算法策略部分

教材:

[王] 王晓东,计算机算法设计与分析(第4版),电子工业.

参考资料:

[C] 潘金贵等译, Cormen等著, 算法导论, 机械工业.

[M] 黄林鹏等译, Manber著, 算法引论-一种创造性方法, 电子.

[刘] 刘汝佳等, 算法艺术与信息学竞赛, 清华大学.

第10章 动态规划
dynamic programming

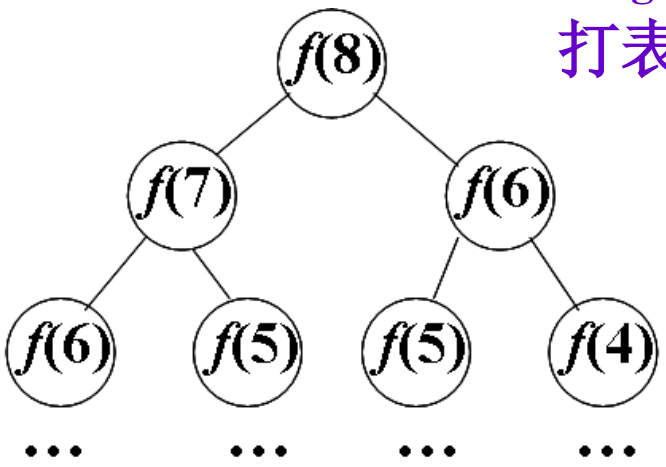
1. 动态规划一般原理(与分治对比), Bellman, OSP
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)

动态规划与分治

- 分治的过程:
  分解—递归解子问题—合并
- 若有大量重复子问题, 则不宜分治
- 举例: Fibonacci数的计算

$f(n) = f(n\text{-}1) + f(n\text{-}2), f(1)=1, f(0)=0.$

输入n, 输出$f(n)$.



Programming的含义:
打表记录中间结果

递归:

```
int f(int n)
{ if(n<2) return(1);
   return(f(n-1)+f(n-2);}
```

$2^{O(n)}$时间, O(n)空间

动态规划:

```
f[1]=1;f[0]=0;
for(i=2,i<n,i++)
     f[i]=f[i-1]+f[i-2];
```

O(n)时间, O(n)空间

```
f=1; b=0;i=1;
while(i++<n){
  temp=f;f=f+b;b=temp;}
```

O(n)时间, O(1)空间

Fibonacci数矩阵算法与动态规划

由 $f(n) = f(n\text{-}1) + f(n\text{-}2)$, 知

$$\begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f(1) \\ f(0) \end{bmatrix}$$

求幂问题: 输入: x, n; 输出: x^n.
设二进制表示$n=(b_{k-1}\ldots b_1 b_0)_2$, 则

1. 对$i = 0$到$k-2$, 计算$x^{2^{i+1}} = x^{2^i} \times x^{2^i}$

2. 计算$x^n = x^{\sum_{i=0}^{k-1} b_i \times 2^i} = \prod_{i=0}^{k-1} \left(x^{2^i}\right)^{b_i}$

- 乘法次数O(log$n$)
- 记录个数O(log$n$)
- 输入规模O(log$n$)

Fibonacci数$f(n)$ mod M矩阵算法

- 问题: 输入n, 输出 $f(n)$ mod M.
- 输入规模: Θ(log n). 算法:

  0. A=(b_k,…,b_1,b_0), B[0]=((1,1),(1,0)), ans = ((1,0),(0,1))
  1. 对 i = 0..k-1,
  2. 　计算B[i+1] = B[i]*B[i] mod M
  3. 对 i = 0..k,
  4. 　若A[i]==1, 则 ans = ans * B[i] mod M
  5. 输出ans

- 时间O(log n), 空间O(log n), 打表记录中间结果

DP适用条件

- 最优子结构性质（OSP optimal substructure property）

  OSP: 最优策略的子策略也是最优.

- 重叠子结构性质

  Programming是指使用表格化的算法.

- Bellman，1955，奠定DP数学基础.

•“An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.”

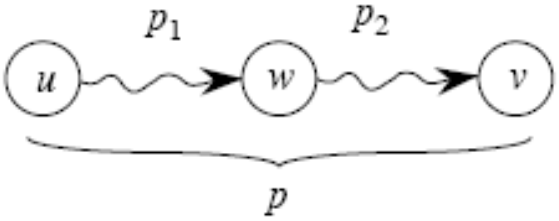
•It can be summarized simply as follows: “every optimal policy consists only of optimal sub policies.”

动态规划适用条件

•无权最短简单路径: 边数最少. 具有最优子结构性质.

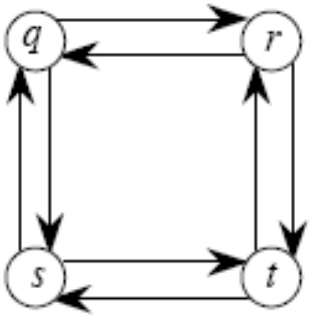
•无权最长简单路径问题.

如果把最长简单路径地分为几个子路径，
那么 p_1 是 $u \rightarrow w$的最长简单路径吗？
p_2 是$w \rightarrow v$的最长简单路径吗？
不是！



考察$q \rightarrow t$的最长简单路径： *q-r-t*

- 子路径： *q-r*,
- *q*到*r*的最长简单路径为 *q-s-t-r*
- 子路径： *r-t*,
- *r*到*t*的最长简单路径为 *r-q-s-t*



动态规划设计步骤

设计步骤 1) 描述最优解的结构

[王] 2) 递归定义最优解

3) 自底向上计算最优值

4) 由计算结果构造最优解

第三章 动态规划
dynamic programming

1. 动态规划一般原理(与分治对比), Bellman, OSP
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)

矩阵连乘问题([王])

- 输入: 给定矩阵$A_1, A_2, \ldots, A_n$, A_i与A_{i+1}可乘
- 输出: 计算量最小的乘法次序
- 输入样例: A_1(10×100阶), A_2(100×5阶), A_3(5×50阶),
  - 两种计算次序: ((A_1×A_2) × A_3), (A_1× (A_2×A_3))
  - A_1×A_2的计算量: 10×100×5 (乘法次数)
  - ((A_1×A_2) ×A_3) : 10×100×5 + 10×5×50 = 7500
  - (A_1× (A_2×A_3)) : 100×5×50 + 10×100×50 = 75000
  - 样例输出: ((A_1×A_2) ×A_3)

设维数序列为(30, 35, 15, 5, 10, 20, 25)

A_1:30×35
A_2:35×15
A_3:15×5
A_4:5×10
A_5:10×20
A_6:20×25

矩阵连乘问题([王])

- 取整数序列$q_0,q_1,\ldots,q_n$, 设A_i是 $q_{i-1} \times q_i$ 阶矩阵
- n个矩阵连乘不同次序个数$p(n)$:

$A_1 | (A_2 \ldots A_i \ldots A_n)$ $(A_1\ A_2) | (\ldots A_i \ldots A_n)$ $(A_1\ A_2 \ldots A_i \ldots) | A_n$

$$p(n)=\begin{cases}1 & n=1\\ \sum_{k=1}^{n-1} p(k)p(n-k) & n>1\end{cases}$$

$$\Rightarrow p(n)=C(n+1)\text{为}Catalan\text{数}$$

$$C(n)=\frac{1}{n-1}\binom{2n-4}{n-2}=\Omega\left(\frac{4^n}{n^{3/2}}\right)$$

呈指数增长

分析最优解结构、建立递推关系

- 假设定好了$A_1 \ldots A_n$一个乘法次序P：
  - 用$A[i:j]$记连乘积 $A_i \ldots A_j$, 相应计算量T[i, j]
  - 设P最后乘法在A_k后断开, 即 A[1:k]×A[k+1:n]
  - 那么P的计算量为 T[1,k] + T[k+1,n] + $q_0 \times q_k \times q_n$.
- 若P是最优解, 则P在A[1:k]和A[k+1:n]上也是最优解
- 最优子结构性质：最优策略的子策略也是最优的.
- 设$A[i:j]$的最小计算量为m[i,j], 那么

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

最优值与最优解的区别

- 输入：给定矩阵$A_1, A_2, \ldots, A_n$的维数序列：
  $q_0, q_1, \ldots, q_n$，即 A_i 是 $q_{i-1} \times q_i$ 阶矩阵
- 输出：计算量最小的乘法次序

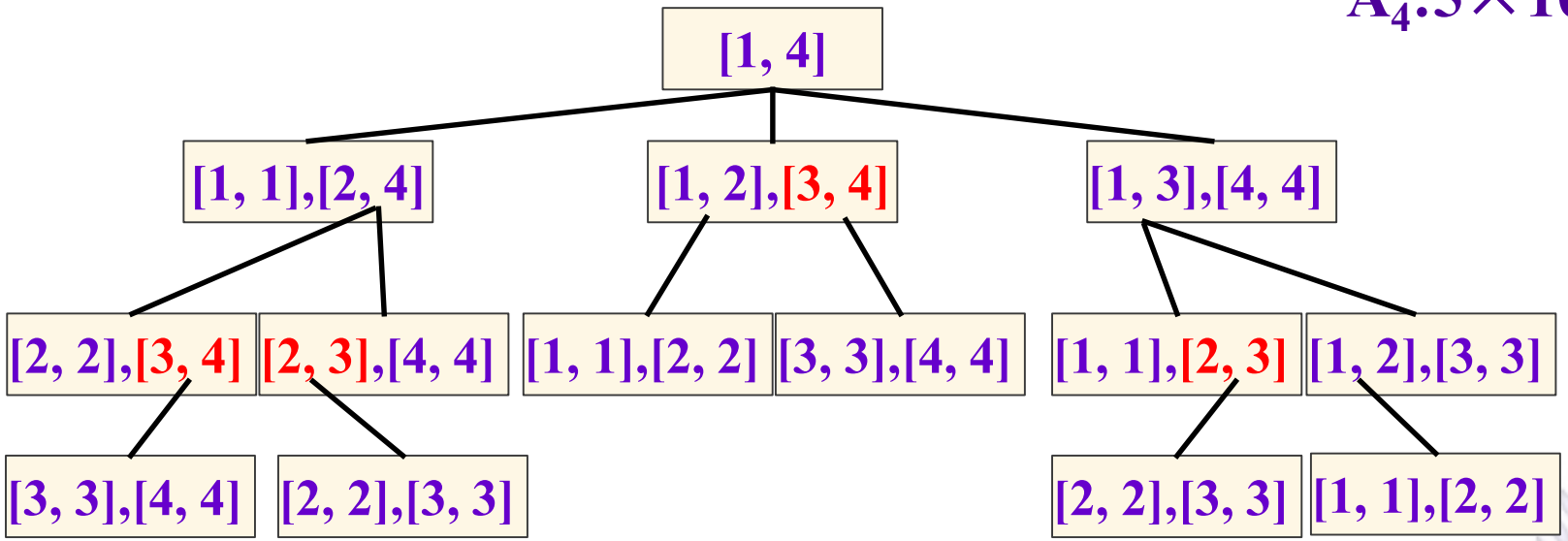
- 最优解：计算量最小的乘法次序
- 最优值：最优解的计算量

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

重叠子问题

设维数序列为(30, 35, 15, 5, 10)

A_1:30×35
A_2:35×15
A_3:15×5
A_4:5×10



DP适用条件和设计步骤

- OSP: 最优策略的子策略也是最优.
- 重叠子问题性质: 记录中间结果.

设计步骤 1) 描述最优解的结构

2) 递归定义最优解

3) 自底向上计算最优值

4) 由计算结果构造最优解

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

观察最优值计算

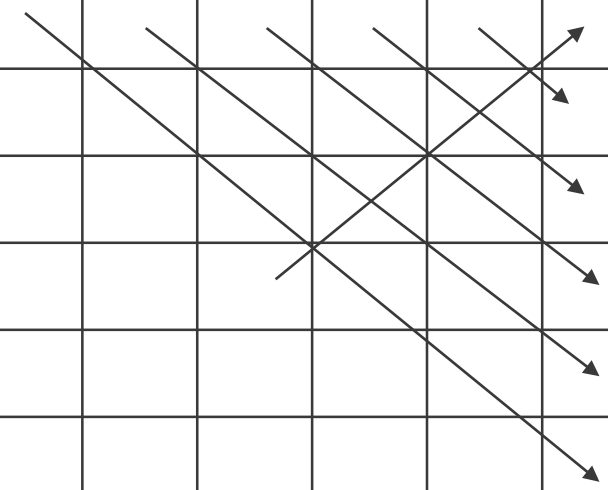
- 需要计算的是m[1, n], 但没法直接计算
  - m[1,1]=m[2,2]=…=m[n,n]=0
  - m[1,2] = ?
  - m[1,2] = $q_0 \times q_1 \times q_2$, m[2,3] = $q_1 \times q_2 \times q_3$, m[3,4], …
  - m[1,3] = ?
  - min{ m[1,1] + m[2,3] + $q_0 \times q_1 \times q_3$, m[1,2] + m[3,3] + $q_0 \times q_2 \times q_3$ }
- 自底向上计算; 表格化方法

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

计算最优值图示

设维数序列为(30, 35, 15, 5, 10, 20, 25)

m		i					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						



m		i					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

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

计算最优值图示

(30, 35, 15, 5, 10, 20, 25)

$m[1,2] = q_0 \times q_1 \times q_2 = 30 \times 35 \times 15 = 15750$

m		i					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

m		i					
		1	2	3	4	5	6
j	1	0	15750				
	2		0	2625			
	3			0	750		
	4				0	1000	
	5					0	5000
	6						0

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

计算最优值图示

$m[1,3] = \min\{ m[1,1]+m[2,3]+q_0\times q_1\times q_3,\ m[1,2]+m[3,3]+q_0\times q_2\times q_3 \}$

$= \min\{ 0+2625+30\times35\times5,\ 15750+0+30\times15\times5 \} = \min\{18000, 7875\}$

m		i					
		1	2	3	4	5	6
j	1						
	2						
	3						
	4						
	5						
	6						

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	1	0	15750	7875			
	2		0	2625	4375		
	3			0	750	2500	
	4				0	1000	3500
	5					0	5000
	6						0

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

计算最优值图示

m[1,4]

= min {m[1,1]+m[2,4]+$q_0 \times q_1 \times q_4$,
m[1,2]+m[3,4]+$q_0 \times q_2 \times q_4$,
m[1,3]+m[4,4]+$q_0 \times q_3 \times q_4$}

= min {0 + 4375 + 30×35×10,
15750 + 750 + 30×15×10,
7875 + 0 + 30×5×10}

= min { 9375, 21000, 14875 }

=9375

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	1	0	15750	7875	9375		
	2		0	2625	4375	7125	
	3			0	750	2500	5375
	4				0	1000	3500
	5					0	5000
	6						0

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

计算最优值图示

m[1,5]
= min { 9375 + 0 + 30×10×20,
7875 + 1000 + 30×5×20,
15750 + 2500 + 30 × 15 × 20,
0 + 7125 + 30 × 35 × 20 }
= min { 15750, 11875, 27250,
28125 }=11875

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	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

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

计算最优值算法

输入: n和维数序列 $(q_0,q_1,\ldots,q_n)$

1. 对 i = 1 到 n, m[i,i]=0,
2. 对 r = 1 到 n-1
3. 　　对i = 1 到 n-r
4. 　　j = i + r; m[i,j] = INF;
5. 　　对 k = i 到 j-1
6. 　　t =m[i,k]+m[k+1,j]+ $q_{i-1}\times q_k\times q_j$,
7. 　　若m[i,j]>t, 则m[i,j]=t
8. 输出: m[1,n]

INF如何处理? 如何构造最优解?

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	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

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

计算最优值同时标记分断点

m		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	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

s		(30, 35, 15, 5, 10, 20, 25)					
		1	2	3	4	5	6
j	1		1	1	3	3	3
	2			2	3	3	3
	3				3	3	3
	4					4	5
	5						5
	6						

计算最优值同时标记分断点

输入: n和维数序列 $(q_0,q_1,\ldots,q_n)$

1. 对 i = 1 到 n, m[i,i]=0,
2. 对 r = 1 到 n-1
3. 对i = 1 到 n-r
4. j = i + r; **s[i,j] = i;**
5. m[i,j] = m[i,i]+m[i+1,j]+ $q_{i-1}\times q_i\times q_j$;
6. 对 k = i + 1 到 j-1
7. t =m[i,k]+m[k+1,j]+ $q_{i-1}\times q_k\times q_j$,
8. 若m[i,j]>t, 则m[i,j]=t; **s[i,j]=k;**

输出: s //**s[i,j]是计算[i:j]段的分断点**

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

构造最优解

Traceback(1, n, s) //输出最优解, s[i,j]是[i:j]的最优分断点

Traceback(i, j, s) //[C]

1. 若i = = j, 打印 “A”,i
2. 否则　打印 “(”
3. 　　Traceback(i, s[i,j], s)
4. 　　Traceback(s[i,j]+1,j,s)
5. 　　打印 “)”

10, 100, 5, 50

s[1,3]=2

输出样例:

((*A*1 *A*2) *A*3)

Traceback2(i,j,s) //[王], 书中解释不匹配

1. 若 i == j 返回
2. Traceback2(i, s[i,j],s)
3. Traceback2(s[i,j]+1,j,s)
4. 打印 “A”, i, “,”, s[i,j], “× A”, s[i,j]+1, “,”, j

输出样例

A 1,1 × *A* 2,2

A 1,2 × *A* 3,3

DP适用条件和设计步骤

- OSP: 最优策略的子策略也是最优.
- 重叠子问题性质：记录中间结果.

设计步骤 1) 描述最优解的结构

2) 递归定义最优解

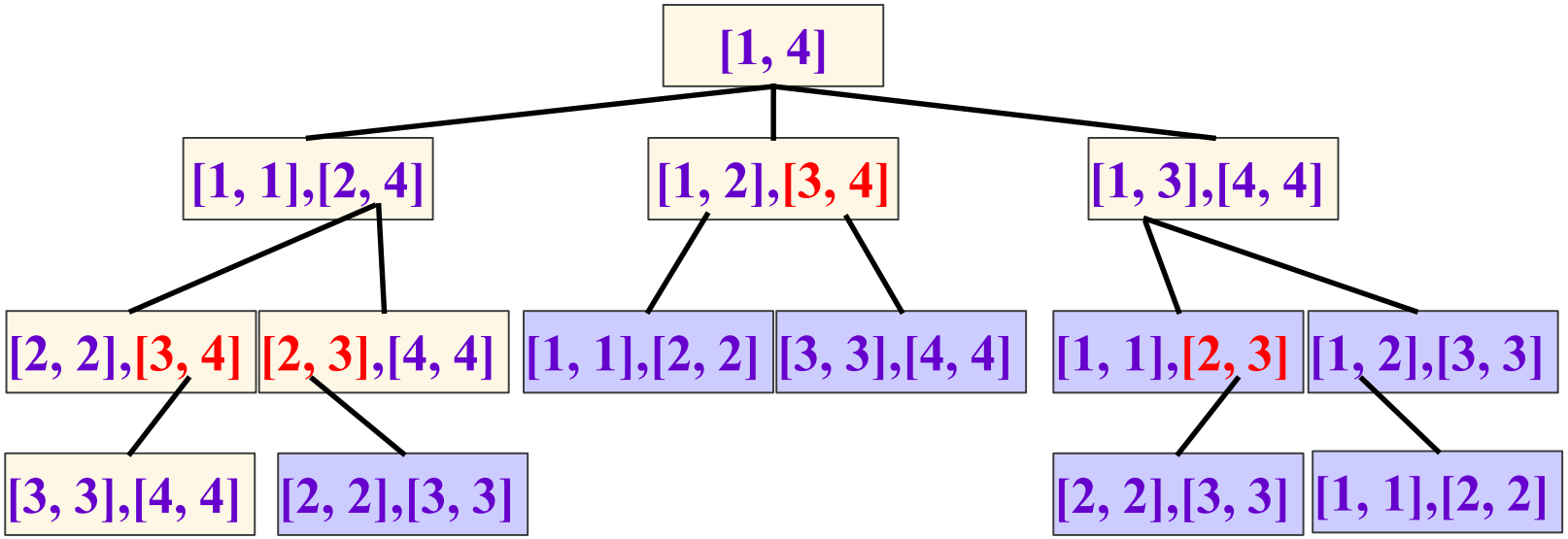
3) 自底向上计算最优值 → 自顶向下

4) 由计算结果构造最优解

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

重叠子问题

自顶向下计算m[1][4]过程如下：



设置备忘录

备忘录方法: 自顶向下

- **MEMOIZED-MATRIX-CHAIN**

  1. 对所有i,j, m[i,j]=0 // m[i,j]=0表示该值尚未计算
  2. LU(1,n)　　//LookUp

- **LU(i,j)//若m[i,j]>0表示该值已经计算，直接返回该值**

  1. 若 m[i,j]>0, 返回 m[i,j]
  2. 若 i = = j, 返回0
  3. s[i,j] = i; m[i,j] = LU(i,i) + LU(i+1,j) + $q_{i-1}\times q_i\times q_j$; //计算m[i,j]
  4. 对 k = i + 1 到 j-1
  5. 　　t = LU(i, k) + LU(k+1,j) + $q_{i-1}\times q_k\times q_j$,
  6. 　　若m[i,j]>t, 则m[i,j]=t; s[i,j]=k;
  7. 返回m[i][j]

自底向上与自顶向下的比较

- 动态规划方法采用自底向上
- 备忘录方法采用自顶向下
- 都能解决重叠子问题
- 当所有子问题都至少要求解一次时，用动态规划方法比较好
- 当部分子子问题不用求解时，用备忘录方法比较好
- 矩阵连乘问题宜用动态规划

练习1

- 在一个铁路沿线顺序存放着n堆装满货物的集装箱。
- 货物储运公司要将集装箱有次序地集中成一堆。
- 规定每次只能选相邻的2堆集装箱合并成新的一堆，所需的运输费用与新的一堆中集装箱数成正比。
- 给定各堆的集装箱数，试制定一个运输方案，使总运输费用最少。
- 例如：
  - 5, 3, 4, 2, 3, 4, 3, 4
  - 为了简化，分析4, 2, 3, 4

练习1

- 分析P[1,2,3,4]: 4, 2, 3, 4
  - 方案1：(4,(2,(3,4))) 总运输费用 (4+3)+(7+2)+(9+4) = 29
  - 方案2：(((4,2),3),4) 总运输费用 (4+2)+(6+3)+(9+4) = 28
  - 方案3：((4,(2,3)),4) 总运输费用 (2+3)+(4+5)+(9+4) = 27
  - 方案4：((4,2),(3,4)) 总运输费用 (4+2)+(3+4)+(6+7) = 26
- P[1,4]的子问题
  - (1),(2,3,4): P[1,1]+P[2,4]+sum(1,4)
  - (1,2),(3,4): P[1,2]+P[3,4]+sum(1,4)
  - (1,2,3),(4): P[1,3]+P[4,4]+sum(1,4)

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

练习　　4, 2, 3, 4

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

	$j=1$	2	3	4
$i=1$				
2				
3				
4				

	$j=1$	2	3	4
$i=1$	0	6	14	26
2		0	5	14
3			0	7
4				0

练习2

- 在一个圆形操场的四周存放着n堆装满货物的集装箱。
- 货物储运公司要将集装箱有次序地集中成一堆。
- 规定每次只能选相邻的2堆集装箱合并成新的一堆，所需的运输费用与新的一堆中集装箱数成正比。
- 给定各堆的集装箱数，试制定一个运输方案，使总运输费用最少。
- 例如：
  - 5, 3, 4, 2, 3, 4, 3, 4
  - 为了简化，分析4, 2, 3, 4

第三章 动态规划
dynamic programming

1. 动态规划一般原理(与分治对比), Bellman, OSP
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)

如何使用DP解决问题

- DP的适用条件是OSP和重叠子问题
- 动态规划的关键是子结构和决策(量)
- 如何设计?
- 递归思考: 由小到大, 由简到繁
- 尝试、调整子结构和决策量.
- 矩阵连乘 :子结构, 决策, 决策量?

  子结构--[i:j], 决策--分段点, 决策量: 计算量
- 这里的决策(量)是自然的, 有时需要调整
- 怎么会想到用这样的子结构呢?

  多试几步就能发现: [1:n]→[1:k], [k+1,n]→…

3.1 最长公共子序列(LCS, [王,M,C])

- 输入: 字符串X=$x_1x_2\ldots x_n$, Y=$y_1y_2\cdots y_m$,
- 输出: X和Y一个最长的公共子序列(LCS) Z = $z_1z_2\cdots z_k$.
- 样例:
  - X=ABCBDAB,
  - Y=BDCABA
- LCS
  - BCBA（输出）
  - BCAB
  - BDAB

- 方法1：穷举
  - 找出X的所有子序列
  - 比较哪些是Y的子序列
  - 记录最长的序列
  - 指数级
- 方法2：
  - 动态规划
  - 多项式级

最长公共子序列(LCS, [王,M,C])

- Longest Common Subsequence
- 输入: 字符串X=$x_1x_2\ldots x_n$, Y=$y_1y_2\ldots y_m$,
- 输出: X和Y一个最长的公共子序列(LCS) Z = $z_1z_2\ldots z_k$.
- 样例: X=ABCBDAB
- Y=BDCABA

- 子序列：字符串中的几个顺序字符
  - BCA、CDB、BCAB都是X的子序列
  - BCA、BCAB也是Y的子序列
  - BCAB是X和Y最长的公共子序列
  - BCBA、BDAB也是X和Y最长的公共子序列

最长公共子序列：子问题和递归结构

- 输入: 字符串$X=x_1x_2\ldots x_n$, $Y=y_1y_2\ldots y_m$,
- 输出: X和Y一个最长的公共子序列(LCS) $Z = z_1z_2\ldots z_k$.
- 样例: X=ABCBDAB
- Y=BDCABA

- 自然的子结构: $X_i = x_1x_2\ldots x_i$, $Y_j = y_1y_2\ldots y_j$.
- 自然的决策(量): LCS(长度)
- 如何寻找最优决策? (回顾矩阵连乘问题)
- 矩阵连乘:找分段点. LCS: x_i =?= y_j .
- 递归思考(OSP?):
  - 若 $x_i = y_j$, 则 (X_i,Y_j)LCS含有(x_i, y_j)配对;
  - 若 $x_i \neq y_j$, 则 (X_i,Y_j)的LCS是(X_{i-1},Y_j)或(X_i,Y_{j-1})的LCS.

LCS: 递归定义最优解

- 输入: $X=x_1x_2\ldots x_n$, $Y=y_1y_2\ldots y_m$, 输出: X和Y的LCS
- 定义: c[i][j] = X_i,Y_j的LCS长度

$$c[i][j]=\begin{cases}0 & i=0或j=0\\ c[i-1][j-1]+1 & i,j>0且x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0且x_i\neq y_j\end{cases}$$

LCSlength(n,m,x,y,c) //求最长公共子序列的长度

1. 初值 c[0][1:m]=0, c[1:n][0]=0
2. 对i=1:n, j=1:m
3. 若x[i]==y[j], 则 c[i][j]=c[i-1][j-1]+1
4. 否则 若c[i][j-1]>c[i-1][j], 则 c[i][j]=c[i][j-1]
5. 否则 c[i][j]=c[i-1][j]

输出c[n][m].

T(n)=O(mn) ;
S(n)=mn

LCS: 递归定义最优解

- c[i][j] = LCS长度
- b[i][j]记录c[i][j]是由哪个子问题决定的
  - ¶ 1↖:x[i]==y[j]
    c[i][j]=c[i-1][j-1]+1
  - ¶ 2↑:c[i-1][j]>=c[i][j-1]
    c[i][j]=c[i-1][j]
  - ¶ 3←:c[i-1][j]<c[i][j-1]
    c[i][j]=c[i][j-1]

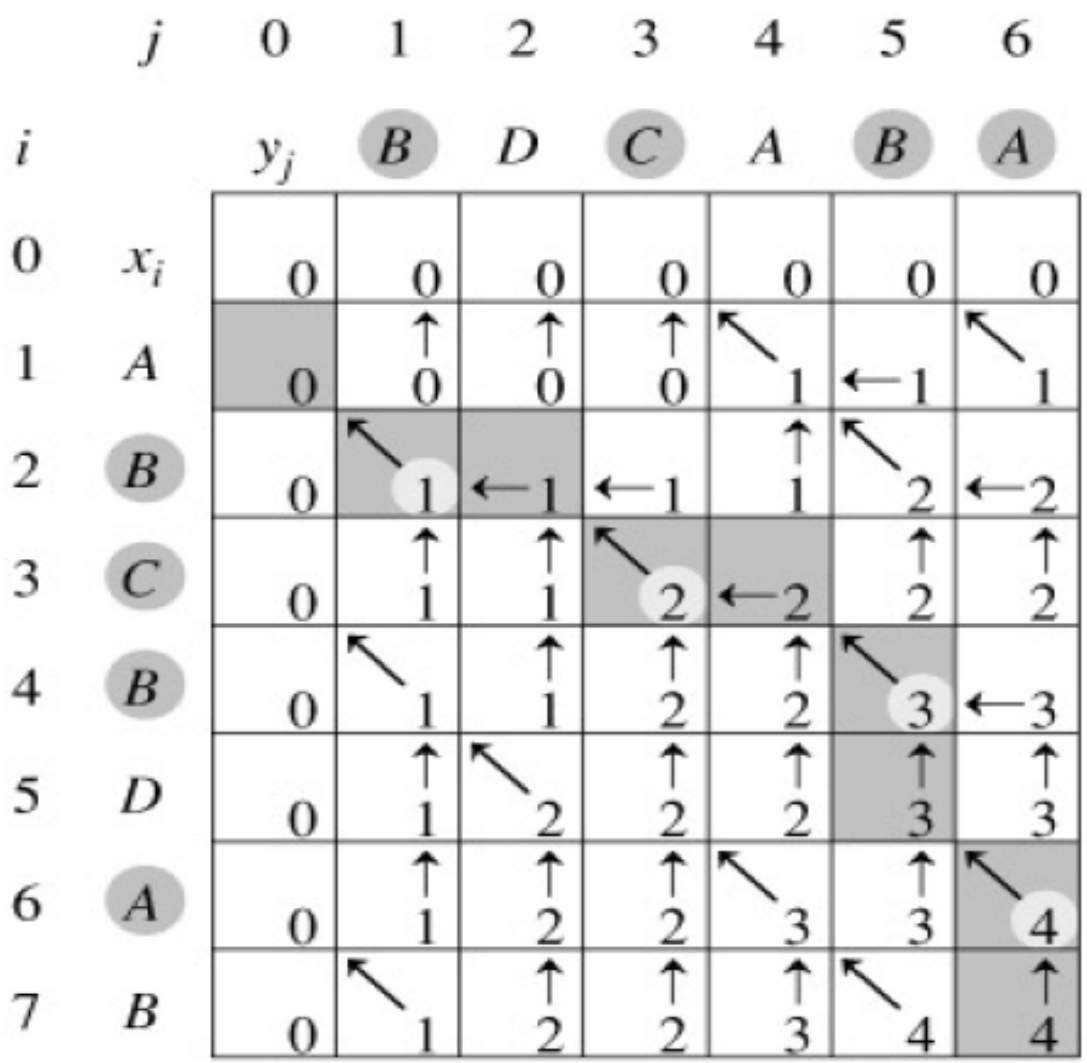
i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	0↑	0↑	0↑	1↖	1←	1↖
2	B		0	1↖	1←	1←	1↑	2↖	2←
3	C		0	1↑	1↑	2↖	2←	2↑	2↑
4	B		0	1↖	1↑	2↑	2↑	3↖	3←
5	D		0	1↑	2↖	2↑	2↑	3↑	3↑
6	A		0	1↑	2↑	2↑	3↖	3↑	4↖
7	B		0	1↖	2↑	2↑	3↑	4↖	4↑

LCS: 构造最优解

```
LCS( )//求最长公共子序列
//b[i][j]记录c[i][j]是由哪个子问题决定的
对i=1:n, j=1:m
1:  if (x[i]==y[j]) {
2:     c[i][j]=c[i-1][j-1]+1;
3:     b[i][j]=1;}
4:  else if (c[i-1][j]>=c[i][j-1]) {
5:     c[i][j]=c[i-1][j];
6:     b[i][j]=2;}
7:  else {
8:     c[i][j]=c[i][j-1];
9:     b[i][j]=3;}
```

解：b[7][6]=1，BCBA

T(n)=O(mn) ; S(n)=mn



空间优化

•数组元素c[i][j]的值仅由c[i-1][j-1]，c[i-1][j]和c[i][j-1]这3个数组元素的值所确定。对于给定的数组元素c[i][j]，可以不借助于数组b而仅借助于c本身在常数时间内确定c[i][j]的值。

•优化1：省略数组b

•优化2：c改为滚动数组

•滚动数组把m*n的二维数组压缩成 2*n的数组

•只需要用到当前层(now)和上一层数据(pre)，通过now和pre的不断交换实现m*n的数组功能，可优化空间复杂度。

$$c[i][j]=\begin{cases}0 & i=0\text{或}j=0\\ c[i-1][j-1]+1 & i,j>0\text{且}x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0\text{且}x_i\neq y_j\end{cases}$$

3.2 最大子段和(ms,[王,M])

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: 有最大和的子段 (x_i, x_{i+1},…,x_j)
- 样例输入: (1, −3, 4, −1, 2, 1, −5, 4)
- 样例输出: (4, -1, 2, 1)
- 注: 规定子段和为负数时， 定义其最大子段和为0 ， ms ≥ 0.

- 穷举、分治、动规都试一下
- 穷举法(穷举法是基础):
  - 1. 对i, 求[1:i]和.
  - 2. 对i, j, 求[i:j]和并取最大.
  - $O(n^2)$

最大子段和(ms)-分治

•样例输入: (1, −3, 4, −1, 2, 1, −5, 4) 输出: (4, -1, 2, 1)

1. 分成两段S_L和S_R.

2. 递归求两段的ms: M_L, M_R.

3. 求跨两段的ms与M_L, M_R的最大.

合并法: S_L的最大尾ms + S_R的最大头ms

$f(n) = \Theta(n)$,

$T(n) = \Theta(n\log n)$.

$$T(n)=\begin{cases} O(1) & n=1 \\ 2T(n/2)+f(n) & n>1 \end{cases}$$

最大子段和(ms)的递归思考

- 输入: $(x_1,x_2,\ldots,x_n)$, 输出: ms.
- 例: (1, −3, 4, −1, 2, 1, −5, 4)
- 规定子段和为负数时，定义其最大子段和为0 .

- 使用什么子结构,决策量? [i:j], [1:i]? 试[1:i], ms.
- 递归尝试: 已知[1:i-1]的ms[i-1], 求[1:i]的ms[i]?
- ms[i-1] = max{ ∅,[1:1],[1:2],…,[i-1,i-1] }. $O(n^2)$项,
- ms[i] : max{ ms[i-1], [1:i], [2:i], …, [i:i] }.
- 复杂度 $O(n^2)$, 怎么改进?
- 定义 tms[i] :=[1:i]的含i最大子段和
- 即tms[i]: max{ [1:i], [2:i], …, [i-1:i], [i:i], ∅ }.

最大子段和(ms)的递归思考

- 输入: $(x_1,x_2,\ldots,x_n)$, 输出: ms. 例: (1, −3, 4, −1, 2, 1, −5, 4)
- 规定子段和为负数时，定义其最大子段和为0 .
- 递归尝试: 已知[1:i-1]的ms[i-1], 求[1:i]的ms[i]?
- ms[i] : max{ ms[i-1], [1:i], [2:i], …, [i:i] }.
- 定义 tms[i] :=[1:i]的含i最大子段和
- 即tms[i]: { [1:i], [2:i], …, [i-1:i], [i:i], ∅ }.

		1	-3	4	-1	2	1	-5	4
tms	0	1	0	4	3	5	6	1	5
ms	0	1	1	4	4	5	6	6	6

$$tms[i] = \max\{ 0,\ x_i + tms[i-1] \},\ i \geq 1$$

$$ms[i] = \max\{ ms[i-1], tms[i] \},\ i \geq 1)$$

最大子段和(ms)的递归思考

- 输入: $(x_1,x_2,\ldots,x_n)$, 输出: ms. 例: (1, −3, 4, −1, 2, 1, −5, 4)
- 规定子段和为负数时，定义其最大子段和为0 .
- 递归尝试: 已知[1:i-1]的ms[i-1], 求[1:i]的ms[i]?
- ms[i] : max{ ms[i-1], [1:i], [2:i], …, [i:i] }.
- 定义 tms[i] :=[1:i]的含i最大子段和
- 即tms[i]: { [1:i], [2:i], …, [i-1:i], [i:i], ∅ }.

- 递归调整: 已知[1:i-1]的ms, tms, 求[1:i]的ms, tms
- tms[i-1] = max { [1:i-1], [2:i-1], …, [i-1:i-1], ∅ }.
- tms[i] = max{ 0, x_i + tms[i-1] }, i ≥ 1
- ms[i] = max{ ms[i-1], tms[i] }, i ≥ 1
- 初始时： tms=0; ms=0

最大子段和(ms)的递归思考

```
int MaxSubSum3(int n, int a[])
{
   int sum=0, tms=0;   //sum存储当前最大
   for(int j=1; j<=n; j++)  {
      tms += a[j];
      if(tms<0) tms=0;  //若和为负，则从下一个位置累和
      if(tms>sum) sum=tms;
    }
    return sum;
}
```

运行时间：T(n)=O(n)

		1	-3	4	-1	2	1	-5	4
tms	0	1	0	4	3	5	6	1	5
ms	0	1	1	4	4	5	6	6	6

最长递增子序列(LIS[M])

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 输入样例: n=10,(2,8,9,4,6,1,3,7,5,10), OSP?
- 输出样例: (2,4,6,7,10)
- 穷搜(指数时间?) 动态规划? 子结构? 决策量?

归纳尝试一： 已知[1:i]的1个LIS, 求[1:i+1]的LIS?

- 假设已知 (2,8,9,4,6,1,3) 的1个LIS (2,8,9), 加入7
- 7不能加长(2,8,9), 但可加长(2,4,6). 启示,调整?
- 启示: x_{i+1}可能可以使得其它LIS变长
- 调整: 记录尾巴最小的LIS: MTLIS

添加修改递归量

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 尾巴最小的LIS: MTLIS

归纳尝试二：知[1:i]的MTLIS, 求[1:i+1]的MTLIS?

- 假设已知(2,8,9,4) 的MTLIS (2,8,9), 加入6
- 6不能加长(2,8,9), 但(2,4,6)是新的MTLIS. 启示调整?
- 启示: x_{i+1}可能可以构成新的MTLIS
- 调整: 需要记录最长、次长、第3长等的MTIS
- 调整: 需要记录长为k的MTIS: MTIS(k), k=1,2,…

归纳尝试三：知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)	2									
MTIS(2)	2	8								
MTIS(3)	2	8	9							

加入4?

能否得到MTIS(4)
能否改变MTIS(3)
能否改变MTIS(2)
能否改变MTIS(1)

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)	2									
MTIS(2)	2			4						
MTIS(3)	2	8	9							

加入6?

能否得到MTIS(4)
能否改变MTIS(3)
能否改变MTIS(2)
能否改变MTIS(1)

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)	2									
MTIS(2)	2			4						
MTIS(3)	2			4	6					

加入1?

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)	2			4						
MTIS(3)	2			4	6					

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)	2			4	6					

MTIS计算举例

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)	2			4	6					
MTIS(4)	2			4	6			7		

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3		5	
MTIS(4)	2			4	6			7		

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3		5	
MTIS(4)	2			4	6			7		
MTIS(5)	2			4	6			7		10

时间复杂度?
$O(n^3)$
如果只记录尾巴?
$O(n^2)$?

MTIS计算举例

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3		5	
MTIS(4)	2			4	6			7		

尾巴递增!

有且仅有一个尾巴改变!

MTIS(k).last

len: 最大长度

- x_{i+1} < MTIS(1).last
- x_{i+1} ≥ MTIS(len).last
- x_{i+1} ≥ MTIS(s-1).last

  x_{i+1} < MTIS(s).last

MTIS计算举例

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)	2			4	6					

尾巴递增?

有且仅有一个尾巴改变?

MTIS(k).last

len: 最大长度

- x_{i+1} < MTIS(1).last
- $x_{i+1} \ge$ MTIS(len).last
- $x_{i+1} \ge$ MTIS(s-1).last

  x_{i+1} < MTIS(s).last

MTIS计算举例

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3			
MTIS(4)	2			4	6			7		

尾巴递增?

有且仅有一个尾巴改变?

MTIS(k).last

len: 最大长度

- x_{i+1} < MTIS(1).last
- $x_{i+1} \ge$ MTIS(len).last
- $x_{i+1} \ge$ MTIS(s-1).last

  x_{i+1} < MTIS(s).last

MTIS计算举例

- 输入: 实数序列(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.
- 长为k, 尾巴最小的IS: MTIS(k), k=1,2,…

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
MTIS(1)						1				
MTIS(2)						1	3			
MTIS(3)						1	3		5	
MTIS(4)	2			4	6			7		

尾巴递增?

有且仅有一个尾巴改变?

MTIS(k).last

len: 最大长度

- x_{i+1} < MTIS(1).last
- $x_{i+1} \ge$ MTIS(len).last
- $x_{i+1} \ge$ MTIS(s-1).last

  x_{i+1} < MTIS(s).last

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0									
MTIS(1).last	2									
MTIS(2).last										
MTIS(3).last										
MTIS(4).last										
MTIS(5).last										

F[i] 当前节点的前驱，即父亲

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1								
MTIS(1).last	2									
MTIS(2).last		8								
MTIS(3).last										
MTIS(4).last										
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2							
MTIS(1).last	2									
MTIS(2).last		8								
MTIS(3).last			9							
MTIS(4).last										
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1						
MTIS(1).last	2									
MTIS(2).last				4						
MTIS(3).last			9							
MTIS(4).last										
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4					
MTIS(1).last	2									
MTIS(2).last				4						
MTIS(3).last					6					
MTIS(4).last										
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0				
MTIS(1).last						1				
MTIS(2).last				4						
MTIS(3).last					6					
MTIS(4).last										
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6			
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last					6					
MTIS(4).last										
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6	5		
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last					6					
MTIS(4).last								7		
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6	5	7	
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last									5	
MTIS(4).last								7		
MTIS(5).last										

只记录尾巴, 增加父亲标记

序号	1	2	3	4	5	6	7	8	9	10
实数序列	2	8	9	4	6	1	3	7	5	10
父亲	0	1	2	1	4	0	6	5	7	8
MTIS(1).last						1				
MTIS(2).last							3			
MTIS(3).last									5	
MTIS(4).last								7		
MTIS(5).last										10

输出：2，4，6，7，10

时间复杂度: $O(n^2) \rightarrow O(n \log n)$

MTIS计算举例

- 输入: 实数序列$(x_1,x_2,\ldots,x_n)$
- 输出: $x_{i_1}\le x_{i_2}\le\ldots\le x_{i_k}$, 其中k最大且$i_1<i_2<\ldots<i_k$.

归纳三:知[1:i]的MTIS(k), 求[1:i+1]的MTIS(k)

- 为什么每添加一数只有一个MTIS(k)会改变?
- 性质：尾数单增，即MTIS(1).last ≤ MTIS(2).last ≤ …
- 证明: 若MTIS(i).last > MTIS(i+1).last, 矛盾.
- x_{i+1}改变MTSI (s): MTIS(s-1).last ≤ x_{i+1} < MTIS(s).last
- 怎么找修改位置?
- 采用二分搜索找s, 时间O(logn)×n.

[王]第三章习题1,2

LIS算法

- 输入: 实数序列X(x_1,x_2,…,x_n)
- 输出: $x_{i_1} \le x_{i_2} \le \ldots \le x_{i_k}$, 其中k最大且$i_1 < i_2 < \ldots < i_k$.

1. 数组X, L, F //X是输入, F[i]: 记录X[i]的父亲
   //L[i]: X[L[i]]=MTIS(i).last, 即MTIS(i).last的位置
2. L[i] = 0, F[i] = 0; L[1] = 1, F[1] = 0, len = 1 //len:最大长度
3. 对于 i = 2 : n, // X[L[1]] ≤ X[L[2]] ≤ … ≤ X[L[len]]; X[i]?
4. 若X[L[1]] > X[i], 则 F[i] = 0; L[1] = i.
5. 若X[L[len]] ≤ X[i], 则 F[i] = L[len]; len++; L[len] = i.
6. 否则 二分搜索L[1:len], 求s使得X[L[s-1]] ≤ X[i] < X[L[s]]
7. F[i]=L[s-1]; L[s]=i.
8. pt = L[len];
9. 对 i = len:1, D[i] = A[pt]; pt = F[pt]. 输出D.

时间复杂度: O(n logn)

第三章 动态规划
dynamic programming

1. 动态规划一般原理(与分治对比), Bellman, OSP
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)

0-1背包问题Knapsack([王]p71)

- 输入: n物品重W[1:n], 价值V[1:n], 背包容量C
- 输出: 装包使得价值最大 (物品重量为整数).
  - 例: W:{2,2,6,5,4},V:{6,3,5,4,6}, C=10?
- 子结构? 决策量? 递推关系? 最优值?
- 1) 子结构[1:i], g[i, k] = 由[1:i]组合出重量≤k的最大价值
  （或者说 前i种物品装入重量上限为k背包的最大价值）
- 2) g[i, k] = max{ g[i-1,k], g[i-1,k-W[i]] + V[i] } (OSP)
  即考虑是否装物品i，若装入后在满足容量限制的前提下总价值更大则装入
- 3)最优值: g[n, C]

0-1背包问题OSP

$$max\sum_{i=l}^{n} v_i x_i$$

$$\begin{cases} \sum_{i=l}^{n} w_i x_i \le c \\ x_i \in \{0,1\} \quad l \le i \le n \end{cases} \overset{记}{\Rightarrow} Knap(l,n,c)$$

- 证明最优子结构性：设$(y_1,y_2,\ldots,y_n)$是Knap(1,n,c)的一个最优解，则
- $(y_2,\ldots,y_n)$是Knap(2,n,$c-w_1y_1$)子问题的一个最优解。反证法
- 否则，设$(z_2,\ldots,z_n)$是Knap(2,n,c-w_1y_1)的最优解，考察解$(y_1,z_2,\ldots,z_n)$

$$\sum_{i=2}^{n} v_i z_i > \sum_{i=2}^{n} v_i y_i \text{ 且 } \sum_{i=2}^{n} w_i z_i \le c - w_1 y_1$$

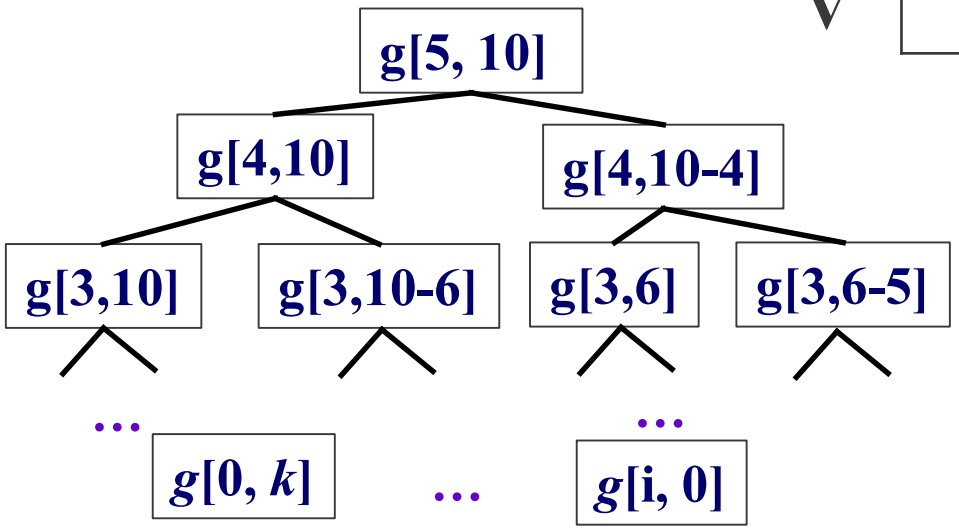
$$\Rightarrow \quad v_1 y_1 + \sum_{i=2}^{n} v_i z_i > \sum_{i=1}^{n} v_i y_i \text{ 又有 } w_1 y_1 + \sum_{i=2}^{n} w_i z_i \le c$$

说明$(y_1,z_2,\ldots,z_n)$是Knap(1,n,c)的一个更优解，矛盾。

0-1背包问题

例： n=5, C=10

	1	2	3	4	5
W	2	2	6	5	4
V	6	3	5	4	6



$g[i, k] = \max\{ g[i-1,k], g[i-1,k-W[i]] + V[i] \}$

$g[5, 10] = \max\{ g[4,10], g[4,10-4] + 6 \}$

0-1背包问题

	1	2	3	4	5
W	2	2	6	5	4
V	6	3	5	4	6

例：n=5, C=10

序号	重量	价值	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	6	0	0									
2	2	3	0	0									
3	6	5	0	0									
4	5	4	0	0									
5	4	6	0	0									

$$g[i,k] = \max\{g[i-1,k], g[i-1,k-w_i]+ v_i\} \quad g[0,k]=0 \quad g[i,0]=0$$

舍　　取

0-1背包问题

	1	2	3	4	5
W	2	2	6	5	4
V	6	3	5	4	6

例：n=5, C=10

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	***6***	6	6	6	6	6	6
2	0	0	6	6	9	9	***9***	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	***15***

最优解
11001

g[i, k] = max{ g[i-1,k], g[i-1,k-W[i]] + V[i] }

g[5, 10] = max{ g[4,10], g[4,10-4] + 6 } = max{14, 9+6}=15

g[4, 6] = max{ g[3,6], g[3,6-5] + 4 } = max{9, 0+4}=9

0-1背包: 编程

- 输入: n物品重W[1:n], 价值V[1:n], 背包容量C
- 输出: 装包使得价值最大 (物品重量为整数).
- [1:i], g[i,k] = 由[1:i]组合出重量≤k的最大价值
- g[i,k] = max{ g[i-1,k], g[i-1,k-W[i]] + V[i] } (OSP)
- 最优值: g[n,C]

1. 初始 g[0,0:C] = 0,
2. 对 k = 1:C,对 i = 1:n,
3. g[i,k] = g[i-1,k]
4. 若k ≥ W[i], pt=g[i-1,k-W[i]]+V[i];
5. 若pt > g[i,k], g[i,k]=pt
6. 输出g[n, C] //时间O(nC). 输入规模?

输入规模: max{ n, $\log_2 C$ }

0-1背包问题推广 找零钱问题

- 出纳员支付一定数量的现金. 假设他手中有几种面值的硬币，每一种都有足够多个，要求他用最少的硬币数支付规定的现金.
- 例如，现有3种硬币：它们的面值分别为1元、4元和6元. 要支付8元.
- 若按现有币制，面值有1元、2元、5元三种
  - 采用贪心算法即可。
  - 贪心策略：选择不大于当前币值的最大硬币
  - 支付8元的选择顺序：5元、2元、1元，共3枚
- 若面值分别为1元、4元和6元
  - 贪心策略是错误的
  - 支付8元的贪心选择顺序：6元、1元、1元，共3枚
  - 最优解为：4元、4元，共2枚

0-1背包问题推广 找零钱问题

$p_i(j)$：前i种硬币支付金额j所需硬币的最少个数。

v_i：第i种硬币的面额

$$p_i(j)=\begin{cases} p_{i-1}(j) & j<v_i \\ \min\{p_{i-1}(j),\ p_i(j-v_i)+1\} & j\geq v_i \end{cases}$$

不用第i种硬币　用1枚第i种硬币

$p_i(j-mv_i)+m$

$p_i(0)=0$　支付0元需要0个硬币

$p_0(j)=\infty$　支付j元但没有硬币
需要无穷个硬币

0-1背包问题推广 找零钱问题

找零钱问题：1元、4元和6元. 要支付8元.

$$p_i(j)=\begin{cases} p_{i-1}(j) & j<v_i \\ \min\{p_{i-1}(j),\ p_i(j-v_i)+1\} & j\geq v_i \end{cases}$$

	0	1	2	3	4	5	6	7	8
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	1	2	3	4	2
3	0	1	2	3	1	2	1	2	2

金额 j

p[3, 8] = min{ p[2,8], p[3,8-6] + 1 } = min{2, 2+1}=2

p[2, 8] = min{ p[1,8], p[2,8-4] + 1 } = min{8, 1+1}=2

p[2, 4] = min{ p[1,4], p[2,4-4] + 1 } = min{4, 0+1}=1

0-1背包问题推广 资源分配问题

某公司准备将4名销售员分配到3个销售点甲乙丙。 各销售点增加若干名销售员后可增加的月销售额如下表：

增加销售额（万元）	增1人	增2人	增3人	增4人
甲	12	22	30	38
乙	11	20	24	30
丙	13	25	30	36

公司每月最多可以增加销售额？万元

定义：A[i,j]表示第i个销售点增加j个人的最大收益；
d[i,j]表示前i个销售点增加j个人的最大收益。

$$d[3,4] = \max\{A[3,4]+d[2,0], A[3,3]+d[2,1], A[3,2]+d[2,2], A[3,1]+d[2,3], A[3,0]+d[2,4]\}$$

0-1背包问题推广 资源分配问题

A[i,j]

增加销售额（万元）	增0人	增1人	增2人	增3人	增4人
甲	0	12	22	30	38
乙	0	11	20	24	30
丙	0	13	25	30	36

d[i,j]

增加销售额（万元）	增0人	增1人	增2人	增3人	增4人
前0个点	0	0	0	0	0
前1个点	0	12	22	30	38
前2个点	0	12	23	33	42
前3个点	0	13	25	37	48

$$d[3,4] = \max\{A[3,4]+d[2,0],\ A[3,3]+d[2,1],\ A[3,2]+d[2,2],\ A[3,1]+d[2,3],\ A[3,0]+d[2,4]\}$$

(分数)背包问题(knapsack Prob)

0-1背包问题

- 输入: n物品重W[1:n],
  价值V[1:n],
  背包容量C
- 物品重量为整数.
- 输出: 装包使得价值最大.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

(分数)背包问题

- 输入:物品重W[1:n],价值V[1:n]
- 输出: 装包使得价值最大.
- 物品i可以取重量0~W[i]

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$0 \le x_i \le 1, 1 \le i \le n$$

(分数)背包问题

(分数)背包问题

- 输入:物品重W[1:n],价值V[1:n]
- 输出: 装包使得价值最大.
- 物品i可以取重量0~W[i]

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$0 \le x_i \le 1, 1 \le i \le n$$

贪心算法:

1. 对单位重量价值 $\{ V[i]/W[i] \}_{i=1}^{n}$ 排序
2. 优先放入单位重量价值大的物品, 直到填满

O(nlogn),

0-1背包能贪心吗? 不可

最优装载([王]p95)

0-1背包问题

- 输入: n物品重W[1:n],
  价值V[1:n],
  背包容量C
- 物品重量为整数.
- 输出: 装包使得价值最大.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} V_i x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

最优装载

- 输入: n物品重W[1:n],
  船容量C
- 输出: 装包使得件数最多.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

最优装载

最优装载

- 输入: n物品重W[1:n],
  背包容量C
- 输出: 装包使得件数最多.
- 每件物品只能取或不取

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} W_i x_i \le C$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

贪心算法(O(nlogn)):

1. 对重量 W[1:n] 升序排列

2. 优先添加重量小的物品, 直到不能再添加

贪心选择性质: 最优解可以包含重量最小的

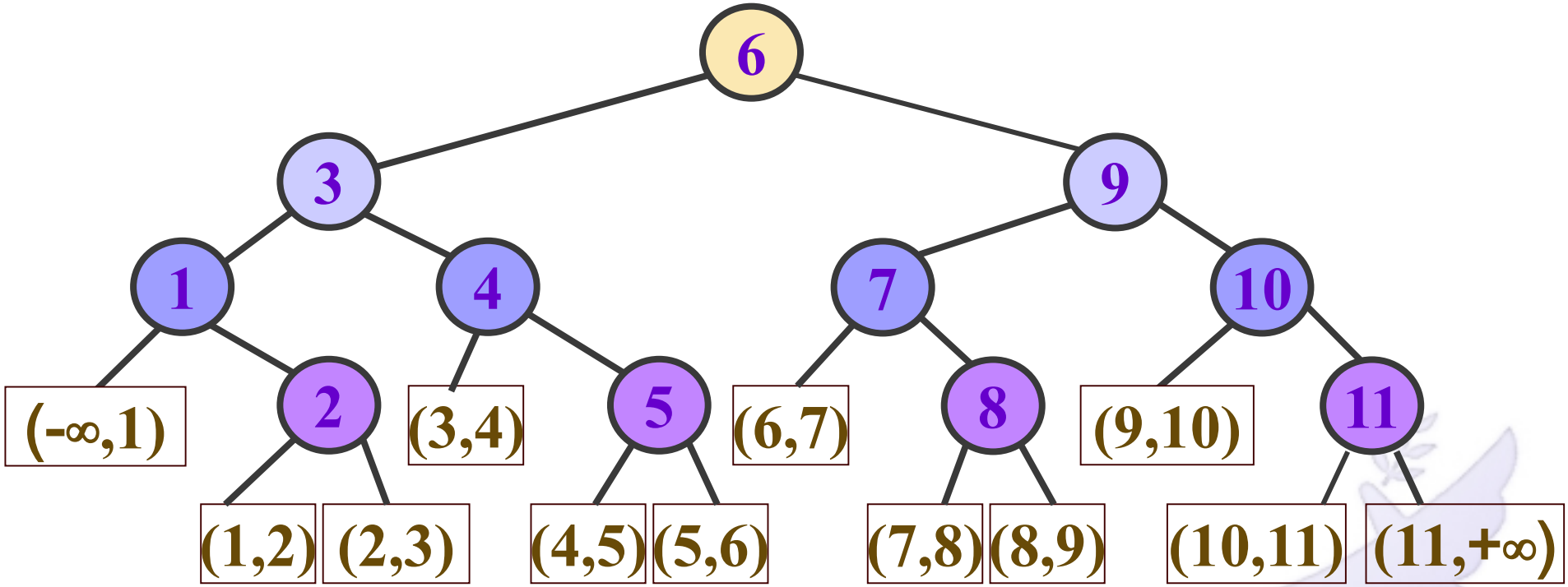
OSP: 最优解([1:n],C)去掉重量最小([2:n],C-W[1])仍是最优解

最优二叉搜索树

- 问题定义：
  - 已知有序关键字序列S={x_1, x_2, …, x_n}
  - 各关键字的搜索概率为B={b_1, b_2, …, b_n}。
  - 又已知n+1个失败区间为(-∞,x_1), (x_1, x_2),…(x_{n-1},x_n)(x_n,+∞)
  - 查询关键字在各个失败区间的概率为A={a_0,a_1, a_2, …, a_n}
  - $\sum_{i=1}^{n} b_i + \sum_{j=0}^{n} a_j = 1$
  - 问题：构造一棵二叉搜索树使其的平均比较次数p最小
  - $p = \sum_{i=1}^{n} b_i (1 + c_i) + \sum_{j=0}^{n} a_j d_j$
  - c_i为查询关键字x_i的比较次数
  - d_j为查询关键字x_i的比较次数

最优二叉搜索树

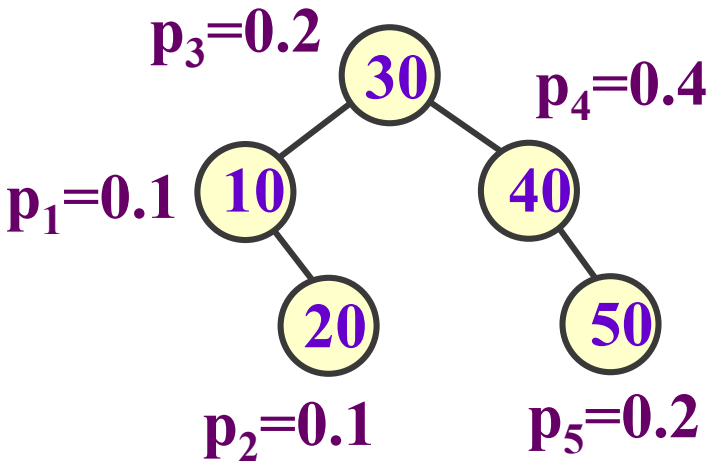
i	1	2	3	4	5	6	7	8	9	10	11
Ci	3	4	2	3	4	1	3	4	2	3	4
key	05	13	19	21	37	56	64	75	80	88	92



具有n个结点的判定树的深度为$\lceil \log_2(n+1) \rceil$

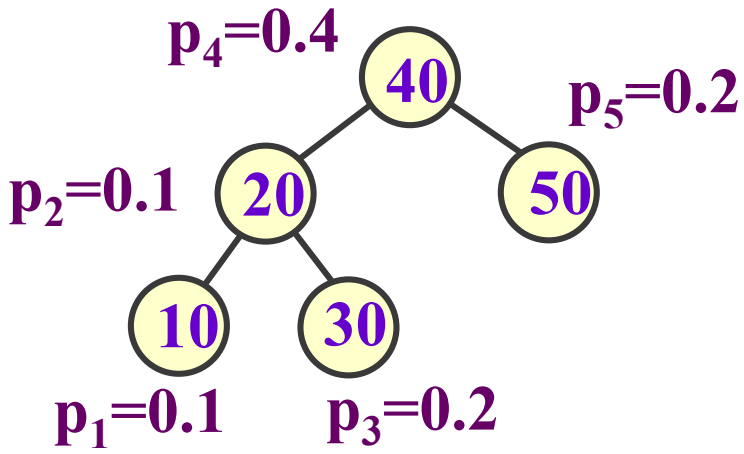
静态搜索树

- 假设已知关键字的查找概率，则最优的搜索树是其带权内路径长度之和最小的二叉树。



折半查找的判定树1

ASL_{succ}= 0.2*1+(0.1+0.4)*2 +(0.1+0.2)*3 = 2.1



折半查找的判定树2

ASL_{succ}= 0.4*1+(0.1+0.2)*2 +(0.1+0.2)*3 = 1.9

最优二叉搜索树

- 最优子结构性:

  若T是输入S、B、A上的一棵最优搜索树，则其左子树T_l和右子树T_r也是最优的。

- 设x_m为T的根结点的关键字，则
  - $S_l=\{x_1, x_2, \ldots, x_m\}$，$B_l=\{b_1, b_2, \ldots, b_m\}$，$A_l=\{a_0,a_1, \ldots, a_m\}$
  - $S_r=\{x_{m+1}, \ldots, x_n\}$，$B_r=\{b_{m+1}, b_2, \ldots, b_n\}$，$A_r=\{a_m,a_1, \ldots, a_n\}$
- 证明：反证法
  - 若T_l不是S_l上的最优树，则
  - 在S_l上的必定存在一棵最优树，设为T_l'。
  - 把T的左子树替换为T_l'则得到一棵ASL更小的二叉树。
  - 这与T是最优树矛盾。

最优二叉搜索树

- 设：
  - $T_{1,n}$为S上的最优树，p_{1n}：S上的最优树的ASL
  - $T_{i,j}$是关键字i到j的最优二叉搜索树
  - $p_{i,j}$：T_{ij}的ASL;p_l和p_r为其左右子树的ASL
  - $w_{i,j}$：关键字i到j的查找概率之和, $w_{i,j}= a_{i-1}+b_i+\ldots+b_j+a_j$;
  - $w_{i,j}p_{i,j}= w_{i,j} + w_{i,m-1}p_l + w_{m,j}p_r$.
- 递推公式

$$w_{i,j}p_{i,j} = w_{i,j} + \min_{i\le k\le j}\{w_{i,k-1}p_{i,k-1} + w_{k+1,j}p_{k+1,j}\}$$

令$m(i,j) = w_{i,j}p_{i,j}$　　$m(i,i-1) = 0$

$$m(i,j) = w_{i,j} + \min_{i\le k\le j}\{m(i,k-1) + m(k+1,j)\}$$

最优二叉搜索树

- 算法基本过程：
  - 1、初始化：m[i+1][i]=0; w[i+1][i]=a[i].
  - 2、计算m[n][n]、w[n][n]，并用s[i][j]记录Tij的根结点
  - 3、根据s构造最优树。
- 复杂度
  - 空间复杂度：$O(n^2)$
  - 时间复杂度：$O(n^3)$-->优化$O(n^2)$

$$m(i,j) = w_{i,j} + \min_{i \le k \le j}\{m(i,k-1) + m(k+1,j)\}$$

本章小结

最优子结构性质OSP

动态规划算法的设计步骤

矩阵连乘

最长公共子序列

最大子段和

最长递增子序列

0-1背包, 分数背包, 最优装载

习题

本章作业

算法分析题

3. 考虑下面的整数线性规划问题 .

即给定 序列$a_1, a_2,\ldots,a_n$, 求

$$\max\ c_1x_1+ c_2x_2+\ldots+ c_nx_n,$$

满足 $a_1x_1+ a_2x_2+\ldots+ a_nx_n \le b$, x_i为非负整数

算法实现题:

数字三角形问题, 游艇出租问题

本章作业

算法实现题:

数字三角形问题

问题描述: 给定一个有n行数字组成的数字三角形, 如下图所示. 试设计一个算法, 计算出从三角形的顶至底的一条路径, 使该路径经过的数字和最大.

算法设计: 对于给定的n行数字组成的三角形, 计算从三角形顶至底的路径经过的数字和的最大值.

数据输入: 由文件input.txt提供输入数据. 文件的第1行数字三角形的行数n, 1≤n≤100. 接下来n行是数字三角形各行中的数字. 所有数字在0~99之间.

结果输出: 将计算结果输出到文件output.txt, 文件第1行中的数是计算出的最大值.

```
   7
  3  8
 8  1  0
2  7  4  4
4 5 2  6  5
```

数字三角形

输入文件示例

input.txt

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

输出文件示例

output.txt

```
30
```

本章作业

算法实现题: 租用游艇问题

问题描述: 长江游艇俱乐部在长江上设置了n个游艇出租站1,2,…,n. 游客可在这些游艇出租站租用游艇, 并在下游的任何一个游艇出租站归还游艇. 游艇出租站i到出租站j之间的租金为r(i,j), $1 \le i < j \le n$. 试设计一个算法, 计算出从游艇出租站1到游艇出租站n所需的最少租金, 并分析算法的计算复杂性.

算法设计: 对于给定的游艇出租站i到游艇出租站j的租金r(i,j), $1 \le i < j \le n$. 计算出租站1到n所需的最少租金.

数据输入: 由文件input.txt提供输入数据. 文件的第1行有一个正整数n, $n \le 200$, 表示有n个游艇出租站. 接下来n-1行是r(i,j), $1 \le i < j \le n$.

结果输出: 将计算出的游艇出租站1到n最少租金输出到文件output.txt.

输入文件示例

input.txt

```
3
5 15
7
```

输出文件示例

output.txt

```
12
```

附录

注记1: 最优子结构性质

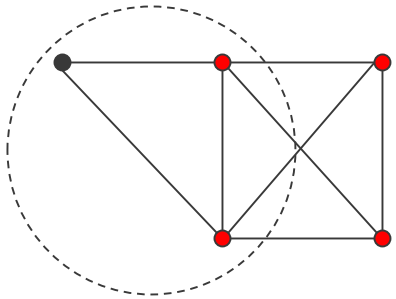
矩阵连乘, 最长公共子序列, 最大子段和

全路径最短路 单源最短路 带负权最短路

最长递增子序列:

0-1背包, 分数背包, 最优装载

最长路径问题，最大团(完全子图)问题不满足OSP



注记2

动态规划 R. Bellman 1955

矩阵连乘 $O(n^3)$, 现$O(n \log n)$

最长公共子序列O(mn), Knuth问是否最优

最大子段和 [王, M]

最长递增子序列 [王, M]Gries 1981

0-1背包, NP完全

递推关系对比:如何节省空间

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

$$c[i][j]=\begin{cases}0 & i=0\text{或}j=0\\ c[i-1][j-1]+1 & i,j>0\text{且}x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0\text{且}x_i\neq y_j\end{cases}$$

LCSlength(n,m,x,y,c)

1. 初值 c[0:1][0:m]=0

2. 对i=1:n, j=1:m

3.　若x[i]==y[j], 则 c[i%2][j]=c[(i-1)%2][j-1]+1

4.　否则 若c[i%2][j-1]>c[(i-1)%2][j], 则 c[i%2][j]=c[i%2][j-1]

5.　　否则 c[i%2][j]=c[(i-1)%2][j]

输出c[n%2][m]. //能否进一步减少存储空间?　**O(min{m,n})**

拆分方案数1

- 输入: 正整数集A[1:k], 正整数n
- 输出: 将n拆分为A中不同数的和的方案数
- 使用什么子结构, 决策(量)?
- [1:i], f[i,s] = s用A[1:i]中不同数拆分的方案数
- 输出什么?
- f[k,n]. f[i,s] = ? 递推关系

f(i,k)	A[i]	k=0	1	2	3	4	5	6	7	8	9	10	11
f(1,k)	1	1	1	0	0	0	0	0	0	0	0	0	0
f(2,k)	2	1	1	1	1	0	0	0	0	0	0	0	0
f(3,k)	3	1	1	1	2	1	1	1	0	0	0	0	0
f(4,k)	4	1	1	1	2	2	2	2	2	1	1	1	0

拆分方案数1

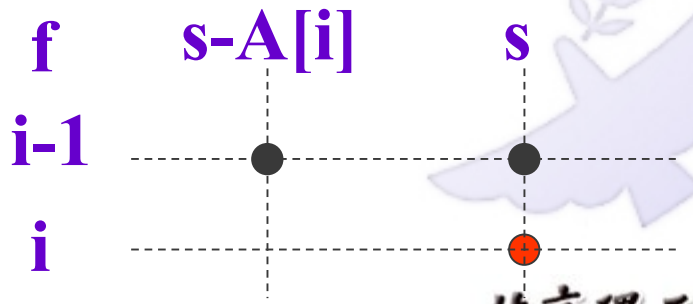
- 输入: 正整数集A[1:k], 正整数n
- 输出: 将n拆分为A中不同数的和的方案数
- [1:i], f[i,s] = s用A[1:i]中不同数拆分的方案数
- 输出f[k,n]. f[i,s] = ?
- 添加A[i]会产生什么变化?
- 添加A[i]后, 拆分中有含和不含A[i]两种情况
- f[i , s] = f[i-1 , s] + f[i-1 , s - A[i]]
- 为计算 f[k,n], 需要计算f[i,s], 0≤i≤k, 0≤s≤n

$$f[i,s]=\begin{cases}1 & i=0,s=0\\ 0 & (i=0,s>0)\text{或}(s<0)\\ f[i-1,s]+f[i-1,s-A[i]] & i>0\end{cases}$$

拆分方案数1示例

$$f[i,s]=\begin{cases}1 & i=0,s=0\\ 0 & (i=0,s>0)\text{或}(s<0)\\ f[i-1,s]+f[i-1,s-A[i]] & i>0\end{cases}$$

f(i,k)	A[i]	k=0	1	2	3	4	5	6	7	8	9	10	11
f(1,k)	1	1	1	0	0	0	0	0	0	0	0	0	0
f(2,k)	2	1	1	1	1	0	0	0	0	0	0	0	0
f(3,k)	3	1	1	1	2	1	1	1	0	0	0	0	0
f(4,k)	4	1	1	1	2	2	2	2	2	1	1	1	0



拆分方案数1算法

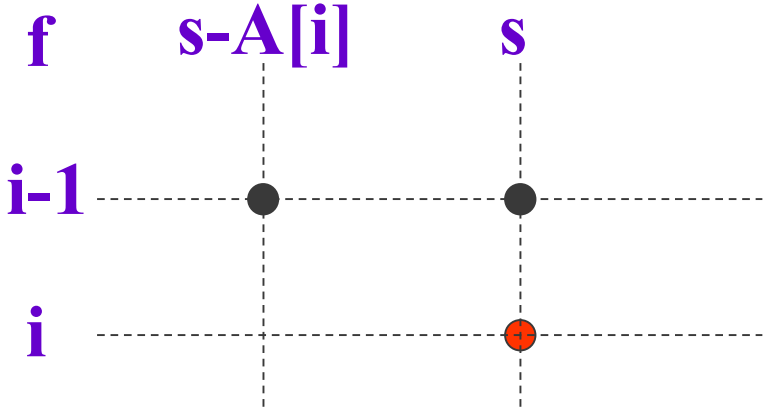
$$f[i,s]=\begin{cases}1 & i=0,s=0\\ 0 & (i=0,s>0)或(s<0)\\ f[i-1,s]+f[i-1,s-A[i]] & i>0\end{cases}$$

1. 初始 f[0,1:n] = 0, f[0,0] = 1
2. 对 i = 1:k,
3. 　　对 s = 0:n,
4. 　　f[i,s] = f[i-1,s];
5. 　　若s ≥ A[i], f[i,s] += f[i-1,s-A[i]]
6. 输出f[k,n]

时间O(kn)　对比输入规模

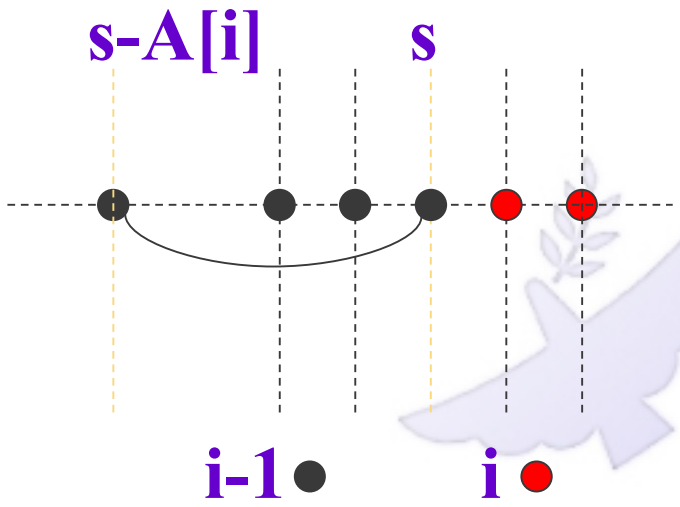
拆分方案数1二维数组改一维

1. 初始 f[0,1:n] = 0, f[0,0] = 1
2. 对 i = 1:k, 对 s = 0:n,
3. f[i,s] = f[i-1,s];
4. 若s ≥ A[i], f[i,s] += f[i-1,s-A[i]]
5. 输出f[k,n]



1. 初始 f[1:n] = 0, f[0] = 1
2. 对 i = 1:k,
3. 对 s = n:1,
4. 若s ≥ A[i], f[s] += f[s-A[i]]
5. 输出f[n]

思考: 第3行改为 s = 1:n会如何?



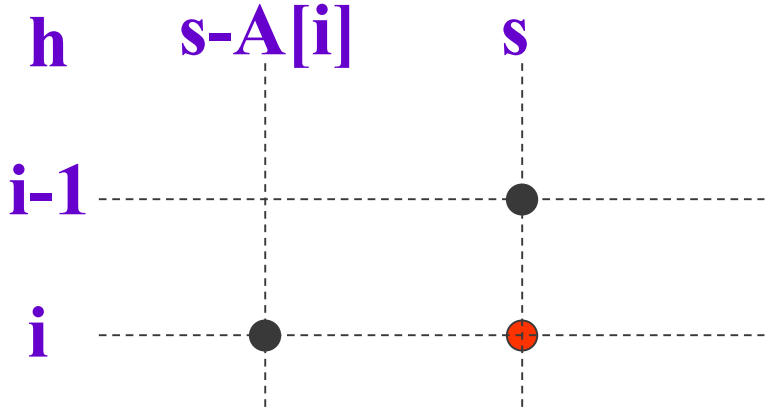
拆分方案数2

- 输入: 正整数集A[1:k], 正整数n
- 输出: 将n拆分为A中数(可用多次)的和的方案数
- 子结构? 决策量?
- [1:i], h[i,s] = s用A[1:i]中数拆分的方案数
- h[i,s] = h[i-1,s] + h[i-1,s-A[i]] + h[i-1,s-2A[i]] + …
- = h[i-1,s] + h[i,s-A[i]]

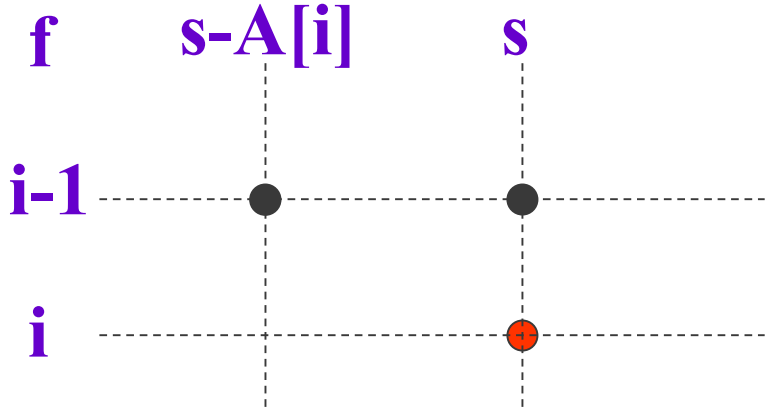
1. 初始h[1:k,0]=0, h[0,0]=1
2. 对i=1:k,
3. 对s=0:n,
4. h[i,s]=h[i-1,s];
5. 若s≥A[i], h[i,s]+=h[i,s-A[i]]
6. 输出h[k,n]

三种计算关系的比较

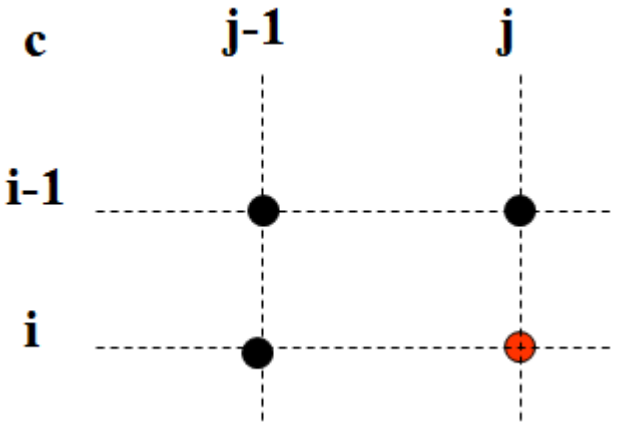
- h[i,s]=h[i-1,s]+h[i, s-A[i]]



- f[i,s]=f[i-1,s]+f[i-1, s-A[i]]



$$c[i][j]=\begin{cases}0 & i=0\text{或}j=0\\ c[i-1][j-1]+1 & i,j>0\text{且}x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0\text{且}x_i\neq y_j\end{cases}$$

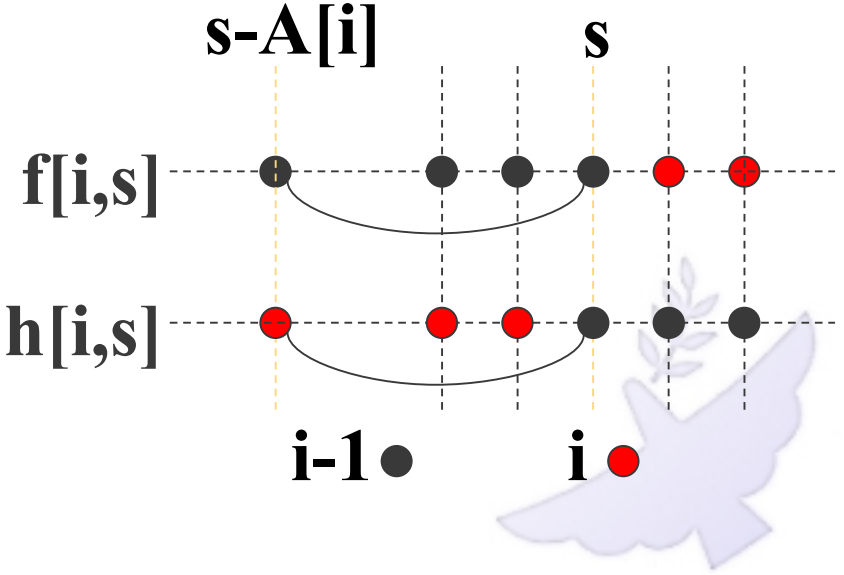


对比拆分1和拆分2的二维改一维

拆分1--每数至多用1

1. 初始 f[1:n] = 0, f[0] = 1
2. 对 i = 1:k,
3. 　　对 s = n:1,
4. 　　　　若s ≥ A[i], f[s] += f[s-A[i]]
5. 输出f[n]

- f[i,s]=f[i-1,s]+f[i-1, s-A[i]]
- h[i,s] = h[i-1,s] + h[i, s-A[i]]



拆分2--每数可用0至多次

1. 初始h[1:n]=0, h[0]=1
2. 对 i = 1:k,
3. 　　对 s = 1:n,
4. 　　　　若s≥A[i], h[s]+=h[s-A[i]]
5. 输出h[n]

END