# 13章 NP-完全问题

# 13. NP-完全问题

- N-Nondeterministic(算法)
- P-Polynomial (时间)
- Complete(封闭的类)

# 13.1 某些NP完全问题的例子

- 问题1 图着色问题
  - 判定问题：是否存在不超过k种颜色的着色方案?
  - 优化问题：求图的最小着色数和着色方案
- 问题2 作业调度问题
  - 判定问题：是否存在罚款额不超过k的作业调度?
  - 优化问题：求最小罚款额调度

- 问题3 Bin packing问题
- 假设有n种物品，它们的尺寸分别为$s_1,\ldots,s_n$，$0<s_i\leq 1$另有任意多个箱子(Bins)，箱子的容量为1，
  - 优化问题：求使用最小箱数的装箱方法
  - 判定问题：任意给定k，是否存在一种装箱方法使用的箱子数≤k?

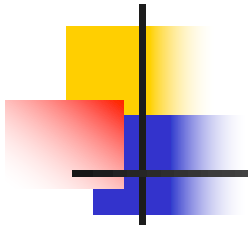
- 问题4 背包问题
  - 判定问题：是否存在效益值至少为k的可行子集?
  - 优化问题

- 问题5 子集和数问题$s_1,s_2,\cdots,s_n,C$
  - 判定问题：是否存在和数等于C的子集?
  - 优化问题：求≤C的最大子集和数.
  - 可归约为背包问题: $p_i=w_i$.

- 问题6 CNF(合取范式)-可满足问题
  - 判定问题：是否存在真假赋值使得该CNF为真.合取范式例: $(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22}) \wedge (x_{31} \vee x_{32} \vee x_{33})$
- 问题7 Hamiltonian 回路
  - 判定问题
- 问题8 TSP(旅行商)问题
  - 判定问题：任意给定一整数k，是否存在一长度不超过k的Hamiltonian回路？
  - 优化问题

- 问题9:节点覆盖(见书,图13.5, 258页)
  - 判定问题：给定无向图G和正整数k,是否存在一k节点覆盖.
  - 优化问题：最小节点覆盖
- 问题10: 给定一无向图G, k-独立集;最大独立集.
- 问题11: 给定一无向图G , k-集团;最大集团.
- 问题10和11可互相转化: 边互补图(图13.5).
- 目标函数取整数值时,优化值问题和判定问题等价

  我们可用二分查找从判定问题解得到优化值的问题的解.



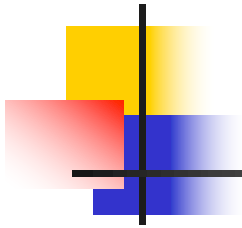
# 13.2 类P与类NP

# 类P

- 类P
  - 算法输入为问题实例的二进制编码. 定义该0/1字符串的长度为输入长度. 例如n个节点和m条边的图的长度为Θ(mlogn)(见图13.1).
  - 多项式界(定义13.2)

如一算法的最坏情形时间复杂度T(n)=O(p(n)), 其中p(n)为输入长度n 的多项式, 则称该算法有多项式界.

- 如一个问题有多项式界的算法,则称该问题有多项式界.



- 类P的定义
  - 一个算法解决判定问题指: 对该判定问题的yes实例, 算法要停机并给出yes回答.对no实例算法要停机并给出no的回答.
  - 有多项式界的判定问题组成的类称为类P.
  - 多项式的相加,相乘及复合仍为多项式; 所以一个有多项式界的算法调用另一有多项式界的算法构成的算法仍有多项式界.

# 类NP

- 不确定的算法：对给定输入字符串w,

  1. Guessing

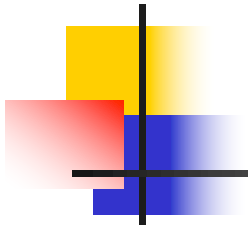
  *不确定地*写一个称为”certificate” (guessed 解)的字符串c .

  c 实际上是可行解的一种表示; 例如, 表示背包问题的n元组, 表示图的字符串或邻接矩阵.

  2. checking

  使用一确定的有多项式界的算法验证c是否为问题的答案. 如果是给出yes, 反之, 不回答.

- 多项式界指: 写c 和验证的时间为O(q(|w|)), q( )为某一多项式, w为输入长度.
- 不能实现的、虚构的算法



- **Void** nondetA(String input)

  **String** s=genCertif();

  **boolean** checkOK=verifyA(input,s)

  **if** (checkOK)

  Output “yes“

  **return**

- **checOK**为**false**时不作反应**.**

- “不确定” 指:对同一输入w, 算法使用任意多个不同的c ,进行验证. 对不同c 的这些计算,可以看作是同时进行的.
- 一个不确定算法解决判定问题指:对输入w, 如果它有解, 不确定算法一定会写出一个c ,使得验证阶段能通过, 并返回一个yes回答.
- 如果一个判定问题有不确定的多项式界的算法则称它属于类NP.

# 例13.1图着色问题

- Input: (着色数)k, (节点数)5 ,(图的边)(1,2)(1,4)...见图13.1.
- Guessing 指写出长度为n(节点数)的字符串,例如,RGRBG, RGRB, RBYGO, RGRBY, R%*$@等.至少有$k^n$个 “guessings”.
- Checking 指检查一guessed字符串是否合法及是否是一个k-着色.
- 如果写字符串的时间是$O(p_1(n))$,checking时间是$O(p_2(n))$;而且$p_1(n)$, $p_2(n)$是n的多项式(从而也是输入长度的多项式),则该不确定算法有多项式界.
- 如果输入的图能k着色则不确定算法停机并给出yes回答.

# P=NP?

- 类**NP**：有不确定的多项式界算法的判定问题构成的类称为**NP**类。
- **P=NP?**是计算机科学中最大的问题之一

# 13.3 The size of input

- Problem13.9-是否存在j, k>1使得n= jk?即n 是否为一合数?

```
factor=0;
for (j=2;j<n;j++)
      if ((n mod j)==0)
            factor=j;
            break;
return factor
```

- (n mod j)计算时间为$\Theta(\log^2 n)$(bit级运算).
- 算法的复杂度为$\Theta(n\log^2 n)$.但n是输入长度 $m=\log_2 n$ 的指数函数.所以它是指数复杂度算法.
- 2002年证明了: ”n是否为一素数?” 属于P.

# The size of input

- 背包问题输入长度m为

  $m=\Theta(\log_2 n+\log_2 c+\Sigma\log_2 p_i+\Sigma\log_2 w_i)$

- 假定$p_i=O(c)$ $w_i=O(c)$,则$m=O(n\log_2 c)$
- $\Theta(nc)$不能以m的多项式加以限界, 即

  $nc=O(m^k)$

  不成立, 因为$nc/(n\log_2 c)^k\to\infty(c\to\infty)$, 对所有常数k.
- 除了涉及数的计算外, 以前我们分析的多项式复杂度算法,在以字符串为输入时,仍是多项式复杂度算法.

# 13.3.1多项式约化

- 问题P 可多项式地约化为问题Q ,如果
- 存在一个有多项式界的确定性算法T,使得：
  - 对每个输入字符串x, T产生一字符串T(x).
  - x是P的合法输入且P对x有yes答案当且仅当T(x) 是Q的合法输入且有yes答案
- 证明多项式约化关键在“当且仅当”
- 问题 P 可多项式地约化为 Q, 表示为：

$$P \leq_p Q$$

# 多项式约化

- 定理13.3 If $P \leq_p Q$且Q在**P**中则P也在**P**中
  - 存在解Q的算法有多项式界q,
  - 设T的复杂为多项式p.则T(x)的长度O(p(|x|))
  - 对输入T(x)所需时间为O(q(p(|x|)))
  - 解P的复杂度为O(p(|x|)+q(p(|x|)))
- 可约化的意义:Q至少和P一样“难”
- 约化关系有传递性

# 子集和数可约化为调度问题

- 多项式变换
  - 子集和数问题的实例:

    $s_1,s_2,\text{---},s_n,C$;假定$S=\Sigma_{1\le i\le n}s_i>C$
  - 对应调度问题实例

    $p_i=t_i=s_i$ ,$d_i=C$, $k=S-C$
- if部分:子集和数有解则调度问题有解
- only if:假定上述调度问题有罚款额≤k的解
  - 该可行调度的执行时间$t_i$之和≤C(可行性)
  - 又因$t_i=p_i=s_i$,所以该可行调度对应罚款额=$S-\Sigma p_i$ =$S-\Sigma t_i\ge S-C=k$
  - 所以其罚款额＝k，而且被调度的作业的时间之和＝C

# NP-难度和NP-完全问题

- 问题Q是NP-难度问题。如果：

  每个NP问题都可多项式地约化为问题 Q.

- 问题 Q 是 *NP*-完全问题. 如果:
  - 它是NP问题, 同时它还是NP-难度问题.
  - 所有NP-完全问题,相对于多项式约化关系,是自反,对称,传递的,即构成一个闭类.
  - 找到一个问题的多项式算法则**P=NP**
- 有NP-难度问题但不知它是否在**NP**类内 (第$k_{th}$重子集问题)

# Problems-unknown in NP

- $K_{th}$重子集问题:任给定n+2 个正整数 $c_1$,---$c_n$, k, L; 是否存在{1,2,---,n} 的k 个不同子集$S_1$,---$S_k$ 使得对所有i=1,---,k 有

$$\sum_{j \in S_i} c_j \geq L$$

- Σ部分称为子集的重量, 重量排序第k的子集.
- 当$k=2^{n-1}$时,表示可行解的字符串的长度有指数的长度.我们不知道该问题是否在**NP**中.
- 图G的最大集团的节点数是否=k? 上述问题是否在NP?也是未知的! (验证最大集团,不能在多项式时间内做到)

# CNF-satisfiablity问题是NP-完全问题

- 定理13.5 CNF-satisfiablity问题是NP-完全问题
- 这是著名的Cook定理
- Cook 定理的推论:如果CNF-可满足问题有多项式界的算法，则P=NP.

# NP-完全问题证明

- 证明问题Q是NP-完全问题的步骤：

  (1)选择一已知的NP-完全问题P。

  (2)证明P可多项式的约化为 Q
- 背包问题属于NP＝>子集和数属于NP
- 子集和数问题属于NP＝>调度问题属于NP
- 不计其数的这种推导.

# 13.3.3

- 13.3.3What makes a problem hard?
- 限定问题的一般性:实际应用中有特殊性,有可能找到多项式算法
- 例：Hamiltonian回路
  - 顶点度<=2时，Hamiltonian回路问题有多项式算法
- 工程中有灵活性,以某种方式优化是NP-难度问题;但以另一方式提出问题可能不是.
- 了解“难问题”的特点

# 13.3.3续

- 3-满足问题仍为NP-完全问题,但2-满足问题有多项式算法
- 集团问题,当顶点度<=常数d时属于类P
- 平面图集团问题属于类P，因为平面图至多有4-集团
- 实际有意义的做法是提出合理的限制条件和求近似解, 研究启发式算法.

# 13.3.4

- 优化问题和判定问题
- 3种问题：

  判定问题

  求优化值问题

  优化解问题
- 优化值问题有多项式算法，则判定问题有多项式算法

# 13.4 近似算法

- 返回次优解的算法。这种算法经常可以通过启发式方法得到，例如：贪心法。
- 近似算法必须是多项式时间算法。
- 为量度近似解对优化解的近似程度定义以下术语
  - FS(I) 是输入I的可行解集。
  - Val(I,x):实例I的可行解x的目标函数值
  - opt(I):实例I的优化解的值

# 13.4近似算法

- 设A为一近似算法,令A(I)为输入I时该算法输出的可行解
- 式(13.3)和(13.4)分别对极小化和极大化问题定义了一类量度近似性能的指标$r_A(I)$

$$r_A(I) = \frac{val(I, A(I))}{opt(I)} \geq 1 \qquad 13.3$$

$$r_A(I) = \frac{opt(I)}{val(I, A(I))} \geq 1 \qquad 13.4$$

# 续

- 式(13.5)定义的$R_A(m)$为最坏情形$r_A(I)$的值,是与输入I无关的指标

$$R_A(m) = \max\{r_A(I) \mid opt(I) = m\} \qquad (13.5)$$

在固定优化值m下求最坏情形的比值

- 式(13.6)定义的$S_A(n)$也是一与输入独立的指标

$$S_A(n) = \max\{r_A(I) \mid size(I) = n\} \qquad (13.6)$$

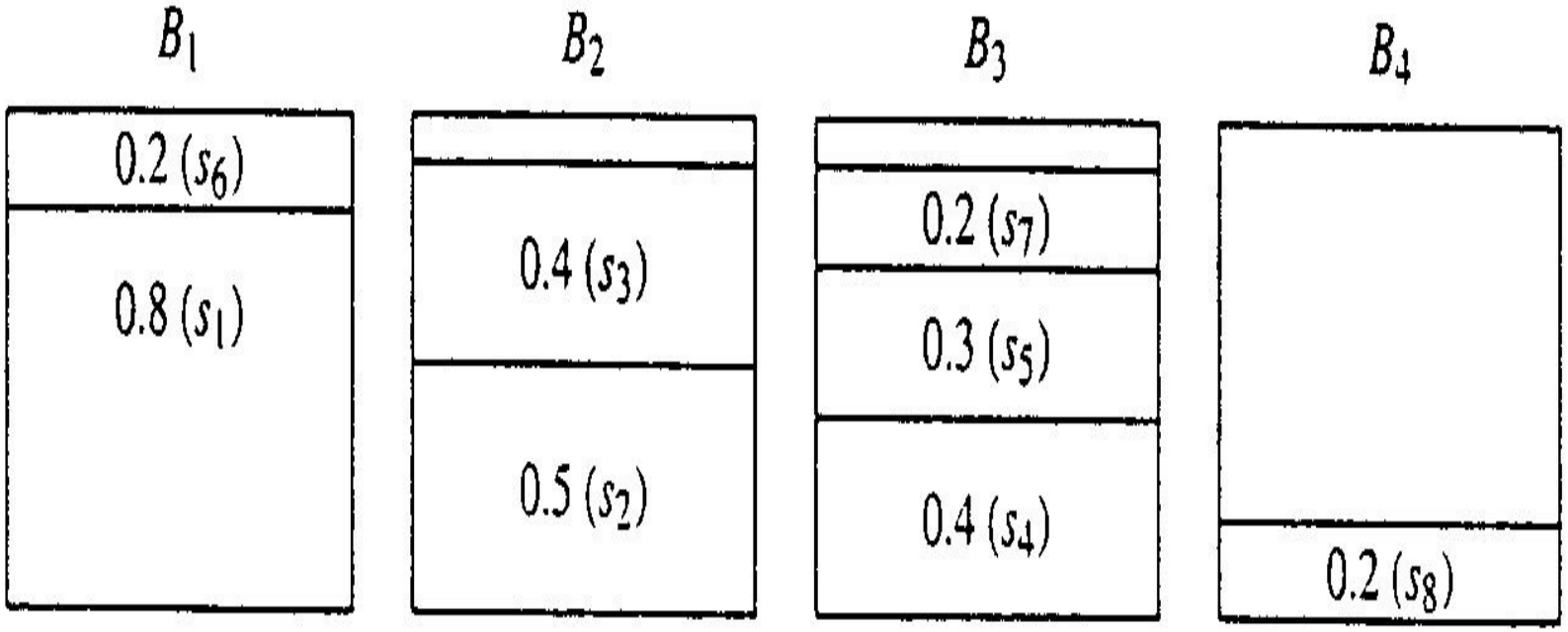
# 13.4例:Bin-Packing的近似算法

- 怎么装不同大小、不同形状的货物才能使占用的箱子数最少。该问题形式化如下：
- 装箱问题
  - 设$S = (s_1, \ldots, s_n)$
    - $0 < s_i <= 1$ ， $1 <= i <= n$
  - 将 $s_1, \ldots, s_n$ 装入尽可能少的箱子里。假定每个箱子都有容量1。
- 装箱问题是NP-难度问题
  - 搜索算法有指数的复杂度：须试所有可能的S的分划。

# 装箱问题：FFD算法（贪心法）

- 将物品按尺寸递减排序，箱子从左到右排列并尽可能放在前面的箱子里。
- 算法的时间复杂度$t(n)=\theta(n^2)$

$S = (0.8, 0.5, 0.4, 0.4, 0.3, 0.2, 0.2, 0.2)$



# 算法：装箱问题

- 输入: $S=(s_{1,\ldots,}s_n)$，$0<s_i\leq 1$，$1\leq i\leq n$. S代表货物1,...,n 的尺寸，每个箱子的容量都是1.0。
- 输出: bin[i]是放物品i的箱子号,$1\leq i\leq n$. 为了使算法简单，在装箱前，货物已经按尺寸从大到小排序。

# 装箱问题的程序

```
binpackFFd(S,n,bin)
{
    float[] used=new float[n+1];
    //used[j] 是箱子j已经使用的空间
    int i,j;
    // used[j] 初始为0，S已按降序排列S1>=S2>=…>=Sn.
    for(i=1;i<=n;i++)
   {//寻找能装下 s[i] 的箱子.
       for(j=1;j<=n;j++)
          {if(used[j]+si<+1.0)
           //+1.0， 每个箱子的容量都是1.0
            bin[i]=j;
            used[j] += si;
            break;  //装完退出循环j,装下一个箱子，继续循环i.
           }
    }
}
```

# 近似分析

- 引理13.9 :设S为算法的输入.令opt(S)为优化的装箱数.令i为第一个被FFD算法装入第opt(S)+1号箱子的物品,则$s_i$<＝1/3

  证明:(反证法)

  假设 $s_i$>1/3, 则FFD算法在装到$s_i$ 时, 前面的箱子不会如图13.7的情形. 产生的装箱情况如图13.8.(k≥0):

  k个箱子只放一个物品; opt-k个箱子放2个size>1/3的物品.

# 近似分析

- FFD算法没把物品k+1,…,i -1放在前k个箱子内(放不下), 这些物品的数目为2(opt-k)
- 尽管优化解的装箱情况和FFD不同, 但前k个物品在优化解中也必须分放在k个箱子里: 前k个物品中任2个都不能放在一个箱子内.
- 优化解中物品k+1,…,i-1,放在其余的opt-k个箱子内. 因为这些物品的尺寸均>1/3. 所以这些箱子中每个箱子都要放2个, 尽管放法和FFD不一定相同.
- 因此 $s_i$ 在优化解中无法安排,矛盾.

# 近似分析

- 引理13.10 :FFD在前opt(S)箱子装不下的物品数量至多为opt(S)-1

反证法:如有opt(S)个物品放在多用的箱内,则有:

$$\sum_{i=1}^{n} s_i \geq \sum_{i=1}^{opt(s)} b_i + \sum_{i=1}^{opt(S)} t_i = \sum_{i=1}^{opt(S)} (b_i + t_i) > opt(S)$$

但

$$\sum_{i=1}^{n} s_i \leq opt(S)$$

其中$b_i$为装在第i个箱子内物品的总量;$t_i$为前opt箱子装不下的物品的重量;按FFD算法后者不能装入第i个箱子.所有$b_i+t_i>1$.

- 定理13.11 $R_{FFD}(m)<=(4/3)+(1/3m)$;

$S_{FFD}(n)<=3/2$

# 近似分析

- FFD至多有m-1个物品放在extra箱子内,放的物品size<=1/3,所以FFD用的箱子数不超过

$$m + \lceil (m-1)/3 \rceil$$

$$r_{FFD}(S) \le \frac{m + \lceil (m-1)/3 \rceil}{m} \le 1 + \frac{m+1}{3m} \le \frac{4}{3} + \frac{1}{3m}$$

- 更强的结果为$R_{FFD}(m) \le 11/9 + 4/m$
- 对任意大的m有例子证明$R_{FFD} \ge 11/9$,即最坏情形22%的误差(较优化解多用的箱数)是紧上界(tight bound)

# Best Fit, Next Fit Heuristics

- Best Fit :尺寸s的物品放入满足条件used[j]+s≤1,且used[j]最大的箱内
- 当物品按尺寸s的非增顺序排列时Best Fit 策略和FFD产生的装箱方法差不多相同
- 图13.6
- Next Fit :物品不排序,所以可用于在线应用;如果下一物品不能放入当前使用的箱子,则使用一个新箱子.见图13.9
- Next Fit 至多用2opt(S)个箱子:产生的方案中相邻的2个箱子装入的物品尺寸>1.

# 13.6背包和子集和数问题

- 当所有$p_i=s_i$时，背包问题转化为子集和数的优化问题
- 算法13.2为上述简化的背包问题的近似算法$sKnap_k$，类似k－优化的背包问题的贪心算法。
- 定理13.13 $R_{sKnapk}(m)$和$S_{sKnapk}(n)$均$\leq 1+1/k$

# 续

- 定理13.13证明要点：设T是优化解中k个重量最大的物品构成的子集, k-优化算法考虑过先装这k 种物品的情形; 设 j是优化解中第一个未被贪心法在这种情形加入T中的物品的下标.

  Opt(I)=m>=k个最重的重量之和
  $+S_j$>=(k+1)$S_j$ ,所以$S_j$ <=m/(k+1)
- 因$S_j$ 被贪心法拒绝，所以
  Val(sKnapk(I))+$S_j$>C>=m ,所以
  Val(sKnap$_k$(I))>m-$S_j$>=m-(m/(k+1))
  =mk/(k+1)
- r(I)=m/Val<(k+1)/k=1+1/k

# 图着色

- 算法13.3

  for (i=1;i≤n;i++)

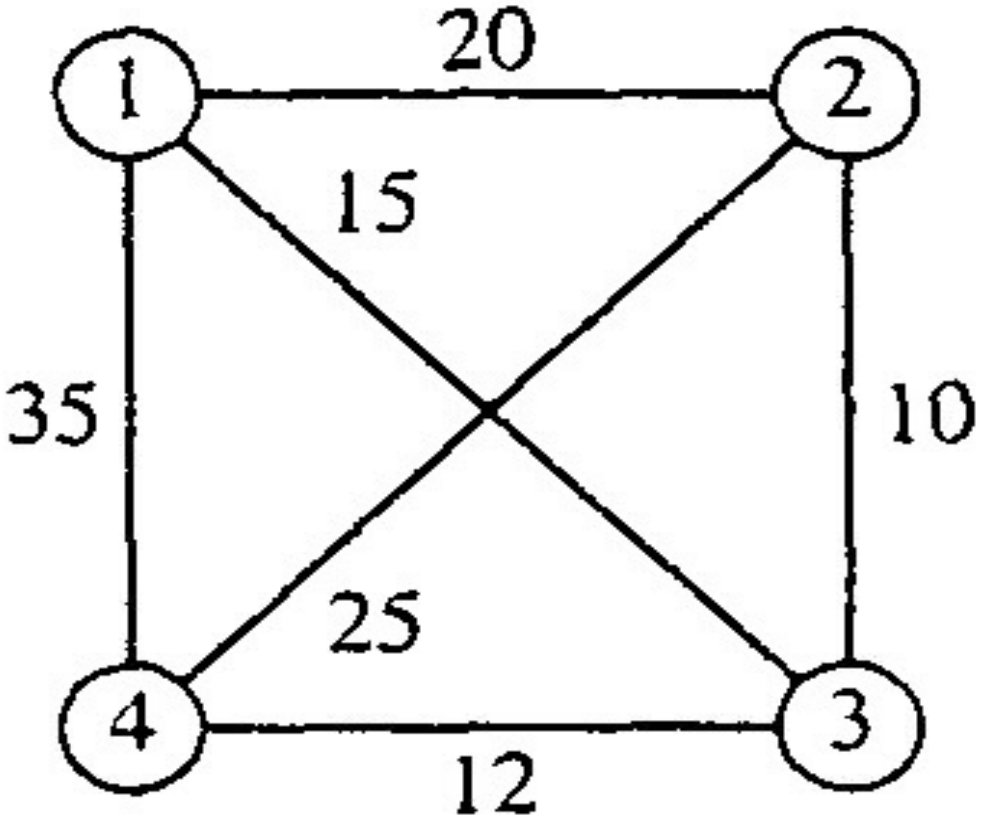
  for (c=1;c≤n;c++)

  {如没有与$v_i$ 相邻的顶点有着色c ,对$v_i$ 着色c 并break;}

- Fig.13.11 如按先a类顶点后b类顶点着色算法13.3只需2种颜色;如交叉进行则需k种颜色
- $R_{SC}(2)=\infty;S_{SC}(n)\geq n/4$ (k=n/2,opt=2)

# 旅行商问题

- 给定一个带权重的完全图
- 找具有最小权重的周游路线（通过所有顶点的环）。



# 旅行商问题的近似算法

- 最近邻居策略

  NearestTSP(V, E, W)

  选择任一顶点s作为周游路线C的起点

  v = s;

  While 有顶点不再C中:

  {选择有最小权重的边vw，其中 w 不在C中.

  将边vw加入C中;

  v = w;}

  将边vs加入C.

  return C;

- 时间复杂度$t(n) = O(n^2)$
  - n 是顶点的数量

# 续

- 最短链路策略:与C中边不构成环路且不构成与v或w伴随的第3条边(无向图)最小权值边(v,w)
- 见591页算法
- 上述两个算法都不保证产生优化解，而且对其近似程度也不能给出界
- 定理13.22 设A是TSP问题的近似算法，如对任何输入I有$r_A(I)<=c$，则P＝NP
- 从该定理可看出TSP的难度