6.6备忘录方法

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

以递归方式进行计算

计算复杂度： $O(n^3)$

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

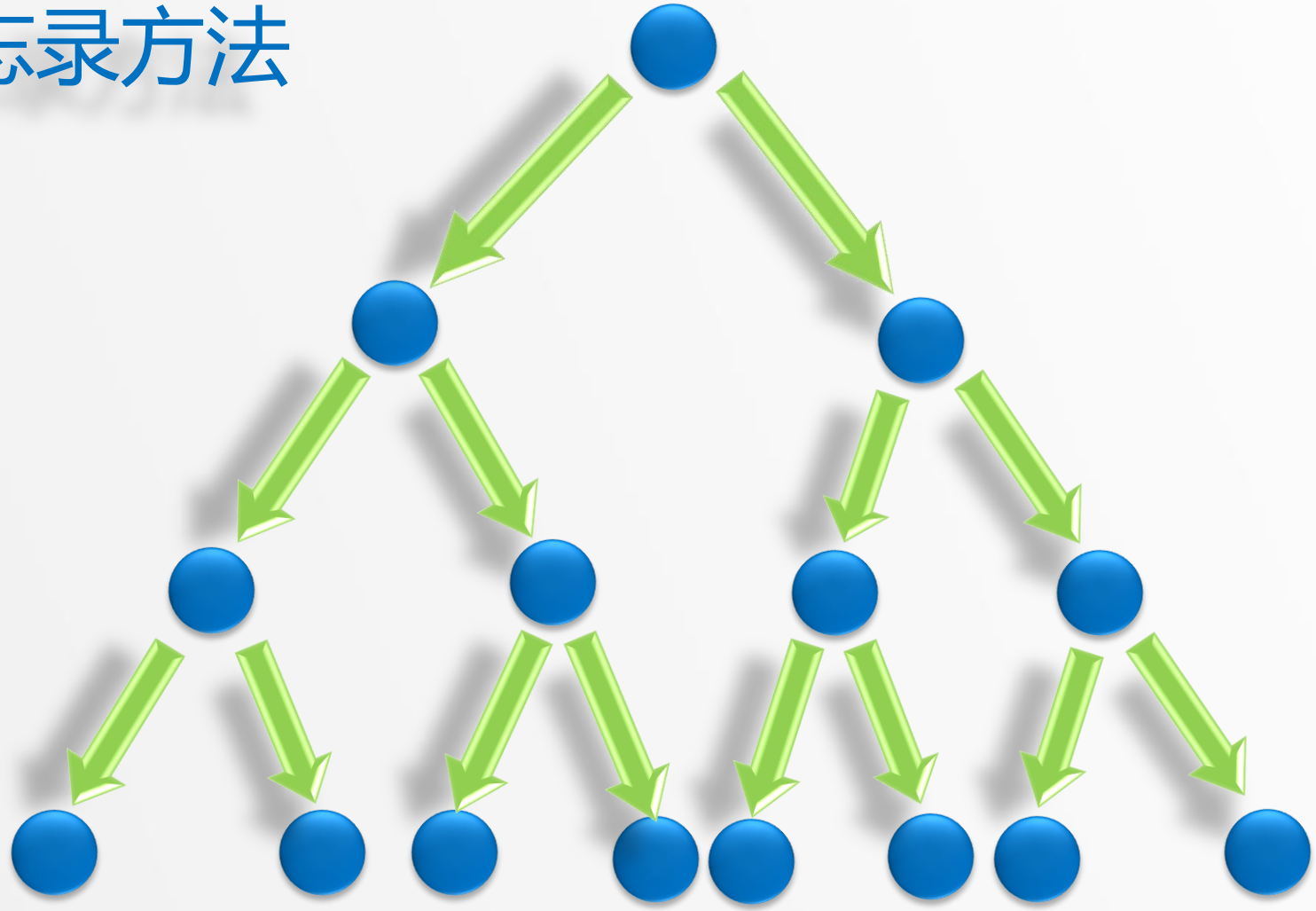
```
m←0
private static int lookupChain(int i, int j)
  {
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = lookupChain(i+1,j) + p[i-1]*p[i]*p[j];
    s[i][j] = i;
    for (int k = i+1; k < j; k++) {
      int t = lookupChain(i,k) + lookupChain(k+1,j) + p[i-1]*p[k]*p[j];
    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

已经计算过直接返回

以递归方式进行计算

计算复杂度： $O(n^3)$

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;}
            }
         }
}
```

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