第九章 NP-完全问题

目录

9.1 判定问题

- 什么是好算法
- 多项式时间算法
- 问题复杂性与算法复杂性
- 现实世界的问题
- 优化问题可以转化为判定问题

什么是好算法

- 运行时间是评价算法好坏的重要标准
  - 解决问题规模n需要的计算时间
  - 单位时间能够解决到的问题规模n
- 下面比较三个算法的问题规模和计算时间之间的关系
  - 快速排序算法 $O(n\log n)$： 排序问题
  - Dijkstra算法 $O(n^2)$： 图的单源最短路径问题
  - 最大团问题的回溯法 $O(n2^n)$： 求最大完全子图问题

解决问题规模n需要的计算时间

- 假设用一台每秒10^9次的超大型计算机来计算

算法名称	时间复杂度	问题规模	运算量	计算时间(/10^9)
快速排序算法	O(nlogn)	10^5个数据	$10^5 \times \log_2 10^5 \approx 1.7\times10^6$	1.7×10^{-3}秒
Dijkstra算法	$O(n^2)$	10^4个顶点	$(10^4)^2 = 10^8$	0.1秒
最大团问题的回溯法	$O(n2^n)$	10^2个顶点	$100 \times 2^{100} \approx 1.8 \times 10^{32}$	1.8×10^{21}秒 ≈ 5.7×10^{15}年

单位时间能够解决到的问题规模n

- 增大计算机的运算能力， 每秒达到6×10^{10}次

算法名称	时间复杂度	问题规模
快速排序算法	O(nlogn)	2×10^9 (即 20亿)个数据排序
Dijkstra算法	$O(n^2)$	2.4×10^5 个顶点的图
最大团问题的回溯法	$O(n2^n)$	一天时间解决41个顶点的图

- 快速排序算法和Dijkstra算法可以快速解决问题，是好算法
- 最大团问题的回溯法只能用于较小的图，对于稍大的图，如100个顶点，根本不可行！

多项式时间算法

- 多项式时间算法
  - $O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3)$
- 非多项式时间算法(指数)
  - $O(2^n) < O(n!) < O(n^n)$

多项式时间算法是好算法

算法复杂性与问题复杂性

- 算法的复杂性
  - 是指解决问题的一个具体的算法的执行时间，是算法的性质。
- 问题的复杂性
  - 是指这个问题本身的复杂程度，是问题的性质。
- 例如排序问题。
  - 排序问题的复杂性是O(nlogn)
  - 冒泡排序法是$O(n^2)$，快速排序平均情况下是O(nlogn)
  - 排序问题的复杂性是指在所有的解决该问题的算法中最好算法的复杂性。

现实世界的问题

- 多项式级的问题：
  - 这类问题已经设计出多项式级算法
  - 如排序、最小生成树、单源最短路径等
- 指数级的问题：
  - 这类问题已经设计出指数级算法，并且已证明不存在多项式级算法
  - 如带幂运算的正则表达式的全体性
- 不满足计算性的问题：
  - 根本不存在求解算法的问题
  - 如希尔伯特第十问题
- 尚不确定多项式级的问题：
  - 这类问题已经设计出指数级算法、但不能证明不存在多项式级算法
  - 如图着色问题、货郎担问题、0/1背包问题等

很多优化问题属于这种情况

优化问题可以转化为判定问题

- 很多优化问题可以转化为判定问题
- 判定问题是指只需要回答“是”和“否”的问题

图着色问题	图着色判定问题
已知图G=<V,E>，求对G的n个顶点进行着色的一种方法（相邻顶点颜色不同），问最少需要多少种颜色	已知图G=<V,E>，问图G是否可用k种颜色进行着色？

最大团问题	最大团判定问题
图G=(V,E)的一个完全子图叫做G的一个团(clique)。团的大小用所含的结点数表示。 求G内最大团和它的大小。	已知图G=<V,E>，问图G是否存在一个大小为k的团？

0/1背包问题	0/1背包判定问题
已知n个物品的效益值pi和重量wi，每个物品只能整取。在背包容量M的限定下如何选择，使选中物品的效益值之和最大	在M限定下，对于给定效益值X，是否存在一组选择策略，使选中物品的效益值之和大于等于X？

思考：如何根据优化问题求解判定问题？反之？

以最大团问题为例相互转化

- 判定问题→优化问题
  - 设DCLIQUE(G,k)为判定算法，n是图G中点的个数，检验k=n, n–1, n–2,…直到DCLIQUE(G,k)输出为1，则找到G的最大团大小
  - 设DCLIQUE的时间复杂度为f(n)，则最大团问题可在n*f(n)时间内求出

- 优化问题→判定问题
  - 求解图G的最大团问题，找到G的最大团大小j，如果j<k，则DCLIQUE(G,k)返回否，否则返回是
  - 设最大团问题的时间复杂度为g(n)，则DCLIQUE也可在g(n)时间内求出

结论：最大团问题与其判定问题可以在多项式时间内相互转换。

优化问题转化为判定问题的好处

- 最大团问题可以在多项式时间内求解，当且仅当其判定问题可以在多项式时间内求解
- 许多尚不确定多项式时间的优化问题都可以改写成判定问题，并具有如下性质：
  - 该判定问题可以在多项式时间内求解，当且仅当与它相应的优化问题可以在多项式时间内求解
  - 如果判定问题不能在多项式时间内求解，那么与它相应的优化问题也不能在多项式时间内求解

将优化问题转化为判定问题的好处是消除了不同优化问题的输出差异性，通过转化为判定问题，输出统一成“是”和“否”

8.2 不确定的判定问题

- 不确定算法
- 不确定机
- 检索问题的不确定算法
- 不确定算法设计
- 排序问题的不确定算法
- 不确定的判定算法
- 最大团判定问题的不确定算法
- 问题规模的二进制表示

不确定算法

回顾确定算法:运算满足确定性，即运算结果是唯一的。

- 不确定算法
  - 从理论角度看，取消对运算“确定性”这一限制。即允许运算结果受限于某个特定的集合。包含这种不确定运算的算法称为不确定的算法
- 为描述不确定算法，SPARKS中引进了一个新函数和两个新语句，时间复杂度恒为O(1)：
  - choice(S)：按题意选取集合S中的一个元素
  - failure：发出不成功完成的信号
  - success：发出成功完成的信号

不确定机

- 能按上述方式执行不确定算法的机器称为不确定机

上帝视角

- 不确定机是一个“有魔力”的机器，总在O(1)时间内执行完函数choice(S)
  - 如果问题有解，它直接猜中，从S中返回一个需要的元素
  - 如果问题无解，它从S中随机返回一个元素

- 在现实技术条件下，不确定机实际上是不存在的，是为了分析时间复杂度难题而虚构的理论模型

检索问题的不确定算法

- 问题描述：考察给定元素集A(1:n),n>=1中，元素X的检索问题。需确定下标j，使得A(j)=x，或者当x不在A中时有j=0。

```
j←choice(1:n)
if A(j)=x then success endif
j←0;  print('0'); failure
```

- 当且仅当不存在一个j，使得A(j)=x时输出 “0”
- 该不确定算法的复杂度为O(1)

不确定算法设计

- 不确定算法的设计步骤
  - 猜想阶段：也称为不确定阶段，基于不确定函数choice猜出一个解
  - 验证阶段：也称为确定阶段，用确定语句验证构造出的解是否是答案
- 不确定算法在多项式时间可验证
  - 当不确定算法在验证阶段的时间复杂度是多项式级的
- 不确定算法的时间复杂度
  - 若返回failure，不关注失败情况，不妨认为恒为O(1)
  - 若返回success，则为不确定阶段和确定阶段的时间复杂度之和

排序问题的不确定算法

- 问题描述：设A(1:n),1≤i≤n，是一个待排序的正整数集，要求将其按非降次序排序

```
procedure NSORT(A,n)//对n个正整数排序
    integer A(1:n),B(1:n),n,i,j
    B(1:n)←0 //辅助数组
    for i←1 to n do//构造
        j←choice(1:n), if B(j) ≠0 then failure endif；
       B(j)←A(i)
    repeat
    for i←1 to n-1 do//验证B的次序
        if B(i)>B(i+1) then failure endif
    repeat
    print(B)
    success
end NSORT
```

若B(j)已被占用，失败

若存在降序，失败

- 辅助数组B，初始化为0
- 逐个将A中的元素，放入B(j)，j为“猜测”出的正整数，
- 检查B中元素是否按非降次序排列。

O(n)+O(n)

不确定的判定算法

- 不确定的判定算法只需要回答yes或者no的问题，只产生0/1输出：
  - 0：当且仅当没有一种选择可导致success
  - 1：当且仅当至少存在一种选择导致success
- 为描述简洁，输出隐含于success和failure，此外无输出语句

最大团判定问题的不确定算法

```
procedure  DCK(G,n,k)//n表示图G中点的个数
  S←∅//求得的点集
  for i←1  to  k  do
      t ←choice(1:n)
      if  t∈S  then  failure  endif
      S ←S∪t
  repeat
  for 使得i∈S, j∈S的每一对(i,j),and i≠j do
      if (i,j)不是此图的边  then  failure  endif
repeat
success
end DCK
```

S是“猜测”的k个顶点的集合，初值为空集

若t是一个已生成过的顶点，失败

不确定阶段：$O(k)$
确定阶段：$O(k^2)$
算法：$O(k+k^2)=O(n^2)$

问题规模的二进制表示

已知算法时间复杂度基于问题规模或问题规模的某种度量

- 为了统一度量不同判定问题的算法时间复杂度，不同算法的输入参数均转换为二进制形式，算法时间复杂度基于二进制输入长度来考虑
- 例如最大团判定问题DCK(G,n,k)的输入
  - 设图G由其邻接矩阵表示，n是点个数，k是判定值
  - 二进制转换输出参数集，其长度和为m，$m=n^2+\lfloor\log_2 k\rfloor + \lfloor\log_2 n\rfloor +2$

邻接矩阵，用01矩阵表示

待判定的结点个数

结点个数

算法DCK的时间复杂度：$O(n^2)=O(m)$

9.3 NP问题与NP完全性

- P问题与NP问题
- 归约定义
- 多项式时间归约
- NP-完全问题与NP-难问题
- 可满足性问题
- 可满足性问题的不确定算法
- Cook定理
- 恰好覆盖问题
- 子集和问题

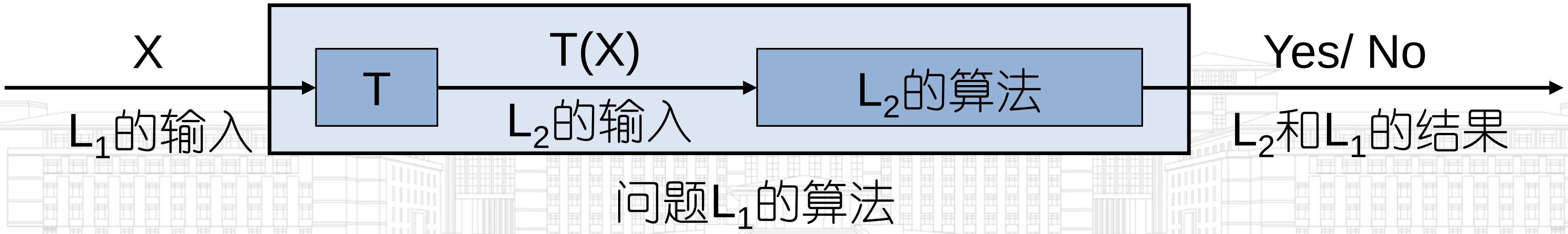
P问题与NP问题（NP:Non-deterministic polynomial）

- P问题
  - 所有可在多项式时间内用确定算法求解的判定问题的集合
- NP问题
  - 所有可在多项式时间内用不确定算法求解的判定问题的集合
  - 即能够用不确定算法在多项式时间里猜出一个解和验证一个解
- P问题、NP问题与非P问题
  - 可以认为P问题是NP问题的一种特例，即choice(S)中，S仅有唯一个元素，因此P⊆NP成立，但尚不确定是否有P=NP或者P≠NP
  - NP≠非P问题，它是非P问题中用不确定算法能在多项式时间内求解的那部分判定问题的集合，因此NP⊂非P问题

归约定义

不同NP问题的难度是不同的，用归约来表达它们的难度关系

- 定义（归约）：令L_1和L_2是两个问题，如果有一确定的多项式时间算法求解L_1，而这个算法使用了一个在多项式时间内求解L_2的确定算法，则称L_1归约为L_2（$L_1 \le L_2$或$L_1 \propto L_2$）
- 问题L_1归约为问题L_2的含义是：可以用问题L_2的解法解决问题L_1，或者说，问题L_1可以“变成”问题L_2
- 判定问题L_1到判定问题L_2的归约: 存在一个转换函数T，可以把问题L_1的输入x转换为问题L_2的输入T(x)，满足：问题L_1对于输入x得到正确结果当且仅当问题L_2对于输入T(x)得到正确结果。
- 归约具有传递性



一个归约的例子

- 归约的直观意义
  - L_1可归约为L_2，则L_2的时间复杂度不低于L_1的时间复杂度，即L_1不比L_2难
- 判定问题L_1：判定n个布尔值中是否至少有一个为true
  - L_1的输入：$X_1,X_2,\ldots X_n$, X_i=true or false$(1\le i\le n)$
- 判定问题L_2：判定n个整数中的最大元是否为正数
  - L_2的输入：$Y_1,Y_2,\ldots Y_n$, Y_i为整数$(1\le i\le n)$
- 转换函数T：
  - $T(X_1,X_2,\ldots X_n)=(Y_1,Y_2,\ldots Y_n)$，其中 $Y_i = \begin{cases} 1 & X_i\text{=true} \\ 0 & X_i\text{=false} \end{cases}$

多项式时间归约

- 多项式时间归约
  - T是问题L_1归约到L_2的转换函数，如果计算T的算法是多项式级的，则称问题L_1可以多项式时间归约到L_2，记为$L_1 \leq_p L_2$
  - T称为归约函数，其算法称为归约算法

$L_1 \propto_p L_2$

- 多项式时间归约具有传递性
  - 对于问题L_1、L_2、L_3, 如果$L_1 \leq_p L_2$，$L_2 \leq_p L_3$, 则$L_1 \leq_p L_3$
- 当L_1可以多项式时间归约到L_2时
  - 如果L_2多项式级可解决，L_1亦然
  - 如果L_2指数级可解决，L_1亦然

下文讨论的归约均是指多项式时间归约

NP完全问题（NPC问题）

- 基于归约的定义，可以科学地比较两个问题的难易，以此为基础，可以将一类常见而重要的难问题清晰地刻画出来，这就是NP完全问题。
- 简单地说，它就是最难的那些NP问题。

NP-完全问题与NP-难问题

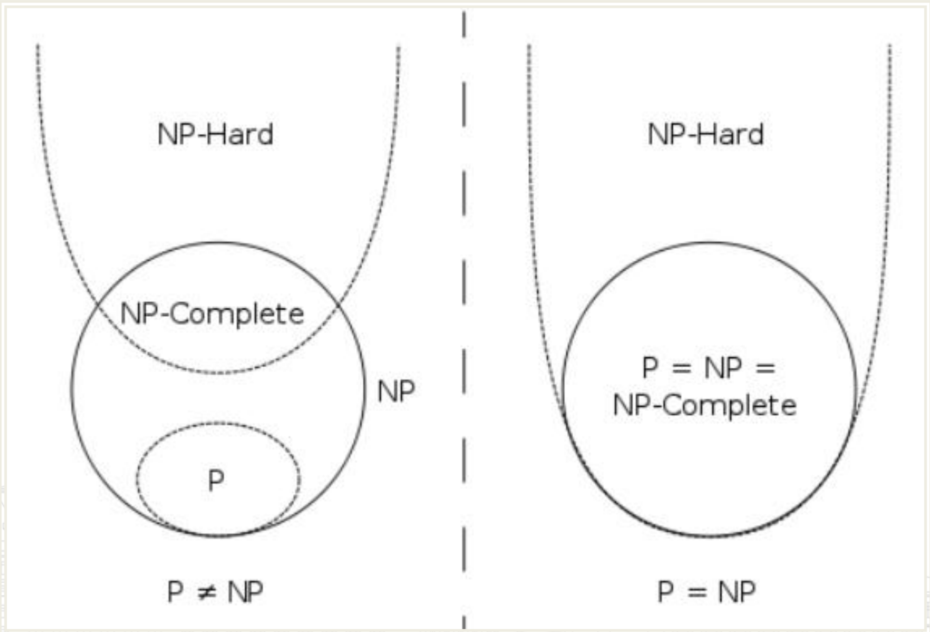
- 问题L是NP-完全的，或NPC的，当满足：
  - $L \in NP$
  - 对于每个$L' \in NP$，有$L' \leq_p L$
- 问题L是NP-难的，或NP-hard的，当满足：
  - 对于每个$L' \in NP$，有$L' \leq_p L$

没限定属于NP，
尚未找到多项式
级的不确定算法

- 比较NP-完全问题与NP-难问题
  - NP-hard比NPC范围广， NP-Hard同样难以找到多项式算法，有可能比所有NPC的时间复杂度更高从而更难以解决
  - NPC不是NP-hard，但它是NP中最难的问题

P问题、NP问题、NP完全问题

NP问题的定义引出未解问题：P是否等于NP. 基于NP完全问题发现，如果任意一个NP完全问题可以在多项式时间解决，即所有NP问题均可在多项式时间解决，即P=NP. 如果证明任意一个NP完全问题不存在多项式时间的解，则所有NP问题均不可能在多项式时间内解决。



- NPC的意义
  - 所有的NP问题都可以多项式时间归约到一个NPC问题，这意味着一旦一个NPC问题多项式可解，则所有NP问题都多项式可解
  - P=NP充要条件是存在NPC问题 $L \in$ P

- 证明 NPC的 “捷径”
  - 根据多项式时间归约的传递性， 对于问题$L \in$NP，如果存在NPC-问题 L' 使得 $L' \leq_p L$，则L是NPC的
- 证明问题L是 NPC的, 需要两个步骤:
  - 证明 $L \in$NP
  - 已知一个 NPC-问题L', 证明 $L' \leq_p L$

是否存在NPC-问题？如果存在，第一个找到的NPC-问题是什么？

可满足性(SAT)问题

SAT问题是第一个被发现的NPC问题，它约化出了其他NPC问题，如果证明出SAT问题是P问题，就能证明出P=NP。

- 问题描述：对于任意给定的一个合取范式F, F是否可满足?
- 合取范式F ：
  - 令$x_1,x_2,\ldots,x_n$表示布尔变量，$\neg x_i$表示x_i的非。一个文字σ即可以是一个变量也可以是它的非
  - 合取符号∧表示与关系；析取符号∨表示或关系
  - 公式形如$A_1 \wedge A_2 \wedge \ldots \wedge A_k$, A_j是形如∨σ的子式，称为合取范式(CNF)
- F可满足
  - 存在一组赋值给$x_1,x_2,\ldots,x_n$，使F值为真

问题实例

- 合取范式$F_1=(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_2$
  - 令 $(x_1, x_2, x_3)=(1,0,1)$, F_1成真赋值, F_1是可满足的.
- 合取范式$F_2=(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge x_2 \wedge \neg x_3$
  - F_2不存在真赋值，不是可满足的

可满足性问题的不确定算法

```
Procedure EVAL(F,n)
//判定命题F是否为可满足的。变量为Xi，1≤i≤n
Boolean x(1:n)
for i←1 to n do//选取一组真值指派
    xi←choice(true,false)
repeat
if F(x1,…,xn)=true
   then success
   else failure
endif
end EVAL
```

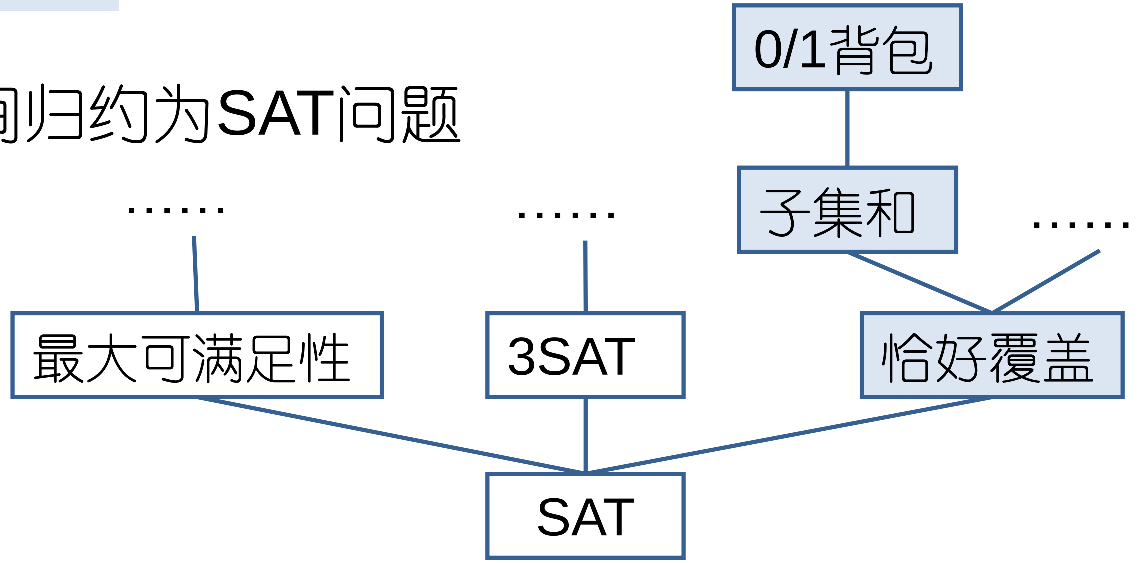
对n个变量赋予一组真值指派

$\boxed{O(n)}$

Cook定理

Cook定理：SAT是NP完全的

- 即任何NP问题都可以在多项式时间归约为SAT问题
- SAT问题是第一个NPC问题
- 证明略



现在NPC问题已超过3000个

证明问题$L\in NP$是NPC的关键在于，基于一个已知 NPC-问题L'，证明 $L'\le_p L$，而 “$\le_p$” 的关键在于能够从L'的实例I'(在多项式时间内) 构建出L的一个实例I'

恰好覆盖问题

- 问题描述：给定有穷集 $A = \{a_1, a_2, \ldots, a_n\}$ 和 A 的子集的集合 $W = \{S_1, S_2, \ldots, S_m\}$，设 $U \subseteq W$，如果 U 中的集合元素不相交且并集等于 A，则称 U 是 A 的恰好覆盖。问当前实例是否存在这样的U？
- 设 $A=\{1,2,3,4,5\}$, $W=\{S_1, S_2, S_3, S_4\}$, $S_1=\{1,2\}$, $S_2=\{1,3,4\}$, $S_3=\{2,4\}$, $S_4=\{2,5\}$,
  - 则$\{S_2, S_4\}$是A的恰好覆盖
  - 如果把S_4修改为$S_4=\{3,5\}$, 则不存在A的恰好覆盖
- 恰好覆盖问题是NP-完全的

子集和问题

- 问题描述：给定正整数集合 $X=\{x_1, x_2,\ldots, x_n\}$及正整数 N, 问存在X的子集 T, 使得 T 中的元素之和等于N？
- 求证子集和问题是 NP完全的
- 证明思想：往证恰好覆盖$\leq_p$子集和
  - 给定有穷集 $\mathrm{A}=\{\mathrm{a}_1, \mathrm{a}_2,\ldots, \mathrm{a}_\mathrm{n}\}$和 A的子集的集合 $\mathrm{W}=\{\mathrm{S}_1, \mathrm{S}_2,\ldots, \mathrm{S}_\mathrm{m}\}$
  - 对应的子集和形如 $\mathrm{X}=\{\mathrm{x}_1, \mathrm{x}_2,\ldots,\mathrm{x}_\mathrm{m}\}$及非负整数N
  - 从W中选择S_j恰好覆盖A相当于从X中选择x_j等于N

S_j对应x_j

构造思想

- 每个x_j 和 N 都可表示成 kn 位的二进制数, $k=\lceil \log_2(m+1) \rceil$
- 将这kn 位分成 n段, 每段 k 位二进制
- 构造二进制N，每段的最右位值为1,其余的为0
- 基于子集S_j构造二进制x_j，如果$a_i \in S_j$, 从左到右x_j的第i段的最右位值为1，其余全为0

构造举例

- 恰好覆盖问题原始实例：
  - $A=\{a_1, a_2, a_3, a_4\}=\{2,3,5,8\}$, $W=\{S_1, S_2, S_3\}$,
  - $S_1=\{a_1, a_2\}= \{2,3\}$, $S_2=\{a_1, a_3, a_4\}= \{2,5,8\}$, $S_3=\{a_2\}= \{3\}$
  - $S_2 \cup S_3$是A的全覆盖
- 转化为二进制的子集和问题
  - n=4
  - m=3，则$k=\lceil \log_2(3+1) \rceil=2$
  - N = 01 01 01 01
  - x_1= 01 01 00 00
  - x_2= 01 00 01 01
  - x_3= 00 01 00 00

如果$a_i \in S_j$, 从左到右x_j的第i段的最右位值为1，其余全为0

思考：为什么每段需要k位？
求子集和问题时，最多m个数相加，k保证了不会进位到上一段中。

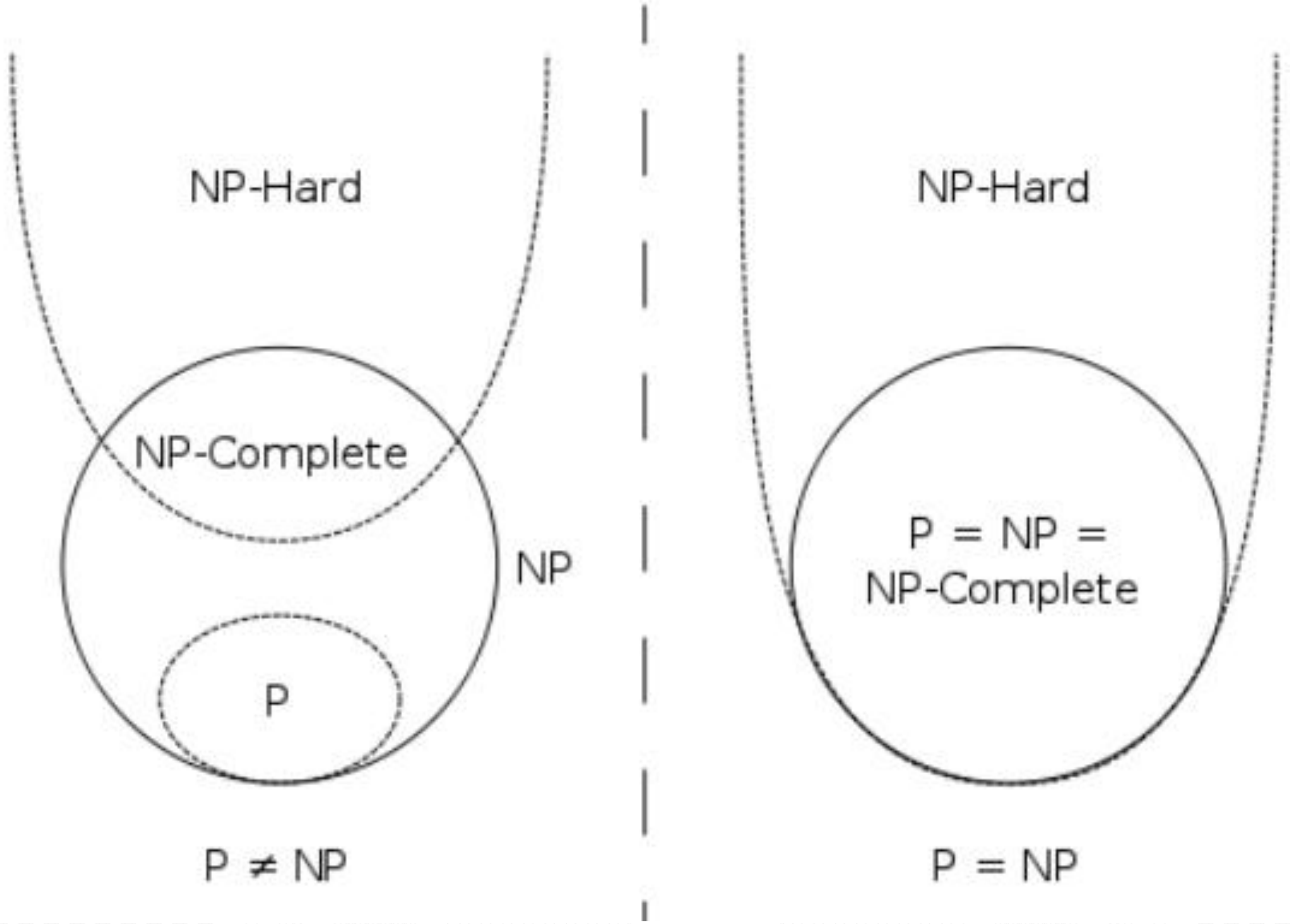
易见$x_2+x_3=N$

思考：子集和 $\leq_p$ 0/1背包问题？

9.4 小结

- P-问题：已经找到多项式算法的问题
- NP-问题：可能找到多项式算法的问题
- 在研究NP问题的过程中找出了一类非常特殊的NP问题，即NPC问题，只要证明出NPC中的某个问题是P问题，那么所有的NP问题都是P问题，即P=NP
- SAT问题是第一个被发现的NPC问题，它归约出了其他NPC问题
- NPC-问题：SAT问题可以多项式时间归约到的NP问题
- NP-难问题： SAT问题可以多项式时间归约到的问题
- 归约的目的：通过不断归约，不断寻找复杂度虽然不会降低，但应用范围更广的算法来代替复杂度低，应用范围小的一类问题的算法。

P、NP、NP-完全、NP-难之间的关系



本 章 结 束