南京郵電大學
Nanjing University of Posts and Telecommunications

《算法设计与分析》课内实验报告

实验题目：动态规划法

实验日期：2023 年 12 月 26 日

指导教师：张博

学生姓名：单家俊

学生学号：B21080526

实验成绩：

一、实验目的

掌握动态规划法的思想，及动态规划法在实际中的应用；分析最长公共子序列的问题特征，选择算法策略并设计具体算法，编程实现两输入序列的比较，并输出它们的最长公共子序列。

二、实验内容

用动态规划法实现求两序列的最长公共子序列。

三、实验过程描述

算法思想：

设有序列 Xm=（x1，x2，……，xm）和 Yn=（y1，y2，……yn）：

（1） 若 xm=yn，则先求 Xm-1 和 Yn-1 的最长公共子序列，并在其尾部加上 xm 便可以得到 Xm 和 Yn 的最长公共子序列；

（2） 若 xm 不等于 yn，则必须分别求解两个子问题 Xm-1 和 Yn，以及 Xm 和 Yn-1 的最长公共子序列，这两个公共子序列中的较长者就是 Xm 和 Yn 的最长公共子序列。

需要使用一个二维数组来保存最长公共子序列的长度，设 c[i][j]保存 Xi=（x1，x2，……，xi）和 Yj=（y1，y2，……，yj）的最长公共子序列的长度。当 i=0 或 j=0 时。Xi 和 Yj 的最长公共子序列为空序列，故此时 c[i][j]=0；若 xi=yj（i，j>0），则 c[i][j]=c[i-1][j-1]+1；若 xi 不等于 yj（i，j>0），则 c[i][j]=max{c[i-1][j],c[i][j-1]}。

计算时间复杂度：

本问题中，不同的子问题数目总计为Θ（m×n），采用动态规划法进行自底向上求解，可在多项式时间内完成计算。由于每个数组元素的计算时间为 O（1），则最长公共子序列算法的时间复杂度为 O（m×n）。

代码：

```
#include <iostream>
#include <chrono>

char* x;
char* y;
int** c;
int** s;

int LCSLenth(int m, int n)
{
```

```
    for (int i = 1; i <= m; i++)
        c[i][0] = 0;
    for (int j = 1; j <= n; j++)
        c[0][j] = 0;
    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (x[i] == y[j])
            {
                c[i][j] = c[i - 1][j - 1] + 1;
                s[i][j] = 1;
            }
            else if (c[i - 1][j] >= c[i][j - 1])
            {
                c[i][j] = c[i - 1][j];
                s[i][j] = 2;
            }
            else
            {
                c[i][j] = c[i][j - 1];
                s[i][j] = 3;
            }
        }
    }
    return c[m][n];
}

void CLCS(int i, int j)
{
    if (i == 0 || j == 0)
        return;
    if (s[i][j] == 1) {
        CLCS(i - 1, j - 1);
        std::cout << x[i];
    }
    else if (s[i][j] == 2)
        CLCS(i - 1, j);
    else
        CLCS(i, j - 1);
}

int main()
{
```

```
    char a[] = {'0','x','z','y','z','z','y','x'};
    char b[] = {'0','z','x','y','y','z','x','z'};
    int m = sizeof(a)/sizeof(char); //8
    int n = sizeof(b)/sizeof(char); //4
    x = a;
    y = b;

    c = new int* [m];   s = new int* [m];
    for (int i = 0; i < m; i++)
    {
        c[i] = new int[n];
        s[i] = new int[n];
    }
    auto start_time = std::chrono::high_resolution_clock::now();
    int len = LCSLenth(m-1,n-1);
    std::cout << "a='xzyzzyx'  b='zxyyzxz'" <<std::endl;
    std::cout << "最长公共子序列长度为: " << len << std::endl;
    std::cout << "最长公共子序列为: ";
    CLCS(m-1,n-1);
    std::cout << std::endl;
    auto end_time = std::chrono::high_resolution_clock::now();
    auto duration =
std::chrono::duration_cast<std::chrono::microseconds>(end_time -
start_time);
    std::cout << "Execution Time: " << duration.count() << " microseconds"
<< std::endl;
    std::cout << std::endl;
    // 释放 c 和 s 的内存
    for (int i = 0; i < m; i++)
    {
        delete[]c[i];
        delete[]s[i];
    }
    delete[]c;   delete[]s;

    char a1[] = {'0','a','b','c','b','d','a','b'};
    char b1[] = {'0','b','d','c','a','b','a'};
    int m1 = sizeof(a1)/sizeof(char); //8
    int n1 = sizeof(b1)/sizeof(char); //4
    x = a1;
    y = b1;

    c = new int* [m1];   s = new int* [m1];
```

```
    for (int i = 0; i < m1; i++)
    {
        c[i] = new int[n1];
        s[i] = new int[n1];
    }
    auto start_time1 = std::chrono::high_resolution_clock::now();
    int len1 = LCSLenth(m1-1,n1-1);
    std::cout << "a='abcbdab'  b='bdcaba'" <<std::endl;
    std::cout << "最长公共子序列长度为：" << len1 << std::endl;
    std::cout << "最长公共子序列为：";
    CLCS(m1-1,n1-1);
    std::cout << std::endl;
    auto end_time1 = std::chrono::high_resolution_clock::now();
    auto duration1 =
std::chrono::duration_cast<std::chrono::microseconds>(end_time1 -
start_time1);
    std::cout << "Execution Time: " << duration1.count() << " microseconds"
<< std::endl;
    std::cout << std::endl;
    // 释放 c 和 s 的内存
    for (int i = 0; i < m1; i++)
    {
        delete[]c[i];
        delete[]s[i];
    }
    delete[]c;   delete[]s;

    char a2[] =
{'0','a','b','c','b','d','a','b','c','a','f','g','h','k'};
    char b2[] = {'0','b','d','c','a','b','a','a','f','c','e'};
    int m2 = sizeof(a2)/sizeof(char); //8
    int n2 = sizeof(b2)/sizeof(char); //4
    x = a2;
    y = b2;

    c = new int* [m2];   s = new int* [m2];
    for (int i = 0; i < m2; i++)
    {
        c[i] = new int[n2];
        s[i] = new int[n2];
    }
    auto start_time2 = std::chrono::high_resolution_clock::now();
    int len2 = LCSLenth(m2-1,n2-1);
    std::cout << "a='abcbdabcafghk'  b='bdcabaafce'" <<std::endl;
```

```
    std::cout << "最长公共子序列长度为：" << len2 << std::endl;
    std::cout << "最长公共子序列为：";
    CLCS(m2-1,n2-1);
    std::cout << std::endl;
    auto end_time2 = std::chrono::high_resolution_clock::now();
    auto duration2 =
std::chrono::duration_cast<std::chrono::microseconds>(end_time2 -
start_time2);
    std::cout << "Execution Time: " << duration2.count() << " microseconds"
<< std::endl;
    // 释放 c 和 s 的内存
    for (int i = 0; i < m2; i++)
    {
        delete[]c[i];
        delete[]s[i];
    }
    delete[]c;   delete[]s;


    return 0;
}
```

四、实验结果

```
a='xzyzzyx'  b='zxyyzxz'
最长公共子序列长度为：4
最长公共子序列为：xyzz
Execution Time: 1997 microseconds

a='abcbdab'  b='bdcaba'
最长公共子序列长度为：4
最长公共子序列为：bcba
Execution Time: 996 microseconds

a='abcbdabcafghk'  b='bdcabaafce'
最长公共子序列长度为：6
最长公共子序列为：bcbaaf
Execution Time: 2528 microseconds
```

由实验结果可知，随着两个序列的长度不断增长，求最长公共子序列的时间也在不断增长。

五、实验小结

在这次算法实验中进行了最长公共子序列问题的求解。以下是我对实验的一些心得体会：

动态规划法的应用：通过实现最长公共子序列问题，我深入理解了动态规划法在实际问题中的应用。动态规划通过将问题划分为子问题，并保存子问题的解来避免重复计算，从而高效地解决大规模问题。

最长公共子序列问题的特征和算法选择：最长公共子序列问题是在两个序列中找到一个最长的公共子序列。在解决该问题时，我分析了问题的特征，并选择了动态规划作为解决策略。通过设计具体的算法，我成功地实现了两个输入序列的比较并输出它们的最长公共子序列。

遇到的问题及解决方法：在实验过程中，可能会遇到一些问题。例如，算法实现错误、计算结果不正确或指针访问内存冲突。为了解决这些问题，我采取了仔细检查代码、调试和输出调试信息的方法。同时，我也可以通过优化算法实现或者使用合适的数据结构来提高算法的效率。

在本次实验过程中，采用了动态规划法进行了算法设计与应用，加深了对动态规划法的算法原理及实现过程的理解，学习了用动态规划法解决实际应用中的最长公共子序列问题，体会到了动态规划法的优点，受益匪浅。