3.2动态规划算法的基本要素

一、最优子结构

- 矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为**最优子结构性质**。
  - 在分析问题的最优子结构性质时，所用的方法是反证法。

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

3.2 动态规划算法的基本要素

一、最优子结构

- 利用问题的最优子结构性质，以自底向上的方式递归地从子问题的最优解逐步构造出整个问题的最优解
  - 最优子结构是问题能用**动态规划算法求解的前提**。

3.2 动态规划算法的基本要素

二、重叠子问题

- 递归算法求解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。这种性质称为子**问题的重叠性质**。
  - $A_1(A_2A_3A_4)$
  - $(A_1A_2)(A_3A_4)$

3.2 动态规划算法的基本要素

二、重叠子问题

- 动态规划算法，对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。

- 通常不同的子问题个数随问题的大小呈多项式增长。因此用动态规划算法只需要多项式时间，从而获得较高的解题效率。

用递归算法求解矩阵连乘

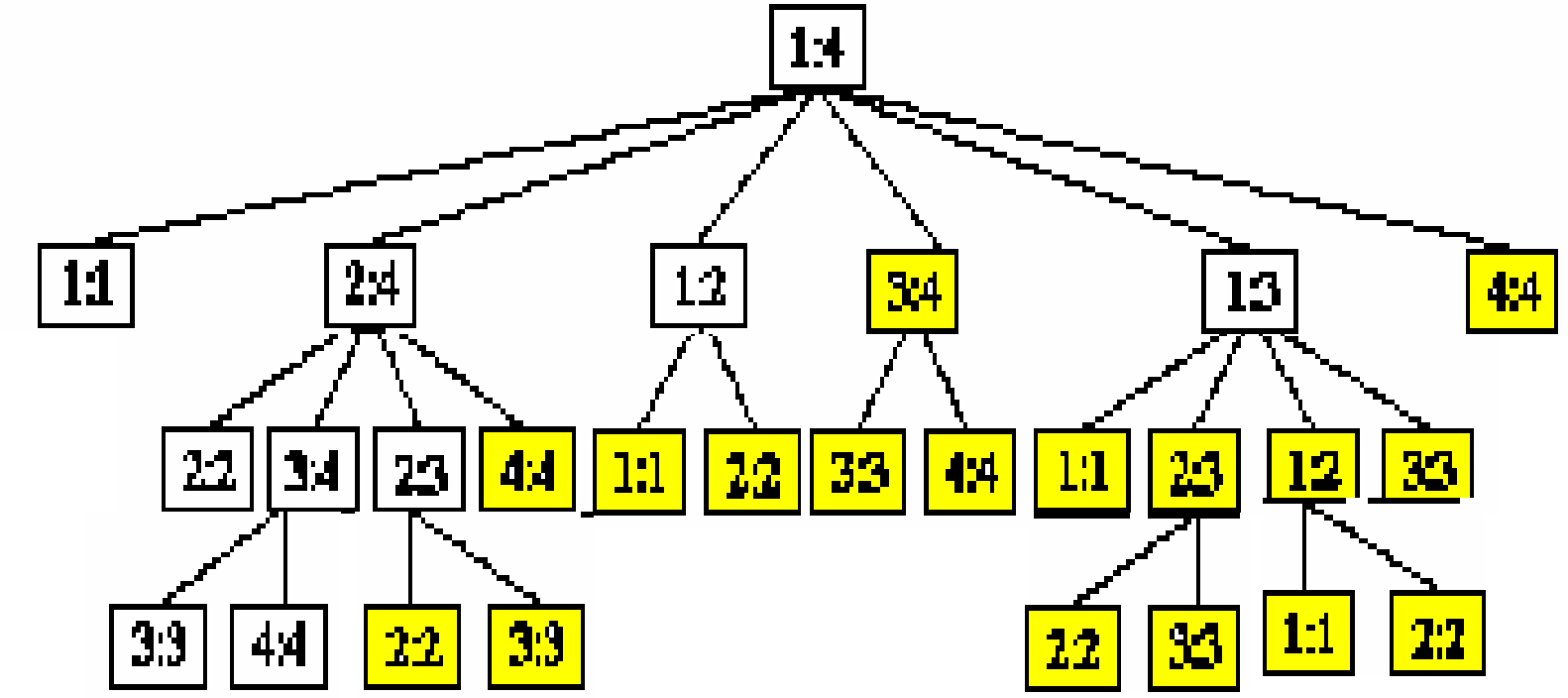
递归定义：

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

```
public static int recurMatrixChain(int i, int j){
    if( i==j) return 0;
    int u = recurMatrixChain(i+1,j) + p[i-1]*p[i]*p[j];
    s[i][j]=i;
    for( int k=i+1;k<j;k++) {
        int t = recurMatrixChain(i,k) + recurMatrixChain(k+1,j) +p[i-1]*p[k]*p[j];
        if( t<u) {
            u=t;
            s[i][j]=k;
        }
    }
    return u;
}
```

- 递归算法的时间复杂度为$\Omega(2^n)$

重叠子问题


1:4
1:1
2:4
1:2
3:4
1:3
4:4
2:2
3:4
2:3
4:4
1:1
2:2
3:3
4:4
1:1
2:3
1:2
3:3
3:3
4:4
2:2
3:3
2:2
3:3
1:1
2:2

三、备忘录方法

- **备忘录方法是动态规划的一种变形**
  - 既具有动态规划方法的效率，又采用了一种自顶向下的策略
- **备忘录方法与递归方法：**
  - 相同点：
    - 备忘录方法的控制结构与直接递归方法的控制结构相同
  - 不同点：
    - 区别在于备忘录方法为每个解过的子问题建立了备忘录以备需要时查看，避免了相同子问题的重复求解。

三、备忘录算法

```
public static int
memorizedMatrixChain(int n){
    for(int i=1;i<=n;i++)
        for(int j=l;j<=n;j++)
            m[i][j] = 0;
    return LookupChain(1,n);
}
```

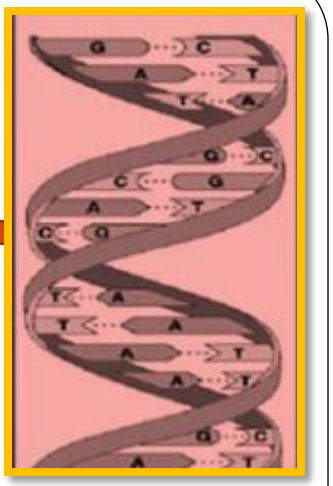
```
int LookupChain(int i，int j)
{
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = LookupChain(i，i) + LookupChain(i+1，j) + p[i-1]*p[i]*p[j];
    s[i][j] = i;
    for (int k = i+1; k < j; k++) {
        int t = LookupChain(i，k) + LookupChain(k+1，j) + p[i-1]*p[k]*p[j];
        if (t < u) { u = t; s[i][j] = k;}
        }
    m[i][j] = u;
    return u;
}
```

与直接递归算法的唯一区别（备忘录的主要操作）

备忘录方法与动态规划

- **动态规划算法没有递归代价**
  - 在应用中，如果所有的子问题都至少要被计算一次，则一个自底向上的动态规划算法通常要比一个自顶向下的备忘录方法好出一个常数因子，因为前者没有递归代价，而且维护表格的开销要小些。
- **备忘录方法只解必须求解的子问题**
  - 如果子问题空间中的某些子问题根本没有必要求解，则备忘录方法更优，因为它只需求解必须求解的子问题。

3.3 最长公共子序列问题



- 程序代码相似度比较问题
- 序列比对问题（Sequence Alignment）

  - 在生物学的应用中比较两个不同有机体的DNA序列，生物有机体的DAN可以表示为四种碱基 {A,C,T,G} 的字符序列，可以通过序列间相似性比较来推断不同物种之间的进化关系。
  - 这种相似度概念形式化为最长公共子序列问题。公共子序列越长，可认为相似度越高。

S_1= ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

GTCGTCGGAAGCCGGCCGAA

序列的定义

子序列

- 给定序列$X=\{x_1,x_2,\ldots,x_m\}$，另一序列$Z=\{z_1,z_2,\ldots,z_k\}$，若存在一个严格递增下标序列$\{i_1,i_2,\ldots,i_k\}$使得对于所有$j=1,2,\ldots,k$有：$z_j=x_{i_j}$，则Z是X的子序列。
- 例如：
  - 序列X={A，B，C，B，D，A，B}（1，2，3，4，5，6，7）
  - 有序列Z_1={A，D，C}（1，5，3）×
  - 有序列Z_2={B，C，D，B}（2，3，5，7）√

序列的定义

- 公共子序列
  - 给定2个序列X和Y，当另一序列Z既是X的子序列又是Y的子序列时，称Z是序列X和Y的公共子序列。

3.3 最长公共子序列问题
(Longest Common Subsequence)

最长公共子序列

- 给定2个序列$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$，找出X和Y的最长公共子序列。

例：$\sum=\{x,y,z\}$,A="zxyxyz",B="xyyzx"

- 考查 "xyy“
- 是A（$a_2a_3a_5$）和B（$b_1b_2b_3$）长度为3的公共子序列
- 不是A和B的最长公共子序列。

3.3 最长公共子序列问题
(Longest Common Subsequence)

- 例：∑={x,y,z},A="zxyxyz",B="xyyzx"
- 考查"xyyz“
- 是A（$a_2a_3a_5a_6$）和B（$b_1b_2b_3b_4$）长度为4的公共子序列
- 并且是A和B的最长公共子序列。

3.3 最长公共子序列问题
(Longest Common Subsequence)

- 最长公共子序列
  - 给定2个序列X={x_1, x_2, …, x_m}和Y={y_1, y_2, …, y_n}，找出X和Y的最长公共子序列。
    - 找出“一个”而不是“唯一的”
    - 公共子序列在原序列当中不一定是连续的。

x:	A	B	C	B	D	A	B
y:	B	D	C	A	B	A	

穷举搜索法 (Brute-force LCS Algorithm)

X="abcbda",Y="abcdd"

- **用穷举搜索法求解**
  - 对X的所有子序列，检查它是否是Y的子序列，并记录最长的公共序列
  - X有2^m个子序列
  - 对每条子序列，检查是否是Y的子序列，需要O(n)时间
  - 从而需要$O(n*2^m)$，即**指数时间**来搜索

动态规划方法求最长公共子序列

- 设有序列$X=\{x_1, x_2, \cdots, x_{m-1}, x_m\}$和$Y=\{y_1, y_2, \cdots, y_{n-1}, y_n\}$，如何找最优子结构？

- 尝试从最后一个元素处划分，四种情况：
  - x_m是LCS的一部分，而y_n不是；
  - y_n是LCS的一部分，而x_m不是；
  - x_m和y_n都不是LCS的一部分；

  $x_m \neq y_n$
  - x_m和y_n都是LCS的一部分；

  $x_m = y_n$

动态规划方法求最长公共子序列

- 子问题题的描述
  - X的终止位置是i，0<=i<m
  - Y的终止位置是j，0<=j<n
  - $X_i=\{x_1,x_2,\ldots,x_i\}$和$Y_j=\{y_1,y_2,\ldots,y_j\}$

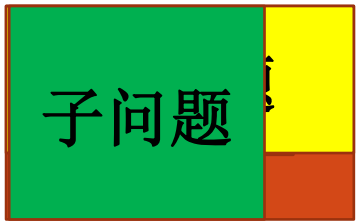
3.3 动态规划求解LCS

第一步：分析最长公共子序列的最优子结构

LCS最优子结构定理：

设序列$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$的最长公共子序列为$Z=\{z_1,z_2,\ldots,z_k\}$，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LCS。
- (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。



- 由此可见，2个序列的最长公共子序列包含了这2个序列的前缀的最长公共子序列。
- 因此，最长公共子序列问题具有**最优子结构性质**。

3.3 动态规划求解LCS

第二步：递归定义最优值

求$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$的一个LCS

- 当$x_m==y_n$时，须找出$LCS(X_{m-1},Y_{n-1})$，然后将$x_m=y_n$添加到LCS上，可以产生一个$LCS(X_m,Y_n)$
- 当$x_m\neq y_n$时，就必须解决两个问题：找出一个$LCS(X_{m-1},Y)$和一个$LCS(X,Y_{n-1})$，这两个LCS中较长的就是最长公共子序列。

由此递归结构可以看成LCS问题具有**重叠子问题性质**。

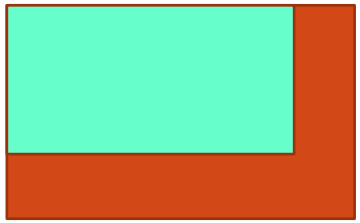
- 例： $LCS(X_{m-1},Y)$和$LCS(X,Y_{n-1})$都包含着$LCS(X_{m-1},Y_{n-1})$的子子问题。

3.3 动态规划求解LCS

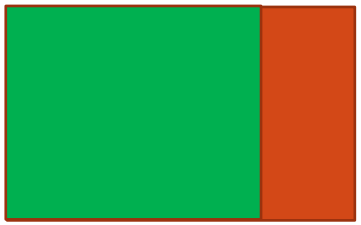
第二步：定义递归值

- 用c[i][j]记录序列X_i和Y_j的**最长**公共子序列的**长度**。
  - 其中，$X_i=\{x_1,x_2,\ldots,x_i\}$；$Y_j=\{y_1,y_2,\ldots,y_j\}$。
- 由最优子结构性质可建立递归关系如下：

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



b[i][j]



3、计算最优值

- 由于在所考虑的子问题空间中，总共有 θ(mn) 个不同的子问题

- 因此，用动态规划算法自底向上地计算最优值能提高算法的效率。

3、计算最优值

c[i][j]：最长

b[i][j]：记录c[i][j]的值是有哪个子问题的解得到的

```
void  LCSLength(int m,int n, char *x,char *y, int **c, int **b)
{
1  for (int i = 0; i <= m; i++)  c[i][0]=0;
2: for (int i = 1; i <= n; i++) c[0][i]=0;
3: for (int i = 1; i <= m; i++)
4:    for (int j = 1; j <= n; j++) {
5:       if (x[i]==y[j])
6:          {c[i][j]=c[i-1][j-1]+1;
7:          b[i][j]='↖'; }
8:       else if (c[i-1][j]>=c[i][j-1])
9:       {  c[i][j]=c[i-1][j];
10:         b[i][j]='↑'; }
11:      else
12:        {   c[i][j]=c[i][j-1];
13:          b[i][j]= '←';}
}}
```

时间复杂度：O（nm）
空间复杂度：O(nm)

练习

- 已知X=<BCADB>,Y=<ABCBAB>,求最长公共子序列，要求画出c[m][n]和b[m][n]矩阵。

		A	B	C	B	A	B
	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
B 1	0	0 ↑	1 ↖	1 ←	1 ↖	1 ←	1 ↖
C 2	0	0 ↑	1 ↑	2 ↖	2 ←	2 ←	2 ←
A 3	0	1 ↖	1 ↑	2 ↑	2 ↑	3 ↖	3 ←
D 4	0	1 ↑	1 ↑	2 ↑	2 ↑	3 ↑	3 ↑
B 5	0	1 ↑	2 ↖	2 ↑	3 ↖	3 ↑	4 ↖

4、构造最长公共子序列

```
lcs(int i,int j,char x[ ],int b[ ][ ])
  {
     if (i ==0 || j==0) return ;
     if (b[i][j]== '↖')
     {
       lcs(i-1,j-1,x,b);
       print(x[i]);
      }
     else if (b[i][j]== '↑')
                lcs(i-1,j,x,b);
       else lcs(i,j-1,x,b);
  }
```

时间复杂度：O（n+m）

课堂练习

- 已知G1=<ABCBDAB>,G2=<BDCABA>,求最长公共子序列，要求画出c[m][n]和b[m][n]矩阵。