序列长度为 0。接着，通过两层嵌套循环遍历两个字符串。若当前字符相等，c 数组对应位置的值为左上角元素加 1，回溯方向标记为 1；若不等，则从上方和左方元素中选较大值作为当前值，回溯方向分别标记为 2 或 3。

```
// 最长公共子序列函数
void LCS(char* x, char* y, int m, int n, int** c, int** b) {
    int i, j;
    // 初始化第一行和第一列为 0
    for (i = 0; i <= m; i++) {
        c[i][0] = 0;
    }
    for (j = 0; j <= n; j++) {
        c[0][j] = 0;
    }
    // 填充 c 和 b 数组
    for (i = 1; i <= m; i++) {
        for (j = 1; j <= n; j++) {
            if (x[i - 1] == y[j - 1]) {
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i][j] = 1;
            } else if (c[i - 1][j] >= c[i][j - 1]) {
                c[i][j] = c[i - 1][j];
                b[i][j] = 2;
            } else {
                c[i][j] = c[i][j - 1];
                b[i][j] = 3;
            }
        }
    }
}
```

4、复杂度分析：

- **时间复杂度**：函数中有两层嵌套循环，外层循环次数为 m ，内层循环次数为 n ，每次循环执行的操作基本是常数时间的，所以整体时间复杂度为 O(m ×n) ，其中 m 和 n 分别是两个字符串的长度。
- **空间复杂度**：使用了两个二维数组 c 和 b ，大小均为 (m + 1) \times (n + 1) ，所以空间复杂度为 O(m ×n) 。

代码附在最后

运行结果

输入 x 和 y 序列后，打印出 c 和 b 数组求解的表格，以及输出最长公共子序列的长度及序列。

```
D:\Desktop\实验报告总\算法设
请输入X: ABCBDAB
请输入Y: BDCABA

最长公共子序列长度是: 4

C 数组求解表格:
0 0 0 0 0 0 0
0 0 0 0 1 1 1
0 1 1 1 1 2 2
0 1 1 2 2 2 2
0 1 1 2 2 3 3
0 1 2 2 2 3 3
0 1 2 2 3 3 4
0 1 2 2 3 4 4
b 数组求解表格:
  B  D  C  A  B  A

A    2  2  2  1  3  1
B    1  3  3  2  1  3
C    2  2  1  3  2  2
B    1  2  2  2  1  3
D    2  1  2  2  2  2
A    2  2  2  1  2  1
B    1  2  2  2  1  2
最长公共子序列是: BCBA

--------------------------------
Process exited after 8.646 seconds with return value 0
请按任意键继续. . .
```

总结与体会

1、 实验总结

本次实验聚焦于最长公共子序列问题，借助动态规划算法实现了对该问题的有效求解。通过本次实验，我对动态规划思想有了更为深入的掌握，即它通过将复杂问题拆解为相互关联的子问题，通过求解子问题的最优解来获取原问题的最优解。在最长公共子序列的求解过程中，明确了其核心在于依据字符是否相等的条件，合理构建递推关系来填充记录长度和回溯方向的数组。通过对不同序列的分析计算，熟练掌握了动态规划算法在该问题上的应用流程，包括状态定义（利用 c 数组记录最长公共子序列长度）、状态转移方程的推导与运用。

2、 踩坑与解决

在代码实现过程中，遇到了一些问题。在初始化 c 和 b 数组时，对边界条件的理解不够准确，导致后续计算出现偏差。比如在考虑当其中一个字符串为空串时，对最长公共子序列长度应为 0 的逻辑在代码实现上不够严谨。通过反复检查代码逻辑，重新梳理了初始化的思路，明确了首行首列元素值为 0 的依据，最终正确完成了数组初始化。在填充数组的嵌套循环中，由于对字符相等和不等时状态转移的条件判断不够细致，导致 c 和 b 数组的值计算错误。经过在纸上手动模拟计算过程，逐步理清了状态转移的逻辑，成功解决了该问题。

3、 学习收获

通过本次实验，在知识层面，我不仅深入掌握了动态规划算法原理及其在最长公共子序列问题中的应用，还对序列结构和字符匹配关系有了更深刻的认识，明白了不同字符组合对最长公共子序列结果的影响。在技能层面，极大地提高了算法设计与实现能力，包括代码编写的规范性、调试技巧以及对复杂算法逻辑的梳理能力。同时，培养了问题分析和解决能力，面对算法实现过程中的问题，学会了从算法原理出发，通过查阅资料、手动模拟计算等方法寻找解决方案。在未来的学习和实践中，我会更加注重对算法思想的理解与运用，不断提升自己解决实际问题的能力，积极尝试将动态规划等算法应用到更多实际场景中。

完整代码（c++）：

```
1.  #include <iostream>
2.  #include <cstring>
3.  using namespace std;
4.
5.  // 最长公共子序列函数
6.  void LCS(char* x, char* y, int m, int n, int** c, int** b) {
7.      int i, j;
8.      // 初始化第一行和第一列为 0
9.      for (i = 0; i <= m; i++) {
10.         c[i][0] = 0;
11.     }
12.     for (j = 0; j <= n; j++) {
```

```
13.             c[0][j] = 0;
14.         }
15.         // 填充 c 和 b 数组
16.         for (i = 1; i <= m; i++) {
17.             for (j = 1; j <= n; j++) {
18.                 if (x[i - 1] == y[j - 1]) {
19.                     c[i][j] = c[i - 1][j - 1] + 1;
20.                     b[i][j] = 1;
21.                 } else if (c[i - 1][j] >= c[i][j - 1]) {
22.                     c[i][j] = c[i - 1][j];
23.                     b[i][j] = 2;
24.                 } else {
25.                     c[i][j] = c[i][j - 1];
26.                     b[i][j] = 3;
27.                 }
28.             }
29.         }
30. }
31.
32. // 输出最长公共子序列
33. void printLCS(int** b, char* x, int i, int j) {
34.         if (i == 0 || j == 0) {
35.             return;
36.         }
37.         if (b[i][j] == 1) {
38.             printLCS(b, x, i - 1, j - 1);
39.             cout << x[i - 1];
40.         } else if (b[i][j] == 2) {
41.             printLCS(b, x, i - 1, j);
42.         } else {
43.             printLCS(b, x, i, j - 1);
44.         }
45. }
46.
47. // 打印 C 数组表格
48. void printCTable(int** c, int m, int n) {
49.         cout << "C 数组求解表格：" << endl;
50.         for (int i = 0; i <= m; i++) {
51.             for (int j = 0; j <= n; j++) {
52.                 cout << c[i][j] << " ";
53.             }
54.             cout << endl;
55.         }
```

```
}

// 打印 b 数组表格
void printBTable(int** b, char* x, char* y, int m, int n) {
    cout << "b 数组求解表格：" << endl;
    // 打印列首字母
    cout << " ";
    for (int j = 0; j < n; j++) {
        cout << " " << y[j] << " ";
    }
    cout << endl;

    for (int i = 0; i <= m; i++) {
        if (i > 0) {
            cout << x[i - 1];
        } else {
            cout << " ";
        }
        for (int j = 0; j <= n; j++) {
            if (b[i][j] == 1) {
                cout << " 1 ";
            } else if (b[i][j] == 2) {
                cout << " 2 ";
            } else if (b[i][j] == 3) {
                cout << " 3 ";
            } else {
                cout << "   ";
            }
        }
        cout << endl;
    }
}

int main() {
    char x[100];
    char y[100];
    cout << "请输入 X：";
    cin >> x;
    cout << "请输入 Y：";
    cin >> y;
    int m = strlen(x);
    int n = strlen(y);

```

```
99.     // 动态分配 c 和 b 数组
100     int** c = new int*[m + 1];
101     int** b = new int*[m + 1];
102     for (int i = 0; i <= m; i++) {
103         c[i] = new int[n + 1];
104         b[i] = new int[n + 1];
105     }
106
107     // 调用 LCS 函数
108     LCS(x, y, m, n, c, b);
109
110     // 输出最长公共子序列长度
111     cout << endl;
112     cout << "最长公共子序列长度是: " << c[m][n] << endl;
113     cout << endl;
114     // 打印 C 数组表格
115     printCTable(c, m, n);
116
117     // 打印 b 数组表格
118     printBTable(b, x, y, m, n);
119
120     // 输出最长公共子序列
121     cout << "最长公共子序列是: ";
122     printLCS(b, x, m, n);
123     cout << endl;
124
125     // 释放动态分配的内存
126     for (int i = 0; i <= m; i++) {
127         delete[] c[i];
128         delete[] b[i];
129     }
130     delete[] c;
131     delete[] b;
132
133     return 0;
134 }
```