第8章
NP完全性理论

基本概念

计算复杂性理论是理论计算机科学的分支学科，使用数学方法对计算中所需的各种资源的耗费作定量分析，并研究各类问题之间在计算复杂程度上的相互关系和基本性质，是算法分析的理论基础。

如何区分一个问题是“难”是“易”？

通常将可在多项式时间内解决的问题看作是“易”解问题，而将需要指数时间解决的问题看作是“难”解问题。

图灵机

- 一条无限长的纸带 TAPE。纸带被划分为一个接一个的小格子，每个格子上包含一个来自有限字母表的符号，字母表中有一个特殊的符号表示空白。纸带上的格子从左到右依此被编号为 0，1，2，…，纸带的右端可以无限伸展。
- 一个读写头 HEAD。该读写头可以在纸带上左右移动，它能读出当前所指格子上的符号，并能改变当前格子上的符号。
- 一套控制规则 TABLE（程序、状态转移函数）。它根据当前机器所处的状态以及当前读写头所指的格子上的符号来确定读写头下一步的动作，并改变状态寄存器的值，令机器进入一个新的状态。

图灵机

- 图灵机 M = (Q，Σ，Γ，δ，q0，qaccept，qreject)，Q是状态集合，Σ是输入字母表，Γ是带字母表，以如下方式运作：
  - 开始的时候将输入符号串从左到右依此填在纸带的格子上，其他格子保持空白(即填以空白符)。
  - M 的读写头指向第 0 号格子，M 处于状态 q0。
  - 机器开始运行后，按照转移函数 δ 所描述的规则进行计算
    - 若当前机器的状态为 q，读写头所指的格子中的符号为 x，设 δ(q，x)= (q'，x'，L)，则机器进入新状态 q'，将读写头所指的格子中的符号改为 x'，然后将读写头向左移动一个格子。
    - 若在某一时刻，读写头所指的是第 0 号格子，但根据转移函数它下一步将继续向左移，这时它停在原地不动。换句话说，读写头始终不移出纸带的左边界。

图灵机

- 图灵机 M = (Q，Σ，Γ，δ，q0，qaccept，qreject) 以如下方式运作：
  - 机器开始运行后，按照转移函数 δ 所描述的规则进行计算
    - 若在某个时刻 M 根据转移函数进入了状态 qaccept， 则它立刻停机并接受输入的字符串；
    - 若在某个时刻 M 根据转移函数进入了状态 qreject， 则它立刻停机并拒绝输入的字符串。
  - 注意，转移函数 δ 是一个部分函数， 换句话说对于某些 q，x， δ(q，x)可能没有定义， 如果在运行中遇到下一个操作没有定义的情况， 机器将立刻停机。

非确定性图灵机

- **非确定型图灵机**和确定型图灵机的不同之处在于，在计算的每一时刻，根据当前状态和读写头所读的符号，机器存在多种状态转移方案，机器将任意地选择其中一种方案继续运作，直到最后停机为止。具体而言，其状态转移函数为

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$$

- 例如，对于非确定图灵机M的当前状态为q，当前读写头的状态x， $\delta(q, x)= \{(q_1,x_1,d_1),\ldots, (q_k,x_k,d_k)\}$

非确定性算法

- **确定性算法**

  在算法中使用的每个操作的结果唯一确定，算法操作的结果也唯一确定。

- **非确定性算法**

  算法操作结果不唯一，而是来自可能值的集合。

引入三个函数：

- Choice(S) …选择集合S中的任一元素；
- Failure() …失败信号，表示失败的终止；
- Success() …成功信号，表示成功的终止；

- 当且仅当不存在产生成功信号的一套选择时，非确定性算法才不能成功终止；
- 三个函数计算时间均为O(1)；
- 按此方式执行非确定性算法的机器称为非确定性机。

非确定性算法

例1: 给定集合A[1:n](n≥1)中寻找某一元素x，确定使A[j]=x的索引j，如果x不在A中，则 j=0。

算法1:

```
int j=Choice(1,n);
if(A[j]==x)
        System.out.println(j);
        Success();
System.out.println(0);
Failure();
```

非确定性算法

例2：设A[i](1≤i≤n)是正整数组成的数组，非确定性算法Nsort(A, n)按非降序排序数组并按次序输出数组元素。

算法2：

```
void Nsort (int A[],int n){
    int B[size], i, j;
    for (i=1; i<=n; i++)   B[i]=0;
    for (i=1; i<=n; i++)
        j = Choice(1,n);
        if (B[j])  Failure();  else   B[j] = A[i];
    for (i=1; i<=n-1; i++)     if (B[i] > B[i+1])  Failure();
    for (i=1;i<=n; i++)   System.out.print(B[i]);
    Success();
}
```

判定算法与最优化算法

- 判定问题、判定算法
  - 答案非0即1的所有问题称为判定问题
  - 求解判定问题的算法称为判定算法
- 最优化问题、最优化算法
  - 确定代价函数最优值的问题称为最优化问题
  - 求解最优化问题的算法称为最优化算法
- 判定问题与最优化问题
  - 最优化问题可在多项式时间求解，则对应判定问题也可以
  - 判定问题不能在多项式时间求解，则对应最优化问题也不能

判定算法与最优化算法

例3：背包判定问题是确定是否存在$x_i(1\le i\le n)$的0/1赋值序列，满足$\sum p_i x_i \ge r$和$\sum w_i x_i \le m$。如果背包判定问题不能在多项式时间求解，则最优化问题也不能。

算法3 [背包判定问题]：

```
void DKP (int p[], int w[], int n, int m, int r, int x[]){
    int W=0, P=0;
    for(int i=1; i<=n; i++)
        x[i]  =  Choice(0,1);
        W += x[i]*w[i];  P += x[i]*p[i];
    if((W>m) || (P<r))  Failure();
    else Success();
}
```

非确定性算法的复杂度

- 在给定的输入下，如果存在序列使算法成功终止，则所需要的时间是成功终止所需的最小步数。如果不能成功终止，所需时间O(1)。
- 如果对于能让算法成功终止的所有大小为n($n \geq n0$)的输入，算法所需时间至多是cf(n)，其中c和n0都是常数。则非确定性算法复杂度为O(f(n))。

例如：算法3解背包判定问题的时间复杂度为O(n)，如果输入用二进制表示长度为q，则时间复杂度为O(q)。

问题变换与计算复杂性归约

假设有2个问题A和B，将**问题A变换为问题B**是指：

1) 将问题A的输入变换为问题B的适当输入；
2) 解出问题B；
3) 把问题B的输出变换为问题A的正确解。

若用O(τ(n))时间能完成上述变换的第(1)步和第(3)步，则称问题A是τ(n)时间可变换到问题B，且简记为$\mathbf{A} \propto_{\tau(n)} \mathbf{B}$。其中的n通常为问题A的规模(大小)。

当τ(n)为n的多项式时，称问题A可在多项式时间内变换为问题B。特别地，当τ(n)为n的线性函数时，称问题A可线性地变换为问题B。

问题变换与计算复杂性归约

问题的变换与问题的计算复杂性归约的关系：

命题1(计算时间下界归约)：若已知问题A的计算时间下界为T(n)，且问题A是τ(n)可变换到问题B，即$A\propto_{\tau(n)}B$，则T(n)-O(τ(n))为问题B的一个计算时间下界。

命题2(计算时间上界归约)：若已知问题B的计算时间上界为T(n)，且问题A是τ(n)可变换到问题B，即$A\propto_{\tau(n)}B$，则T(n)+O(τ(n))是问题A的一个计算时间上界。

在命题1和命题2中，当τ(n)=o(T(n))时，问题A的下界归约为问题B的下界，问题B的上界归约为问题A的上界。

问题变换与计算复杂性归约

通过问题变换获得问题的计算时间下界的例子：

(1)判别函数问题：给定n个实数 $x_1, x_2, \ldots, x_n$ ，计算其判别函数 $\prod_{i \neq j}(x_i - x_j)$。

元素惟一性问题(判定序列$\{x_1, x_2, \ldots, x_n\}$中元素互不相同)可以在O(1)时间内变换为判别函数问题。任何一个计算判别函数的算法，计算出判别函数值后，再作一次测试，判断其值是否为0，即可得到元素惟一性问题的解。由命题1即知，元素惟一性问题的计算时间下界 $\Omega(n \log n)$ 也是判别函数问题的一个计算时间下界。

问题变换与计算复杂性归约

通过问题变换获得问题的计算时间下界的例子：

(2) 最接近点对问题：给定平面上n个点，找出这n个点中距离最近的2个点。

在元素惟一性问题中，将每一个实数 x_i，1≤i≤n，变换为平面上的点（x_i，0)，1≤i≤n，则元素惟一性问题可以在线性时间内变换为最接近点对问题。

集合不相交问题

问题描述：设S1和S2是包含n个元素的集合，判定两个集合是否相交，即$S1 \cap S2 = \Phi$。

设：P1代表判定集合不相交问题，
　　P2代表排序问题。

已经知道排序问题的复杂度上界和下界都是$\Theta(n\log n)$。现在证明集合不相交问题的复杂度上界和下界也都是$\Omega(n\log n)$。

集合不相交问题

证明 集合不相交问题的复杂度上界

P1代表判定集合不相交问题，P2代表排序问题

则很容易证明$P1 \propto_{O(n)} P2$：

设P1的输入为：$S1=\{a_1, a_2, \ldots, a_n\}$，$S2=\{b_1, b_2, \ldots, b_n\}$

（1）构造P2的输入

$$X=\{(a_1,1), (a_2,1), \ldots, (a_n,1), (b_1,2), (b_2,2), \ldots, (b_n,2)\}$$

（2）对X按字典序排序，得到有序序列X’；

（3）在X’中顺序检查是否存在连续的两个元素$(x,1)$和$(y,2)$，满足$x=y$。

因为$P1 \propto_{O(n)} P2$，故$O(nlogn)+O(n)=O(nlogn)$为集合不相交问题的一个计算时间上界。

集合不相交问题

证明 集合不相交问题的复杂度下界

证明$P2 \propto_{T(n)} P1$

构造排序算法：设排序算法P2的输入为$X=\{x_1, x_2, \ldots, x_n\}$

1）构造集合不相交问题的输入： 时间O(n)

$S1=\{(x_1,1), (x_2,1), \ldots, (x_n,1)\}$，$S2=\{(x_1,2), (x_2,2), \ldots, (x_n,2)\}$

2）求解集合不相交问题P1（$S1 \cap S2=\Phi$？） 设时间为T(n)

分析：设$Y=\{y_1, y_2, \ldots, y_n\}$是X的有序排列，则$(y_i,1) \in S1$，$(y_i,2) \in S2$。任何求解P1的算法都需要比较$(y_i,2)$与$(y_{i+1},1)$，否则，用$(y_i,2)$替换S1中的$(y_{i+1},1)$将不会影响算法的比较结果。可以在求解P1时输出所比较的元素偶对$((x_i,1),(x_j,2))$。这样的偶对最多T(n)个。

集合不相交问题

3）构造X的有序排列： 时间O(n+T(n))

- 建立一个有向图G=(V,E)，V={x_1, x_2,…, x_n}。对求解P1算法输出的偶对((x_i,1),(x_j,2))，对x_i和x_j进行比较。如果$x_i<x_j$，则添加一个有向边<x_i, x_j>；否则添加<x_j, x_i>。建立图G所需时间为n+T(n)。
- 在图的顶点{x_1, x_2,…, x_n}找出最小元素y_1，然后在y_1的邻居顶点中找出最小元素y_2，以此类推。所需时间为n+T(n)。

因此构造X的有序排列过程所需时间为2n+2T(n)。

综合上述三个步骤，这个排序算法的时间为3n+3T(n)。因为排序问题的下界为Ω($n$log$n$)，所以3n+3T(n)≥Ω($n$log$n$)，即，T(n)≥Ω($n$log$n$)。故集合不相交问题的下界为Ω($n$log$n$)。

集合不相交问题

根据集合不相交问题的下界，可以推导出很多有关集合问题的下界。如:

1）求两个集合的交集

2）求两个集合的并集

3）求两个集合的差

4）元素唯一性问题

P类与NP类问题

- 美剧《基本演绎法》第2季第2集中，两位研究NP问题的数学家被谋杀了，凶手是同行，因为被害者即将证明”P=NP问题”，他为独吞成果而下了毒手。
- 凶手的动机，并不是千禧年大奖难题那100万美元的奖金，而是解决了”P=NP”问题，就能够破解世界上所有的密码系统，这里面的利益比100万美元多得多…
- 千禧年七大难题是由七个最顶尖的数学机构于2000年公布的一组数学难题，这些难题被认为是在当时最具挑战性的。
- 只有NP问题是计算机界的重大问题，而且排名榜首，足见其在当今数学界的重要地位！

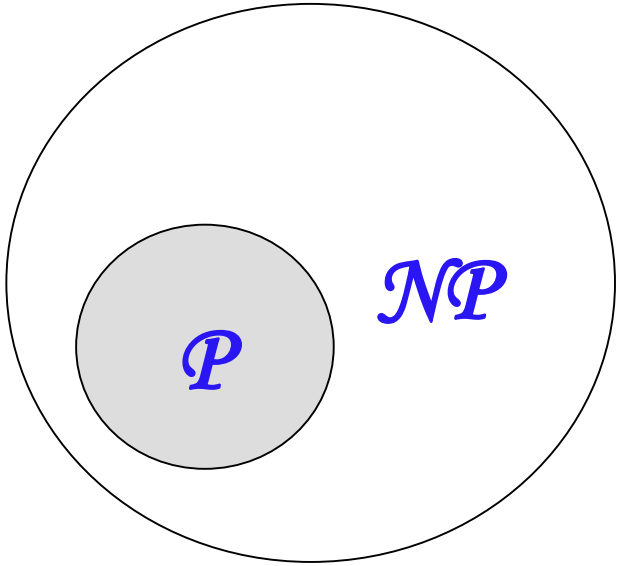
P类与NP类问题

- 多项式时间复杂度：若算法A在所有规模为n的输入下计算时间均为O(p(n))，其中p( )是某个多项式，则称算法A具有多项式时间复杂度。
- P 问题的P 是Polynomial Time(多项式时间)的头一个字母, NP 是Non deterministic Polynomial time (非确定性多项式时间)的缩写；
- P是在多项式时间内使用确定性算法可解的所有判定问题的集合，NP是在多项式时间内使用非确定性算法可解的所有判定问题的集合。
- P类问题是可以在多项式时间内解决并验证的一类问题，NP类问题是可以多项式时间验证但是不确定能否在多项式时间内解决的一类问题。

NP类问题

- 对NP问题，不关注求解本身所花的时间是否多项式时间，可能有多项式时间算法，可能没有，也可能不知道。
- NP概念的奥秘在于，它躲开了求解到底需要多少时间这样的问题，而仅仅强调验证需要多少时间。
- 判断是否NP问题：
  - 给出一个证书，可否在多项式时间内判断它是否是原问题的一个解；
  - 最优化问题需要转化为判定问题。
    - 优化问题：找到一个给定图G的最大团；
    - 判定问题：图G是否存在一个大小为k的团。

P类与NP类问题



- $\mathcal{P}$和$\mathcal{NP}$的关系：$\mathcal{P} \subseteq \mathcal{NP}$

- $\mathcal{P}=\mathcal{NP}$或者$\mathcal{P}\neq\mathcal{NP}$？

是否所有$\mathcal{NP}$类问题都可以在多项式时间内解决并验证，也就是转化为$\mathcal{P}$类问题。在理论信息学中计算复杂度理论领域里没有解决。

很可能，计算理论最大的未解决问题就是关于这两类的关系的:P和NP相等吗?

多项式时间变换

设X和Y是两个分别定义在实例集I和J上的判定问题。所谓问题X能在**多项式时间内变换**为问题Y(简记为$X \propto_p Y$)是指存在映射 $f:I \to J$，且 f 满足：

(1) f 是在多项式时间内可计算的函数；

(2) 对于所有$X \in I$，$x \in X$，当且仅当$f(x) \in Y$。

即$x \in X \Leftrightarrow f(x) \in Y$。

NPC

定义：问题L是**NP完全**的当且仅当

(1) $L \in NP$；

(2) 对于所有$L' \in NP$有$L' \propto_p L$。

如果有一个问题L满足上述性质(2)，但不一定满足性质(1)，则称该问题是**NP难**的。所有NP完全问题构成的问题类称为**NP完全问题类**，记为**NPC**。

NP完全问题具有如下性质：1）可在多项式时间内验证；2）它可以在多项式时间内求解，当且仅当所有的其他NP完全问题也可以在多项式时间内求解。

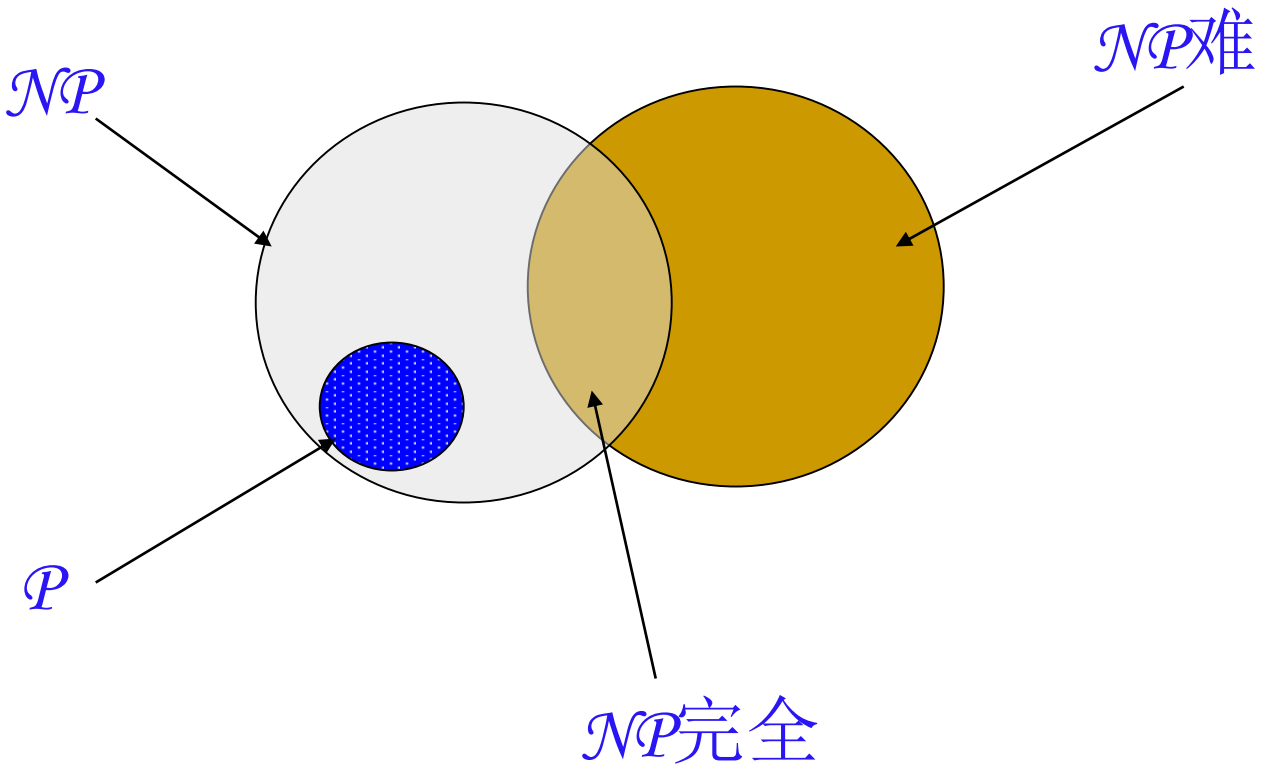
NP-Hard

- 连验证解都不能多项式时间解决的问题；
- 对于这一类问题，用一句话概括他们的特征“at least as hard as the hardest problems in NP Problem”，就是说NP-hard问题至少和NP问题一样难；
- 所有NP问题都能归约到NP-hard问题，但NP-hard问题不一定是NP问题；
- 即使NPC问题发现了多项式级的算法，NP-Hard问题有可能仍然无法得到多项式级的算法。事实上，由于NP-Hard放宽了限定条件，它将有可能比所有的NPC问题的时间复杂度更高从而更难以解决。

NPC和NP-Hard的不同

- 如果所有NP问题都可以多项式时间变换到问题A，那么问题A就是NP-Hard；
- NP-Hard问题类包含了NPC；
- 如果问题A既是NP-Hard又是NP，那么它就是NPC。

P、NP、NP完全和NP难问题的关系



NPC

定理8-2：设L是NP完全的，则

(1)$L \in P$当且仅当$P = NP$；

(2)若$L \propto_p L_1$，且$L_1 \in NP$，则L_1是NP完全的。

由(1)可知：如果任一NPC问题可在P时间内求解，则所有NPC问题可在P时间内求解。反之，若$P \neq NP$，则所有NPC问题不可在P时间内求解。

(2)可用来证明问题的NP完全性。但前提是：要有第一个NP完全问题L。

Cook定理

- 布尔表达式可满足性问题: 给定一个布尔表达式，是否存在一组变量的赋值，使得布尔表达式取值为TRUE。如果存在，称为可满足，如果不存在，称为不可满足。

算法4： 判断可满足性的非确定性算法

```
void Eval(cnf E, int n){
    int x[SIZE];
    for ( int i=1; i<=n; i++)
        x[i] = Choice(0,1);
    if (E(x, n))   Success();
    else   Failure();
}
```

- 所需时间O(n)包括选择E的赋值所需时间和确定性的计算E所需时间。该时间与E的长度成正比。

Cook定理

定理8-3(Cook定理)：布尔表达式的可满足性问题SAT是NP完全的。

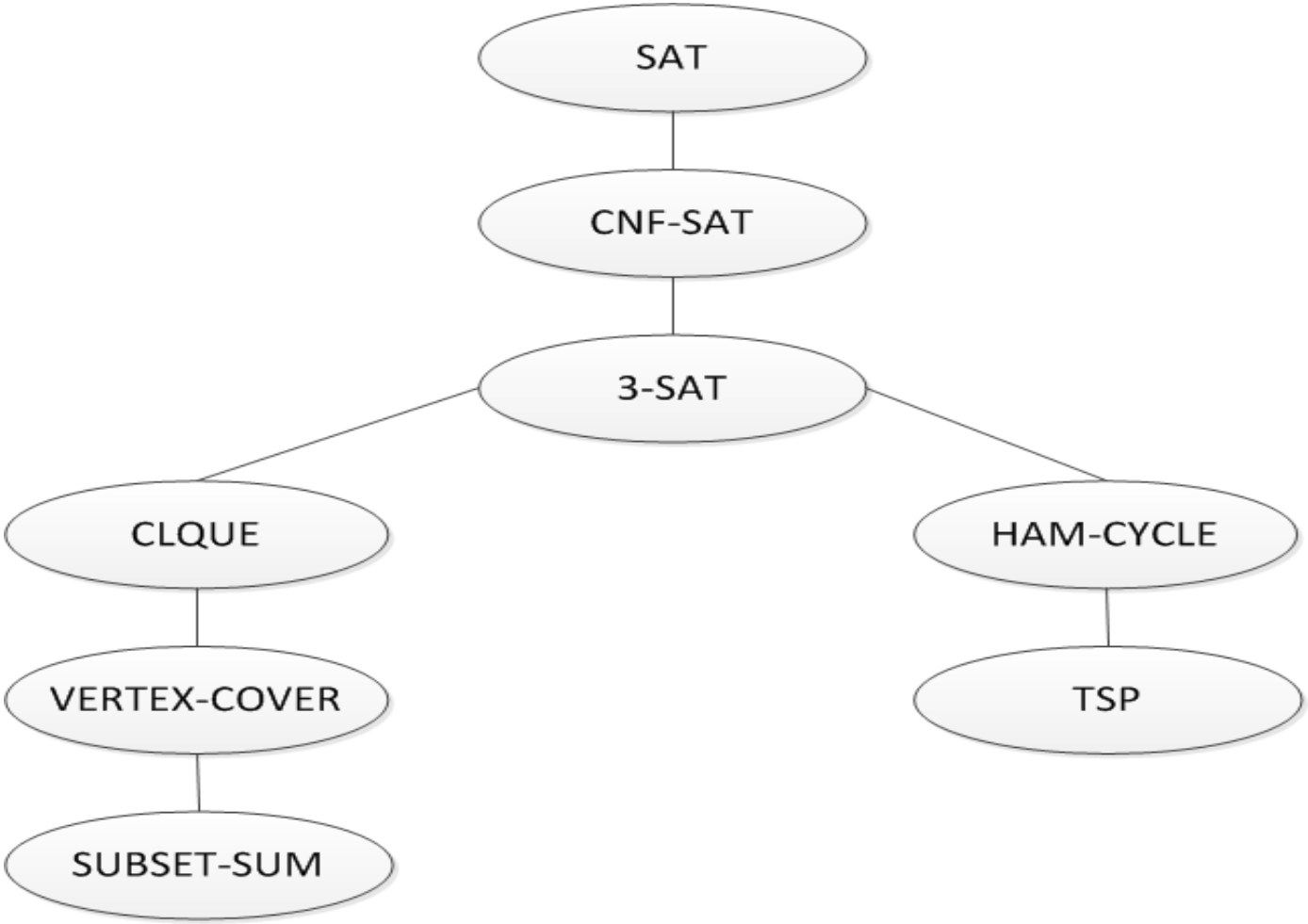
证明：SAT的一个实例是k个布尔变量$x_1, \ldots, x_k$的m个布尔表达式$A_1, \ldots, A_m$。若存在各布尔变量x_i $(1 \le i \le k)$的0，1赋值，使每个布尔表达式A_i $(1 \le i \le m)$都取值1，则称布尔表达式$A_1 A_2 \ldots A_m$是可满足的。

SAT∈NP是很明显的。对于任给的布尔变量$x_1, \ldots, x_k$的0，1赋值，容易在多项式时间内验证相应的$A_1 A_2 \ldots A_m$的取值是否为1。因此，SAT∈NP。

现在只要证明对任意的L∈NP有$L \propto_p$SAT即可。

(详细证明见书本P226～229)

一些典型的NP完全问题



部分NP完全问题树

1. 合取范式的可满足性问题（CNF-SAT）

问题描述：给定一个合取范式α，判定它是否可满足。

如果一个布尔表达式是一些因子和之积，则称之为合取范式，简称CNF(Conjunctive Normal Form)。这里的因子是变量 x 或 $\bar{x}$ 。例如 $(x_1+x_2)(x_2+x_3)\ (\bar{x}_1+\bar{x}_2+x_3)$ 就是一个合取范式，而 $x_1x_2+x_3$ 就不是合取范式。

我们已经知道，SAT∈NPC。因此，要证明CNF-SAT∈NPC，只需证明:

（1） CNF-SAT∈NP

（2） SAT$\propto_p$CNF-SAT

CNF-SAT问题

不含有任何连接词的谓词公式叫原子公式，简称原子，而原子或原子的否定统称文字。
子句就是由一些文字组成的析取式。

CNF-SAT∈NP 是显然的。因为合取范式是布尔表达式的一种特殊形式，由于SAT∈NP，所以CNF-SAT∈NP。

为了证明SAT∝pCNF-SAT，需要将一个布尔表达式满足性问题在多项式时间内转化为一个合取范式满足性问题。通过下列两个步骤来完成：

第一步，利用等价公式把所有的¬移到紧靠变元的位置，使布尔表达式变换为文字的逻辑与和逻辑或的形式，这种形式我们称之为与或形。

第二步，将第一步所得到的与或形公式转化为合取范式形式。转化方法如下：

CNF-SAT问题

设E是一个与或形公式，如果E中只包含1个或2个文字，则E已经是一个合取范式。否则，E可以表示为E1∧E2或E1∨E2的形式，可采用递归方式先将E1和E2分别转化为可满足性与之等价的合取范式F1和F2，再用F1和F2构造E的合取范式F。分两种情况处理：

1：若E= E1∧E2，则F= F1∧F2。显然F是一个合取范式并且F的可满足性与E等价。

2：若E= E1∨E2，不妨设由E1和E2转化所得到的合取范式分别为F1=c1∧c2∧…∧ck和F2=d1∧d2∧…∧dm。其中，ci和di都是子句。引入新变元u，构造合取范式如下：

$$F=(u+c1)(u+c2)\ldots(u+ck)(\bar{u}+d1)(\bar{u}+d2)\ldots(\bar{u}+dm)$$

可以证明，E和F的可满足性是等价的。

CNF-SAT问题

设与或形表达式E中包含的∧和∨运算的次数为n，那么由上述变换所得到的合取范式F中最多有n+1个子句。因此，把E分裂成E1和E2，以及由F1和F2构造F，每部分都是n的线性函数。设bn是把E分裂成E1和E2加上由F1和F2构造F所需时间总和的上界，于是可以得出由E构造F所需时间T(n)的递推方程：

$$T(n) \leq \begin{cases} a & n \leq 1 \\ bn + \max_{0 \leq i \leq n-1} \{T(i) + T(n-i-1)\} & n > 1 \end{cases}$$

运用归纳法可以证明$T(n)=O(n^2)$。

从而证得，布尔表达式满足性问题可在多项式时间内转化为一个合取范式满足性问题。

2．3元合取范式的可满足性问题（3-SAT）

问题描述：给定一个3元合取范式α，判定它是否可满足。

在一个合取范式中，如果每个子句最多含有k个文字，则称这个合取范式为k元合取范式，简记为k-CNF。一个 k-SAT问题是判定一个k-CNF是否可满足。当k=3时，即为3-SAT问题。

证明思路：

3-SAT∈NP是显而易见的。为了证明3-SAT∈NPC，只要证明CNF-SAT∝p3-SAT，即合取范式的可满足性问题可在多项式时间内变换为3-SAT。

3-SAT问题

现证明CNF-SAT∝p3-SAT。解决该问题的关键是如何将合取范式中每个子句转化成可满足性与其等价的3元合取范式。

设一个长度为k的子句，$L = x_1+x_2+\ldots+x_k$

现在来构造一个可满足性与L等价的3元合取范式F，即当且仅当F可满足时，L才是可满足的。

如果k<4，L就是一个3元合取范式。

如果k≥4，先利用子句L构造一个合取范式：

$$F = (x_1+x_2+y_1)(\bar{y}_1+x_3+\ldots+x_k)$$

其中y1为新增变量。 F与L的可满足性是等价的。依此，L可变换为一个3元合取范式，且变换时间正比于L的长度。

3. 团问题CLIQUE

问题描述：给定一个无向图G=(V，E)和一个正整数k，判定图G是否包含一个k团，即是否存在，$V' \subseteq V$，$|V'|=k$，且对任意u，$w \in V'$有$(u，w) \in E$。

证明思路：

(1) $CLIQUE \in NP$。因为验证图G的一个子图是否构成团只需要多项式时间，所以，$CLIQUE \in NP$。

(2) 通过$3\text{-}SAT \propto_p CLIQUE$来证明CLIQUE是NP难的，从而证明团问题是NP完全的。

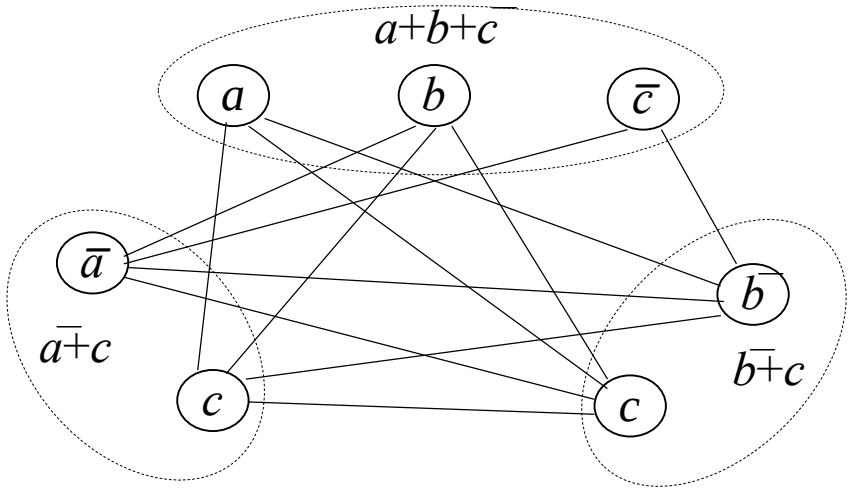
团问题CLIQUE

对于任意一个3元合取范式F，按照如下方法构造相应的图G。

1）图G的每个顶点对应F中的每个文字，对于多次出现在不同子句中的文字用不同的顶点重复表示；

2）若图G中两个顶点对应的文字不互补且不出现在同一子句中，则将其连线。例如，合取范式$(a+b+\bar{c})(\bar{a}+c)(\bar{b}+c)$对应的图如图所示。

设合取范式f有n个子句，则f可满足当且仅当f对应的图G中有n个顶点的团。



4. 顶点覆盖问题（VERTEX-COVER）

问题描述：给定一个无向图G=(V，E)和一个正整数k，判定是否存在$V' \subseteq V$，$|V'|=k$，使得对于任意$(u，v) \in E$有$u \in V'$或$v \in V'$。如果存在这样的V'，就称V'为图G的一个大小为k的顶点覆盖。

证明思路：

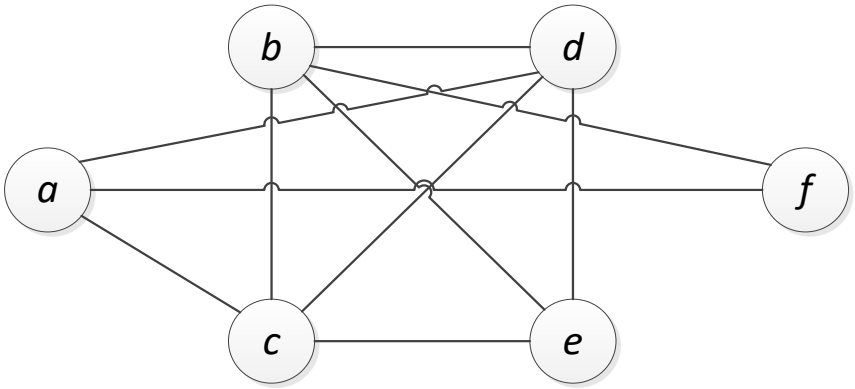
首先，VERTEX-COVER$\in$NP。因为对于给定的图G和正整数k以及一个顶点集V'，验证$|V'|=k$，然后对每条边(u,v) 检查是否有$u \in V'$或$v \in V'$，显然可在多项式时间内完成。

其次，通过CLIQUE$\propto_p$VERTEX-COVER来证明顶点覆盖问题是NP难的。

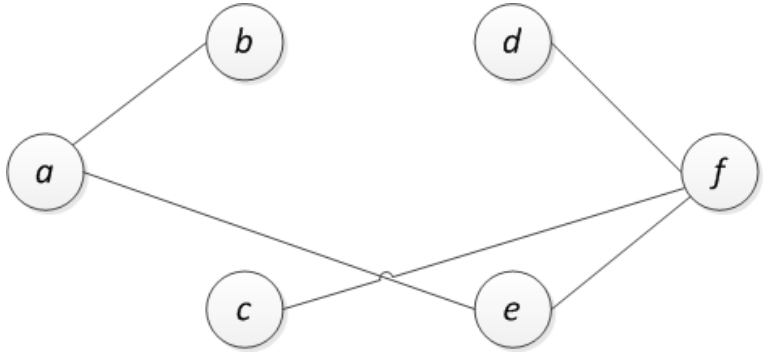
VERTEX-COVER问题

给定无向图G=(V, E)，构造G的补图G*=(V, E*)。其中：

E*={(u,v)|(u,v) ∉E}



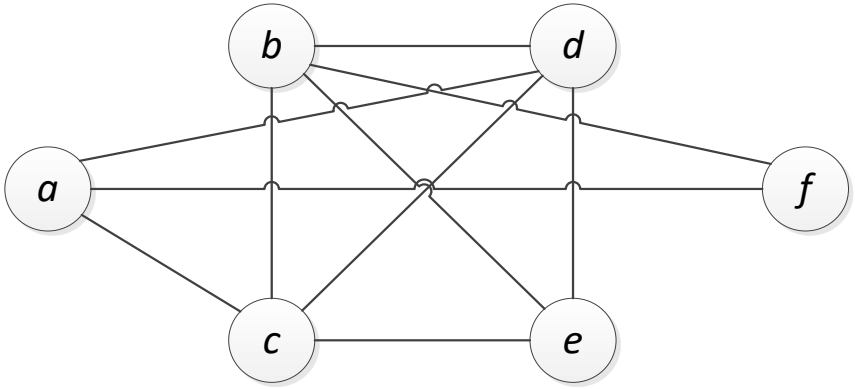
(G)



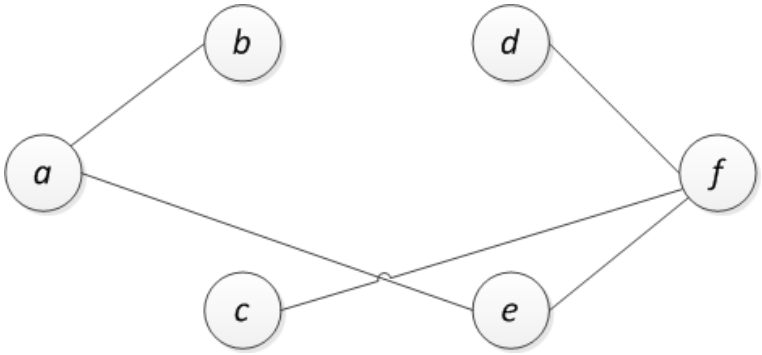
(G*)

根据图G与其补图G*之间的关系，可以证明图G中有一个大小为k的团当且仅当G*中有一个大小为|V|-k的顶点覆盖。

VERTEX-COVER问题



(G)



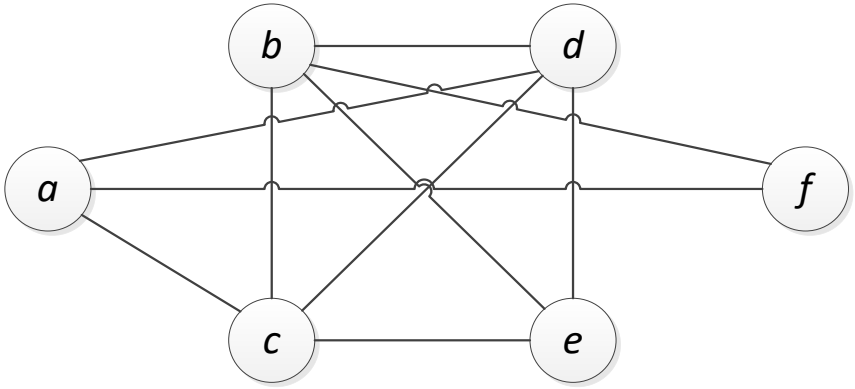
(G*)

若G有一个k团V’，则V-V’是G*的一个大小为|V|-k的顶点覆盖。

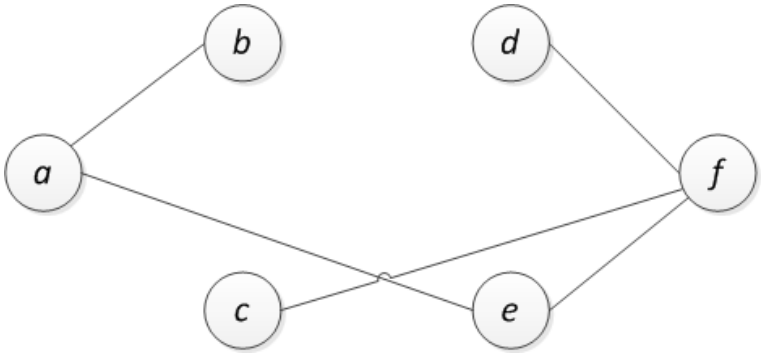
证明：设(u,v)是E*中任意一边，则(u,v) ∉ E。由团的性质知，u和v中至少有一个顶点不属于V’(因为若u和v都属于k团V’，则u和v之间必有边连接)。也就是说，u和v中至少有一个顶点属于V-V’，

即边(u,v)被V-V’覆盖。由(u,v) ∈ E*的任意性即知E*被V-V’覆盖。

VERTEX-COVER问题



(G)



(G*)

若G*有一个|V|-k的顶点覆盖V’，则V-V’是G的一个k团。

证明：若G*有一顶点覆盖V’⊆V，且|V’|=|V|-k。对任意u，v ∈V，若(u,v)∈E*，则u和v中至少有一个顶点属于V’。这等价于，对任意的u，v ∈V，若u∉V’且v∉V’，知(u,v) ∉E*，则 (u,v) ∈E。换句话说，若u ∈V-V’且v ∈V-V’，则(u,v) ∈E，即V-V’是G的一个团，其大小为|V|-|V’|=k，即V-V’是G的一个k团。

5. 子集和问题（SUBSET-SUM）

问题描述：给定整数集合S和一个整数t，判定是否存在S的一个子集$S' \subseteq S$，使得S’中整数的和为t。

例如，若S={1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344}且t=3754，则子集S’={1, 16, 64, 256, 1040, 1093, 1284}是一个解。

证明思路：

首先，对于子集和问题的一个实例<S, t>，给定一个“证书”S’，要验证$t=\sum_{i \in s'} i$是否成立，显然可在多项式时间内完成。因此，SUBSET-SUM∈NP；

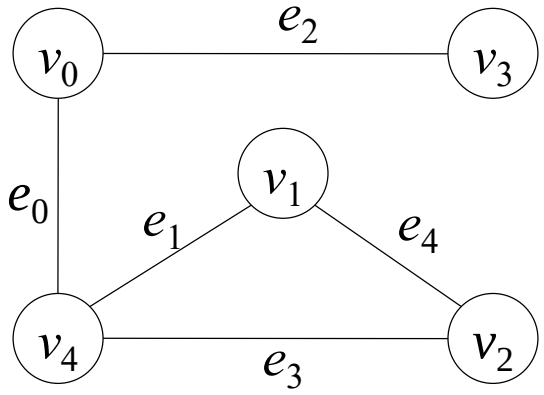
其次，证明VERTEX-COVER$\propto_p$SUBSET-SUM。

SUBSET-SUM问题

给定一个图的顶点覆盖问题的实例<G,k>，在多项式时间内将其变换为子集和问题的一个实例<S,t>，使得G有一个大小为k的顶点覆盖当且仅当S有一个子集S'，其元素和为t。

变换要用到图G的关联矩阵B。定义B=$\{b_{ij}\}$如下：

$$b_{ij}=\begin{cases}1 & \text{顶点}v_i\text{与边}e_j\text{相关联}\\ 0 & \text{其他}\end{cases}$$



	e_4	e_3	e_2	e_1	e_0
v_0	0	0	1	0	1
v_1	1	0	0	1	0
v_2	1	1	0	0	0
v_3	0	0	1	0	0
v_4	0	1	0	1	1

关联矩阵B

SUBSET-SUM问题

构造实例<S,t>的过程如下：

对于每个顶点vi，构造整数xi：

$$x_i = 4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} \cdot 4^j$$

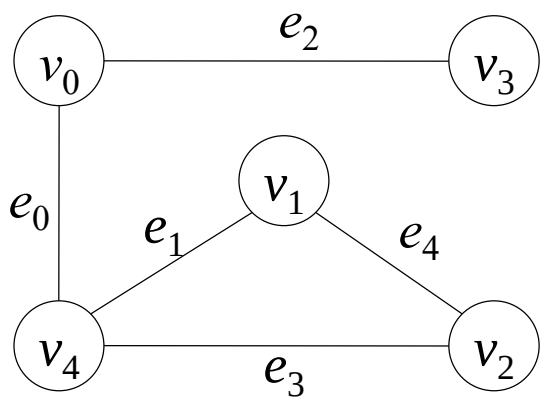
对于每条边ej，构造整数yj：

$$y_j = 4^j$$

则S={$x_0, x_1, \ldots, x_{|V|-1}, y_0, y_1, \ldots, y_{|E|-1}$}

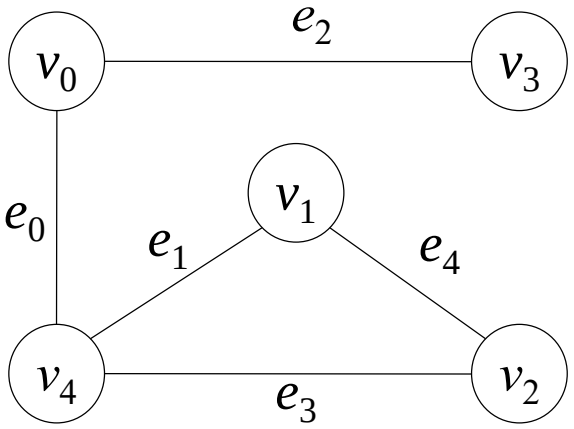
构造整数 t： $t = k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$

证明：当G有一个大小为k的顶点覆盖当且仅当S有一个子集S'，其元素和为t。



			e_4	e_3	e_2	e_1	e_0		
x_0	=	1	0	0	1	0	1	=	1041
x_1	=	1	1	0	0	1	0	=	1284
x_2	=	1	1	1	0	0	0	=	1344
x_3	=	1	0	0	1	0	0	=	1040
x_4	=	1	0	1	0	1	1	=	1093
y_0	=	0	0	0	0	0	1	=	1
y_1	=	0	0	0	0	1	0	=	4
y_2	=	0	0	0	1	0	0	=	16
y_3	=	0	0	1	0	0	0	=	64
y_4	=	0	1	0	0	0	0	=	256
t	=	3	2	2	2	2	2	=	3754

SUBSET-SUM问题



对于每个顶点vi：$x_i = 4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} \cdot 4^j$

对于每条边ej：$y_j = 4^j$

集合：$S=\{x_0, x_1, \ldots, x_{|V|-1}, y_0, y_1, \ldots, y_{|E|-1}\}$

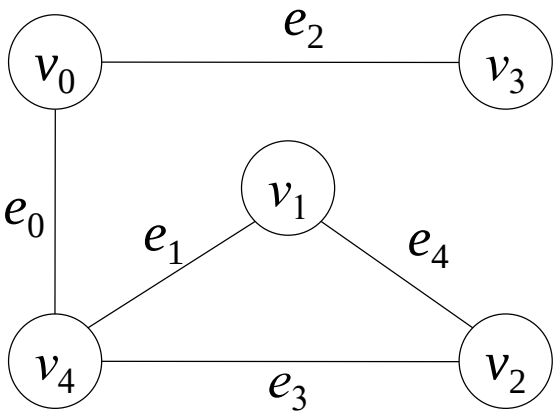
整数：$t = k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$

设G有一大小为k的顶点覆盖$V'=\{v_{i1}, v_{i2}, \ldots, v_{ik}\}$。定义

$S'=\{x_{i1}, x_{i2}, .., x_{ik}\} \cup \{yj \mid ej$恰与V'中一个顶点相关联。

则 $\sum_{i \in S'} i = k * 4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j = t$。

SUBSET-SUM问题



对于每个顶点vi： $x_i = 4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} \cdot 4^j$

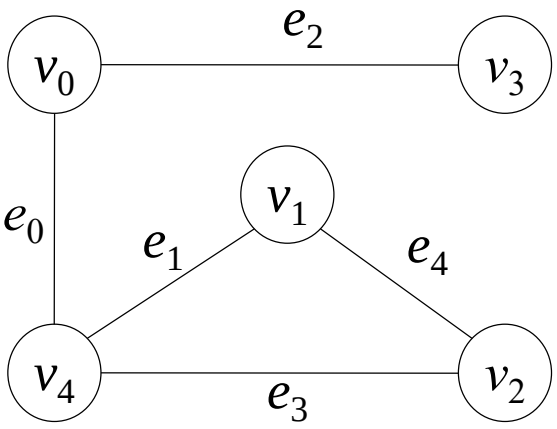
对于每条边ej： $y_j = 4^j$

集合： $S=\{x_0, x_1, \ldots, x_{|V|-1}, y_0, y_1, \ldots, y_{|E|-1}\}$

整数： $t = k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$

反之，设有一S的子集S’，其和$t=k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$。若$S'=\{x_{i1}, x_{i2}, .., x_{im}\} \cup \{y_{j1}, y_{j2}, \ldots, y_{jp}\}$, 则m=k, 且$V'=\{v_{i1}, v_{i2}, \ldots, v_{im}\}$是G的一个顶点覆盖。

SUBSET-SUM问题



对于每个顶点vi: $x_i = 4^{|E|} + \sum_{j=0}^{|E|-1} b_{ij} \cdot 4^j$

对于每条边ej: $y_j = 4^j$

构造整数t: $t = k4^{|E|} + \sum_{j=0}^{|E|-1} 2 \cdot 4^j$

顶点覆盖$\{v_0, v_1, v_2\}$对应有：$t = \boldsymbol{x_0 + x_1 + x_2 + y_0 + y_1 + y_2 + y_3}$

顶点覆盖中有v_i,，t 中有x_i，图中边 e_j关联的两顶点都在顶点覆盖中，则 t 中无 y_j，否则 t 中有y_j。

$t = \boldsymbol{x_0 + x_1 + x_2 + y_0 + y_1 + y_2 + y_3}$，对应有顶点覆盖$\{v_0, v_1, v_2\}$。

T中有 x_i,，顶点覆盖中有v_i，t 中从0到|E|-1的每一位对应图中一条边，每一位上的2至多由该位对应的边贡献一个1，另外一个1必须由该边关联的点贡献一个1，也即至少覆盖该边的一个点。

6. 哈密顿回路问题（HAM-CYCLE）

问题描述：给定无向图G=(V,E)，判定其是否含有一哈密顿回路。

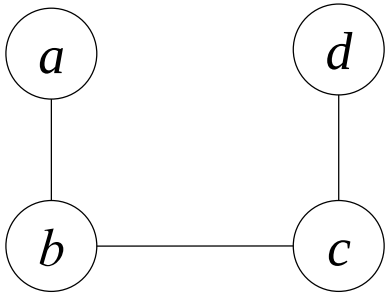
哈密顿回路：在一个回路中，除了经过初始结点两次以外，恰好经过每个结点一次，则称此回路为哈密顿回路。

证明思路：

首先，对于HAM-CYCLE问题的一个实例，给定所有顶点的一个排列，要验证这个序列是否是一个回路，显然可在多项式时间内完成。因此， $\text{HAM-CYCLE} \in \text{NP}$；

其次，证明$\text{VERTEX-COVER} \propto_p \text{HAM-CYCLE}$。

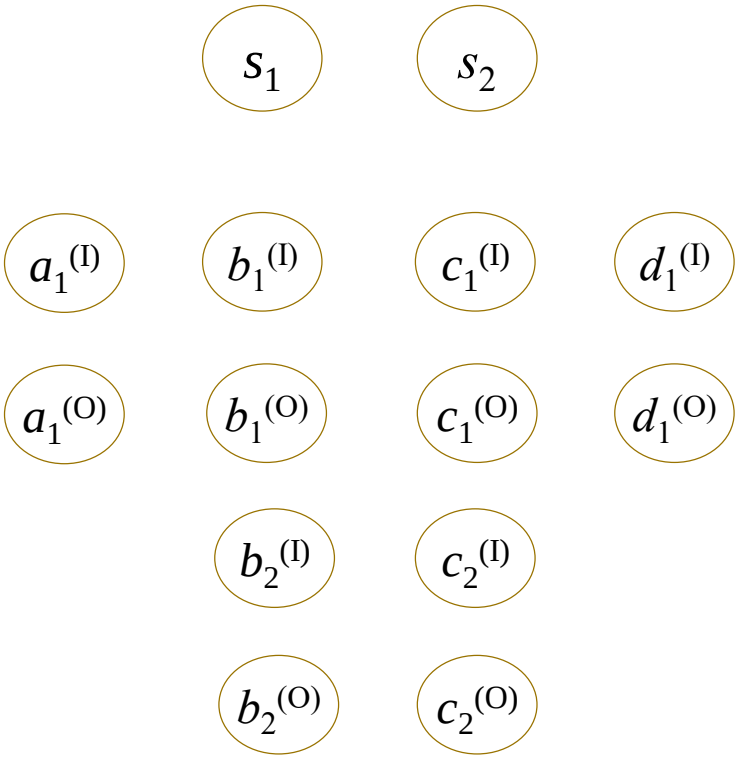
HAM-CYCLE问题



G，k=2

对于无向图$G=<V,E>$，给定G的顶点覆盖实例$<G,k>$。设顶点$v\in V$的度为d_v，与v相关联的边记为：$e_1^v, e_2^v, \cdots, e_{dv}^v$

现构造一个有向图G'=<V',E'>，过程如下：

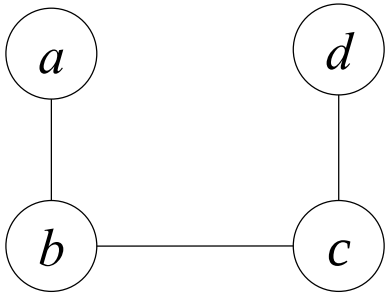


顶点集合V'由两部分组成：

1）新增k个顶点：$s_1, s_2, \ldots, s_k$

2）对于每个顶点$v\in V$，对应有$2d_v$个顶点：$v_1^{(I)}, v_1^{(O)}, v_2^{(I)}, v_2^{(O)}, \cdots, v_{dv}^{(I)}, v_{dv}^{(O)}$

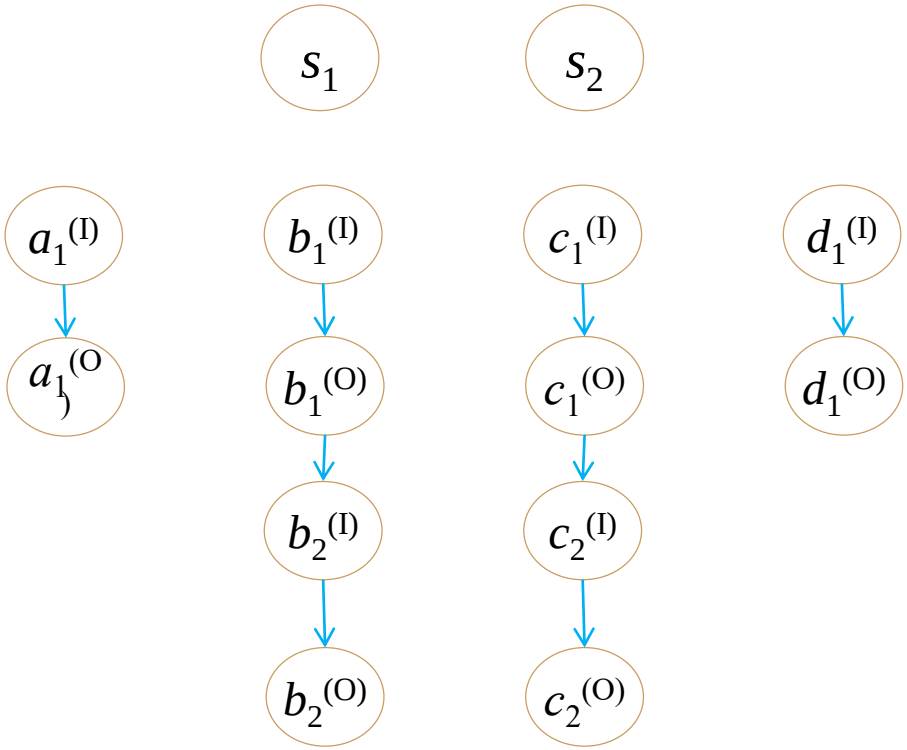
HAM-CYCLE问题



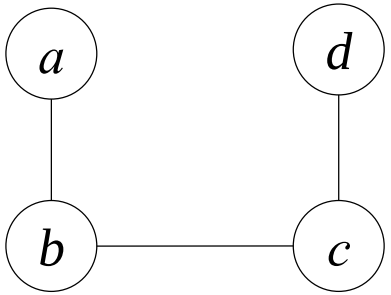
边集合E' 的组成部分有：

1）对于每个顶点v∈V，对应有有向边：

$$(v_1^{(I)}, v_1^{(O)}),\quad (v_1^{(O)}, v_2^{(I)}),\quad (v_2^{(I)}, v_2^{(O)}),\quad \cdots\quad (v_{dv}^{(I)}, v_{dv}^{(O)})$$

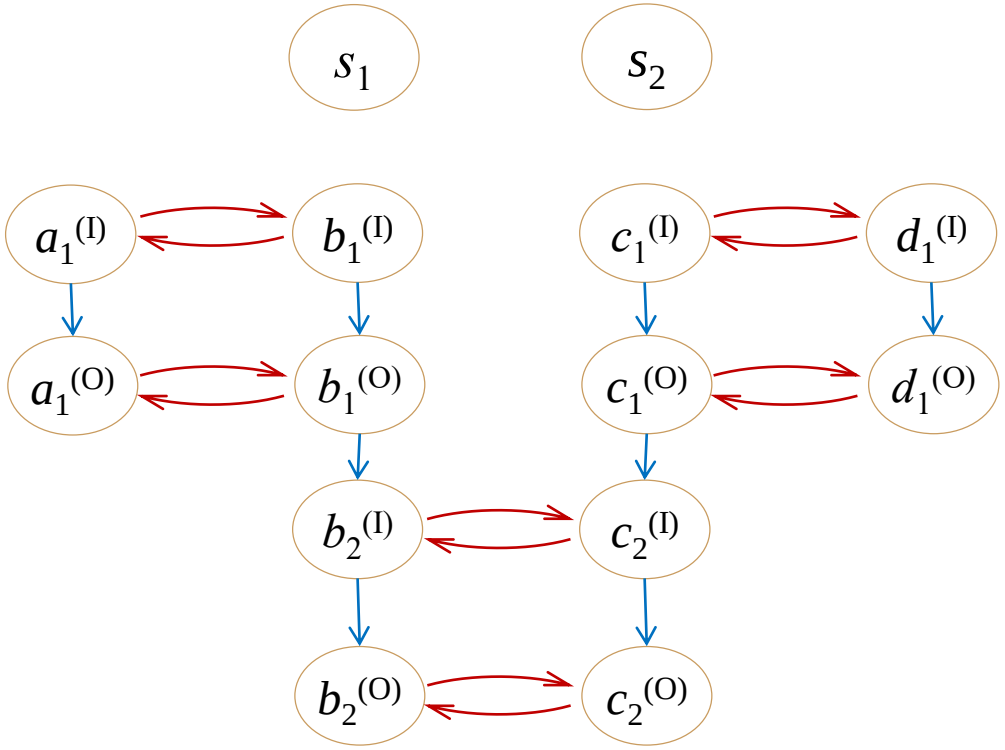


HAM-CYCLE问题

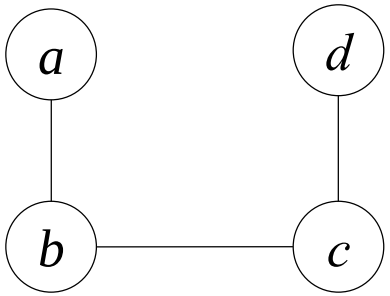


2）对于每条边(u, v)∈E ($e_i^u = e_j^v$)，对应有向边：

$(u_i^{(I)}, v_j^{(I)})$，　$(v_j^{(I)}, u_i^{(I)})$，　$(u_i^{(O)}, v_j^{(O)})$，　$(v_j^{(O)}, u_i^{(O)})$

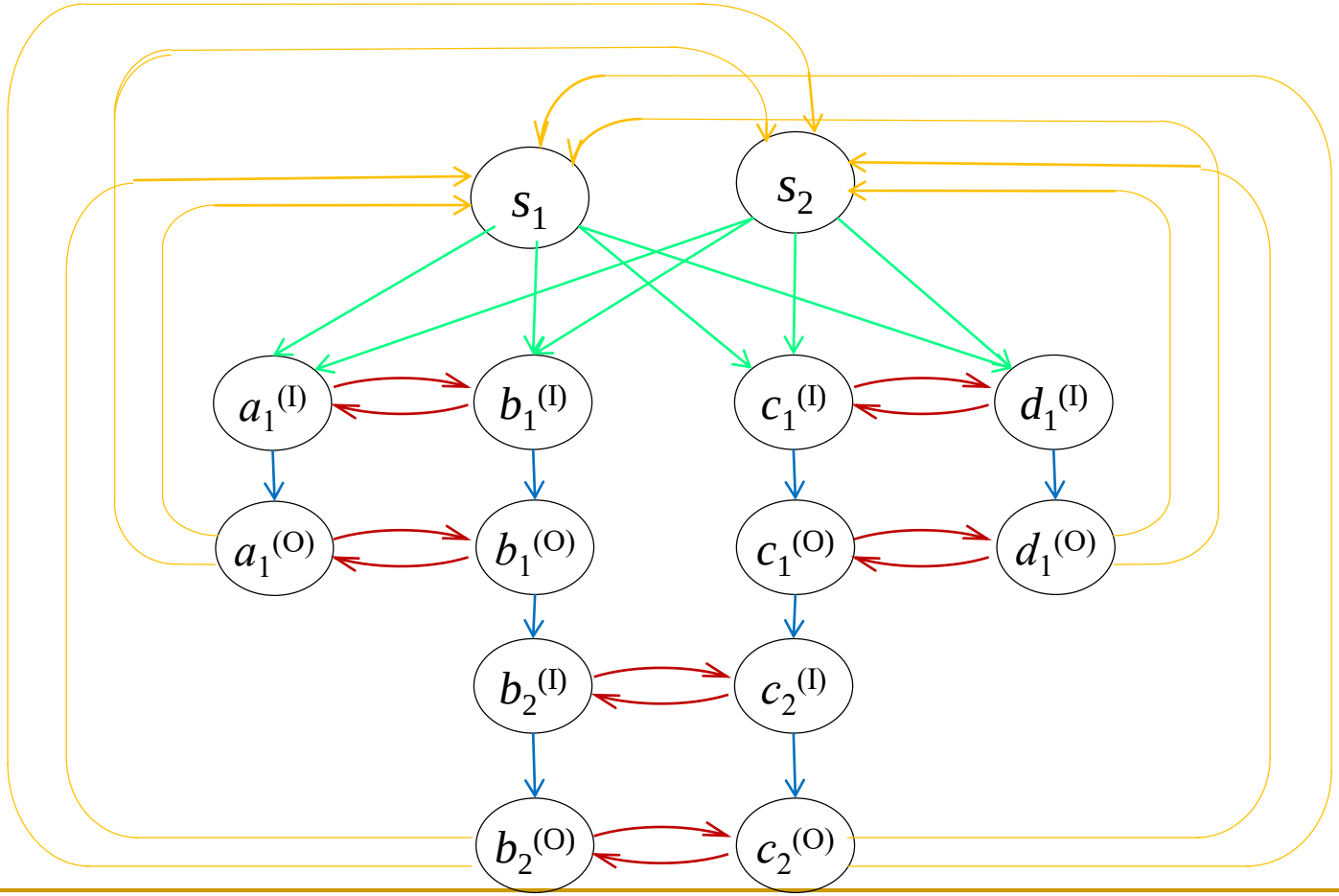


HAM-CYCLE问题

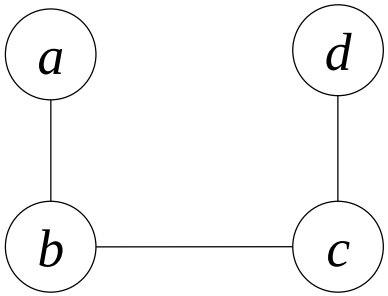


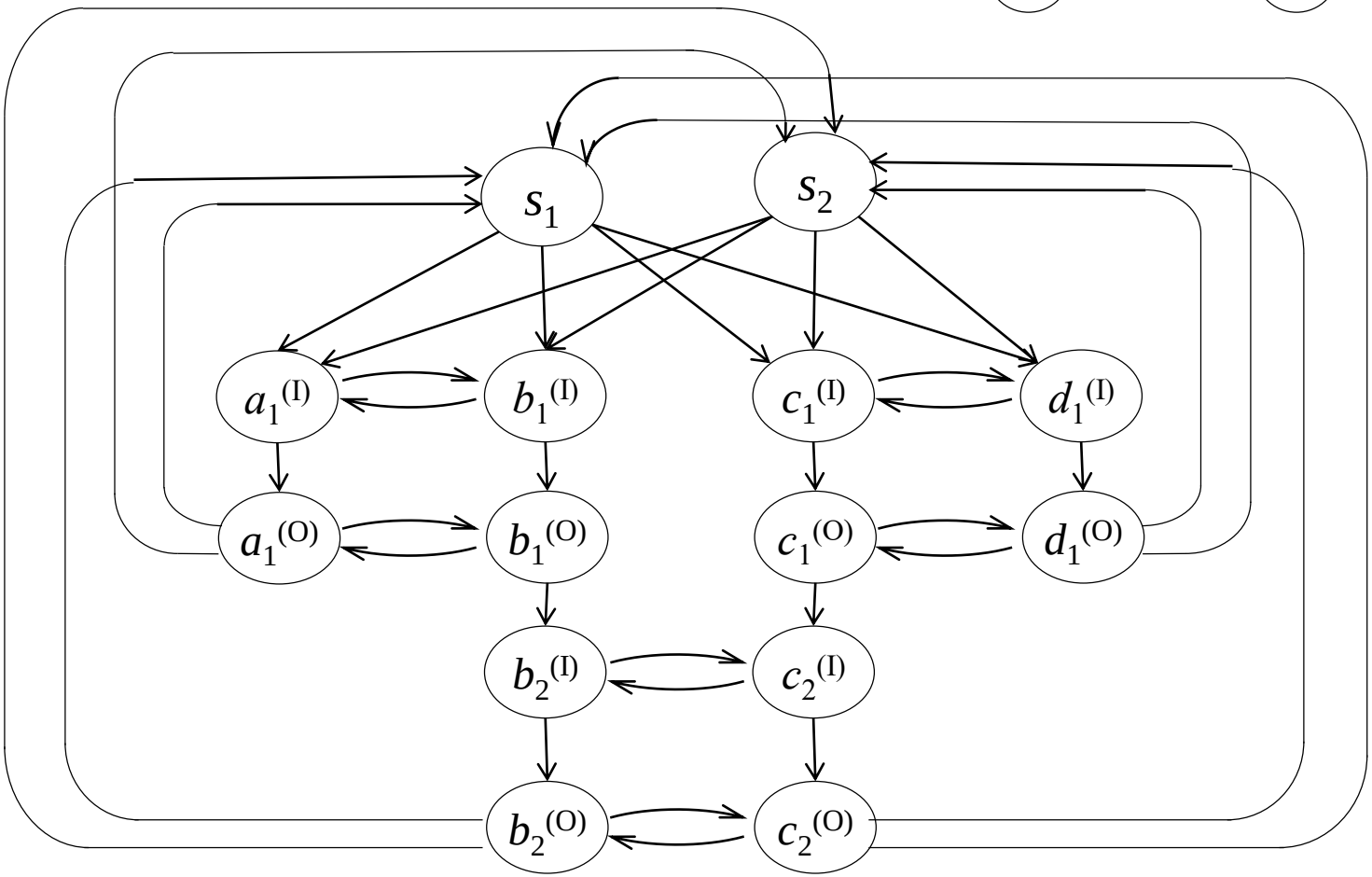
3）对于s_i与每个顶点$v \in V$，添加有向边：

$(s_i, v_1^{(I)})$，　$(v_{dv}^{(O)}, s_i)$　　$(1 \le i \le k)$



最终，构造图G’如下：





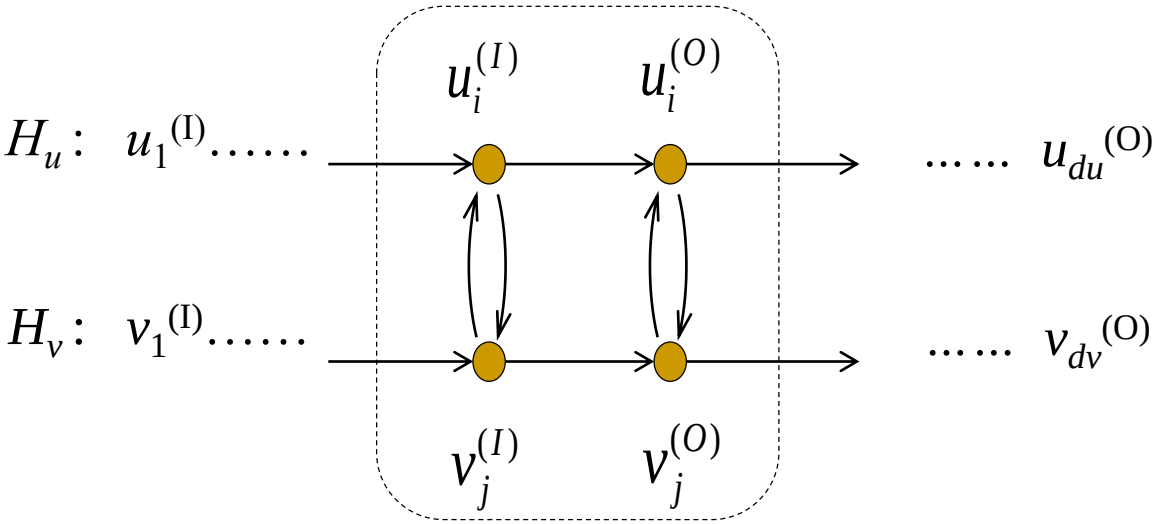
HAM-CYCLE问题

对于G中的每个顶点$v \in V$，在G'中对应有一个通路H_v：

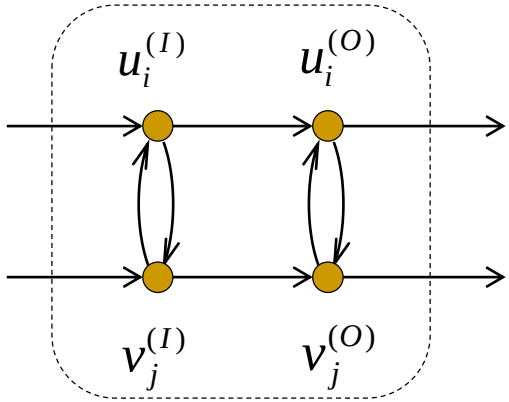
$$H_v : v_1^{(I)} \to v_1^{(O)} \to v_2^{(I)} \to v_2^{(O)} \to \cdots \to v_{dv}^{(I)} \to v_{dv}^{(O)}$$

对于G中的每条边$(u, v) \in E$，通路H_u和H_v之间存在边：

$$(u_i^{(I)}, v_j^{(I)}), \quad (v_j^{(I)}, u_i^{(I)}), \quad (u_i^{(O)}, v_j^{(O)}), \quad (v_j^{(O)}, u_i^{(O)})$$



HAM-CYCLE问题



这个子图有下列特点：

（1）每个顶点的入度和出度都为2；

（2）若有哈密顿回路由$u_i^{(I)}$进入，则必须从$u_i^{(O)}$退出该子图。若从$v_j^{(O)}$退出，则$u_i^{(O)}$或$v_j^{(I)}$将始终无法被经过。

因此，从$u_1^{(I)}$进入的哈密顿回路必须从$u_{du}^{(O)}$退出，即必定经过H_u通路中的所有顶点。

接下来，我们证明G’存在哈密顿回路的充要条件是图G有k个顶点的覆盖。

HAM-CYCLE问题

设图G有一个k个顶点的覆盖C={$v1,v2,\ldots vk$}，可根据C构造G’的一条哈密顿回路。步骤如下：

（1）先构造一条回路P

$$s_1 \to H_{v1} \to s_2 \to H_{v2} \to \cdots \to s_k \to H_{vk} \to s_1$$

（2）对于任意一条边$(u,v)=e_i^u=e_j^v$，u和v中至少有一个是在C中。如果u和v都在C中，则边(u,v)在G’中对应的4个顶点已经包含在回路P中。不妨设u在C中而v不在，那么在回路P中的H_u可以由$u_i^{(I)} \to v_j^{(I)} \to v_j^{(O)} \to u_i^{(O)}$ 绕道经过$v_j^{(I)}$ 和 $v_j^{(O)}$，从而将$v_j^{(I)}$ 和 $v_j^{(O)}$加入到回路P中。由于C是顶点覆盖， G’中的所有顶点都可添加到P中。

反之，也可从G’的哈密顿回路构造图G的顶点覆盖。

7. 旅行售货员问题TSP

问题描述：给定一个无向完全图$G=(V,E)$及定义在$V\times V$上的一个费用函数c和一个整数k，判定G是否存在经过V中各顶点恰好一次的回路，使得该回路的费用不超过k。

- 首先，给定TSP的一个实例(G,c,k)，和一个由n个顶点组成的顶点序列。验证算法要验证这n个顶点组成的序列是图G的一条回路，且经过每个顶点一次。另外，将每条边的费用加起来，并验证所得的和不超过k。这个过程显然可在多项式时间内完成，即TSP$\in$NP。
- 其次，旅行售货员问题与哈密顿回路问题有着密切的联系。哈密顿回路问题可在多项式时间内变换为旅行售货员问题。

TSP问题

设图G=(V, E)是HAM-CYCLE问题的一个实例，构造一个完全图G’=(V, E’)，且定义费用函数c为：

$$c(i, j) = \begin{cases} 0 & (i, j) \in E \\ 1 & (i, j) \notin E \end{cases}$$

则相应的TSP的实例为G’=(V,c,0)。这个构造过程可以在$\Theta(n^2)$时间内完成。

TSP问题

$$c(i,j)=\begin{cases}0 & (i,j)\in E\\ 1 & (i,j)\notin E\end{cases}$$

证明：G有一个哈密顿回路当且仅当G’有一个费用为0的旅行售货员回路。

若G有一个哈密顿回路H，显然H也是G’的一个旅行售货员回路，且由于H的每一边均属于E，故每边费用均为0。因此，H是G’的一个费用为0的旅行售货员回路。

若G’有一个费用为0的旅行售货员回路H，由费用函数c的定义可知，H的每边费用均为0，从而H的每条边均属于E。故H为G的一条哈密顿回路。

得证HAM-CYCLE$\propto$pTSP。从而，旅行售货员问题是NP难的。因此，TSP$\in$NPC。