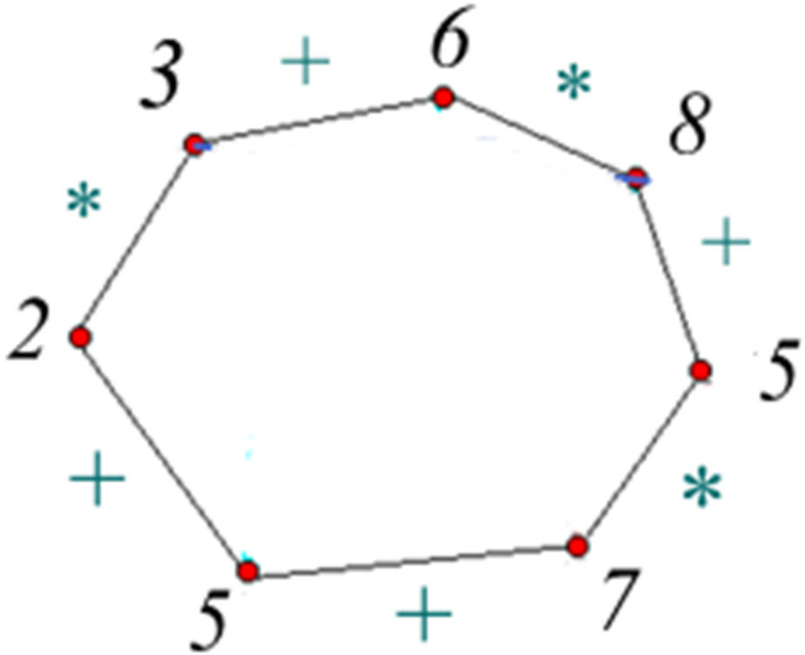
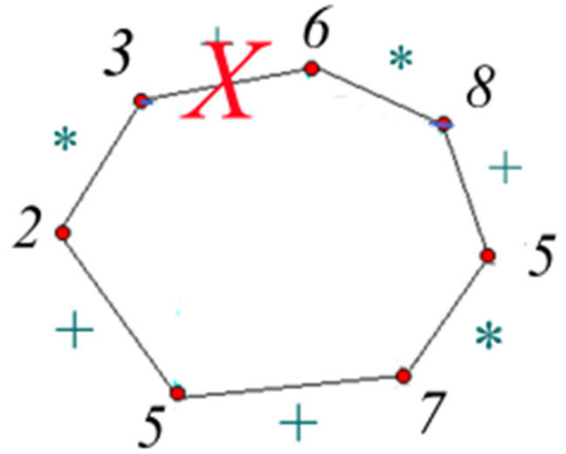
多边形游戏

多边形游戏是一个单人玩的游戏，开始时有一个由n个顶点构成的多边形。每个顶点被赋予一个整数值，每条边被赋予一个运算符 “+”或 “*” 。所有边依次用整数从1到n编号。



多边形游戏



游戏第1步，将一条边删除。

随后n-1步按以下方式操作：

(1)选择一条边E以及由E连接着的2个顶点V_1和V_2；

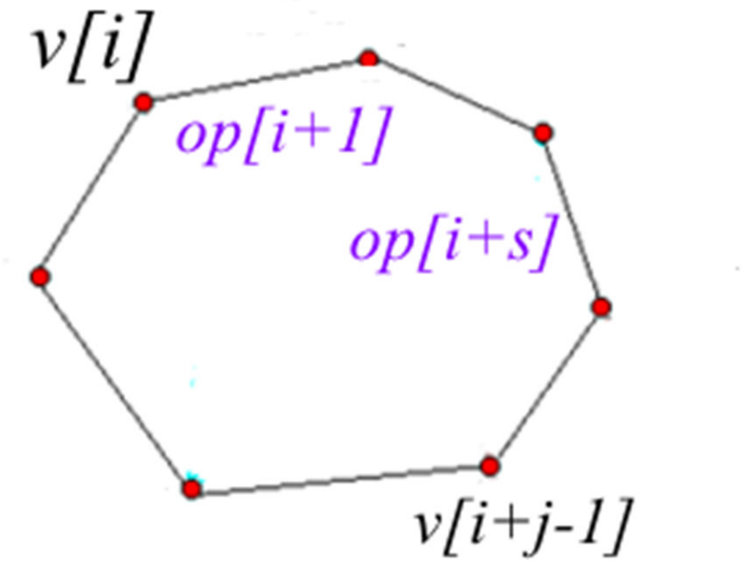
(2)用一个新的顶点取代边E以及由E连接着的2个顶点V_1和V_2。将由顶点V_1和V_2的整数值通过边E上的运算(顺时针)得到的结果赋予新顶点。

最后，所有边都被删除，游戏结束。游戏的得分就是所剩顶点上的整数值。

问题:对于给定的多边形，计算最高得分。

最优子结构性质

- 在所给多边形中，从顶点i(1≤i≤n)开始，长度为j(链中有j个顶点)的顺时针链p(i，j) 可表示为v[i]，op[i+1]，…，v[i+j-1]。
- 如果这条链的最后一次合并运算在op[i+s]处发生(1≤s≤j-1)，则可在op[i+s]处将链分割为2个子链p(i，s)和p(i+s，j-s)。



最优子结构性质

- 设m_1是对子链p(i，s)的任意一种合并方式得到的值，而a和b分别是在所有可能的合并中得到的最小值和最大值。m_2是p(i+s，j-s)的任意一种合并方式得到的值，而c和d分别是在所有可能的合并中得到的最小值和最大值。依此定义有

$a \le m_1 \le b$，$c \le m_2 \le d$

m=(m1)op[i+s](m2)

(1)当op[i+s]='+'时，显然有$a+c \le m \le b+d$

(2)当op[i+s]=‘*’时，有

$$\min\{ac, ad, bc, bd\} \le m \le \max\{ac, ad, bc, bd\}$$

- 换句话说，主链的最大值和最小值可由子链的最大值和最小值得到。

用动态规划法求最优解

```
int PolyMax(int n){
      int minf, maxf;
      for(int j=2;j<=n;j++){ //遍历链的长度
           for(int i=1;i<=n;i++){ //遍历链的起点
                for(int s=1;s<j;s++){ //遍历断开的链
                     MinMax(n, i, s, j, minf, maxf, m, op);
                     if(m[i][j][0] > minf)
                          m[i][j][0] = minf;
                     if(m[i][j][1] < maxf)
                          m[i][j][1] = maxf;
       int temp = m[1][n][1];
       for(int i=2;i<=n;i++) //该循环找出从哪个点开始的包含所有节点的链取最大值
            if(temp<m[i][n][1])
                 temp = m[i][n][1];
       return temp;
}
```

用动态规划法求最优解

```
void MinMax(int n, int i, int s, int j, int& minf, int& maxf, int m[ ][ ][ ], int op){
      int e[4], a=m[i][s][0], b=m[i][s][1];
      int r = (i+s-1)%n+1, c=m[r][j-s][0], d=m[r][j-s][1];
      if(op[r] == '+’)
            minf = a+c;
            maxf = b+d;
      else
            e[1] = a*c;   e[2] = a*d;
            e[3] = b*c;   e[4] = b*d;
            minf = e[1];
            maxf = e[1];
            for(int k=2;k<5;k++)
                  if(minf>e[k])    minf = e[k];
                  if(maxf<e[k])    maxf = e[k];
}
```

该函数处理从v[i]起始，长度为j的链，在op[i+s]处断开的值。
第一条子链起始点为i，长度为s，第二条子链起始点为r，长度为j-s
将最小值赋给minf，最大值赋给maxf

图像压缩

像素点灰度值：0~255，8位二进制数

图像的灰度值序列：{p1, p2, …, pn}，

pi为第i个像素点灰度值

图像存储：每个像素的灰度值占8位，总计空间为8n

有没有更好的存储方法

图像压缩

图像的变位压缩存储格式将所给的像素点序列 {p1, p2, ..., pn} 分割成m个连续段S1,S2,...,Sm，同一像素段占位相同。

第i个像素段Si(1≤i≤m)，有$l[i]$个像素点，且该段中每个像素点都只用b[i]位表示。设$t[i]=\sum_{k=1}^{i-1} l[k]$，则第i个像素段Si为

$S_i = \{p_{t[i]+1},\ldots,p_{t[i]+l[i]}\}$ 1≤i≤m。

* | * * * * | * * *

pi： 88 255 16 9 145 23 2 10

段头：记录l[i] (8位)和b[i](3位) 需要11位

总位数为：b[1] l[1]+b[2]l[2]+……+b[m]l[m]+11m

图像压缩

图像压缩问题要求确定象素序列{p1, p2, …, pn}的最优分段，使得依此分段所需的存储空间最少。每个分段的长度不超过256位。

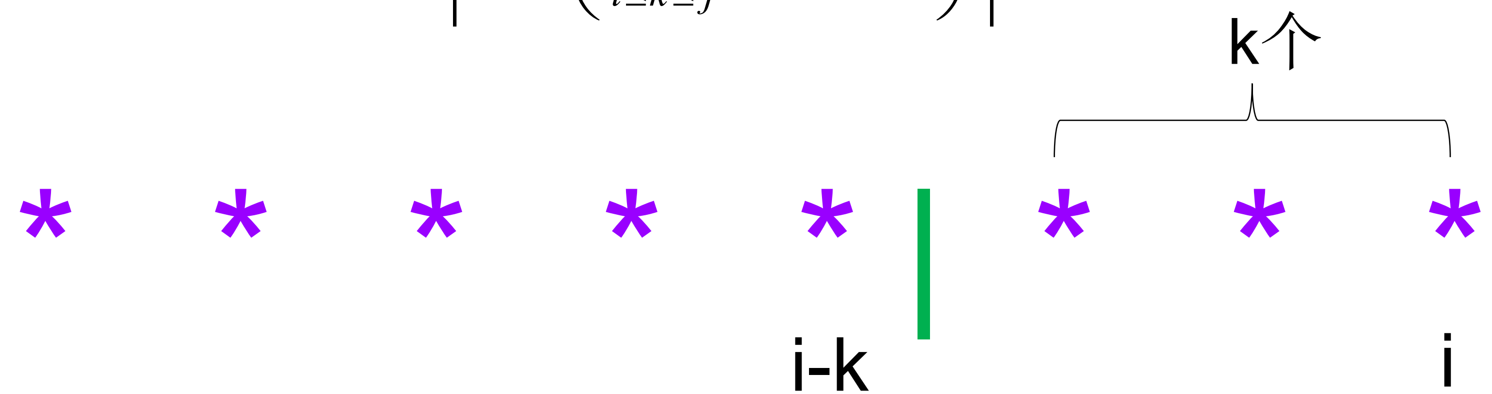
设$l[i]$，$b[i]$，$1\le i\le m$是$\{p1,p2,\ldots,pn\}$的最优分段。显而易见，$l[1]$，$b[1]$是$\{p_1,\ldots,p_{l[1]}\}$的最优分段，且$l[i]$，$b[i]$，$2\le i\le m$是$\{p_{l[1]+1},\ldots,p_n\}$的最优分段。即图像压缩问题满足最优子结构性质。

图像压缩

设l[i]，b[i]是{p1,p2,…,pn}的最优分段。设s[i]，1≤i≤n，是像素序列{p1,…,pi}的最优分段所需的存储位数。由最优子结构性质易知：

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

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



用动态规划法求最优值

```
void compress(int n, int
        int Lmax = 256,
        for(int i = 1; i
            b[i] = leng
            bmax = b[i]
            s[i] = s[i-
            l[i] = 1;
            for(int j=2; j <= i && j<= Lmax; j++)   //最后一段包含的像素数
                if(bmax < b[i-j+1])
                    bmax = b[i-j+1];
                if(s[i] > s[i-j] + j*bmax)
                    s[i] = s[i-j] + j*bmax;
                    l[i] = j;
            s[i] += header;
}
```

算法复杂度分析：

由于算法**compress**中对j的循环次数不超过256，故对每一个确定的i，可在时间O(1)内完成的计算。因此整个算法所需的计算时间为O(n)。

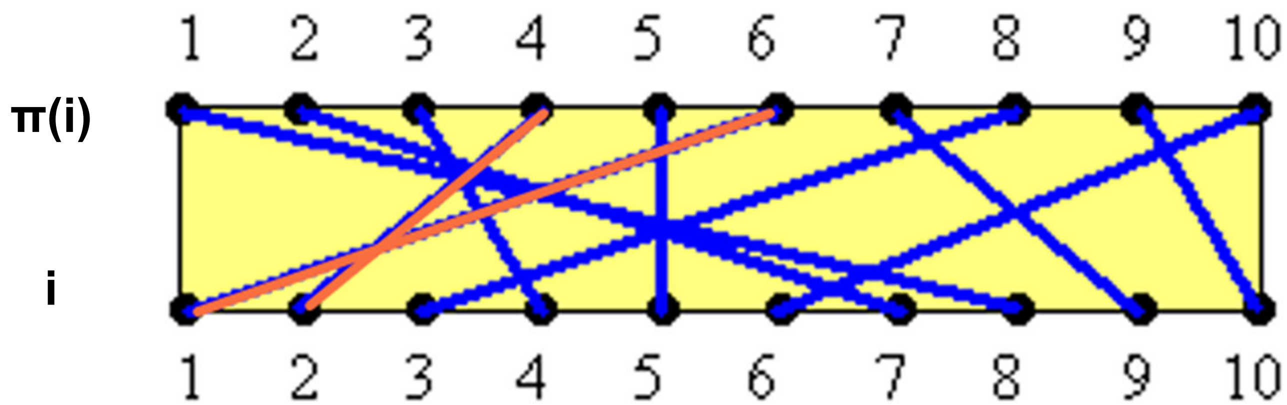
存储灰度值为i所需的位数

```
int length(int i){
        int k = 1, i = i/2;
        while(i>0)
            k++;
            i = i/2;
        return k;
}
```

电路布线

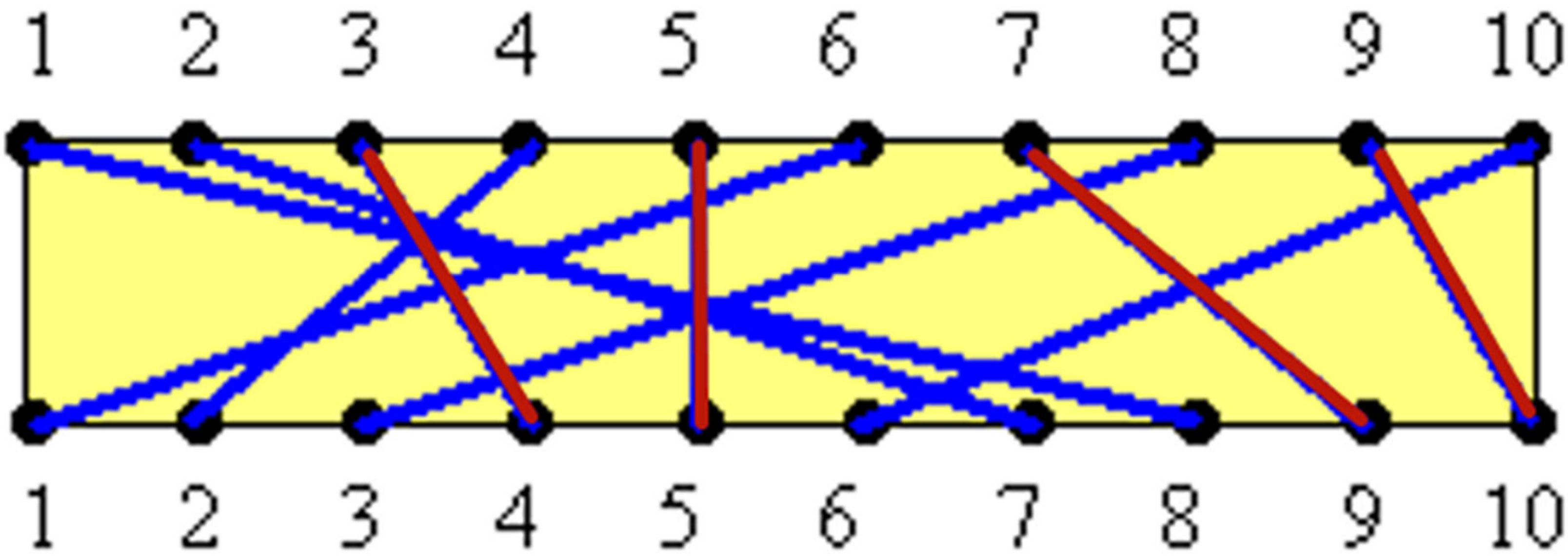
在一块电路板的上、下2端分别有n个接线柱。根据电路设计，要求用导线(i,π(i))将上端接线柱与下端接线柱相连，如图所示。其中π(i)是{1,2,…,n}的一个排列。导线(i,π(i))称为该电路板上的第i条连线。对于任何1≤i<j≤n，第i条连线和第j条连线相交的充分且必要的条件是π(i)>π(j)。

π (2) =4 π(1)=6



电路布线

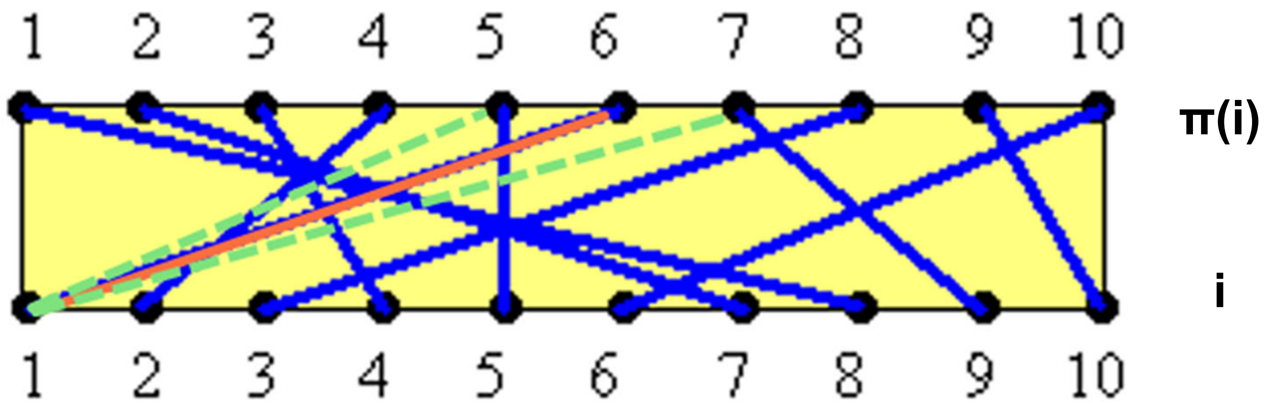
电路布线问题要确定将哪些连线安排在第一层上，使得该层上有尽可能多的连线。换句话说，该问题要求确定导线集 Nets={(i,π(i)),1≤i≤n}的最大不相交子集。



计算最优值

- Nets = {(i, π(i)),1≤i≤n}为导线集。
- 记 $N(i,j)=\{t\mid (t,\pi(t))\in Nets, t\le i, \pi(t)\le j\}$ 。
- N(i, j)的最大不相交子集为MNS(i, j)。
- Size(i, j) = |MNS(i, j)| 。

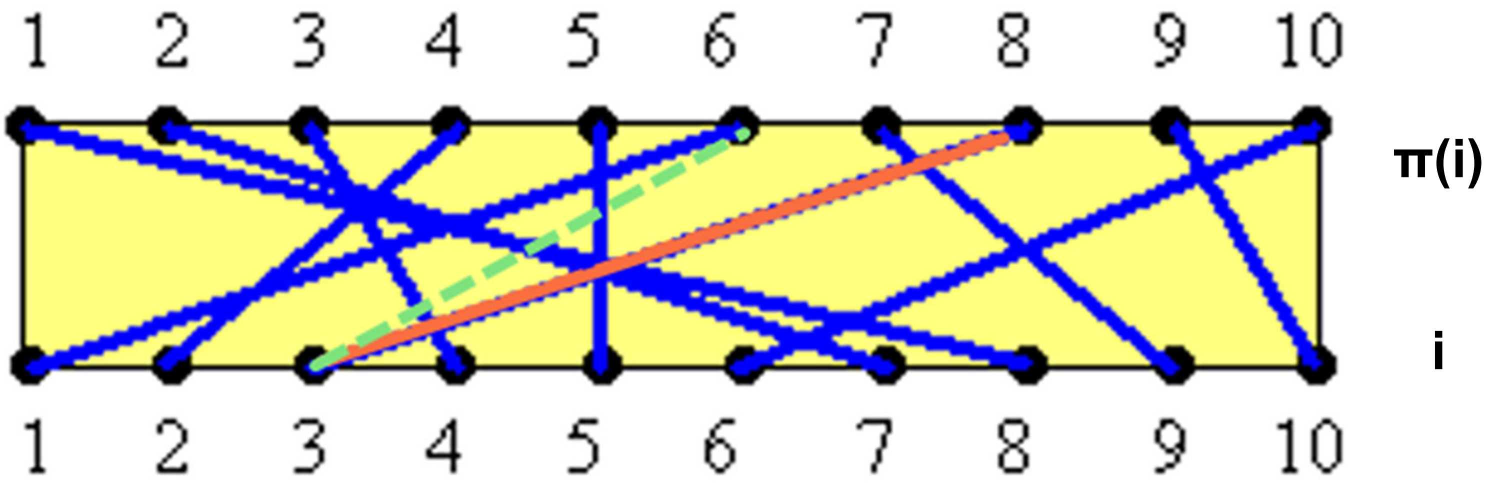
1） 当i=1时，$MNS(1,j)=N(1,j)=\begin{cases}\varnothing & j<\pi(1)\\ \{(1,\pi(1))\} & j\ge \pi(1)\end{cases}$



计算最优值

2) 当i>1时，

① j<π(i)。此时，$(i,\pi(i)) \notin N(i,j)$ 。故在这种情况下，
N(i,j)=N(i-1,j)，从而Size(i,j)=Size(i-1,j)。

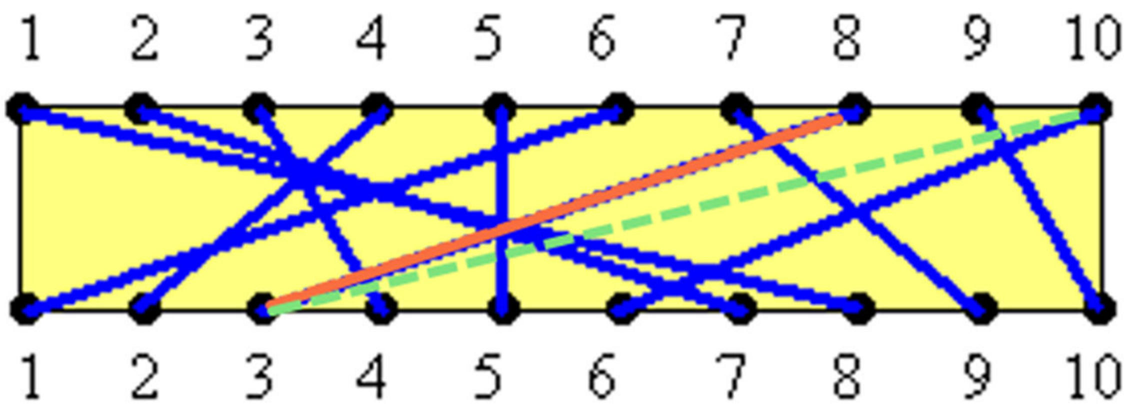


计算最优值

2) 当i>1时，

② j≥π(i)。

- 若(i, π(i))∈MNS(i, j) 。则对任意(t, π(t)) ∈MNS(i,j)，有t<i且π(t)<π(i)。在这种情况下MNS(i, j)-{(i, π(i))}是 N(i-1,π(i)-1)的最大不相交子集。
- 若(i, π(i))∉MNS(i, j)，则对任意(t,π(t)) ∈MNS(i,j)有t<i，Size(i, j)=Size(i-1, j)。



计算最优值

电路布线问题的最优值为Size(n, n)

1)当i=1时

$$Size(1,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j\ge\pi(1)\end{cases}$$

2)当i>1时

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

用动态规划法求最优值

```
void MNS(int C[], int n,  int[][] size){
        for(int j = 0; j < C[1]; j++)
               size[1][j] = 0;
         for(int j = C[1]; j <= n; j++)
               size[1][j] = 1;
         for(int i = 2; i < n; i++)
               for(int j = 0; j < C[i]; j++)
                     size[i][j] = size[i-1][j];
         for(int j = C[i]; j <= n; j++)
               size[i][j] = max(size[i-1][j], size[i-1][C[i]-1] + 1);
          size[n][n] = max(size[n-1][n], size[n-1][C[n]-1] + 1);
}
```

流水作业调度

n个作业{1，2，…，n}要在由2台机器M1和M2组成的流水线上完成加工。每个作业加工的顺序都是先在M1上加工，然后在M2上加工。M1和M2加工作业i所需的时间分别为a_i和 b_i。

流水作业调度问题要求确定这n个作业的最优加工顺序，使得从第一个作业在机器M1上开始加工，到最后一个作业在机器M2上加工完成所需的时间最少。

流水作业调度

分析：

- 直观上，一个最优调度应使机器M1没有空闲时间，且机器M2的空闲时间最少。在一般情况下，机器M2上会有机器空闲和作业积压2种情况。
- 设全部作业的集合为N={1，2，…，n}。S⊆N是N的作业子集。在一般情况下，机器M1开始加工S中作业时，机器M2还在加工其他作业，要等时间t后才可利用。将这种情况下完成S中作业所需的最短时间记为T(S, t)。流水作业调度问题的最优值为T(N, 0)。

流水作业调度

机器M1开始加工S中作业时，机器M2还在加工其他作业，要等时间t后才可利用。将这种情况下完成S中作业所需的最短时间记为T(S, t)

设π是所给n个流水作业的一个最优调度，它所需的加工时间为$a_{\pi(1)}+T'$。其中T'是在机器M2的等待时间为$b_{\pi(1)}$时，安排作业π(2)，…，π(n)所需的时间。记S=N-{π(1)}，则有$T'=T(S, b_{\pi(1)})$。

证明：

- 事实上，由T的定义知$T' \geq T(S, b_{\pi(1)})$。
- 若$T' > T(S, b_{\pi(1)})$，设π'是作业集S在机器M2的等待时间为$b_{\pi(1)}$情况下的一个最优调度。则π(1)，π'(2)，…，π'(n)是N的一个调度，且该调度所需的时间为$a_{\pi(1)}+T(S, b_{\pi(1)}) < a_{\pi(1)}+T'$。这与π是N的最优调度矛盾。故$T' \leq T(S, b_{\pi(1)})$。从而$T'=T(S,b_{\pi(1)})$

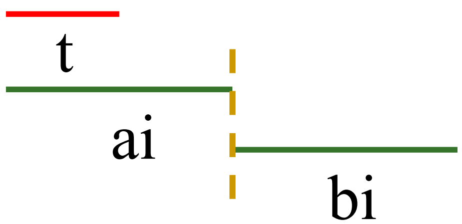
这就证明了流水作业调度问题具有最优子结构性质。

流水作业调度

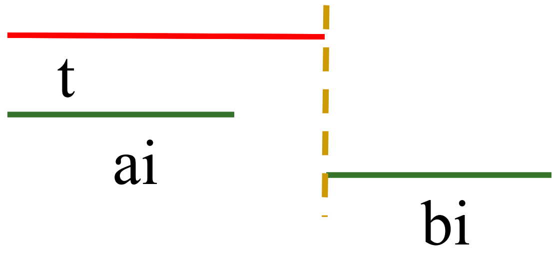
由流水作业调度问题的最优子结构性质可知，

$$T(N,0)=\min_{1\le i\le n}\{a_i+T(N-\{i\},b_i)\}$$

$$T(S,t)=\min_{i\in S}\{a_i+T(S-\{i\},b_i+\max\{t-a_i,0\})\}$$



t-ai<0



t-ai>0

流水作业调度

对递归式的深入分析表明，算法可进一步得到简化。

设π是作业集S在机器M2的等待时间为t时的任一最优调度。

若π(1)=i, π(2)=j。则由动态规划递归式可得:

$T(S, t)=a_i+T(S-\{i\},b_i+\max\{t-a_i,0\})=a_i+a_j+T(S-\{i,j\}, t_{ij})$

其中，

$$
\begin{aligned}
t_{ij} &= b_j + \max\{b_i + \max\{t-a_i,0\} - a_j, 0\} \\
&= b_j + b_i - a_j + \max\{\max\{t-a_i,0\}, a_j - b_i\} \\
&= b_j + b_i - a_j + \max\{t-a_i, a_j - b_i, 0\} \\
&= b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_i, a_i\}
\end{aligned}
$$

流水作业调度的Johnson法则

$\pi(1)=i$, $\pi(2)=j$

$T(S, t) = a_i+a_j+T(S-\{i,j\}, t_{ij})$

$$t_{ij}=b_j+b_i-a_j-a_i+\max\{t,a_i+a_j-b_i,a_i\}$$

交换作业i和作业j的加工顺序，得到作业集S的另一调度π'，它所需的加工时间为：$T'(S, t) = a_i+a_j+T(S-\{i,j\}, t_{ji})$

其中，

$$t_{ji}=b_j+b_i-a_j-a_i+\max\{t,a_i+a_j-b_j,a_j\}$$

流水作业调度的Johnson法则

$$t_{ij} = b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_i, a_i\}$$

$$t_{ji} = b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_j, a_j\}$$

如果作业i和j满足$\min\{b_i,a_j\} \geq \min\{b_j,a_i\}$，则称作业i和j满足**Johnson不等式。**

$$\max\{-b_i,-a_j\} \leq \max\{-b_j,-a_i\}$$

$$a_i + a_j + \max\{-b_i,-a_j\} \leq a_i + a_j + \max\{-b_j,-a_i\}$$

$$\max\{a_i + a_j - b_i, a_i\} \leq \max\{a_i + a_j - b_j, a_j\}$$

$$\max\{t, a_i + a_j - b_i, a_i\} \leq \max\{t, a_i + a_j - b_j, a_j\}$$

$$t_{ij} \leq t_{ji}$$

$$T(S,t) \leq T'(S,t)$$

流水作业调度的Johnson法则

- 由此可见当作业i和作业j不满足Johnson不等式时，交换它们的加工顺序后，不增加加工时间。
- 对于流水作业调度问题，必存在最优调度π，使得作业π(i)和π(i+1)满足Johnson不等式。进一步还可以证明，调度满足Johnson法则当且仅当对任意i<j有

$$\min\{b_{\pi(i)}, a_{\pi(j)}\} \geq \min\{b_{\pi(j)}, a_{\pi(i)}\}$$

由此可知，**所有满足Johnson法则的调度均为最优调度。**

算法描述

$\min\{b_i,a_j\} \geq \min\{b_j,a_i\}$

流水作业调度问题的Johnson算法

(1)令 $N_1 = \{i \mid a_i < b_i\}, N_2 = \{i \mid a_i \geq b_i\};$

(2)将N_1中作业依a_i的非减序排序；将N_2中作业依b_i的非增序排序；

(3)N_1中作业接N_2中作业构成满足Johnson法则的最优调度。

算法复杂度分析：

算法的主要计算时间花在对作业集的排序。因此，在最坏情况下算法所需的计算时间为O(nlogn)。所需的空间为O(n)。