算法分析与设计 第八讲

动态规划及实例分析 续

主要内容

- 最大子段和问题
- 凸多边形最优三角剖分
- 多边形游戏
- 图像压缩
- 电路布线
- 0-1背包问题

最大子段和问题

- n个整数序列$a_1 \ldots a_n$，求该序列形如 $\sum_{k=i}^{j} a_k$ 的子段和的最大值
- 当所有整数均为负整数时定义其最大子段和为0
- 根据以上，$\max\left\{0, \max_{1\le i\le j\le n}\sum_{k=i}^{j} a_k\right\}$
  - 例：当$(a_1, a_2, \ldots, a_6)$= (-2,11 -4,13,-5,-2)
  - 此最大子段和为 $\sum_{k=2}^{4} a_k = 20$

最大子段和问题简单算法

- 用数组a[]存储n个整数$a_1 \ldots a_n$

```
void MaxSum(int n，int *a，int& besti，int& bestj)
{       int sum=0;
      for (int i = 1; i <= n; i++)
        for (int j = i; j <= n; j++){
            int thissum=0;
            for (int k = i; k<=j; k++) thissum+=a[k]; //i→j
            if (thissum>sum) {//记录i，j
                sum=thissum;
                besti=i;
                bestj=j;
            }
      }
      return sum;
}
```

可进行改进

显然计算时间是$O(n^3)$

最大子段和算法改进

```
void MaxSum(int n，int *a，int& besti，int& bestj)
{
      int sum=0;
      for (int i = 1; i <= n; i++){
         int thissum=0;
         for (int j = i; j <= n; j++){
             thissum+=a[j];                    //i→n的和
             if (thissum>sum) {
                 sum=thissum;
                 besti=i;
                 bestj=j;
             }
         }
      }
      return sum;
}
```

从算法设计技巧上的改进，
计算时间是$O(n^2)$

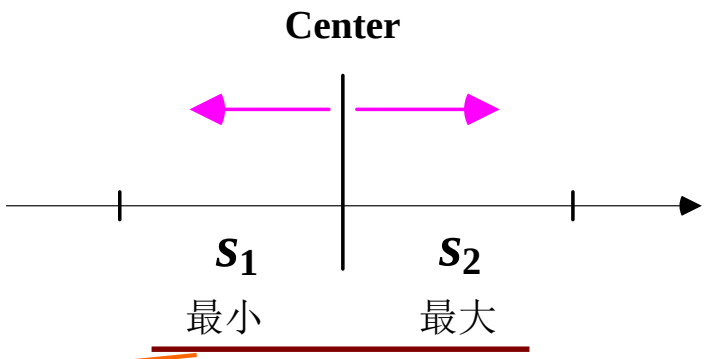
最大子段和算法进一步改进

- 如果将所给的序列$a[1..n]$分为长度相等的两段$a[1:n/2]$和$a[n/2+1:n]$,分别求出这两段最大子段和，则$a[1..n]$的最大子段和有三种情形：
  1. $a[1:n]$的最大子段和与$a[1:n/2]$的最大子段和相同；
  2. $a[1:n]$的最大子段和与$a[n/2+1:n]$的最大子段和相同；
  3. $a[1:n]$的最大子段和为 $\sum_{k=i}^{j} a_k$ ，且 $1 \le i \le n/2$ ，$n/2+1 \le j \le n$ 。
- 对于1，2两种情况可以递归求解
- 对于3， $a[n/2]$与$a[n/2+1]$在最优子序列中
  - 可以在a[1:n/2]中计算出 $s_1 = \max_{1 \le i \le n/2} \sum_{k=i}^{n/2} a[k]$
  - 可以在a[n/2+1:n]中计算出 $s_2 = \max_{n/2+1 \le i \le n} \sum_{k=\frac{n}{2}+1}^{i} a[k]$

  s_1+s_2即为最优

最大子段和算法进一步改进

```
void MaxSum(int *a，int left，int right)
{
  int sum=0;
  if (left==right) sum=a[left]>0?a[left]:0;
  else{ int center=(left+right)/2;
    int leftsum=MaxSubSum(a,left,center);
    int rightsum=MaxSubSum(a,center+1,right);
    int s1=0;      int lefts=0;
    for(int i=center;i>=left;i--){
      lefts+=a[i];   if (lefts>s1) s1=lefts;
    }
    int s2=0;  int rights=0;
    for (int i=center+1; i<=right; i++){
    rights+=a[i];   if (rights>s2) s2=rights;
    }
   sum=s1+s2;
   if (sum<leftsum) sum=leftsum;
   if (sum<rightsum) sum=rightsum;}
return sum;
}
```



最大子段和分治算法分析

- 算法所需的计算时间T(n)满足典型的分治算法递归式：

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

- 基于主方法和主定理
  - $T(n)=O(n\log n)$

最大子段和动态规划算法

- 若记 $b[j]=\max\limits_{1\le i\le j}\left\{\sum\limits_{k=i}^{j}a[k]\right\}$，$1\le j\le n$，则所求最大子段和为

$$\max_{1\le i\le j\le n}\sum_{k=i}^{j}a[k]=\max_{1\le j\le n}\max_{1\le i\le j}\sum_{k=i}^{j}a[k]=\max_{1\le j\le n}b[j]$$

- 由b[j]定义：
  - 当$b[j\text{-}1]>0$，$b[j]=b[j\text{-}1]+a[j]$
  - 否则，$b[j]=a[j]$

$$\Rightarrow b[j]=\max_{1\le j\le n}\{b[j-1]+a[j],a[j]\}$$

- 据此，可设计出求最大子段和的动态规划算法

最大子段和动态规划算法

```
int MaxSum(int n，int *a)
{
    int sum=0, b=0;//初始化最大子段和为0，b[0]=0
    for (int i = 1; i <= n; i++){
        if (b>0)  b+=a[i];
        else  b=a[i];
        if (b>sum) sum=b;//更新当前找到的最大子段和
    }
    return sum;
}
```

算法时间复杂度$O(n)$

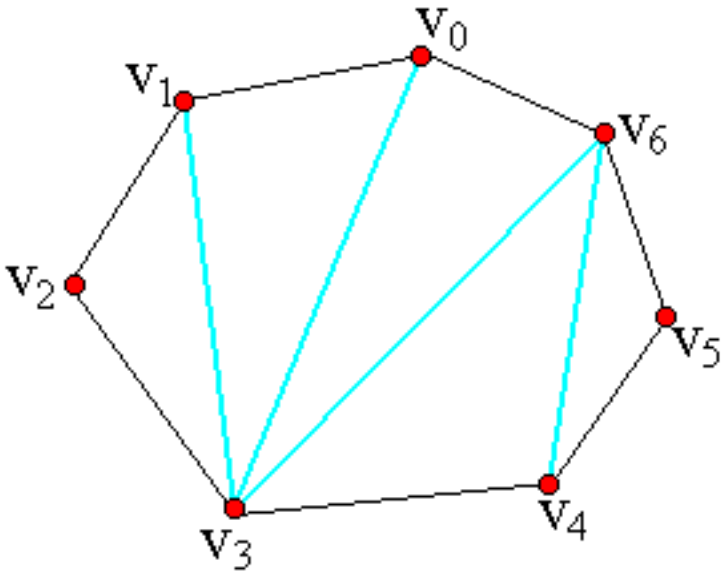
凸多边形最优三角剖分

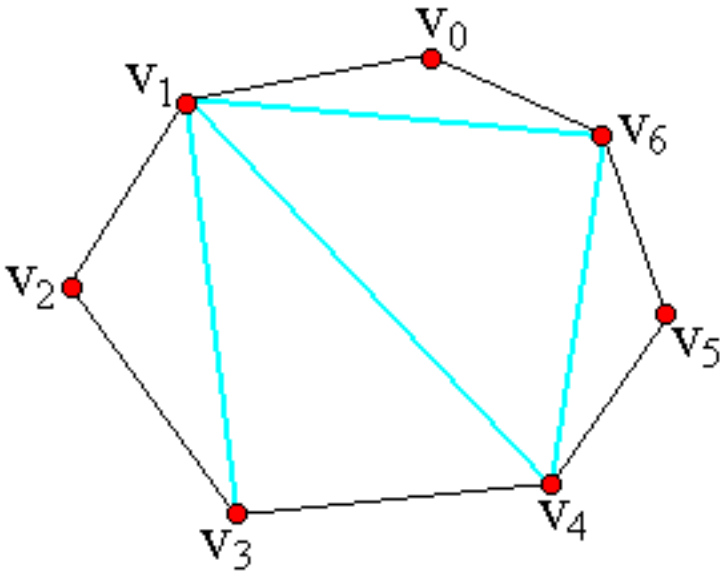
- 凸多边形的特点：
  - 1围在多边形内的所有点
  - 2多边形本身构成边界
  - 3其余部分构成多边形外部
- 凸多边形边界上/内部任意两点所连成的直线段上所有点均在凸多边形的内部或边界上
- 用多边形顶点的逆时针序列表示凸多边形，即$P=\{v_0,v_1,\ldots,v_{n-1}\}$表示具有n条边的凸多边形
- 若v_i与v_j是多边形上不相邻的2个顶点，则线段v_iv_j称为多边形的一条弦。弦将多边形分割成2个多边形$\{v_i,v_{i+1},\ldots,v_j\}$和$\{v_j,v_{j+1},\ldots v_i\}$

凸多边形最优三角剖分

- 多边形的三角剖分是将多边形分割成互不相交的三角形的弦的集合T
- 给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得即该三角剖分中诸三角形上权之和为最小

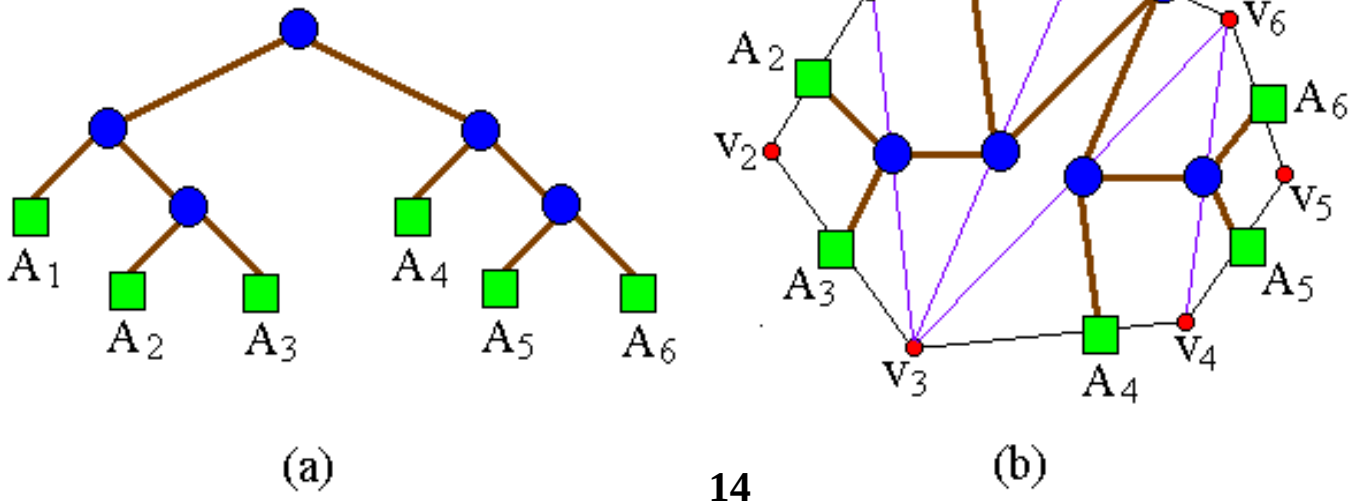
凸多边形最优三角剖分





三角剖分的结构及其相关问题

- 一个表达式的完全加括号方式相应于一棵完全二叉树，称为表达式的语法树
  - 例如，完全加括号的矩阵连乘积 $((A_1(A_2A_3))(A_4(A_5A_6)))$所相应的语法树如图 (a)所示

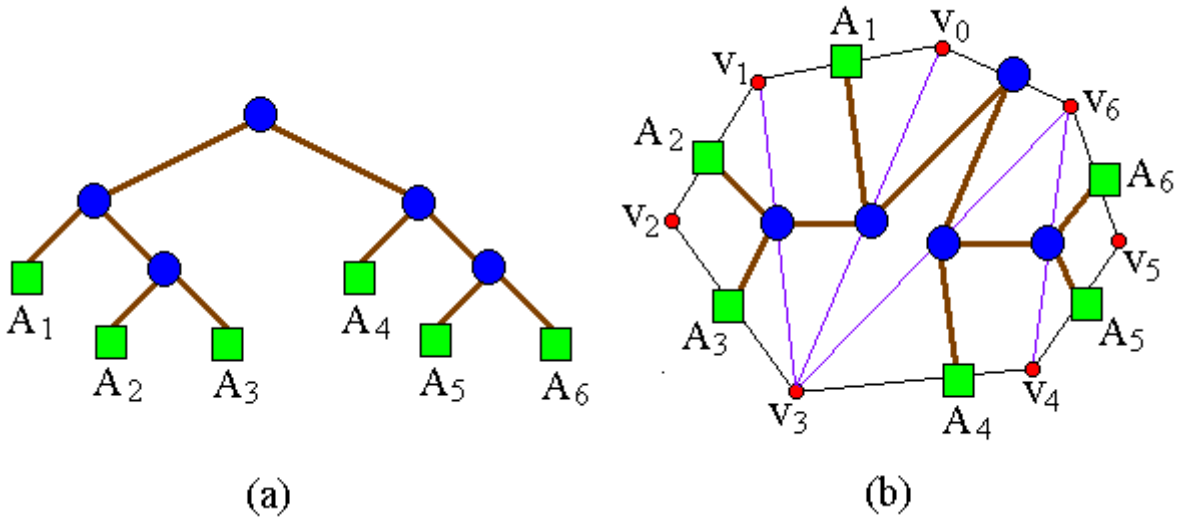


三角剖分的结构及其相关问题

- 凸多边形$\{v_0,v_1,\ldots v_{n-1}\}$的三角剖分也可以用语法树表示。例如，图 (b)中凸多边形的三角剖分可用图 (a)所示的语法树表示

三角剖分的结构及其相关问题

- 矩阵连乘积中的每个矩阵A_i对应于凸多边形中的一条边$v_{i-1}v_i$。三角剖分中的一条弦v_iv_j，$i<j$，对应于矩阵连乘积$A[i+1:j]$



$v_0v_3v_6$将原凸多边形分为多边形$\{v_0 \ldots v_3\}$ $\cup \{v_3 \ldots v_6\}$和$\{v_0v_3v_6\}$三角形。

最优三角剖分最优子结构性质

- 凸多边形的最优三角剖分问题有最优子结构性质
- 事实上，若凸$(n+1)$边形$P=\{v_0,v_1,\ldots,v_n\}$的最优三角剖分T包含三角形$v_0v_kv_n$，$1\le k\le n-1$，则T的权为3个部分权的和：三角形$v_0v_kv_n$的权，子多边形$\{v_0,v_1,\ldots,v_k\}$和$\{v_k,v_{k+1},\ldots,v_n\}$的权之和
- 可以断言，由T所确定的这2个子多边形的三角剖分也是最优的。因为若有$\{v_0,v_1,\ldots,v_k\}$或$\{v_k,v_{k+1},\ldots,v_n\}$的更小权的三角剖分将导致$T$不是最优三角剖分的矛盾

最优三角剖分的递归结构

- 定义$t[i][j]$，$1 \le i < j \le n$为凸子多边形$\{v_{i-1}, v_i, \ldots, v_j\}$的最优三角剖分所对应的权函数值，即其最优值。为方便起见，设退化的多边形$\{v_{i-1}, v_i\}$具有权值0。据此定义，要计算的凸$(n+1)$边形P的最优权值为$t[1][n]$
- $t[i][j]$的值可以利用最优子结构性质递归地计算。当$j-i \ge 1$时，凸子多边形至少有3个顶点。由最优子结构性质，$t[i][j]$的值应为$t[i][k]$的值加上$t[k+1][j]$的值，再加上三角形$v_{i-1}v_kv_j$的权值，其中$i \le k \le j-1$。由于在计算时还不知道k的确切位置，而k的所有可能位置只有$j-i$个，因此可以在这$j-i$个位置中选出使$t[i][j]$值达到最小的位置

最优三角剖分的递归结构

- 由此，t[i][j]可递归地定义为：

$$t[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

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V1和V2；
  - (2)用一个新的顶点取代边E以及由E连接着的2个顶点V1和V2。将由顶点V1和V2的整数值通过边E上的运算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

多边形游戏

- 设所给的多边形的顶点和边的顺时针序列为op[1],v[1],op[2],v[2],…,op[n],v[n]
- 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)

多边形游戏

- 对于2个子链p(i，s)和p(i+s，j-s)
- 设m1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
- 设m2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

多边形游戏的最优子结构性质

- 根据上述定义有a≤m1≤b，c≤m2≤d
  - (1)当op[i+s]='+'时，显然有a+c≤m≤b+d
  - (2)当op[i+s]='*'时，有min{ac，ad，bc，bd}≤m≤max{ac，ad，bc，bd}
- 因此，主链的最大值和最小值可由子链的最大值和最小值得到
- 可以根据上述分析递归求解

图像压缩

- 计算机中常用像素点灰度值序列$\{p_1,p_2,\ldots,p_n\}$表示图像，p_i $(1\leq i\leq n)$表示像素点i的灰度值
- 一般灰度值0-255
- 需要8位表示一个像素

图像压缩

- 图像的变位压缩存储格式将所给的像素点序列$\{p_1,p_2,\ldots,p_n\}$，$0\le p_i\le 255$ 分割成m个连续段$S_1,S_2,\ldots,S_m$
- 第i个像素段S_i中$(1\le i\le m)$，有l[i]个像素,且该段中每个像素都只用b[i]位表示

图像压缩

- 设 $t[i]=\sum_{k=1}^{i-1} l[k]$ 则第i个像素段S_i为

$$S_i=\{p_{t[i]+1},\cdots,p_{t[i]+l[i]}\}$$

- 设 $h_i=\left\lceil \log\left(\max_{t[i]+1\le k\le t[i]+l[i]} p_k+1\right)\right\rceil$ ，则$h_i \le b[i] \le 8$，因此需要用3位表示$b[i]$
- 如果限制$1\le l[\mathrm{i}]\le 255$，则需要用8位表示$l[\mathrm{i}]$

图像压缩

- 因此，第i个像素段所需的存储空间为l[i]*b[i]+8+3位
- 按此格式存储像素序列$\{p_1,p_2,\cdots,p_n\}$，需要 $\sum_{i=1}^{m} l[i]*b[i]+11m$ 位的存储空间。

- 图象压缩问题要求确定像素序列$\{p_1,p_2,\cdots,p_n\}$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

图像压缩的最优子结构性质

- 设$l[i]$，$b[i]$，$1 \le i \le m$是$\{p_1, p_2, \ldots, p_n\}$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
- 即图象压缩问题满足最优子结构性质

图像压缩

- 设$s[i]$，$1 \le i \le n$，是像素序列$\{p_1, p_2, \ldots, p_i\}$的最优分段所需的存储位数
- 由最优子结构性质易知：

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

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

- 可根据以上分析进行求解

0-1背包问题

- 给定n种物品和一背包。物品i的重量是w_i，其价值为v_i，背包的容量为C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?
- 0-1背包问题是一个特殊的整数规划问题。

$$\max \sum_{i=1}^{n} v_i x_i \qquad \begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

0-1背包问题

设所给0-1背包问题的子问题 $\max \sum_{k=i}^{n} v_k x_k$

$$\begin{cases} \sum_{k=i}^{n} w_k x_k \le j \\ x_k \in \{0,1\}, i \le k \le n \end{cases}$$

的最优值为$m(i,j)$，即$m(i,j)$是背包容量为j，可选择物品为i, $i+1, \ldots, n$时0-1背包问题的最优值。由0-1背包问题的最优子结构性质，可以建立计算$m(i,j)$的递归式如下。

$$m(i,j) = \begin{cases} \max\{m(i+1,j), m(i+1,j-w_i)+v_i\} & j \ge w_i \\ m(i+1,j) & 0 \le j < w_i \end{cases}$$

$$m(n,j) = \begin{cases} v_n & j \ge w_n \\ 0 & 0 \le j < w_n \end{cases}$$

0-1背包问题

算法复杂度分析：

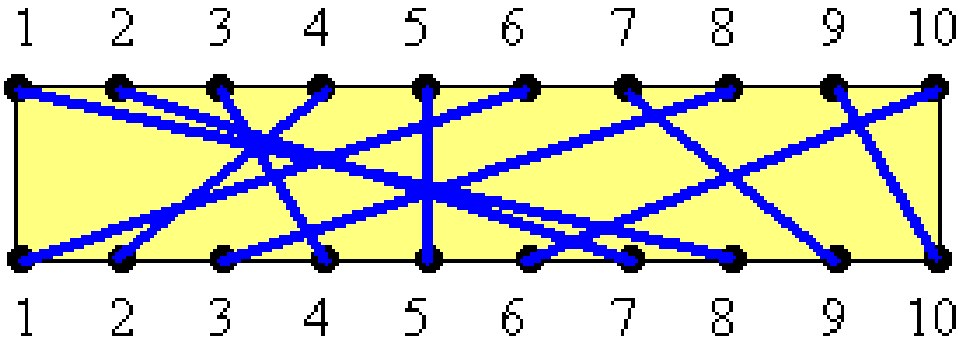
从$m(i,j)$的递归式容易看出，算法需要$O(nc)$计算时间。当背包容量c很大时，算法需要的计算时间较多。例如，当$c>2^n$时，算法需要$\Omega(n2^n)$计算时间。

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$(i,\pi(i))$将上端接线柱与下端接线柱相连，如图所示。其中$\pi(i)$是$\{1,2,\ldots,n\}$的一个排列。
- 导线$(i,\pi(i))$称为该电路板上的第i条连线。
- 对于任何$1\le i<j\le n$，第i条连线和第j条连线相交的充分且必要的条件是$\pi(i)>\pi(j)$。

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 $Nets=\{(i,\pi(i)),1\leq i\leq n\}$的最大不相交子集。



电路布线的最优子结构性质

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

(2)当i>1时，

2.1 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)。

2.2 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$(i,\pi(i)) \notin MNS(i,j)$则对任意(t,π(t)) ∈MNS(i,j)有t<i。从而 $MNS(i,j) \subseteq N(i-1,j)$，因此，Size(i,j)≤Size(i-1,j)。另一方面 $MNS(i-1,j) \subseteq N(i,j)$，故又有Size(i,j)≥Size(i-1,j)，从而Size(i,j)=Size(i-1,j)。

电路布线

- 根据最优子结构性质，可得

(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

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