动态规划算法

张里博

lbzhang@swu.edu.cn

目录

PART 01

动态规划算法的设计思想

Dynamic Programming

动态规划的设计思想

动态规划算法

动态规划(dynamic programming)算法是一种求解**多阶段决策问题**的算法设计技术。动态规划是运筹学的一个分支，也是求解决策过程(decision process)最优化问题的数学方法。

主要思想：将原问题归约为**规模较小、结构相同**的子问题，建立原问题与子问题**优化函数间的依赖关系；从规模最小的子问题开始**，利用上述依赖关系求解规模**更大的子问题**，直至得到原问题的解。

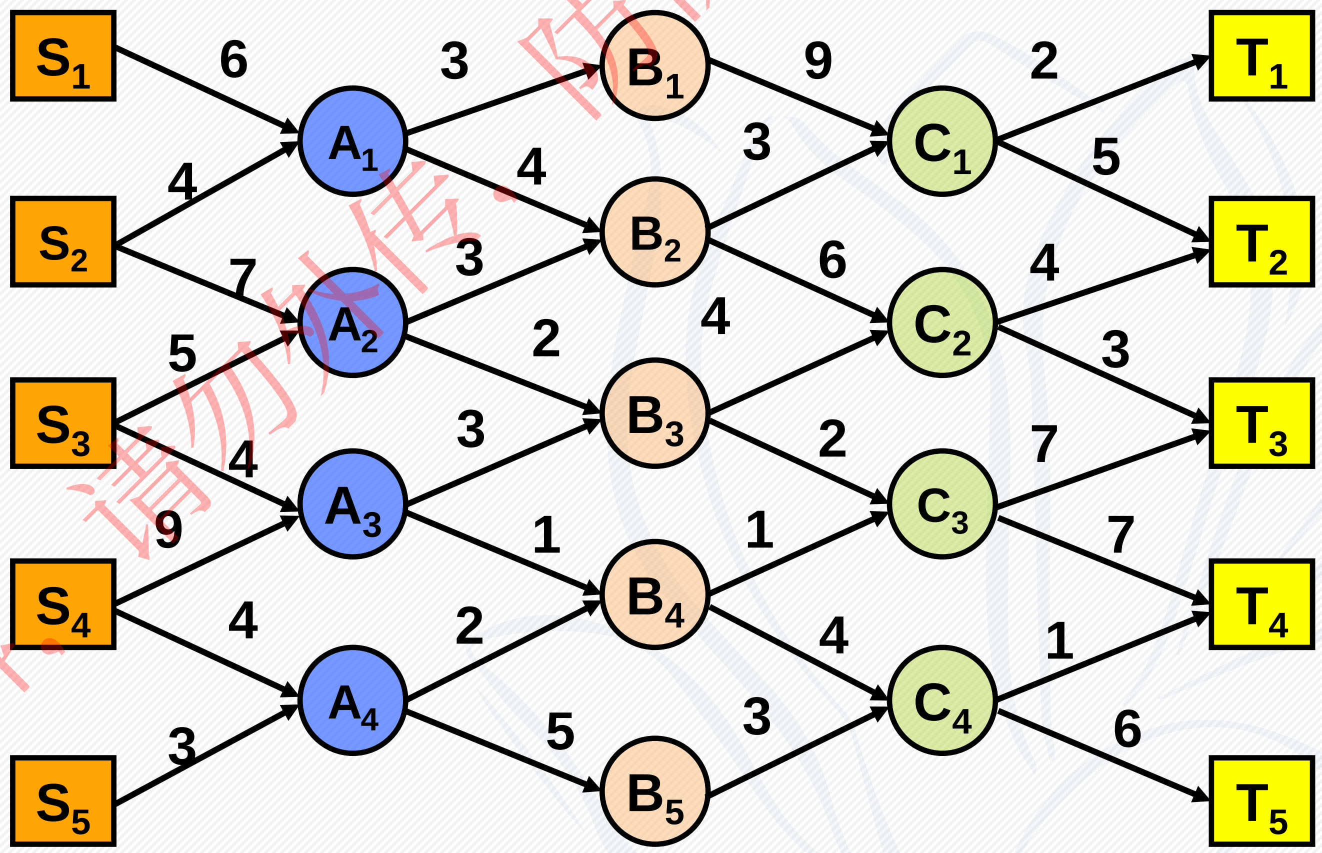
问题描述

输入：

- 起点集合$\{S_1, S_2, ..., S_n\}$，
- 终点集合$\{T_1, T_2, ..., T_m\}$，
- 中间结点集，
- 边集E，其中任意边e有长度；

输出：

一条从（任意）起点到（任意）终点的最短路径（路径总长度最短）



蛮力算法

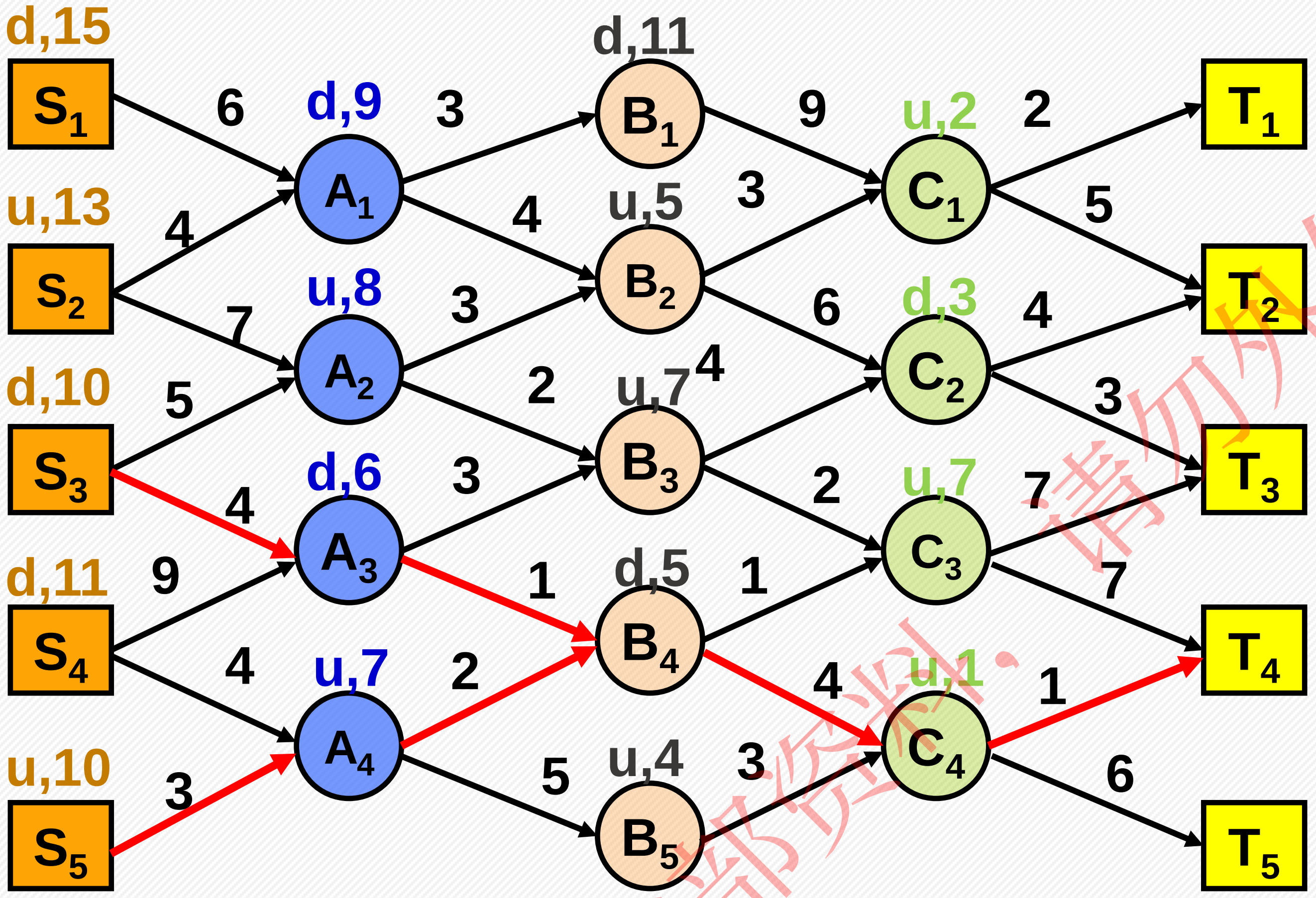
- 计算每一条所有从起点到终点的路径，计算长度，从其中找出最短路径。

- 在上述实例中，如果网络的层数为n，起点个数为m个，那么每个起点到所有终点路径条数将接近于2^{n-1}（每个结点最多有2个分支与下一层相连），总时间复杂度为$O(m2^n)$

- 动态规划算法：
- 从终点出发，起点逐步前移，每一步计算当前层的节点与终点的最短路径及长度。将**多阶段决策过程转化成多个单阶段决策过程**。

最短路径问题

$$F(C_1)=\min\{C_1T_2, C_1T_1\}$$



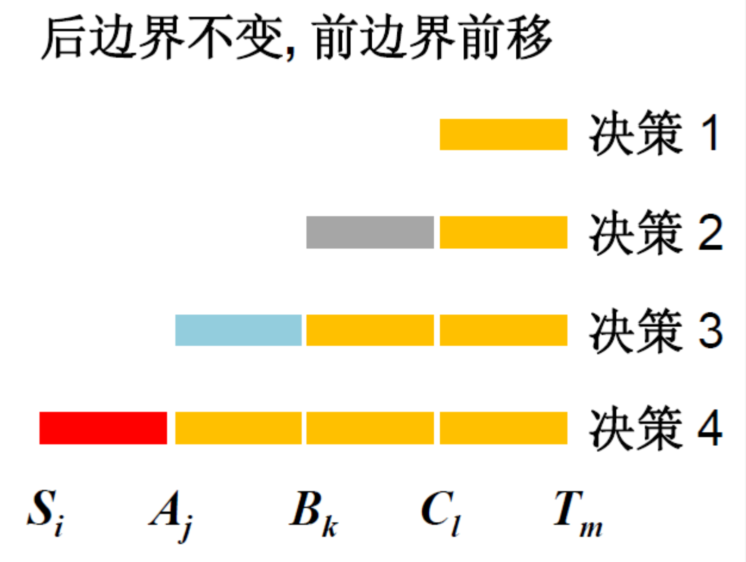
$$F(B_2)=\min\{B_2C_1+F(C_1), B_2C_2+F(C_2)\}$$

含弘光大　继往开来

$$=\min\{\min\{B_2C_1+C_1T_2, B_2C_1+C_1T_1\}, \min\{B_2C_2+C_2T_2, B_2C_2+C_2T_3\}\}$$

子问题的界定

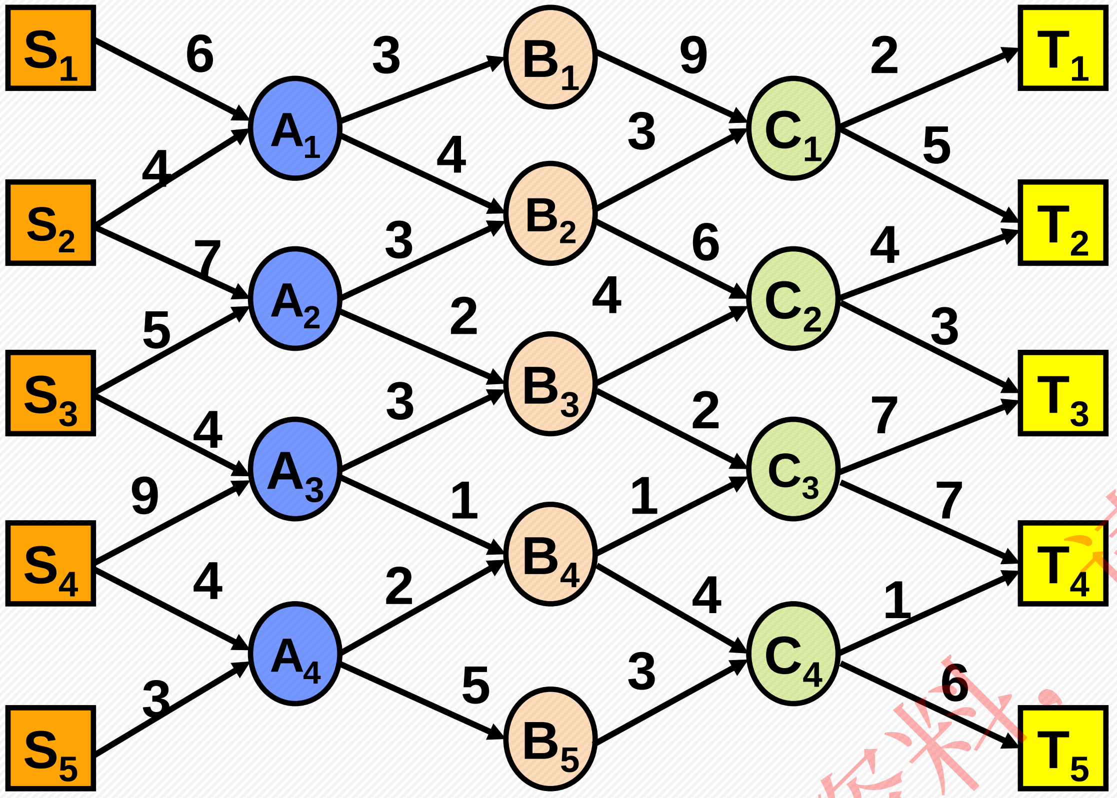
从终点层出发，起点逐步前移，每步计算当前层节点与终点的最短路径及长度



每步决策将依赖于以前步骤的决策结果。

最短路径问题

$$F(B_2)=\min\{B_2C_1+F(C_1),B_2C_2+F(C_2)\}$$



动态规划只考虑当前子问题的最优解延伸的结果，许多不可能最优的路径已被删除。

最短路径长度的依赖关系

决策一 $F(C_l)=\min\limits_{m}\{C_lT_m\}$

决策二 $F(B_k)=\min\limits_{l}\{B_kC_l+F(C_l)\}$

决策三 $F(A_j)=\min\limits_{k}\{A_jB_k+F(B_k)\}$

决策四 $F(S_i)=\min\limits_{j}\{S_iA_j+F(A_j)\}$

时间复杂度 O(3*m*(n-1))=O(mn)

每步求解的问题是后面阶段求解问题的子问题。

问题特点

- 例3.1动态规划算法的特征：
- 1.求解问题是多阶段决策问题；
- 2.求解过程是多步判断（多个子问题），每步求解的问题是后面阶段求解问题的子问题，最后一步求解出原问题的解；
- 3.前面一个子问题的最小值与后面问题的最小值有依赖关系。

$$F(B_2) = \min\{B_2C_1 + F(C_1), B_2C_2 + F(C_2)\}$$

PART 02

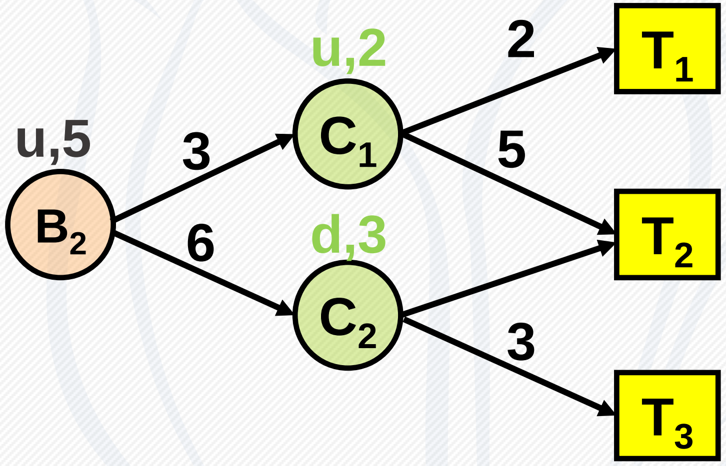
动态规划算法的必要条件

Dynamic Programming

动态规划的必要条件

优化函数的特点：

任何最短路径的子路径都是相对于子路径始点和终点的最短路径



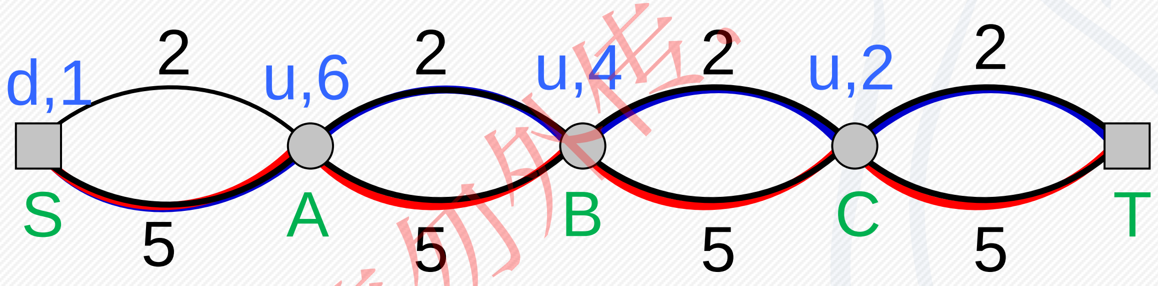
优化原则（最优子结构性质）

一个最优决策序列的任何子序列本身，一定是相对于子序列的初始和结束状态的最优决策序列

分析和证明问题的最优子结构性质时，一般采用反证法

动态规划的必要条件

例：求总长模10的最小路径



动态规划算法的解：下、上、上、上； ≠ 实际最优解：下、下、下、下；

实际最优解不满足优化原则：

全局最优路径在A->T和C->T的子问题中不是最优；

因此，不能使用动态规划算法。

动态规划的设计要素

- 1. 如何划分子问题（边界）？
- 2. 原问题的优化函数值与子问题的优化函数值存在着什么依赖关系？
- 3. 是否满足优化原则？
- 4. 递推方程是什么？初值等于什么？

PART 03

动态规划算法的递归实现

Dynamic Programming

矩阵A： i行j列，矩阵B: j行k列
以元素相乘作基本运算，计算 $C=AB$的工作量

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} \\ a_{21} & a_{22} & \cdots & a_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{j1} & b_{j2} & \cdots & b_{jk} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i1} & c_{i2} & \cdots & c_{ik} \end{bmatrix}$$

$$c_{mn} = \sum_{l=1}^{j} a_{ml} b_{ln}$$

矩阵C: i行 k列，元素个数为 ik，计算每个元素需要做j次乘法和j-1次加法，
总乘法次数为 ijk

一个实例

- 实例： $P = <10, 100, 5, 50>$
- A_1: 10×100, A_2: 100×5, A_3: 5×50

$$\overbrace{(\underbrace{(A_1 X A_2)}_{10 X 100 X 5 = 5000}\ X\ A_3)}^{10 X 5 X 50 = 2500}$$

$$\overbrace{(A_1\ X\ (A_2}^{10X\ 100\ X\ 50 = 50000}\ \underbrace{X\ A_3))}_{100 X 5 X 50 = 25000}$$

- 乘法次序：
- $((A_1 A_2)A_3)$: $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$ ✓
- $(A_1(A_2 A_3))$: $10 \times 100 \times 50 + 100 \times 5 \times 50 = 75000$

为什么3个矩阵要用2个括号？

例3.3 矩阵链相乘

- 问题：设$A_1, A_2, \ldots, A_n$为矩阵序列，A_i为 $P_{i-1} \times P_i$阶矩阵，$i = 1, 2, \ldots, n$. 确定乘法次序使得元素相乘的总次数最少。

- 输入：向量 $P = <P_0, P_1, \ldots, P_n>$，其中，$P_0, P_1, \ldots, P_n$是$n$个对应矩阵的行数和列数。

- 输出：使得元素相乘的总次数最少的矩阵链乘法加括号的位置。

列举出所有可能的计算次序，从中找出数乘次数最少的情况

算法复杂度分析

每加一个括号原问题就可以分解为两个子矩阵的加括号问题。

假设n个矩阵的连乘问题中，不同的计算次序总数为P(n)。假设最后一次相乘发生在第k(1≤k≤n-1)个矩阵的位置，即$(A_1 \ldots A_k)(A_{k+1} \ldots A_n)$，则P(n)= P(k)P(n-k). 因此，P(n)可表示为:

$$P(n) = \begin{cases} 0 & , n=1; \\ \sum_{k=1}^{n-1} P(k)P(n-k), & n>1. \end{cases}$$

n个矩阵相乘，需要加n-1对括号，P(n) 是一个Catalan数 C(n-1):

$$P(n) = \frac{1}{n} C_{n-1}^{2(n-1)} = \Omega(4^n / n^{3/2})$$

子问题划分

- 问题：$A_{i..j}$ $(i<j)$表示矩阵链 $A_i A_{i+1} \ldots A_j$ 的乘积，矩阵的维数存储向量：　$P=< P_{i-1}, P_i, \ldots, P_j>$

- 假设已知最优划分的运算次数为$m[i, j]$，并且最后一次相乘发生在第k个矩阵的位置，即

$$A_{i..j} = A_{i..k} A_{k+1..j}$$

- 只有在矩阵子链 $A_{i..k}$和$A_{k+1..j}$取得最优次序（子问题最优解）时，$A_{i..j}$才能取得最优次序（原问题的最优解）

含弘光大　继往开来

优化原则证明

- 已知原问题$A_{i..j}$最优划分导致的运算次数为$m[i, j]$，那么原问题$A_{i..j}$的最优次序（原问题的最优解）所包含的子问题$A_{i..k}$和$A_{k+1..j}$中的次序也是最优的（子问题最优解）。 $f[i,k]$ $f[k+1, j]$

反证法：

假设子问题$A_{i..k}$存在（比原问题的最优解在该问题上的解）更优的解，其运算次数为$f'[i, k](<f[i, k])$，则原问题一个新的可行解（$A_{i..k}$“新解”+ $A_{k+1...j}$ “老解”），其运算次数为：

$$f'[i, j] = f'[i,k] + f[k+1, j] + P_{i-1}P_kP_j$$

$f'[i, j] < m[i, j]$，与已知矛盾。

递推方程

- 已知原问题$A_{i\,..\,j}$最优划分导致的运算次数为$m[i, j]$，那么原问题$A_{i\,..\,j}$的最优次序（原问题的最优解）所包含的子问题$A_{i\,..\,k}$和$A_{k+1\,..\,j}$中的次序也是最优的（子问题最优解），因此

$$m[i, j] = m[i, k] + m[k+1, j] + P_{i-1}P_kP_j$$

- **递推方程和初值（最后一次相乘的位置k未知）：**

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

遍历所有可能划分的位置

算法3.1 RecurMatrixChain (P, i, j)

首次调用，i←1, j ←n

```
if i==j then
    m[i,j]←0;  s[i,j]←i;  return m[i,j]
end
m[i,j]←∞;  s[i,j]←i;
for k←i to j–1 do
    q ← RecurMatrixChain( P, i, k )+ RecurMatrixChain(P, k+1, j ) + p_{i–1} p_k p_j;
    if q < m[i,j] then
        m[i,j]←q;  s[i,j]←k;
    end
end
return m[i,j]
```

单个矩阵，直接返回

划分位置$k=i$

遍历所有可能划分的位置

找到更好的解

设立标记函数

为了追踪最优解，设计表$s[i,j]$，记录求得最优运算次序中**最后一次乘法**的位置

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

时间复杂度的递推方程

$$T(n) \ge \begin{cases} 0 & n = 1 \\ \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1)) & n > 1 \end{cases}$$

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

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

定理3.1：对于$n>1$，$T(n)=\Omega(2^{n-1})$

数学归纳法证明.

当$n=2$，$T(2) \geq c = c_1 2^{2-1}$, $c_1 = c/2$为某个正数

假设对于任何小于n 的 k 命题为真，则存在c'使得

$$T(n) \geq c'n + 2\sum_{k=1}^{n-1} T(k)$$ 代入归纳假设

$$\geq c'n + 2\sum_{k=2}^{n-1} c_1 2^{k-1}$$ 等比数列求和

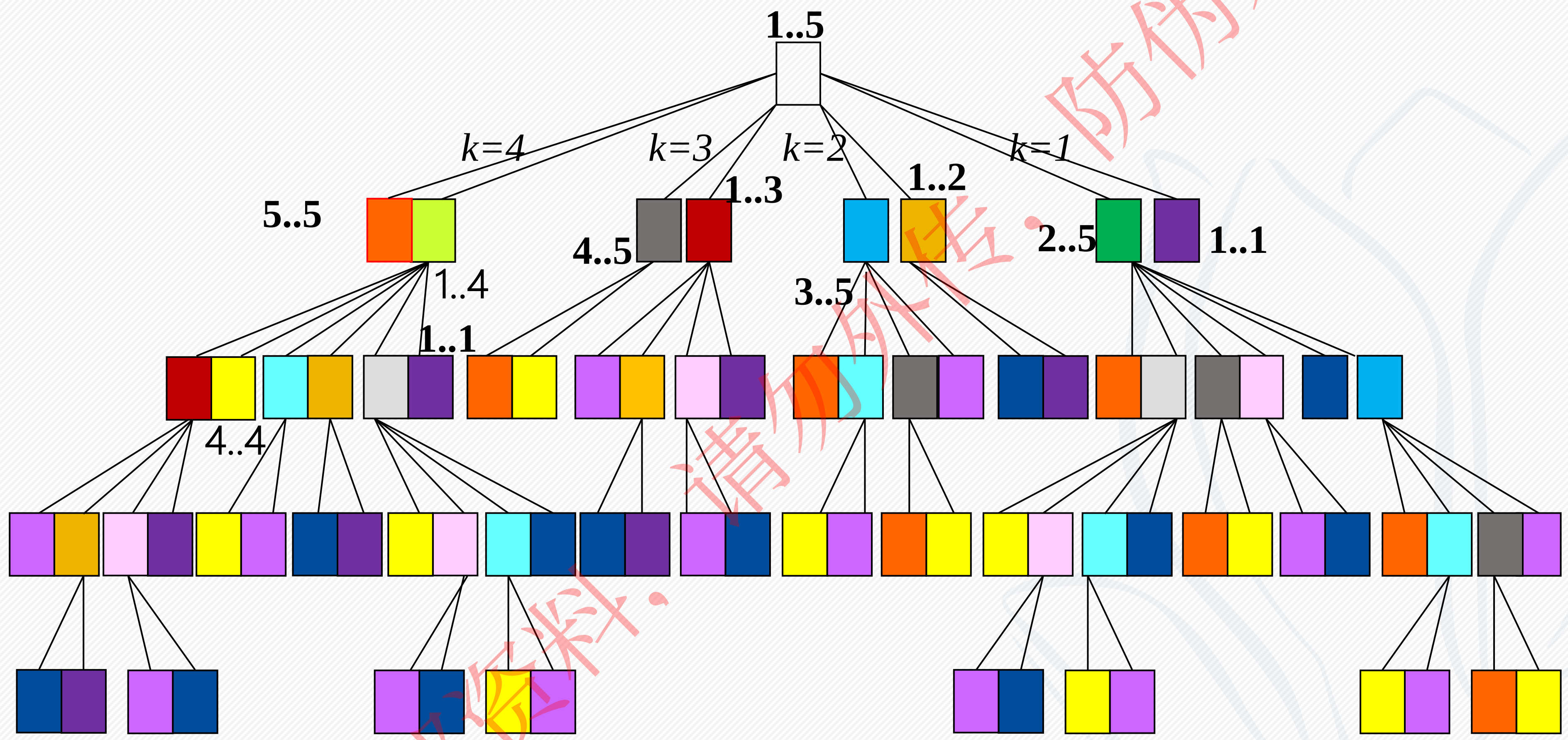
$$= c'n + c_1(2^n - 4) \geq c_1 2^{n-1}$$

PART 04

动态规划算法的递归+备忘录实现

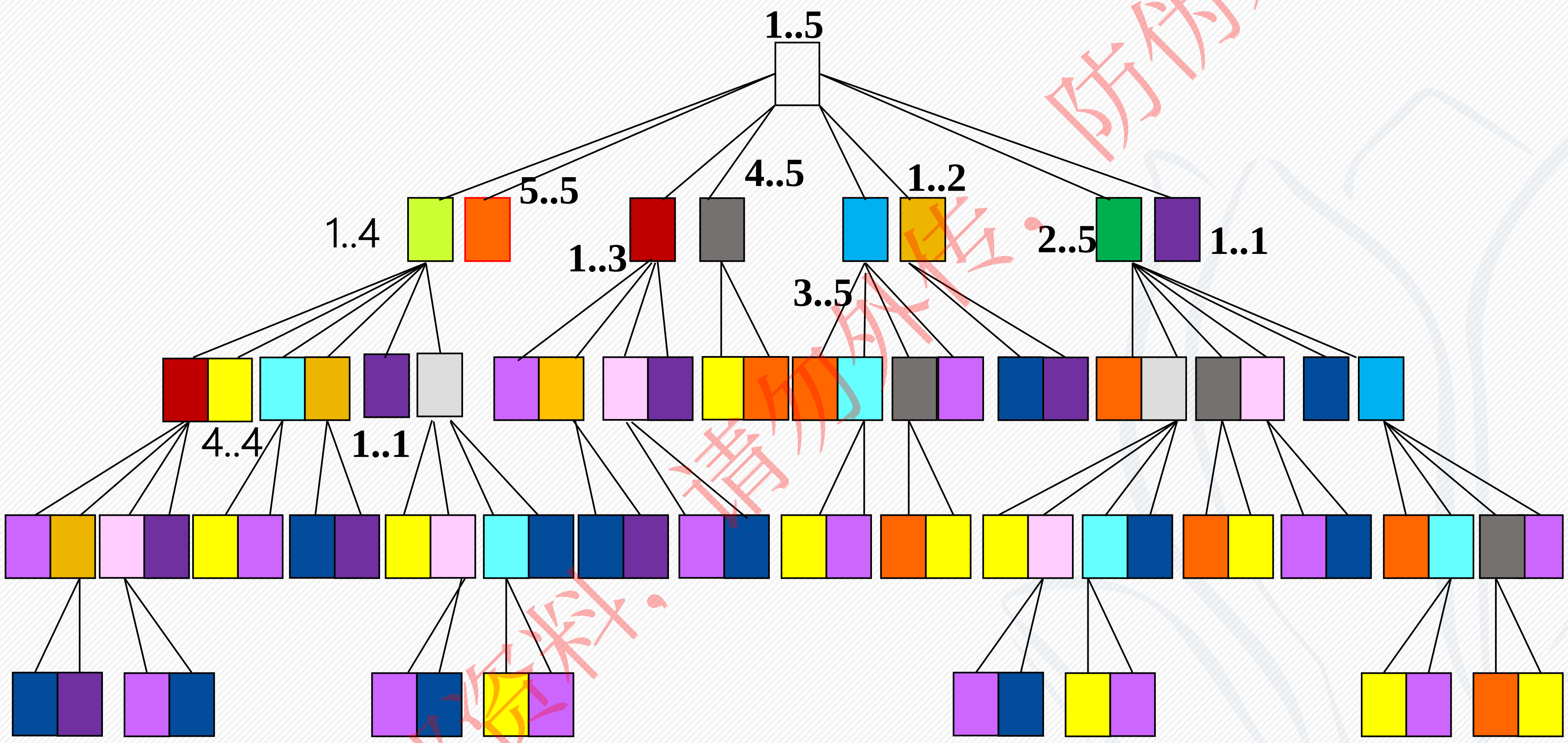
Dynamic Programming

子问题的产生



n=5，第一层8个子问题，节点8个；
子节点共计：81个；

子问题的产生



n=5，第一层8个子问题，节点8个；

子节点共计：81个；

子问题的类别

n=5

子问题的种类

子问题	1-1	2-2	3-3	4-4	5-5
个数	**8**	**12**	**14**	**12**	**8**
子问题	1-2	2-3	3-4	4-5	
个数	**4**	**5**	**5**	**4**	
子问题	1-3	2-4	3-5		
个数	**2**	**2**	**2**		
子问题	1-4	2-5			
个数	**1**	**1**			
子问题	1-5				
个数	**1**				

n=5，计算子问题：81个；
不同的子问题：15个

动态规划算法的递归实现效率不高，
原因：同一子问题多次重复出现，
每次出现都需要重新计算一遍

递归+备忘录

Those who cannot remember the past are doomed to repeat it.

-----George Santayana,
The life of Reason: Introduction and Reason in Common Sense

- 如果保存已解决的子问题的答案，在需要时直接查找和调用，就可以避免重复计算，提高算法效率。

- 改进途径一：
- 开辟一个存储空间（“**备忘录**”），记录子问题的**划分边界（标记函数，最优解）和优化函数值（元素计算次数，最优值）**。
- 计算子问题时，先检索备忘录，如有就直接调用，否则计算并记录。
- 通过备忘录提高效率的同时，增大了空间开销。

递归+备忘录

首次调用，i←1, j ←n, m←0, s[i][j]←0

```
lookupChain(p, i, j)
if (i == j) then
    return 0
end
if m[i][j] > 0 then
    return m[i][j]
end
u ←0+ lookupChain(p,i+1,j) + p[i-1]*p[i]*p[j];
s[i][j] ← i;
for k ← i+1 to j-1 do
    t ← lookupChain(p,i,k) + lookupChain(p,k+1,j) + p[i-1]*p[k]*p[j];
    if (t < u) then
        u ← t; s[i][j] ← k;
    end
end
m[i][j] ← u;        // m矩阵保存在内存中
return m[i][j]
```

单个矩阵，直接返回

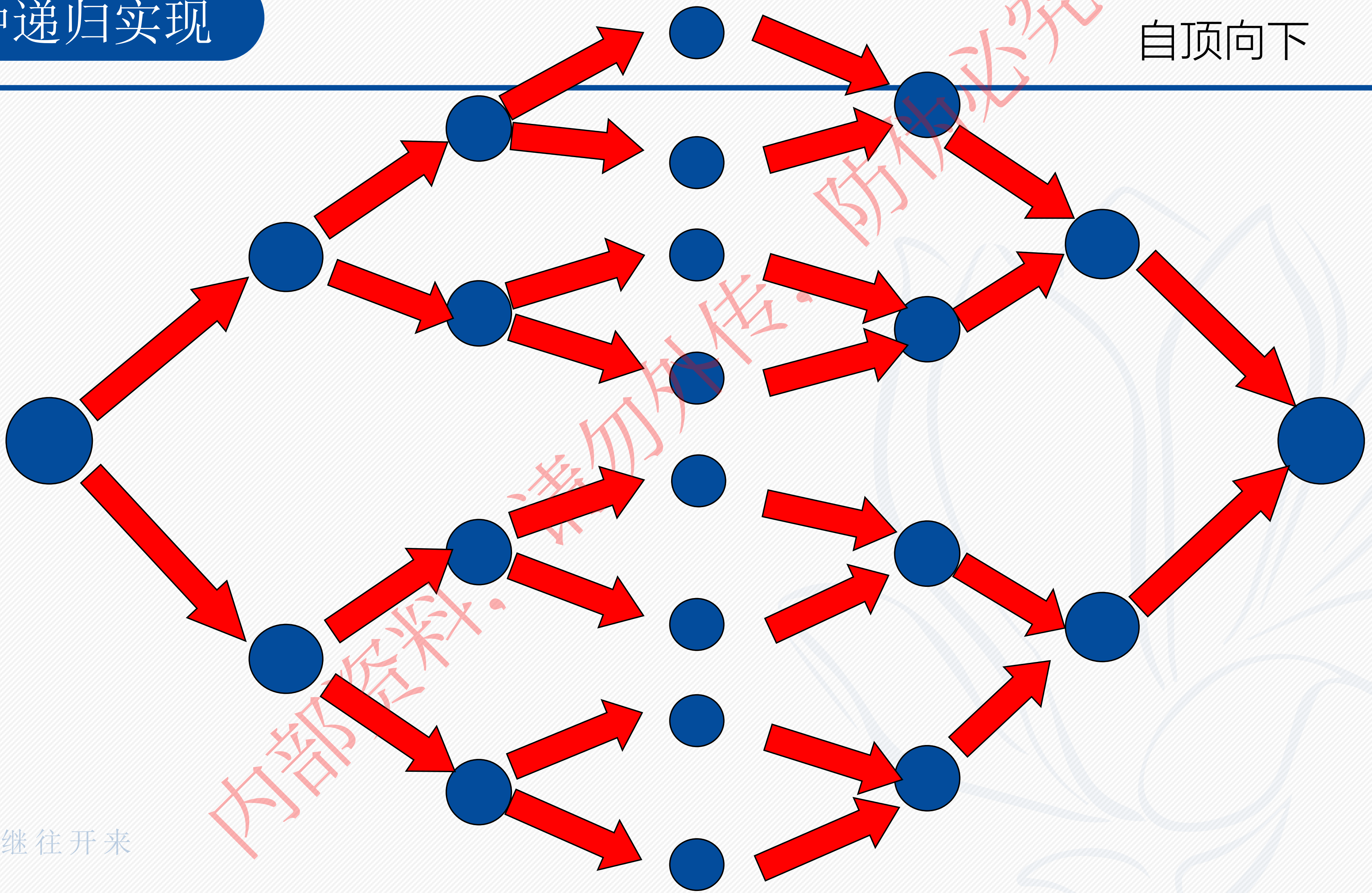
子问题若已经计算过，则直接查找后返回

如果最后划分位置k取i

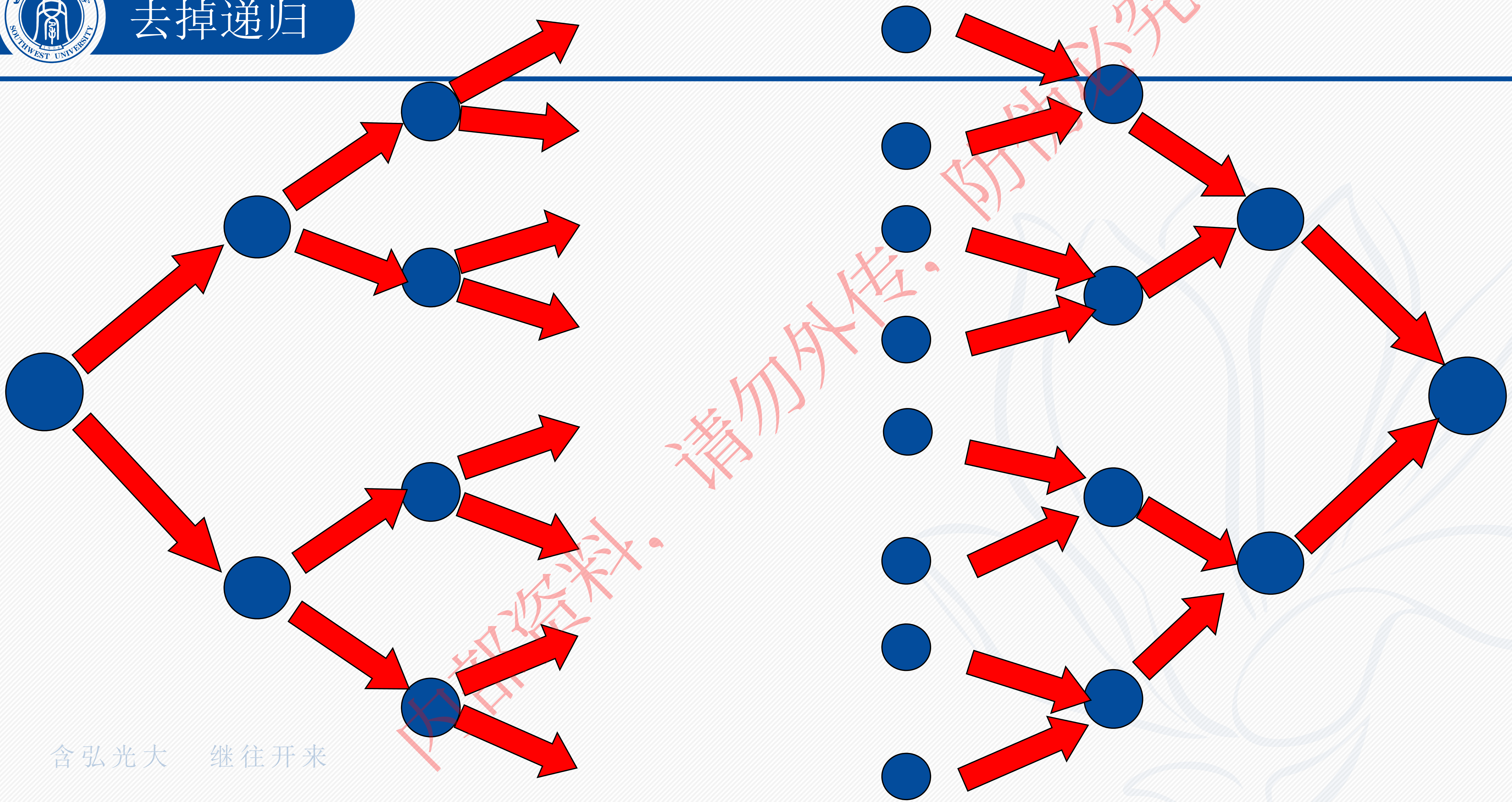
找到更好（工作量更小）的划分，就替换原有的值

遍历所有可能划分的位置

两种递归实现



去掉递归



PART 05

动态规划算法的迭代实现

Dynamic Programming

迭代实现

- 改进途径二：
- 设计自底向上的求解过程（计算顺序），保证后面用到的值前面已经计算好。

- 从最小规模（只有一个矩阵，最小子问题）的问题开始算起，直到规模为n的子问题（原问题），保证每个问题只计算一次。

- 将已求解的小规模问题的解合并成为一个更大规模问题的解，自底向上逐步得到原问题解

迭代实现

- 子问题的类别
- 长度为1： 只含1 个矩阵，有n 个不同的子问题(不需要计算)；
- 长度为2： 含2 个矩阵，n-1 个不同的子问题；
- 长度为3： 含3 个矩阵，n-2 个不同的子问题；
- …
- 长度为n-1： 含n-1 个矩阵，2 个不同的子问题；
- 长度为n： 原始问题，只有1 个；

迭代顺序

- 长度为1： 初值， m[i][i]=0；
- 长度为2 ： 1..2, 2..3, 3..4, ..., n-1..n；
- 每个问题中最后一次相乘发生的位置有1种情况
- 长度为3 ： 1..3, 2..4, 3..5, ..., n-2..n；
- 每个问题中最后一次相乘发生的位置有2种情况
- …
- 长度为n-1： 1.. n-1, 2..n；
- 长度为n ： 1.. n；

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

迭代实现

n=5时的实例

$$\min_{i \le k < j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\}$$

子问题	1-1	2-2	3-3	4-4	5-5
子问题	1-2	2-3	3-4	4-5	
子问题	1-3	2-4	3-5		
子问题	1-4	2-5			
子问题	1-5				

k=3, 子问题: 3..3, 4..5

k=4, 子问题: 3..4, 5..5

k=2, 子问题: 2..2, 3..5

k=3, 子问题: 2..3, 4..5

k=4, 子问题: 2..4, 5..5

伪码表示

算法3.2 MatrixChain(*P*, *n*)

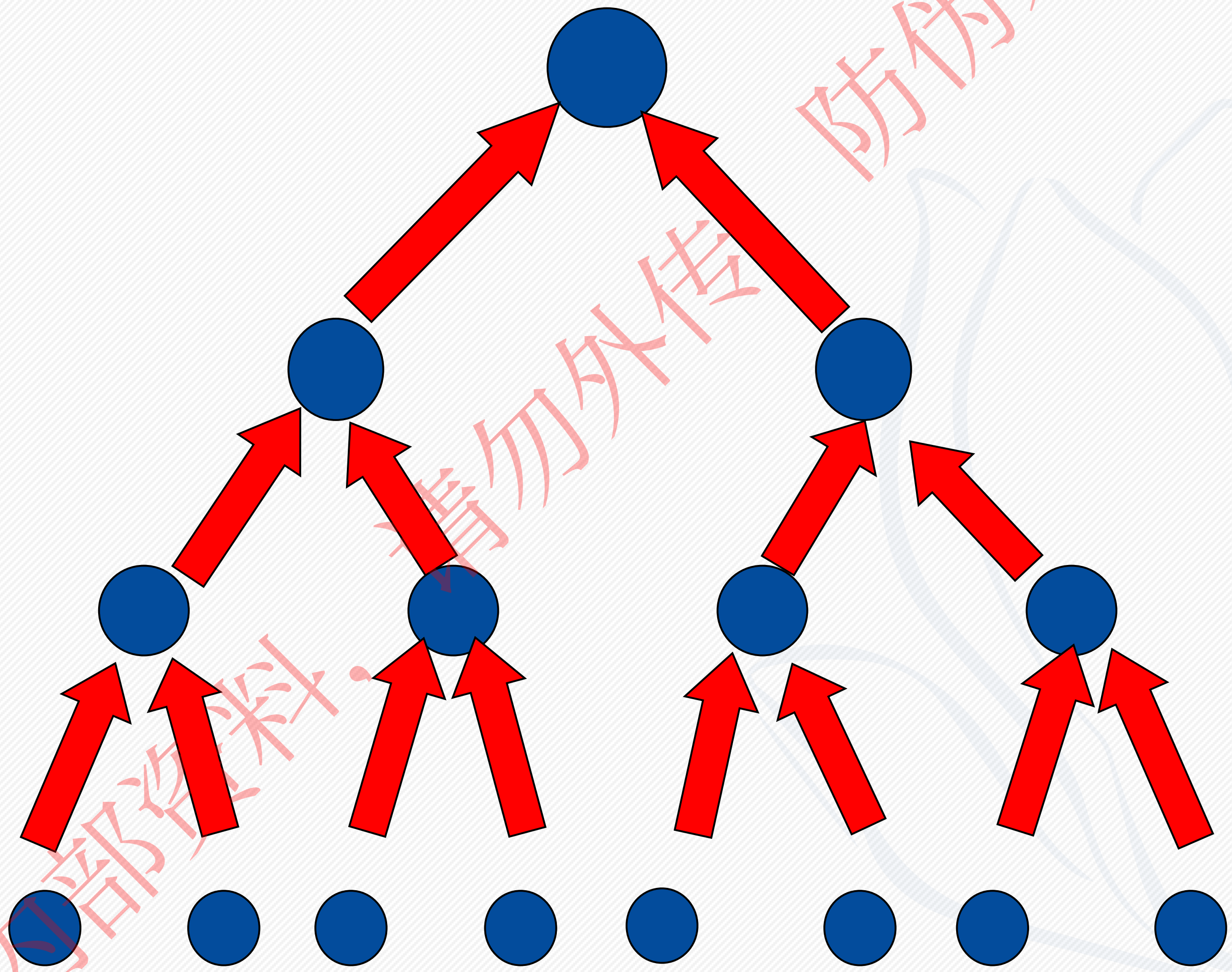
```
1.  令所有的 m[i, i]初值为0      1≤ i ≤ n
2.  for r ←2 to n do                              // r为矩阵链长度(子问题规模)
3.     for i ←1 to n−r+1 do                       //子问题的左边界
4.        j ← i +r−1;                             //子问题的右边界
5.        m[i, j] ← 0+m[i+1, j] + p_{i−1}*p_i*p_j;  //分割位置为i时的计算量
6.        s[i, j] ← i;                            //记录分割位置
7.        for k ← i +1 to j−1 do                  //分割位置k向右移动
8.           t ← m[i, k]+m[k+1, j]+ p_{i−1}*p_k*p_j ;
9.           if t < m[i, j] then
10.             m[i, j]← t; s[i, j]← k;           //更新结果
11.          end
12.       end
13.    end
14. end
```

遍历所有规模为r的子问题

遍历所有可能的分割

时间复杂度：
行2,3,7循环都是 $O(n)$，循环内为 $O(1)$，因此$W(n) = O(n^3)$

迭代实现



一个实例

- 输入： P = <30, 35, 15, 5, 10, 20>, n=5；

- 矩阵链：$A_1\,A_2\,A_3\,A_4\,A_5$，其中
- A_1:30×35，A_2:35×15，A_3:15×5，A_4:5×10，A_5:10×20

- 备忘录：存储所有子问题的最小乘法次数（最优值）及得到这个值的划分位置（最优解）.

一个实例

P = <30, 35, 15, 5, 10, 20>

$$\min_{i \le k < j}\{m[i,k] + m[k+1, j] + P_{i-1}P_kP_j\}$$

备忘录m[i,j]

r=1	*m*[1,1]=0	*m*[2,2]=0	*m*[3,3]=0	*m*[4,4]=0	*m*[5,5]=0
r=2	*m*[1,2]=15750	*m*[2,3]=2625	*m*[3,4]=750	*m*[4,5]=1000	
r=3	*m*[1,3]=7875	*m*[2,4]=4375	*m*[3,5]=2500		
r=4	*m*[1,4]=9375	*m*[2,5]=7125			
r=5	*m*[1,5]=11875				

k=2, 子问题: 2..2, 3..5 (35, 15) (15, 20)

k=3, 子问题: 2..3, 4..5 (35, 5) (5, 20)

k=4, 子问题: 2..4, 5..5 (35, 10) (10, 20)

$m[2,5] = \min\{ 0+2500+35\times15\times20,\ 2625+1000+35\times5\times20,\ 4375+0+35\times10\times20\} = 7125$

$s[2,5]=3$

$s[i,j]$，记录求得最优运算次序中最后一次乘法的位置

一个实例

r=2	s[1,2]=1	s[2,3]=2	s[3,4]=3	s[4,5]=4	
r=3	s[1,3]=1	s[2,4]=3	s[3,5]=3		
r=4	s[1,4]=3	s[2,5]=3			
r=5	s[1,5]=3				

解的追踪：$s[1,5]=3 \Rightarrow (A_1A_2A_3)(A_4A_5)$

$s[1,3]=1 \Rightarrow A_1(A_2A_3)$

输出：$((A_1(A_2A_3))(A_4A_5))$ ✓

计算顺序：$(A_1(A_2A_3))(A_4A_5)$ ✗

最少的乘法次数：$m[1,5]=11875$

两种实现的比较

修改：三种不同实现的对比

- 递归动态规划算法
  - ➢采用自顶向下的策略，从最大规模问题开始；
  - ➢子问题被多次重复计算，子问题计算次数呈指数增长；
- 迭代动态规划算法
  - ➢采用自底向上的策略，从最小规模问题开始；
  - ➢每个子问题只计算一次，子问题的计算随问题规模成多项式增长。
- 设计动态规划算法，默认使用迭代实现的方式

动态规划算法的设计步骤

01 **划分子问题：** 用参数表达子问题的边界，将问题求解转变成多步判断的过程

02 **确定优化函数：** 以该函数的极大(或极小)作为判断的依据，确定是否满足优化原则

03 **列出递推方程：** 列出关于优化函数的递推方程(或不等式)和边界条件

04 **设立标记函数：** 考虑是否需要设立标记函数

05 **存储备忘录：** 自底向上计算，以备忘录方法(表格)存储中间结果

06 **解的追踪：** 根据备忘录和优化函数追踪解

含弘光大　继往开来

总结

- 1.动态规划的思路：把求解多阶段决策问题转化为一系列单阶段决策问题（子问题与单阶段）；
- 2.动态规划的适用条件：最优子结构性质；
- 3.动态规划的设计要素：界定子问题的边界，子问题优化函数的递推方程及初值；
- 4.动态规划的递归实现效率不高，原因是同一子问题多次重复计算，可采用空间换时间策略。