第三章 动态规划
dynamic programming

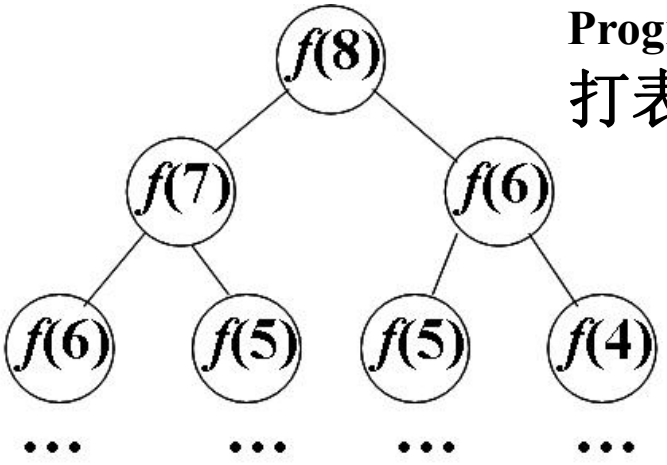
1. 动态规划一般原理(与分治对比), Bellman, OSP
2. 动态规划设计步骤: 矩阵连乘
3. 如何设计动态规划算法: 子结构和策略
   最长公共子序列, 最大子段和, 最长递增子序列
4. 背包问题 (动态规划与贪心对比)
5. 最短路问题

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

DP适用条件

- 最优子结构性质, 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.

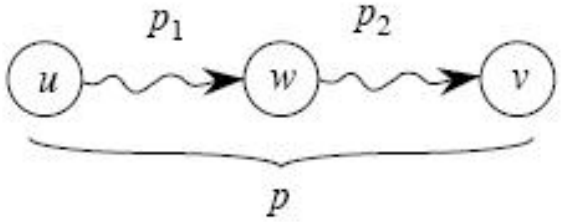
- Bellman，1955，奠定DP数学基础.

•“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.”

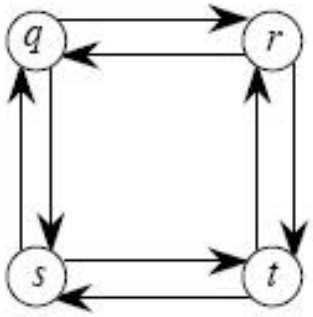
DP适用条件

•无权最短路径: 边数最少. 具有最优子结构性质.



•无权最长路径: 一定简单.

if we decompose a longest simple path into subpaths ,
then mustn't p_1 be a longest simple path from u to w,
and mustn't p_2 be a longest simple path from w to v?
The answer is no!



Consider $q \to r \to t$ = longest path $q \rightsquigarrow t$. Are its subpaths longest paths?

No!

- Subpath $q \rightsquigarrow r$ is $q \to r$.
- Longest simple path $q \rightsquigarrow r$ is $q \to s \to t \to r$.
- Subpath $r \rightsquigarrow t$ is $r \to t$.
- Longest simple path $r \rightsquigarrow t$ is $r \to q \to s \to t$.

除接合点外,不能共享资源.

DP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)
5. 最短路问题

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: (($A_1 \times A_2$) × A_3), (A_1× ($A_2 \times A_3$))

  $A_1 \times A_2$的计算量: 10×100×5 (乘法次数)

  (($A_1 \times A_2$) ×A_3) : 10×100×5 + 10×5×50 = 7500

  (A_1× ($A_2 \times A_3$)) : 100×5×50 + 10×100×50 = 75000
- 样例输出: (($A_1 \times A_2$) ×A_3)

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:

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

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

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

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最优, 则P在A[1:k]和A[k+1:n]上也最优

  最优子结构性质：最优策略的子策略也是最优.

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

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

- 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
- 最优解是要输出的次序

  最优值是最优解的计算量

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

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

- 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

计算最优值图示

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

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						

计算最优值图示

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

(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

计算最优值图示

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

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

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

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

计算最优值图示

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

$$\begin{aligned}m[1,4] &= \min\{m[1,1]+m[2,4]+q_0\times q_1\times q_4,\\ &\qquad m[1,2]+m[3,4]+q_0\times q_2\times q_4,\\ &\qquad m[1,3]+m[4,4]+q_0\times q_3\times q_4\}\\ &= \min\{0+4375+30\times35\times10,\\ &\qquad 15750+750+30\times15\times10,\\ &\qquad 7875+0+30\times5\times10\}\\ &= \min\{9375, 21000, 14875\}\end{aligned}$$

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

计算最优值图示

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

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 }

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	9375	11875	15125
	2		0	2625	4375	7125	10500
	3			0	750	2500	5375
	4				0	1000	3500
	5					0	5000
	6						0

计算最优值算法

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

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?

计算最优值同时标记分断点

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

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

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

$A_{1,1} \times A_{2,2}$

$A_{1,2} \times A_{3,3}$

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

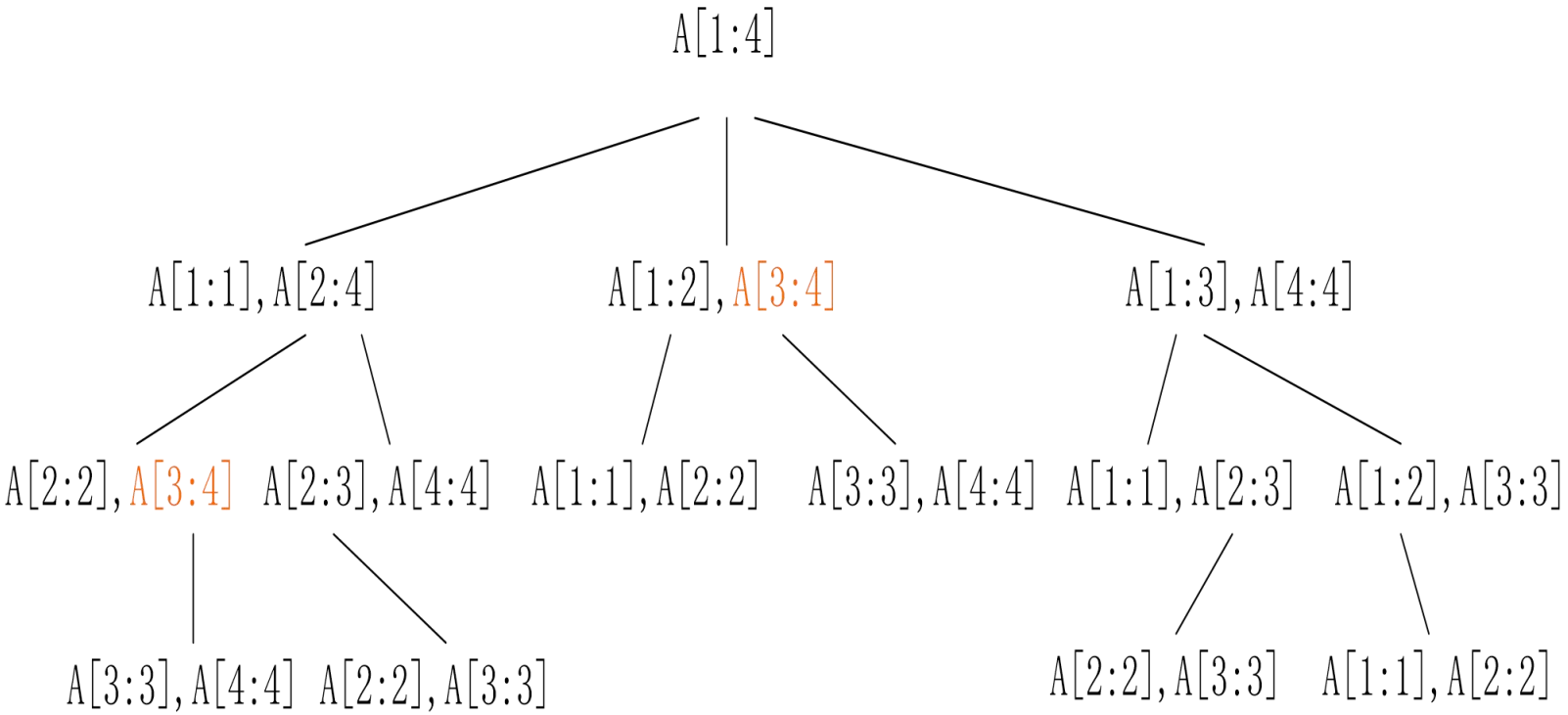
3) 自底向上计算最优值 → 自顶向下

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

  2. LU(1,n)　　//LookUp

- **LU(i,j)**

  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) + $q_{i-1}\times q_i\times q_j$;

  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

•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。5, 3, 4, 2, 3, 4, 3, 4　为了简化，只算4, 2, 3, 4

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				

练习

•在一个圆形操场的四周顺序存放着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。5, 3, 4, 2, 3, 4, 3, 4　为了简化，只算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)
5. 最短路问题

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最长的公共子序列(LCS)
- 样例: X=ACTLE, Y=WARE, 输出: AE
- 若 $x_i = y_j$, 则
  (X_i,Y_j)有LCS含(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.

例如:

ISIST

BIT

最长公共子序列OSP

设序列$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$的最长公共子序列为$Z=\{z_1,z_2,\ldots,z_k\}$，则

(1)若$x_m=y_n$，则$z_k=x_m=y_n$，且z_{k-1}是x_{m-1}和y_{n-1}的最长公共子序列。

(2)若$x_m\neq y_n$且$z_k\neq x_m$，则Z是x_{m-1}和Y的最长公共子序列。

(3)若$x_m\neq y_n$且$z_k\neq y_n$，则Z是X和y_{n-1}的最长公共子序列。

由此可见，2个序列的最长公共子序列包含了这2个序列的前缀的最长公共子序列。因此，最长公共子序列问题具有最优子结构性质。

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

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

```
LCSlength(n,m,x,y,c)
1. 初值 c[0][1: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].
```

LC: 构造最优解

```
LCS( )
    …
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;}
```

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

最大子段和

- 给定整数序列$a_1,a_2,\ldots,a_n$，求形如$\sum_{k=i}^{j} a_k$的子段和的最大值。规定子段和为负整数时，定义其最大子段和为0，即

$$max\left\{0,\quad \max_{1\le i\le j\le n}\sum_{k=i}^{j} a_k\right\}$$

- 例如，$(a_1,a_2,a_3,a_4,a_5,a_6)=(-2,11,-4,13,-5,-2)$
  最大子段和为

$$\sum_{k=2}^{4} a_k=20$$

最大子段和-动规

```
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;  例: (-1, 5, -3, 6, 4, 2, -7, 8)
     }
     return sum;
}
```

$$b[j]=\max\{b[j-1]+a_j, 0\} \quad 1\le j\le n$$

运行时间：T(n)=O(n)

引例

- 某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够达到任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

引例

- ***D*[*i*]:表示第*i*枚导弹被拦截以后，还能最多拦截的导弹数(包括被拦截的第*i*枚), 则**

$$D[i]=\begin{cases} 1 & i=n \quad or \quad (i<j\le n \text{ and } x(j)>x(i)) \\ \max\{D[j]+1\} & i<n \quad (i<j\le n \text{ and } x(j)\le x(i)) \end{cases}$$

  - 例如，60，40，45，70，42

i	5	4	3	2	1
D[*i*]	1	2	2	1	3

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: (2,8,9,4,6,1,3,7,5,10)
- 输出样例: (2,4,6,7,10)
- 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: 需要记录最长和次长的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
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
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($x_1,x_2,\ldots,x_n$)
- 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)						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
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
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?
$O(n^3)$?
只记录尾巴?
$O(n^2)$?

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)						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($x_1,x_2,\ldots,x_n$)
- 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					

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($x_1,x_2,\ldots,x_n$)
- 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)						1				
MTIS(2)						1	3			
MTIS(3)	2			4	6					
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
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										

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								
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
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
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
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
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
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
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
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

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,…,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: MTIS(1).last ≤ MTIS(2).last ≤ …
- 证明: 若MTIS(i).last > MTIS(i+1).last, 矛盾.
- x_{i+1}改变MTIS(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_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.

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)
5. 最短路问题

0-1背包问题([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.
- 例: W:{2,2,6,5,4}

  V:{6,3,5,4,6}

  C=10

0-1背包问题OSP

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

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

- 证明：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。
- 若不然，设$(z_2,\ldots,z_n)$是Knap(2,n,c-w_1y_1)的最优解，因此有

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

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

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

0-1背包问题及(分数)背包OSP

此问题的形式化描述是，给定 $c>0, w_i>0, v_i>0, 1\leqslant i\leqslant n$，要求找出一个 n 元向量 $(x_1, x_2, \cdots, x_n), 0\leqslant x_i\leqslant 1, 1\leqslant i\leqslant n$，使得 $\sum_{i=1}^{n} w_i x_i\leqslant c$，而且 $\sum_{i=1}^{n} v_i x_i$ 达到最大。

这两类问题都具有最优子结构性质。对于0-1背包问题，设 A 是能够装入容量为 c 的背包的具有最大价值的物品集合，则 $A_j = A - \{j\}$ 是 $n-1$ 个物品 $1,2,\cdots,j-1,j+1,\cdots,n$ 可装入容量为 $c-w_j$ 的背包的具有最大价值的物品集合。对于背包问题，类似地，若它的一个最优解包含物品 j，则从该最优解中拿出所含的物品 j 的那部分重量 w，剩余的将是 $n-1$ 个原重物品 $1,2,\cdots,j-1,j+1,\cdots,n$ 以及重为 w_j-w 的物品 j 中可装入容量为 $c-w$ 的背包且具有最大价值的物品。

例：有5个物体，重量2,2,6,5,4，价值6,3,5,4,6，背包的载重量为10，求装入背包的物体及其总价值

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
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***

$$dp_i(k) = \max\{dp_{i-1}(k),\ dp_{i-1}(k-w_i)+ v_i\}$$

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], dp[i,k] = 由[1:i]组合出重量≤k的最大价值
- dp[i,k] = max{ dp[i-1,k], dp[i-1,k-W[i]] + V[i] } (OSP)
- 最优值: dp[n,C]

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

0-1背包问题推广 找零钱问题

一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，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.

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。

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

不用第i种硬币

用第i种硬币

$$p_i(0)=0$$

$$p_0(j)=\infty$$

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

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

(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

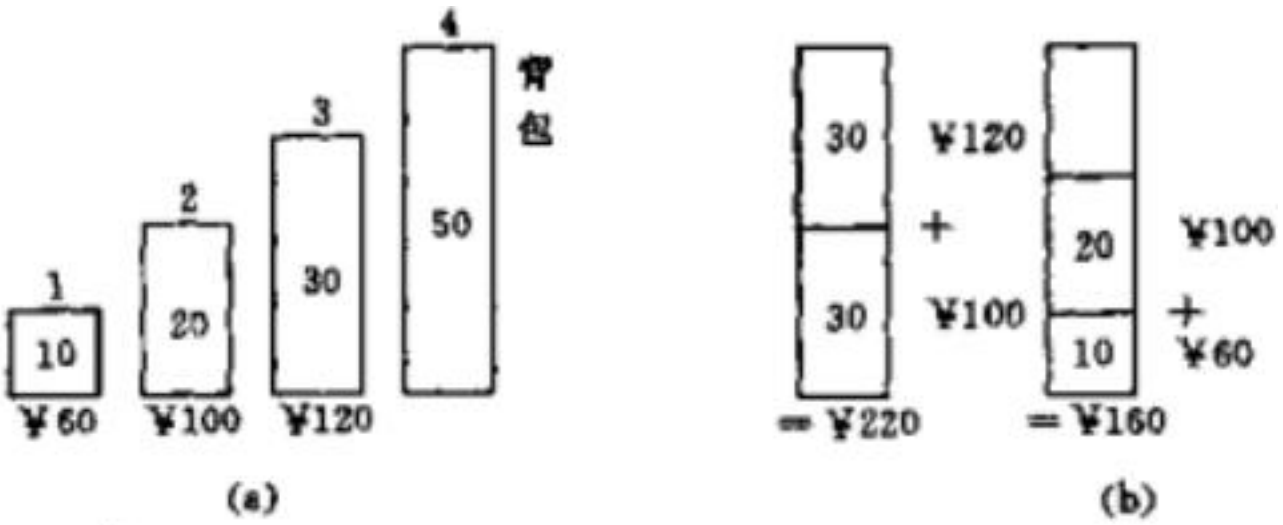
2. 优先放入单位重量价值大的物品, 直到填满

O(nlogn),

0-1背包能贪心吗?

(分数)背包问题

0-1背包能贪心吗?



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],
  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

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

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

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

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

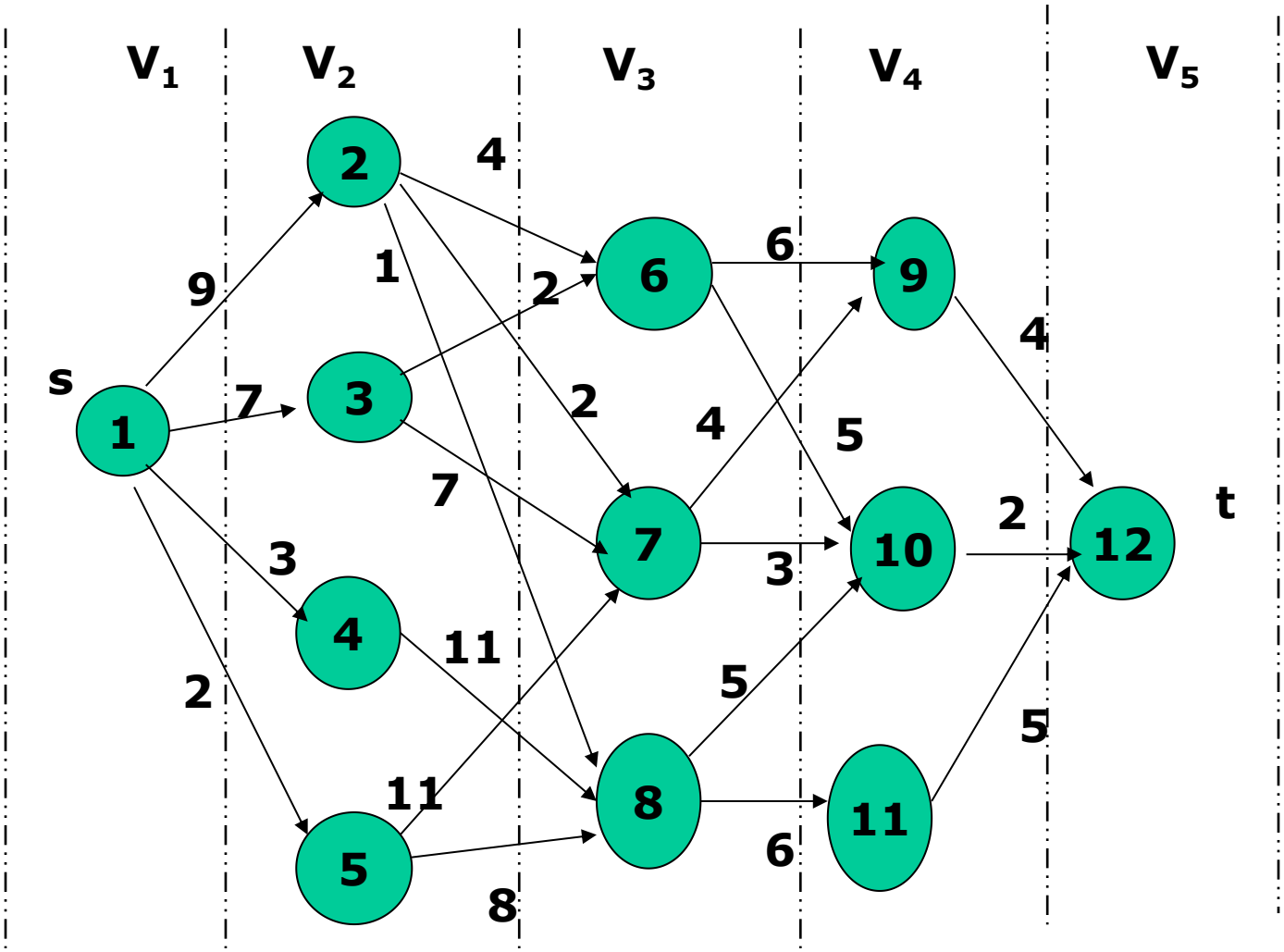
第三章 动态规划
dynamic programming

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)
5. 最短路问题

多段图的最短路问题

多段图：设$G=<V,E>$是一个有向连通图，其中$|V|=n$, $|E|=m$, V有划分$\{V_1,V_2,\cdots,V_k\}$，这里$V_1=\{s\}$,s称为源点，$V_k=\{t\}$,t称为终点，其中$k\geq 2$。对于每条有向边$<u,v>\in E$都存在$V_i\in V$，使得$u\in V_i$和$v\in V_{i+1}$, 其中$1\leq i<k$且每条边$<u,v>$均附有代价C(u,v)，则称G是一个k段图。

多段图的最短路问题


V1
V2
V3
V4
V5
s
1
2
3
4
5
6
7
8
9
10
11
12
t
9
7
2
3
4
1
4
2
7
2
4
6
5
3
11
11
8
5
6
2
5
5

多段图的最短路问题

- 假设$s,v_2,v_3,\cdots,v_{k-1},t$是一条从s到t的最短路径，还假定从源点s（初始状态）开始，已做出了到结点v_2的决策（初始决策），因此v_2就是初始决策所产生的状态。如果把v_2看成是原问题的一个子问题的初始状态，解这个子问题就是找出一条由v_2到t的最短路径。这条路径显然是 $v_2,v_3,\cdots,v_{k-1},t$，否则设$v_2,q_3,\cdots,q_{k-1},t$是一条由v_2到t的更短路径，则$s,v_2,q_3,\cdots,q_{k-1},t$是一条比路径 $s,v_2,v_3,\cdots,v_{k-1},t$ 更短的由s到t的路径，与假设矛盾，故最优化原理成立。

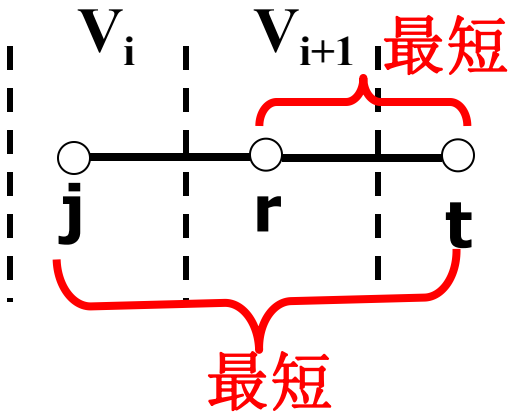
多段图的最短路问题

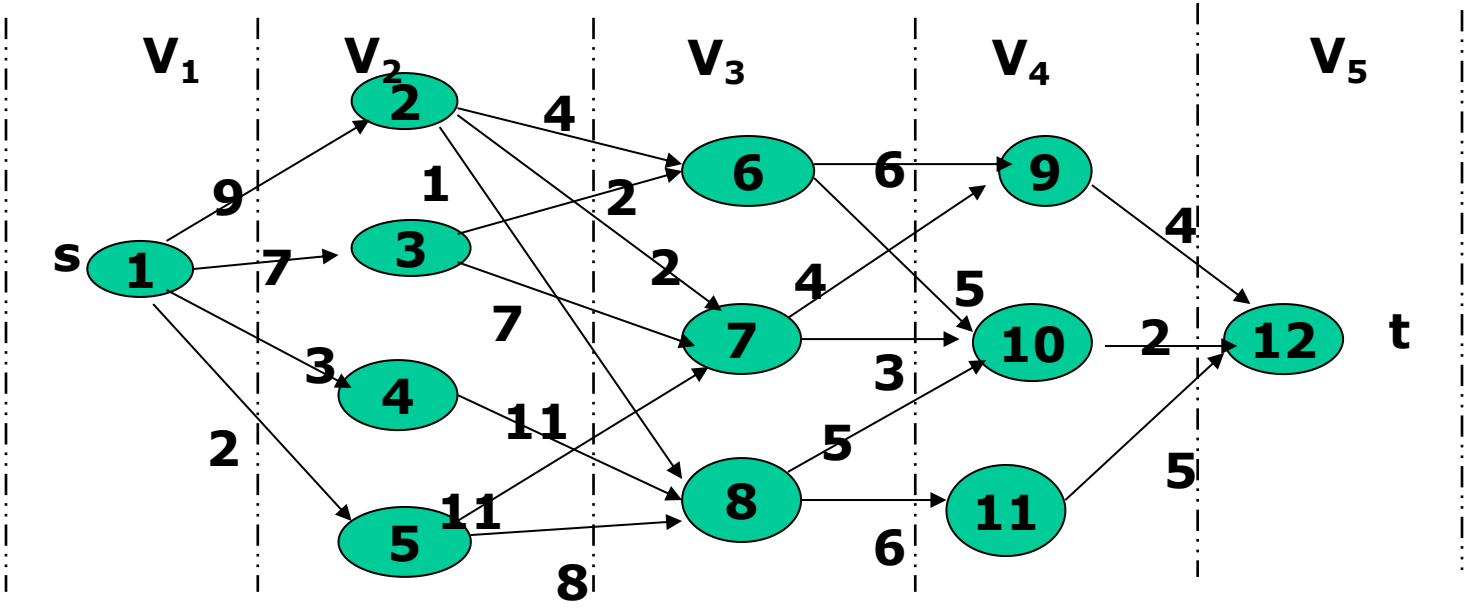
向前处理法：设P(i,j)是从V_i中的点j到t的一条最短路，cost(i,j)是这条路线的耗费。由后向前推算，得到

$$cost(i,j)=\min\{C(j,r)+cost(i+1,r)\}$$

$r \in V_{i+1}$

$<j,r> \in E$





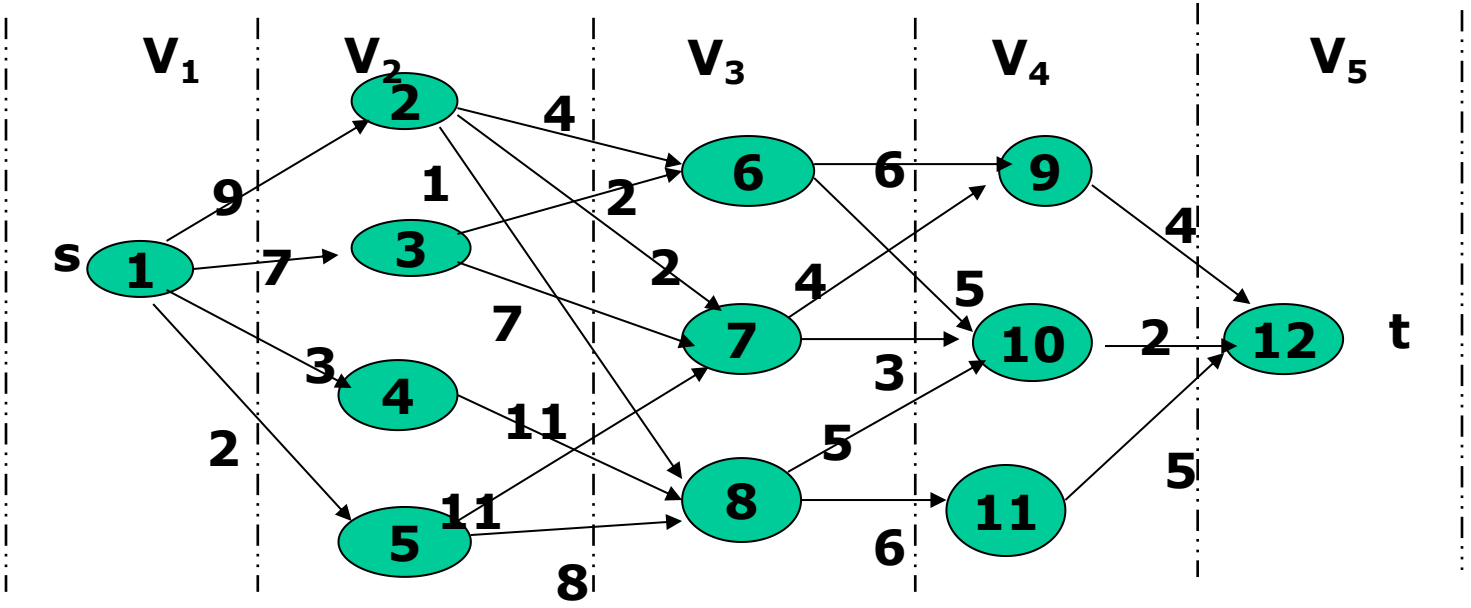
cost(i,j)=min{C(j,r)+cost(i+1,r)}

cost(4,9)=4　　cost(4,10) =2　cost(4,11)=5

cost(3,6)=　min{6+cost(4,9),5+cost(4,10)}

=min{6+4,5+2}=7

从第3段的顶点6到t的最短路径是6-10-t

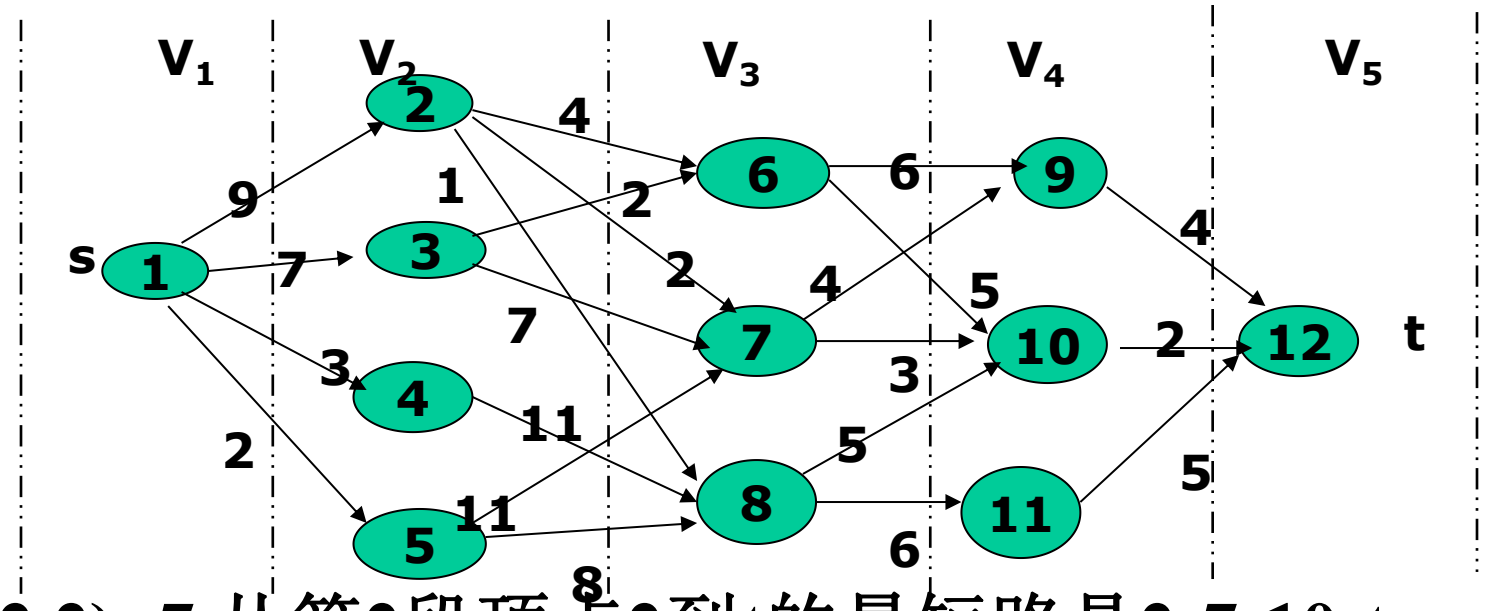


cost(3,7)= min{4+cost(4,9),3+cost(4,10)}
=min{4+4,3+2}=5

- **从第3段的顶点7到t的最短路径是7-10-t**

cost(3,8)=7

- **从第3段的顶点8到t的最短路径是8-10-t**



cost(2,2)=7 从第2段顶点2到t的最短路是2-7-10-t

cost(2,3)=9从第2段顶点3到t的最短路是3-6-10-t

cost(2,4)=18从第2段顶点4到t的最短路是4-8-10-t

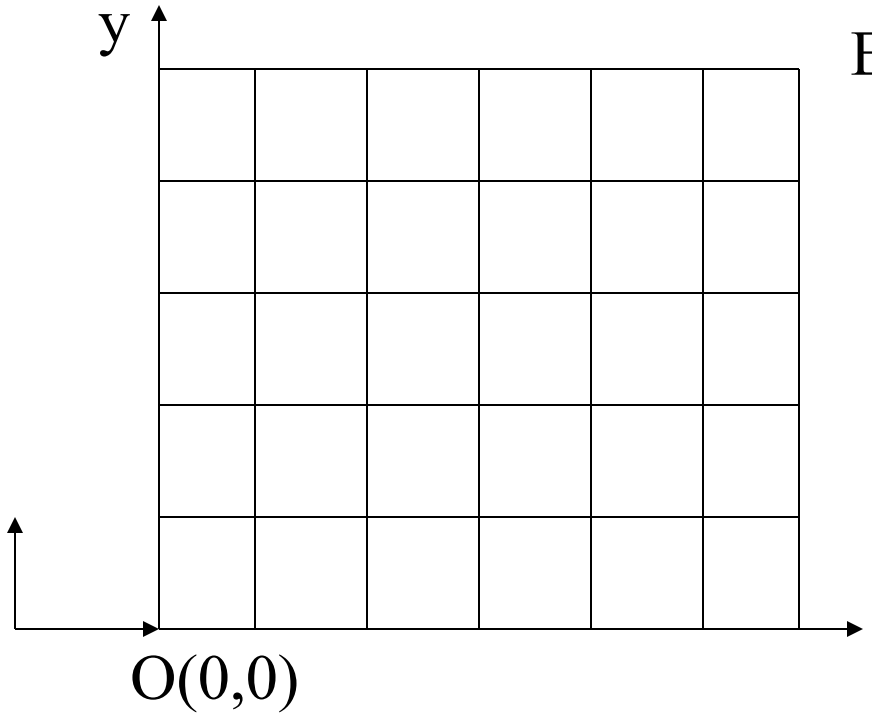
cost(2,5)=15从第2段顶点5到t的最短路是5-8-10-t

cost(1,1)=16 从第1段顶点1到t的最短路径由两条：

1-2-7-10-t和1-3-6-10-t

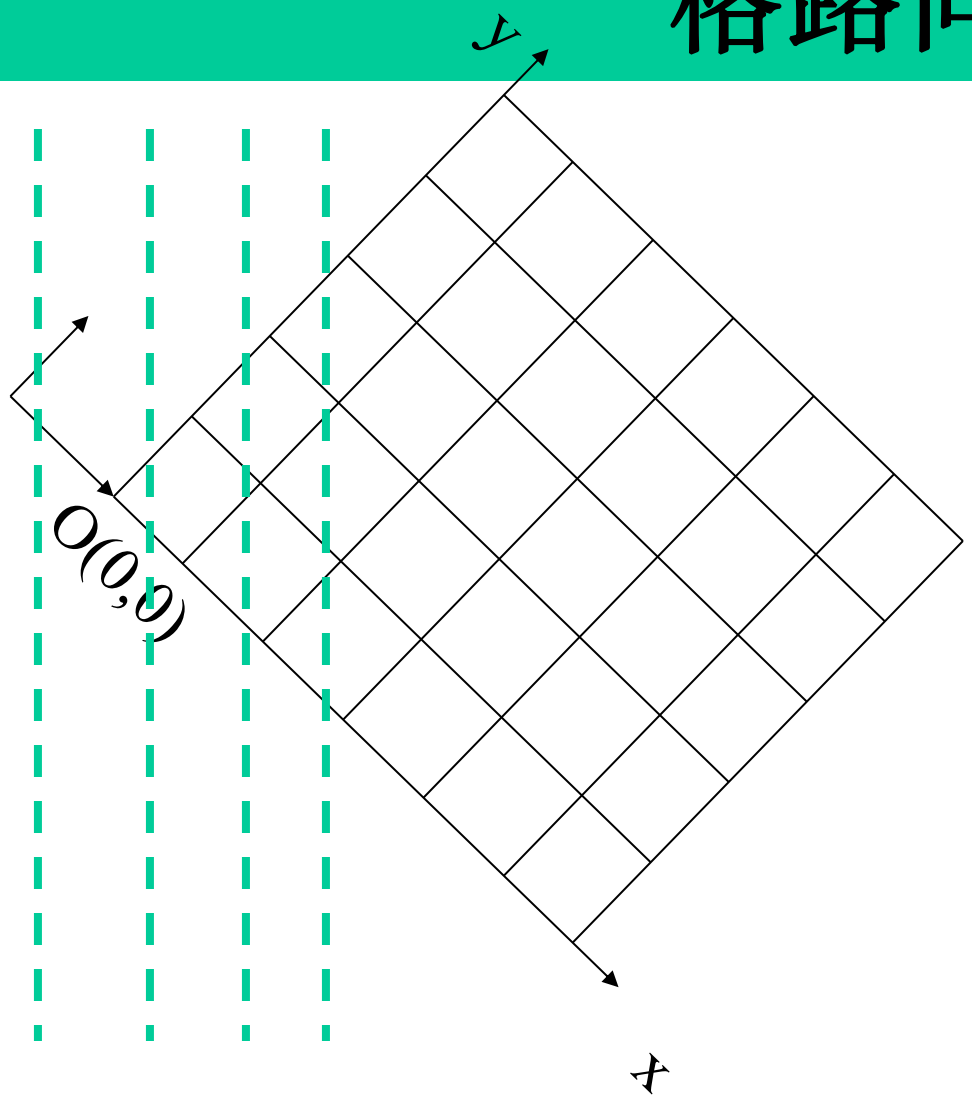
格路问题

- **：求从起点O(0,0)到终点E(m,n)的最短路。则分别用穷举法和动态规划法的加法次数和比较次数各是多少？**



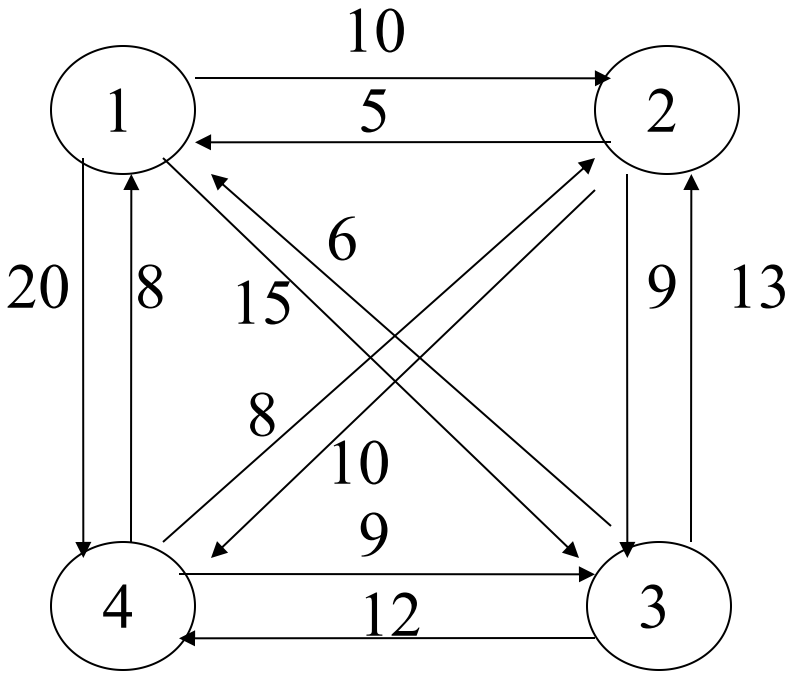
加法次数	比较次数

格路问题



点数	加法次数	比较次数

货郎担问题



	v_1	v_2	v_3	v_4
v_1	∞	**10**	**15**	**20**
v_2	**5**	∞	**9**	**10**
v_3	**6**	**13**	∞	**12**
v_4	**8**	**8**	**9**	∞

货郎担问题

- 不失一般性，考虑以结点1为起点和终点的一条哈密顿回路。每一条这样的路线都由一条边<1,k>和一条由结点k到结点1的路径组成，其中k∈V-{1},而这条由结点k到结点1的路径通过V-{1，k}的每个结点各一次。如果这条周游路线是最优的，则这条由结点k到结点1的路径必定是通过V-{1，k}的每个结点各一次的由k到1的最短路径。

货郎担问题

- 设T(i,S)是由结点 i出发，经过结点集S中每个结点各一次并回到初始结点1的一条最短路径长度，则T(1,V-{1})就是一条最优的周游路线长度。动态规划模型：

$$T(1,V-\{1\})=\min\{d_{1k}+T(k,V-\{1,k\})\}$$
$$2 \le k \le n$$

$$T(i,s)=\min\{d_{ij}+T(j,S-\{j\})\}$$
$$j \in S, i \notin S$$

货郎担问题

	v_1	v_2	v_3	v_4
v_1	∞	**10**	**15**	**20**
v_2	**5**	∞	**9**	**10**
v_3	**6**	**13**	∞	**12**
v_4	**8**	**8**	**9**	∞

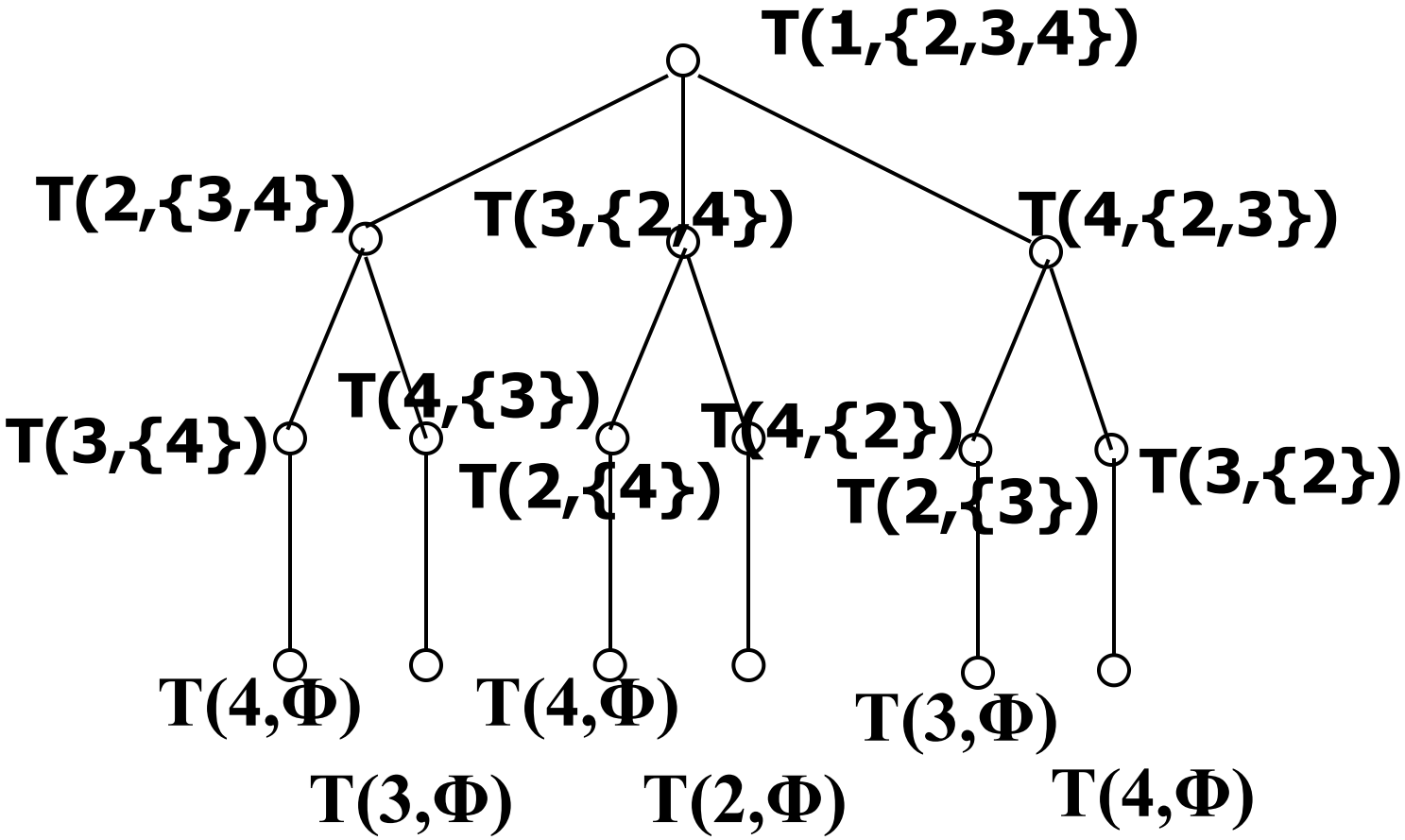
- **$T(1, \{2,3,4\})=\min\{d_{12}+T(2,\{3,4\}),\ d_{13}+T(3,\{2,4\}),\ d_{14}+T(4,\{2,3\})\}$**

$T(2,\{3,4\}) =\min\{\ d_{23}+T(3,\{4\}),\ d_{24}+T(4,\{3\})\}$

$T(3,\{4\}) = d_{34}+\ T(4,\Phi)$

$T(4,\Phi)=\ d_{41}$

货郎担问题


T(1,{2,3,4})
T(2,{3,4})
T(3,{2,4})
T(4,{2,3})
T(3,{4})
T(4,{3})
T(2,{4})
T(4,{2})
T(2,{3})
T(3,{2})
T(4,Φ)
T(3,Φ)
T(4,Φ)
T(2,Φ)
T(3,Φ)
T(4,Φ)

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

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
```

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

最短路

习题