6.5动态规划基本要素

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，所用的方法具有普遍性：首先假设由问题的最优解导出的子问题的解不是最优的，然后再设法说明在这个假设下可构造出比原问题最优解更好的解，从而导致矛盾。
- 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。

注意：同一个问题可以有多种方式刻划它的最优子结构，有些表示方法的求解速度更快（空间占用小，问题的维度低）

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**。

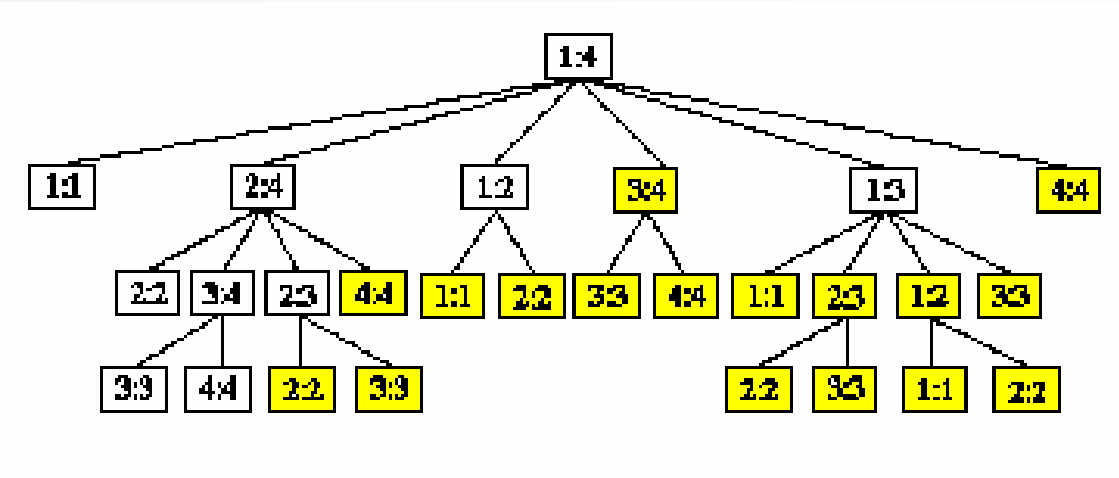
- 动态规划算法，对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。

- 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。

二、重叠子问题

考虑用递归式直接计算矩阵连乘中的A[i:j]

```
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=recurMatricChain(i,k)+ recurMatricChain(k+1,j)+p[i-1]*p[k]*p[j]
If(t<u){
u=t;
S[i][j]=k;
}
Return u;
```



二、重叠子问题

- 计算时间T(n)有指数下界：
- 可以推得T(n)的递归关系式如下：

$$T(n)\geq\begin{cases}O(1), n=1\\ 1+\sum_{k=1}^{n-1}(T(k)+T(n-k)+1), n>1\end{cases}$$

$$T(n)\geq 1+(n-1)+\sum_{k=1}^{n-1}T(k)+\sum_{k=1}^{n-1}T(n-k)=n+2\sum_{k=1}^{n-1}T(k)$$

- 可用数学归纳法证明 $T(n)\geq 2^{n-1}=\Omega(2^n)$
- 因此直接递归算法的计算时间随n指数增长，相比之下动态规划算法计算复杂度要低，其有效性就在于它充分利用了问题的重叠性质

二、重叠子问题

- 计算时间T(n)有指数下界：
- 可以推得T(n)的递归关系式如下：

$$\begin{cases} O(1) & n=1 \\ \end{cases}$$

可不可以在递归算法中
通过添加表记录已计算结果，减少计算量?

$$T(n) \geq 1+(n-1)+\sum_{k=1} T(k)+\sum_{k=1} T(n-k) = n+2\sum_{k=1} T(k)$$

- 可用数学归纳法证明 $T(n) \geq 2^{n-1} = \Omega(2^n)$
- 因此直接递归算法的计算时间随n指数增长，相比之下动态规划算法计算复杂度要低，其有效性就在于它充分利用了问题的重叠性质

三、备忘录方法

- 备忘录方法的控制结构与直接递归方法的控制结构相同，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。

备忘录：自顶向下！　　动态规划：自底向上！

- 备忘录方法为每个子问题建立一个记录项，初始化时，该记录项存入一个特殊值，表示该子问题尚未求解。在求解过程中对每个待求子问题，首先查看其相应的记录项。若记录项中存储的是初始化时存入的特殊值，则表示该子问题是第一次遇到，此时计算出该子问题的解，并保存在其相应的记录项中，备以后查看，若记录项中存储的已不是初始化时存入的特殊值，表示该问题已经被计算过，此时只要从记录项中取出该问题的解答即可。

递归算法

```
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=recurMatricChain(i,k)+ recurMatricChain(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

```
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=recurMatricChain(i,k)+ recurMatricChain(k+1,j)+p[i-1]*p[k]*p[j]
If(t<u){
u=t;
S[i][j]=k;
}
Return u;
```

以递归方式进行计算

三、备忘录方法

用备忘录方法解矩阵连乘问题

```
m←0
private static int lookupChain(int i, int j)
  {
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = 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;
  }
```

三、备忘录方法

用备忘录方法解矩阵连乘问题

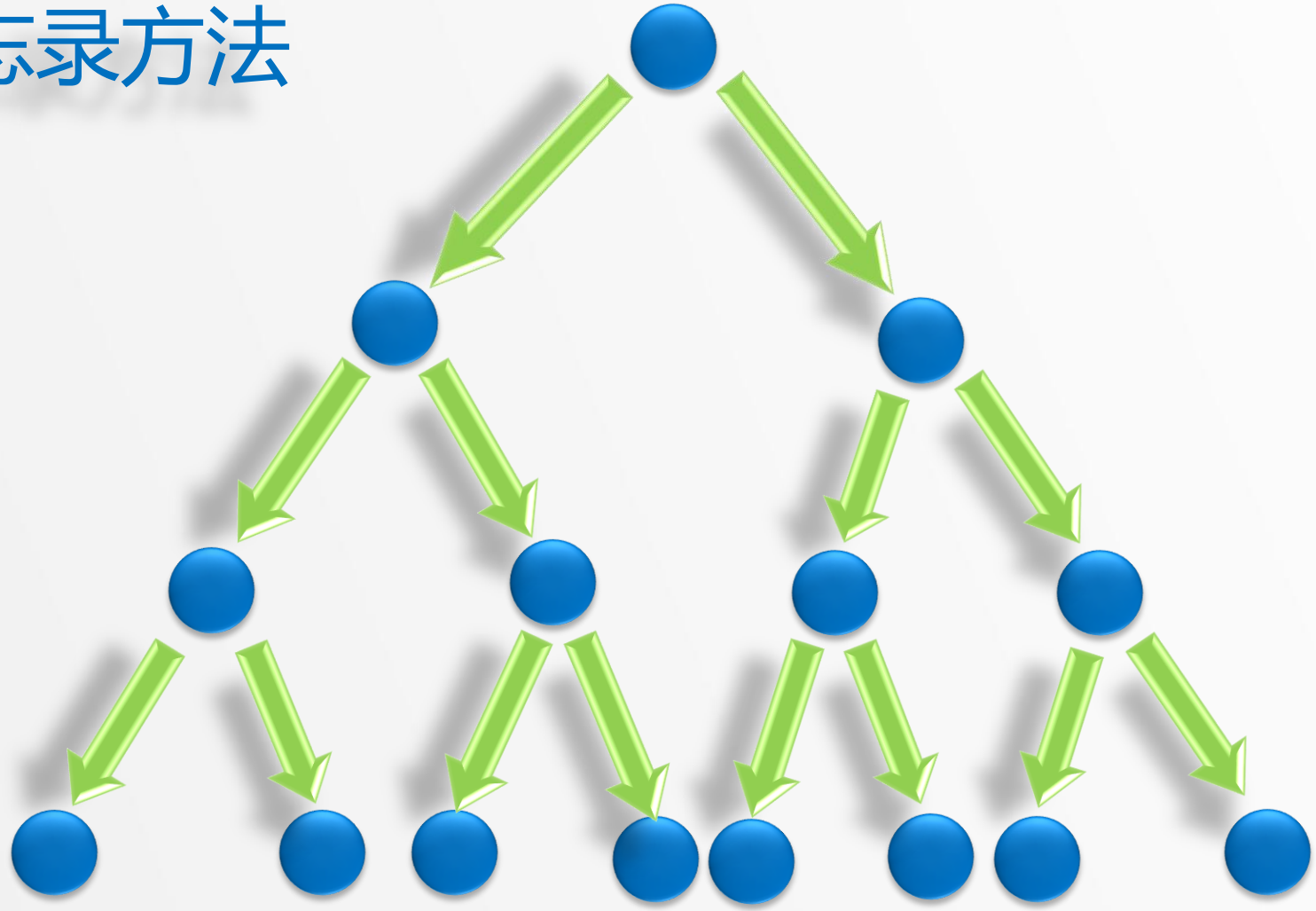
```
m←0
private static int lookupChain(int i, int j)
  {
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = lookupChain(i+1,j) + p[i-1]*p[i]*p[j];
    s[i][j] = i;
    for (int k = i+1; k < j; k++) {
      int t = lookupChain(i,k) + lookupChain(k+1,j) + p[i-1]*p[k]*p[j];
    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

已经计算过直接返回

计算复杂度： $O(n^3)$

以递归方式进行计算

递归与备忘录方法



用动态规划法

```
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

```
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;}
            }
         }
}
```

计算最小规模子问题

计算规模从2到n,规模逐渐增大的各子问题

动态规划方法