

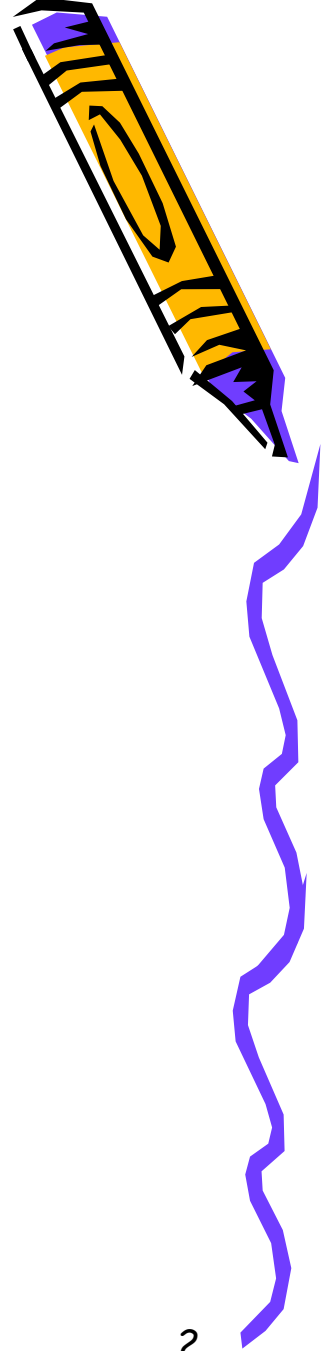


第5章 复杂性问题的分类 和NP完全性

杨莹，冷芳玲

5.1 图灵机计算复杂性量度

- 复杂性的量度
- 复杂性量度的记法
- 算法分析
- 复杂性类
- NP完全问题



计算量

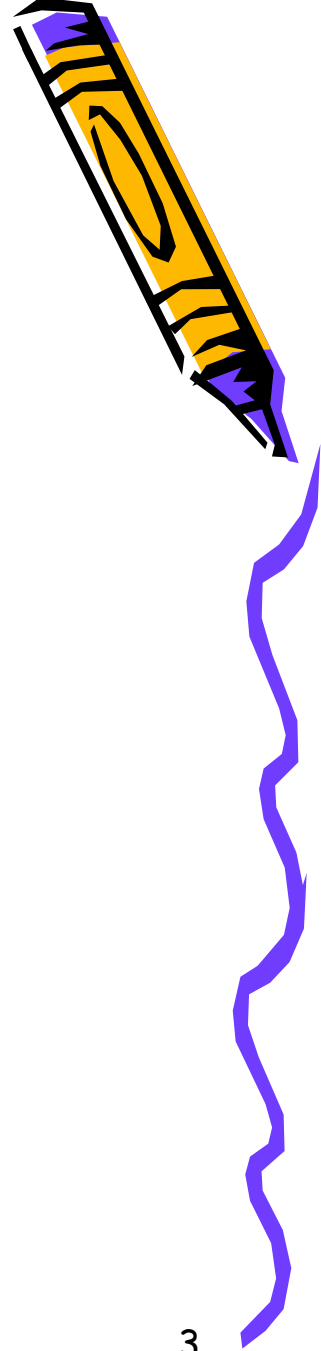
$\{a_1, a_2, \dots, a_n\} : n$ 个整数

Q1. 求和 (1) : $a_1 + a_2 + \dots + a_n$.
 $n-1$ steps.

Q2. 求和 (2) :

(1) $2 \times a_1 + \dots + 2 \times a_n$, $2n-1$ steps.

(2) $2 \times (a_1 + \dots + a_n)$, n steps.



計算量的膨脹

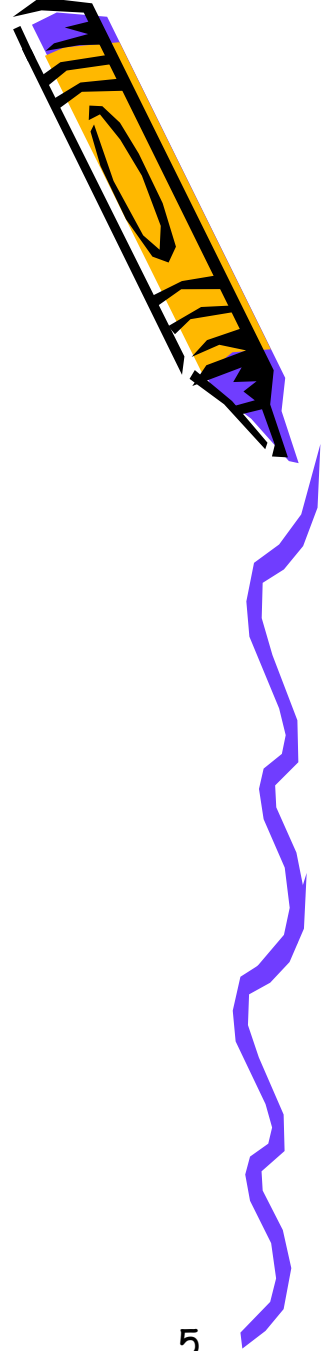
- 100MIPS (mega instructions per second)
 - 1秒100萬次的計算 = 1step 用 10^{-6} 秒

• n	10	100	1,000	10,000
• n	10^{-5} 秒	10^{-4} 秒	10^{-3} 秒	0.01 秒
• n^2	10^{-4} 秒	0.01 秒	1 秒	100 秒
• n^3	0.001 秒	1 秒	16.6 分	277 小時
• 2^n	0.001 秒	10^{14} 世紀	10^{284} 世紀	
• $n!$	0.036 秒	10^{141} 世紀	10^{2551} 世紀	
• 字齡: 宇宙的年齡	1.5×10^8 世紀 (150 億年)			

5.1 图灵机计算复杂性量度

- 5.1.1 复杂性的量度

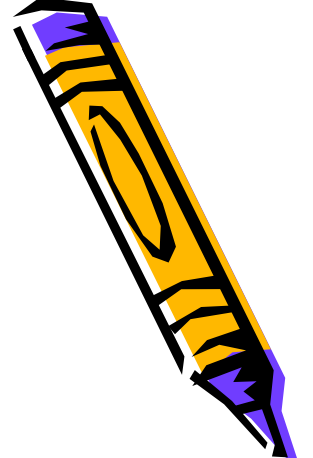
- 1. 空间复杂度
- 2. 时间复杂性
- 3. 巡回复杂性



5.1.1 复杂性的量度

- 1. 空间复杂度

定义5-1 令 M 是一个对于所有输入都停机的离线图灵机， $s: N \rightarrow N$ 是一个函数。如果对于每个长度为 n 的输入字， M 在任一条存贮带（或工作带）上至多扫视 $s(n)$ 个单元，那么称 M 是 $s(n)$ 空间有界图灵机，或称 M 具有空间复杂度 $s(n)$ 。



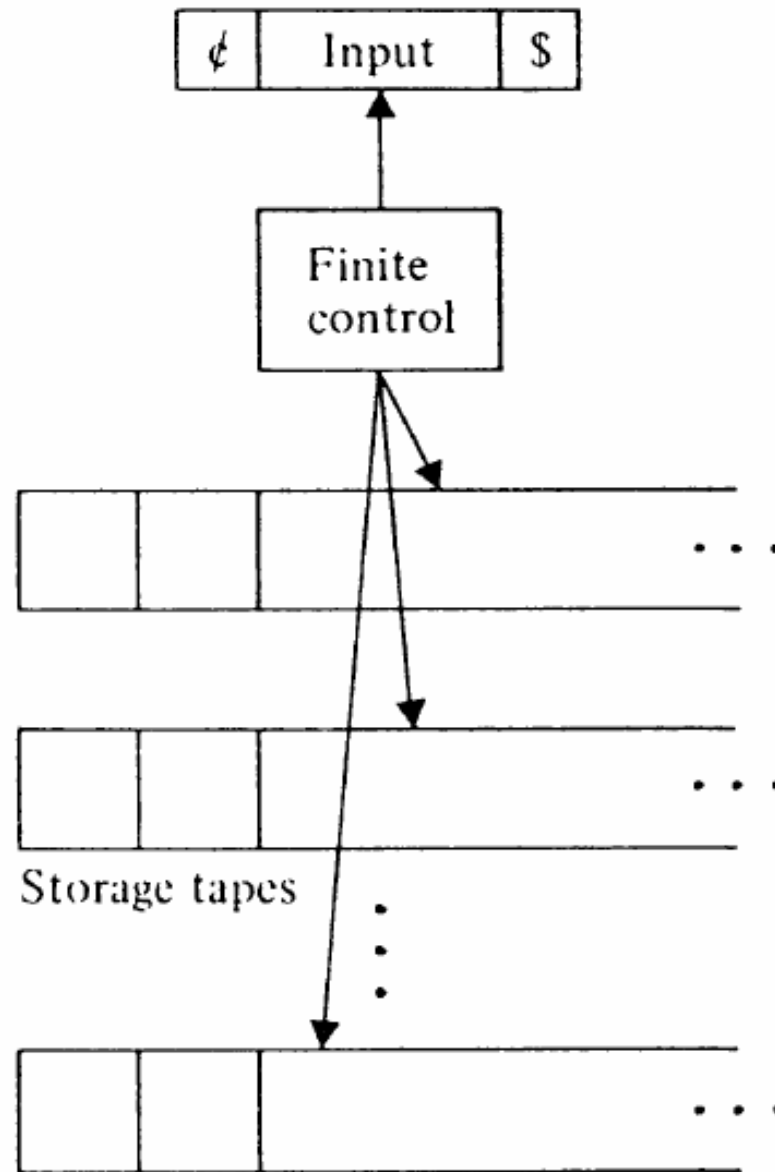


Fig. 12.1 Multitape Turing machine with read-only input.



5.1.1 复杂性的量度

– 2. 时间复杂性

定义5-2 设 M 是一个对于所有输入都停机的确定图灵机， $t: N \rightarrow N$ 是一个函数。如果对于每个长度为 n 的输入， M 在停机前最多做 $t(n)$ 动作（步骤），那么就称 M 是一个 $t(n)$ 时间有界的图灵机，或称 M 具有时间复杂度 $t(n)$ 。

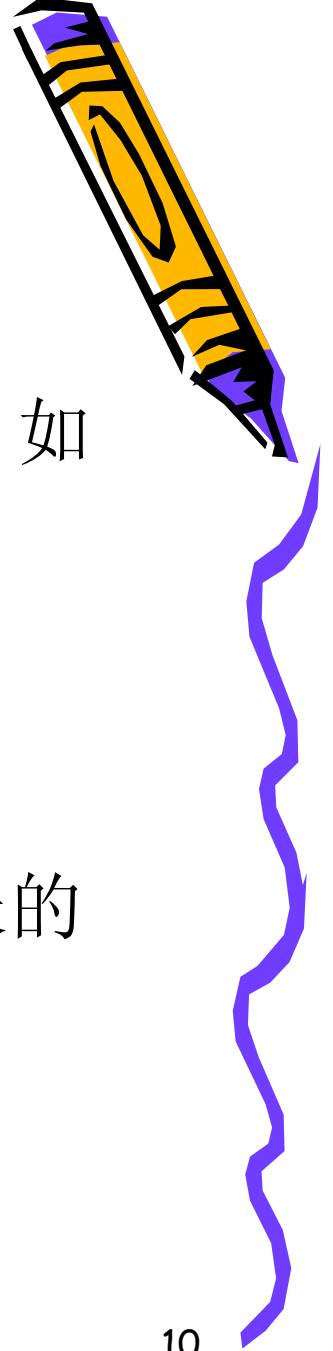


ID序列的长与宽

5.1.1 复杂性的量度

– 3. 巡回复杂性

定义5-3 TM M 对于给定的输入 w ，其运行的周相为 $(0, i_1), (i_1, i_2), (i_2, i_3), \dots, (i_{r-2}, i_{r-1})$ ，则称 $0, i_1, i_2, \dots, i_{r-1}$ ，是一个有限周相的划分，数 r 称为该划分的巡回数。



5.1.2 复杂性量度的记法

- 定义5-4 设 f 和 g 是两个函数, $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$ 。如果

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = C$$

-
- 则称 $f(n) = O(g(n))$ 。此时, $g(n)$ 是 $f(n)$ 渐近增长的一个上界。
- 例如, $f(n) = 0.5 * n^3 + 2n^2 + 1 = O(n^3)$



5.1.2 复杂性量度的记法

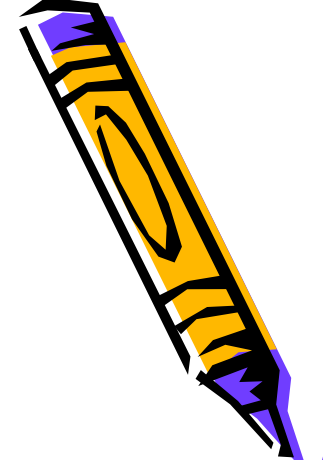
- 定义5-5 设 f 和 g 是两个函数, $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$, 称 $f(n) = o(g(n))$, 如果有
-

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

5.1.2 复杂性量度的记法

【例5-3】不难验证下面各式具有 o 关系:

- $n^2 = o(n^3)$
- $n = o(n \log \log n)$
- $n \log n = o(n^2)$



5.1.2 复杂性量度的记法

定义 5-6 设 $f: N \rightarrow R^+$ 是一个函数。定义

- (1) $\sup_{n \rightarrow \infty} f(n)$ 表示当 $n \rightarrow \infty$ 时, $f(n)$ 的最小上界;
- (2) $\inf_{n \rightarrow \infty} f(n)$ 表示当 $n \rightarrow \infty$ 时, $f(n)$ 的最大下界。

如果当 $n \rightarrow \infty$ 时, $f(n)$ 收敛于一个极值, 那么就有

$$\sup_{n \rightarrow \infty} f(n) = \inf_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} f(n)。$$



5.1.3 算法分析

【例5-4】 考虑接受语言 $L = \{WCW^R \mid W \in \{0,1\}^*\}$ 的图灵机的复杂性。

语言 L 具有时间复杂度 $O(n)$ ，因为存在一个图灵机 M ，它具有两条带，并把 C 左边的内容复制到第二条带上，然后当遇到 C 时， M 第二带的读头向左，边比较边移动：比较输入带 C 右侧的字符和第二带的字符，如果每对字符都相同，且 C 左边的符号数相等，那么 M 接受。易见，如果输入长度是 n ，则 M 最多做 $n+1$ 个动作。





5.1.3 算法分析

- 【例5-4】 考虑接受语言 $L = \{WCW^R \mid W \in \{0,1\}^*\}$ 的图灵机的复杂性。

存在另一个图灵机M2，它接受L，具有空间复杂度 $\log_2 n$ 。M2用二条工作带来作二进制计数器，首先，检查输入，看看是否只出现一个C，以及C左边和右边的符号数是否相等。然后，逐个字符地比较C左边和右边的字，同时用上述计数器找出对应的符号，如果它们不一致，M2停机且不接受，如果所有的符号都一致，M2就接受。





5.1.3 算法分析

【例5-5】自然数的二进制表示转变为一进制表示。

图灵机T1的设计如下：设输入为： $a_0a_1a_2\dots a_{n-1}$ ($a_i \in \{0,1\}$)，则输出为 a^m ，其中 $m = \sum a_i 2^i$ ($i: 0 \sim n-1$)。T1有两条工作带x和y，T1的工作过程如下：

- (1) 在x上写一个a；
- (2) 若输入为空格，则停机；
- (3) 若输入为1，工作带x的内容送至输出带，并把x的内容在y上抄两遍，然后用y的内容覆盖原x的内容，清除y；
否则若输入符号为0时，只把x的内容在y上抄两遍，然后用y的内容覆盖原x的内容，清除y；
- (4) 输入带头右移一格，转至步(2)。



5.1.4 复杂性类

- 定义5-6 设 $t:N \rightarrow N$ 是一个函数，定义时间复杂性类 $DTIME(t(n))$ 为

$DTIME(t(n)) = \{L \mid L \text{ 是由一个确定图灵机在 } O(t(n)) \text{ 时间内识别的语言}\}。$

- 定义5-7 设 $s:N \rightarrow N$ 是一个函数，定义空间复杂性类 $DSPACE(s(n))$ 为

$DSPACE(s(n)) = \{L \mid L \text{ 是由一个确定图灵机在 } O(s(n)) \text{ 空间内识别的语言}\}。$

5.1.4 复杂性类

- 定义：我们称一个非确定图灵机TM具有时间复杂度 $t(n)$ ，是指“它解决规模为 n 的问题过程中，任何动作选择序列都不会使TM做多于 $t(n)$ 个动作”。
- 定义5-8 设 $t:N \rightarrow N$ 是一个函数，定义非确定时间复杂性类 $NTIME(t(n))$ 为

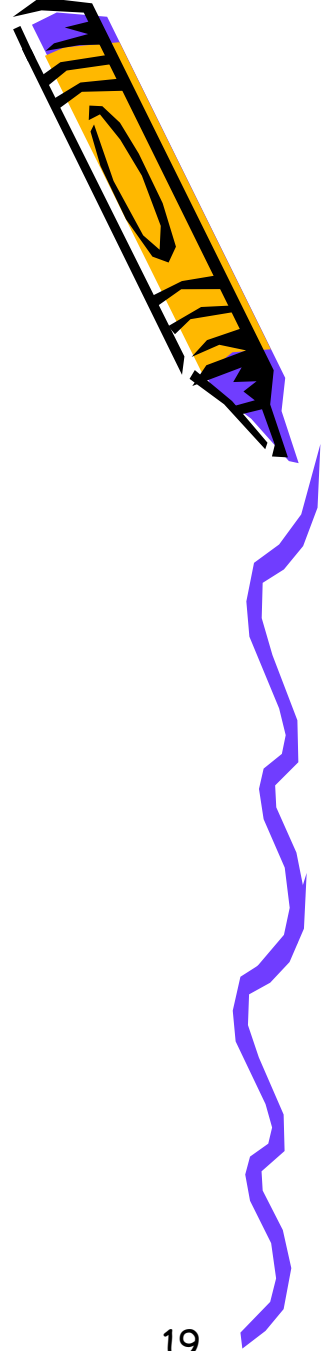
$NTIME(t(n)) = \{L \mid L \text{ 是由一个非确定图灵机在 } O(t(n)) \text{ 时间内识别的语言}\}。$

- 定义5-9 设 $s:N \rightarrow N$ 是一个函数，定义非确定空间复杂性类 $NSPACE(s(n))$ 为

$NSPACE(s(n)) = \{L \mid L \text{ 是由一个由 } O(s(n)) \text{ 空间的非确定图灵机识别的语言}\}。$

5.2 复杂性问题的分类

- P 类
- 可归约性
- 多项式时间归约
- NP 类



5.2.1 P类

- 定义5-12 P类是确定型单带图灵机在多项式时间内可判定的语言类，即

$$P = \bigcup_k \text{DTIME}(n^k)$$

- 对于所有与确定型单带图灵机多项式等价的计算模型来说，P是不变的；

P大致对应于在计算机上实际可解的问题类。

5.2.1 P类

- P表示确定的图灵机在多项式时间(步数)内可判定的语言类。这些语言对应的问题称为P类问题，这种语言称为多项式可判定的。
- 例如，判定一个有向图中的两个顶点之间是否存在有向路的问题；检查两个数是否互素的问题；判定一个字符串是否为一个上下文无关语言的句子的问题都是P类问题。



5.2.1 P类

- **【例5-8】** 有向图中两个节点的连通性判定问题是P类问题。

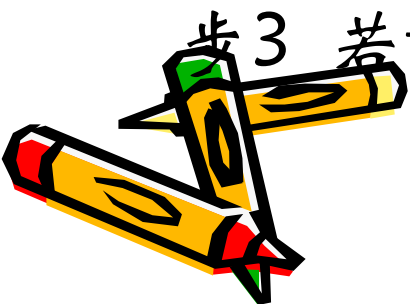
证明：设有向图 $G = \langle V, E \rangle$, $s, t \in V$, $n = |V|$ 。不失一般性可设 G 是简单图。下面是该问题的判定算法。

步1 标记节点 s ;

步2 重复步2.1直至不再有节点需要标记:

步2.1 扫描 G 的所有边。如果找到一条边 (u, v) , $u, v \in V$, u 被标记, 而 v 未被标记, 则标记 v ;

步3 若 t 被标记, 则接受; 否则拒绝。



二元布尔逻辑公式可满足问题



假设我们把可满足性的实例限制在所有子句只有两个或更少文字的公式上,于是得到我们称为二元可满足性的新问题。二元可满足性是可满足性的特殊情形。这样说的意思是所有可能的输入只是可满足性的输入的子集合,并且在两个问题里对每个公共输入的回答都是同样的。二元可满足性的典型实例如下所示。

$$\{(x_1 \vee x_2), (x_3 \vee \overline{x_2}), (x_1), (\overline{x_1} \vee \overline{x_2}), (x_3 \vee x_4), (\overline{x_3} \vee x_5), (\overline{x_4} \vee \overline{x_5}), (x_4 \vee \overline{x_3})\}$$



假设在我们的公式里存在只有一个文字的子句,比方说在式(2)里的第三个子句(x_1)。那么显然这个文字在任何满足的真值赋值里都必须是 \top ,因此我们立即决定这个变元的值。即在本例中我们立即决定 $T(x_1) = \top$, 然后继续进行。既然我们知道 $T(x_1) = \top$, 我们就从公式里删除包含 x_1 作为文字的所有子句, 因为这些子句已经被满足了(在本例中我们删除第一个子句)。不过如果子句包含否定文字 \bar{x}_1 , 那么我们就从子句里删除这个文字, 因为这个文字是 \perp 因此它不能用来满足子句。在本例中第一个子句被删除,

我们把寻找单文字子句直到不存在这样的子句为止的过程称为清洗。如果在清洗的任何一步产生了空子句——假定因为对某个 i , 子句(x_i)和(\bar{x}_i)都在前一步出现——那么我们说清洗已经失败。无论如何, 在 $\mathcal{O}(n)$ 步之后, 其中 n 是给定公式的子句数, 这个清洗过程必定要么失败(在这种情形里我们判定公式是不可满足的), 要么停止并且剩下一组子句, 其中每个子句都有两个不同的文字。例如, 在式(2)里初始清洗结束时设置 $T(x_1) = \top, T(x_2) = \perp$, 并删除前四个子句。





因此假定我们的公式在每个子句里恰有两个文字。那么如何寻找可满足真值赋值呢？这里是简单的想法：选择还没有赋真值的任何变元，试验设置它的真值是 \top 并完成清洗；然后把公式恢复原状，把同一个变元设置成 \perp 并再次完成清洗。若两次清洗都失败则搜索结束，公式是不可满足的。若两次清洗中至少有一次成功，则设置变元等于成功的清洗中的真值并继续。在我们的例子里在第一遍清洗之后剩下四个子句

$$\{(x_3 \vee x_4), (\overline{x_3} \vee x_5), (\overline{x_4} \vee \overline{x_5}), (x_4 \vee \overline{x_3})\} \quad (3)$$

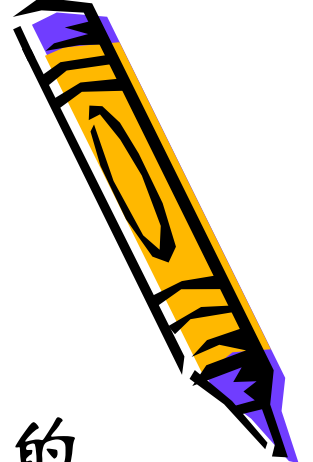
试验真值 $T(x_3) = \top$ 启动新的清洗，它在设置 $T(x_5) = \top$ 之后失败（产生子句 (x_4) 和 $(\overline{x_4})$ ），所以我们必须恢复在式(3)里的四个子句并试验 $T(x_3) = \perp$ 。这次成功地找到整个公式的可满足真值赋值（没有剩下任何子句），即 $T(x_4) = \top$ 和 $T(x_5) = \perp$ 。

容易看出(问题 6.3.2)当每个子句最多有两个文字时，这个简单算法正确地解决可满足性问题。因为算法对每个变元最多完成两遍清洗并且每遍清洗都在多项式时间里完成，所以由此得出二元可满足性属于 P。



5.2.2 可归约性

- 如何确定一个问题是属于P类的，最直接的办法就是设计一个多项式时间算法来解决这个问题。
- 另一种办法就是将这个问题归约到一个已知的P类问题。



5.2.2 可归约性

- 可归约性是一个功能强大的工具,它在可计算性理论中的两个主要应用,一是解决问题(把一个问题A转化为问题B,通过解决B来解决A),二是证明某些问题的不可判定性,即通过适当的归约来证明一个给定的语言是不可判定的。



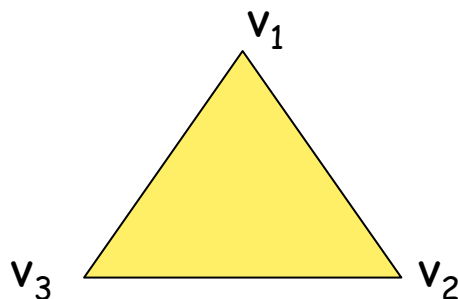
归约的例子

- 例：给定一个图 $G=(N,E)$, 2可着色问题需要判定一个全函数 $\text{color}: N \rightarrow \{1,2\}$, 存在否：当 $\langle u,v \rangle \in E$ 时, $\text{color}(u) \neq \text{color}(v)$ 。
- 如果不想直接设计一个算法来解决该问题，那么可以将其归约（转换）到一个已知的有解法问题来解决。
- 为每一个节点 n_i 赋一个布尔变量 x_i ，用 x_i 取真表示 n_i 赋着红色而取假表示着黄色：对每一条边 (n_i, n_j) ，有 $\sim((x_i \wedge x_j) \vee (\sim x_i \wedge \sim x_j)) = (x_i \vee x_j) \wedge (\sim x_i \vee \sim x_j)$ ，即两个变量不能同时为 true 或同时为 false。

这就是
归约

于是图 G 的 2可着色问题被转换为(归约为) 2变量公式集的可满足问题。

- 对下图,有公式集:
- $S = \{ (x_1 \vee x_2) \wedge (\sim x_1 \vee \sim x_2), \\ (x_1 \vee x_3) \wedge (\sim x_1 \vee \sim x_3), \\ (x_2 \vee x_3) \wedge (\sim x_2 \vee \sim x_3) \}$
- 这是一个2变量公式集的可满足问题。我们用清洗法可以在多项式时间内解决这个问题(本例答案是NO),从而证明了2可着色问题也是多项式时间内解决的问题,即P类问题。



5.2.2 可归约性

- 称语言 A 是可归约到语言 B (用 $A \leq_m B$ 来表示): 当存在一个完全可计算函数 f , 对于任意的 $x \in \Sigma^*$, 有:

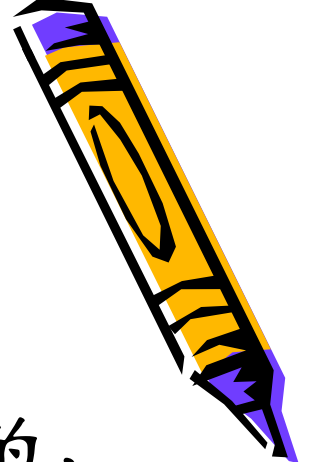
$$x \in A \leftrightarrow f(x) \in B.$$

则函数 f 被称为 A 与 B 之间的 m -归约。

- 当 f 是一一映射时, 称一一归约, 记 $A \leq_1 B$.

5.2.2 可归约性

- 引理：如果 $L_1 \leq_m L_2$ ，并且 L_1 是不可判定的，则 L_2 也是不可判定的。
- 证明：假设存在图灵机 T_2 判定 L_2 ，并且从 L_1 到 L_2 的归约函数为 f 。则对 $x \in L_1$ ？：可以计算 $f(x)$ ，然后使用 T_2 来验证是否有 $f(x) \in L_2$ ，从而判定 $x \in L_1$ ，或者 $x \notin L_1$ 。这与 L_1 是不可判定的矛盾。因此， L_2 也不能被判定。



5.2.2 可归约性

- 例：证明语言 $L_{halt-e} = \{T: T(e) \text{ 停机}\}$ ，即计算图灵机在空句子 e 上停机，是不可判定的。
- 语言 L_{halt} 是已知的不可判定问题，上述问题的证明可以通过语言 L_{halt} m -归约到语言 L_{halt-e} 来证明。
- 给定任意图灵机 T 和任意输入 x ，总是存在一个图灵机 T' ，在 T' 上运行空句子，即 $T'(e)$ 停机，当且仅当 $T(x)$ 停机。具体做法如下： T' 将 x 存储在内部（由于 x 是有限的，一定可以做到），首先将 x 写到输入带上，然后开始模拟 T ，显然这种从 $\langle T, x \rangle$ 到 T' 的转换是一个 m -归约。



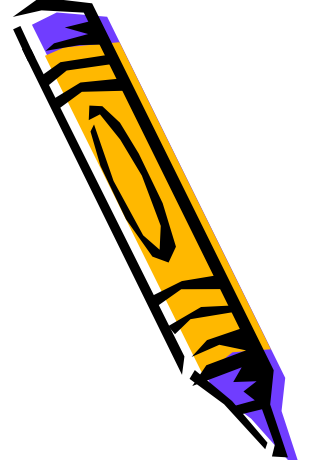
5.2.2 可归约性

- m-归约的性质:
- (1) \leq_m 是自反的, 传递的。对于所有的语言 $A, B, C \subseteq \Sigma^*$,

$$A \leq_m A \quad \text{and} \quad A \leq_m B \wedge B \leq_m C \rightarrow A \leq_m C.$$

- (2) 对于所有的语言对 $A, B \subseteq \Sigma^*$

$$A \leq_m B \rightarrow A^c \leq_m B^c.$$



5.2.2 可归约性

- (3)定理：任意语言 B , $B \neq \Phi$, $B \neq \Sigma^*$, 对于任意的可判定语言 A , $A \leq_m B$ 。
- 证明：用 y 表示 B 中的一个句子, z 表示 B^c 中的一个句子(根据 B 的定义, y 和 z 必然存在), 定义函数 f :

$$f(x) = \begin{cases} y & \text{if } x \in A, \\ z & \text{otherwise.} \end{cases}$$

由于 A 是可判定的, f 是全的可计算函数。

又 $x \in A \leftrightarrow f(x) \in B$, 因此 f 是 A 到 B 的 m 归

约。

5.2.3 多项式时间归约

- 定义5-15 称语言 L_A 多项式时间映射可归约到语言 L_B , 记为

$$L_A \leq_p L_B,$$

若存在多项式时间可计算函数 $f: \Sigma^* \rightarrow \Sigma^*$,
对于每一个 w , 有

$$w \in L_A \leftrightarrow f(w) \in L_B$$

- 称函数 f 为 L_A 到 L_B 多项式时间归约。

5.2.3 多项式时间归约

- 定理5-18 L_A 与 L_B 是两个语言, 若 $L_A \leq_P L_B$, 且 $L_B \in P$, 则 $L_A \in P$ 。

证明: 设 M 是判定 L_B 的多项式时间算法, f 是从 L_A 到 L_B 的多项式时间归约。判定 L_A 的多项式时间算法 M_A 如下:

对于输入 w ,

步1 计算 $f(w)$;

步2 在输入 $f(w)$ 上运行 M , 输出 M 的输出。

因为 $w \in L_A \leftrightarrow f(w) \in L_B$, 又 f 、 M 都是多项式时间可计算的, 所以 M_A 也是多项式时间可完成的。

5.2.4 NP 类

即这台机器的计算不会持续到超过多项式那么多步。定义 NP(表示非确定型多项式)是被多项式界限非确定型 Turing 机判定的所有语言的类。

此时此刻重要的是回忆对非确定型 Turing 机判定语言 L 是什么意思所下的独特定义：对每个不属于 L 的输入，机器的所有计算都必须拒绝输入；对每个属于 L 的输入，我们仅仅要求存在至少一个计算接受输入——只要存在一个接受计算，在其余的计算里就可以没有、或有些、或大多数、或全部都拒绝这个输入。

非确定型 Turing 机在给定输入上所有可能的计算最好是画成树形的结构（见图 6-3）。顶点表示格局，向下的线表示步。非确定性选择表示成有多条线离开的顶点。用垂直维度量时间。例如在图 6-3 里，输入在 5 步之后被接受。

这样的图形使得非确定性是如此强的计算方式的原因变得一目了然：在非常短的时间里（从根上算起的垂直距离）产生了天文数字那么多的格局。我们在下一个例子将看到用非确定性的这种能力来“解决”在上一节我们看到的某些棘手问题。

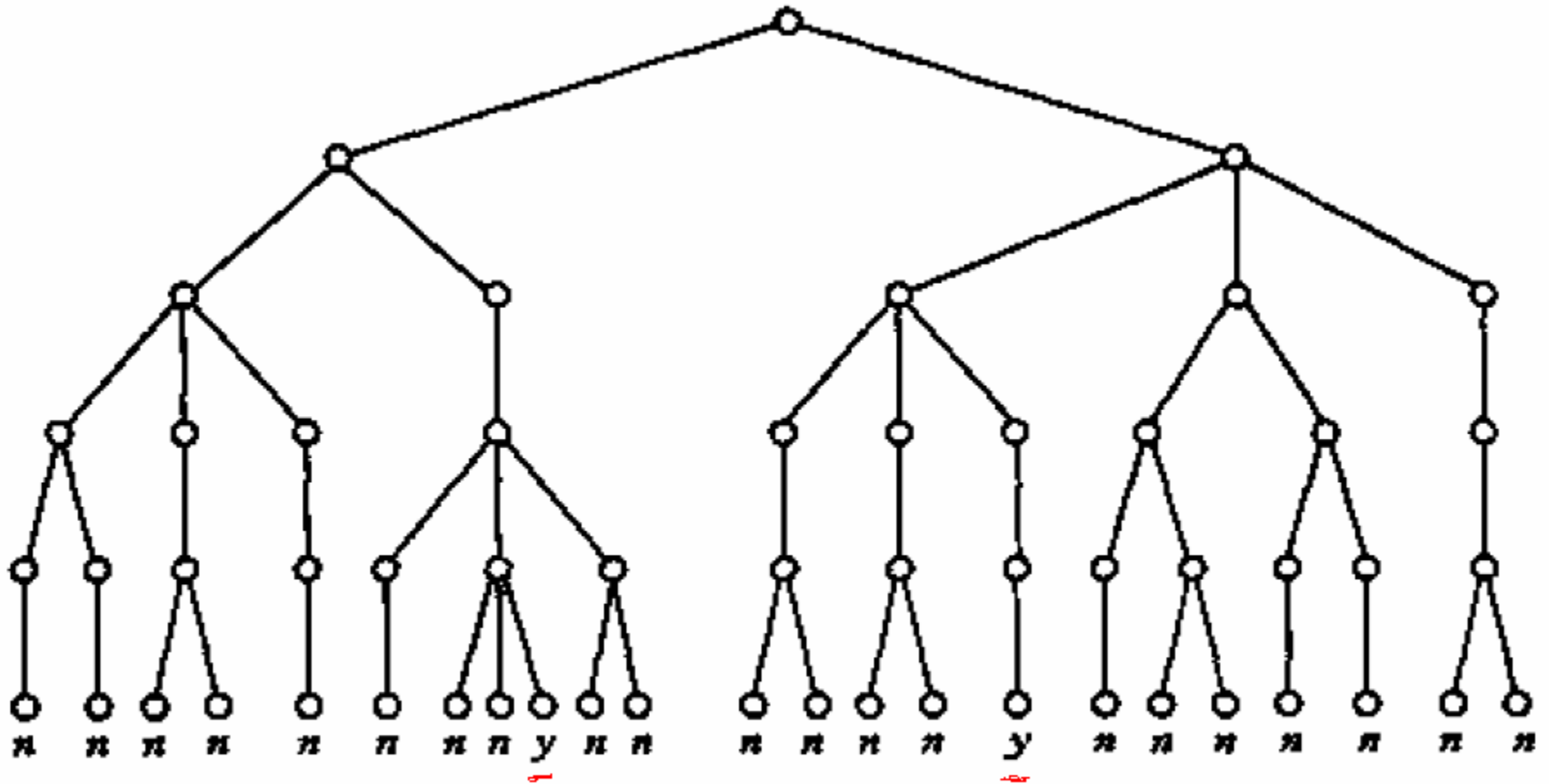
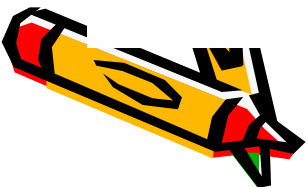
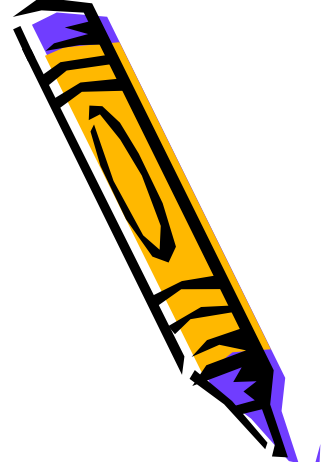


图 6-3





NP类语言(问题)的另外一个描述：它们相当简单和有些相似的：它们首先非确定性地产生字符串，然后确定性地验证所产生的字符串是否具有与输入相关的某种需要的性质。若输入属于这个语言则至少存在一个合适的字符串。若输入不属于这个语言则找不到具有所需要性质的字符串。

这样的字符串称为证书,或者见证。我们将看到所有 NP 里的问题都有证书,并且只有 NP 里的问题才有证书。证书必须是多项式那么短的,即它的长度最多是输入长度的多项式。它还必须是在多项式时间里可验证的。





SAT问题

- SAT问题也称为合取范式的可满足问题，一个合取范式形如： $A_1 \wedge A_2 \wedge \dots \wedge A_n$ ，子句 A_i ($1 \leq i \leq n$) 形如： $a_1 \vee a_2 \vee \dots \vee a_k$ ，其中 a_i 为基本式（文字），为某一布尔变量或该布尔变量的非。SAT问题是指：是否存在一组对所有布尔变量的赋值（TRUE或FALSE），使得整个合取范式取指为真。
- 理解这样一个观点：NP问题的算法具有指数时间，即算法的计算时间随着问题规模 n 的增长而快速增长。

$$F' = \{(x_1 \vee x_2 \vee x_3), (\overline{x_1} \vee x_2), (\overline{x_2} \vee x_3), (\overline{x_3} \vee x_1), (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})\}$$

它是否可满足？在这里正确回答是“否”。让我们证明它。

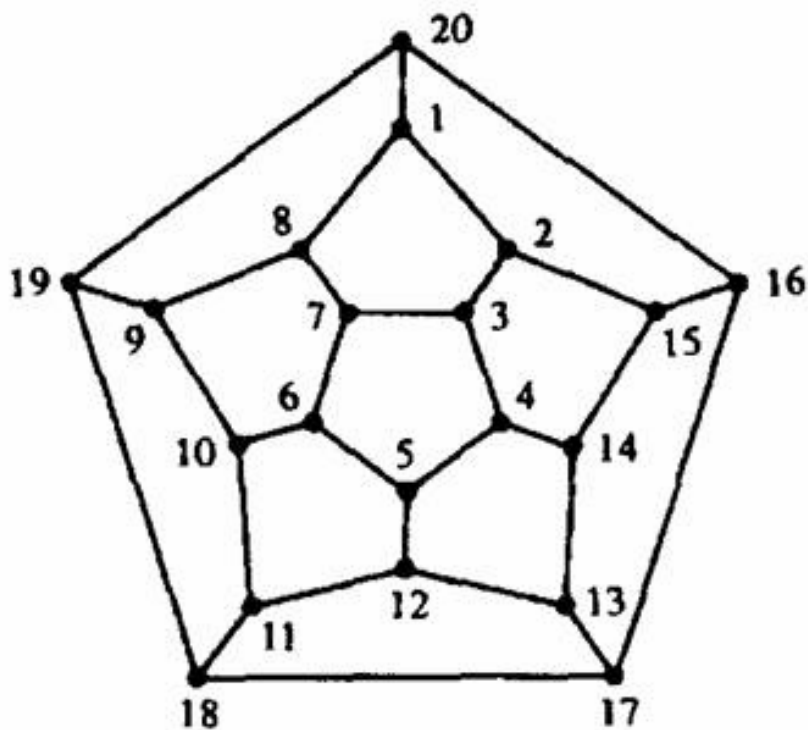


真值表法（时间复杂度是指数级的）

[BACK](#)

Hamilton(哈密尔顿)问题

- 图论中的著名问题之一。英国数学家哈密尔顿于**1859**年以游戏的形式提出：把一个正十二面体的二十个顶点看成二十个城市，要求找出一条经过每个城市恰好一次而回到出发点的路线(如图)。这条路线就称“哈密顿圈”。





下列算法确实解决这个问题：

- 检查所有可能的顶点排列；

- 对每种排列验证它是否 Hamilton 圈。

不幸的是，做了所有简单的改进和加速之后，它仍不是多项式的。



5.2.4 NP类

- Hamilton路问题的验证机设计如下：

对于输入图 G ，节点 s 、 t ，

步1 写一系列 m 个数 v_1, v_2, \dots, v_m ， m 是 G 的节点数，列中的每一数，都是从1到 m 中非确定挑选的；

步2 在列中检查重复性，若发现重复，则拒绝；

步3 检查 $s=v_1, t=v_m$ 是否成立。若有一个不成立，则拒绝；

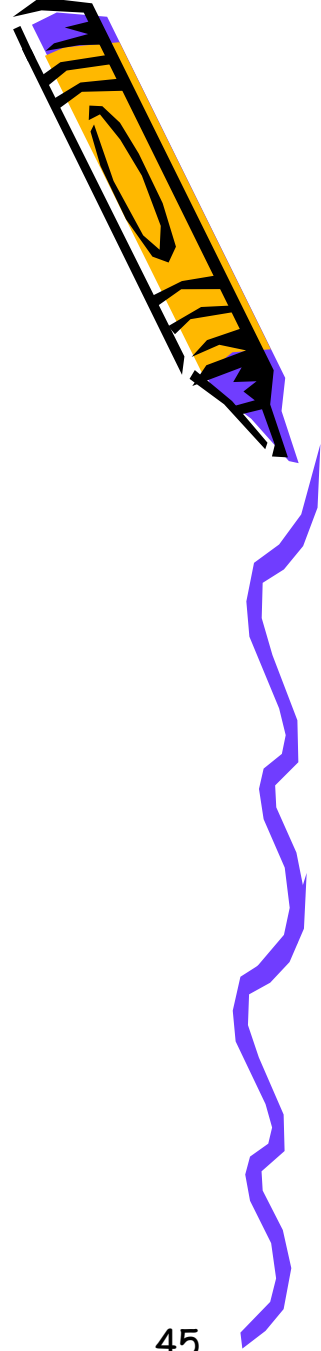
步4 对于 $i=1, 2, \dots, m-1$ ，检查 (v_i, v_{i+1}) 是否是 G 的边，若都是 G 的边，则接受；否则拒绝。



5.2.4 NP类

- 由于确定的图灵机是一种特殊的不确定的图灵机，所以，所有的P类问题都是NP类问题。
- 但是，是否所有的NP类问题都是P类问题？也就是说， $P=NP$ 成立吗？换句话说，在多项式时间内，所有非确定TM都能被确定的TM所模拟吗？这个问题是理论计算机科学和当代数学中悬而未决的问题之一。否定它和证明它都是令人难以置信的困难！
- 例如，SAT问题至今也没有找到一个多项式时间复杂度的解法；但是也没有人能证明SAT问题一定不属于P。这两个努力的任何一个突破都将是惊人的结果。
- 如果这两类相等，那么所有的多项式可验证的问题都将是多项式可判定的。大多数研究人员相信它们不相等。

5.3 NP完全性



NP完全性

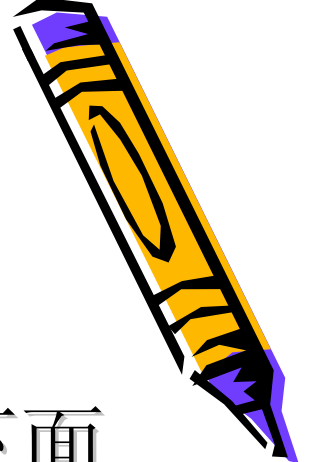
- NP完全问题在互联网应用中大量出现,成为新热点。 **LEWIS**说,快速识别出它们是一种重要能力。可以摆脱无谓的尝试,把努力定向到有希望的路径上。
- 5.3.1 NP完全性定义
- 5.3.2 NP完全问题
- 5.3.3 NP完全性的计算讨论



5.3.1 NP完全性

定义7-1 语言 L 是NP完全的, 若它满足下面两个条件:

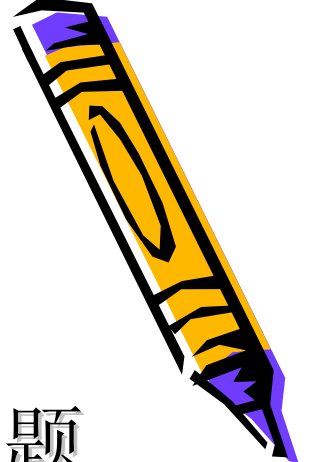
- (1) $L \in \text{NP}$;
- (2) NP中的每个 L_A 都多项式时间可归约为 L , 即 $L_A \leq_p L$ 。



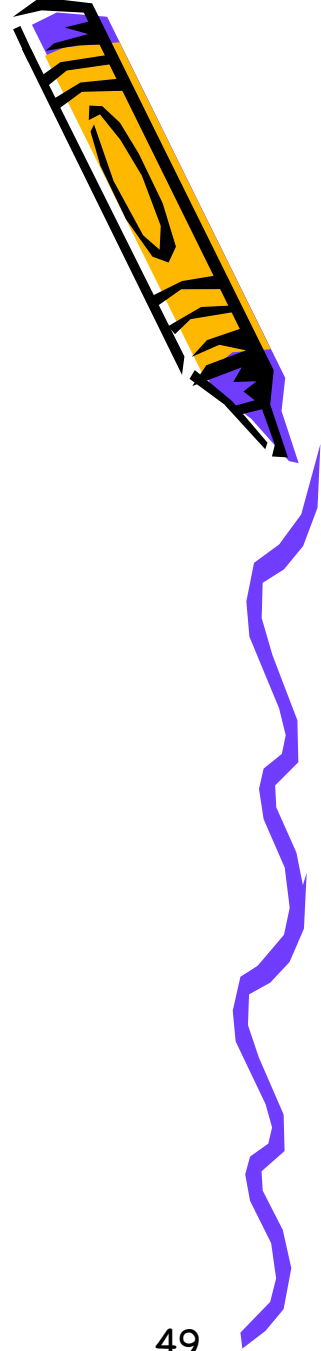
5.3.2 NP完全问题

- 定理7-19 [库克—列文定理] 可满足问题 SAT 是 NP 完全的。

证明：NP 中的每一个语言都可以多项式时间归约到 SAT。从 NP 中任取一个语言 L，设 $M_N = \{Q, \Sigma, \Gamma, \delta, q_0, B, F\}$ 是 n^k 多项式时间内判定 L 的非确定型图灵机，其中 k 是常数。对于 M_N 的任意输入 w ，在多项式时间内构造公式 Φ ，使得 $w \in L \leftrightarrow \Phi \in \text{SAT}$ 。



$(q,0)$	1	0	1	1	B
0	$(q,1)$	0	1	1	B
0	1	$(q,0)$	1	1	B
0	1	0	$(q,1)$	1	B
0	1	0	1	$(q,1)$	B
0	1	0	1	1	(q, B)
0	1	0	1	$(p,1)$	B
0	1	0	$(p,1)$	0	B
0	1	$(p,0)$	0	0	B
0	1	$(p_f,1)$	0	0	B



BACK

5.3.2 NP完全问题



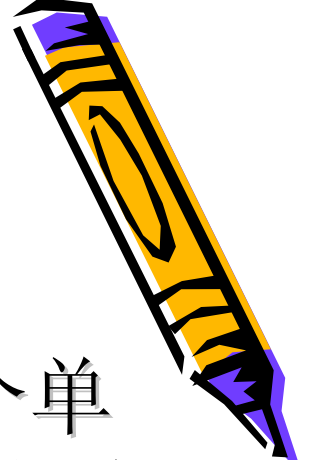
设 f 是由 w 到 Φ 的归约，下面开始描述归约 f 。

考虑 M_N 在 w 上的执行过程。定义 M_N 在 w 上的画面是一张 $n^k \times n^k$ 表格，其中行代表 M_N 在输入上的一个计算分支的瞬时描述。并约定每一个ID都以“#”开始和结束。当然画面的第一行是初始ID，每一行都根据 M_N 的转移函数从上一行得到，如果画面的某一行是接受ID，则称该画面是接受的。



5.3.2 NP 完全问题

我们把画面 $n^k \times n^k$ 个的格子中每一格称为一个单元。第 i 行第 j 列的单元称为 $\text{cell}[i,j]$ ，它应包含 $C = Q \cup \Gamma \cup \{\#\}$ 中的一个符号。定义变量 $x_{i,j,s}$ ($1 \leq i \leq n^k, 1 \leq j \leq n^k, s \in C$) 表示单元中的内容。 $x_{i,j,s} = 1$ ，意味着 $\text{cell}[i,j]$ 包含 s 。



5.3.2 NP完全问题



c	q1	a
q2	c	b

当 $\delta(q1, a) = (q2, b, L)$ 时

q1	a	c
q2	b	c

当 $\delta(q1, a) = (q2, b, S)$ 时

q1	a	c
b	q2	c

当 $\delta(q1, a) = (q2, b, R)$ 时

图 7-2 画面中行间瞬时描述的可能变化情形

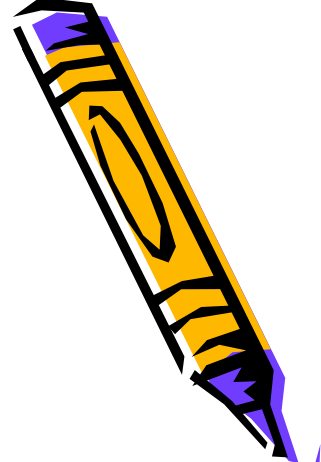


5.3.2 NP完全问题

现在设计公式 ϕ ，使得变量的一个满足赋值确实对应 M_N 在 w 上的一个接受画面。为此要满足以下4点：

- 每个单元只能包含一个符号；
- 第一行为初始ID
- 存在接受ID
- 表的每一行都对应于从上一行的ID、按照 M_N 的规则、合法转移得到ID。

5.3.2 NP 完全问题



而其它这样的 6 个单元格中，同列两行的单元包含相同的符号。我们将满足上面规则的 6 个单元的包含的符号记为 $c_1, c_2, c_3, c_4, c_5, c_6$ (见图 7-3), 并称为合法的。

这 4 个方面，分别用公式 ϕ_{cell} 、 ϕ_{start} 、 ϕ_{move} 、 ϕ_{accept} 描述，
则

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

其中

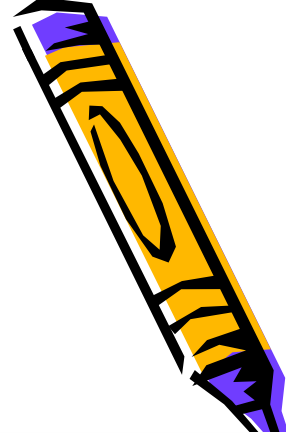
$$\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n-1} \left[\left(\bigvee_{s \in C} X_{ijs} \right) \wedge \left(\bigwedge_{\substack{s, t \in C \\ s \neq t}} \left(\overline{X_{ijs}} \vee \overline{X_{ijt}} \right) \right) \right]$$

c_1	c_2	c_3
c_4	c_5	c_6

图 7-3 合法的 6 个单元



5.3.2 NP完全问题



$$\phi_{\text{start}} = X_{1,1,\#} \wedge X_{1,2,q_0} \wedge X_{1,2,a_1} \wedge \dots \wedge X_{1,n+2,a_n} \wedge X_{1,n+3,\cup} \wedge X_{1,n^1-1,\cup} \wedge \dots \wedge X_{1,n^1,\#}$$

其中 $\cup = a_1 a_2 \dots a_n$; \cup 表示空格。

$$\phi_{\text{accept}} = \bigvee_{\substack{1 \leq i,j \leq n^1 \\ q_f \in F}} X_{i,j,q_f}$$


$$\phi_{\text{move}} = \bigwedge_{1 \leq i,j \leq n^1} \left[\bigvee_{\substack{c_1, c_2, c_3, c_4, \\ c_5, c_6 \text{ 是合法的}}} (X_{i,j-1,c_1} \wedge X_{i,j,c_2} \wedge X_{i,j+1,c_3} \wedge X_{i+1,j-1,c_4} \wedge X_{i+1,j,c_5} \wedge X_{i+1,j+1,c_6}) \right]$$

其中 $c_1, c_2, c_3, c_4, c_5, c_6$ 是合法的两行三列 6 个单元的内容，见图 7-3。



显然，若 $w \in L$ 当且仅当 $\phi \in \mathbf{SAT}$ ，即 ϕ 是可满足的，于是把任一 **NP** 问题转化（规约）为 **SAT** 问题。

5.3.2 NP 完全问题



c1	c2	c3
c4	c5	c6

现在证明上面由 w 到 Φ 的映射可以在多项式内完成。画面是一个 $n^k \times n^k$ 的表格，所以它包含 n^{2k} 个单元，每个单元有与它相关的 l 个变量， $l = |C|$ ，因为 l 只依赖于 M_N ，所以变量的总数是 $O(n^{2k})$ 。

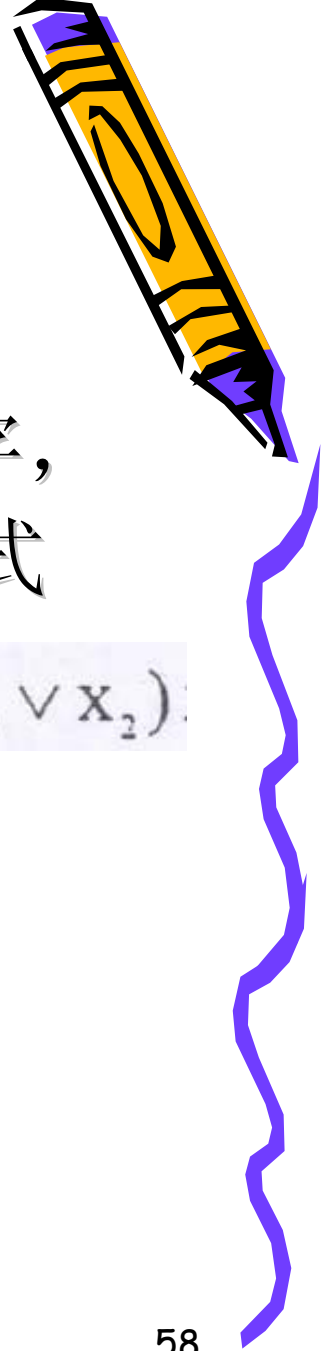


5.3.2 NP 完全问题

对画面的每个单元，公式 Φ_{cell} 包含固定长度的公式片段，故长度为 $O(n^{2k})$ ， Φ_{start} 对顶行的每个单元包含一个片段，所以长度为 $O(n^k)$ ， Φ_{move} 和 Φ_{accept} 对画面的每个单元包含固定长度的公式片段，所以它们的长度为 $O(n^{2k})$ ，于是 Φ 总长度为 $O(n^{2k})$ ，因此可在多项式时间内从 w 生成 Φ ，所以，SAT 是 NP 完全问题。■



5.3.2 NP完全问题



- **3SAT**问题：将布尔变量或其非称为文字，将由三个文字构成的子句组成的合取范式

- 例如：
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

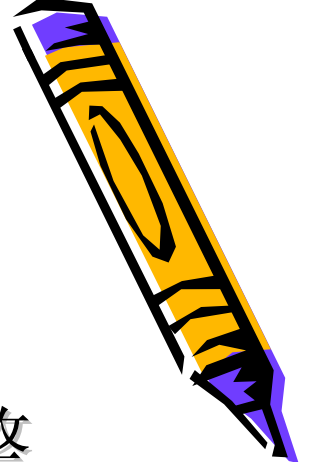
- 推论7-3 3SAT是NP完全的。

证明：证明SAT多项式时间归约到3SAT。



5.3.2 NP完全问题

- **Node Cover**问题: 若 G 是无向图, k 是一整数, 则 G 的顶点覆盖问题是否存在最小的一个节点子集 M , 使得 $|M| \leq k$, 且 G 的每条边都与子集 M 的节点之一关联。
- 注意顶点覆盖问题要求最小的顶点覆盖规模。



5.3.2 NP完全问题

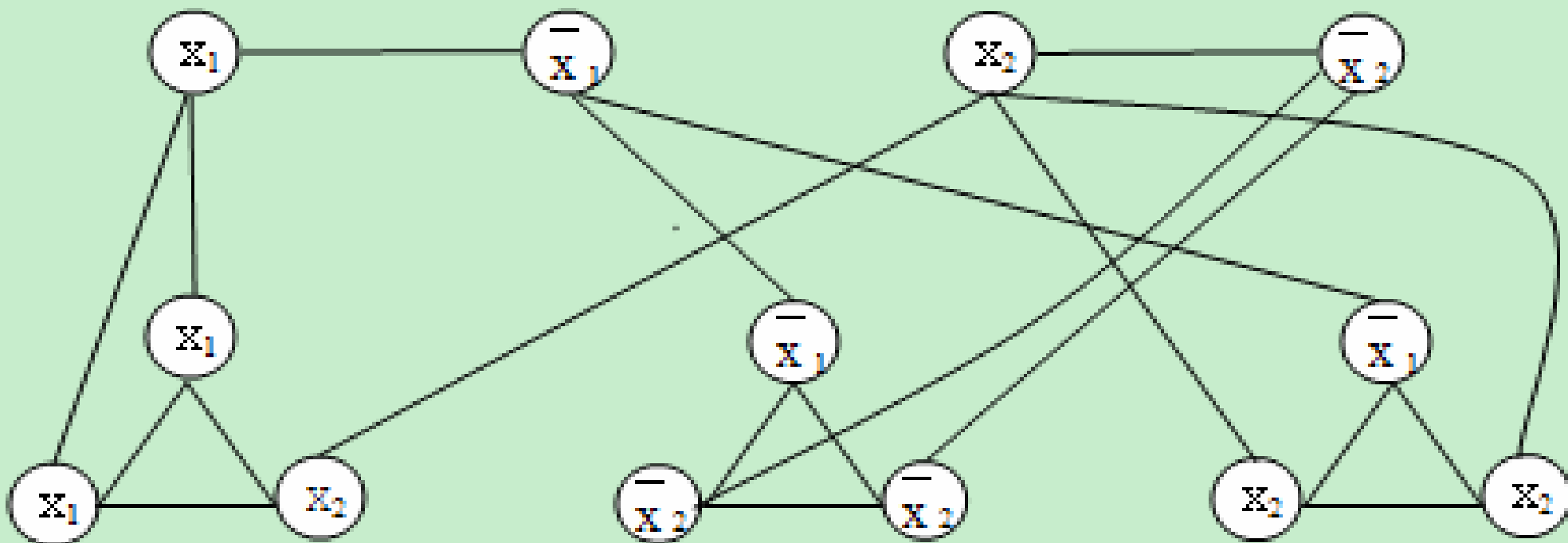
“顶点覆盖问题 (Node Cover) 是NP完全的”

证明:我们可以给出一个3SAT到这个问题的多项式时间内运算的归约, 即把布尔公式 ϕ 映射为 $\langle G, k \rangle$, 其中 G 是一个图, k 是一个正数, 表示 G 的顶点覆盖子集的节点数。为此,

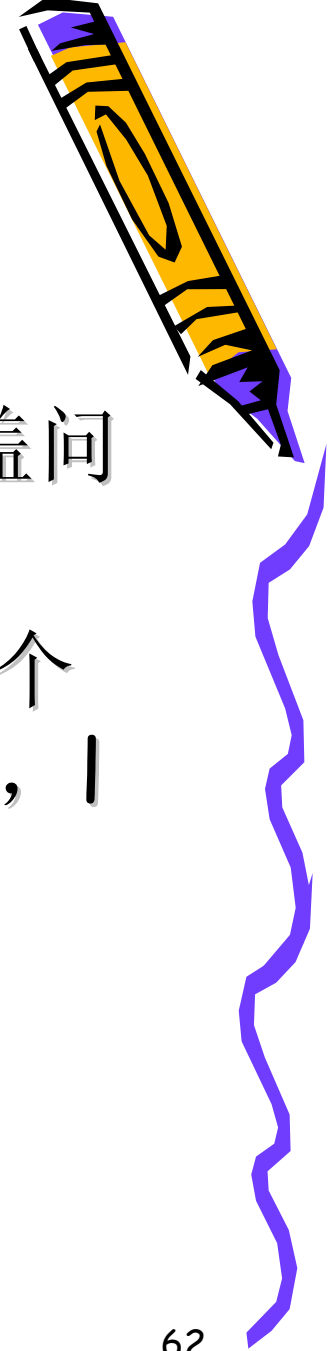
(1) 对于 $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$ 边, 边的两个顶点记为 x 和 $\neg x$ 。当 x 赋值为 TRUE 时, 表示对应的覆盖选择 x ; 当 x 赋值为 FALSE 时, 表示对应的覆盖选择 $\neg x$ 。

5.3.2 NP完全问题

(2) ϕ 子句中的三个文字对应于 G 的三个节点，这三节点相互连接，并分别于(1)中具有相同标记的节点相连。



5.3.2 NP完全问题



易见，3SAT可多项式时间归约为顶点覆盖问题。

不难证明 Φ 可满足当且仅当 G 中有 $k=m+2l$ 个节点的顶点覆盖，其中 m 是 Φ 的变元数， l 是 Φ 子句数。

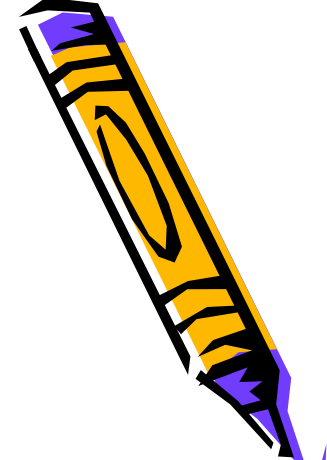


5.3.2 NP完全问题

对于一个满足赋值，

- (1) 把变量构件中真文字节点放入 M 中，
- (2) 把子句中，除为真的文字外的另两个文字加入 M 中。

可见 M 中 G 的一个覆盖。



5.3.2 NP完全问题

对于 G 的覆盖 M ,

把 M 中的文字置为真。因为 M 中包含每个变量构件的一个节点和子句构件的两个节点, 所以这个赋值满点3SAT。



5.3.2 NP完全问题

- **Clique**（团）是图的一个子图，其中任何两点都有边相连。
- **K-Clique**问题：给定 $\langle G, k \rangle$, G 是无向图， G 中包含至少 k 节点的团吗？
- **k-Clique**是NP完全的

证明：

(1) **k-Clique**是NP问题

(2) $3SAT \leq_p k\text{-Clique}$



5.3.2 NP完全问题



设 ϕ 是3SAT公式，子句数是 k ，则 G 中的节点分成 k 组，每组三个节点，称为三元组，每个三元组对应 ϕ 中的一个子句。 G 中的边定义为：不同三元组的节点，若不是相反文字，则两两相连；三元组内节点间无边。如，

可见 $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

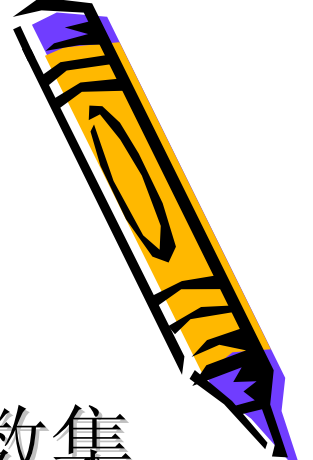
(1) ϕ 有一可满足赋值， G 存在一个 k -团；

(2) G 存在一个 k -团， ϕ 可满足。



5.3.2 NP完全问题

- SUBSET-SUM（子集和）问题:有一个数集 x_1, x_2, \dots, x_k 和一个目标数 t ，要判定数集是否包含一个加起来等于 t 的子集。
- 子集和问题是NP完全的
 - (1) 子集和问题属于NP类
 - (2) $3SAT \leq_p SUBSET-SUM$



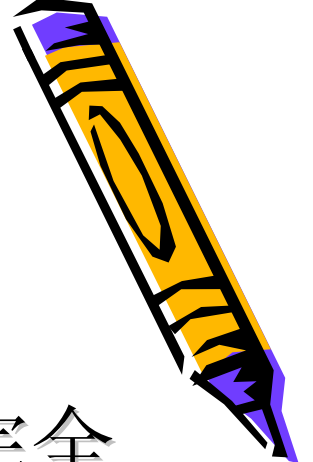
5.3.2 NP完全问题

- **HAMPATH** (Hamilton 路)问题也是NP完全的。

(1) **HAMPATH** 问题属于NP类;

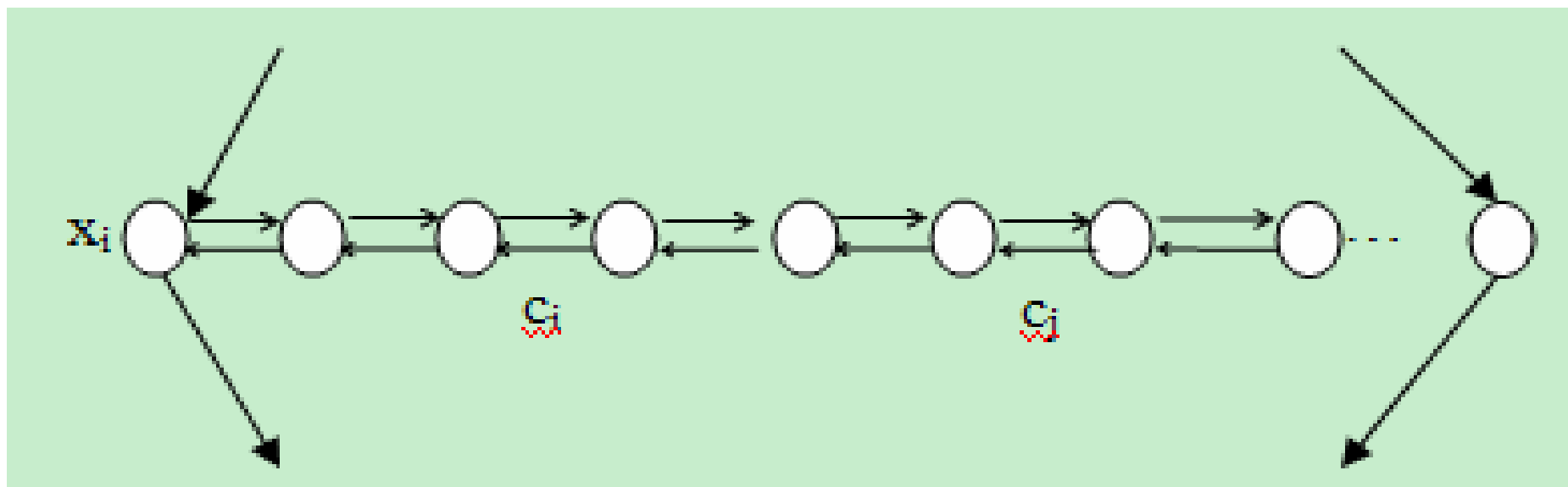
(2) $3SAT \leq_p \text{HAMPATH}$ 。

从 Φ 构造图 **G**: 把变量 x_i 表示为钻石结构, 把子句 c_i 表示为节点。



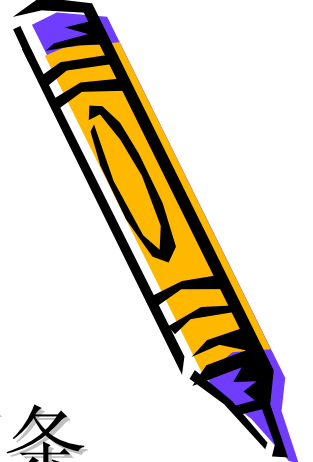
5.3.2 NP完全问题

每个钻石结构都包含一行水平节点，它们由双向边连接起来。



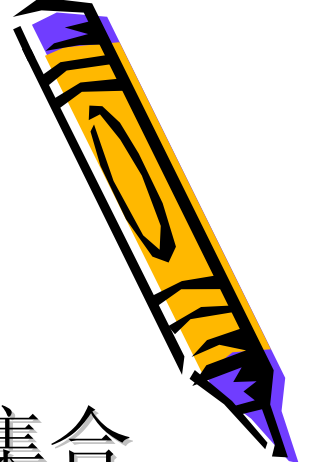
5.3.2 NP完全问题

- 如果变量 x_i 出现在子句 c_j 中，就把图中两条从第 i 个钻石的第 j 对节点到第 j 个子句节点的边添加进去。
- 如果变量 $\neg x_i$ 出现在子句 c_j 中，就把图中两条从第 i 个钻石的第 j 对节点到第 j 个子句节点的边添加进去（这两条边与前者方向相反）。



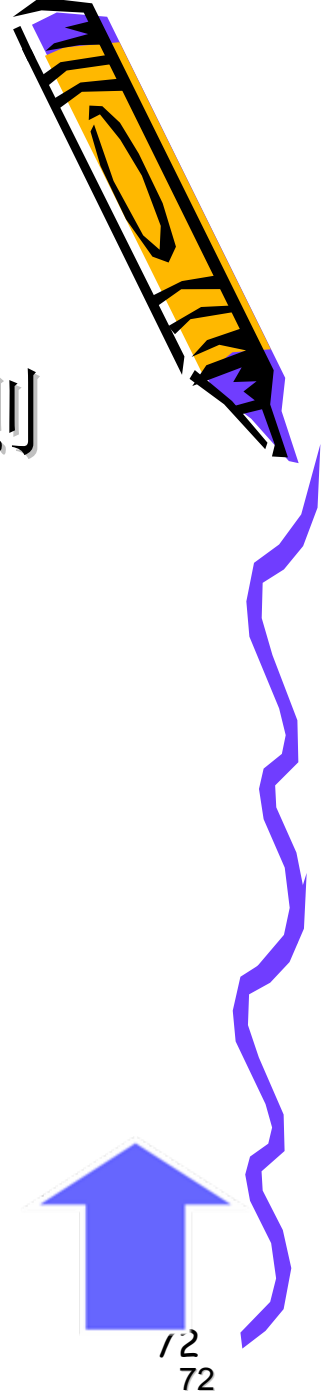
5.3.2 NP完全问题

- **Hitting Set**问题：给定集合 A 的子集的集合 C 和整数 k ， A 是否有一个子集 A' ， $|A'|=k$ ，且对于 C 中的任意元素（ A 的子集）， A' 至少包含其中的一个元素。



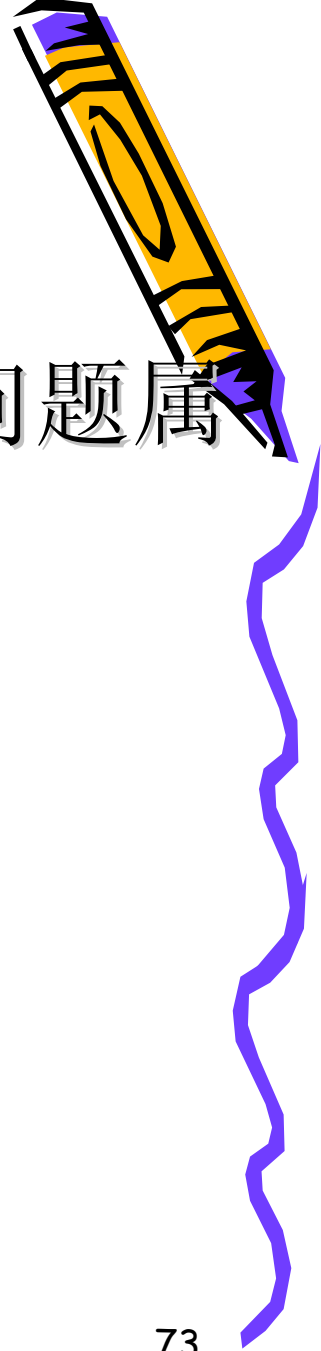
5.3.3 NP完全性的计算讨论

定理7-18 若 $L_A \leq_p L_B$, 且 $L_B \in P$, 则
 $L_A \in P$ 。



5.3.3 NP完全性的计算讨论

定理7-17 [库克—列文定理] 可满足问题属于P，当且仅当 $P=NP$ 。



NP问题的近似解法示例



例 7.4.2 让我们描述有关顶点覆盖的 1 近似算法,即对任意图,返回最多两倍于最优解规模的顶点覆盖的算法。算法非常简单:

$C := \emptyset$

while 在 G 里存在剩余的边 $[u, v]$ do

 把 u 和 v 添加到 C 里并从 G 里删除它们



ϵ 近似公式:

$$\frac{|\text{opt}(x) - A(x)|}{\text{opt}(x)} \leq \epsilon$$





例如在图 7-13 里,算法开始选择边 $[a,b]$ 并把两个端点都插入到 C 里;然后从 G 里删除这两个顶点(当然还有它们所关联的边)。然后选择 $[e,f]$,最后选择 $[g,h]$ 。所得出的集合 C 是顶点覆盖,因为在 G 里的每条边必须接触到它里面的一个顶点(要么是因为这个算法选择了它,要么是因为这个算法删除了它)。在这个例子中, $C=\{a,b,e,f,g,h\}$ 有 6 个顶点,它最多两倍于最优值,最优值是 4。

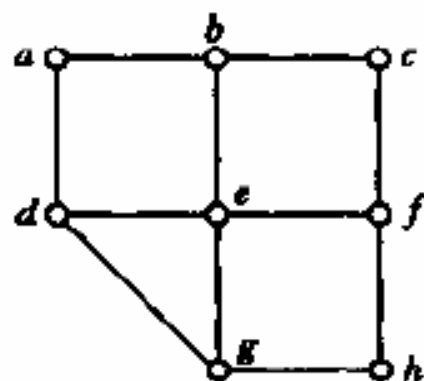
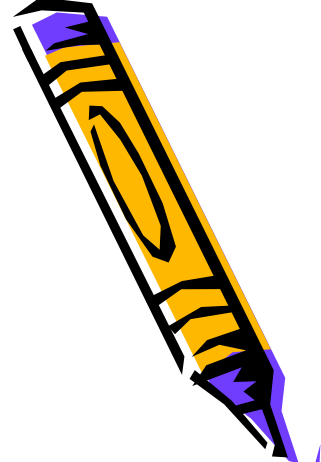


图 7-13

为证明“最多两倍”的保证,考虑算法所返回的覆盖 C ,并设 \hat{C} 是最优顶点覆盖。 $|C|$ 恰好两倍于算法所选择的边数。而通过上述方式所选择的这些边都没有公共顶点,并且对它们中的每条来说至少有一个顶点必





须属于 \hat{C} , 因为 \hat{C} 是顶点覆盖。由此得出算法所选择的边数不大于最优顶点覆盖中的顶点数, 因此 $|C| \leq 2 \cdot |\hat{C}|$, 所以这个算法确实是 1 近似算法。

我们能否做得更好? 令人失望的是, 这个简单算法是有关顶点覆盖的已知的最好算法。仅仅在不久以前我们才证明除非 $P=NP$, 对任意 $\epsilon < \frac{1}{6}$, 不存在有关顶点覆盖的 ϵ 近似算法。



The End

