6.4.3建立递归关系式

建立递归关系

- 设计算A[i:j]，1≤i≤j≤n，所需要的最少数乘次数m[i,j]，则原问题的最优值为m[1,n]
- 当i=j时，A[i:j]=A_i，因此，m[i,i]=0，i=1,2,…,n
- 当i<j时，

$$m[i,j]=\min_{k}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}$$

这里 A_i 的维数为 $p_{i-1}\times p_i$

可以递归地定义m[i,j]为：

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

k 的位置只有 $j-i$ 种可能

建立递归算法

思考：根据上面的递归式与前面分治递归部分学到的算法设计知识建立递归算法

经分析递归算法消耗指数计算时间

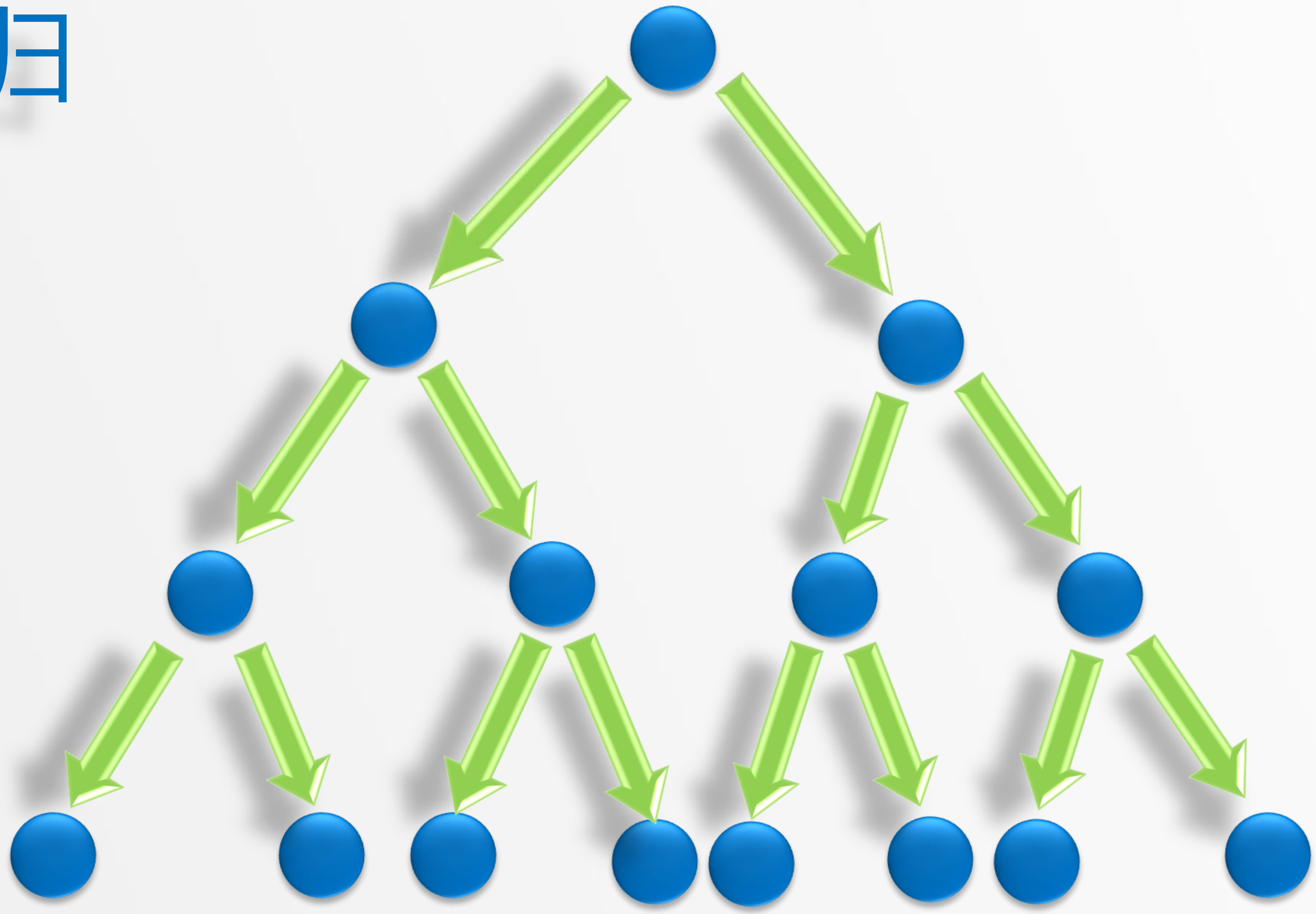
实际子问题数目

- 对于1≤i≤j≤n不同的有序对(i,j)对应于不同的子问题。因此，不同子问题的个数最多只有

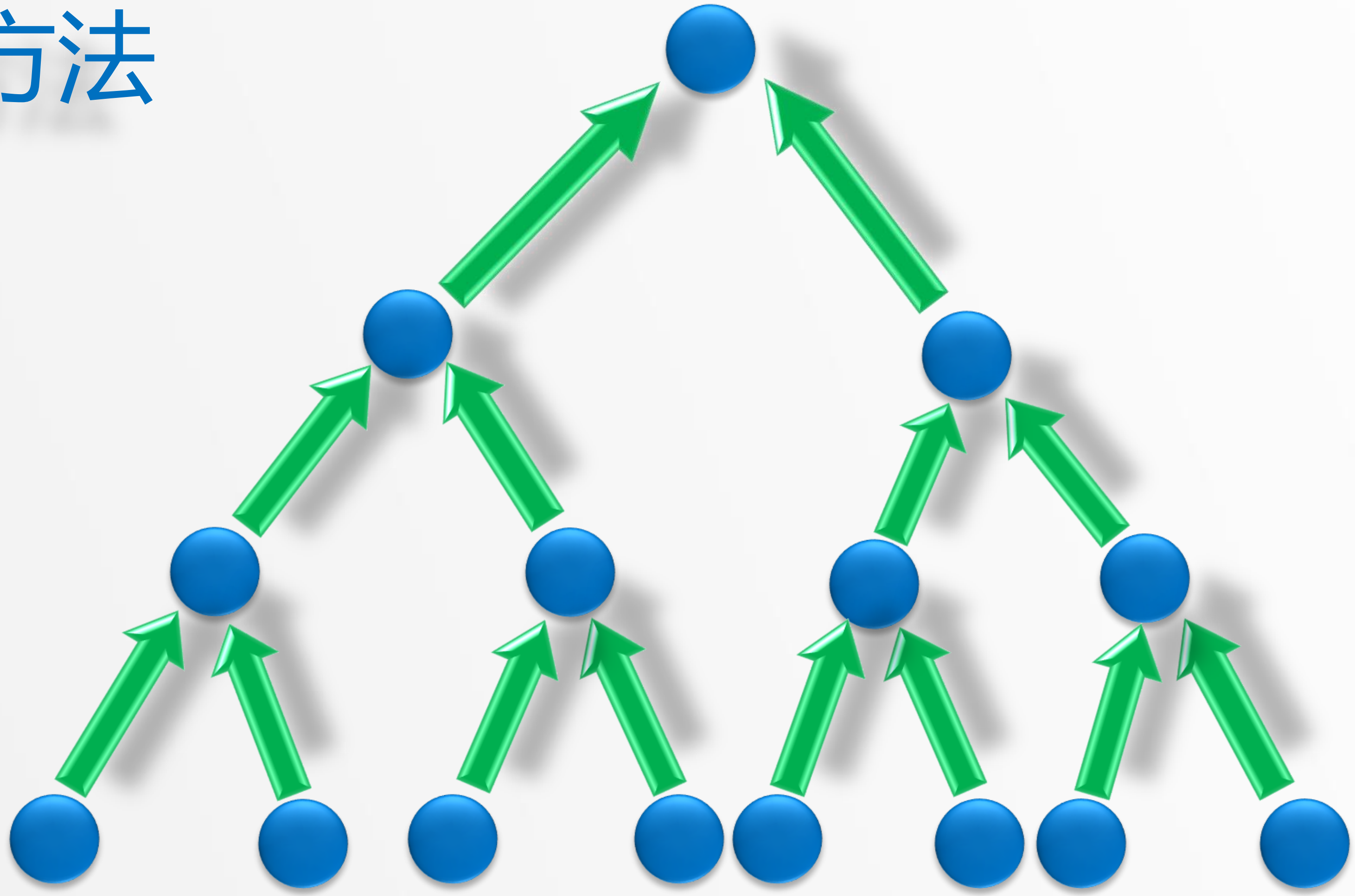
$$\binom{n}{2} + n = \Theta(n^2)$$

- 由此可见，在递归计算时，**许多子问题被重复计算多次。**这也是该问题可用动态规划算法求解的又一显著特征。
- 用动态规划算法解此问题，可依据其递归式以**自底向上**的方式进行计算。在计算过程中，**保存已解决的子问题答案。每个子问题只计算一次**，而在后面需要时只要简单查一下，从而避免大量的重复计算，最终得到多项式时间的算法

递 归



动态规划方法



用动态规划法求最优解

```
public static void matrixChain(int [] p, int [][] m, int [][] s)
 {
   int n=p.length-1;
   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-1]*p[i]*p[j];
       s[i][j] = i;
       for (int k = i+1; k < j; k++) {
         int t = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
         if (t < m[i][j]) {
          m[i][j] = t;
          s[i][j] = k;}
         }
       }
 }
```

用动态规划法求最优解

```
public static void matrixChain(int [] p, int [][] m, int [][] s)
  {
    int n=p.length-1;
    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-1]*p[i]*p[j];
        s[i][j] = i;
        for (int k = i+1; k < j; k++) {
          int t = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
          if (t < m[i][j]) {
            m[i][j] = t;
            s[i][j] = k;}
          }
        }
  }
```

m[i,i]

m[i,i+r]

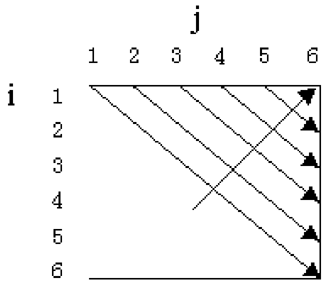
算法复杂度分析：
算法matrixChain的主要计算量取决于算法中对r，i和k的3重循环。循环体内的计算量为O(1)，而3重循环的总次数为O(n3)。因此算法的计算时间上界为O(n3)。算法所占用的空间显然为O(n2)。

用动态规划法求最优解

例子：下表6个矩阵连乘问题

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25

$$m[2][5]=\min\begin{cases} m[2][2]+m[3][5]+p_1p_2p_5 = 0+2500+35\times15\times20=13000 \\ m[2][3]+m[4][5]+p_1p_3p_5 = 2625+1000+35\times5\times20=7125 \\ m[2][4]+m[5][5]+p_1p_4p_5 = 4375+0+35\times10\times20=11375 \end{cases}$$



(a) 计算次序

i \ j	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

(b) m[i][j]

i \ j	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

(c) s[i][j]