第8章 空间复杂度

主要内容

空间复杂性

- 在寻找计算问题的可行解时，时间与空间是最重要的两个需要衡量的因素。本章从计算问题所需要的空间大小出发考察其复杂性。
- 与时间复杂性类似，需要选择一个模型来度量算法所需要消耗的空间。TM模型：属性形式简单、近似实际计算机。

空间复杂性

定义 8.1

令M是一个在所有输入上都停机的确定型图灵机。M的空间复杂度是一个函数f：N →N，其中f(n)是M在任何长为n的输入上扫描带方格的最大数。若M的空间复杂度为f(n)，也称M在空间f(n)内运行。

如果M是对所有输入在所有分支上都停机的非确定型图灵机，则定义它的空间复杂度f(n)为M在任何长为n的输入上，在任何计算分支上所扫描的带方格的最大数。

与时间复杂度类似——渐近记法估计图灵机的空间复杂度

空间复杂性

定义 8.2

令f：$N \to R^+$是一个函数，空间复杂性类SPACE(f(n))和NSPACE(f(n))定义如下：

SPACE(f(n))={L|L是被O(f(n))空间的确定型图灵机判定的语言}

NSPACE(f(n))={L|L是被O(f(n))空间的非确定型图灵机判定的语言}

空间复杂性

例8.3 NP完全问题SAT，证明用线性空间算法能求解SAT问题。因为SAT是NP完全的，所以SAT不能用线性时间算法求解。

因为空间可重用，时间不能，所以空间的能力比时间强

算法：M_1="对输入<ϕ>，ϕ是布尔公式：

1)对于ϕ中变量X_1，…，X_m的每个真值赋值：

2)计算ϕ在该真值赋值下的值。

3)若ϕ的值为1，则接受；否则拒绝。"

显然机器M_1 是在线性空间内运行，因为每一次循环都可以复用带子上的同一部分。该机器只需存储当前的真值赋值，这只需消耗O(m)空间办到。因为变量数m最多是输入长度n，所以该机器在空间O(n)内运行。

空间复杂性

例8.4非确定性空间复杂性。判定一个非确定型有穷自动机是否接受所有字符串问题。

令ALL_{NFA}＝{<A>|A是一个NFA且L(A)= Σ*}

- 首先给出非确定型线性空间算法来判定该语言的补$\overline{ALL_{NFA}}$。算法的思想是利用非确定性猜测一个被NFA拒绝的字符串，然后用线性空间跟踪该NFA，看它在特定时刻处在什么状态。

N＝“对于输入<M>，M是NFA：

1）置标记于NFA的起始状态。

2）重复执行下面的语句2^q次，这里q是M的状态数。

3）非确定地选择一个输入符号并移动标记到M的相应状态，来模拟读取那个符号。

4）如果步骤2和3表明M拒绝某些字符串，即如果在某一时刻所有标记都不落在M的接受状态上，则接受；否则拒绝。”

空间复杂性

- 如果M拒绝某个字符串，则它必定拒绝一个长度不超过2^q的字符串，因为在任何被拒绝的更长的字符串中，上面算法中所报述的标记的位置分布必定重复出现。介于两次重复出现之间的那一段字符串可以删去，从而得到更短的被拒绝的字符串。所以N可判定ALL_{NFA}的补。
- 该算法仅需要的空间是用来存放标记的位置和重复计数器，这在线性空间就可以得到解决。因此该算法在非确定的空间O(n)内运行。

§8.1 萨维奇定理

定理 8.5

萨维奇定理

对于任何函数f：$N \to R^+$，其中$f(n) \geq n$，

$NSPACE(f(n)) \subseteq SPACE(f^2(n))$

- ✓ 证明思路：需要确定地模拟一个f（n）空间的NTM。我们采用的方法思想：
- ✓ 给定NTM的两个格局c1和c2，以及一个整数t，要求判定该NTM能否在t步内从c1变到c2，称该问题的可产生性问题。如果c1是起始格局，c2是接受格局，t是非确定型机器所使用的最大步数，那么通过求解可产生性问题，就能判定机器是否接受输入。

§8.1 萨维奇定理

定理 8.5

萨维奇定理

对于任何函数f：$N \to R^+$，其中$f(n) \geq n$，

$NSPACE(f(n)) \subseteq SPACE(f^2(n))$

证明：设N是在空间f(n)内判定语言A的NTM。构造一个判定A的确定型TM M。机器M利用过程CANYIELD，该过程检查N的一个格局能否在指定的步数内产生另一个格局，它求解了在证明思路中所描述的可产生性问题。

设w是输入到N的字符串。对于N在w上的格局c_1，c_2以及整数t，如果从格局c_1出发，N有一系列非确定的选择能使它在t步内进入格局c_2，则CANYIELD(c_1，c_2，t)输出接受，否则，CANYIELD输出拒绝。

§8.1 萨维奇定理

定理 8.5

萨维奇定理

对于任何函数f：$N \to R^+$，其中$f(n) \geq n$，

$NSPACE(f(n)) \subseteq SPACE(f^2(n))$

CANYIELD="对输入c_1，c_2和t：

1）若t=1，则直接检查是否有$c_1=c_2$，或者根据N的规则检查c_1能否在一步内产生c_2。如果其中之一成立，则接受；如果两种情况都不成立，则拒绝。

2）若t>1，则对于N在w上消耗空间f(n)的每一个格局c_m：

3） 运行CANYIELD(c_1，c_m，t/2)。

4） 运行CANYIELD(c_m，c_2，t/2)。

5） 若第3和第4步都接受，则接受。

6）若此时还没有接受，则拒绝。"

§8.1 萨维奇定理

现在定义M来模拟N。首先修改N，当它接受时，把带子清空并把读写头移到最左边的单元，从而进入称为c_{accept}的格局。令c_{start}是N在w上的起始格局。选一个常数d，使得N在f(n)空间上的格局数不超过$2^{df(n)}$，其中n是w的长度。$2^{df(n)}$是N在所有分支上的运行时间的上界。

M＝“对输入w：

1）输出CANYIELD (c_{start}，c_{accept}，$2^{df(n)}$)的结果。”

算法CANYIELD显然求解了可产生性问题，因此M正确地模拟N。下面需要分析M，确信它在O(f(n))空间内运行。

§8.1 萨维奇定理

CANYIELD在递归调用自己时，它都把所处的步骤号、c_1、c_2和t的都存储在栈中，所以递归调用返回时能恢复这些值。因此在递归的每一层需要增加O(f(n))空间。此外，递归的每一层把t的值减小一半。开始时t等于$2^{df(n)}$，所以递归的深度是$O(\log 2^{df(n)})$ 或O(f(n))，因此共消耗空间是$O(f^2(n))$，正如断言所述。

§8.2 PSPACE类

定义 8.6

PSPACE是在确定型图灵机上、在多项式空间内可判定的语言类。换言之，

$$\mathbf{PSPACE} = \bigcup \mathbf{SPACE}(n^k)$$

PSPACE类的非确定型版本NPSPACE，可以类似地用NSPACE类来定义。然而，任何多项式的平方仍是多项式，根据萨维奇定理，则NPSPACE = PSPACE 。

- ✓ 在例8.3和8.4中，已经说明了SAT属于SPACE(n)， $\mathbf{ALL}_{\mathbf{NFA}}$ 属于coNSPACE(n)，而根据萨维奇定理确定性空间复杂度对补运算封闭，所以 $\mathbf{ALL}_{\mathbf{NFA}}$ 也属于 $\mathbf{SPACE}(n^2)$ 。因此，SAT和 $\mathbf{ALL}_{\mathbf{NFA}}$ 这两个语言都在PSPACE中。

§8.2 PSPACE类

考察PSPACE与P和NP的关系。

- 显而易见，**$P \subseteq PSPACE$**，因为运行快的机器不可能消耗大量的空间。更精确地说，当**$t(n) \geqslant n$**时，由于在每个计算步上最多能访问一个新单元，因此，任何在时间**$t(n)$**内运行的机器最多能消耗**$t(n)$**的空间。
- 类似地，**$NP \subseteq NPSPACE$**，所以**$NP \subseteq PSPACE$**。

§8.2 PSPACE类

反过来，可根据图灵机的空间复杂性也可以界定它的时间复杂性。

- 对于$f(n) \geq n$，通过简单推广引理5.7的证明可知，一个消耗$f(n)$空间的TM至多有$f(n)2^{O(f(n))}$个不同的格局，图灵机的停机计算不能出现重复格局。因此消耗空间$f(n)$的图灵机必定在时间$f(n)2^{O(f(n))}$内运行，得出：

$$\mathbf{PSPACE} \subseteq \mathbf{EXPTIME} = \cup k\ \mathbf{TIME}(2^{n^k})$$

- 用下面的一系列包含式总结迄今为止所定义的复杂性类之间的关系：

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}=\mathbf{NPSPACE} \subseteq \mathbf{EXPTIME}$$

是否有某一个为等式？？？

第9章证明$\mathbf{P} \neq \mathbf{EXPTIME}$

§8.3 PSPACE完全性

7.4节介绍了NP完全语言类，它代表NP中最难的语言类。证明一个语言是NP完全的是说明它不属于P的强有力的证据。若它属于P，则P将和NP相等。本节为PSPACE类引入一个类似概念PSPACE完全性。

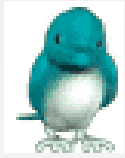
定义 8.7

语言B是**PSPACE完全的**。若它满足下面两个条件：

1) B属于PSPACE。

2) PSPACE中的每一个语言A多项式时间可规约到B 。

若B只满足条件2，则称它为**PSPACE难的**。



§8.3 PSPACE完全性

- ✓ B是PSPACE完全的
- ✓ B是PSPACE难的

- **定义中利用了多项式时间可归约性。为什么不定义多项式空间可规约？**

考察定义完全问题的动机：

1. 完全问题是重要的：因为是复杂类中最困难问题的样例。
2. 完全问题是最难的：因为该类中的其他问题很容易规约到它。如果找到一种简便的方法求解完全问题，就能很容易求解该类中的其他所有问题。
3. 为了使这种逻辑能够成立，相对于该类中典型问题的复杂性，规约过程就必须是容易的。如果规约过程本身就很难算，那么针对完全问题的容易解法，就不一定能推导出其他规约到它问题的容易解法。
4. **因此，规约模型必须比用来定义类本身的模型更加受限。**

§8.3.1 TQBF问题

- **PSPACE**第一个例子是可满足性问题的推广。回忆一下，布尔公式是一个包含布尔变量、常数**0**、**1**以及布尔运算符与、或、非的表达式。现在定义更一般形式的布尔表达式。
- 量词 ∀（对所有）和 ∃（存在）在数学命题中经常出现。语句 $\forall x\phi$ 的含义是，对于变量**x**的每个值，语句 ϕ 都是真。类似，语句 $\exists x\phi$ 的含义是，对于变量**x**的某个值，语句 ϕ 是真。 分别称为全称量词，存在量词，紧跟在量词后面的**x**受到该量词的约束。
- 解释包含量词的语句的含义时，必须考虑所取值的域。
- 量词的次序很重要。
- 前束范式
- 带量词的布尔公式称为量词布尔公式。
- 例子，见**191**页。

§8.3.1 TQBF问题

- TQBF问题就是要判定一个全量词化的布尔公式是真是假。定义语言

  TQBF={< ϕ >| ϕ是真的全量词化的布尔公式}

定理 8.8 TQBF是**PSPACE**完全的。

- 证明思路：为了证明TQBF属于PSPACE，给出一个简单算法。首先，给变量赋值，然后递归地计算公式在这些值下的真值。
- 为了证明每个语言A在多项式时间内可规约到TQBF，从判定A的多项式空间界限图灵机开始。然后给出多项式时间规约，把一个字符串映射为一个量词化的布尔公式，模拟机器对这个输入的计算。公式为真当且仅当机器接受。

§8.3.1 TQBF问题

定理 8.8 TQBF是PSPACE完全的。

证明：首先，给出一个判定TQBF的多项式空间算法：

T＝“对输入<ϕ>， ϕ是一个全量词化的布尔公式：

1）若ϕ不含量词，则它是一个只有常数的表达式。计算ϕ的值，若为真，则接受；否则，拒绝。

2）若ϕ等于$\exists x\psi$，在ψ上递归地调用T，首先用0替换X，然后用1替换X。只要有一个结果是接受，则接受；否则，拒绝。

3）若ϕ等于$\forall x\psi$，在ψ上递归地调用T，首先用0替换X，然后用1替换X。只要有一个结果是接受，则接受；否则，拒绝。”

算法T显然判定TQBF 。为了分析它的空间复杂性，我们发现它递归的深度最多等于变量的个数。在每一层只需存储一个变量的值，所以全部空间消耗是O(m)，其中m是ϕ中出现的变量的个数。因此T在线性空间内运行。

下面，证明TQBF是PSPACE难的。设A是一个由TM M在n^k空间内判定的语言，k是某个常数。下面给出一个从A到TQBF的*多项式时间归约*。

该归约把字符串w*映射*为一个量词化的布尔公式ϕ， ϕ为真当且仅当M接受w。为了说明怎样构造ϕ，需解决一个更一般的问题。利用两个代表格局的变量集合c_1和c_2及一个数t>0，构造一个公式$\phi_{c_1, c_2, t}$。如果把c_1和c_2赋为实际的格局，则该公式为真当且仅当M能够在最多t步内从c_1到达c_2。然后，可以令ϕ是公式$\phi_{c_{start}, c_{accept}, h}$，其中$h= 2^{df(n)}$，d是一个选取的常数，使得M在长为n的输入上可能的格局数不超过$2^{df(n)}$。

§8.3.1 TQBF问题

例如库克-列文定理的证明，该公式对带单元的内容编码。对应于单元的可能设置，每个单元有几个相关的变量，每个带符号和状态有一个变量。每个格局有n^k个单元，所以用$O(n^k)$个变量编码。

若t=1，则容易构造$\phi_{c1, c2, t}$。设计公式，使之表达(1)要么c1等于c2，(2)要么c1能在M的一步内变到c2 。为了表达相等性，使用一个布尔表达式来表示：代表c1的每一个变量与代表c2的相应变量包含同样的布尔值。为表达第二种可能性，采用库克-列文定理的证明中给出的技巧，构造布尔表达式表示，代表c1的每个三元组的值能正确产生相应的c2的三元组的值，从而就能够表达c1在M的一步内产生c2。

若t>1，递归的构造$\phi_{c1, c2, t}$。作为预演，先尝试一种不太好的想法，然后再修正它。令：

$$\phi_{c1, c2, t} = \exists m1[\phi_{c1, m1, t/2} \wedge \phi_{m1, c2, t/2}]$$

§8.3.1 TQBF问题

符号m_1表示M的一个格局。∃ m_1是∃ X_1，…， X_f的缩写，其中$l= O(n^k)$， X_1，…， X_f是对m_1编码的变量。所以$\phi_{c_1, c_2, t}$的这个构造是说：如果存在某个中间格局m_1，使得M在至多t/2步内从c_1变到m_1，并且在至多t/2步内从tm_1变到c_2，那么M就能在至多t步内从c_1变到c_2。然后再递归地构造$\phi_{c_1, m_1, t/2}$和$\phi_{m_1, c_2, t/2}$这两个公式。

公式$\phi_{c_1, c_2, t}$具有正确值，就是说，只要M能在t步内从c_1变到c_2，它就是TRUE。然而，它太长了。构造 过程中涉及的递归的每一层都把t的值减小一半，却把公式的长度增加了大约一倍，

最后导致公式的长度大约是t，开始时$t= 2^{df(n)}$，所以这种方法结出的公式是指数长的。

为了缩短公式的长度，除了使用∃量词以外，再利用∀量词。

令 $\phi_{c_1, c_2, t} = \exists m_1 \forall (c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\}[\phi_{c_3, c_4, t/2}]$

§8.3.1 TQBF问题

新引入的变量代表格局c_3和c_4，它允许把两个递归的子公式“折叠”为一个子公式，而保持原来的意思。通过写成

$$\forall(c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\},$$

就指明了代表格局c_3和c_4的变量可以分别取c_1和m_1的变量的值，或者m_1和c_2的变量的值，结果公式$\phi_{c_3, c_4, t/2}$在两种情况下都为真。

可以把结构$\forall(c_3, c_4) \in \{(c_1, m_1), (m_1, c_2)\}$替换为等价的结构$\forall[(x=y \vee x=z) \rightarrow \ldots]$，从而得到语法正确的量化布尔公式。

为了计算公式个$\phi_{c_{start}, c_{accept}, h}$的长度，其中$h= 2^{df(n)}$，注意到递归增加的那部分公式的长度与格局的长度呈线性关系，所以长度是$O(f(n))$。递归的层数是$\log 2^{df(n)}$或$O(f(n))$。所以所得到的公式的长度是$O(f^2(n))$。

§8.3.2 博弈的必胜策略

博奕和量词紧密相关。一个量化的语句有一个对应的博弈；反之，一个博奕也经常有一个对应的量化语句。

这种对应关系在以下这几个方面是有用的。

1. *首先．把一个包含许多量词的数学语句用对应的博弈表达出来，可以洞悉该语句的含义。*
2. *其次，把一个博弈用量化语句表达出来有助于理解该博弈的复杂性。*

✓ 为了阐明博弈与量词之间的对应关系，本节描述一种称为公式博弈的人工博弈。

设$\phi = \exists X_1 \forall X_2 \exists X_3 \ldots Q X_k[\psi]$是一个前束范式的量词化布尔公式。这里**Q代表量词∀或者∃**。

- 让ϕ与下面的博弈相关联。两名选手，称为选手**A**和选手**E**，轮流为变量**X1**，…，**Xk**选值。选手**A**为那些∀量词约束的变量选值，选手**E**为那些∃量词约束的变量选值。进行的顺序与公式开头量词出现的顺序相同。在游戏结束时，利用选手给变量挑选的值宣布结果。如果ψ (即删去量词后的那部分公式)此时是**TRUE**，则选手**E**赢；如果ψ此时是**FALSE**，则选手**A**赢。

§8.3.2 博弈的必胜策略

例8.9设ϕ_1是公式

$$\exists x_1 \forall x_2 \exists x_3[(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})]$$

在ϕ_1的公式博弈中，选手E挑选x_1的值，选手A挑选x_2的值，最后选手E挑选x_3的值。

举一个该博奕游戏过程的例子。照例，用1表示布尔值TRUE，用0表示FALSE。设选手E挑选$x_1=1$，然后选于A挑选$x_2=0$，最后选手E挑选$x_3=1$，当x_1、x_2和x_3取这些值时，子公式$(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})$是1，所以选手E赢。

实际上，选手E总可以赢这场博弈。

只要它挑选 X1 =1，然后不论选手A给X2选什么值，E都给X3选与X2相反的值。我们称选手E有这场博弈的必胜策略。如果双方都下出最佳步骤时，某个选手能赢，则该选手有该*博弈的必胜策略*。

§8.3.2 博弈的必胜策略

现在稍微修改一下公式，得到一个博弈，使得选手A有必胜策略。设ϕ_2是公式

$$\exists X_1 \forall X_2 \exists x_3[(X_1 \vee X_2) \wedge (X_2 \vee X_3) \wedge (X_2 \vee \overline{X_3})]$$

现在选手A有必胜策略，因为不论选手E为x_1选什么值，选手A可以选$x_2=0$，从而使量词后面出现的那部分公式为假，而不论选手E在最后一步选什么值。

下面考虑判定在与某个具体公式相关联的公式博弈中哪一方有必胜策略的问题。令

FARMULA-GAME={$<\phi>$|在与ϕ相关联的公式博弈中选手E有必胜策略}

§8.3.2 博弈的必胜策略

定理 8.10

FARMULA-GAME是PSPACE完全的。

证明思路：该问题是PSPACE完全的，理由很简单，即它与TQBF是一样的。

§8.3.2 博弈的必胜策略

证明：

- ✓ 公式$\phi = \exists X_1 \forall X_2 \exists X_3 \ldots [\psi]$是TRUE的条件是：*存在$X_1$的某种赋值，使得对于$X_2$的任意赋值，存在$X_3$的某种赋值，使得…等*等，其中$\psi$ 在这些变量的赋值下是TRUE。
- ✓ 类似地，选手E在与ϕ关联的博弈中有必胜策略的条件是：选手E可以给X_1赋某个值，使得对于X_2的任意赋值，选手E可以给X_3赋一个值，使得…等等， ψ在变量的这些赋值下为TBUE 。

当公式不在存在量词与全称量词之间交替时，同样的推理也能成立。如果ϕ的形式为 $\forall X_1$，X_2 ，$X_3 \exists X_4$ ，$X_5 \forall X_6[\psi]$，选手A在公式博弈中走头三步，给X_1，X_2 和X_3赋值；然后选手E走两次，给X_4 和X_5赋值；最后选手A给X_6赋一个值。 因此， $\phi \in$ TRUE恰好当$\phi \in$ FARMULA-GAME时成立，由定理8.8，本定理成立。

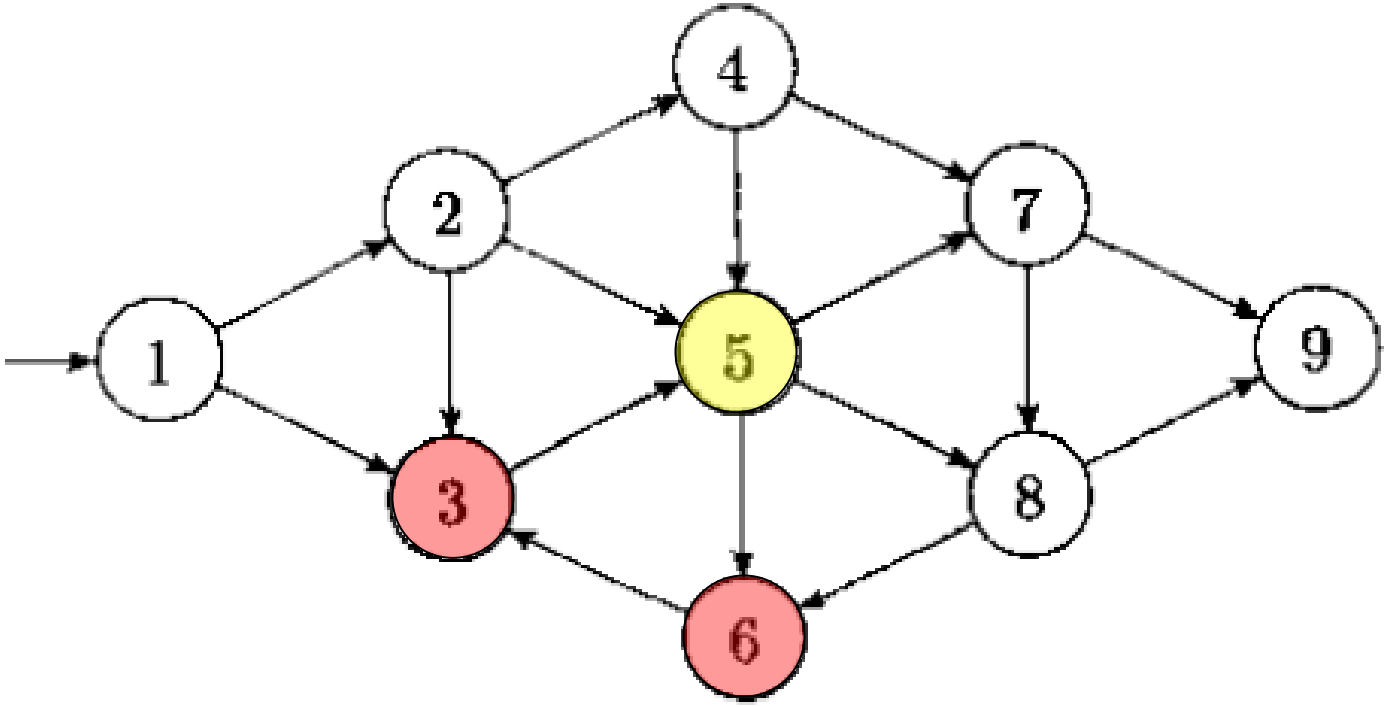
§8.3.3 广义地理学

地理学是一种儿童游戏，选手们轮流给世界各地的城市命名。每一座选中的城市的首字母必须与前一座城市的尾字母相同，不允许重复。游戏从某个指定的起始城市开始，以某一方因无法延续而认输为止。

- 例如，如果游戏从**Peoria**开始，那么下面可以跟**Amherst**，然后可以是**Tucson**．然后是**Nashua**，等等。直到一方被难倒，输掉比赛为止。

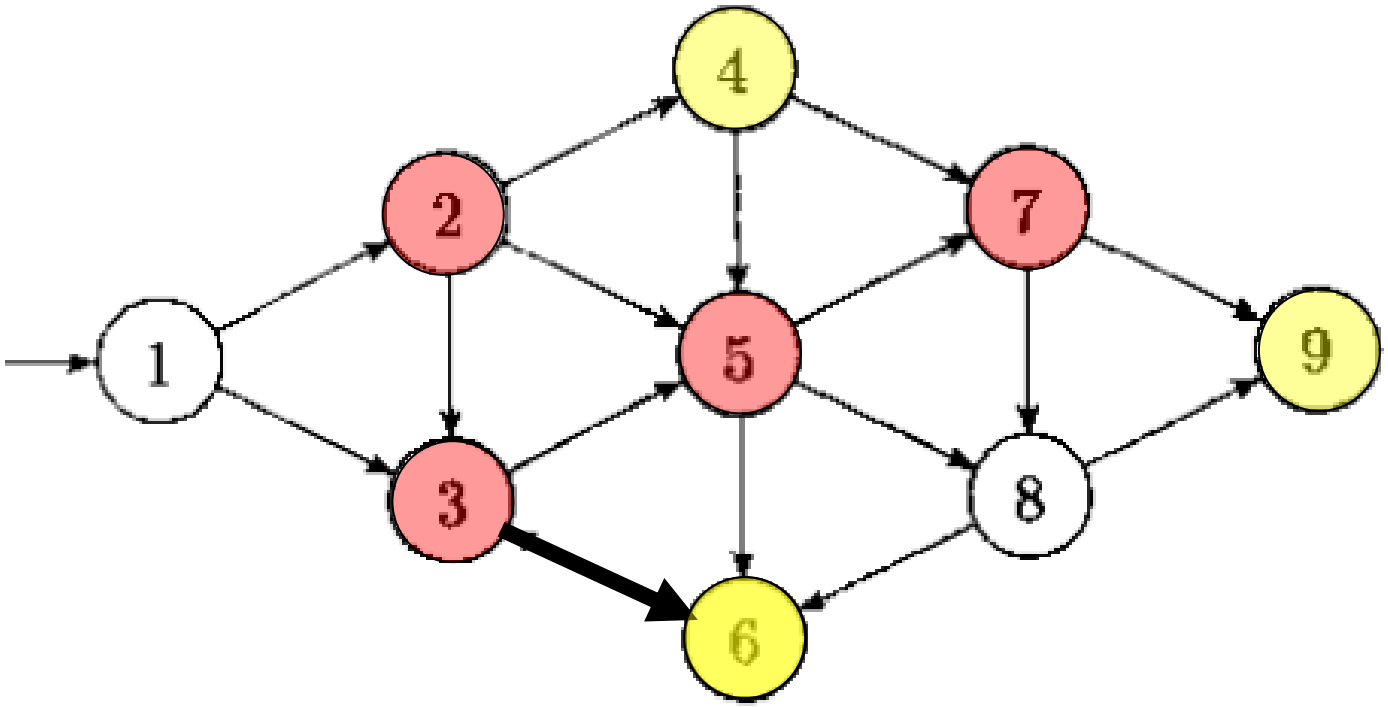
§8.3.3 广义地理学

- 广义地理学中，用任意的带有指定起始节点的有向图代替了与实际城市相关联的图。如图8-3所示，选手I先走，选手II随后走，I有必胜策略。



§8.3.3 广义地理学

- 如果颠倒节点3与6间边的方向，则选手II有必胜策略。



§8.3.3 广义地理学

- 判定在广义地理学游戏中哪一方有必胜策略的问题是PSPACE全的。令

CG={<G,b>|在图G上以节点b起始的广义地理学游戏中，选手I有必胜策略}

定理 8.11

GG是PSPACE完全的。

- **证明思路：**
  - 利用定理8.8中判定TQBF时所用的算法相似的递归算法，就能判定哪方有必胜策略。该算法在多项式时间内运行，所以GG∈PSPACE。
  - 证明GG是PSPACE难的。给出一个从FORMULA-GAME到GG的多项式时间规约。该规约将一个公式博弈转化为一个广义地理学图，使得图上的游戏过程模拟公式博弈的游戏过程。

§8.3.3 广义地理学

定理 8.11

GG是PSPACE完全的。

- 证明：下面的的算法判定在广义地理学实例中，选手I是否有必胜策略。换句话说，它判定GG。现证明它在多项式空间内运行：

  M=“对输入<G，b>，G是有向图，b是G的结点：

  1）若b出度为0，则拒绝，因为选手I立即输。

  2）删去结点b以及与它关联的所有箭头，得到一个新图G1。

  3）对于b原先指向的每个节点b1， b2， …， bk，在<G1 ， bi>上递归地调用M。

  4）若所有调用都接受，则选手II在原先博弈中有必胜策略，所以拒绝。否则，选手II没有必胜策略，而选手I有必胜策略，因此接受。”

§8.3.3 广义地理学

- 本算法仅需要用来存储递归栈的空间。递归的每一层给栈中添加一个节点，最多可能有m层，m是G的结点数。因此算法在线性空间内运行。
- 下一步，证明GG的PSPACE难解性。证明*PORMULA-GAME*多项式时间可规约到GG。规约把公式

$$\phi = \exists X1 \ \forall \ X2 \ \exists X3 \ldots [\psi]$$

映射为广义地理学的一个实例**<G,b>**。

假设ϕ以∃开头，以∃结尾，并且∃与∀严格交替出现，

§8.4 L类和NL类

到目前为止，只考虑了时间和空间复杂性界限至少是线性的情况，即界限 $f(n)$ 至少是n。现考察更小的亚线性（sublinear）空间界限。

在时间复杂性中，亚线性界限还不够读完输入，所以这里不考虑它们。在亚线性空间复杂性中机器可以读完整个输入，但是它没有足够的空间存贮输入。为了使对这种情况的考虑有意义，必须修改计算模型。

下面引入一种有两条带的图灵机：一条只读输入带和一条读写工作带。在只读带上输入头能读取符号，但不能改变它们。这个头必须停留在包含输入的那部分带子上。可以给机器提供一个方法，使它能够检测读写头处在输入的左端和右端的时刻。工作带可以用通常的方式读写。只有工作带上被扫描的单元才构成这种形式的图灵机的空间复杂性。

§8.4 L类和NL类

- 一条只读输入带：相当于CD_ROM
- 一条读写工作带：可修改。
- 只有工作带上被扫描的单元才构成这种图灵机的空间复杂性。

§8.4 L类和NL类

定义 8.12

L是确定型图灵机在对数空间内可判定的语言类。换言之，

$$L = SPACE(\log n)$$

NL是非确定型图灵机在对数空间内可判定的语言类。换言之，

$$NL = NSPACE(\log n)$$

注：指向输入的指针可以在对数空间内表示，所以考虑对数空间算法计算能力的一种方式是考虑固定输入数目的输入指针的计算能力。

§8.4 L类和NL类

- 线性空间复杂性界限：f(n)=n
- 亚线性空间复杂性界限：f(n)<n
  - 在时间复杂性中不考虑亚线性，因为亚线性连输入都不能读完
  - 亚线性空间复杂性中能读完全部输入，但没有足够的空间存储全部输入。解决办法是修改计算模型——包含只读带的两带图灵机。

§8.4 L类和NL类

- 例8.13：A={0k1k|k>=0}是L的成员。
- 在算法8.1中描述了一个判定A的图灵机，左右来回扫描输入，删除匹配的0和1。它使用线性空间记录哪些位置被删除了，但可以修改为只使用对数空间。
- 在对数空间空间内由于输入带是只读的，故不能删除已经匹配的0和1。用工作带存放两个计数器记录0和1 的个数，这两个计数器用二进制表示只消耗对数空间。即算法可在O(log n)空间内运行。

§8.4 L类和NL类

- 例8.14：8.2节定义的语言PATH={<G,s,t>|G是包含从s到t的有向路径的有向图}。定理8.12已经证明了PATH属于P，但原来的算法消耗线性空间。现在能否找到只消耗对数空间的算法？
- 目前还没有找到确定性对数空间图灵机判定PATH，但有非确定性对数空间图灵机判定PATH，即PATH属于NL

§8.4 L类和NL类

非确定性对数空间图灵机判定PATH

- 从节点s开始运算，非确定的猜测从s到t的路经的每一步。机器在工作带上只记录每一步当前节点的位置，而不是整条路径（如果是整条路径会超出对数空间）。
- 机器从当前节点指向的节点中非确定的选择下一个节点，直到达到节点t接受，或指向m步后拒绝，其中m是图的节点数。
- 所以PATH属于NL

§8.4 L类和NL类

对数空间界限的运行时间界限

- f(n)空间界限的图灵机也在$2^{O(f(n))}$时间内运行。这个断言在对数空间仍然成立。

定义 8.15

若***M***是一个有单独的只读输入带的图灵机，**w**是输入，则***M***在 **w**上的格局包含状态、工作带和两个读写头位置。输入**w**不作为***M***在**w**上的格局的一部分。

- 现在要计算M在w上的时间上限

§8.4 L类和NL类

对数空间界限的运行时间界限

M空间为f(n)，|w|=n，作为M在w上的格局数为$n2^{O(f(n))}$.

① 设M有c个状态

② 带符号g个，则可能出现在工作带上的字符串数为$g^{f(n)}$

③ 输入带上输入头有n种位置。

④ 工作带头有f(n)中可能位置

⑤ 由①②③④得到M的总格局数为$cnf(n)g^{f(n)}$，或表示为$n2^{O(f(n))}$即M在w上的运行上界。

⑥ 当$f(n) \geqslant \log n$时，$n2^{O(f(n))}$等于$2^{O(f(n))}$

§8.5 NL完全性

- 类似于P=NP是否成立，也有N=NL是否成立的问题。
- 前面知道PATH∈NL，但PATH∈L？我们相信PATH不属于L，但不知道如何证明。还没有证明NL中哪一个问题不属于L。
- 为了解决N和NL的问题，引入了NL完全NL_complete, NL完全语言是NL中最困难的语言。如果L≠NL，则所有NL完全语言就不属于L。

§8.5 NL完全性

- NL完全语言定义为属于NL，并且NL中的其他语言都可归约到它。
- 这里归约不用多项式时间归约，因为所有NL中的问题都是多项式时间可解的。
- 多项式时间归约性太强，区分不了NL中的问题。
- 引入对数空间可归约性。

§8.5 NL完全性

定义 8.16

对数空间转换器(log space transducer)是有一条只读输入带、一条只写输出带和一条读写工作带的图灵机。工作带可以包含O(logn)个符号。对数空间转换器M计算一个函数$f: \Sigma^* \to \Sigma^*$，其中f(w)是把w放在M的输入带上启动M运行，到M停机时输出带上存放的字符串。称f为对数空间可计算函数。如果语言A通过对数空间可计算函数f映射可归约到语言B，则称A对数空间可归约到B，记为$A \leqslant_L B$。

§8.5 NL完全性

定义 8.17

语言B是NL完全的，如果

1）$B \in NL$，并且

2）NL中的每个A对数空间可归约到B。

如果一个语言对数空间可归约到另一个已知属于L的语言，则这个语言也属于L，如下述定理所阐明的。

§8.5 NL完全性

定理 8.18

$A \leqslant_L B$ 且$B \in L$， 则$A \in L$。

- 按照定理8.25的证明思路：A的对数空间算法先通过对数空间可计算函数f将w映射为f(w)，然后用B的对数空间算法。但对数空间放不下f(w)，所以需要修改算法。

§8.5 NL完全性

- 证明：A的机器M_A不再算出整个f(w)，而是根据B的机器M_B的需要计算个别符号。在模拟过程中，M_A记录M_B的输入头在f(w)上的位置。每一次B移动时，M_A重新开始在W上计算f，除了M_B所要的f(w)上的特定位置的字符以外，其他的忽略。
- 这就造成了重复计算，在时间复杂性方面的性能就非常低。
- 但好处就是在任何时候只需要存储f(w)的一个字符。即以时间换空间。

推论 8.19

若有一个NL完全语言属于L，则L=NL。

§8.5 NL完全性

图中的搜索

定理 8.20

PATH是NL完全的。

证明思路：

- 例8.14已经证明PATH是NL，现在只需证明PATH是NL难的。即必须证明NL中的每个语言A对数空间可归约到PATH
- 从A到PATH的对数空间归约背后的思想是构造一个图，用来表示判定A的非确定对数空间图灵机的计算过程。
- 归约把W映射为有一个图，图中的节点对应于NTM在w上的格局。一个节点能指向另一个节点的条件是第一个节点对应的格局能在NTM下一步产生第二个节点对应的格局。因此只要从对应初始格局的节点到对应接受格局的节点之间存在一条路经，则机器接受w。

§8.5 NL完全性

证明：

- 设NTM M在$O(\log n)$空间内判定A。给定输入w，在对数空间内构造<G,s,t>,其中有向图包含从s到t的路径当且仅当M接受w.如何构造？G的节点是M在w上的格局。对于M在w上的格局c1和c2，如果c2是M从c1出发的下一个可能的格局，则(c1,c2)是G的一条边。更精确地说，如果M的转移函数指出，c1的状态和输入带头和工作带头下的符号能产生下一个状态和带头动作，使c1变成c2，则(c1,c2)是G的一条边。节点s是M在w上的初始格局，机器被修改为只有唯一的接受格局，把该格局指定为节点s。
- 该映射把A映射到PATH，因为M只要接受输入，就有一个计算分支接受，这对应s到t的一条路径。反之，如果G中存在从s到t的路径，M在输入w运行时，某个计算分支必然接受，从而M接受w。
- 证明该归约在对数空间内运算,使用一个对数空间转器，它在输入w上输出G的一个描述。

§8.5 NL完全性

推论 8.21

NL ⊆ P。

- 定理8.20说明NL中的任何语言对数空间可归约到PATH。回忆一下，消耗空间f(n)的图灵机在$n2^{O(f(n))}$内运行，所以在对数空间内运行的的归约器也在多项式时间内运行。因此NL中的任何语言多项式时间可归约到PATH。由定理8.12知道后者属于P。又知每一个多项式时间可归约到P中的语言也在P中。证毕。

§8.6 NL等于coNL

- 一般认为：NP<>coNP
- 这里：NL＝coNL

定理 8.22 **NL =coNL。**

证明思路：

- 为了证明coNL中的每个问题也在NL中，先证明 $\overline{PATH}$是NL，因为PATH是NL完全的。所给出的判定 $\overline{PATH}$ 的NL算法*M*，在图G不包含从s到t的路径时，必须有一个接受计算。
- 首先解决一个容易些的问题。令c是G中从s可达的结点数。假定c作为输入提供给*M*，先说明怎样利用c求解 $\overline{PATH}$ ，然后再说明怎样计算c。

§8.6 NL等于coNL

定理 8.22 **NL =coNL。**

给定G，s，t和c，机器*M*如下运算。

- M逐个地走遍G的所有m个结点，非确定地猜测每个节点是否从s可达。
- 一旦节点u被猜测为可达，*M*就通过猜测一条从s到t的路径来验证这一点。如果一个计算分支没能在m步内验证这一猜测(m是G的节点数)，就拒绝。
- 另外，如果一个分支猜测t可达，就拒绝。机器*M*对那些已经被验证为可达的结点计数。当一个分支走遍G的所有结点后．它查验从s可达的节点数是否等于c，即实际可达的节点数，如果不等就拒绝。否则该分支接受。

§8.6 NL等于coNL

- 换句话说，如果*M*非确定地恰好挑选从s可达的c个节点(不包括t)，并且通过猜测路径证实了每一个都从s可达，则*M*就知道剩余的节点(包括t)，都不可达，所以它就可以接受。
- 接下来说明怎样计算c，即从s可达的结点数。描述一个非确定的对数空间过程，它至少有一个计算分支具有正确的c值，而所有其他分支都拒绝。
- 对于从0到m的每个值i，定义A_i为*G*中与s的距离不超过i的节点的集合(即从s出发有一条长度不超过i的路径)。于是$A_0=\{s\}$，每个$A_i \subseteq A_{i+1}$，A_m包含从s可达的所有结点。令c_i是A_i中的结点数。下面描述一个过程，从c_i中计算出c_{i+1}。反复应用这一过程就获得所需的值$c=c_m$。
- 从c_i中计算出c_{i+1}所用的思路与本证明思路前面给出的思路相似。算法走遍G的所有结点，决定每一个节点是否是A_{i+1}的成员，然后数成员的个数。

§8.6 NL等于coNL

- 为了判定节点v是否在A_{i+1}中，用一个内层循环走遍G的所有节点，猜测每个节点是否在A_{i+1}中。每一次成功的猜测是由猜测一条从s出发、长度全多为i的路径来证实。对于每个证实了在A_i中的节点u，算法检查(v，u)是否是G的一条边。如果是一条边，则v在A_{i+1}中。另外，证实了在A_i中的节点的数目也被计算出来。在内层循环结束时，如果证实属于A_i的节点总数不等c_i，则A_i的全部节点还没有被找完，所以该计算分支拒绝。如果总数的确等于c_i，并且v还没有证实属于A_{i+1}，则可以下结论说它不在A_{i+1}中。然后走到下一个v，开始外层循环。

§8.6 NL等于coNL

证明：判定$\overline{PATH}$的算法如下：这里令m为G的结点数。

M="对输入<G，s，t>：

1)令c_0=1,d=0。

2)对于i=0到m-1：

3) 令c_{i+1}=1。

4) 对G中每个节点v(v不等于s)：

5) 令d=0。

6) 对G中每个节点u：

7) 非确定地执行或者跳过下列步骤：

8) 非确定地沿着从s出发、长度至多为i的路径进行，如果没有碰到结点u，就拒绝。

9) d+1。

10) 如果(u，v)是G的一条边，则c_{i+1}加1。v变为下一个结点，转向步骤5。

§8.6 NL等于coNL

11) 若d不等于ci，则拒绝。
12) 对G中每个节点u：
14) 非确定地执行或者跳过下列步骤：
15) 非确定地沿着从s出发、长度至多为m的路径进行，如果没有碰到结点u，就拒绝。
16) 若u=t，则拒绝。
17) d+1。
18)若d不等于c_m，则拒绝，否则，接受。”

Summaries

$$L \subseteq NL = coNL \subseteq P \subseteq PSPACE$$

作业

- **8.1、8.3、8.12、8.21、8.25、8.28**