3.7 图像压缩

•数字化图象：

黑白（单色0，1）图象

m x m像素阵列

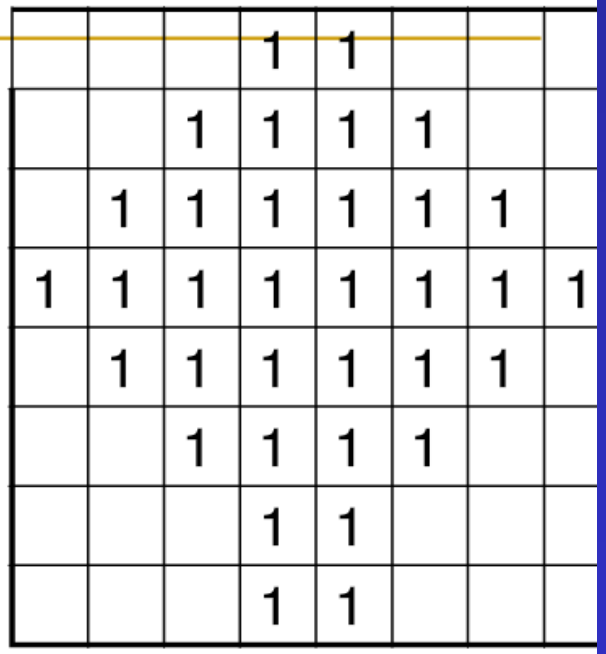
存储空间m^2

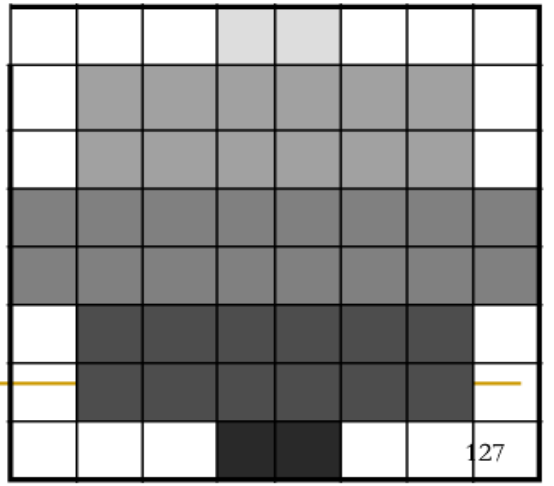
灰度图象(0~255灰度值)

8xmxm像素阵列

存储空间$8m^2$

颜色数为2^8 ＝256种





3.7 图像压缩

彩色图象：

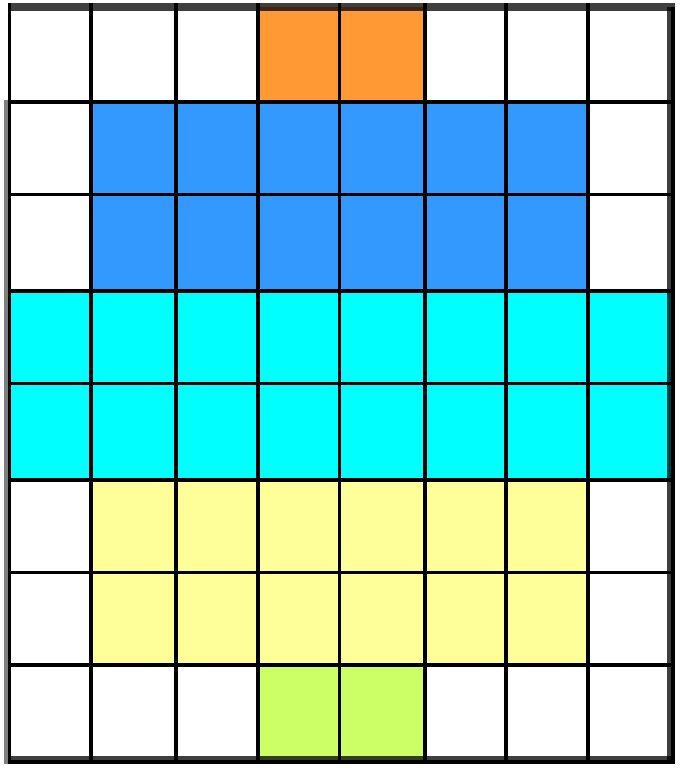
一幅彩色图像的每个像素用R，G，B三个分量表示，若每个分量用8位，那末一个像素共用24位表示。

颜色数为2^{24} ＝16 777 216种

存储空间24m^2

用RGB 5：5：5表示 的彩色图像
颜色数为2^{15} ＝ 32K

存储空间15m^2



3.7 图像压缩

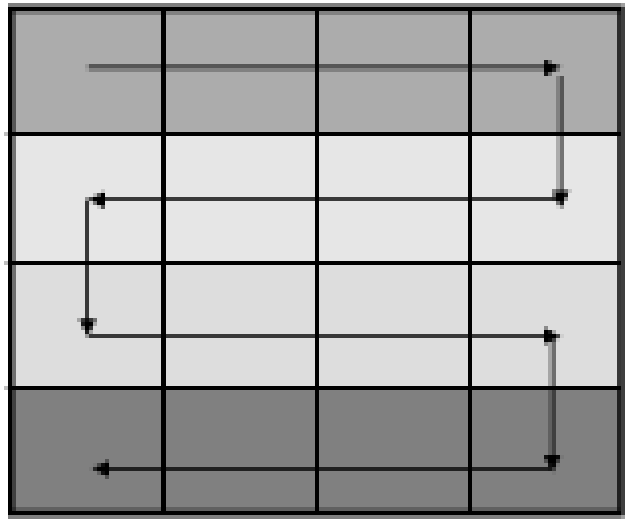
减少存储空间，采用变长模式，不同像素用不同位数来存储。

像素值	0,1	2,3	4,5,6,7	8~15	16~31	32~63	64~127	128~255
所需位数	1	2	3	4	5	6	7	8

3.7 图像压缩

使用变长模式的步骤：

1. 图象线性化： $mxm \rightarrow 1xm^2$



10	9	12	40
12	15	35	50
8	10	9	15
240	160	130	11

3.7 图像压缩

使用变长模式的步骤：

2. 分段，分段规则：

每段中的像素位数相同；

每个段是相邻像素的集合；

每个段最多含256个像素，若相邻的相同位数的像素超过256个，则用2个以上的段表示。

3.7 图像压缩

使用变长模式的步骤：

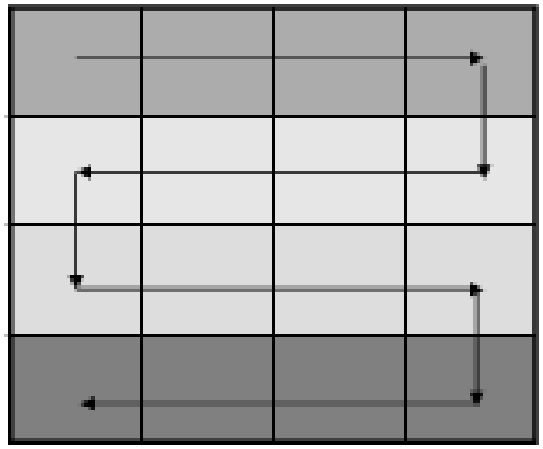
3. 创建文件，创建3个文件：

SegmentLength: 包含步骤2中所建的段的长度(减1)，文件中各项均为8位长。

BitsPerPixels: 包括各段中每个像素的存储位数(减1)，文件中各项均为3位长。

Pixels: 以变长格式存储的像素的二进制串。

4 压缩文件，压缩步骤3所建立的文件，以节约空间。



10	9	12	40
12	15	35	50
8	10	9	15
240	160	130	11

例子：图中4x4图象

灰度值:

{10,9,12,40,50,35,15,12,8,10,9,15,11,130,160,240}

所需位数:

{ 4,4,4, 6,6,6, 4,4,4,4,4,4,4,4, 8,8,8 }

分段:

[10,9,12], [40,50,35], [15,12,8,10,9,15,11], [130,160,240]

像素值	0,1	2,3	4,5,6,7	8~15	16~31	32~63	64~127	128~255
所需位数	1	2	3	4	5	6	7	8

例：{6, 5, 7,5, 245, 180, 28,28,19, 22, 25,20}　12*8=96位

存储位数B[i]　3　8　5

像素个数L[i]　4　2　6

像素值	0,1	2,3	4,5,6,7	8~15	16~31	32~63	64~127	128~255
所需位数	1	2	3	4	5	6	7	8

4*3+2*8+6*5=58

4*3+2*8+6*5 +3*3 +3*8 =91

$l[i](1 \le l[i] \le 255)$

3 *（3+8）

=3 *11

m个段

用来记录段长度的8位+用来记录像素存储位数的3位

记录所建的每个段的长度，各项为8位长

记录各段中每个像素的存储位数，各项3位长

3.7 图像压缩

将像素分成连续的m段$S_1, S_2, \cdots, S_m$，使每段中的像素存储位数相同。第$i(1 \le i \le m)$个像素段S_i中有$l[i]$个像素，且该段中每个像素都只用$b[i]$位表示。

$$\sum_{i=1}^{m} l[i] * b[i] + 11m$$

要求找到一个最优分段，使存储空间最少。

3.7 图像压缩

例1:

$P=\{10, 9, 12, 40, 50, 35, 15, 12, 8, 10, 9, 15, 11, 130, 160, 240\}$

S_1 S_2 S_3 S_4

$l[1]=3$ $b[1]=4$

$l[2]=3$ $b[2]=6$

$l[3]=7$ $b[3]=4$

$l[4]=3$ $b[4]=8$

$$3\times4+3\times6+7\times4+3\times8+11\times4=126$$

定长: $16\times8=128$

像素值	0,1	2,3	4,5,6,7	8~15	16~31	32~63	64~127	128~255
所需位数	1	2	3	4	5	6	7	8

3.7 图像压缩

像素值	0,1	2,3	4,5,6,7	8~15	16~31	32~63	64~127	128~255
所需位数	1	2	3	4	5	6	7	8

例1:

$$P=\{10,9,12,40,50,35,15,12,8,10,9,15,11,130,160,240\}$$

S_1 S_2 S_3 S_4

$l[1]=3$ $b[1]=4$

$l[2]=3$ $b[2]=6$

$l[3]=7$ $b[3]=4$

$l[4]=3$ $b[4]=8$

$$3\times4+3\times6+7\times4+3\times8+11\times4=126$$

定长: $16\times8=128$

$$P=\{10,9,12,40,50,35,15,12,8,10,9,15,11,130,160,240\}$$

6 7 3

6 4 8

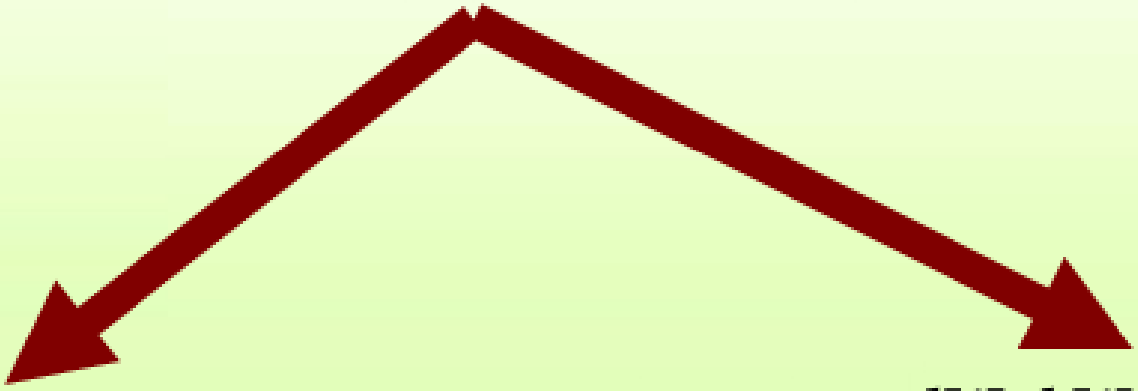
少了5

$$6\times6+7\times4+3\times8+11\times3=121$$

3.7 图像压缩

图像压缩-最优子结构性质

设 $l[i],b[i],1 \le i \le m$ 是 $\{p_1, p_2, \cdots, p_n\}$ 的一个最优分段。



$l[1],b[1]$是$\{p_1, \cdots, p_{l[1]}\}$

的一个最优分段。

$l[i],b[i],2 \le i \le m$是

$\{p_{l[1]+1}, \cdots, p_n\}$

的一个最优分段。

3.7 图像压缩

图像压缩-递归计算最优值

设 $s[i](1 \le i \le n)$ 是像素序列 $\{p_1, p_2, \cdots, p_i\}$ 的最优分段所需的存储位数。

$$s[0] = 0$$

$i = 1$　　p_1　　$s[1] = s[0] + b[1] + 11$

$i = 2$　　p_1 | p_2

$$s[2] = \min\begin{cases} s[0] + 2 \times \max\{b[1], b[2]\} + 11 \\ s[1] + b[2] + 11 \end{cases}$$

S[2]+2*max{b[1],b[2]}+2*11-1*11

$P = \{10, 9, 12, 40, 50, 35$

S_1　S_2

$l[1] = 3$　$b[1] = 4$

$l[2] = 3$　$b[2] = 6$

$3 \times 4 + 3 \times 6 + 7$

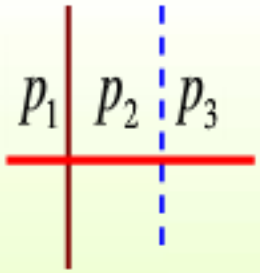
定长: $16 \times 8 = 12$

$P = \{10, 9, 12, 40, 50,$

6

6

3.7 图像压缩

$i=3$ 

$$s[3]=\min\begin{cases} s[0]+3\times\max\{b[1],b[2],b[3]\}+11 \\ s[1]+2\times\max\{b[2],b[3]\}+11 \\ s[2]+b[3]+11 \end{cases}$$

$$=\min_{1\le k\le 3}\{s[i-k]+k\times b\max(i-k+1,i)\}+11$$

$$b\max(i,j)=\left\lceil \log(\max_{i\le k\le j}\{p_k\}+1)\right\rceil$$

S[0]+3*max{b[1],b[2],b[3]}+3*11-2*11
S[1]+2*max{b[2],b[3]}+2*11-1*11
S[2]+b[3]+11

$$s[i]=\min_{1\le k\le \min\{i,256\}}\{s[i-k]+k\times b\max(i-k+1,i)\}+11$$

$$s[i]=\min_{1\le k\le \min\{i,256\}}\{s[i-k]+k\times b\max(i-k+1,i)\}+11$$

3.7 图像压缩

图像压缩-练习

求像素序列**4，6，5，7，129，138，1**的最优分段。

解：$s[0]=0$

$$s[1]=3+11=14 \qquad l[1]=1 \qquad b[1]=3$$

$$s[2]=\min\begin{cases}s[0]+2\times3+11\\ s[1]+3+11\end{cases}=\min\{17,28\}=17 \qquad l[2]=2 \quad b[2]=3$$

$$s[3]=\min\begin{cases}s[0]+3\times3+11\\ s[1]+2\times3+11\\ s[2]+3+11\end{cases}=\min\{20,31,31\}=20 \qquad l[3]=3 \quad b[3]=3$$

像素值	0,1	2,3	4,5,6,7	8~15	16~31	32~63	64~127	128~255
所需位数	1	2	3	4	5	6	7	8

$$s[4]=\min\begin{cases}s[0]+4\times3+11\\ s[1]+3\times3+11\\ s[2]+2\times3+11\\ s[3]+3+11\end{cases}=\min\{23,34,34,34\}=23 \qquad l[4]=4 \quad b[4]=3$$

3.7 图像压缩

$$s[5]=\min\{51,57,52,47,42\}=42 \qquad l[5]=1 \qquad b[5]=8$$

$$s[6]=\min\{59,65,60,55,50,61\}=50 \qquad l[6]=2 \qquad b[6]=8$$

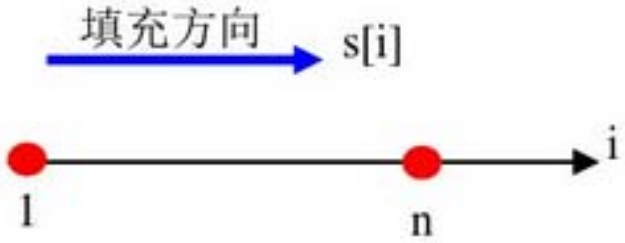
$$s[7]=\min\{67,73,68,63,58,69,62\}=58 \qquad l[7]=3 \qquad b[7]=8$$

$$l=[1\ \ 2\ \ 3\ \ 4\ \ 1\ \ 2\ \ 3]$$

4，6，5，7 | **129，138，1**

实现及复杂性分析

最优值的填充方向如下：



算法实现

图像压缩最优分段问题的动态规划算法如书：P71所示。

计算复杂性

由于算法compress中对k的循环次数不超这256，故对每一个确定的i，可在时间O(1)内完成最优分段的计算。因此整个算法所需的计算时间为O(n)。

目标：设计一种算法，使得在产生 n 个段之后，能对相邻段进行合并，以便产生一个具有最小空间需求的新的段集合。

令:Sq为前q个段的最优合并所需要的空间(位数）；

l_i表示第i段的段长；

b_i表示该段每个像素的长度。

定义：$S_0 = 0$

假设在最优合并C中，第i段与i-1，i-2，...i-r+1段合并，而不包括第i-r段。合并C所需要的空间消耗等于：

第1段到第i-r段所需要的空间+lsum(i-r+1,i)*bmax(i-r+1,i)+11

其中：$lsum(a,b)=\sum l_j$, j=a...b

$bmax(a,b)=max\{b_a, ..., b_b\}$

图像压缩补充部分

1 最优子结构

设S[i]，1≤i≤n，是像素序列{p1,...,pn}的最优分段所需的存储位数。

$S_i=S_{i-r}+lsum(i-r+1,i)*bmax(i-r+1,i)+11$

其中：r=1,...i；lsum<256

寻找r，产生最小的空间需求。

令kay_i 存储取得最小值时k的值。最优子结构如下：

$$s[i]=\min_{\substack{1\le k\le i\\ lsum(i-k+1,i)\le 256}}\{s[i-k]+lsum(i-k+1,i)*\mathrm{b}\max(i-k+1,i)\}+11$$

例子：一副图象的像素组分为5段，
长度为　　[6, 3, 10, 2, 3]
像素位数为[1, 2, 3,　2, 1]

$S_0=0$

$S_1=S_0+l_1*b_1+11=0+6*1+11=17$　　$kay_1=1$

$S_2=\min\{S_1+l_2*b_2\ , S_0+(l_1+\ l_2)*\max\{b_1, b_1\}\}+11$

$=\min\{17+6, 0+9*2\}+11=29$　　$kay_2=2$

……

S[1:5]=[17, 29, 67, 73, 82]

kay[1:5]=[1, 2, 2, 3, 4]

2 迭代计算最优值

```
public static void compress(int[ ] l, int[ ] b, int[ ] s,int[ ] kay)
{
        int n=b.length-1;
        s[0] =0;
        for (int  i=1;i<=n; i++ ){
            int lsum=l[i], bmax=b[i];  //k=1时计算最小值
            s[i]=s[i-1]+lsum*bmax;
            kay[i]=1;
            //lmax=256，对其余的 k 计算最小值并更新
            for (int  k=2;k<=i && lsum+l[i-k+1]<=lmax; k++ ) {
                lsum+=l[i-k+1];
                if( bmax<b[i-k+1] ) bmax=b[i-k+1];
                if( s[i]>s[i-k]+lsum*bmax ) {
                    s[i]=s[i-k]+lsum*bmax;
                    kay[i]=k;   }
                }
                s[i]+=header; //header=11
        }
}
```

2 迭代计算最优值

例子：一副图象的像素组分为5段，
长度为[6, 3, 10, 2, 3]，像素位数为[1, 2, 3, 2, 1]

l[i]

i	0	1	2	3	4	5
	0	6	3	10	2	3

b[i]

i	0	1	2	3	2	1

s[i]

i	0	1	2	3	4	5
	0	0	0	0	0	0

kay[i]

	0	0	0	0	0	0

2 迭代计算最优值

例子：长度为[6, 3, 10, 2, 3]，像素位数为[1, 2, 3, 2, 1]

s[i]

i	0	1	2	3	4	5
	0	17	29	67	73	82

kay[i]

	0	1	2	2	3	4

s[1]=s[0]+lsum*bmax+11=0+6*1+11=17
kay[1]=1

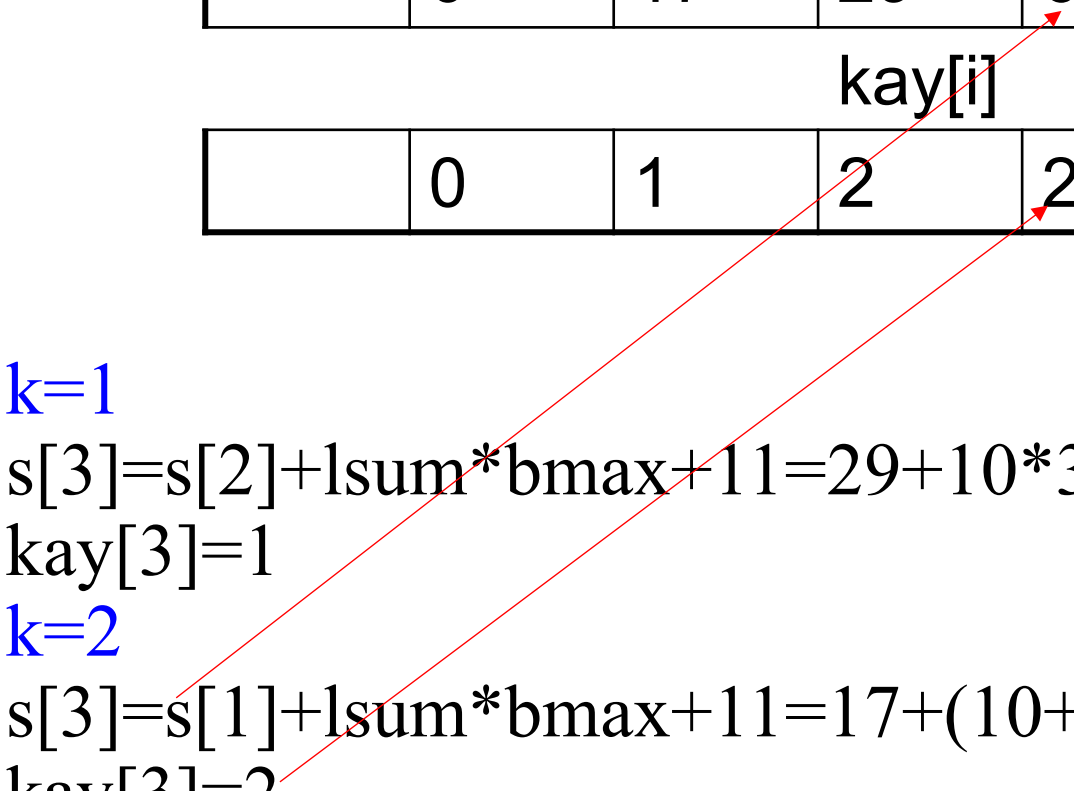
s[2]=s[1]+lsum*bmax+11=17+3*2+11=34 ×
kay[1]=1
s[2]=s[0]+lsum*bmax+11=0+ (3+6)*2+11 =29
kay[1]=2

s[i]

i	0	1	2	3	4	5
	0	17	29	67	73	82

kay[i]

	0	1	2	2	3	4



k=1

s[3]=s[2]+lsum*bmax+11=29+10*3+11=70 ×

kay[3]=1

k=2

s[3]=s[1]+lsum*bmax+11=17+(10+3)*3+11=67

kay[3]=2

k=3

s[3]=s[0]+lsum*bmax+11=0 + (10+3+6)*3+11=68 ×

s[i]

i	0	1	2	3	4	5
	0	17	29	67	73	82

kay[i]

	0	1	2	2	3	4

k=1

s[4]=s[3]+lsum*bmax+11=67+2*2+11=82 ×

kay[4]=1

k=2

s[4]=s[2]+lsum*bmax+11=29+(2+10)*3+11=76 ×

kay[4]=2

k=3

s[4]=s[1]+lsum*bmax+11=17+(2+10+3)*3+11=73

kay[4]=3

k=4

s[4]=s[0]+lsum*bmax+11=0 + (2+10+3+6)*3+11=74 ×

s[i]

i	0	1	2	3	4	5
	0	17	29	67	73	82

kay[i]

	0	1	2	2	3	4

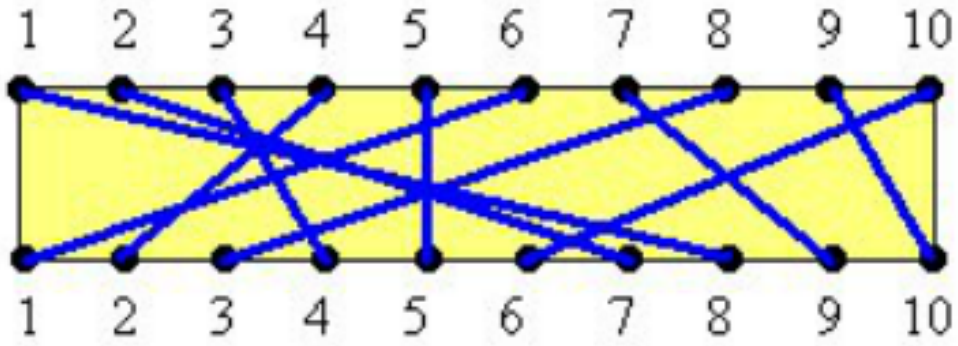
k=1 s[5]=s[4]+lsum*bmax+11=73+3*+1+11=87 ×
kay[5]=1
k=2 s[5]=s[3]+lsum*bmax+11=67+(3+2)*2+11=88×
kay[5]=2
k=3 s[5]=s[2]+lsum*bmax+11=29+(3+2+10)*3+11=85 ×
kay[5]=3
k=4 s[5]=s[1]+lsum*bmax+11=17 + (3+2+10+3)*3+11=82
kay[5]=4
k=5 s[5]=s[0]+lsum*bmax+11=0 + (3+2+10+3+6)*3+11=83×

[6, 3, 10, 2, 3]

3.8电路布线

在一块电路板的上、下2端分别有n个接线柱。根据电路设计，要求用导线(i,π(i))将上端接线柱与下端接线柱相连，如图所示。其中π(i)是{1,2,…,n}的一个排列。导线(i,π(i))称为该电路板上的第i条连线。对于任何1≤i<j≤n，第i条连线和第j条连线相交的充分且必要的条件是π(i)>π(j)。

电路布线问题要确定将哪些连线安排在第一层上，使得该层上有尽可能多的连线。换句话说，该问题要求确定导线集Nets={(i,π(i)),1≤i≤n}的最大不相交子集。

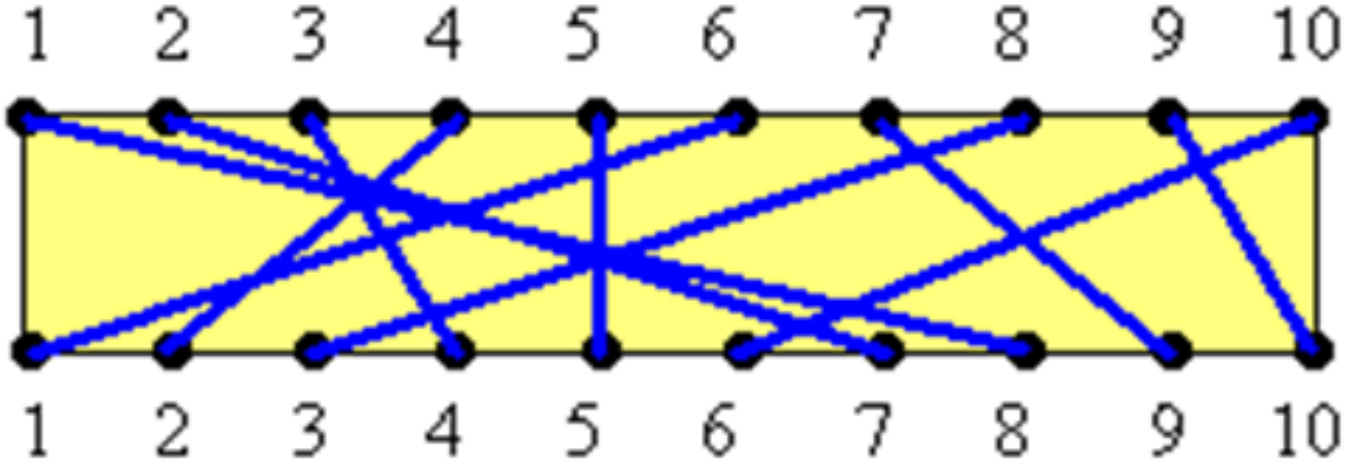


3.8 电路布线

目的：

电路布线问题要确定将哪些连线安排在最优层上，使得该层上有尽可能多的连线。即该问题要求确定导线集 $Nets=\{(i, \pi(i)), 1 \leqslant i \leqslant n\}$ 的最大不相交子集 MNS(Maximum Noncrossing Subset)。

3.8 电路布线



例子: (1,8)和(2,7)交叉，不能布在同一层中；

(1,8), (7,9)和(9,10)未交叉，可以布在同一层中；但还不是MNS。

{(4,2), (5,5),(7,9)和(9,10)} 是一个含4个连线的MNS。

MNS（10，10）

size(10,10)=4

3.8 电路布线

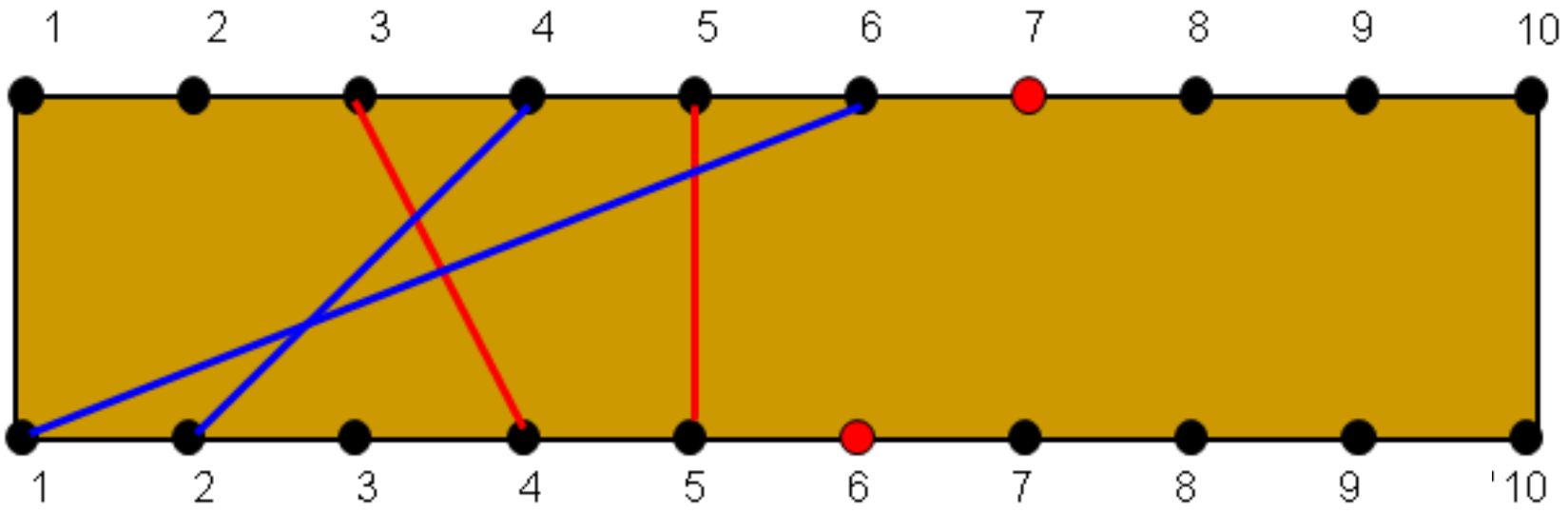
令MNS（i，j）表示一个MNS，其中所有的(t,π(t))满足 t≤i，π(t) ≤j。

令size（i，j）表示一个MNS（i，j）的大小（即连线数目）。

则MNS（n，n）对应给定输入的MNS，size（n，n）为其大小。

例：MNS（7，6） {(3,4),(5,5)}

size(7,6)=2



3.8 电路布线

1 最优子结构性质

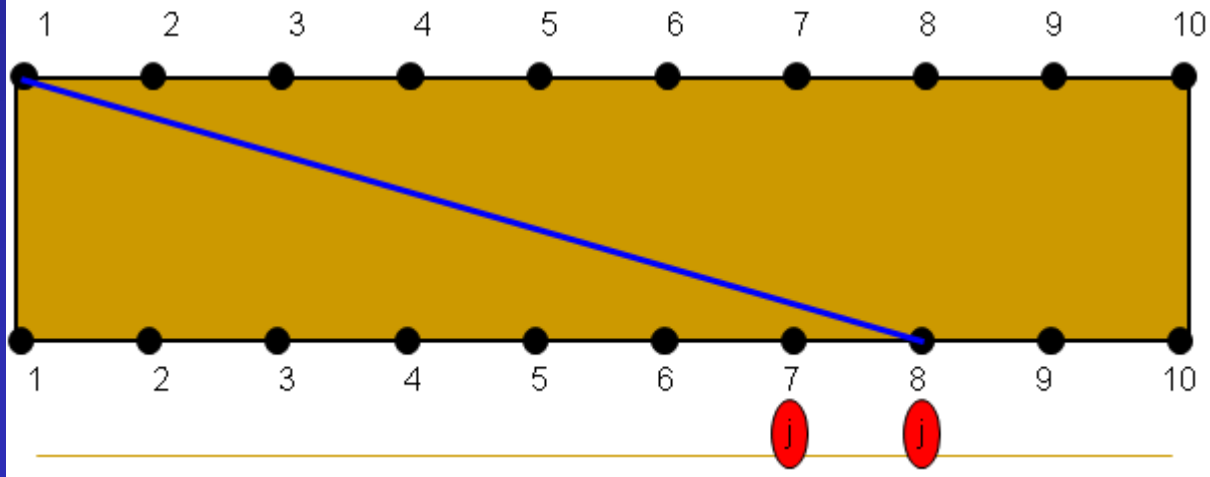
记 N(i,j)={ t |(t, π(t))∈Nets, t ≤ i, π(t) ≤j }

N(i,j)的最大不相交子集为MNS(i,j)

size(i,j)=|MNS(i,j)|

(1)当i=1时，

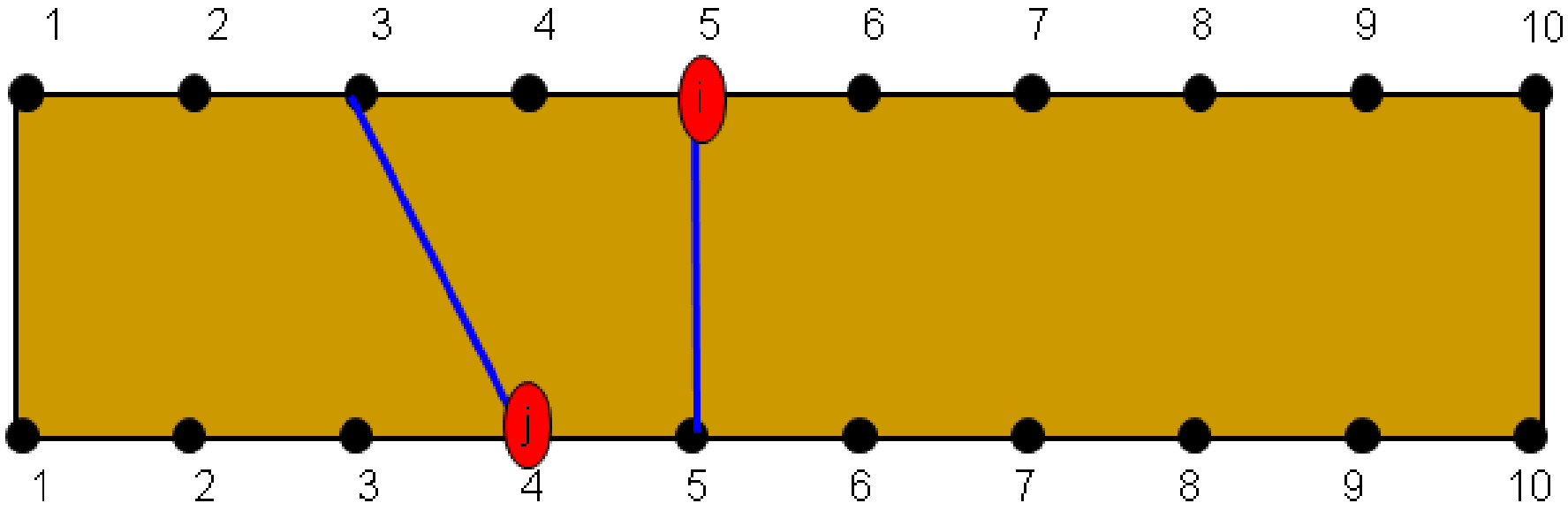
$$MNS(1,j)=N(1,j)=\begin{cases}\varnothing & j<\pi(1)\\ \{(1,\pi(1))\} & j\geq\pi(1)\end{cases}$$



3.8 电路布线

(2)当i>1时，

- j<π(i)： (i, π(i))∉N(i,j)， 故 N(i,j) = N(i-1,j)
  从而 size(i,j) = size(i-1,j)

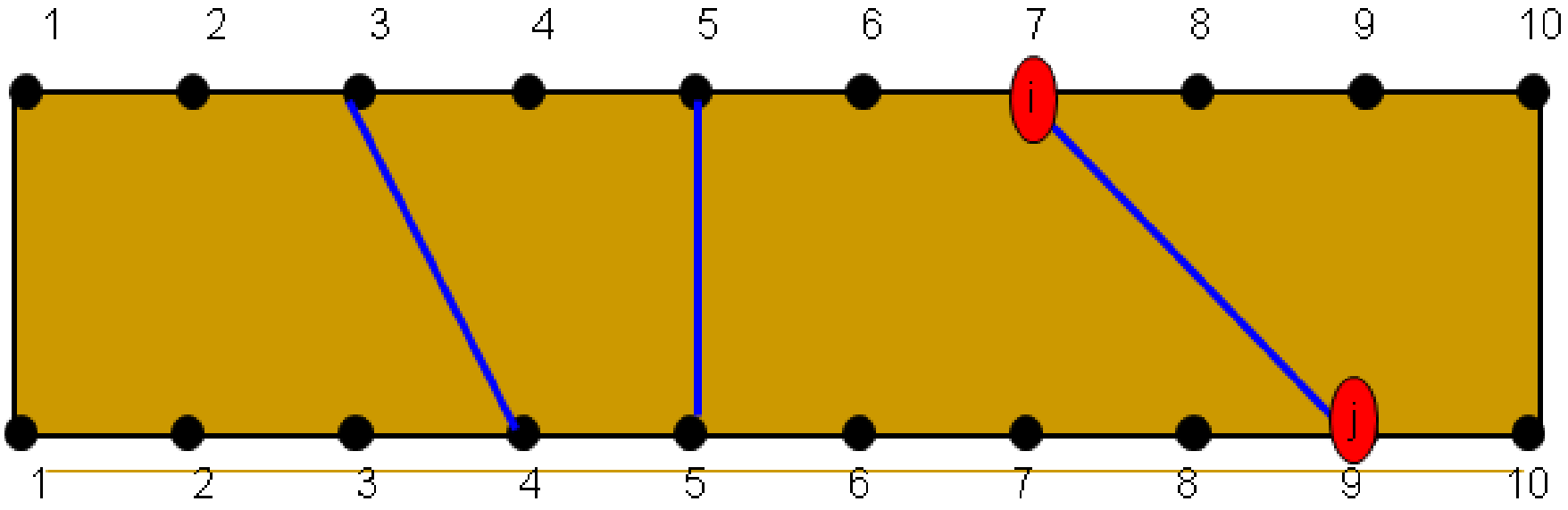


3.8 电路布线

(2)当i>1时，

- $j \geq \pi(i)$：
  - $(i,\pi(i)) \in MNS(i,j)$

    对任意$(t,\pi(t)) \in MNS(i,j)$ 有 $t<i$ 且$\pi(t)<\pi(i)$。故 $MNS(i,j)-\{(i,\pi(i))\}$ 是 $N(i-1,\pi(i)-1)$的最大不相交子集。



3.8 电路布线

(2)当i>1时，

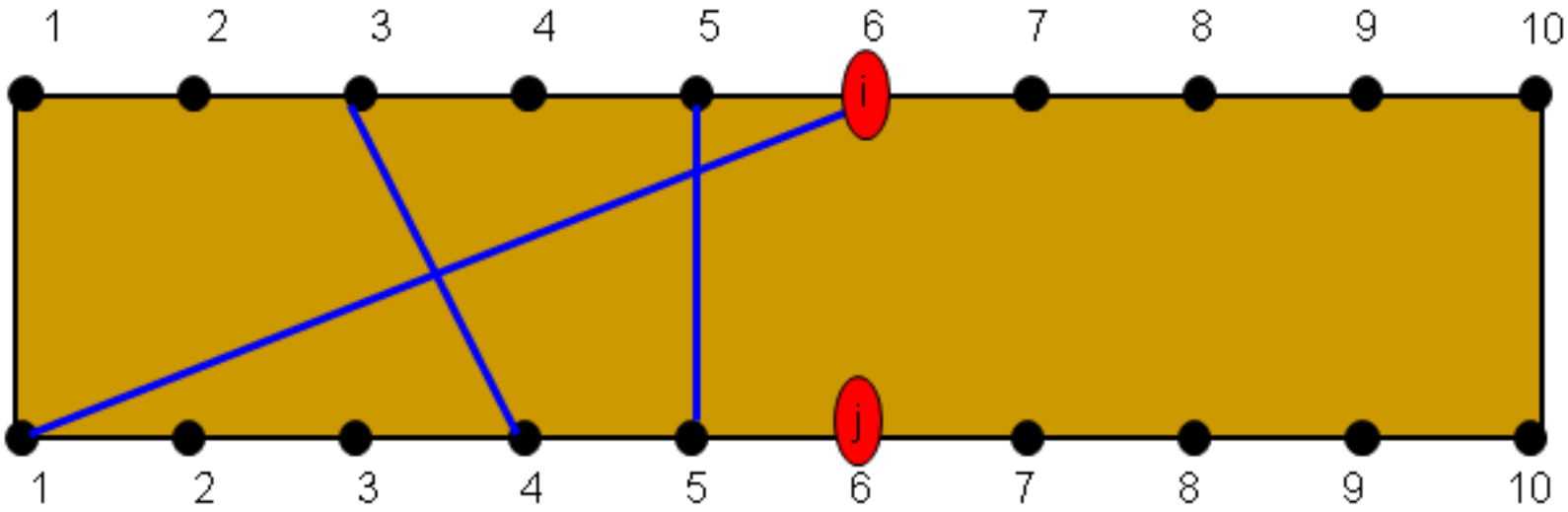
- j≥π(i)：
  - (i, π(i))∉ MNS(i,j)

    对任意 (t,π(t)) ∈MNS(i,j) 有 t<i。从而 MNS(i,j)⊆N(i-1,j) .因此,size(i,j)≤size(i-1,j)。

    同时

    MNS(i-1,j)⊆N(i-1,j)， 故又有size(i,j)≥size(i-1, j)，

    从而 size(i,j) = size(i-1,j)。



3.8 电路布线

(2)当i>1时，

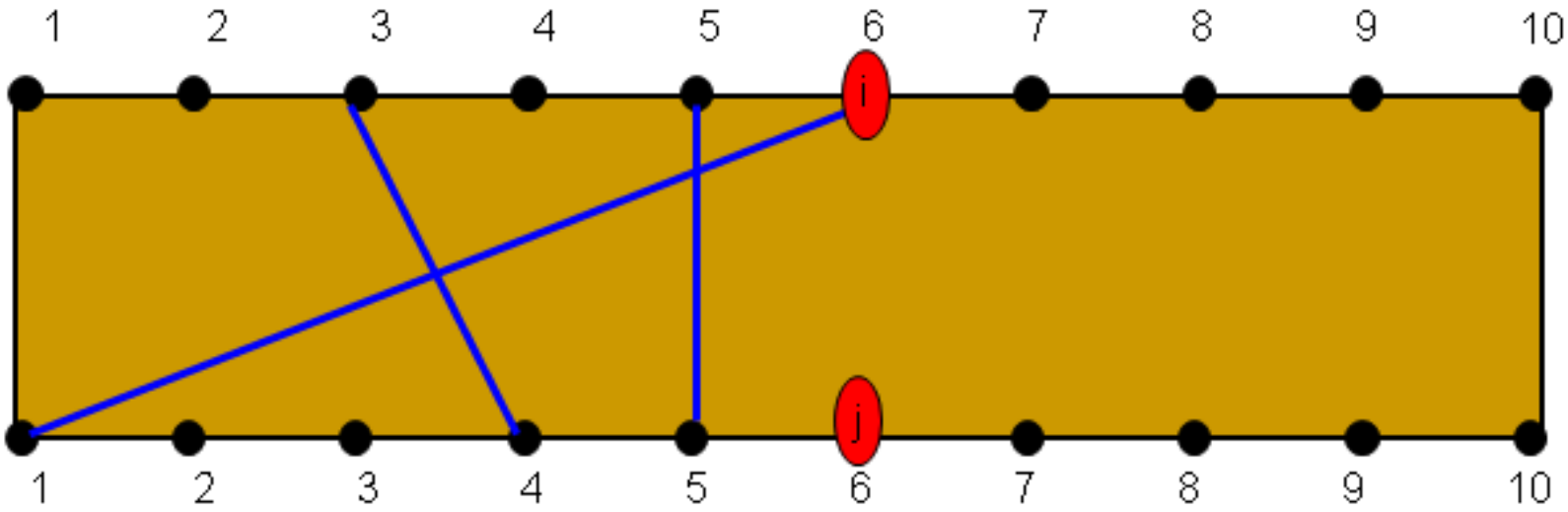
- j≥π(i)：
  - (i, π(i))∉ MNS(i,j)

    对任意 (t,π(t)) ∈MNS(i,j) 有 t<i。从而 MNS(i,j)⊆N(i-1,j) .因此,size(i,j)≤size(i-1,j)。

    同时

    MNS(i-1,j)⊆N(i-1,j)， 故又有size(i,j)≥size(i-1, j)，

    从而 size(i,j) = size(i-1,j)。



3.8 电路布线

2 递归计算最优值

(1)当i=1时 $Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\geq\pi(1)\end{cases}$

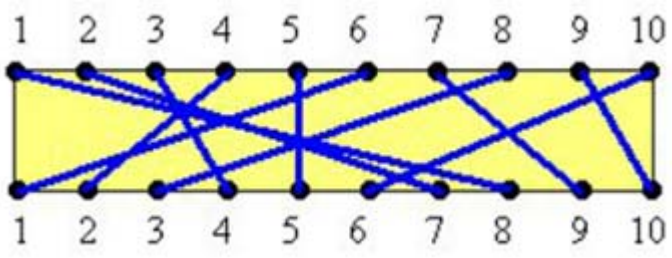
(2)当i>1时

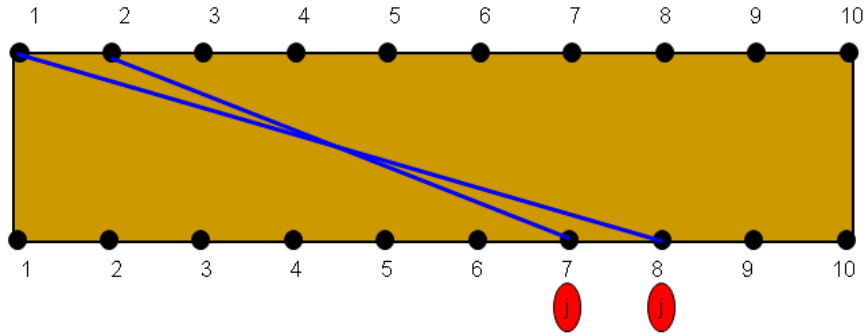
$$Size(i,j)=\begin{cases}Size(i-1,j) & j<\pi(i)\\ \max\{Size(i-1,j),Size(i-1,\pi(i)-1)+1\} & j\geq\pi(i)\end{cases}$$

```
public static void mnset(int[] c, int[][] size)
{
                int n=c.length-1;
                for(int j=0;j<=c[1];j++) size[1][j] =0;      //初始化
                for (int  j=c[1];j<=n; j++ ) size[1][j] =1;  // size[1][*]
                for (int i=2;i<n; i++ ){  //计算size[i][*], 1<i<n
                    for (int  j=0;j<=c[i]; j++ )  // j < c(i)的情况
                        size[i][j]=size[i-1][j];
                    for (int  j=c[i];j<=n; j++ ){ // j ≥c(i)的情况
                        if(size[i-1][j]>(size[i-1][c[i]-1]+1))
                            size[i][j]=size[i-1][j];
                        else size[i][j]=size[i-1][c[i]-1]+1;
                }
        }
                 if(size[n-1][n]>(size[n-1][c[n]-1]+1)) //计算size[n][n]
                        size[n][n]=size[n-1][n];
                else size[n][n]=size[n-1][c[n]-1]+1;
}
```

(1)当i=1时 $Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\geq\pi(1)\end{cases}$

i	j										
	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	1	1	1	1
3	0	0	0						1	1	1
4	0	0	1						1	1	1
5	0										
10	0										





(1)当i=1时 $Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\geq\pi(1)\end{cases}$

(2)当i>1时

$$Size(i,j)=\begin{cases}Size(i-1,j) & j<\pi(i)\\ \max\{Size(i-1,j),Size(i-1,\pi(i)-1)+1\} & j\geq\pi(i)\end{cases}$$

c[1] =8 size[1][0]~ size[1][7]=0 size[1][8]~ size[1][10] =1

c[2] =7 0≤j<6 size[2] [j]= size[i-1] [j] =0

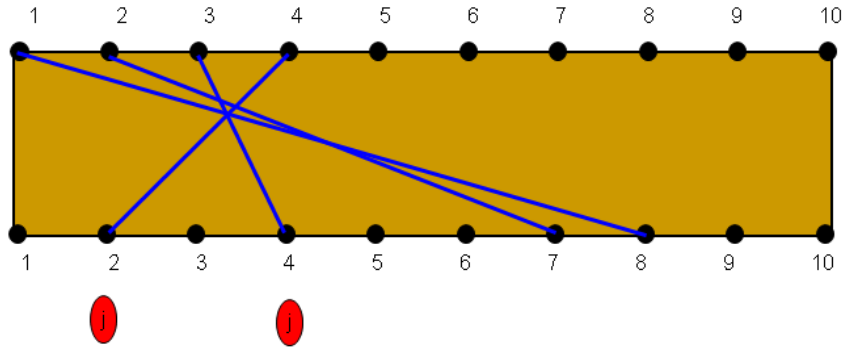
7≤ j ≤ 10 size[2][j]=max{size[1][j]，size[1][6]+1}=1

i	j										
	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	1	1	1	1
3	0	0	0	0	1	1	1	1	1	1	1
4	0	0	1	1	1	1	1	1	1	1	1
5	0										
10	0										

(1)当i=1时 $Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\geq\pi(1)\end{cases}$

(2)当i>1时

$$Size(i,j)=\begin{cases}Size(i-1,j) & j<\pi(i)\\ \max\{Size(i-1,j),Size(i-1,\pi(i)-1)+1\} & j\geq\pi(i)\end{cases}$$



c[3] =4 size[3][4]=max{size[2][4]，size[2][3]+1}=1

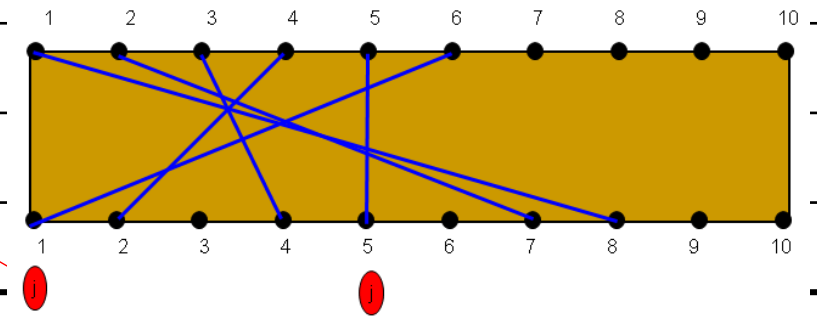
c[4] =2 size[4][2]=max{size[3][2]，size[3][1]+1}=1

i	j										
	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	1	1	1	1
3	0	0	0	0	1	1	1	1	1	1	1
4	0	0	1	1	1	1	1	1	1	1	1
5	0	0	1	1	1	2	2	2	2	2	2
6	0	1	1	1	1	2	2	2	2	2	2
7	0										

(1)当i=1时 $Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\geq\pi(1)\end{cases}$

(2)当i>1时

$$Size(i,j)=\begin{cases}Size(i-1,j) & j<\pi(i)\\ \max\{Size(i-1,j),Size(i-1,\pi(i)-1)+1\} & j\geq\pi(i)\end{cases}$$



C[5]=5 size[5][5]=max{size[4][5]，size[4][4]+1}=2

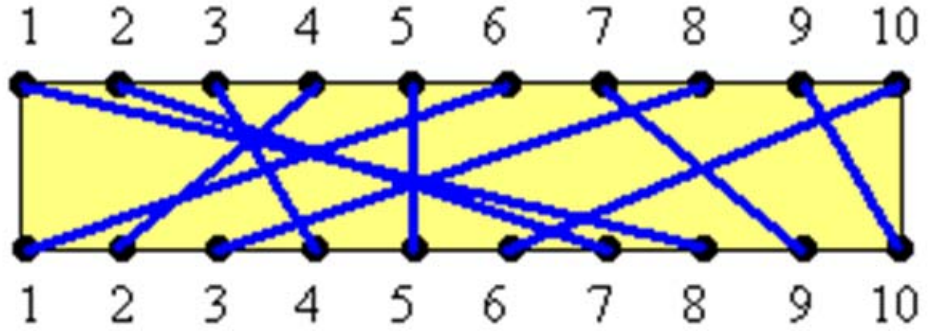
C[6]=1 size[6][1]=max{size[5][1]，size[5][0]+1}=1

i	j										
	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1	1
6	0	1	1	1	1	2	2	2	2	2	2
7	0	1	1	1	1	2	2	2	2	3	3
8	0	1	1	2	2	2	2	2	2	3	3
9	0	1	1	2	2	2	2	2	2	3	4
10	0	1	1	2	2	2	3	3	3	3	4

(1)当i=1时 $Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\geq\pi(1)\end{cases}$

(2)当i>1时

$$Size(i,j)=\begin{cases}Size(i-1,j) & j<\pi(i)\\ \max\{Size(i-1,j),Size(i-1,\pi(i)-1)+1\} & j\geq\pi(i)\end{cases}$$



C[7]=9 size[7][9]=max{size[6][9]，size[6][8]+1}=3

C[8]=3 size[8][3]=max{size[7][3]，size[7][2]+1}=2

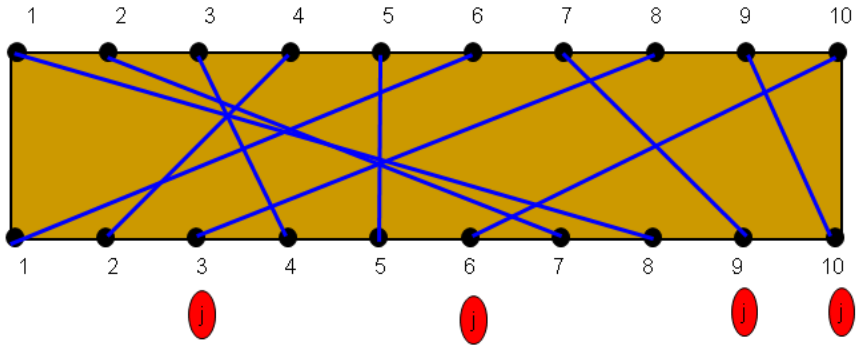
C[9]=10 size[9][10]=max{size[8][10]，size[8][9]+1}=4

i	j										
	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	1	1	1	1
7	0	1	1	1	1	2	2	2	2	3	3
8	0	1	1	2	2	2	2	2	2	3	3
9	0	1	1	2	2	2	2	2	2	3	4
10	0	1	1	2	2	2	3	3	3	3	4

(1)当i=1时 $Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\geq\pi(1)\end{cases}$

(2)当i>1时

$$Size(i,j)=\begin{cases}Size(i-1,j) & j<\pi(i)\\ \max\{Size(i-1,j),Size(i-1,\pi(i)-1)+1\} & j\geq\pi(i)\end{cases}$$



C[10]=6 size[10][6]=max{size[9][6]，size[9][5]+1}=3

C[10]=6 size[10][10]=max{size[9][10]，size[9][9]+1}=4

3 构造最优解

size[i][j]和net[i]记录了MNS的信息。
size[i][j]存储了MNS[i][j]的值；
net[0:m-1]存储了所得到的MNS。

```
public static  int traceback(int[] c, int[][] size, int[] net)
{
      int n=c.length-1, j=n, m=0;
      for(int i=n; i>1; i--)
         if (size[i][j]!=size[i-1][j])
      {
            net[m++]=i;
            j=c[i]-1;
      }
      if(j>=c[1])  net[m++]=1;
      return  m;
}
```

说明当前这条线属于最大不相交子集

j的位置随着已经确定的最大不相交子集的范围而变化

最后一条线，即i=1的那条线，必定可取）

m = 1
net[0：m] = 9 7 5 3

i	j										
	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	1	1	1	1
3	0	0	0	0	1	1	1	1	1	1	1
4	0	0	1	1	1	1	1	1	1	1	1
5	0	0	1	1	1	2	2	2	2	2	2
6	0	1	1	1	1	2	2	2	2	2	2
7	0	1	1	1	1	2	2	2	2	3	3
8								2	2	3	3
9								2	2	3	4
10								3	3	3	4

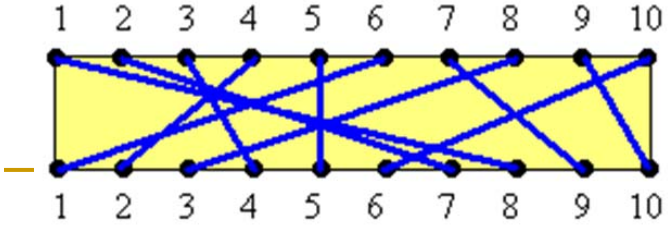
```
public static int traceback(int[] c, int[][] size, int[] net)
{
    int n=c.length-1, j=n, m=0;
    for(int i=n; i>1; i--)
        if (size[i][j]!=size[i-1][j])
        {
            net[m++]=i;
            j=c[i]-1;
        }
    if(j>=c[1]) net[m++]=1;
    return m;
}
```

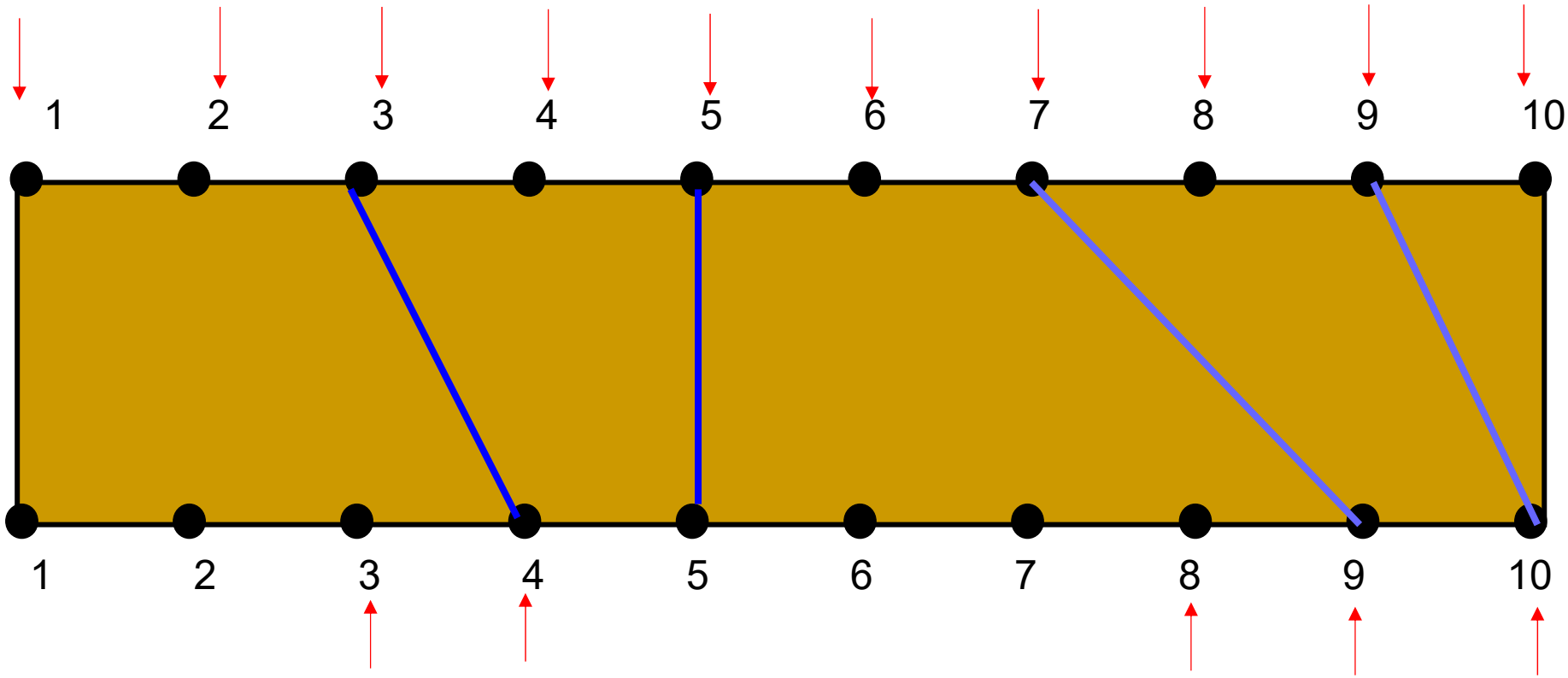
说明当前这条线属于最大不相交子集

最后一条线，即i=1的那条线，必定可取）

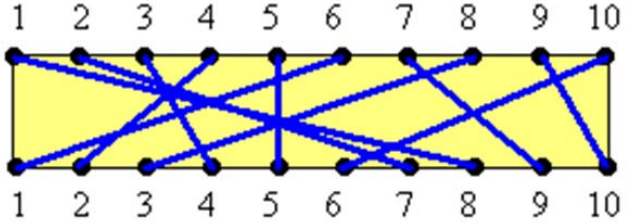
j的位置随着已经确定的最大不相交子集的范围而变化

m = 1
net[0：m] = 9 7 5 3





i	j										
	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	1	1	1	1
3	0	0	0	0	1	1	1	1	1	1	1
4	0	0	1	1	1	1	1	1	1	1	1
5	0	0	1	1	1	2	2	2	2	2	2
6	0	1	1	1	1	2	2	2	2	2	2
7	0	1	1	1	1	2	2	2	2	3	3
8	0	1	1	2	2	2	2	2	2	3	3
9	0	1	1	2	2	2	2	2	2	3	4
10	0	1	1	2	2	2	3	3	3	3	4



m = 4

net[0：m] = 9 7 5 3