

动态规划经典问题

刘汝佳

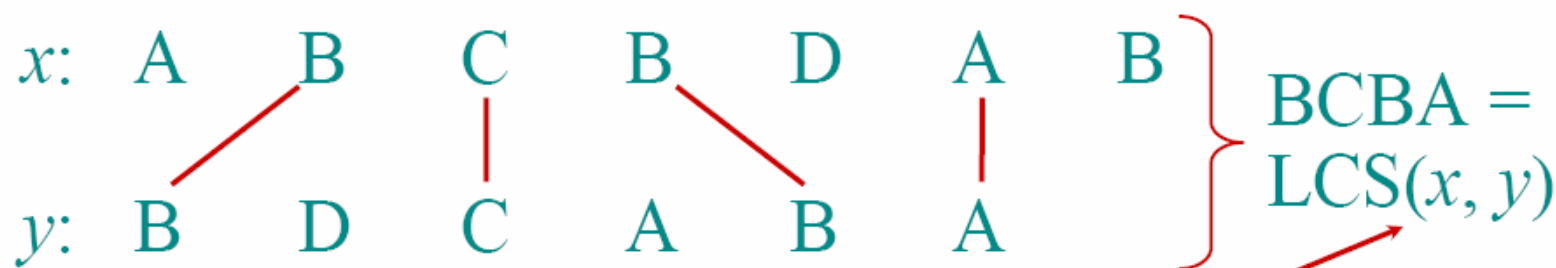
目录

- 一、最长公共子序列 $O(mn)$
- 二、最优排序二叉树 $O(n^3)$
- 三、最长上升子序列 $O(n\log n)$
- 四、最优三角剖分 $O(n^3)$
- 五、最大 m 子段和 $O(mn)$
- 六、0-1背包问题 $O(\min\{nc, 2^n, n1.44^n\})$
- 七、最优排序二叉树 $O(n^2)$
- 八、最优合并问题 $O(n\log n)$

一、最长公共子序列

- **Longest Common Subsequence(LCS)**
 - Given two sequences $x[1 \dots m]$ and $y[1 \dots n]$, find a longest subsequence common to them both.

“a” *not* “the”



functional notation,
but not a function

分析

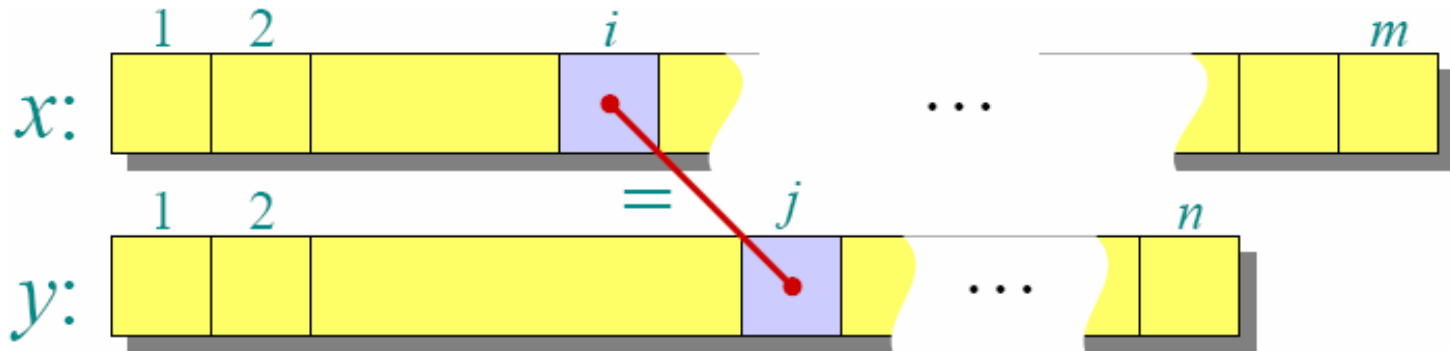
- 考虑前缀 $x[1..i]$ 和 $y[1..j]$, 定义

$$c[i,j] = |\text{LCS}(x[1..i], y[1..j])|$$

- 则 $c[m,n] = |\text{LCS}(x, y)|$. 递推公式为

$$c[i,j] = \begin{cases} c[i-1,j-1] + 1 & \text{if } x[i] = y[j], \\ \max\{c[i-1,j], c[i,j-1]\} & \text{otherwise.} \end{cases}$$

- 很直观. 考虑 $x[i]=y[j]$ 的情形:



关键点一: 最优子结构

- 为了使用动态规划, 问题需具备最优子结构 (**Optimal Substructure**)

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

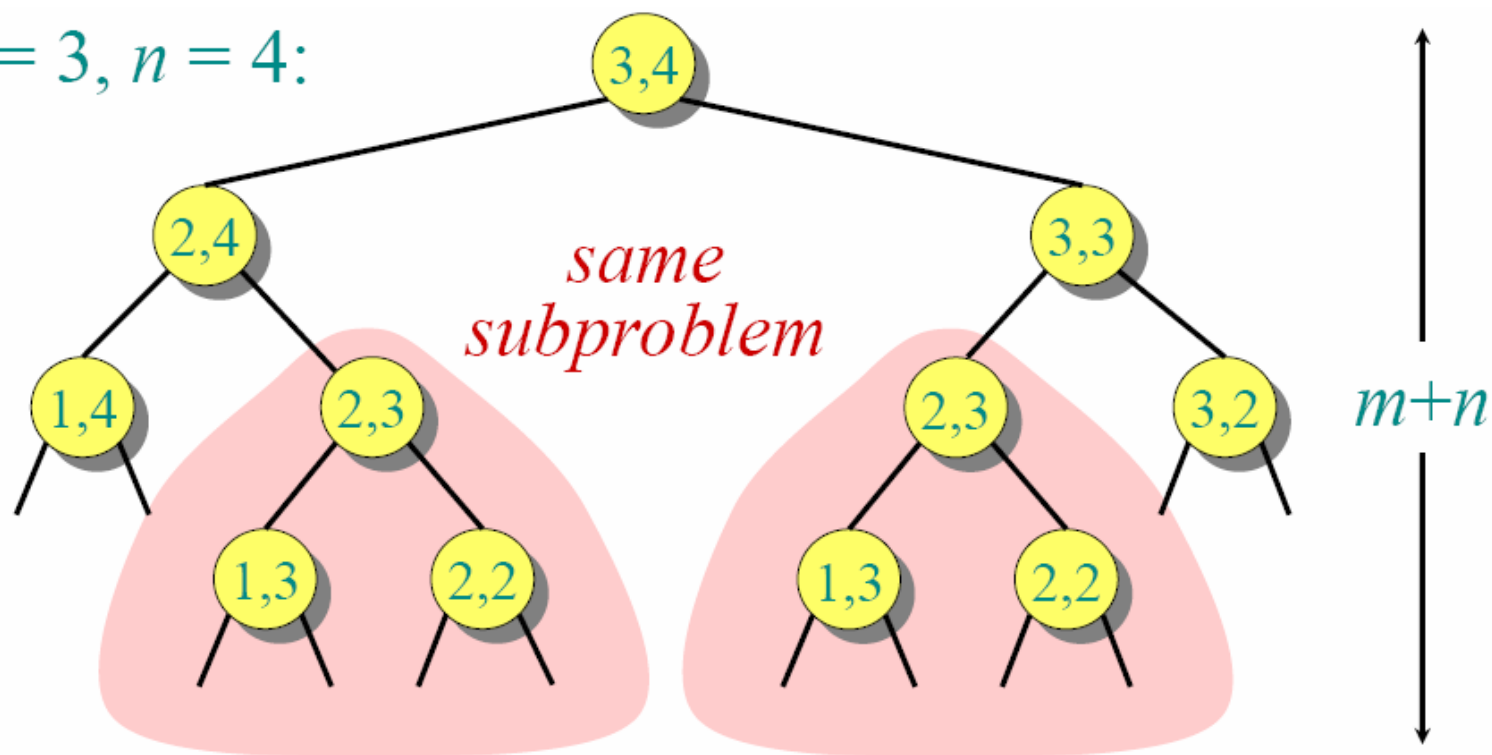
If $z = \text{LCS}(x, y)$, then any prefix of z is an LCS of a prefix of x and a prefix of y .

直接书写的程序

```
LCS( $x, y, i, j$ )  
  if  $x[i] = y[j]$   
    then  $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$   
    else  $c[i, j] \leftarrow \max \{ \text{LCS}(x, y, i-1, j),$   
                                    $\text{LCS}(x, y, i, j-1) \}$ 
```

递归树分析

$m = 3, n = 4$:



Height = $m + n \Rightarrow$ work potentially exponential,
but we're solving subproblems already solved!

关键点二：重叠子问题

- 为了让动态规划确实发挥功效, 问题应该包含尽量多的重叠子问题(**overlapping subproblems**)

Overlapping subproblems

A recursive solution contains a “small” number of distinct subproblems repeated many times.

The number of distinct LCS subproblems for two strings of lengths m and n is only mn .

解决方法: 记忆化

- 注意memoization不是memorization

Memoization: After computing a solution to a subproblem, store it in a table. Subsequent calls check the table to avoid redoing work.

$\text{LCS}(x, y, i, j)$

if $c[i, j] = \text{NIL}$

then if $x[i] = y[j]$

then $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$

else $c[i, j] \leftarrow \max \{ \text{LCS}(x, y, i-1, j), \text{LCS}(x, y, i, j-1) \}$

} *same
as
before*

Time = $\Theta(mn)$ = constant work per table entry.

Space = $\Theta(mn)$.

自底向上递推

IDEA:

Compute the table bottom-up.

Time = $\Theta(mn)$.

Reconstruct LCS by tracing backwards.

Space = $\Theta(mn)$.

Exercise:

$O(\min\{m, n\})$.

	A	B	C	B	D	A	B
B	0	0	0	0	0	0	0
D	0	0	1	1	1	2	2
C	0	0	1	2	2	2	2
A	0	1	1	2	2	3	3
B	0	1	2	2	3	3	4
A	0	1	2	2	3	4	4

空间优化

- 如果只需要最优值, 可以用滚动数组实现
- 按照 i 递增的顺序计算, $d[i,j]$ 只和 $d[i-1,j]$ 和 $d[i,j-1]$ 以及 $d[i-1,j-1]$ 有关系, 因此只需要保留相邻两行, 空间复杂度为 $O(\min\{m,n\})$
- 更进一步的, 可以只保留一行, 每次用单独的变量 x 保留 $d[i-1,j]$, 则递推方程为

if($i==j$) $d[j]=x$;

else { $x = d[j]$; $d[j]=\max\{d[j-1], d[j]\}$ };

变形. 回文词

- 给一个字符串**a**, 保持原字符的顺序不变, 至少要加几个字符才能变成回文词?
- 例: **abfcbfa** → **afbcbcbfa**

分析

- 红、绿色表示原字符, 白色为新增字符
- 显然, **s**和**s'**在任何一个位置不可能都是白色(不需要加那个字符!)
- 应该让红色字符尽量多! 相当于求**s**和逆序串**s'**的**LCS**, 让**LCS**中的对应字符(红色)对齐, 中间的每个绿色字符都增加一个字符和它相等

1		2	3		4	5
5	4		3	2		1

二、最优排序二叉树

- 给 n 个关键码和它们的频率，构造让期望比较次数最小的排序二叉树

分析

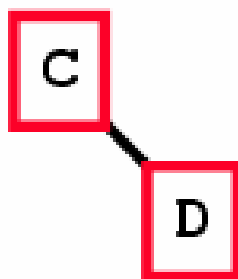
- 定理：最优排序二叉树的子树也是最优排序二叉树
- 给出关键码-频率对照表（升序排列）
- 问题：把哪个关键码做为根？则左右子树可以递归往下做

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	..
23	10	8	12	30	5	14	18	20	2	4	11	7	22	22	10	..

分析

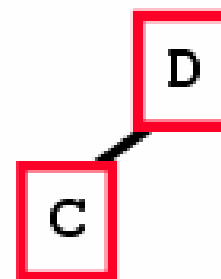
- 用递归来思考，但用递推来做
- 先考虑两个结点的情形

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	..
23	10	8	12	30	5	14	18	20	2	4	11	7	22	22	10	..



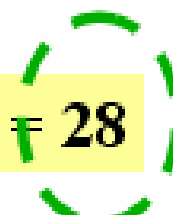
Cost

$$8 \times 1 + 12 \times 2 = 32$$



$$8 \times 2 + 12 \times 1 = 28$$

Min



分析

- 可以用矩阵来保存结果
- $C[j,k]$ 表示从j到k的关键码组成的最优排序二叉树
- $Root[j,k]$ 记录这棵排序二叉树的根

[illegible][illegible]

分析

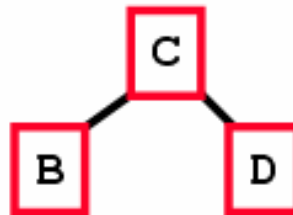
- 考虑三个结点的情形
- 最优值放在 $C[B,D]$ 中，根放在 $root[B,D]$ 中

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	..
23	10	8	12	30	5	14	18	20	2	4	11	7	22	22	10	..

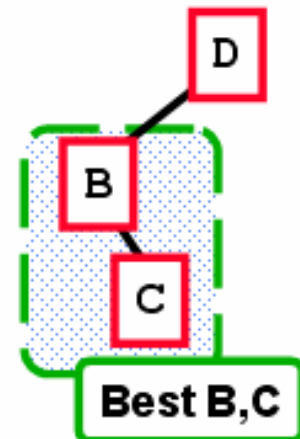
★ Root = B



★ Root = C



★ Root = D



分析

- 类似地，更新所有 $C[j-2,j]$ 和 $root[j-2,j]$

	A	B	C	D	E	F	G	H	I	J	K
A	23										
B	43	10									
C	67	26	8								
D		52	28	12							
E			78	54	30						
F				64	40	5					
G					73	24	14				
H						60	46	18			
I							86	56	20		
J								60	24	2	

Costs

	A	B	C	D	E	F	G	H	I	J	K
A	0										
B	0	1									
C	0	1	2								
D		2	3	3							
E			4	4	4						
F				4	4	5					
G					4	6	6				
H						6	7	7			
I							7	8	8		
J								8	8	9	

Roots

分析

- 四个结点的情形（如A-D）

- **Choose A as root**

Use 0 for left
Best B-D is known

- **Choose B as root**

A-A is in C[0,0]
Best C-D is known

- **Choose C as root**


A-B is in C[0,1]
D is in C[3,3]

- **Choose D as root**

A-C is in C[0,2]
Use 0 in C[4,3] for right

分析

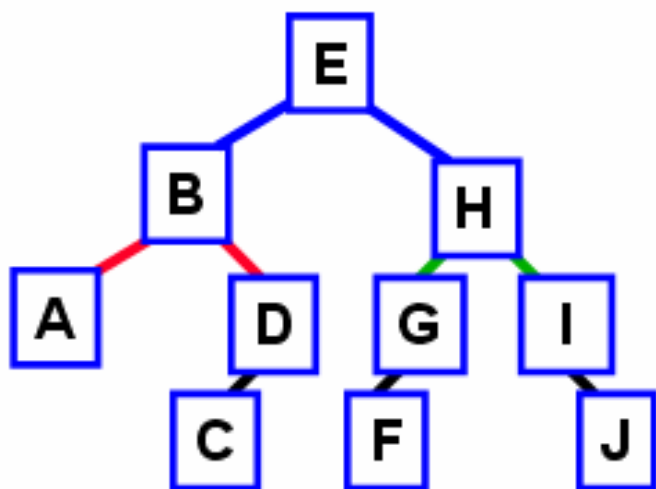
- 最终计算结果为



	A	B	C	D	E	F	G	H	I	J
A	23									
B	43	10								
C	67	26	8							
D	104	52	28	12						
E	180	112	78	54	30					
F	195	122	88	64	40	5				
G	230	155	121	97	73	24	14			
H	284	209	175	151	125	60	46	18		
I	345	270	236	212	180	101	86	56	20	
J	353	278	244	220	186	107	92	60	24	2

分析

- 可以利用root矩阵递归地构造出最优树



	A	B	C	D	E	F	G	H	I	J
A	0									
B	0	1								
C	0	1	2							
D	1	2	3	3						
E	2	4	4	4	4					
F	2	4	4	4	4	5				
G	4	4	4	4	4	6	6			
H	4	4	4	4	6	6	7	7		
I	4	4	4	4	7	7	7	8	8	
J	4	4	4	4	7	7	8	8	9	

分析

- 时间复杂度：计算每个 $C[i,j]$ 和 $root[i,j]$ 需要枚举根结点，故为 $O(n^3)$
- 空间复杂度：需要两个 $n*n$ 矩阵， $O(n^2)$

三、最长上升子序列

- 最长上升子序列问题（**LIS**）给一个序列，求它的一个递增子序列，使它的元素个数尽量多。例如序列**1,6,2,5,4,7**的最长上升子序列是**1,2,5,7**（还有其他的，这里略去）

分析

- 定义 $d[i]$ 是从第1个元素到第 i 个元素为止的最长子序列长度, 则状态转移方程为

$$d[i] = \min_{k < i \text{ 且 } a[k] < a[i]} \{d[k] + 1\}$$

- 直接使用这个方程得到的是 $O(n^2)$ 算法
- 下面把它优化到 $O(n \log n)$

状态的组织

- d值相同的a值只需要保留最小的, 因此用数组g[i]表示d值为i的数的a最小值, 显然

$$g[1] \leq g[2] \leq \dots \leq g[k]$$

- **计算d[i]:** 需要在g中找到大于等于a[i]的第一个数j, 则d[i]=j
- **更新g:** 由于g[j]>a[i], 需要更新g[j]=a[i]

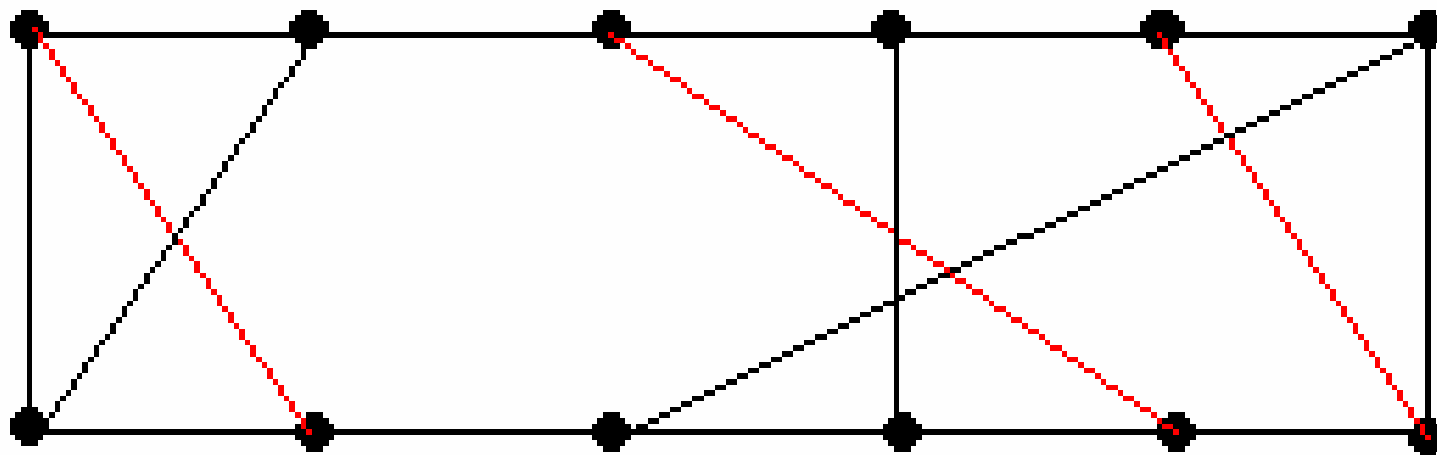
代码

- 使用STL的lower_bound可以直接求出比a[i]大的第一个数, 用二分查找实现, 每次转移时间 $O(\log n)$, 总时间 $O(n \log n)$

```
fill(g, g + n, infinity);  
for(int i = 0; i < n; i++){  
    int j = lower_bound(g, g + n, a[i]) - g;  
    d[i] = j + 1;  
    g[j] = a[i];  
}
```

变形1: 航线问题

- 有两行点, 每行 n 个. 第一行点和第二行点是一一对应的, 有线连接, 如下图所示
- 选择尽量多的线, 两两不交叉



分析

- 设与第1行第 i 个点对应的是第2行第 $f[i]$ 个点
- 假设 $i < j$, 两条线 $(i, f[i])$ 和 $(j, f[j])$ 的充要条件是 $f[i] < f[j]$, 因此问题变成了

求 f 的最长上升子序列

- 时间复杂度为 $O(n \log n)$

变形2: 两排列的LCS

- 给 $1 \sim n$ 的两个排列 p_1, p_2
- 求 p_1 和 p_2 的最长公共子序列
- 例: **1 5 3 2 4** \Leftrightarrow **5 3 4 2 1**

分析

- 算法一: 直接套用LCS算法, 时间 $O(n^2)$
- 算法二: 注意到把两个排列做相同的置换, LCS不变, 可以先把 p_1 排列为 $1, 2, 3, \dots, n$

1 5 3 2 4 \Leftrightarrow **1 2 3 4 5**

- 即映射 $5 \rightarrow 2, 2 \rightarrow 4, 4 \rightarrow 5$, p_2 作同样置换

5 3 4 2 1 \Leftrightarrow **2 3 5 4 1**

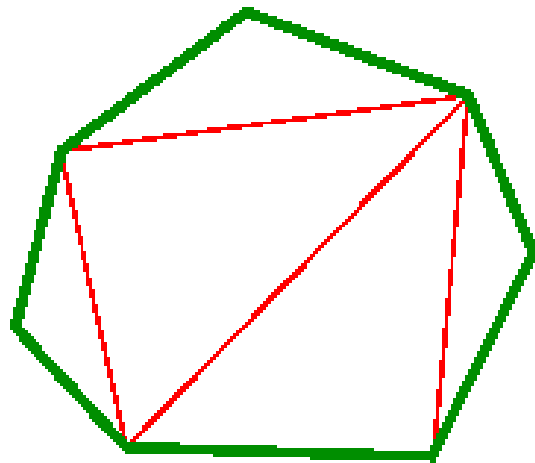
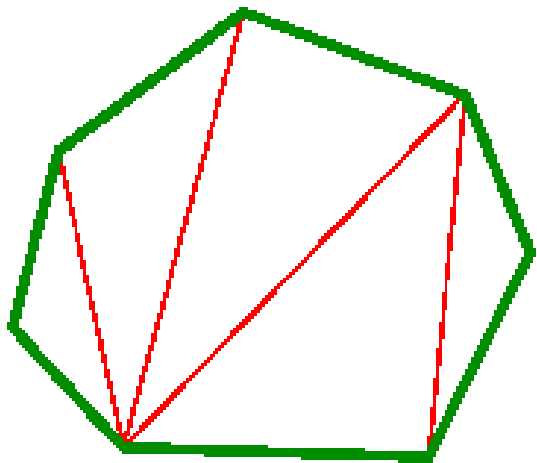
- 与 $1, 2, 3, \dots, n$ 的LCS显然是最长上升子序列, 时间降为 $O(n \log n)$

推广: DAG上的最短路

- “上升”依赖于序关系 \leq , 它具有一般性
- **DAG**(有向无环图)的最长路径问题: 把有向边看成偏序关系, 则本题的算法一仍然适用, 时间复杂度为 $O(n^2)$. 如果图的边数 m 比较, 可进一步优化到 $O(m)$, 因为每条边恰好考虑一次(用邻接表或前向星, 而不是邻接矩阵)
- 算法二不再适用! 因为 d 值相同的 a 不一定可以两两相互比较, 不一定存在最小值.

四、最优三角剖分

- 给一个 n 个顶点的凸多边形, 有很多方法对它进行三角剖分(polygon triangulation)
- 每个三角形有一个权计算公式(如周长, 顶点权和), 求总权最小(大)的三角剖分方案

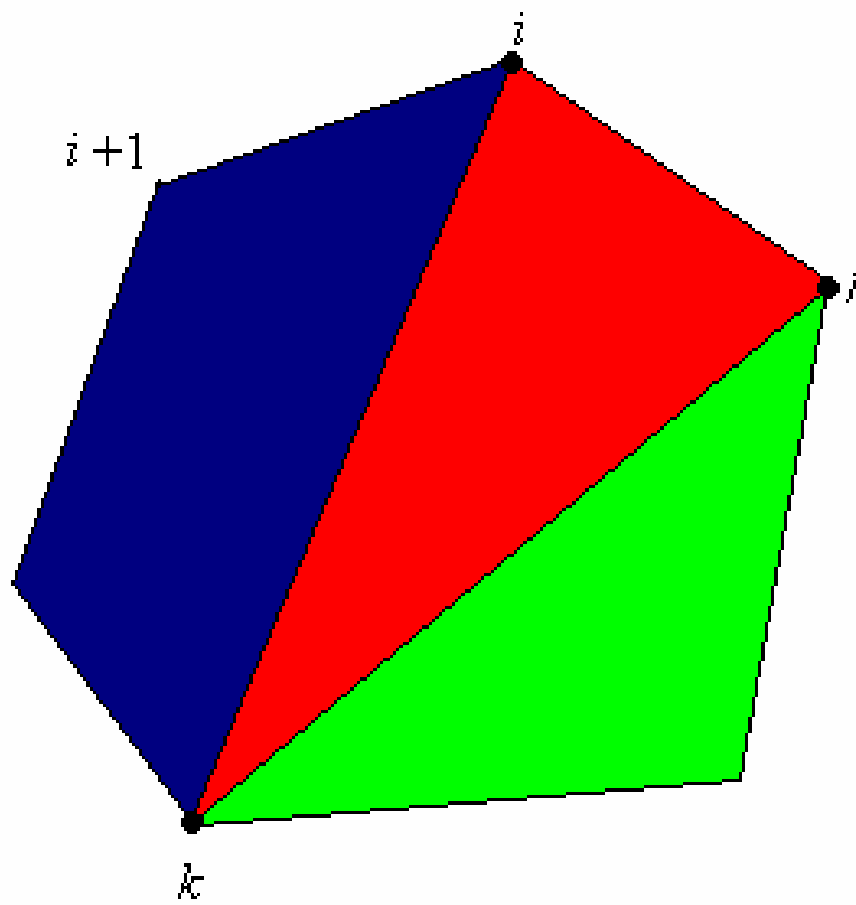


分析

- 用 $d[i,j]$ 表示由顶点 $i, i+1, \dots, j$ 组成的多边形 (注意 i 可以大于 j) 的最小代价
 - 方案一: 枚举三个顶点, 组成一个三角形, 决策是 $O(n^3)$ 的
 - 方案二: 边 (i,j) 一定属于一个唯一的三角形, 设第三个顶点为 k , 则决策仅为 $O(n)$
- 采用方案二, 状态 $O(n^2)$, 决策 $O(n)$, 总 $O(n^3)$

分析

- 确定 k 后, 最优方案应该是 $d[i,k]+d[k,j]+w(i,k,j)$
- 因此转移方程 $d[i,j]=\max\{d[i,k]+d[k,j]+w(i,k,j)\}$

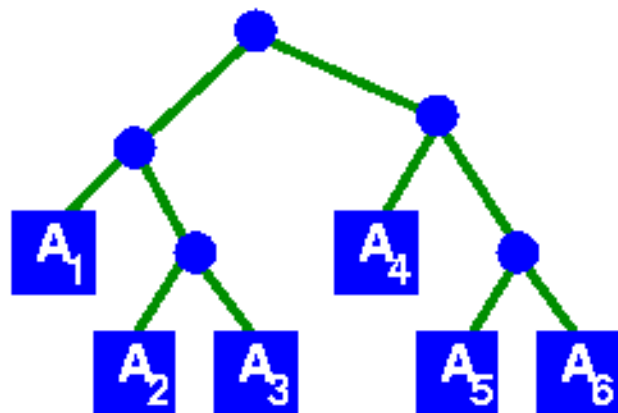


变形1: 最优矩阵乘法链

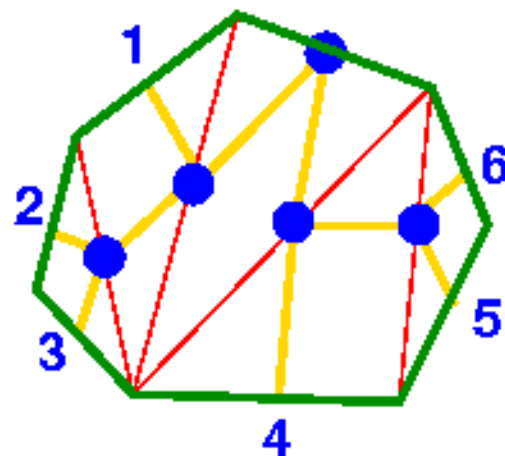
- 需要计算n个矩阵的乘积 $A_1A_2\dots A_n$
- 由于矩阵乘法满足结合律, 可以有多种计算方法. 例如 A_1 是 $10*100$, A_2 是 $100*5$, A_3 是 $5*50$, 则
 - 顺序1: $(A_1A_2)A_3$, 代价为 $10*100*5+10*5*50=7500$
 - 顺序2: $A_1(A_2A_3)$, 代价为 $100*5*10+10*100*50=75000$
- 求代价最小的方案(加括号方法)

共同的结构

- 用二叉树(**binary tree**)可以表示两个问题相同的结构, 每个结点表示一个区间(结点区间 / 矩阵区间), 左子树和右子树表示分成的两个序列
- 如果在原问题中让 $d[i,j]$ 表示 $i-1 \sim j$ 的最优值, 则在方程形式上也完全等价



$(A_1(A_2A_3))(A_4(A_5A_6))$

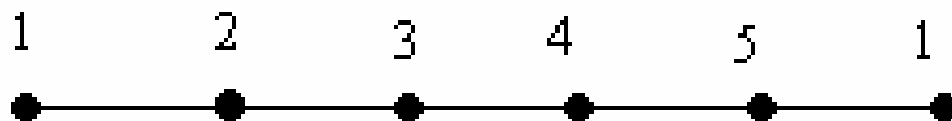


变形2. 决斗

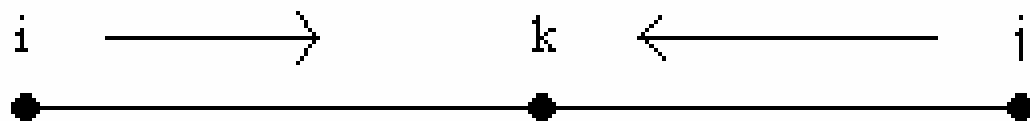
- 编号为 $1\sim n$ 的 n 个人按逆时针方向排成一圈，他们要决斗 $n-1$ 场。每场比赛在某相邻两人间进行，败者退出圈子，紧靠败者右边的人成为与胜者直接相邻的人。
- 任意两人之间决斗的胜负都将在一矩阵中给出（如果 $A[i,j]=1$ 则 i 与 j 决斗 i 总是赢，如果 $A[i,j]=0$ 则 i 与 j 决斗时 i 总是输），
- 求出所有可能赢得整场决斗的人的序号

分析

- 首先把圈想象成一条链



- 设 $d[i,j]$ 表示 i 是否能和 j 相遇, 则相遇的充要条件是存在 k , i 和 k , k 和 j 都能相遇, 且 i 或 j 能打败 k



- 同样是 $O(n^2)$ 个状态, 决策 $O(n)$, 总 $O(n^3)$

五、最大m子段和

- 给一个序列 a_1, a_2, \dots, a_n
- 求m个不相交(可以相接)的连续序列, 总和尽量大
- 例如 $m=2$, **1 2 -3 4 5 -6 7**

分析

- 设 $d[i,j]$ 为以 j 项结尾的 i 段和的最大值, 则需要枚举此段开头 y 和上一段结尾 x , 即

$$d[i,j]=\max\{d[i-1,x] + a[y..j]\}$$

- 每次需要枚举 $x < y \leq j$, 决策量为 $O(n^2)$, 状态为 $O(nm)$, 共 $O(n^3m)$
- 注意到如果 $a[j-1]$ 也是本段的, 答案变成成为 $d[i,j-1]+a[j]$, 因此方程优化为

$$d[i,j]=\max\{d[i,j-1]+a[j], d[i-1,x]+a[j]\}, x < j$$

分析

- 优化后状态仍然是二维的，但决策减少为 $O(n)$ ，总 $O(n^2m)$
- 可以继续优化. 注意到时间主要耗费在对 x 的枚举上, 计算 $\max\{d[i-1, x]\}$. 这个值...
- 我们把 d 的第一维称为“阶段”，则本题是典型的多阶段决策问题
 - 计算一个阶段时, 顺便记录本阶段最大值
 - 只保留相邻两个阶段(滚动数组)
- 则时间降为 $O(nm)$, 空间降为 $O(n)$

六、0-1背包问题

- 给定 n 种物品和一个背包, 物品 i 的重量是 w_i , 价值是 v_i , 背包容量为 c
- 对于每个物品, 要么装背包, 要么不装
- 选择装背包的物品集合, 使得物品总重量不超过背包容量 c , 且价值和尽量大

分析

- 设 $d[i,j]$ 为背包容量为 j 时, 只考虑前 i 个物品时的最大价值和
 - 如果装第 i 个物品, 背包容量只剩 $j-w_i$
 - 如果不装, 背包容量不变
- 因此 $d[i,j]=\max\{d[i,j-w_i]+v_i, d[i-1,j]\}$
- 状态有 nc 个, 每个状态决策只有两个, 因此总时间复杂度为 $O(nc)$. 用滚动数组后, 空间复杂度只有 $O(c)$

分析

- 当 c 大时, 算法效率非常低. 事实上, 由于 c 是数值范围参数, 一般不把它看作输入规模. 这样的 $O(nc)$ 只是一个伪多项式算法
- 事实上, 如果物品重量和背包容量都是实数时, 算法将失败, 因为看起来物品的重量和可以是“任何实数”.
- 但事实是: 物品重量和只有 2^n 种可能的取值, 并不是无限多种

分析

- 算法一：枚举 2^n 个子集合，再计算，枚举 2^n ，计算 n ，共 $n2^n$
- 算法二：采用递归枚举，共 2^n
- 算法三：先考虑一半元素，保存 $2^{n/2}$ 个和。再考虑后一半元素，每计算出一个和 w ，查找重量 $\leq C-w$ 的元素中价值的最大值。

下面考虑实现细节

算法三

- 前一半元素的 $2^{n/2}$ 个和按重量从小到大排序后放在表**a**里. 对于任何两个和 i, j , 如果 $w_i < w_j$ 且 $v_i > v_j$, 则 j 是不需要保存的, 因此按重量排序好以后也是按价值排序的
- 考虑后一半元素时, 每得到一个重量 w , 用二分查找得到重量不超过 $c-w$ 的最大元素, 则它的价值也最大.
- 预处理时间复杂度 $2^{n/2} \log 2^{n/2}$, 每个重量 w 二分查找时间为 $\log 2^{n/2}$, 因此总 $2^{n/2} \log 2^{n/2} = O(n 1.44^n)$

七、再谈最优排序二叉树

- 给 n 个关键码和它们的频率，构造让期望比较次数最小的排序二叉树

基本分析

- 设结点*i..j*的最优代价为*d[i,j]*, 则

$$d[i, j] = \min_{i \leq k \leq j} \{d[i, k-1] + d[k+1, j]\} + w[i, j]$$

- 其中*w[i,j]=f_i+f_{i+1}+...+f_j*, (如果认为根结点的代价为0, 则*w[i,j]*要减去*f_k*). 直接计算是*O(n³)*的
- 设*d[i,j]*的最优决策为*K[i,j]*, 下面证明
$$K[i, j-1] \leq K[i, j] \leq K[i+1, j]$$
- 从而把时间复杂度降到*O(n²)*

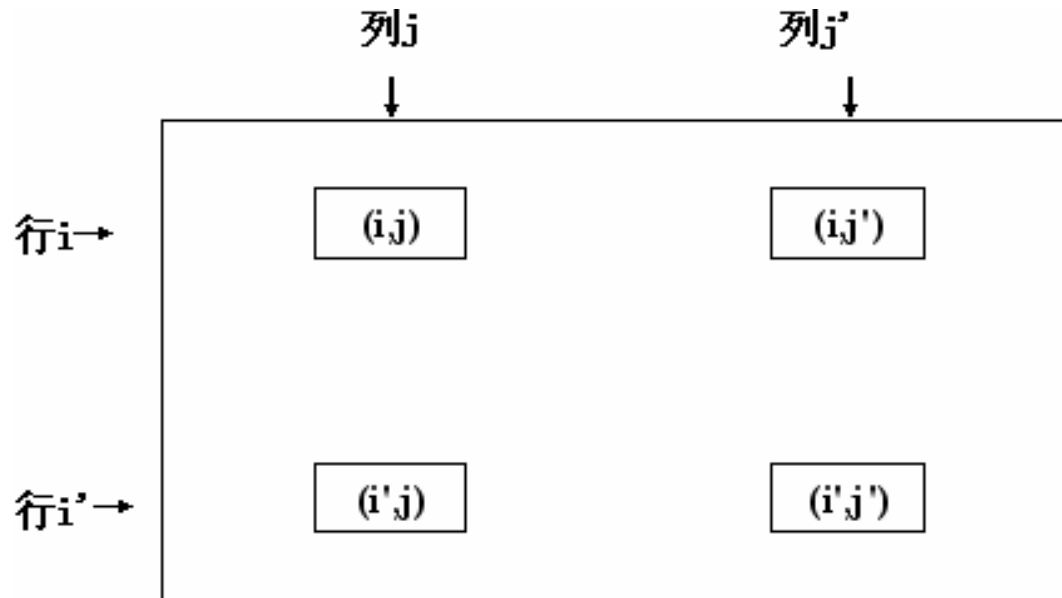
四边形不等式

- 凸性(Monge condition/quadrangle inequality)

$$w[i,j]+w[i',j'] \leq w[i',j]+w[i,j'], i \leq i' < j \leq j'$$

- 单调性(区间包含格上)

$$w(i',j) \leq w(i,j'), i \leq i' < j \leq j'$$



验证四边形不等式

- 只需验证

$$w[i,j]+w[i+1,j+1]\leq w[i+1,j]+w[i,j+1]$$

- 移项得

$$w[i+1,j+1]-w[i+1,j]\leq w[i,j+1]-w[i,j]$$

- 当j固定时记函数 $f(x) = w[x,j+1]-w[x,j]$, 则上式变为: $f(i+1)\leq f(i)$, 因此

$f(i)$ 是减函数, w 为凸; $f(i)$ 是增函数, w 为凹

- 固定i有同样的结论(减函数时为凸)

本题中的w

- 本题中, w的凸性更好证明:

$$\begin{aligned} & w[i,j] + w[i+1,j+1] \\ &= \textcolor{red}{w[i,j]} + (\textcolor{teal}{w[i,j]} + \textcolor{teal}{f[j+1]} - \textcolor{red}{f[i]}) \\ &= w[i+1,j] + w[i,j+1] \end{aligned}$$

- 两边是完全相等的. 或者计算

$$f(x) = w[x,j+1] - w[x,j] = f[i+1] = \text{常数}$$

- 常数既是增函数也是减函数, 因此
本题中, w既为凸也为凹

定理

- 考虑递归式 $d[i,j] = \min\{d[i,k-1] + d[k,j] + w[i,j]\}$
- 定理(**F.Yao**): 若 w 满足四边形不等式, 则 d 也满足四边形不等式, 即

$$d[i,j] + d[i',j'] \leq d[i',j] + d[i,j'], \quad i \leq i' \leq j \leq j'$$

- 证明: 对长度 $i=j'-i$ 归纳, 显然 $i \leq 1$ 时正确. $i=i'$ 或 $j=j'$ 时(同一行或同一列), 等式显然成立
 - 情形1: $i'=j$, 退化为反三角不等式
 - 情形2: $i' < j$

情形1. 反三角不等式

- $i'=j$ 时, $d[i,j]+d[i',j'] \leq d[i',j]+d[i,j']$ 退化为
$$d[i,j]+d[j,j'] \leq d[i,j']$$
- 设 k 为让 $d[i,j']$ 取最小值的决策(有多个时取最大的一个 k , 后同).
- 若 $k \leq j$, 则 k 是计算 $d[i,j]$ 考虑过的合法决策
$$d[i,j] \leq w[i,j]+d[i,k-1]+d[k,j]$$
- 两边加上 $d[j,j']$, 得
$$d[i,j]+d[j,j'] \leq w[i,j]+d[i,k-1]+d[k,j]+d[j,j']$$

情形1. 反三角形不等式

- 设k为让 $d[i,j']$ 取最小值的决策. $k \leq j$ 时有
$$d[i,j] + d[j,j'] \leq w[i,j] + d[i,k-1] + d[k,j] + d[j,j']$$
- 用单调性和反三角形不等式(归纳假设), 有
$$d[i,j] + d[j,j'] \leq w[i,j'] + d[i,k-1] + d[k,j']$$
- 由于k是最佳决策, 右边恰好是 $d[i,j']$, 这就证明了情形1中 $k \leq j$ 的情形, $k > j$ 时类似

情形2. 非退化的情形

- $i' < j$ 时, 右边保留两项 $d[i', j]$ 和 $d[i, j']$. 设二者取最小值时的决策分别为 y 和 z , 仍需分 $z \leq y$ 和 $z > y$ 两种情况(对称). 下面只考虑 $z \leq y$ 时
- y 和 z 是合法决策, 因此 $y \leq j$, $z > i$, 且
$$d[i, j] \leq w[i, j] + d[i, z-1] + d[z, j]$$
$$d[i', j'] \leq w[i', j'] + d[i', y-1] + d[y, j']$$
- 两式相加并整理, 对应项写在一起, 得

情形2. 非退化的情形

- 两式相加并整理, 对应项写在一起, 右边 \leq
 $w[i,j]+w[i',j']+\text{d}[i,z-1]+\text{d}[i',y-1]+\text{d}[z,j]+\text{d}[y,j']$
- 因 $z\leq y$, 红蓝色分别用四边形不等式, 右边 \leq
 $w[i,j']+\text{d}[i,z-1]+\text{d}[z,j']+\text{d}[i',y-1]+\text{d}[y,j]$
- 按红蓝色分别组合, 得
$$\text{d}[i,j]+\text{d}[i',j']\leq \text{d}[i,j']+\text{d}[i',j]$$
- $z\leq y$ 时命题得证. $z>y$ 时类似

决策单调性

- 进一步地, d 的凸性可以推出决策的单调性
- 设 $k[i,j]$ 为让 $d[i,j]$ 取最小值的决策, 下面证明
$$k[i,j] \leq k[i,j+1] \leq k[i+1,j+1], i \leq j$$
- 即: k 在同行同列上都是递增的
- 证明: $i=j$ 时显然成立. 由对称性, 只需证明 $k[i,j] \leq k[i,j+1]$. 记 $d_k[i,j] = d[i,k-1] + d[k,j] + w[i,j]$, 则只需要证明对于所有的 $i < k \leq k' \leq j$, 有

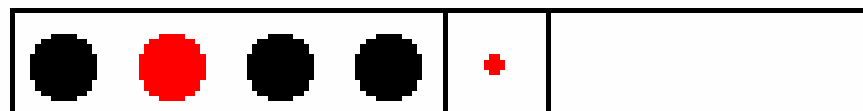
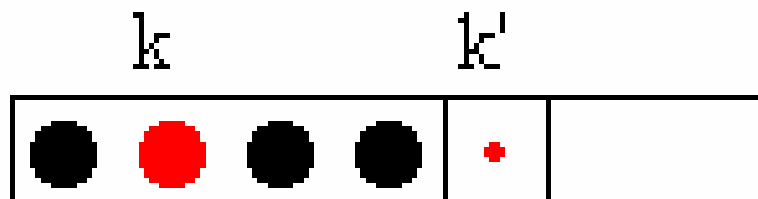
$$d_{k'}[i,j] \leq d_k[i,j] \rightarrow d_{k'}[i,j+1] \leq d_k[i,j+1]$$

决策单调性

- 事实上，我们可以证明一个更强的式子

$$d_k[i,j]-d_{k'}[i,j] \leq d_k[i,j+1]-d_{k'}[i,j+1] \quad (i \leq k \leq k' \leq j)$$

- k' 在 $[i,j]$ 更优(左 ≥ 0) $\rightarrow k'$ 在 $[i,j+1]$ 上也更优(右 ≥ 0)
- 设 k' 是 $[i,j]$ 的最优值，则对于它左边的任意 k , k' 在 $[i,j]$ 更优可推出 k' 在 $[i,j+1]$ 上也更优, 因此在区间 $[i,j+1]$ 上, k' 左边的值都比它大, 如下图



决策单调性

- 欲证 $d_k[i,j]-d_{k'}[i,j]\leq d_k[i,j+1]-d_{k'}[i,j+1]$, 移项得
$$d_k[i,j]+d_{k'}[i,j+1]\leq d_k[i,j+1]+d_{k'}[i,j]$$
- 按定义展开, 两边消去 $w[i,j]+w[i,j+1]$, 得
$$d[i,k-1]+d[k,j]+d[i,k'-1]+d[k',j+1]\leq d[i,k-1]+d[k,j+1]+d[i,k'-1]+d[k',j]$$
- 同时消去红色部分, 得
$$d[k,j]+d[k',j+1]\leq d[k,j+1]+d[k',j]$$
- 这就是 $k\leq k'\leq j<j+1$ 时 d 的凸性

算法优化

- 由于 k 是单调, 计算 $d[i,j]$ 时决策只需从 $k[i,j-1]$ 枚举到 $k[i+1,j]$ 即可
- 当 $L=j-i$ 固定时,
 - $d[1,L+1]$ 的决策是 $k[1,L]$ 到 $k[2,L+1]$
 - $d[2,L+2]$ 的决策是 $k[2,L+1]$ 到 $k[3,L+2]$
 - $d[3,L+3]$ 的决策是 $k[3,L+2]$ 到 $k[4,L+3]$
 - ...
 - 总决策是从 $k[1,L]$ 到 $k[n-L+1,n]$, 共 $O(n)$
- 对于 $O(n)$ 个 L , 共 $O(n^2)$ 个决策.
- 本题方程略有不同, 但也可用类似方法证明

八、最优合并问题

- 有 n 个正整数, 每次可以合并两个相邻的数(相加), 代价为相加后的新数.
- 按如何的顺序把所有的数合并成一个, 使得代价总和尽量小?

分析

- 假设数 $i \sim j$ 的最小合并代价为 $d[i,j]$, 考虑最后一次合并, 有

$$d[i,j] = \min\{d[i,k-1]+d[k,j]+w[i,j]\}$$

- 其中 $w[i,j] = a[i]+\dots+a[j]$
- 显然 $w[i,j]$ 是凸的和增的, 因此用四边形不等式优化后时间复杂度降为 $O(n^2)$
- 下面进一步优化到 $O(n \log n)$

错误的贪心法

- 贪心法: 每次采取代价最少的合并方案
- 不一定得到最优解! 最优解为74

5	3	4	1	3	2	3	4
5	3	4	4		2	3	4
5	3	4	4		5		4
5	7		4		5		4
5	7		9				4
12			9				4
12			13				
25							

$$4+5+7+9+12+13+25=75$$

(a)

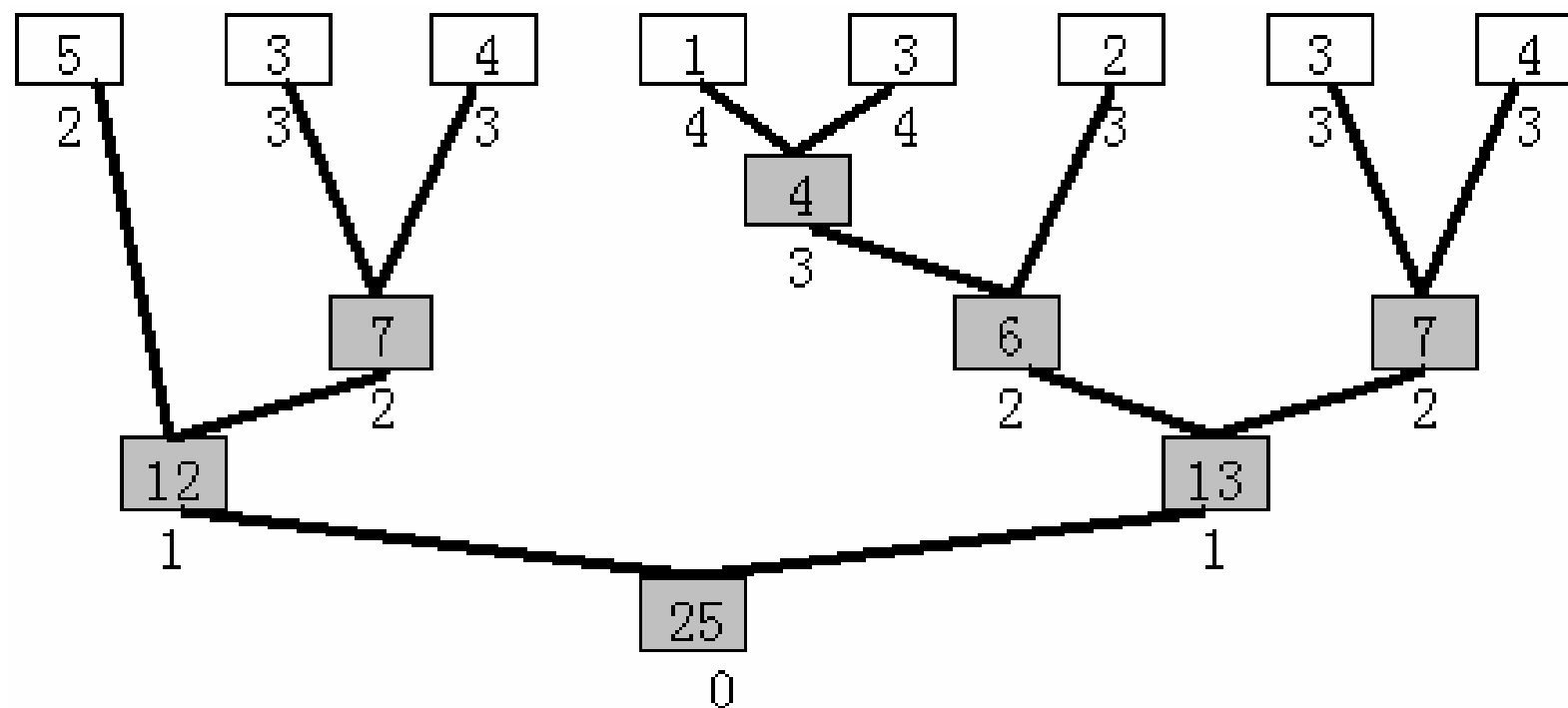
5	3	4	1	3	2	3	4
5	3	4	4		2	3	4
5	3	4	4		5		4
5	7		4		5		4
5	7		4		9		
5	11				9		
16					9		
25							

$$4+5+7+9+11+16+25=77$$

(b)

分析

- 可以把合并过程画成一棵树, 标记结点深度

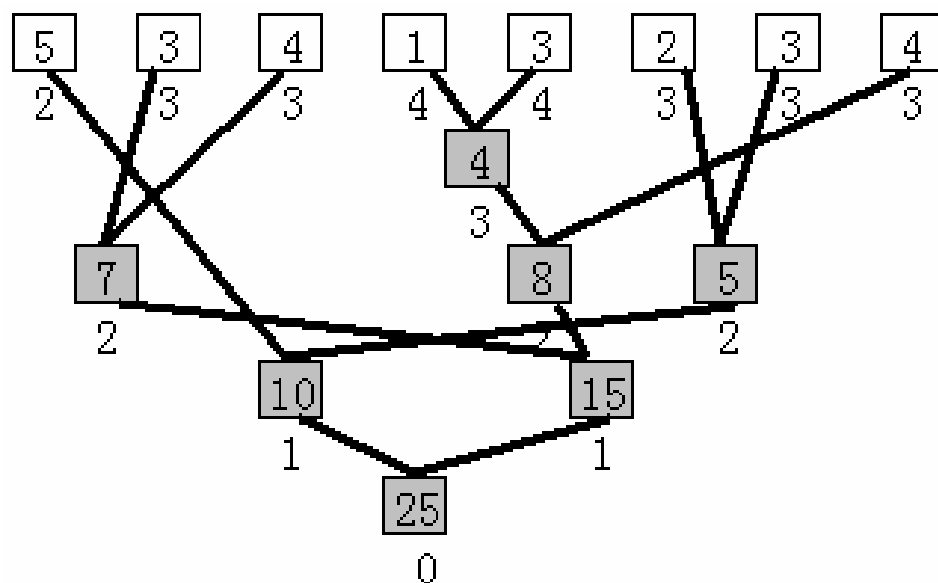


计算方法一: $4+7+6+7+12+13+25=74$

计算方法二: $5*2+3*3+4*3+1*4+3*4+2*3+3*3+4*3=74$

相容结点贪心法

- 相容结点对: 中间没有原始结点的结点对
- 修改的贪心法:** 每次合并和最小的相容结点*i*, *j*. 如果有多个, 因此让*i*和*j*尽量小



计算方法一: $4+7+8+5+10+15+25=74$

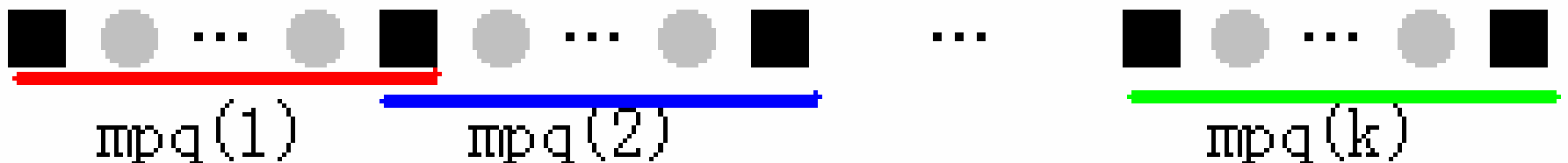
计算方法二: $5*2+3*3+4*3+1*4+3*4+2*3+3*3+4*3=74$

重组

- 修改的贪心法没有按照题目要求合并, 得到的合并树不一定是合法的答案, 但它同样能得到了一棵树, 代价和仍然是 $\sum\{d_i * a_i\}$, 其中 d_i 是叶子 i 的深度
- **定理:** 用相容结点贪心法得到的树, 在保持各叶子的深度 d_i 不变的情况下可以重组成一棵满足题目要求的合并树
- **算法:** 用修改贪心法求出深度序列, 再重组

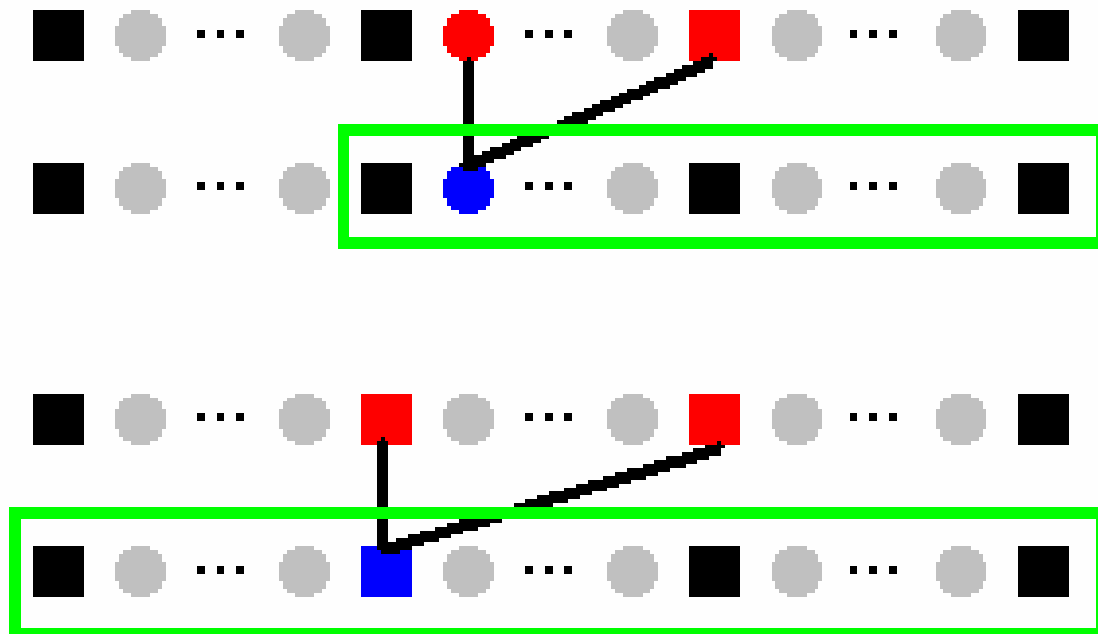
实现上的考虑

- 合并结点用圆形表示, 原始结点用正方形表示
- 第 i 个圆形序列片段连同它左边、右边的正方形组成一个结点集合 $mpq(i)$
- 每次合并的两个结点一定是某一个 mpq 的最小值 $\min1(i)$ 和次小值 $\min2(i)$
- **贪心过程:** 每次选择使 $\min1(i)+\min2(i)$ 最小的 i , 合并结点 $\min1(i)$ 和 $\min2(i)$



分析

- 合并操作
 - 两个圆形: 没有影响
 - 一个圆形和一个正方形: 合并两个mpq
 - 两个正方形: 三个mpq合并



算法梗概

- 贪心过程: $O(n \log n)$, 使用可并优先队列
- 标记深度: $O(n)$
- 树重组: $O(n)$