实验二 动态规划法

实验目的

加深对动态规划法的算法原理及实现过程的理解，学习用动态规划法解决实际应用中的最长公共子序列问题和矩阵连乘问题，体会动态规划法和备忘录方法的异同。

实验内容

一、用动态规划法和备忘录方法实现求两序列的**最长公共子序列**问题。要求掌握动态规划法思想在实际中的应用，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、用动态规划法和备忘录方法求解**矩阵相乘问题**，求得最优的计算次序以使得矩阵连乘总的数乘次数最少，并输出加括号的最优乘法算式。

实验步骤

一、最长公共子序列

1、最长公共子序列（Longest Common Subsequence,LCS）问题是：给定两个字符序列 $X=\{x_1,x_2,\ldots\ldots,x_m\}$和 $Y=\{y_1,y_2,\ldots\ldots,y_n\}$，要求找出 A 和 B 的一个最长公共子序列。

例如：X={a,b,c,b,d,a,b}，Y={b,d,c,a,b,a}。它们的最长公共子序列 LSC={b,c,b,a}。通过“穷举法”列出 X 的所有子序列，检查其是否为 Y 的子序列并记录最长公共子序列的长度这种方法，求解时间为指数级的，因此不可取。

2、分析 LCS 问题特征可知，如果 $Z=\{z_1,z_2,\ldots\ldots,z_k\}$为它们的最长公共子序列，则它们一定具有以下性质：

（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。

这样就将求 X 和 Y 的最长公共子序列问题，分解为求解较小规模的子问题：

- 若 $x_m=y_n$，则进一步分解为求解两个（前缀）子字符序列 X_{m-1} 和 Y_{n-1} 的最长公共子序列问题；
- 如果 $x_m\neq y_n$，则原问题转化为求解两个子问题，即找出 X_{m-1} 和 Y 的最长公共子序列与找出 X 和 Y_{n-1} 的最长公共子序列，取两者中较长者作为 X 和 Y 的最长公共子序列。

由此可见，两个序列的最长公共子序列包含了这两个序列的前缀的最长公共子序列，具有**最优子结构**性质。

3、令 c[i][j]保存字符序列 $X_i=\{x_1,x_2,\ldots\ldots,x_i\}$和 $Y_j=\{y_1,y_2,\ldots\ldots,y_j\}$的最长公共子序列的长度。由上述分析可得如下递推式：

$$c[i][j]=\begin{cases}0 & i=0或j=0\\ c[i-1][j-1]+1 & i,j>0且x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0且x_i\neq y_j\end{cases}$$

由此可见，最长公共子序列的求解具有**重叠子问题**性质，如果采用递归算法实现，会得到一个指数时间算法。因此需要采用**动态规划法**自底向上求解，并保存子问题的解，这样可以避免重复计算子问题，在多项式时间内完成计算。

4、为了能由最优解值进一步得到最优解（即最长公共子序列），还需要一个二维数组 s[][]，数组中的元素 s[i][j]记录 c[i][j]的值是由三个子问题 c[i-1][j-1]+1,c[i][j-1]和 c[i-1][j]中的哪一个计算得到，从而可以得到最优解的当前解分量（即最长公共子序列中的当前字符），最终构造出最长公共子序列自身。

5、编程定义 LCS 类，计算最长公共子序列长度，并给出最长公共子序列：

（**注意：C 语言中数组下标由 0 开始，而实际数据在一维数组 a、b 和二维数组 c、s 中的存放却是从下标为 1 处开始。**）

类中数据成员主要有二维数组 c 和 s 用于动态规划法求解过程中保存子问题的求解结果，一维数组 a 和 b 用于存放两个字符序列，m 和 n 为两个字符序列中实际字符的个数。这些数据成员均应在 LCS 类的构造函数中进行初始化：

```
#include <iostream.h>
#include <string.h>
#define maxlength 11
class LCS
{
public:
    LCS(int nx,int ny,char *x,char *y)  //对数据成员m、n、a、b、c、s初始化
    {    m=nx;        n=ny;    //对m和n赋值
         a=new char[m+2];    //考虑下标为0的元素和字符串结束标记
         b=new char[n+2];
         memset(a,0,m+2);
         memset(b,0,n+2);
         。。。。。。          //将x和y中的字符写入一维数组a和b中
         c=new int[maxlength][maxlength];     //maxlength为某个常量值
         s=new int[maxlength][maxlength];
         。。。。。。          //对二维数组c和s中元素进行初始化
    }
    int LCSLength();
    void CLCS()
    {
         。。。。。。      //调用私有成员函数CLCS(int,int)
    }
private:
    void CLCS(int i,int j) const;
    int (*c)[maxlength],(*s)[maxlength];
```

```
    int m,n;
    char *a,*b;
};
```

6、类中成员函数主要有 LCSLength()和 CLCS()两个公有成员函数，CLCS()通过调用私有递归成员函数 CLCS(int i,int j)实现：

```
int LCS::LCSLength()
{
    for (int i=0;i<=m;i++) c[i][0]=0;      //初始化行标或列标为0的元素
    for (int j=1;j<=n;j++) c[0][j]=0;
    for (i=1;i<=m;i++)
        for (j=1;j<=n;j++)
            if (a[i]==b[j])       //由c[i-1][j-1]得到c[i][j]
            {
                c[i][j]=c[i-1][j-1]+1;    s[i][j]=1;
            }
            else
                if (c[i-1][j]>=c[i][j-1])  //由c[i-1][j]得到c[i][j]
                {
                    c[i][j]=c[i-1][j];    s[i][j]=2;
                }
                else      //由c[i][j-1]得到c[i][j]
                {
                    c[i][j]=c[i][j-1];    s[i][j]=3;
                }
    return c[m][n];   //返回最优解值
}

void LCS::CLCS(int i,int j) const
{
    if (i==0||j==0||s[i][j]==0) return;     //终止条件
    if (s[i][j]==1)
    {
        CLCS(i-1,j-1);
        cout<<a[i];         //输出最优解的当前分量
    }
    else
        if (s[i][j]==2) CLCS(i-1,j);
        else CLCS(i,j-1);
}
```

7、主函数中负责输入两个待比较的字符串 x 和 y（长度不超过 maxlength-1），求得他们的实际字符长度 nx 和 ny，并调用 LCS 类的成员函数 LCSLength()和 CLCS()求最优解值和最优解：

```
void main()
{    int nx,ny;
     char *x, *y;
。。。。。。
     LCS lcs(nx,ny,x,y);
     。。。。。。
     delete []x;
     delete []y;
}
```

8、输入序列 X=(x1,x2,……,xm)=(a,b,c,b,d,a,b)和 Y=(y1,y2,……,yn)=(b,d,c,a,b,a)作为测试数据，测试程序是否能够正确运行？输出结果是什么？

9、分析该动态规划算法的两个主要成员函数 int LCSLength()和 void CLCS()的时间复杂性。

二、矩阵连乘

1、**求解目标**

若 n 个矩阵{$\mathbf{A_1,A_2,\ldots\ldots A_n}$}中两个相邻矩阵 A_i 和 A_{i+1} 均是可乘的，$\mathbf{A_i}$ 的维数为 $\mathbf{p_i\times p_{i+1}}$，$\mathbf{A_{i+1}}$ 的维数为 $\mathbf{p_{i+1}\times p_{i+2}}$。求 n 个矩阵 $\mathbf{A_1A_2\ldots\ldots A_n}$ 连乘时的最优计算次序，以及对应的最少数乘次数。

两矩阵相乘 $\mathbf{A_iA_{i+1}}$ 做 $\mathbf{p_i\times p_{i+1}\times p_{i+2}}$ 次数乘，可得 $\mathbf{p_i\times p_{i+2}}$ 的结果矩阵。

而矩阵连乘 $\mathbf{A_iA_{i+1}\ldots\ldots A_j}$（简记为 **A[i：j]**）求得 $\mathbf{p_i\times p_{j+1}}$ 的结果矩阵时，采用不同的计算次序，对应的总数乘次数也不同。

2、**例如**：4 个矩阵连乘 $A_1A_2A_3A_4$，其中 A_1 的维数：50×10，A_2 的维数：10×40，A_3 的维数：40×30，A_4 的维数：30×5。有 5 种不同的计算次序：

次序 1：（（（A_1A_2）A_3）A_4）　　需要 50×10×40+50×40×30+50×30×5=87500 次

次序 2：（（A_1A_2）（A_3A_4））　　需要 50×10×40+40×30×5+50×40×5=36000 次

次序 3：（（A_1（A_2A_3））A_4）　　需要 10×40×30+50×10×30+50×30×5=34500 次

次序 4：（A_1（（A_2A_3）A_4））　　需要 10×40×30+10×30×5+50×10×5=16000 次

次序 5：（A_1（A_2（A_3A_4）））　　需要 40×30×5+10×40×5+50×10×5=10500 次

3、将**二维数组 m[i][j]**定义为：计算 A[i：j]所需的最少数乘次数；

二维数组 s[i][j]定义为：计算 A[i：j]的最优计算次序中的断开位置（例如：若计算 A[i：j]的最优次序在 A_k 和 A_{k+1} 之间断开，i≤k<j，则 s[i][j]=k）。

4、**当 i=j 时**，A[i：j]=A_i 是单一矩阵，无须计算，因此 **m[i][j]=0**；

当 i<j 时，m[i][j]=min{m[i][k]+m[k+1][j]+$p_ip_{k+1}p_{j+1}$}　(i≤k<j)

5、**算法思路**

因为计算 m[i][j]时，只用到已计算出的 m[i][k]和 m[k+1][j]。所以首先计算出 m[i][i]=0，i=1,2,……n；然后再根据递归式，按矩阵链长递增的方式依次计算 m[i][i+1]，i=1，2，…n-1（矩阵链长度为 2）；m[i][i+2]，i=1，2，…n-2（矩阵链长度为 3）；……则 m[1][n]就是问题的最优解值（最少数乘次数）。

要构造问题的最优解，根据 s 数组可推得矩阵乘法的次序。从 s[1][n]可知计算 A[1：n]的最优加括号方式为(A[1：s[1][n]])(A[s[1][n]+1：n])。其中 A[1：s[1][n]]的最优加括号方式又为（A[1：s[1][s[1][n]]]）(A[s[1][s[1][n]]+1：s[1][n]])。……照此递推下去，最终可以确定 A[1：n]的最优完全加括号方式，构造出问题的一个最优解。

6、动态规划法实现的算法提示

（请分别对实例 1：A1 维数：50×10，A2 维数：10×40，A3 维数：40×30，A4 维数：30×5 和实例 2：A1 维数：30×35，A2 维数：35×15，A3 维数：15×5，A4 维数：5×10；A5 维数：10×20，A6 维数：20×25 分别求解。）

```
void MatrixChain(int n,int *p,int **m,int **s)
{
    for (int i=1;i<=n;i++) m[i][i]=0;
    for (int r=2;r<=n;r++)
        for(int i=1;i<=n-r+1;i++)
        {
            int j=i+r-1;

            m[i][j]=m[i+1][j]+p[i]*p[i+1]*p[j+1];      //省略了m[i][i]=0项

            s[i][j]=i;
            for(int k=i+1;k<j;k++)
            {
                int t=m[i][k]+m[k+1][j]+p[i]*p[k+1]*p[j+1];
                if (t<m[i][j])
                {
                    m[i][j]=t;

                    cout<<"更新m["<<i<<"]["<<j<<"]的值为："<<t<<endl;

                    s[i][j]=k;

                    cout<<"更新s["<<i<<"]["<<j<<"]的值为："<<k<<endl;

                }
            }

            cout<<"最终求出：m["<<i<<"]["<<j<<"]的值为："<<m[i][j]<<endl;

        }
}
void Traceback(int i,int j,int **s)
{
    if (i==j) {cout<<'A'<<i<<' ';return;}
    if (i<s[i][j]) cout<<'(';
    Traceback(i,s[i][j],s);
    if (i<s[i][j]) cout<<')';
    if (s[i][j]+1<j) cout<<'(';
    Traceback(s[i][j]+1,j,s);
    if (s[i][j]+1<j) cout<<')';
}
```

思考

1、**备忘录方法**是动态规划法的一个变种，它采用分治法思想，自顶向下直接递归求最优解。但与分治法不同的是，备忘录方法为每个已经计算的子问题建立备忘录，即保存子问题的计算结果以备需要时引用，从而避免子问题的重复求解。

试改写当前程序的 int LCS::LCSLength()函数，用备忘录方法来求解最长公共子序列。（**提示：**备忘录方法采用的是递归求解方式，因此需要用一个公有成员函数 int LCSLength()；来调用私有递归成员函数 int LCSLength(int i,int j);共同实现。）

```
int LCS::LCSLength(int i,int j)
{
    if (i==0||j==0) return 0;
    if (c[i][j]!=0) return c[i][j];
    else
        if (a[i]==b[j])
        {   c[i][j]=LCSLength(i-1,j-1)+1;  s[i][j]=1;
        }
        else
            if (LCSLength(i-1,j)>=LCSLength(i,j-1))
            {   c[i][j]=LCSLength(i-1,j);  s[i][j]=2;
            }
            else
            {   c[i][j]=LCSLength(i,j-1);  s[i][j]=3;
            }
return c[i][j];
}
```

2、若省去原程序中的二维数组 s，是否还能求得最长公共子序列问题的最优解？

请编写一个类似的 CLCS 算法实现：不借助二维数组 s 在 O(m+n)的时间内构造最长公共子序列的功能。

（**提示：此时可在当前 c[i][j]处比较 a[i]和 b[j]。**

如果相等，则调用 CLCS(i-1,j-1)，输出 a[i](或 b[j])。

如果不等，则比较 c[i-1][j]和 c[i][j-1]。若 c[i-1][j]≥c[i][j-1],则递归调用 CLCS(i-1,j)；否则，递归调用 CLCS(i,j-1)。）

```
void LCS::CLCS(int i,int j) const
{   if (i==0||j==0) return;
    if (a[i]==b[j])
    {
        CLCS(i-1,j-1);
        cout<<a[i];
    }
    else
        if (c[i-1][j]>=c[i][j-1]) CLCS(i-1,j);
        else CLCS(i,j-1);
}
```

3、如果只需计算最长公共子序列的长度，而无须构造最优解，则如何改进原有程序可以使得算法的空间需求大大减少？请改写原程序，使算法的空间复杂度减少为 O(min{m,n})。

（提示：计算 c[i][j]仅用到第 i 行和第 i-1 行元素。因此，只需两行元素空间就可计算最长公共子序列的长度。并且选用序列长度较短的一个作为 y 序列，可以缩短每行元素的个数，从而进一步减少空间复杂度。）

```
class LCS
{
public:
    LCS(int nx,int ny,char *x,char *y)
    {   。。。。。。
        if (m>n)      {   。。。。。。}
        else     {   。。。。。。}
        c1=new int[s];
        c2=new int[s];
        。。。。。。      //为c1、c2赋初值
    }
    int LCSLength();
private:
。。。。。。
    int *c1,*c2;
    int l,s;        //l存放较长序列的长度，s存放较短序列的长度。
    。。。。。。
};
int LCS::LCSLength()
{
    int *temp;
    for (int i=0;i<=s;i++) c1[i]=0;
    。。。。。。
    return c2[s];      //返回最优解值
}
void main()
{   。。。。。。
    LCS lcs(nx,ny,x,y);
    cout<<"最长公共子序列长度值（最优解值）："<<lcs.LCSLength()<<endl;
    delete []x;
    delete []y;
}
```

4、思考：若要求输出所有可能的最长公共子序列，该如何修改LCS算法？

（提示：原CLCS函数if (c[i-1][j]>=c[i][j-1]) ...语句中没有区分c[i-1][j]>c[i][j-1]和c[i-1][j]=c[i][j-1]这两种不同的情况。因此要找出所有的LCS，就必须在a[i]!=b[j]且c[i-1][j]==c[i][j-1]的时候，分别沿着c[i-1][j]向上和c[i][j-1]向左两个搜索方向分别构造最优解，才能据此找出所有的LCS。）

5、矩阵连乘问题的备忘录方法如何实现？

（备忘录方法求解矩阵相乘问题的算法提示：）

```
int LookupChain(int i,int j,int *p,int **m,int **s)
{                                          //返回Ai连乘到Aj的最小计算量，同时求断开位置
    if (m[i][j]>0)  return m[i][j];
    if (i==j) return 0;
    m[i][j] = LookupChain(i+1,j,p,m,s) + p[i]*p[i+1]*p[j+1];
    s[i][j]=i;
    for (int k=i+1;k<j;k++)
    {
        int v=LookupChain(i,k,p,m,s)+LookupChain(k+1,j,p,m,s)+p[i]*p[k+1]*p[j+1];
        if (v< m[i][j])
        {
            cout<<"更新m["<<i<<"]["<<j<<"]的值为："<<v<<endl;
            m[i][j]=v;
            cout<<"更新s["<<i<<"]["<<j<<"]的值为："<<k<<endl;
            s[i][j]=k;
        }
    }
    cout<<"最终求出：m["<<i<<"]["<<j<<"]的值为："<<m[i][j]<<endl;
    return m[i][j];
```