算法分析与
计算复杂性理论

屈 婉 玲

qwl@pku.edu.cn

课 程 简 介

课程名称：算法分析与计算复杂性理论

Analysis of Algorithms and

Theory of Computational Complexity

课程类型：计算机专业研究生必修课

基本目的：

掌握组合算法设计的基本技术

掌握算法分析的基本方法

掌握计算复杂性理论的基本概念

学习应用算法理论处理实际问题

课 程 内 容

顺序算法设计的基本技术

分治策略　　动态规划

回朔算法　　贪心法

概率算法

顺序算法分析的基本方法

评价算法的标准　　算法复杂性的估计

问题复杂性的下界　　算法分析的实例

计算复杂性理论

Turing 机　　计算复杂性的概念

NP 完全性理论及其应用

NP 完全理论的拓广

1	前言	13	Turing机
2	数学基础	14	计算复杂性理论
3	分治策略(1)	15	NP完全性理论(1)
4	分治(2)、动态规划(1)	16	NP完全性理论(2)
5	动态规划(2)	17	Cook定理
6	回溯法	18	NP完全性证明(1)
7	分支估界	19	NP完全性证明(2)
8	贪心法(1)	20	NP完全理论应用(1)
9	贪心法(2)	21	NP完全理论应用(2)
10	概率算法	22	NP完全理论的拓广
11	算法分析技术(1)	23	小结
12	算法分析技术(2)		

参　考　书

1. Algorithm Design, Jon Kleinberg, Eva Tardos, Addison-Wesley, 清华大学出版社影印版，2006.
2. Introduction to Algorithms（Second Edtion）, Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest, The MIT Press 2001. 高教出版社影印版，2002.
3. 计算机和难解性 NP完全性理论导引，M. R. 加里, D. S. 约翰逊, 张立昂等译，科学出版社, 1987.

*4. 计算复杂性导论，堵丁柱，葛可一，王洁，高教出版社，2002.

*5. Limits to Parallel Computation: P- Completeness Theory, Raymond Greenlaw, H. James Hoover, Walter L. Ruzzo, Oxford University Press, 1995.

学 习 安 排

平时作业：40%

期末笔试：50%

小论文：　10%

结合研究工作

算法设计或分析

辅 助 教 学

- 网上教学平台：北大主页 / 网络教学
  - 教案PPT发布
  - 通知发布
  - 作业发布
  - 讨论
- 助教：
  - 批改作业
  - 讨论

助教联系方法

- 朱成：zhucheng@tcs.pku.edu.cn
- 李鑫：lixin@tcs.pku.edu.cn
- 严浩：yanhao@tcs.pku.edu.cn
- 徐重远：xzy@tcs.pku.edu.cn
- 熊文英：flykite@tcs.pku.edu.cn

- 地点：理科一号楼1708

引　言

理论上与现实上的可计算

- 几个实例
- 算法研究的基本问题
- 理论上可计算——可计算性理论
- 现实上可计算——计算复杂性理论

几个实例

1. 投资问题

条件：

m 元钱，投资给 n 个项目，

n 个效益函数 $f_i(x)$，第 i 个项目投入 x 元钱的效益

问题：

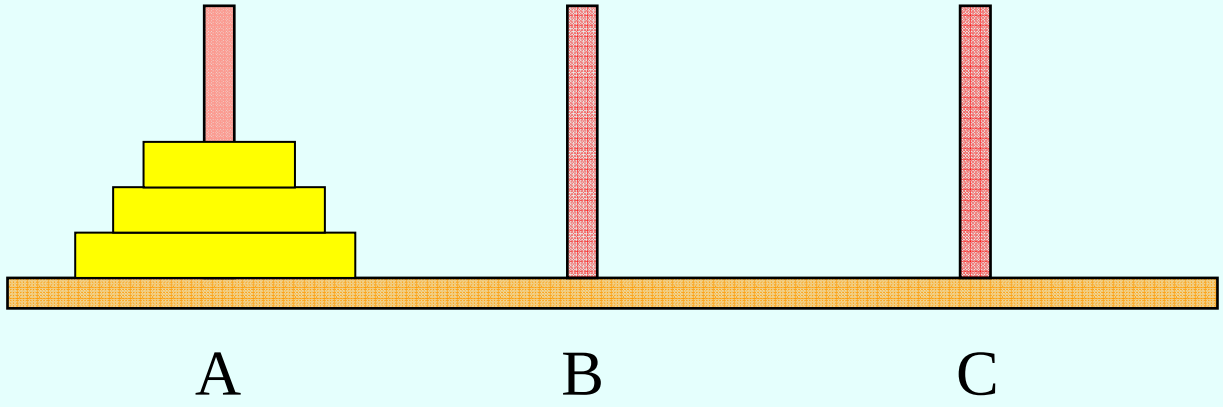
如何分配每个项目的钱数使得总效益最大？

需要：

求解方法

方法的效率

2. Hanoi塔问题



$T(n) = 2\,T(n-1) + 1$，$T(1) = 1$，解得 $T(n) = 2^n - 1$

1秒移1个，64个盘子要多少时间？（5000亿年）

思考：是否存在更好的解法？

Reve难题：Hanoi塔变种，柱数增加，允许盘子相等.

其他变种：奇偶盘号分别放置

3. 搜索问题

输入：数组 L，x

输出：x 是否在 L 中？如果在输出它的下标

4. 排序问题

输入：n 个数

输出：按递增顺序排好的 n 个数

5. 选择问题

输入：n 个数的集合 S，正整数 k（$1 \le k \le n$)

输出：S 中第 k 小的数

需要：

现有的算法中哪个算法最好？

是否存在更有效的算法？

著名公式

Algorithm + Data Structure = Programming

好的算法

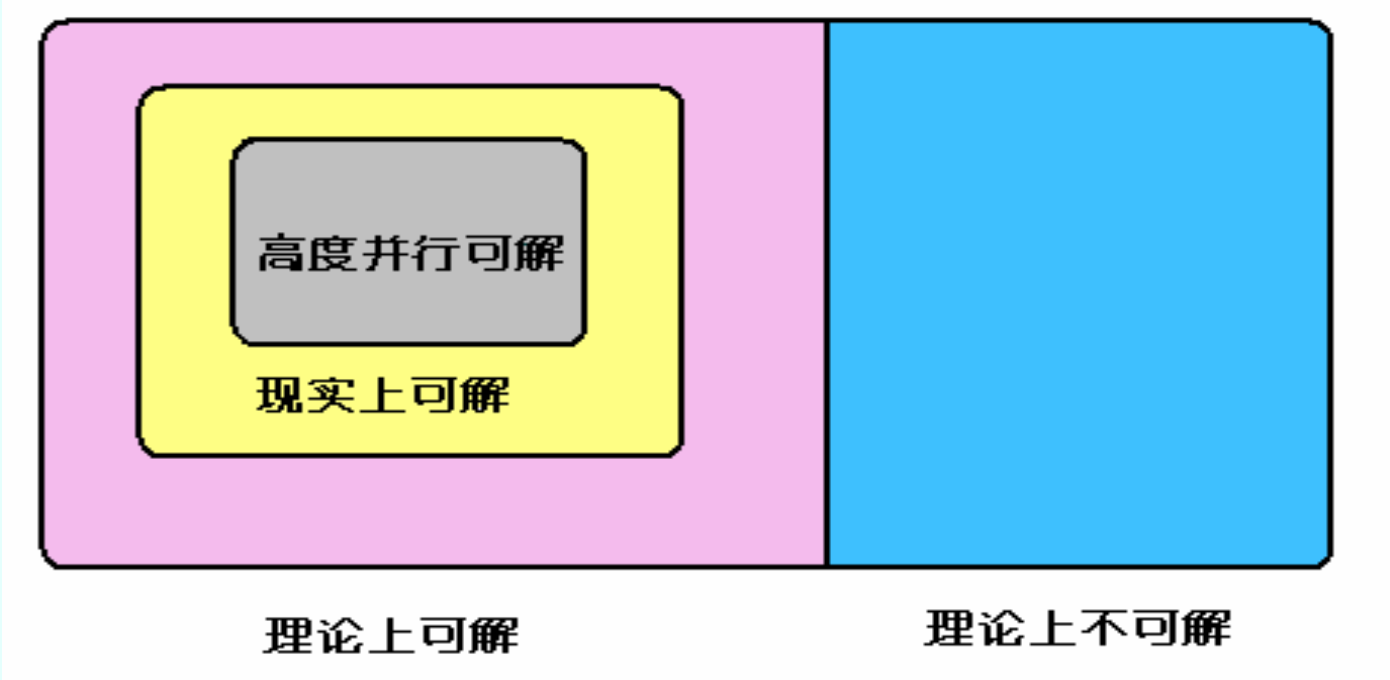
提高求解问题的效率

节省存储空间

需要解决的问题

问题→一个求解算法：	算法设计技术
算法→算法的评价：	算法分析技术
算法类→问题复杂性下界：	问题复杂性分析
问题类→能够求解的边界：	计算复杂性理论

问题类的复杂性



理论上不可解的问题：没有算法

理论上可解的问题：存在确定型算法(Turing 机)

现实上可解的问题：存在多项式时间的算法

高度并行可解的问题：存在对数多项式时间并行算法

理论上的可计算——可计算性理论

研究目标

确定什么问题是可计算的

合理的计算模型

已有的：递归函数、**Turing** 机、λ 演算、**Post** 系统等

条件：计算一个函数只要有限条指令

每条指令可以由模型中的有限个计算步骤完成

指令执行的过程是确定的

Church-Turing 论题

如果一个函数在某个合理的计算模型上可计算，那么它在 Turing 机上也是可计算的

可计算性是不依赖于计算模型的客观性质

现实上可计算——计算复杂性理论

内容

算法复杂性——算法所使用的时间、空间的估计

问题复杂性——估计问题的难度

术语和概念

问题

算法

算法的时间复杂性

函数的阶

多项式时间的算法与指数时间的算法

问题的复杂性分析

问　题

问题： 需要回答的一般性提问，通常含有若干参数.

问题描述所包含的内容：

对问题参数的一般性描述

解满足的条件

问题的实例： 对问题的参数的一组赋值

例 1　巡回售货员

问题：有穷个城市的集合 $C = \{c_1, c_2, \ldots, c_m\}$,

正整数 $d(c_i, c_j) = d(c_j, c_i)$, $1 \le i < j \le m$.

解：　$< c_{k_1}, c_{k_2}, \ldots, c_{k_m} >$

使得 $k_1, k_2, \ldots, k_m$ 是 1, 2, …, m 的置换且满足

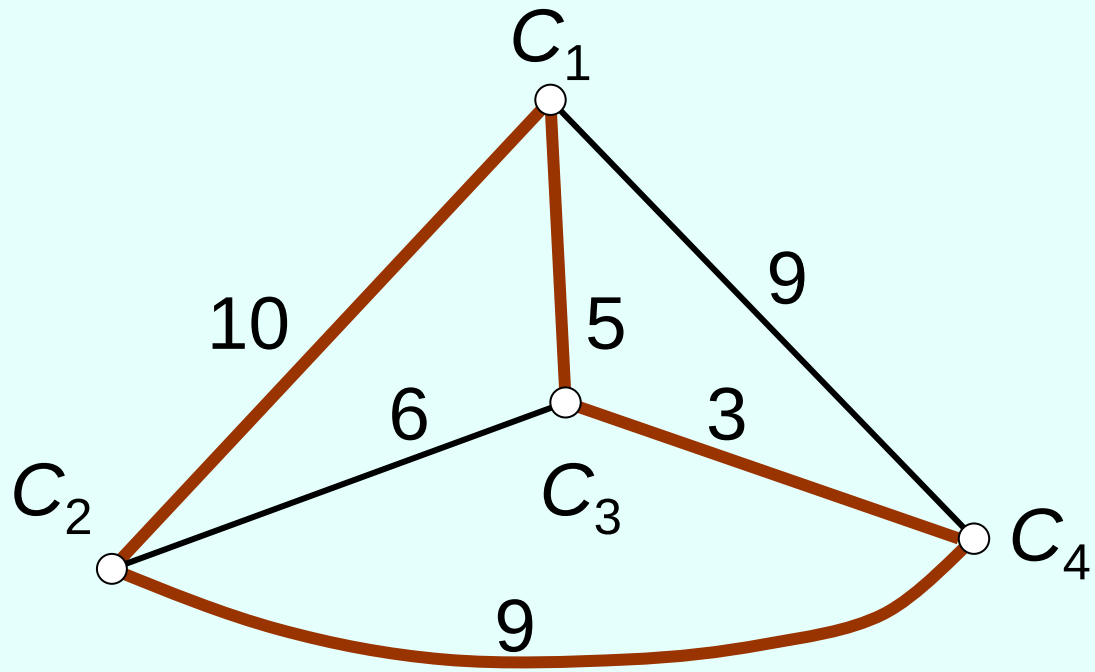
$$\min\left(\sum_{i=1}^{m-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_m}, c_{k_1})\right)$$

实例

$C = \{c_1, c_2, c_3, c_4\}$,

$d(c_1,c_2) = 10$, $d(c_1,c_3) = 5$, $d(c_1,c_4) = 9$

$d(c_2,c_3) = 6$, $d(c_2,c_4) = 9$, $d(c_3,c_4) = 3$



算　法

非形式定义

有限条指令的集合

指令集确定了解决某个问题的运算或操作的序列

输入个数大于等于0

输出个数大于0

形式定义

对所有的有效输入停机的Turing机

算法*A*解问题*P*

把问题*P*的任何实例作为算法*A*的输入， *A*能够在有限步停机，并输出该实例的正确的解。

算法的描述——伪码

- 保持程序的主要结构
  - 类C, Pascal
  - 赋值语句：←
  - 分支语句：if …then …[else…]
  - 循环语句：while, for，repeat until
  - 转向语句：goto
  - 调用
  - 注释：***//*…***
- 允许使用自然语言
- 常忽略数据结构、模块、异常处理等细节
- 常忽略变量说明

伪码的例子

例2　插入排序

算法 **INSERTION-SORT(*A*)**

1. **for** $j \leftarrow 2$ **to** ***length*[*A*]**

2.　　$key \leftarrow A[j]$　//*将$A[j]$插入排好序的$A[1..j-1]$

3.　　$i \leftarrow j-1$

4.　　**while** $i > 0$ **and** $A[i] > key$

5.　　　　$A[i+1] \leftarrow A[i]$

6.　　　　$i \leftarrow i-1$

7.　　$A[i+1] \leftarrow key$

算法的时间复杂度

最坏情况下的时间复杂度

算法求解输入规模为 n 的实例所需要的最长时间 $W(n)$

平均情况下的时间复杂度

算法求解输入规模为 n 的实例所需要的平均时间 $A(n)$

复杂度表示

针对问题选择基本运算

将基本运算次数表示为输入规模的函数

例 3　搜索问题

输入：非降顺序排列的数组 L，元素数为 n. 数 x

输出：j. 若 x 在 L 中，j 是 x 首次出现的序标；

否则 $j = 0$

算法　顺序搜索

假设　x 在 L 中的概率为 p

x 在 L 中不同位置是等概分布的，则

$$W(n) = n$$

$$A(n) = \sum_{i=1}^{n} i\frac{p}{n} + (1-p)n = \frac{p(n+1)}{2} + (1-p)n$$

复杂度函数的阶

设 f 和 g 是定义域为自然数 N 上的函数，

$f(n)=O(g(n))$.

若存在正数 c 和 n_0 使得对一切 $n\geq n_0$ 有 $0\leq f(n)\leq cg(n)$

$f(n)=\Omega(g(n))$

若存在正数 c 和 n_0 使得对一切 $n\geq n_0$ 有 $0\leq cg(n)\leq f(n)$

$f(n)=o(g(n))$.

若对任意正数 c 存在 $n_0>0$ 使得对一切 $n\geq n_0$ 有 $0\leq f(n)<cg(n)$

$f(n)=\omega(g(n))$当且仅当 $g(n)=o(f(n))$

$f(n)=\Theta(g(n))$当且仅当 $f(n)=O(g(n))$ 且 $f(n)=\Omega(g(n))$

$O(1)$表示常数函数

函数渐近阶的基本性质

1. 设f 和 g 是两个函数，$\lim_{n\to\infty}\frac{f(n)}{g(n)}$ 存在，并且等于某个常数$c>0$，那么$f(n)=\Theta(g(n))$.

2. (a) 如果$f=O(g)$且 $g=O(h)$，那么$f=O(h)$.
   (b) 如果$f=\Omega(g)$且 $g=\Omega(h)$，那么$f=\Omega(h)$.
   (c) 如果$f=\Theta(g)$ 且 $g=\Theta(h)$，那么$f=\Theta(h)$.

3. 假设 f 和 g是两个函数，若对某个其它的函数h, 我们有$f=O(h)$和$g=O(h)$，那么$f+g=O(h)$.

4. 假设 f 和 g是两个函数（取非负的值），满足$g=O(f)$. 那么$f+g=O(f)$.

基本函数类

- 阶的高低

  指数级：$2^n, 3^n, n!, \ldots$

  多项式级：$n, n^2, n\log n, n^{1/2}, \ldots$

  对数多项式级：$\log n, \log^2 n, \ldots$

$$2^{2^n}, \quad n!, \quad n2^n, \quad (3/2)^n, \quad (\log n)^{\log n} = \Theta(n^{\log\log n}),$$

$$n^3, \quad \log(n!) = \Theta(n\log n), \quad n = 2^{\log n},$$

$$\log^2 n, \quad \log n, \quad \sqrt{\log n}, \quad \log\log n,$$

$$n^{1/\log n} = \Theta(1)$$

- 阶的渐近性

例4　PrimalityTest(n)
输入：n, n为大于2的奇整数
输出：true 或者false
1．$s \leftarrow \sqrt{n}$
2．for $j \leftarrow 2$ to s
3．　　if j 整除n
4．　　then return false
5．return true

假设计算$\sqrt{n}$ 可以在$O(1)$时间完成，可以写$O(\sqrt{n})$,
不能写$\Theta(\sqrt{n})$

例 5 设 $f(n)=\frac{1}{2}n^2-3n$,

证明 $f(n)=\Theta(n^2)$

证 若存在正数 c,d 使得

$$c \le \frac{1}{2}-\frac{3}{n} \le d,$$

那么 $d \ge 1/2$, $n \ge 1$; $c \le 1/14$, $n \ge 7$

取 $c=1/14$, $d=1/2$, $n_0=7$

例 6 设 $f(n)=6n^3$, 证明 $f(n) \ne \Theta(n^2)$

证 要使 $6n^3 \le cn^2$, 则 $6n \le c$,

即 $n \le c/6$, 与 c 是常数矛盾

多项式时间的算法
与指数时间的算法

多项式时间的算法

时间复杂度函数为 $O(p(n))$的算法，

其中 $p(n)$是 n 的多项式

指数时间的算法

不存在多项式 $p(n)$使得该算法的时间复杂度为 $O(p(n))$

时间复杂度函数	问题规模					
	10	20	30	40	50	60
n	10^{-5}	$2*10^{-5}$	$3*10^{-5}$	$4*10^{-5}$	$5*10^{-5}$	$6*10^{-5}$
n^2	10^{-4}	$4*10^{-4}$	$9*10^{-4}$	$16*10^{-4}$	$25*10^{-4}$	$36*10^{-4}$
n^3	10^{-3}	$8*10^{-3}$	$27*10^{-3}$	$64*10^{-3}$	$125*10^{-3}$	$216*10^{-3}$
n^5	10^{-1}	3.2	24.3	1.7分	5.2分	13.0分
2^n	.001秒	1.0秒	17.9分	12.7天	35.7年	366世纪
3^n	.059秒	58分	6.5年	3855世纪	$2*10^8$世纪	$1.3*10^{13}$世纪

时间复杂度函数	1小时可解的问题实例的最大规模		
	计算机	快100倍计算机	快1000倍计算机
n	N_1	100 N_1	1000 N_1
n^2	N_2	10 N_2	31.6 N_2
n^3	N_3	4.64 N_3	10 N_3
n^5	N_4	2.5 N_4	3.98 N_4
2^n	N_5	N_5 + 6.64	N_5 + 9.97
3^n	N_6	N_6 + 4.19	N_6 + 6.29

问题的复杂度分析

多项式时间可解的问题与难解的问题

多项式时间可解的问题 P

存在着解 **P** 的多项式时间的算法

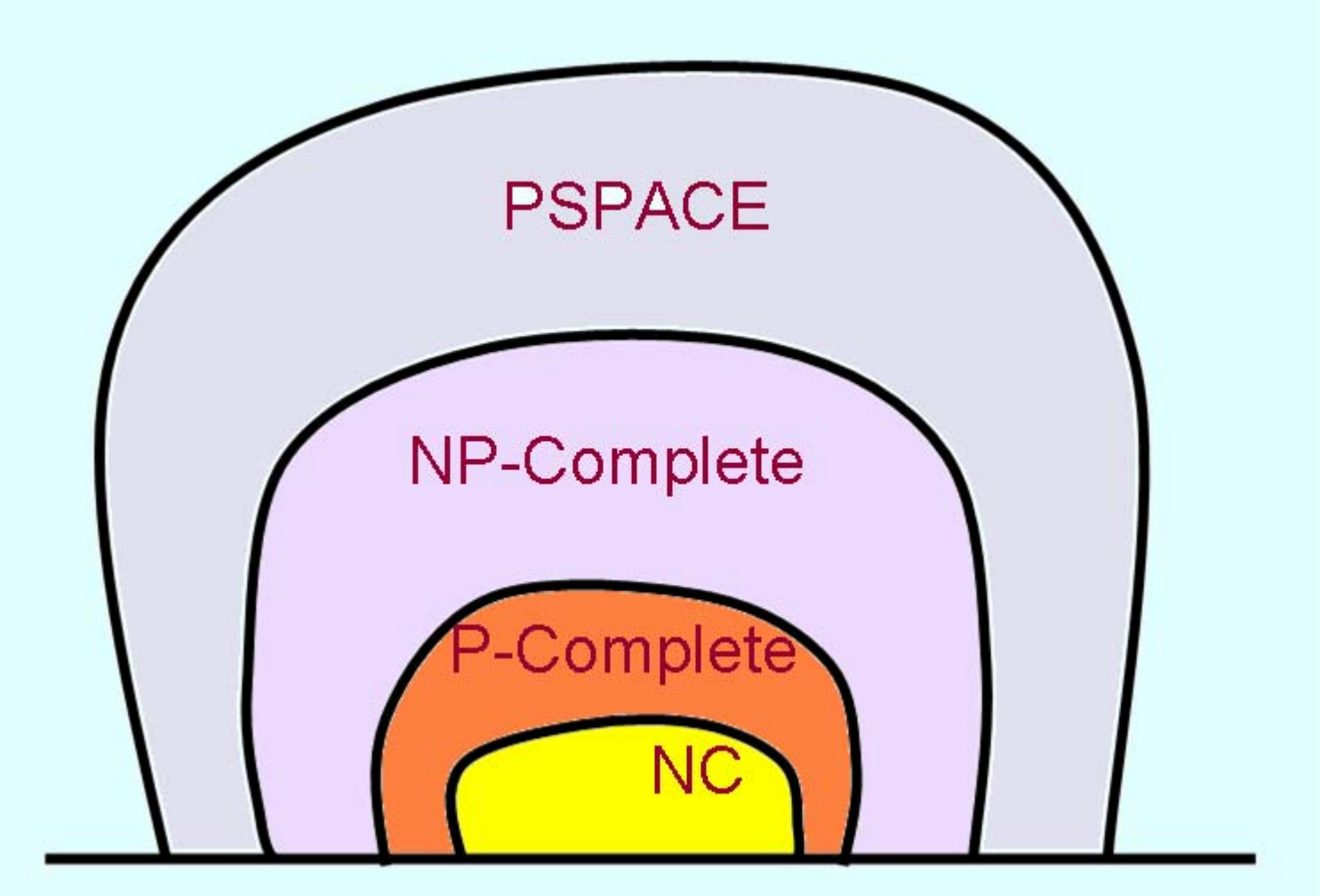
难解的问题 P

不存在解 **P** 的多项式时间的算法

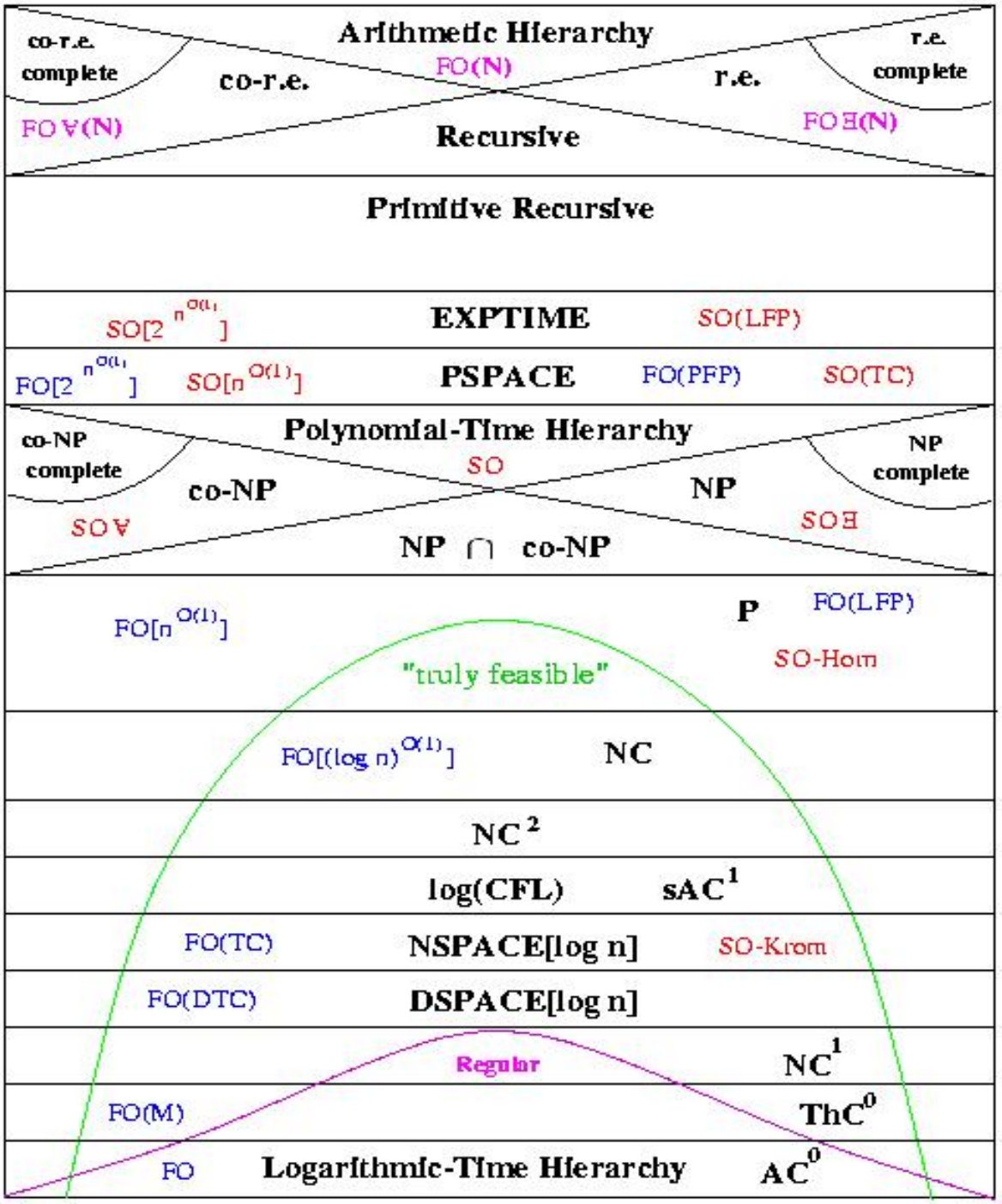
实际上可计算的问题

多项式时间可解的问题

不同判定问题类的层次结构



计算复杂性类的谱系


Arithmetic Hierarchy
co-r.e. complete
co-r.e.
FO(N)
r.e.
r.e. complete
FO∀(N)
FO∃(N)
Recursive
Primitive Recursive
EXPTIME
SO(LFP)
PSPACE
FO(PFP)
SO(TC)
Polynomial-Time Hierarchy
co-NP complete
co-NP
SO
NP
NP complete
SO∀
SO∃
NP ∩ co-NP
P
FO(LFP)
SO-Horn
"truly feasible"
NC
NC²
log(CFL)
sAC¹
FO(TC)
NSPACE[log n]
SO-Krom
FO(DTC)
DSPACE[log n]
Regular
NC¹
FO(M)
ThC⁰
FO
Logarithmic-Time Hierarchy
AC⁰

有关算法及计算复杂性理论的拓广

- 算法：
  - 概率算法
  - 在线算法
- 计算复杂性
  - 概率Turing机与概率复杂性
  - 近似解的复杂性

数 学 基 础

符号说明

取整函数

对数

阶乘

求和

估计和式的上界

递推方程求解

递推方程中涉及$\lfloor x \rfloor$和$\lceil x \rceil$的处理方法

符 号 说 明

取整函数

$\lfloor x \rfloor$：小于等于 x 的最大整数

$\lceil x \rceil$：大于等于 x 的最小整数

性质

$$x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$$

$$\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n$$

$$\left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil, \quad \left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$$

对 数

$$\log n = \log_2 n, \quad \lg n = \log_2 n$$

$$\log^k n = (\log n)^k$$

$$\log\log n = \log(\log n)$$

性质：

$$a^{\log_b n} = n^{\log_b a}$$

$$\log_k n = c \log_l n$$

阶 乘

Stirling 公式

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

函数的阶

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\log n! = \Theta(n \log n)$$

求　和

基本公式　$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$

$$\sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$$

例1　$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} \left(\frac{1}{k} - \frac{1}{k+1}\right)$$

$$= \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=1}^{n-1} \frac{1}{k+1}$$

$$= \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=2}^{n} \frac{1}{k} = 1 - \frac{1}{n}$$

例2

$$\sum_{t=1}^{k} t\,2^{t-1} = \sum_{t=1}^{k} t\left(2^{t} - 2^{t-1}\right)$$

$$= \sum_{t=1}^{k} t\,2^{t} - \sum_{t=1}^{k} t\,2^{t-1}$$

$$= \sum_{t=1}^{k} t\,2^{t} - \sum_{t=0}^{k-1} (t+1)2^{t}$$

$$= \sum_{t=1}^{k} t\,2^{t} - \sum_{t=0}^{k-1} t\,2^{t} - \sum_{t=0}^{k-1} 2^{t}$$

$$= k\,2^{k} - (2^{k} - 1)$$

$$= (k-1)2^{k} + 1$$

估计和式的上界

方法一：放大法

$$\sum_{k=1}^{n} a_k \le n a_{\max}$$

$\frac{a_{k+1}}{a_k} \le r$，　对于一切$k \ge 0$，　$r < 1$为常数，则

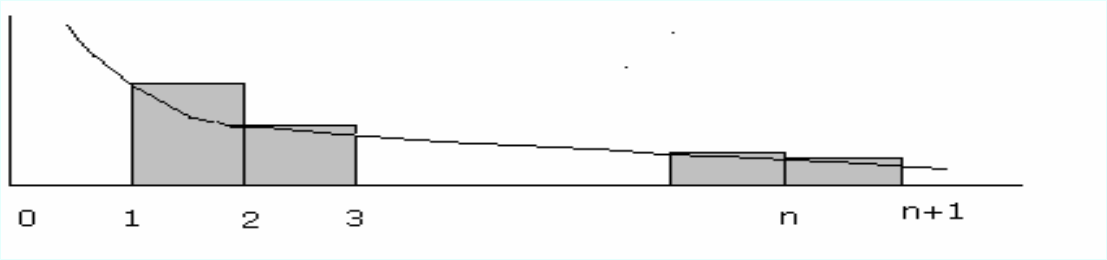
$$\sum_{k=0}^{n} a_k \le \sum_{k=0}^{\infty} a_0 r^k = a_0 \sum_{k=0}^{\infty} r^k = \frac{a_0}{1-r}$$

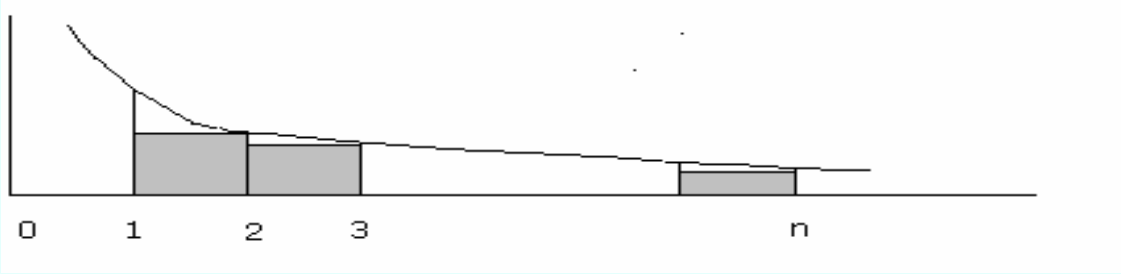
方法二：利用积分

例3

$$\sum_{k=1}^{n}\frac{1}{k}\geq\int_{1}^{n+1}\frac{dx}{x}=\ln(n+1)$$

$$\sum_{k=1}^{n}\frac{1}{k}=\frac{1}{1}+\sum_{k=2}^{n}\frac{1}{k}\leq 1+\int_{1}^{n}\frac{dx}{x}=\ln n+1$$

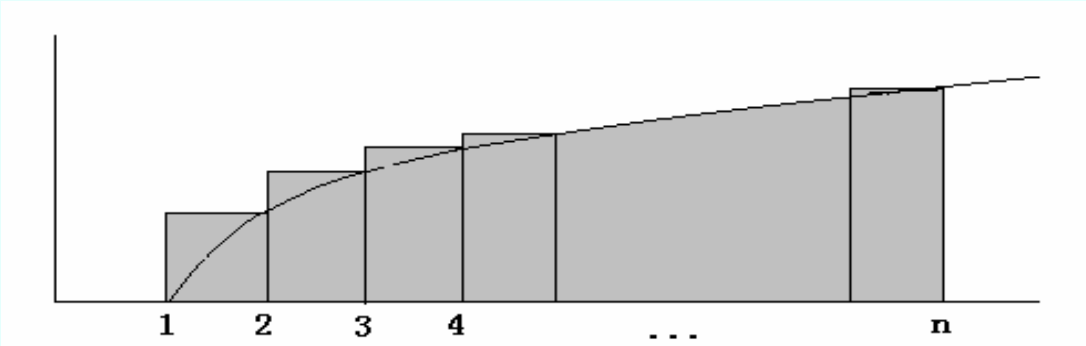


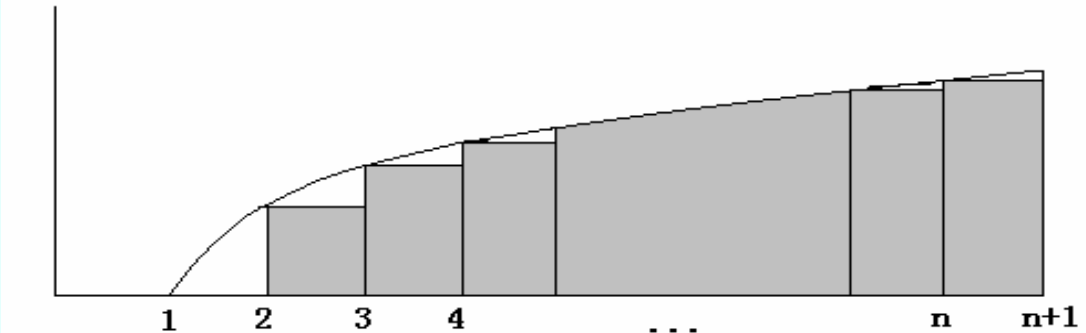


例4　$\log n! = \Theta(n \log n)$

$$\int_2^{n+1} \log x dx \geq \sum_{k=1}^{n} \log k = \log n! \geq \int_1^{n} \log x dx$$

$$\int_2^{n+1} \log x dx = \int_1^{n} \log x dx = \Theta(n \log n)$$





递 推 方 程 求 解

递推方程定义

给定数列 $f(0),f(1),\ldots,f(n)$，一个把 $f(n)$和某些 $f(i)$，$0\le i<n$，联系起来的等式称为递推方程

给定关于 $f(n)$的递推方程和初值，求 $f(n)$称为解递推方程

求解方法

- 公式法
- 迭代归纳法
- 尝试法
- 递归树法
- 换元法
- 差消法
- 生成函数法
- Master 定理

常系数线性齐次递推方程的求解（公式法）

标准形式：k 阶

$$H(n)-a_1H(n-1)-a_2H(n-2)-\cdots-a_kH(n-k)=0,$$

$n\geq k, a_1, a_2, ..., a_k$是常数，$a_k\neq 0$

求通解：

特征方程 $x^k-a_1x^{k-1}-...-a_k=0$

特征根 $q_1, q_2, \ldots, q_k$，

通解 $H(n)=c_1q_1^n+c_2q_2^n+\ldots+c_kq_k^n$

如果有重根，q 是 e 重特征根，通解对应于根 q 的部分

$$(C_1+C_2n+...+C_en^{e-1})q^n$$

整个通解为各个不等的特征根的对应部分之和

代入初值确定待定常数

例 5　$H(n)+H(n-1)-3H(n-2)-5H(n-3)-2H(n-4)=0$

$H(0)=1, H(1)=0, H(2)=1, H(3)=2$

特征方程　$x^4+x^3-3x^2-5x-2=0$，特征根$-1,-1,-1,2$,

通解为 $H(n)=(c_1+c_2n+c_3n^2)(-1)^n+c_4 2^n$

$$\begin{cases} c_1+c_4=1 \\ -c_1-c_2-c_3+2c_4=0 \\ c_1+2c_2+4c_3+4c_4=1 \\ -c_1-3c_2-9c_3+8c_4=2 \end{cases}$$

解得　$c_1=\frac{7}{9}, c_2=-\frac{1}{3}, c_3=0, c_4=\frac{2}{9}$

解为　$H(n)=\frac{7}{9}(-1)^n-\frac{1}{3}n(-1)^n+\frac{2}{9}2^n$

常系数线性非齐次递推方程求解（公式法）

标准形

$$H(n)-a_1H(n-1)-a_2H(n-2)-\ldots-a_kH(n-k)=f(n)$$

$$H(0)=d_0, H(1)=d_1, H(2)=d_2, \ldots, H(k-1)=d_{k-1}$$

通解为对应的齐次通解加上特解

$$H(n)=\overline{H}(n)+H^*(n)$$

特解的函数形式依赖于 $f(n)$

求解的关键是用待定系数法确定一个特解 $H^*(n)$

换元法：转化成常系数线性递推方程

例 6　归并排序

$$T(n) = 2\,T(\,n/2\,) + n-1\,,\quad n = 2^k$$

$$T(2) = 1$$

$$H(k) = 2H(k-1) + 2^k - 1$$

$$H(1) = 1$$

令　$H^*(k) = P_1 k 2^k + P_2$，解得

$$P_1 = P_2 = 1,\quad H^*(k) = k2^k + 1$$

通解　$H(k) = C\,2^k + k2^k + 1,$

代入初值，得

$$C = -1$$

$$H(k) = -2^k + k2^k + 1$$

$$T(n) = n \log n - n + 1$$

叠代归纳法

差消法----化简递推方程

例7　求解递推方程

$$T(n)=\frac{2}{n}\sum_{i=1}^{n-1}T(i)+n-1,\quad n\geq 2$$

$$T(1)=0$$

$$nT(n)=2\sum_{i=1}^{n-1}T(i)+n^2-n$$

$$(n-1)T(n-1)=2\sum_{i=1}^{n-2}T(i)+(n-1)^2-(n-1)$$

相减并化简得

$$nT(n)=(n+1)T(n-1)+2n-2$$

$$\frac{T(n)}{n+1}=\frac{T(n-1)}{n}+\frac{2}{n+1}-\frac{2}{(n+1)n}$$

由叠代得

$$\frac{T(n)}{n+1}=\frac{2}{n+1}+\frac{2}{n}+\frac{2}{n-1}+\ldots+\frac{2}{3}+\frac{T(1)}{2}-O(1)$$

$$=2\left(\frac{1}{n+1}+\frac{1}{n}+\ldots+\frac{1}{3}\right)-O(1)$$

$T(n)=O(n\log n)$

尝试法----估计递推方程的阶

猜想$T(n)$的阶

代入方程验证，比较两边的阶的高低

如果右边高，则提高$T(n)$的阶

否则，降低$T(n)$的阶

例8 求解递推方程

$$T(n)=\frac{2}{n}\sum_{i=1}^{n-1}T(i)+n-1,\quad n\geq 2$$

$$T(1)=0$$

设$T(n)$的阶为常数c, 则

$$c = 2c + n - 1 - \frac{2c}{n}$$

右边高

设$T(n)$的阶为cn，则

$$cn = \frac{2}{n}\sum_{i=1}^{n-1} ci + n - 1$$

$$= c(n-1) + n - 1$$

$$= (c+1)n - c - 1$$

右边高

设$T(n)$的阶为cn^2, 则

$$cn^2 = \frac{2}{n}\sum_{i=1}^{n-1} ci^2 + n - 1$$

$$= \frac{2}{n}[c\frac{n^3}{3} + O(n^2)] + n - 1$$

$$= \frac{2}{3}cn^2 + O(n)$$

右边低

设$T(n)$的阶为$cn\log n$ ，则

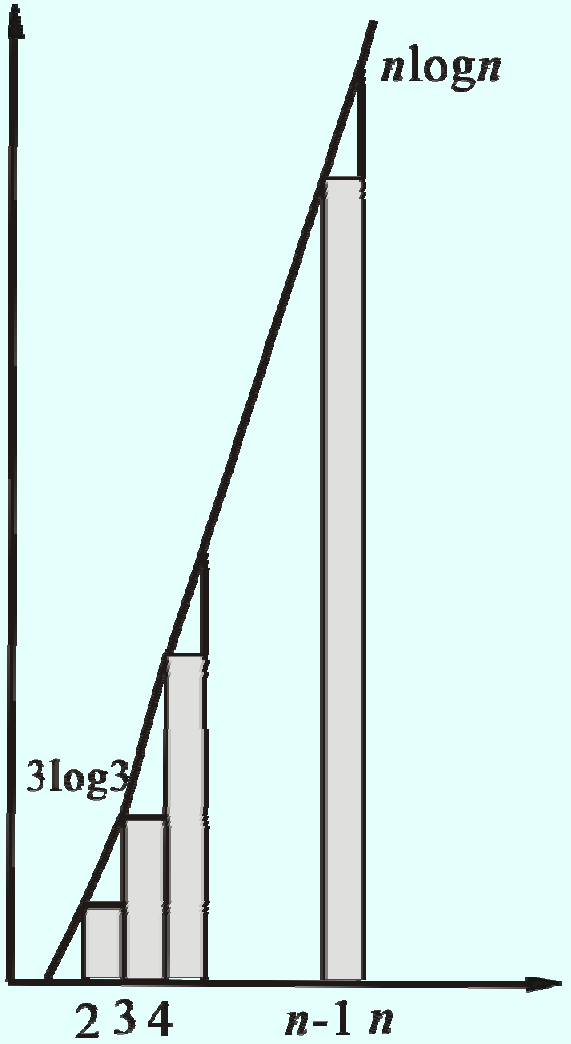
$$cn\log n$$

$$=\frac{2c}{n}\sum_{i=1}^{n-1} i\log i+n-1$$

$$=\frac{2c}{n}[\frac{n^2}{2}\log n-\frac{n^2}{4\ln 2}+O(n\log n)]+n-1$$

$$=cn\log n+(1-\frac{c}{2\ln 2})n+O(\log n)$$

令 $c=2\ln 2$，则方程左、右增长率一致



生成函数法

设序列$\{a_n\}$，构造形式幂级数

$$G(x) = a_0 + a_1x + a_2x^2 + \ldots + a_n x^n + \ldots$$

称 $G(x)$为$\{a_n\}$的生成函数

例如 $\{C(m,n)\}$的生成函数为 $(1+x)^m$

给定正整数 k, $\{k^n\}$的生成函数为

$$G(x) = 1 + kx + k^2x^2 + k^3x^3 + \ldots = 1/(1-kx)$$

生成函数法：

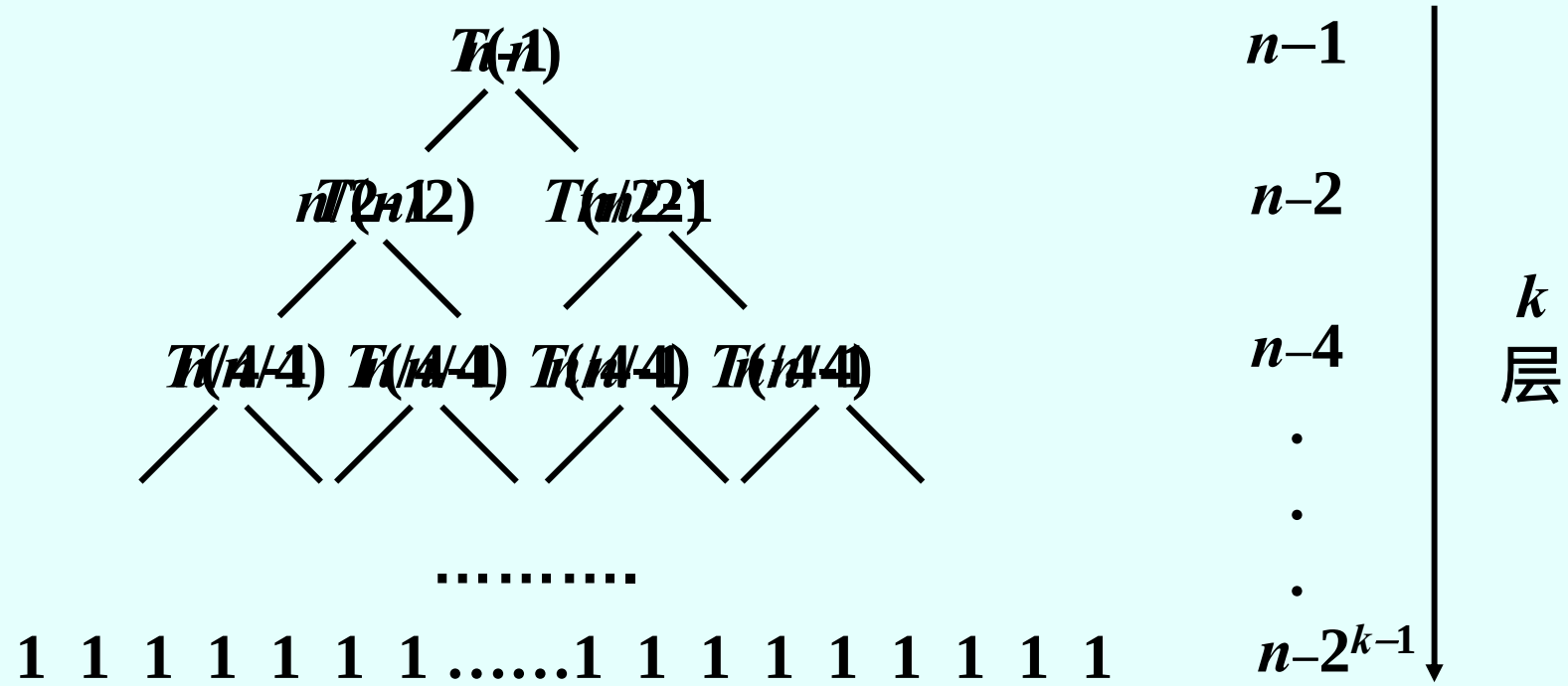
不直接求解递推方程

求出序列对应的生成函数

将函数展开求得序列通项

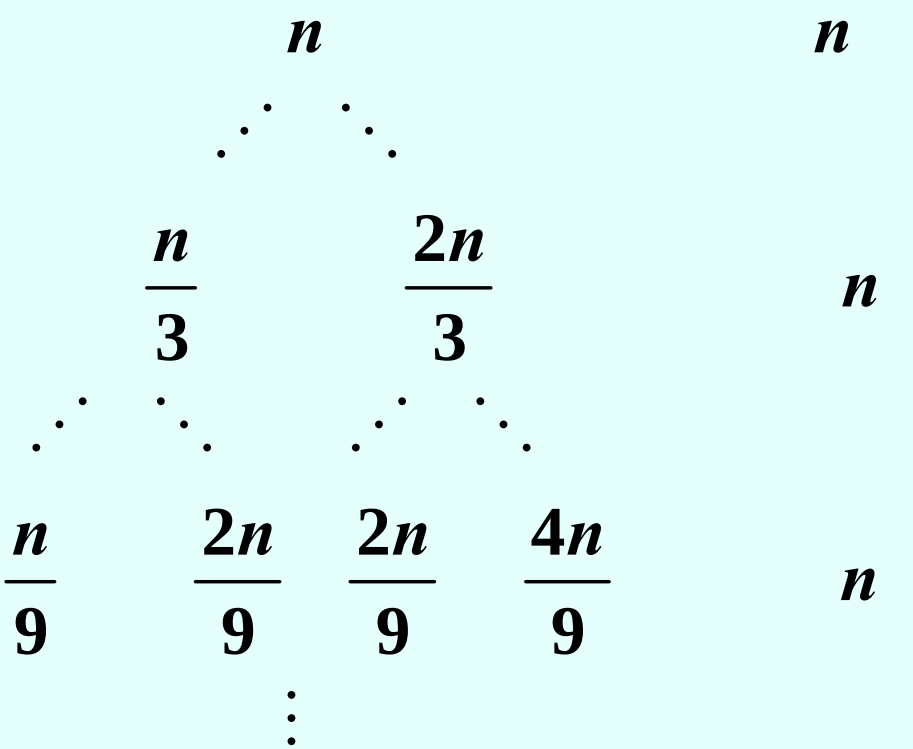
递归树—— 二分归并排序

例9　　$T(n) = 2T(n/2) + n - 1,\ n = 2^k$



$T(n) = nk - (1+2+\ldots+2^{k-1}) = nk - (2^k - 1) = n \log n - n + 1$

例10　$T(n)=T(\frac{n}{3})+T(\frac{2n}{3})+n$



$\log_{3/2} n$ 层　　$\Theta(n \log n)$

Master定理

设$a \geq 1, b > 1$为常数，$f(n)$为函数，$T(n)$为非负整数

$$T(n) = aT(n/b) + f(n),$$

1. $f(n) = O(n^{\log_b a - \varepsilon}), \varepsilon > 0,$

   那么$T(n) = \Theta(n^{\log_b a})$

2. $f(n) = \Theta(n^{\log_b a}),$

   那么$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(f(n) \log n)$

3. $f(n) = \Omega(n^{\log_b a + \varepsilon}), \varepsilon > 0,$且对于某个常数

   $c < 1$和所有的充分大的 n有$af(n/b) \leq cf(n),$

   那么$T(n) = \Theta(f(n))$

迭　代

$$T(n) = aT(\frac{n}{b}) + f(n) = a[aT(\frac{n}{b^2}) + f(\frac{n}{b})] + f(n)$$

$$= a^2 T(\frac{n}{b^2}) + af(\frac{n}{b}) + f(n) = ...$$

$$= a^k T(\frac{n}{b^k}) + a^{k-1} f(\frac{n}{b^{k-1}}) + ... + af(\frac{n}{b}) + f(n)$$

$$= a^{\log_b n} T(1) + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j})$$

$$= cn^{\log_b a} + \sum_{j=0}^{\log_b a - 1} a^j f(\frac{n}{b^j})$$

Case 1： $f(n) = O(n^{\log_b a - \varepsilon})$

$$T(n) = cn^{\log_b a} + \sum_{j=0}^{k-1} a^j f(\frac{n}{b^j})$$

$$= cn^{\log_b a} + O(\sum_{j=0}^{\log_b n - 1} a^j (\frac{n}{b^j})^{\log_b a - \varepsilon})$$

$$= cn^{\log_b a} + O(n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} (b^\varepsilon)^j)$$

$$= cn^{\log_b a} + O(n^{\log_b a - \varepsilon} \frac{b^{\varepsilon \log_b n} - 1}{b^\varepsilon - 1})$$

$$= cn^{\log_b a} + O(n^{\log_b a}) = \Theta(n^{\log_b a})$$

Case 2：$f(n) = \Theta(n^{\log_b a})$

$$T(n) = cn^{\log_b a} + \Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

$$= cn^{\log_b a} + \Theta\left(n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \frac{a^j}{a^j}\right)$$

$$= \Theta(n^{\log_b a} \log n)$$

Case 3 : $f(n) = \Omega(n^{\log_b a + \varepsilon})$

$$T(n) = cn^{\log_b a} + \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j}) \qquad af(n/b) \le cf(n)$$

$$\le cn^{\log_b a} + \sum_{j=0}^{\log_b n - 1} c^j f(n)$$

$$= cn^{\log_b a} + f(n) \sum_{j=0}^{\log_b n - 1} c^j \qquad c < 1$$

$$\le cn^{\log_b a} + f(n)\frac{1}{1-c} = \Theta(f(n)) \qquad f(n) = \Omega(n^{\log_b a + \varepsilon})$$

例11 $T(n)=9T(n/3)+n$

$$a=9, b=3, f(n)=n, \quad n^{\log_3 9}=n^2,$$

$$f(n)=O(n^{\log_3 9-1}), \quad T(n)=\Theta(n^2)$$

例12 $T(n)=T(2n/3)+1$

$$a=1, b=3/2, f(n)=1, n^{\log_{3/2} 1}=n^0=1,$$

$$f(n)=\Theta(n^{\log_{3/2} 1}), T(n)=\Theta(\log n)$$

例13

$$T(n) = 3T(n/4) + n\log n$$

$$a = 3, b = 4, \quad f(n) = n\log n,$$

$$n^{\log_4 3} = O(n^{0.793}),$$

$$f(n) = \Omega(n^{\log_4 3+\varepsilon}), \quad \varepsilon \approx 0.2,$$

$$af(n/b) = 3(n/4)\log(n/4)$$

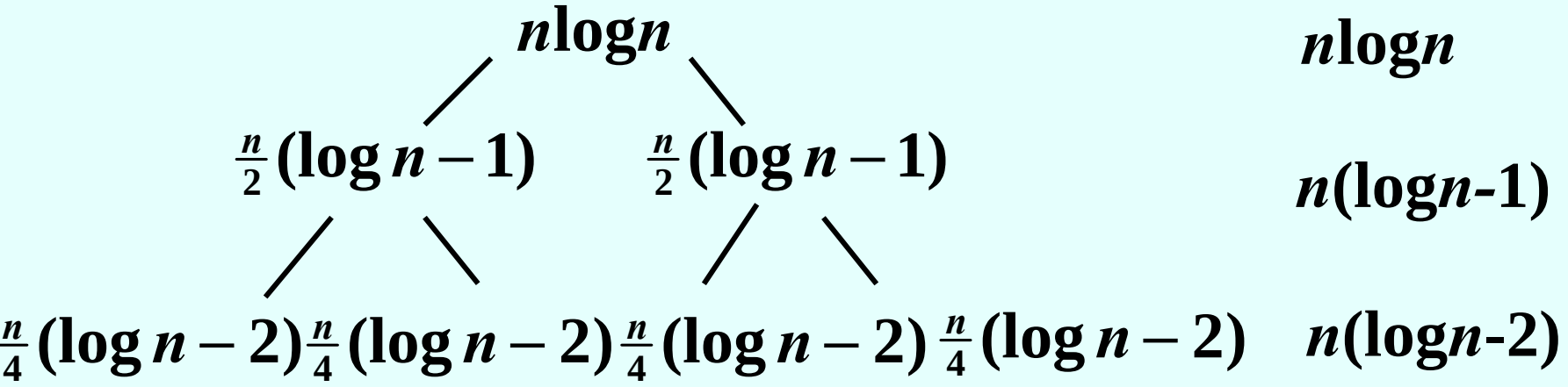
$$\leq (3/4)n\log n = cf(n)$$

$c = 3/4$, n充分大

$$T(n) = \Theta(n\log n)$$

求 解 实 例

例14 $T(n) = 2T(n/2) + O(n\log n)$



$$T(n) = n\log n + n(\log n - 1) + n(\log n - 2) + \ldots + n(\log n - \log n)$$

$$= O(n\log^2 n)$$

关于递推方程中⌊x⌋和⌈x⌉的处理

先猜想解，然后用数学归纳法证明

例15 估计以下递推关系的阶

$$T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + n$$

$$T(1) = 1$$

根据 $T(n) = 2T(\frac{n}{2}) + n$

$$T(n) = O(n \log n)$$

猜想原递推方程的解的阶是$O(n \log n)$
证明：$T(n) \le cn \log n$, 用数学归纳法

归纳基础

对于$T(1)=1$，显然没有$T(1)\le c\ 1\ \log 1$.

考虑$T(2)$

$T(2) = 2T(\lfloor 1 \rfloor)+2 = 4 \le 2\times 2\ \log 2$，$c=2$

归纳步骤

假设对于小于n的正整数命题为真，那么

$$T(n) = 2T(\left\lfloor \frac{n}{2} \right\rfloor) + n \le 2c\left\lfloor \frac{n}{2} \right\rfloor \log(\left\lfloor \frac{n}{2} \right\rfloor) + n$$

$$\le 2c\frac{n}{2}(\log n - \log 2) + n = c\,n\log n - c\,n + n$$

$$\le c\,n\log n, \qquad c = 2$$

顺序算法的设计技术

分治策略

动态规划算法

回溯法与分支估界

贪心算法

概率算法

分治策略

（Divide and Conquer）

分治策略的基本思想

实例、主要思想、算法描述、注意问题

递归算法与递推方程

两类递推方程的求解

降低递归算法复杂性的途径

代数变换减少子问题个数

预处理减少递归的操作

典型实例分析

分治策略的基本思想

分治策略的实例----二分检索、归并排序

主要思想----划分、求解子问题、综合解

算法描述

Divide-and-Conquer(P)

1. if $|P| \le c$ **then** $S(P)$.

2. divide P **into** $P_1, P_2, \ldots, P_k$.

3. for $i = 1$ **to** k

4. y_i **= Divide-and-Conquer(**P_i**)**

5. Return Merge($y_1, y_2, \ldots, y_k$**)**

注意问题----连续划分　平衡原则

划分均匀效率较高

例1 排序算法的比较

插入排序----子问题不均衡

$W(n) = W(n-1) + n - 1$

$W(1) = 0$

解得 $W(n) = O(n^2)$.

归并排序----子问题均衡

不妨设$n = 2^k$.

$W(n) = 2W(n/2) + n-1$

$W(1) = 0$

解得 $W(n) = O(n\log n)$.

递归算法与递推方程

分治策略的算法分析工具----递推方程

两类递推方程

$$f(n)=\sum_{i=1}^{k}a_i f(n-i)+g(n)$$

$$f(n)=af(\frac{n}{b})+d(n)$$

求解方法

第一类递推方程：公式法

第二类递推方程：迭代法、递归树、Master定理

$$f(n) = af(\frac{n}{b}) + d(n)$$

当$d(n)$为常数时

$$f(n) = \begin{cases} O(n^{\log_b a}) & a \neq 1 \\ O(\log n) & a = 1 \end{cases}$$

当$d(n) = cn$时

$$f(n) = \begin{cases} O(n) & a < b \\ O(n\log n) & a = b \\ O(n^{\log_b a}) & a > b \end{cases}$$

例2　芯片测试

A报告	B报告	结论
B是好的	A是好的	A,B都好或A,B都坏
B是好的	A是坏的	至少一片是坏的
B是坏的	A是好的	至少一片是坏的
B是坏的	A是坏的	至少一片是坏的

条件：有$n = 2^k$块芯片，(好芯片至少比坏芯片多1片), 从中挑出一片好芯片

问题：说明测试算法，进行复杂性分析

算法1

1．$k \leftarrow n$;

2．while $k>3$ do

3．　　将芯片分成$\lfloor k/2 \rfloor$组；

4．　　for i =1 to $\lfloor k/2 \rfloor$ do

　　if 2片好，则任取1片留下；

　　else 2片同时丢掉；

5．　　k ←剩下的芯片数;

6．if k =3

　　then 任取2片芯片测试；

　　if 1好1坏，取没测的芯片；

　　else 任取1片被测芯片；

7．if k =2 or 1 then 任取1片

说明：

上述算法只是一个概要说明，对于n为奇数的情况需要进一步处理, 处理时间为$O(n)$.

复杂性分析：

设$W(n)$表示n片芯片测试的次数，则

$$W(n) = W(n/2) + O(n)$$

$$W(1) = 0$$

由Master定理，$W(n) = O(n)$

例3 求一个数的幂

问题：计算 a^n, n为自然数

传统算法: $\Theta(n)$.

分治：

$$a^n = \begin{cases} a^{n/2} \times a^{n/2} & n \text{ 为偶数} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a & n \text{ 为奇数} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\log n) .$$

计算Fibonacci 数

定义

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

0 1 1 2 3 5 8 13 21 34 …

通常算法：从$F_0, F_1, \ldots,$ 根据定义陆续计算

时间为$\Theta(n)$

利用数幂乘法的分治算法

定理 .

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

算法 $T(n) = \Theta(\log n)$.

.

定理归纳证明

Base $n = 1$

$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 .$$

Inductive step ($n \geq 2$):

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdot$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

降低递归算法复杂性的途径

1．代数变换　减少子问题个数

例4 位乘问题

设X ,Y是两个n位二进制数， $n = 2^k$, 求XY.

传统算法 $W(n)=O(n^2)$

分治法 令$X= A2^{n/2} +B$, $Y= C2^{n/2} +D$.

$$XY= AC\ 2^n + (AD + BC)\ 2^{n/2} + BD$$

$$W(n) = 4W(n/2) + cn,$$

$$W(1) = 1$$

解得　$W(n) = O(n^{\log 4}) = O(n^2)$

代数变换

$$AD + BC = (A - B)(D - C) + AC + BD$$

递推方程

$$W(n) = 3\,W(n/2) + cn$$

$$W(1) = 1$$

解

$$W(n) = O(n^{\log 3}) = O(n^{1.59})$$

例5　Strassen 矩阵乘法

A,B 为两个n阶矩阵，$n=2^k$, 计算$C=AB$.

传统算法 $W(n)=O(n^3)$

分治法　将矩阵分块，得

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}=\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

其中

$$C_{11}=A_{11}B_{11}+A_{12}B_{21}$$
$$C_{12}=A_{11}B_{12}+A_{12}B_{22}$$
$$C_{21}=A_{21}B_{11}+A_{22}B_{21}$$
$$C_{22}=A_{21}B_{12}+A_{22}B_{22}$$

递推方程 $W(n)=8\,W(n/2)+cn^2$

$W(1)=1$

解　$W(n)=O(n^3)$.

分治变换方法

$M_1 = A_{11}(B_{12} - B_{22})$　　$M_2 = (A_{11} + A_{12})B_{22}$

$M_3 = (A_{21} + A_{22})B_{11}$　　$M_4 = A_{22}(B_{21} - B_{11})$

$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$

$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$

$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$

$C_{11} = M_5 + M_4 - M_2 + M_6$　　$C_{12} = M_1 + M_2$

$C_{21} = M_3 + M_4$　　$C_{22} = M_5 + M_1 - M_3 - M_7$

由Master定理得

$$W(n) = 7W\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$

$$W(1) = 1$$

$$W(n) = O(n^{\log_2 7}) = O(n^{2.8075})$$

2．预处理

递归算法中的处理步尽可能提到递归外，作为预处理

例6　平面点对问题

输入：集合S中有n个点，$n>1$，

输出：所有的点对之间的最小距离.

通常算法：$C(n,2)$个点对计算距离，比较最小，需$O(n^2)$时间

分治策略：取S 的子集P，将P中的点划分成两个子集P_L和P_R

$$|P_L|=\left\lceil \frac{|P|}{2} \right\rceil \qquad |P_R|=\left\lfloor \frac{|P|}{2} \right\rfloor$$

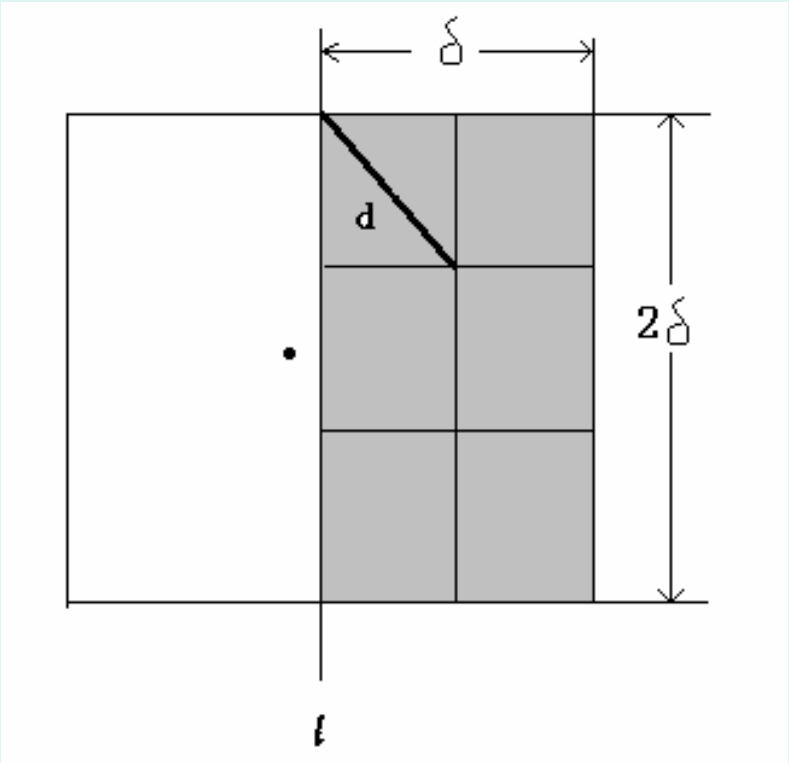
算法2　平面最近点对算法

MinDistance(P,X,Y)

输入：n个点的点集P,X是横坐标的排序数组，Y是纵坐标的排序数组

输出：最近的两个点及距离。

1. 如果P中点数小于等于3，则直接计算其中的最小距离；
2. 排序X,Y;
3. 做垂直线l将P划分为P_L和P_R，P_L的点在l左边，P_R的点在l右边
4. **MinDidtance(P_L,X_L,Y_L)**; $\delta_L = P_L$中的最小距离；
5. **MinDistance(P_R,X_R,Y_R)**; $\delta_R = P_R$中的最小距离；
6. $\delta = \min(\delta_L,\delta_R)$;
7. 对于在垂直线两边距离δ范围内的每个点，检查是否有点与它的距离小于δ，如果存在则将δ修改为新值.



$$d = \sqrt{(\delta/2)^2 + (2\delta/3)^2}$$

$$= \sqrt{\delta^2/4 + 4\delta^2/9} = \sqrt{25\delta^2/36} = 5\delta/6$$

每个点至多比较6个点，n个点需要$O(n)$时间

分析：步1　$O(1)$

步2　$O(n\log n)$

步3　$O(1)$

步4-5　$2T(n/2)$

步6　$O(1)$

步7　$O(n)$

$$T(n) = 2T(\frac{n}{2}) + O(n\log n)$$

$$T(n) = O(1) \qquad n \le 3$$

由递归树估计$T(n) = O(n\log^2 n)$.

预排序的处理方法

在每次调用时将已经排好的数组分成两个排序的子集，每次调用这个过程的时间为$O(n)$

$$W(n) = T(n) + O(n\log n)$$

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

$$T(n) = O(1) \qquad n \le 3$$

解得

$$T(n)=O(n\log n)$$

$$W(n) = O(n\log n)$$

典型实例分析

例7　快速排序

算法3　快速排序 Quicksort

输入：数组$A[p..r]$

输出：排好序的数组A

Quicksort(A,p,r)

1. **if** $p < r$
2. **then** $q \leftarrow$**Partition**(A, p, r)
3. $A[p] \leftrightarrow A[q]$
4. **Quicksort**(A,p,q-1)
5. **Quicksort**(A,q+1,r)

```
Partition(A,p,r)
 1.  x ← A[p]
 2.  i ← p
 3.  j ← r+1
 4.  while true do
 5.      repeat j ← j –1
 6.      until A[ j ] ≤ x
 7.      repeat i ← i +1
 8.      until A[ i ] ≥ x
 9.      if i < j
10.          then A[ i ] ↔ A[ j ]
11.          else return j
```

时间复杂性

最坏情况

$$W(n) = W(n-1) + n - 1$$

$$W(1) = 0$$

$$W(n) = \frac{1}{2} n(n-1) = \Theta(n^2)$$

最好划分

$$T(n) = 2T(\frac{n}{2}) + n - 1$$

$$T(1) = 0$$

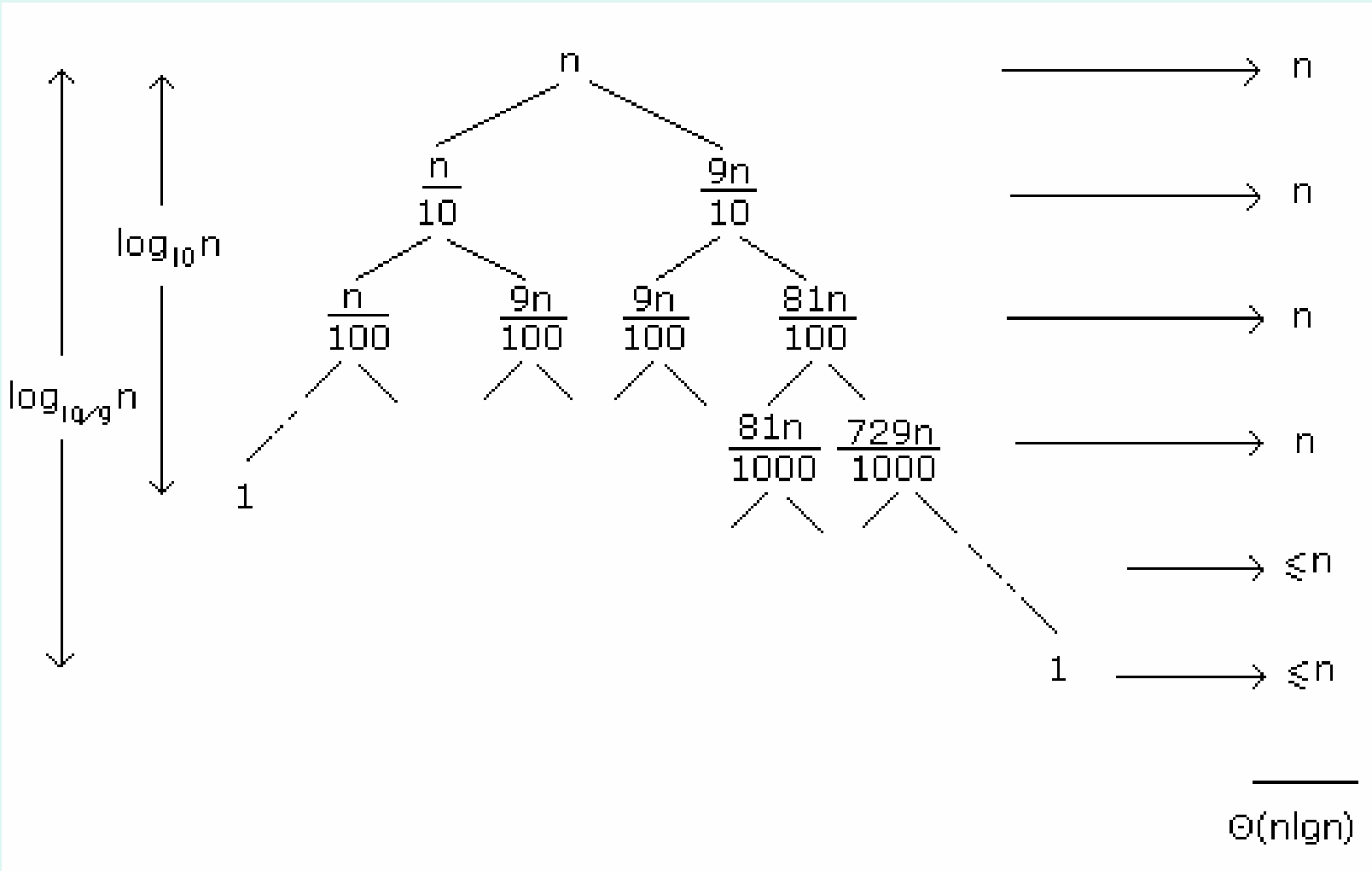
$$T(n) = \Theta(n \log n)$$

均衡划分

$$T(n) = T(\frac{9n}{10}) + T(\frac{n}{10}) + n$$

$$T(1) = 0$$

$$T(n) = \Theta(n \log n)$$


n
n/10
9n/10
n/100
9n/100
9n/100
81n/100
81n/1000
729n/1000
1
1
n
n
n
n
≤n
≤n
log10n
log10/9n
Θ(nlgn)

平均情况

$$T(n) = \frac{1}{n}\sum_{k=1}^{n-1}(T(k) + T(n-k)) + n - 1$$

$$nT(n) = 2\sum_{k=1}^{n-1} T(k) + n(n-1)$$

$$(n-1)T(n-1) = 2\sum_{k=1}^{n-2} T(k) + (n-1)(n-2)$$

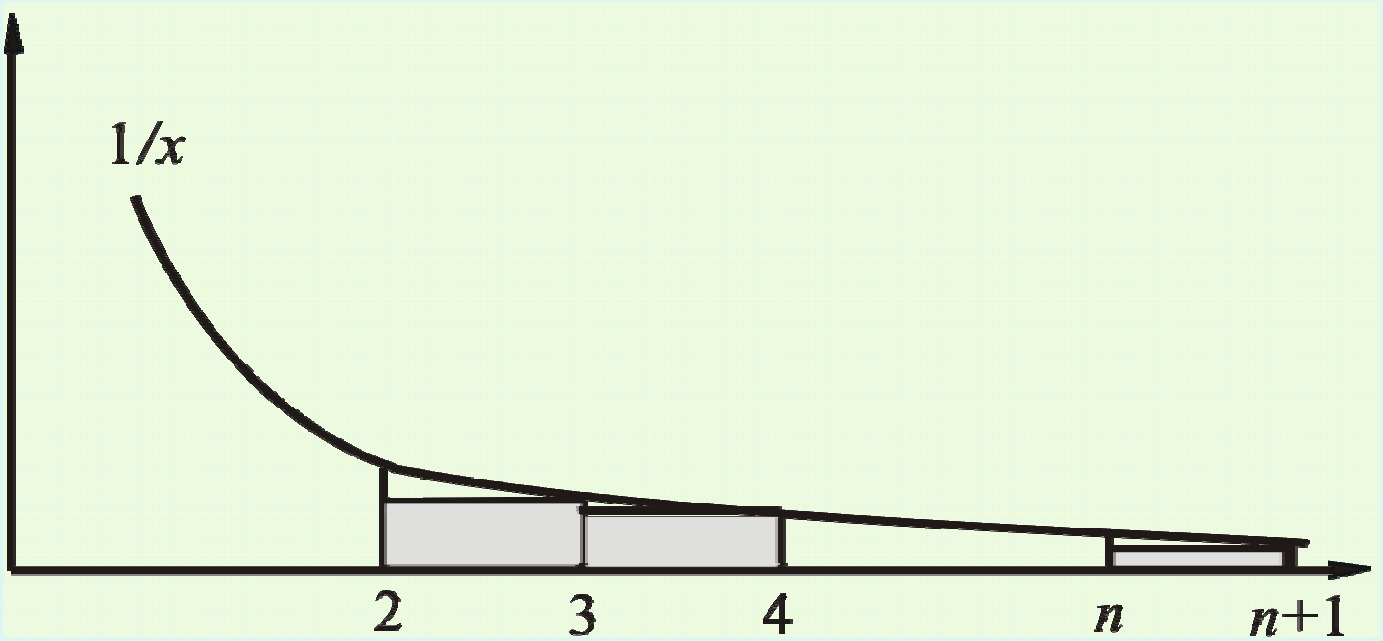
$$nT(n) = (n+1)T(n-1) + 2n - 2$$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2}{n+1} - \frac{2}{n(n+1)}$$

$$= 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right) - O(1) \le 2\int_2^{n+1} \frac{1}{x}dx$$

$$= O(\log n)$$

用积分作为求和的上界



例8．元素选择问题

从给定的集合L中选择第 i 小的元素

不妨设L为n个不等的实数

i=1, 称为最小元素；

i=n，称为最大元素；

位置处在中间的元素，称为中位元素

当n为奇数时，中位数只有1个，i=(n+1)/2；

当n为偶数时，中位数有2个，i=n/2, n/2+1.

1．选最大

输入：*n*个不等的数

输出：*max*

算法4　Findmax

1. *max*←*L*[1]；
2. for *i*←2 to *lenth*[*L*]
   do if *max* <*L*[*i*]
   then *max*←*L*[*i*]
3. return *max*.

算法最坏情况下的时间复杂性为$O(n)$

2．找最大和最小

通常算法：顺序比较

复杂性：$W(n)=2n-3$

算法5　FindMaxMin

1．将n个元素两两一组分成$\lfloor n/2 \rfloor$组；

2．每组比较，得到$\lfloor n/2 \rfloor$个较小和$\lfloor n/2 \rfloor$个较大；

3．在$\lceil n/2 \rceil$个(n为奇数，是$\lfloor n/2 \rfloor+1$)较小中找最小min；

4．在$\lceil n/2 \rceil$个(n为奇数，是$\lfloor n/2 \rfloor+1$)较大中找最大max.

复杂性：行2 比较$\lfloor n/2 \rfloor$次，行3--4比较至多$2\lceil n/2 \rceil-2$次，

$$W(n)=\lfloor n/2 \rfloor+2\lceil n/2 \rceil-2=n+\lceil n/2 \rceil-2=\lceil 3n/2 \rceil-2$$

3．找第二大

通常算法：顺序比较

1．顺序比较找到最大*max*;

2．从剩下的$n-1$个数中找最大，就是第二大

second

复杂性：$W(n)=n-1+n-2=2n-3$

算法6：锦标赛方法

1．$k \leftarrow n$;

2．将k个元素两两一组，分成$\lfloor k/2 \rfloor$组；

3．每组的2个数比较，找到较大的数；

4．将被淘汰的较小的数在淘汰它的数所指向的链表中做记录；

5．if k为奇数 then $k \leftarrow \lfloor k/2 \rfloor + 1$;

6．　else $k \leftarrow \lfloor k/2 \rfloor$;

7．if $k>1$ then goto 2;

8．max←最大数；

9．$second$←max的链表中的最大

复杂性：

$$W(n)=n-1+\lceil \log n \rceil -1= n+\lceil \log n \rceil -2$$

4．一般性选择问题

输入：数组L, L的长度n, 正整数k, $1 \le k \le n$.

输出：第k小的数

通常算法

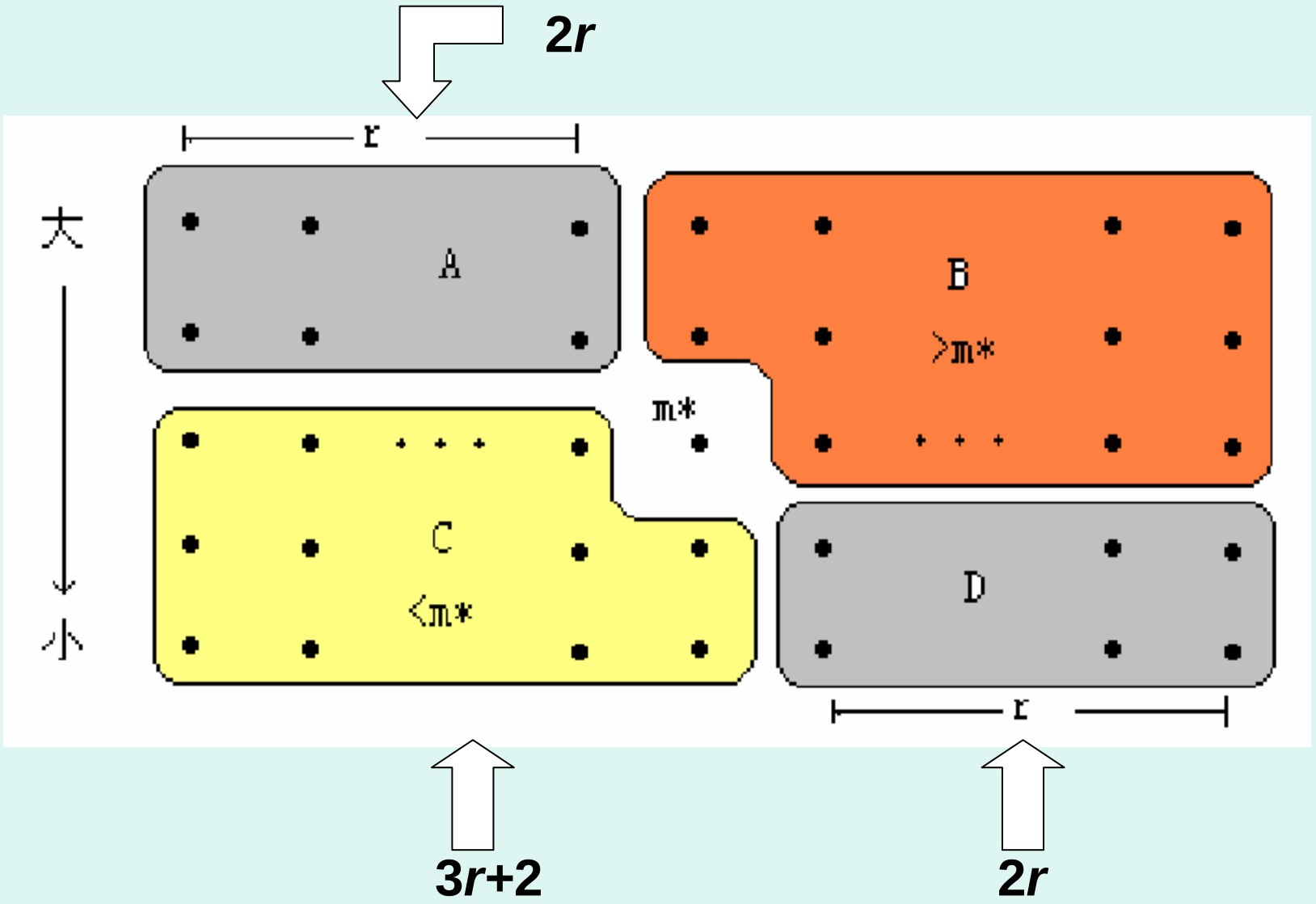
1．排序

2．找第k小的数

时间复杂性：$O(n\log n)$

算法7 Select(S,k)

1. 将S划分成5个一组，共$n_M=\lceil n/5\rceil$个组
2. 每组找中位数，n_M个中位数放到集合M.
3. $m^*\leftarrow$Select($M,\lceil |M|/2\rceil$) 将S中的数划分成A,B,C,D四个集合
4. 把A和D中的每个元素与m^*比较，小的构成S_1, 大的构成S_2;
5. $S_1\leftarrow S_1\cup C$; $S_2\leftarrow S_2\cup B$;
6. if $k=|S_1|+1$ then 输出m^*
7. else if $k\le|S_1|$
8. then Select(S_1,k)
9. else Select($S_2,k-|S_1|-1$)



最坏情况：子问题大小为 $2r+2r+3r+2=7r+2$

复杂性估计

复杂性：$W(n)=O(n)$

不妨设 $n=5(2r+1)$, $|A|=|D|=2r$,

$$r=\frac{\frac{n}{5}-1}{2}=\frac{n}{10}-\frac{1}{2}$$

行2： $O(n)$;

行3： $W(n/5)$;

行4： $O(n)$;

行8-9：

复杂性估计（续）

$$W(7r+2)=W(7(\frac{n}{10}-\frac{1}{2})+2)$$

$$=W(\frac{7n}{10}-\frac{3}{2})\le W(\frac{7n}{10})$$

$$w(n)\le W(\frac{n}{5})+W(\frac{7n}{10})+cn$$

$$\le cn+\frac{9}{10}cn+\frac{81}{100}cn+\ldots=O(n)$$

例9 多项式求值

多项式：$A(x)=a_0+a_1x+a_2x^2+...+a_{n-1}x^{n-1}$

设x为1 的$2n$次方根，对所有的 x 计算 $A(x)$ 的值.

算法1：对每个 x做下述运算：

依次计算每个项a_ix^i，对i 求和得到$A(x)$，

$T_1(n)=O(n^3)$

算法2：$A_1=a_{n-1}$

$A_2=a_{n-2}+A_1x$

$A_3=a_{n-3}+A_2x$

…

$A_n=a_0+A_{n-1}x$

$T_2(n)=O(n^2)$

算法3：分治策略

原理： $A_{even}(x)=a_0+a_2x^2+a_4x^2+\ldots a_{n-2}x^{(n-2)/2}$

$A_{old}(x)=a_1+a_3x^1+a_5x^2+\ldots a_{n-1}x^{(n-2)/2}$

$A(x)=A_{even}(x^2)+xA_{old}(x^2)$， x^2为1的n次根

算法：
1. 计算1 的所有的 n 次根
2. 分别计算$A_{even}(x^2)$与$A_{old}(x^2)$
3. 利用步2 的结果计算$A(x)$

复杂度分析： $T_3(n)=2T(n/2)+O(n)$

$T_3(n)=O(n\log n)$

应用：快速傅立叶变换FFT

动态规划

(Dynamic Programming)

基本思想和使用条件

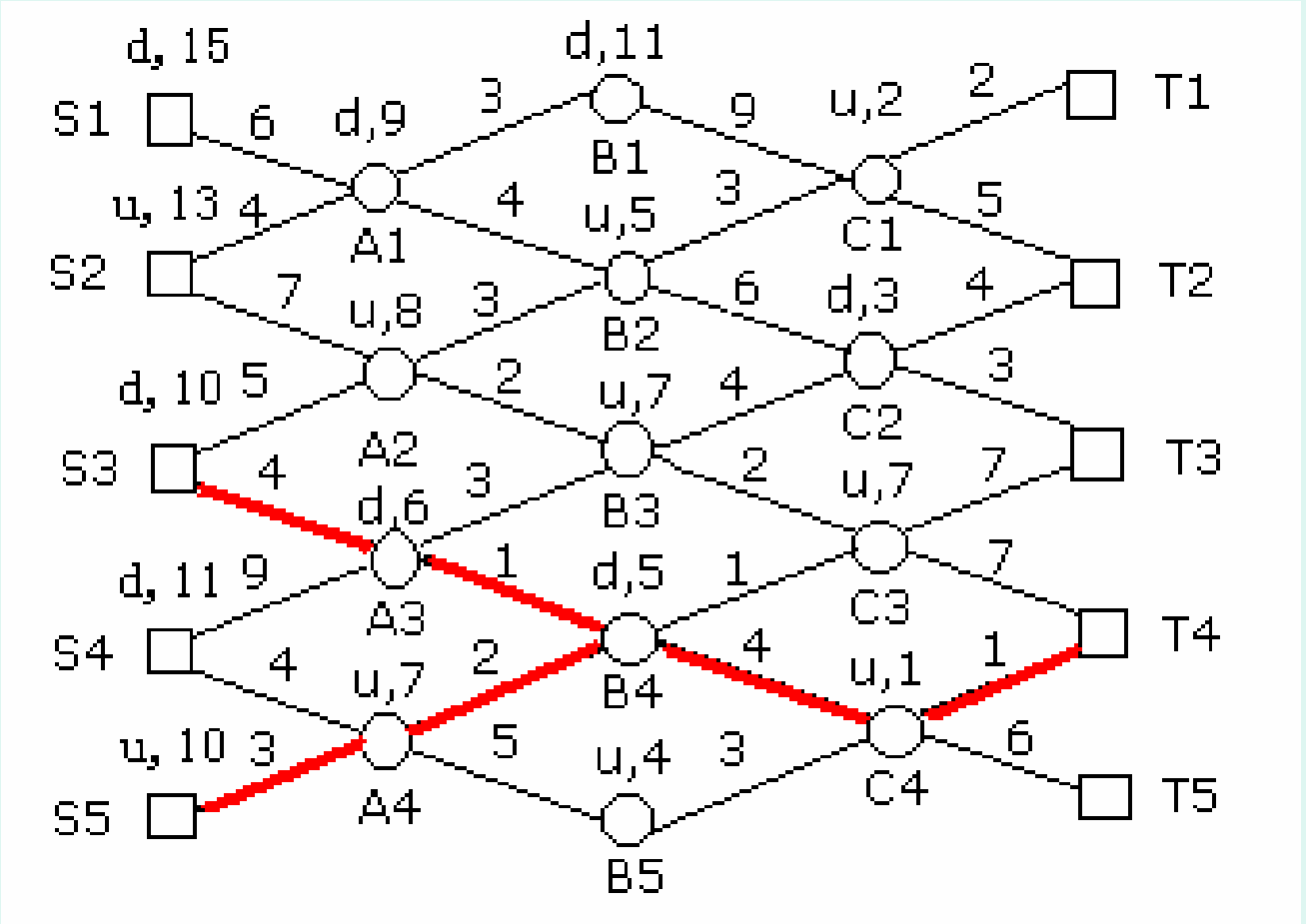
动态规划算法的设计步骤

应用实例

小结

基本思想和使用条件

例1 求从始点到终点的最短路径



基 本 思 想

解：判断序列

$$F(C_l) = \min_m \{C_l T_m)$$

$$F(B_k) = \min_l \{B_k C_l + F(C_l)\}$$

$$F(A_j) = \min_k \{A_j B_k + F(B_k)\}$$

$$F(S_i) = \min_j \{S_i A_j + F(A_j)\}$$

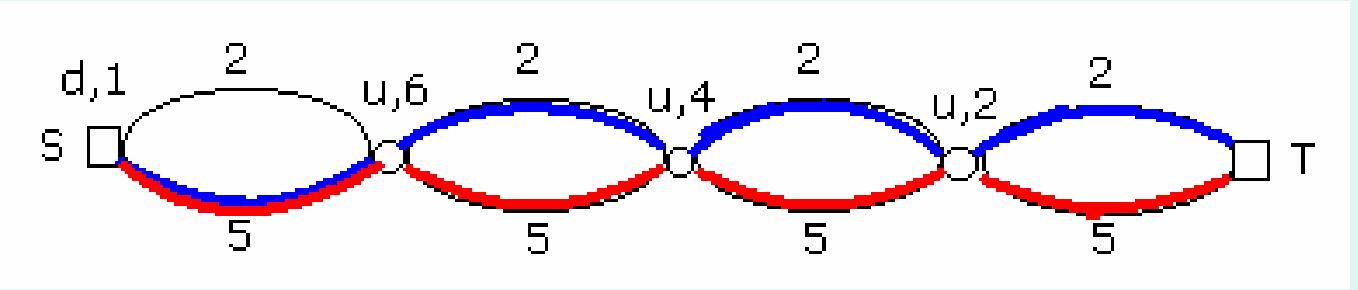
任何最短路径的子路径都是相对于子路径的始点和终点的最短路径

为找到一条最短路径只需从T_j开始进行多步判断

使用条件——优化原则

一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优的决策序列

例2　求总长模10的最小路径



最优解：下、下、下、下

动态规划求解：下、上、上、上

不满足优化原则，不能使用动态规划设计技术

算法设计步骤

例3 矩阵乘法：设$A_1,A_2,\ldots,A_n$为矩阵序列，A_i为$P_{i-1}\times P_i$阶矩阵，$i = 1,2,\ldots,n$. 确定乘法顺序使得元素相乘的总次数最少.

输入：向量$P = <P_0, P_1, \ldots, P_n>$

实例： $P = <10, 100, 5, 50>$

A_1: 10×100, A_2: 100×5, A_3: 5×50,

乘法次序

$(A_1\ A_2)A_3$: $10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$

$A_1(A_2\ A_3)$: $10 \times 100 \times 50 + 100 \times 5 \times 50 = 75000$

一般算法：加括号的方法有 $\frac{1}{n+1}\binom{2n}{n}$ 种，Catalan数

递 推 方 程

输入为$P=<P_0, P_1, \ldots, P_n>$，$A_{i..j}$ 表示乘积$A_iA_{i+1}\ldots A_j$ 的结果，其最后一次相乘是

$$A_{i..j} = A_{i..k}\, A_{k+1..j},$$

$m[i,j]$ 表示得到$A_{i..j}$的最少的相乘次数

递推方程

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + P_{i-1}P_kP_j\} & i < j \end{cases}$$

为了确定加括号的次序，设计表$s[i,j]$,记录求得最优时最后一次运算的位置

算法1：递归算法

算法1 RecurMatrixChain(P,i,j)

1. $m[i,j]\leftarrow\infty$
2. $S[i,j]\leftarrow i$
3. for $k\leftarrow i$ to $j-1$ do
4. $q\leftarrow$ RecurMatrixChain(P,i,k)
   $+$RecurMatrixChain$(P,k+1,j)+p_{i-1}p_kp_j$
5. if $q<m[i,j]$
6. then $m[i,j]\leftarrow q$
7. $S[i,j]\leftarrow k$
8. Return $m[i,j]$

递归算法复杂性

复杂性满足递推关系

$$T(n) \geq \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1} [T(k) + T(n-k) + O(1)] & n > 1 \end{cases}$$

$$T(n) \geq O(n) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k)$$

$$= O(n) + 2\sum_{k=1}^{n-1} T(k)$$

递归算法复杂性

复杂性满足递推关系

$$T(n) = O(n) + 2\sum_{k=1}^{n-1} T(k), \quad T(1) = O(1)$$

数学归纳法证明 $T(n) \geq 2^{n-1}$

n=1, 显然为真

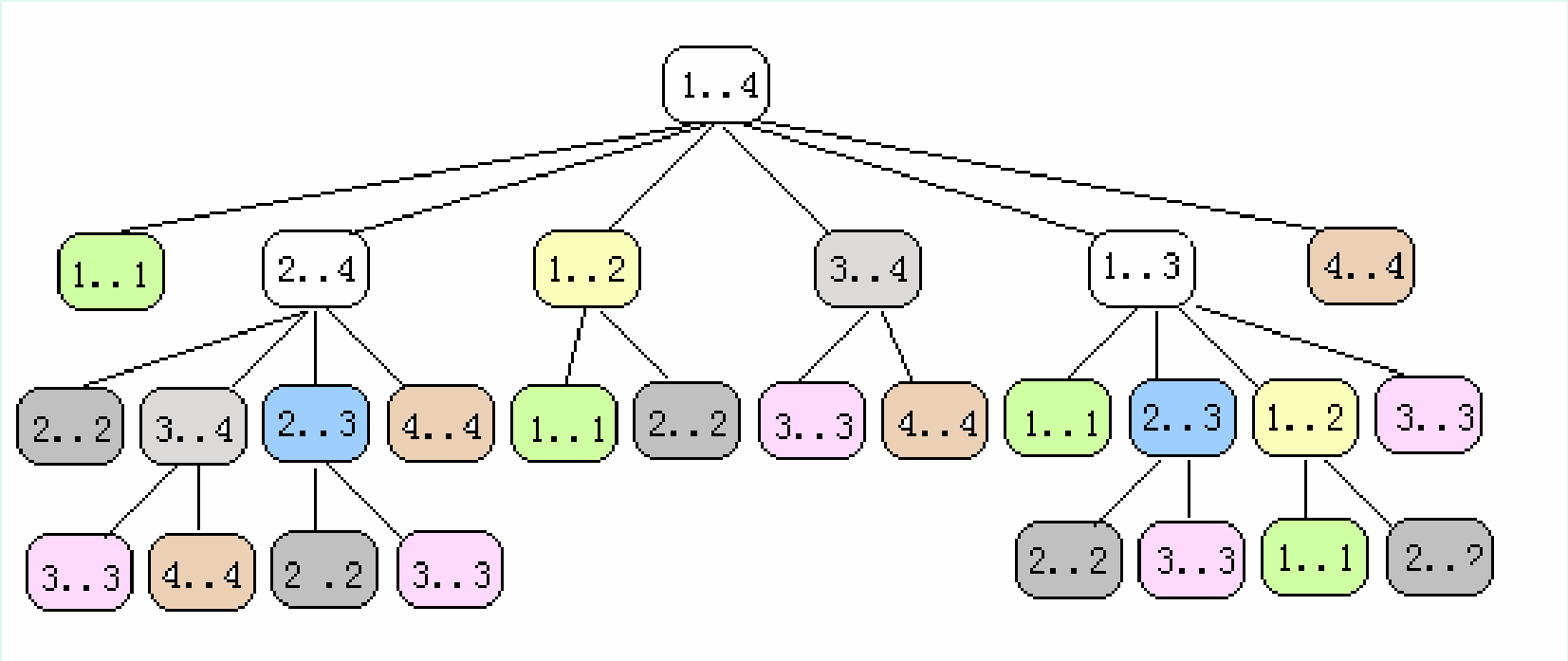
假设对于任何小于n的k命题为真，则

$$T(n) \geq O(n) + 2\sum_{k=1}^{n-1} T(k) \geq O(n) + 2\sum_{k=1}^{n-1} 2^{k-1}$$

$$= O(n) + 2(2^{n-1} - 1) \geq 2^{n-1}$$

复杂性高的原因

原因：子问题重复程度高



算法2：非递归算法

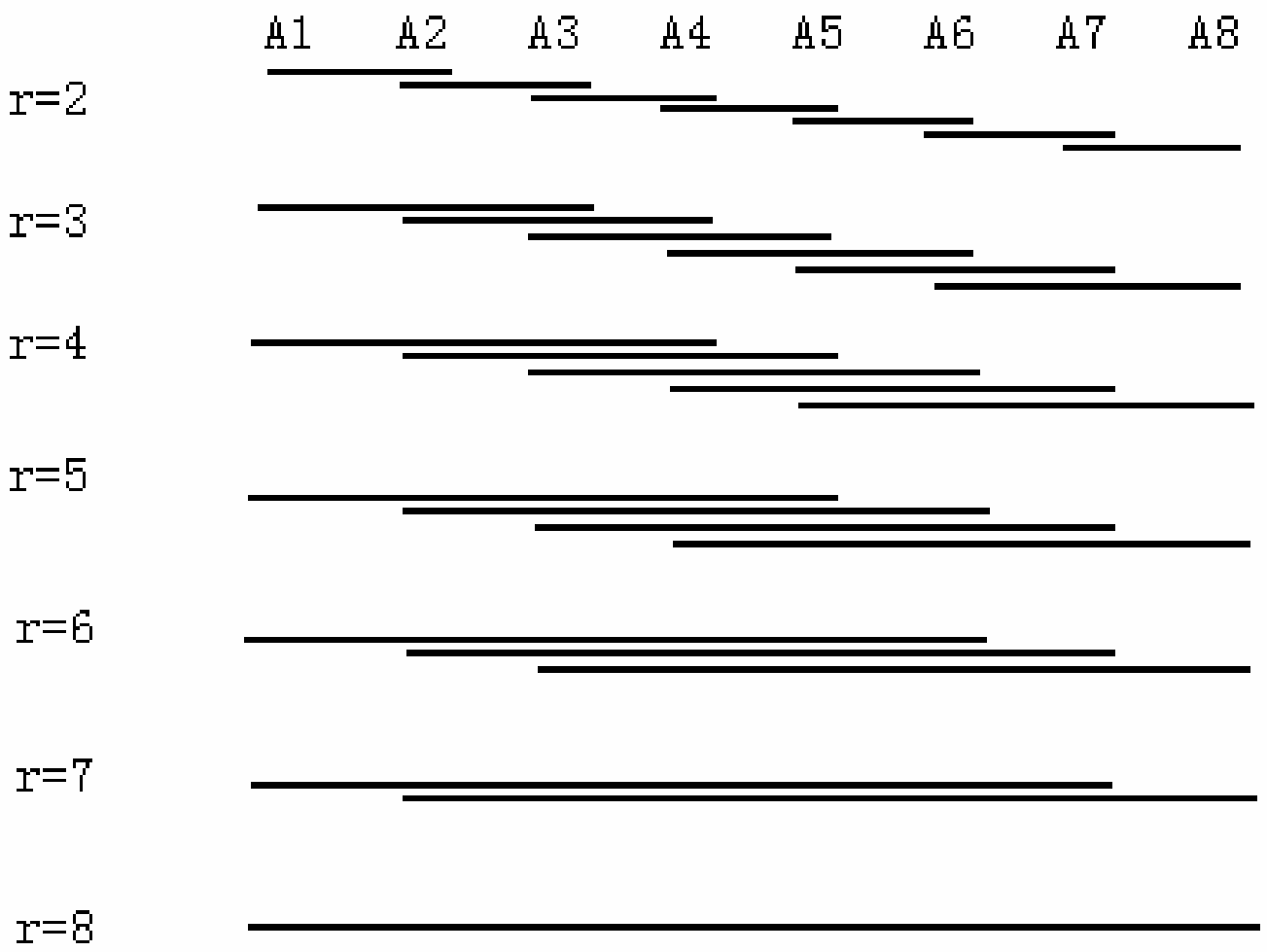
算法2　MatrixChain(P,n)

1. 令所有的$m[i,j]$初值为0
2. for $r\leftarrow 2$ to n do　　// r为计算的矩阵链长度//
3. for $i\leftarrow 1$ to $n-r+1$ do　　//$n-r+1$为最后r链的始位置//
4. $j\leftarrow i+r-1$　　// 计算链i—j //
5. $m[i,j]\leftarrow m[i+1,j]+p_{i-1}*p_i*p_j$　　// $A_i(A_{i+1}..A_j)$//
6. $s[i,j]\leftarrow i$　　//记录分割位置//
7. for $k\leftarrow i+1$ to $j-1$ do
8. $t\leftarrow m[i,k]+m[k+1,j]+p_{i-1}*p_k*p_j$　　//$(A_i..A_k)(A_{k+1}..A_j)$//
9. if $t<m[i,j]$
10. then $m[i,j]\leftarrow t$
11. $s[i,j]\leftarrow k$

复杂性：行2,3,7循环进行都是$O(n)$，循环内为$O(1)$，

$W(n)=O(n^3)$

8个矩阵$A_1A_2\cdots A_8$相乘的子问题计算



两种算法的比较

递归算法：复杂性高

非递归算法：复杂性较低

原因

递归动态规划算法的子问题被多次重复计算

子问题计算次数呈指数增长

非递归动态规划算法每个子问题只计算一次

子问题的计算随问题规模成多项式增长

动态规划算法设计步骤

将问题表示成多步判断

整个判断序列就对应问题的最优解

每步判断对应一个子问题，子问题类型与原问题一样

确定优化函数，以函数的极大(或极小)作为判断的依据

确定是否满足优化原则-----动态规划算法的必要条件

确定子问题的重叠性

列出关于优化函数的递推方程(或不等式)和边界条件

自底向上计算子问题的优化函数值

备忘录方法(表格)存储中间结果

设立标记函数$s[i,j]$求解问题的解

应 用 实 例

例4　投资问题

m元钱，n项投资，$f_i(x)$: 将x元投入第i项项目的效益

目标函数 $\max\{f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)\}$

约束条件 $x_1 + x_2 + \ldots + x_n = m$，$x_i \in N$

实例：5万元钱，4个项目，效益函数如下表所示

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

算 法 设 计

设$F_k(x)$ 表示x元钱投给前k个项目的最大效益

多步判断

假设知道p元钱（$p \le x$）投给前$k-1$个项目的最大效益，决定x元钱投给前k个项目的分配方案

递推方程和边界条件

$$F_k(x) = \max_{0 \le x_k \le x} \{ f_k(x_k) + F_{k-1}(x - x_k) \}$$

$$F_1(x) = f_1(x)$$

备忘录存储与解

x	$F_1(x)$	$x_1(x)$	$F_2(x)$	$x_2(x)$	$F_3(x)$	$x_3(x)$	$F_4(x)$	$x_4(x)$
1	11	1	11	0	11	0	20	1
2	12	2	12	0	13	1	31	1
3	13	3	16	2	30	3	33	1
4	14	4	21	3	41	3	50	1
5	15	5	26	4	43	4	61	1

解 $x_1=1,\ x_2=0,\ x_3=3,\ x_4=1\quad F_4(5)=61$

算法的复杂度分析

表中有m行n列, 共计mn项

$$F_k(x) = \max_{0 \le x_k \le x} \{ f_k(x_k) + F_{k-1}(x - x_k) \}$$

$$F_1(x) = f_1(x)$$

对第$F_k(x)$项 ($2 \le k \le n, 1 \le x \le m$) 需要$x+1$次加法, x次比较

加法次数 $$\sum_{k=2}^{n} \sum_{x=1}^{m} (x+1) = \frac{1}{2}(n-1)m(m+3)$$

比较次数 $$\sum_{k=2}^{n} \sum_{x=1}^{m} x = \frac{1}{2}(n-1)m(m+1)$$

$$W(n) = O(nm^2)$$

例5 背包问题 (Knapsack Problem)

一个旅行者准备随身携带一个背包. 可以放入背包的物品有n种, 每种物品的重量和价值分别为w_j, v_j. 如果背包的最大重量限制是b, 怎样选择放入背包的物品以使得背包的价值最大?

目标函数 $$\max \sum_{j=1}^{n} v_j x_j$$

约束条件 $$\sum_{j=1}^{n} w_j x_j \le b,$$

$$x_j \in N$$

线性规划问题

由线性条件约束的线性函数取最大或最小的问题

整数规划问题 线性规划问题的变量x_j都是非负整数

算 法 设 计

设$F_k(y)$表示只允许装前k种物品,背包总重不超过y时背包的最大价值

递推方程与边界条件

$$F_k(y)=\max\{F_{k-1}(y),F_k(y-w_k)+v_k\}$$

$$F_0(y)=0,\quad 0\le y\le b$$

$$F_k(0)=0,\quad 0\le k\le n$$

$$F_1(y)=\left\lfloor\frac{y}{w_1}\right\rfloor v_1$$

$$F_k(y)=-\infty\quad y<0$$

实 例

$v_1 = 1,\ \ v_2 = 3,\ \ v_3 = 5,\ v_4 = 9,$

$w_1 = 2,\ \ w_2 = 3,\ w_3 = 4,\ w_4 = 7,$

$b = 10$

则 $F_k(y)$ 的计算表如下:

k \ y	1	2	3	4	5	6	7	8	9	10
1	0	1	1	2	2	3	3	4	4	5
2	0	1	3	3	4	6	6	7	9	9
3	0	1	3	5	5	6	8	10	10	11
4	0	1	3	5	5	6	9	10	10	12

标记函数的计算

关于标记函数的递推方程

$$i(k,y)=\begin{cases} i(k-1,y) & F_{k-1}(y)>F_k(y-W_k)+V_k \\ k & F_{k-1}(y)\le F_k(y-W_k)+V_k \end{cases}$$

k \ y	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

标记函数确定问题的解

k \ y	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

在上例中, 求得

$i(4,10)=4 \Rightarrow x_4 \geq 1$

$i(4,10-w_4)=i(4,3)=2 \Rightarrow x_2 \geq 1,\ x_4=1,\ x_3=0$

$i(4,3-w_2)=i(4,0)=0 \Rightarrow x_2 = 1,\ x_1=0$

解 $x_1=0,\ x_2=1,\ x_3=0,\ x_4=1$

例6 最长公共子序列LCS

相关概念

*X*的子序列*Z* ：设序列*X*,*Z*,

$$X =< x_1, x_2, ..., x_m >$$

$$Z =< z_1, z_2, ..., z_k >$$

若存在*X*的元素构成的严格递增序列 $< x_{i_1}, x_{i_2}, ..., x_{i_k} >$ 使得 $z_j = x_{i_j}, j = 1,2,...,k$， 则称*Z*是*X*的子序列

*X*与*Y*的公共子序列*Z* ：*Z*是*X*和*Y*的子序列

子序列的长度：子序列的元素个数

问 题

给定序列

$$X=<x_1,x_2,\ldots,x_m>,\ Y=<y_1,y_2,\ldots,y_n>$$

求X和Y的最长公共子序列

X: A B C B D A B

Y: B D C A B A

蛮力算法：检查X的每个子序列在Y中是否出现用 $O(n)$ 时间，X有2^m个 子序列，最坏情况下时间复杂度：$O(n2^m)$

动态规划算法

递推方程

令 X与Y 的子序列

$$X_i=<x_1,x_2,\ldots,x_i>,\quad Y_j=<y_1,y_2,\ldots,y_j>$$

$C[i,j]$　X_i 与 Y_j 的 LCS 的长度

递推方程

$$C[i,j]=\begin{cases}0 & \text{如果} i=0 \text{或} j=0\\ C[i-1,j-1]+1 & \text{如果} i,j>0, x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{如果} i,j>0, x_i\neq y_j\end{cases}$$

算　　法

算法 LCS_ length(*X*,*Y*)

```
1. for i←1 to m do                    //行1-4边界情况//
2.    C[i,0]←0
3. for i←1 to n do
4.    C[0,i]←0
5.  for i←1 to m do
6.     for j←1 to n do
7.      if X[i]=Y[j]
8.          then C[i,j]←C[i–1,j–1]+1
9.               B[i,j]←'↖'
10.        else if C[i–1,j] ≥ C[i,j–1]
11.             then C[i,j]←C[i–1,j]
12.                  B[i,j]←'↑'
13.             else C[i,j]←C[i,j–1]
14.                  B[i,j]←'←'
```

复杂性　时间：　$T(m,n)=O(mn)$

　　空间：　$S(m,n)=O(mn)$

构造解

算法LCS (*B*, *i*, *j*)

1. **if *i*=0 or *j*=0 then return**
2. **if *B*[*i*,*j*]=’↖’ then**
3. **输出*X*[*i*]**
4. **LCS (*B*,*i*-1,*j*-1)**
5. **else if *B*[*i*,*j*]=’↑’ then LCS (*B*,*i*-1,*j*)**
6. **else LCS (*B*,*i*,*j*-1)**

构造复杂性

时间：$O(m+n)$，每一步至少缩小X 或 Y 的长度，
执行$m+n$ 步，每步使用常数时间

空间：$O(mn)$

例7　图像压缩

像素点灰度值　0~255，表示为8位二进制数

像素点灰度值序列 $\{p_1,p_2,\ldots,p_n\}$，p_i为第 i 个像素点的灰度值

变位压缩存储格式　将$\{p_1, p_2,\ldots, p_n\}$分割成 m 段$S_1,S_2,\ldots,S_m$.
第 i 段有$l[i]$个像素，每个像素有$b[i]$位

$$h_i \le b[i] \le 8,$$

$$h_i = \left\lceil \log(\max_{p_k \in S_i} p_k + 1) \right\rceil$$　// 用二进制表示S_i中的灰度所需位数 //

约束条件：限定每段像素个数 $l[i]$ 不超过256

表示段S_i的总位数=表示每个像素位数$b[i]$的二进制数(3 位)
+表示本段像素个数$l[i]$的二进制数(8位)
+本段每个像素的位数×本段像素个数

问题
给定像素序列$\{p_1,p_2, \ldots,p_n\}$，确定空间最小的分段方式

实　例

灰度值序列　P={10,12,15,255,1,2,1,1,2,2,1,1}

分法 1：　S_1={10,12,15}，S_2={255}, S_3={1,2,1,1,2,2,1,1}

分法 2：　S_1={10,12,15,255,1,2,1,1,2,2,1,1}

分法 3：　分成12组，每组一个数

存储空间

分法 1：11×3+4×3+8×1+2×8=69

分法 2：11×1+8×12=107

分法 3：11×12+4×3+8×1+1×5+2×3=163

算 法 设 计

递推方程

设$s[i]$是像素序列$\{p_1, p_2, \ldots, p_i\}$的最优分段所需存储位数

$$s[i] = \min_{1 \le k \le \min\{i,256\}} \{s[i-k] + k * b\max(i-k+1, i)\} + 11$$

$$b\max(i-k+1, i) = \left\lceil \log\left(\max_{i-k+1 \le j \le i} \{p_j\} + 1 \right) \right\rceil$$

算　法

```
算法Compress (P,n)                    //计算最小位数S[n]//
1. Lmax←256; header←11; S[0]←0  //header每个段附加存贮//
2. for i←1 to n do
3.    b[i]←length(P[i])    //表示第i个像素灰度的二进制位数//
4.    bmax←b[i]             //3-6行分法的最后一段只有p_i自己//
5.    S[i]←S[i–1]+bmax
6.    l[i]←1
7.    for j←2 to min{i,Lmax} do  //最后段含j个像素,j=2,…,i或256
8.       if bmax<b[i–j+1]        //统一段内表示像素的二进制位数//
9.          then bmax←b[i–j+1]
10.      if S[i]>S[i–j]+j*bmax
11.         then S[i]←S[i–j]+j*bmax
12.                 l[i]←j
13.   S[i] ←S[i]+header
```

计算复杂性 $T(n)=O(n)$

例8 最大子段和

问题：给定n个整数（可以为负数）的序列

$$(a_1, a_2, \ldots, a_n)$$

求

$$\max\{\ 0,\ \max_{1\le i\le j\le n}\sum_{k=i}^{j} a_k\ \}$$

实例： (-2, 11, -4, 13, -5, -2)

解：最大子段和 $a_2+a_3+a_4= 20$

算法1---顺序求和+比较

算法2---分治策略

算法3---动态规划

算法1 顺序求和+比较

```
1.  sum←0                          // sum为最终输出
2.  for i←1 to n do
3.     for j←i to n do
4.        thissum←0                 // thissum为a[i]到a[j]之和
5.        for k←i to j do
6.           thissum←thissum+a[k]
7.           if thissum > sum then
8.              sum←thissum
9.              besti←i             // 记录最大和的首位置
10.             bestj←j             // 记录最大和的末位置
```

复杂性：$O(n^3)$

算法2　分治策略

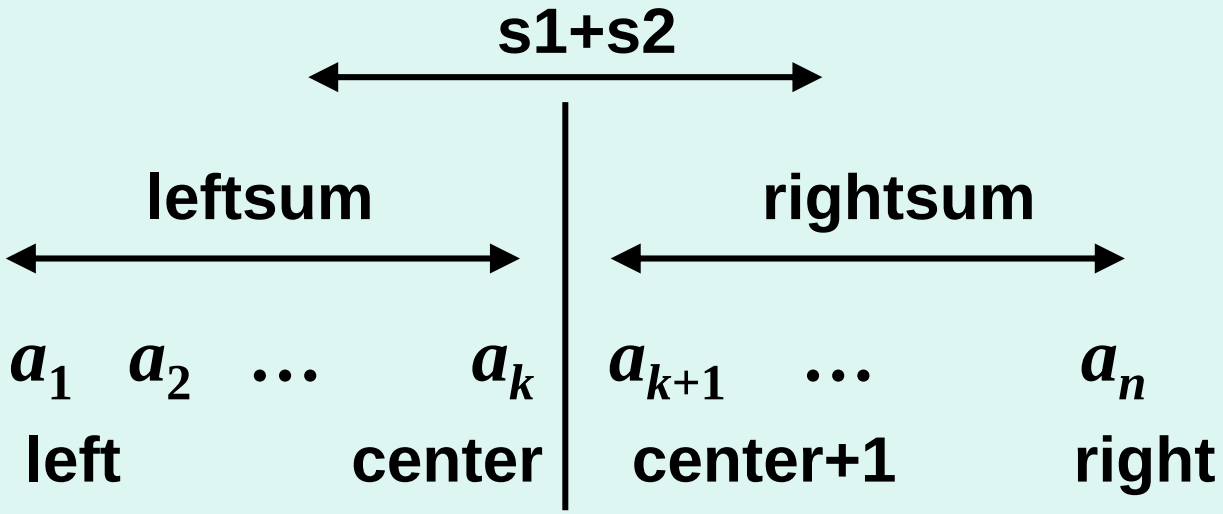
将序列分成左右两半，中间分点*center*

递归计算左段最大子段和 *leftsum*

递归计算右段最大子段和 *rightsum*

$a_{center} \rightarrow a_1$的最大和$s1$，$a_{center+1} \rightarrow a_n$的最大和$s2$

max { *leftsum*, *rightsum*, *s*1+*s*2}



算法2 MaxSubSum

算法 MaxSubSum(*A*, *left*, *right*)

1. If |*A*|=1，则输出元素值（当值为负时输出0）
2. *center*←⌊(*left*+*right*)/2⌋
3. *leftsum*←MaxSubSum(*A*,*left*,*center*)　　//左和
4. *righsum*←MaxSubSum(*A*,*center*+1,*right*)　　//右和
5. *s*1←*A*1[*center*]　　//从center向左的最大和
6. *s*2←*A*2[*center*+1]　　//从center+1向右的最大和
7. *sum*←*s*1+*s*2
8. if *leftsum* > *sum* then *sum*←*leftsum*
9. if *rightsum* >*sum* then *sum*←*rightsum*

时间：$T(n)=2T(n/2)+O(n)$, $T(1)=O(1)$

$T(n)=O(n\log n)$

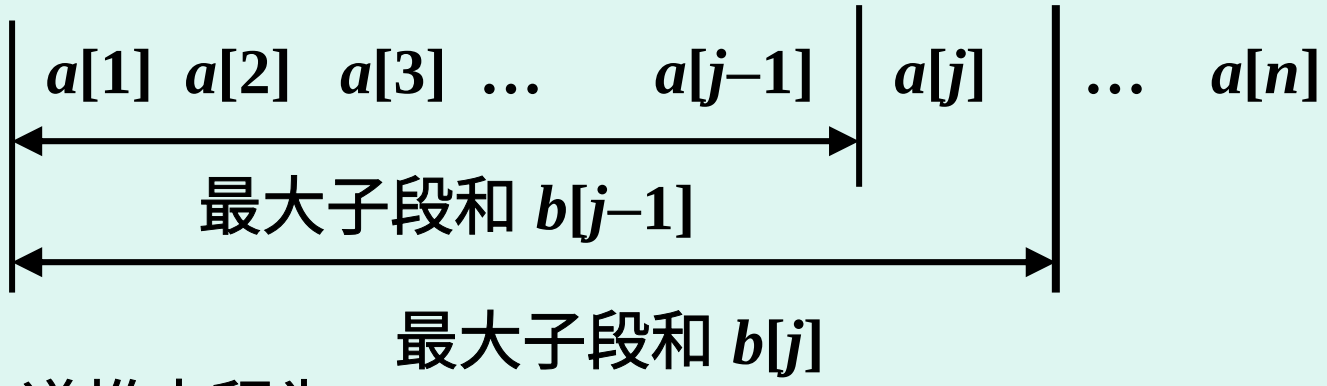
动态规划算法　MaxSum

令　　$b[j]=\max\limits_{1\le i\le j}\{\sum\limits_{k=i}^{j}a[k]\}$

多步判断：

$b[j]$表示最后一项为$a[j]$的序列构成的最大的子段和

最优解为$b[1], b[2], .., b[n]$中的最大值



递推方程为

$b[j]=\max\{b[j-1]+a[j],a[j]\}\ \ j=1,2,\ldots,n$

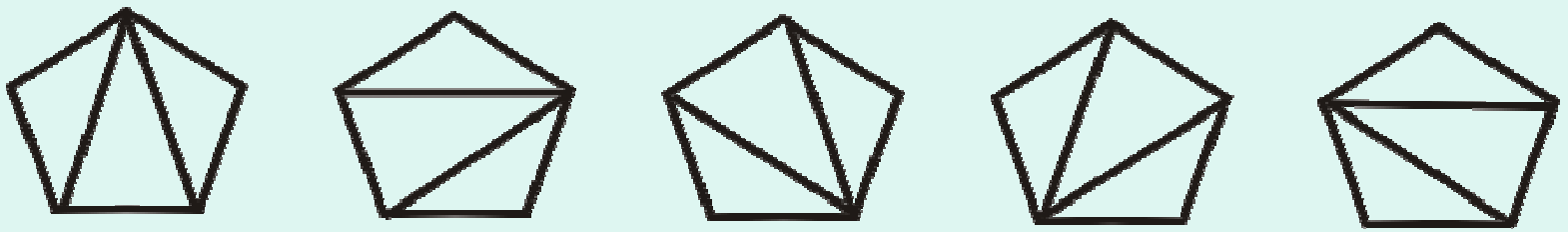
算法 MaxSum

```
MaxSum(A,n)
1. sum←0; b←0
2. for i←1 to n do
3.     if b>0 then b←b+a[i]
4.     else b←a[i]
5.     if b>sum then
6.         sum←b
7.         c←i    //记录最大和的末项标号
8. return sum
```

复杂性 时间：$O(n)$, 空间$O(n)$.

例8　凸多边形的三角划分

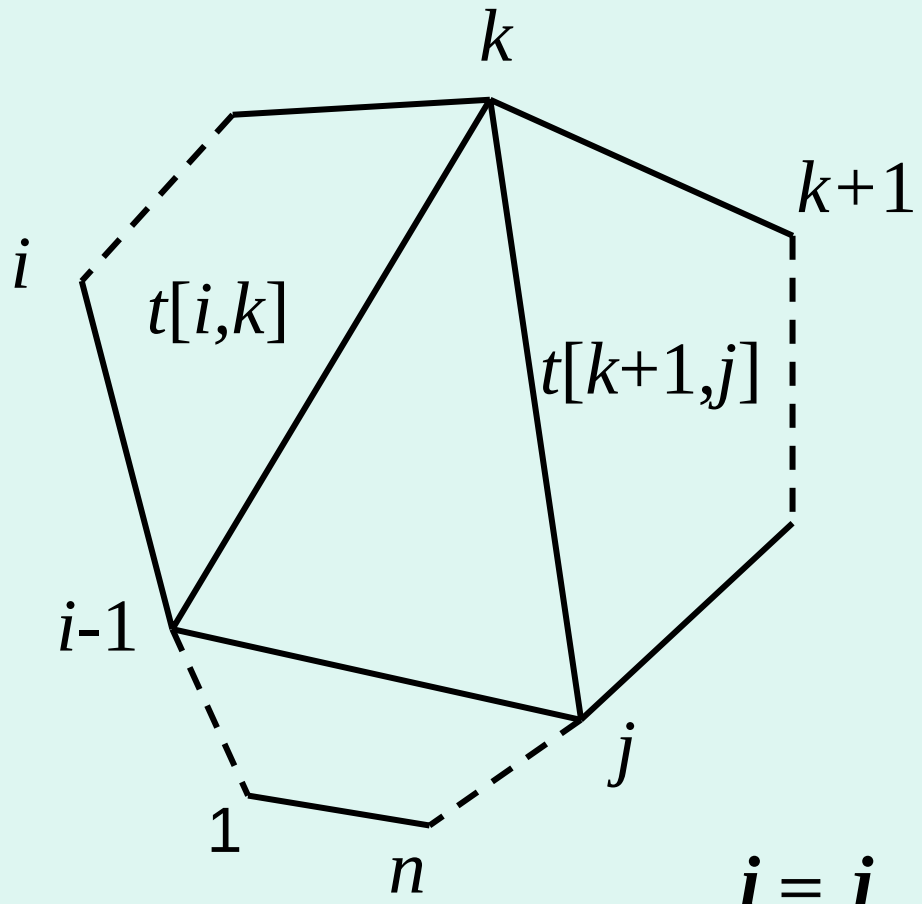
问题：给定凸n边形P，用在内部互不相交的n-3条对角线（弦）将P分割成三角形。设$P=\{v_0,v_1,\ldots,v_{n-1}\}$, 权函数$W$（例如对每个三角形赋权为三边长度之和）, 求具有最小权和（对角线总长度最短）的分割方案.



递 推 方 程

定义

$t[i,j]$为凸多边形 $\{v_{i-1}, v_i, \ldots, v_j\}$ 的最优分割对应的最小权值，则



$$t[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i,k]+t[k+1,j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

凸多边形的三角划分

$$t[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i,k]+t[k+1,j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

矩阵乘法

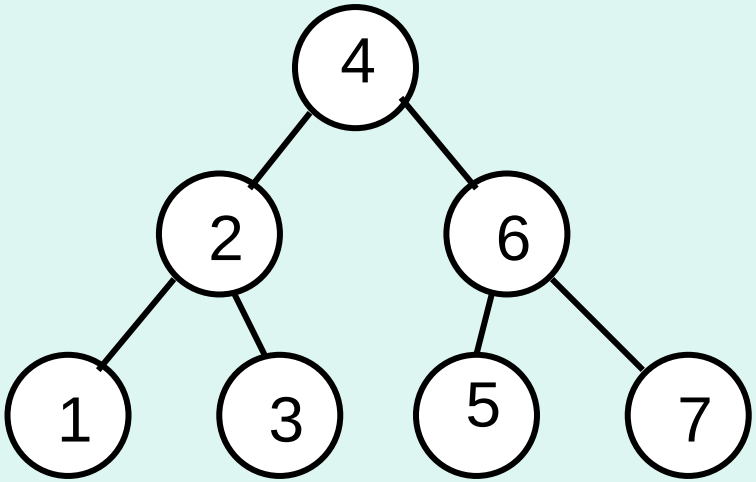
$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\} & i<j\end{cases}$$

求解算法参照算法 **MatrixChain,**

时间为$O(n^3)$，空间为$O(n^2)$.

例11　最优二叉搜索树

设集合S为排序的n个元素 $x_1<x_2<\ldots<x_n$，将这些元素存储在一棵二叉树的结点上，以查找x是否在这些数中. 如果x不在，确定x在那个空隙.



S={ 1, 2, 3, 4, 5, 6, 7 }, 等概分布下

存取概率不等情况

空隙：

$(-\infty, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n), (x_n, +\infty)$,

$x_0 = -\infty, x_{n+1} = \infty$

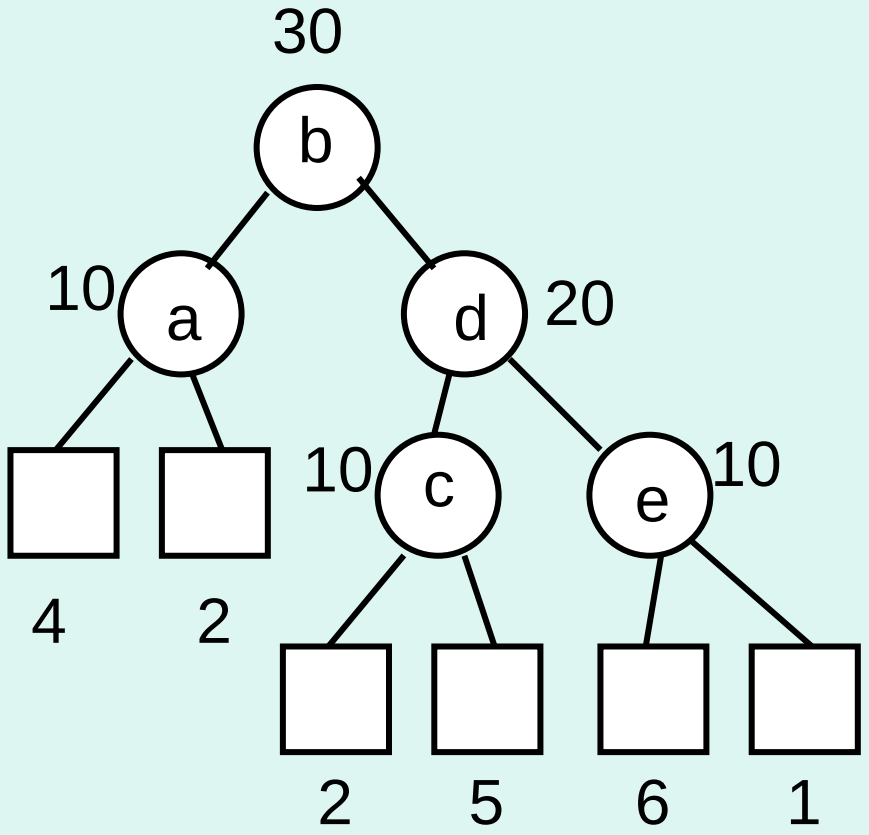
给定序列 $S = <x_1, x_2, \ldots, x_n>$,

x在 x_i 的概率为b_i ,

x在 (x_i , x_{i+1}) 的概率为a_i ,

得到存取概率分布如下：

$C=(a_0, b_1, a_1, b_2, a_2, \ldots , b_n, a_n)$



实例

$S = <a, b, c, d, e>$

$C = (4, 10, 2, 30, 2, 10, 5, 20, 6, 10, 1) /100$

$p = [0.3\times1+(0.1+ 0.2)\times2 +(0.1+ 0.1)\times3]$

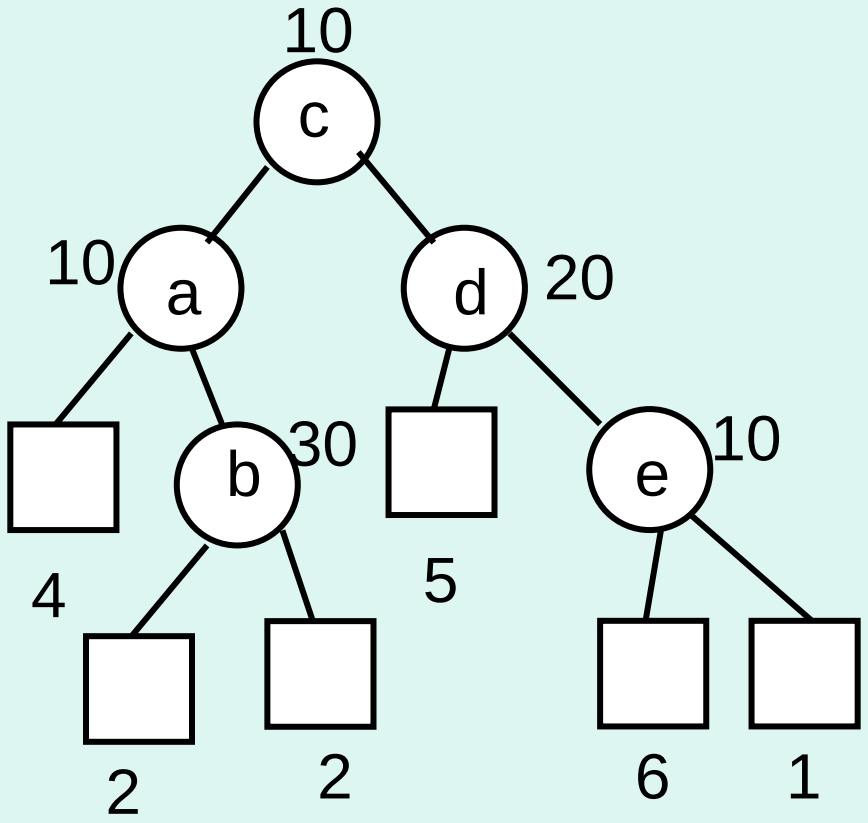
$+ [(0.04 + 0.02)\times2+ (0.02+0.05+0.06+0.01)\times3]$

$= 2.04$

另一种搜索树

$S = \langle a, b, c, d, e \rangle$

$C = (4, 10, 2, 30, 2,$
$10, 5, 20, 6,$
$10, 1) /100$



$p = [0.1\times1+ (0.1+0.2)\times2+(0.1+0.3)\times3]$
$+ [(0.04 + 0.05)\times2+ (0.02+0.02+0.06+0.01)\times3]$
$= 2.41$

设 x_i 的结点深度为 c_i，

(x_j, x_{j+1}) 空隙结点（用叶结点表示）的深度为 d_j，

则平均比较次数为：

$$p = \sum_{i=1}^{n} b_i (1 + c_i) + \sum_{j=0}^{n} a_j d_j$$

问题：给定集合 S 和存取概率分布 C，

求一棵平均查找次数（平均路长）p 最小的二叉搜索树，即最优二叉搜索树

设 T_{ij} 为$\{ x_i , x_{i+1}, \cdots , x_j \}$
关于存取概率 (条件概率) 分布

$$(\overline{a_{i-1}}, \overline{b_i}, \overline{a_i}, ..., \overline{b_j}, \overline{a_j})$$

其中：

$$\overline{b_k} = b_k / w_{ij}, \quad i \le k \le j$$

$$\overline{a_h} = a_h / w_{ij}, \quad i-1 \le h \le j$$

$$w_{ij} = a_{i-1} + b_i + ... + b_j + a_j$$

的最优二叉搜索树，其平均路长 p_{ij},

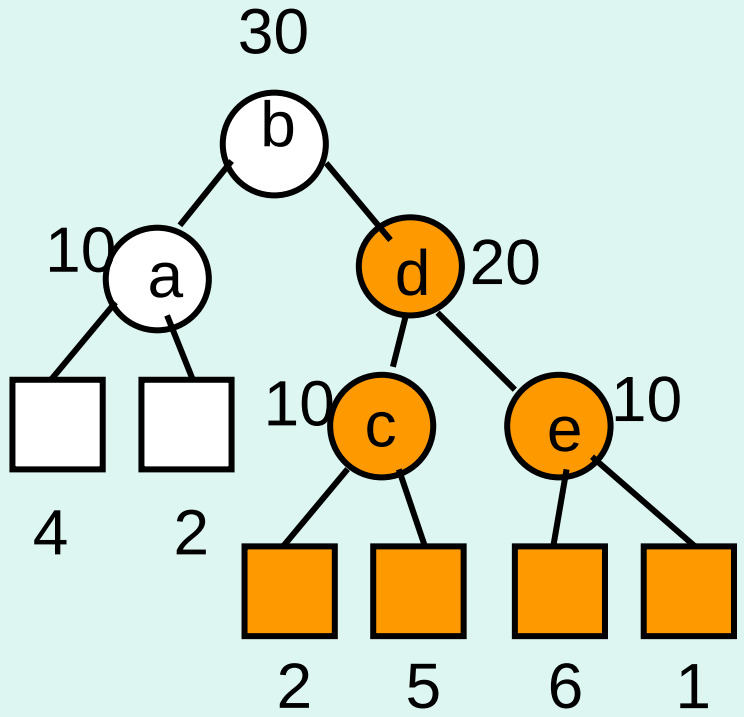
$$m(i,j) = w_{ij} p_{ij}$$

递推方程

$$T_{35}=\{x_3, x_4, x_5\}=\{c,d,e\}$$

$$(\overline{a_2}, \overline{b_3}, \overline{a_3}, \overline{b_4}, \overline{a_4}, \overline{b_5}, \overline{a_5})$$

$$=(\frac{2}{54},\frac{10}{54},\frac{5}{54},\frac{20}{54},\frac{6}{54},\frac{10}{54},\frac{1}{54})$$



$$m(3,5)=1+\{1\times\frac{17}{54}+1\times\frac{17}{54}\}=1+\frac{17}{27}\approx 1.63, \quad k=4$$

$$m(3,5)=\frac{20}{54}\times 1+\frac{34}{54}\times 2\approx 0.37+1.26=1.63$$

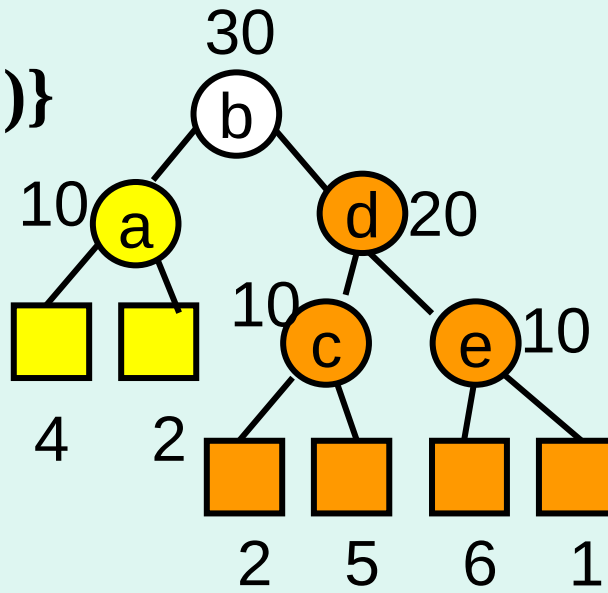
递推方程

$$m(3,5)=1.63$$

$$m(1,5)=1+\min_{k=2,3,4}\{m(1,k-1)+m(k+1,5)\}$$

$$=1+\{m(1,1)+m(3,5)\} \qquad k=2$$

$$=1+\{1\times\frac{16}{100}+1.63\times\frac{54}{100}\}=2.04$$



$$m(i,j)=1+\min_{i<k<j}\{m(i,k-1)+m(k+1,j)\}, \quad i<j$$

$$m(i,i)=1, \quad 1\le i\le n$$

复杂性估计：$T(n)=O(n^3)$, $S(n)=O(n^2)$.

可以改进为 $T(n)=O(n^2)$

RNA二级结构

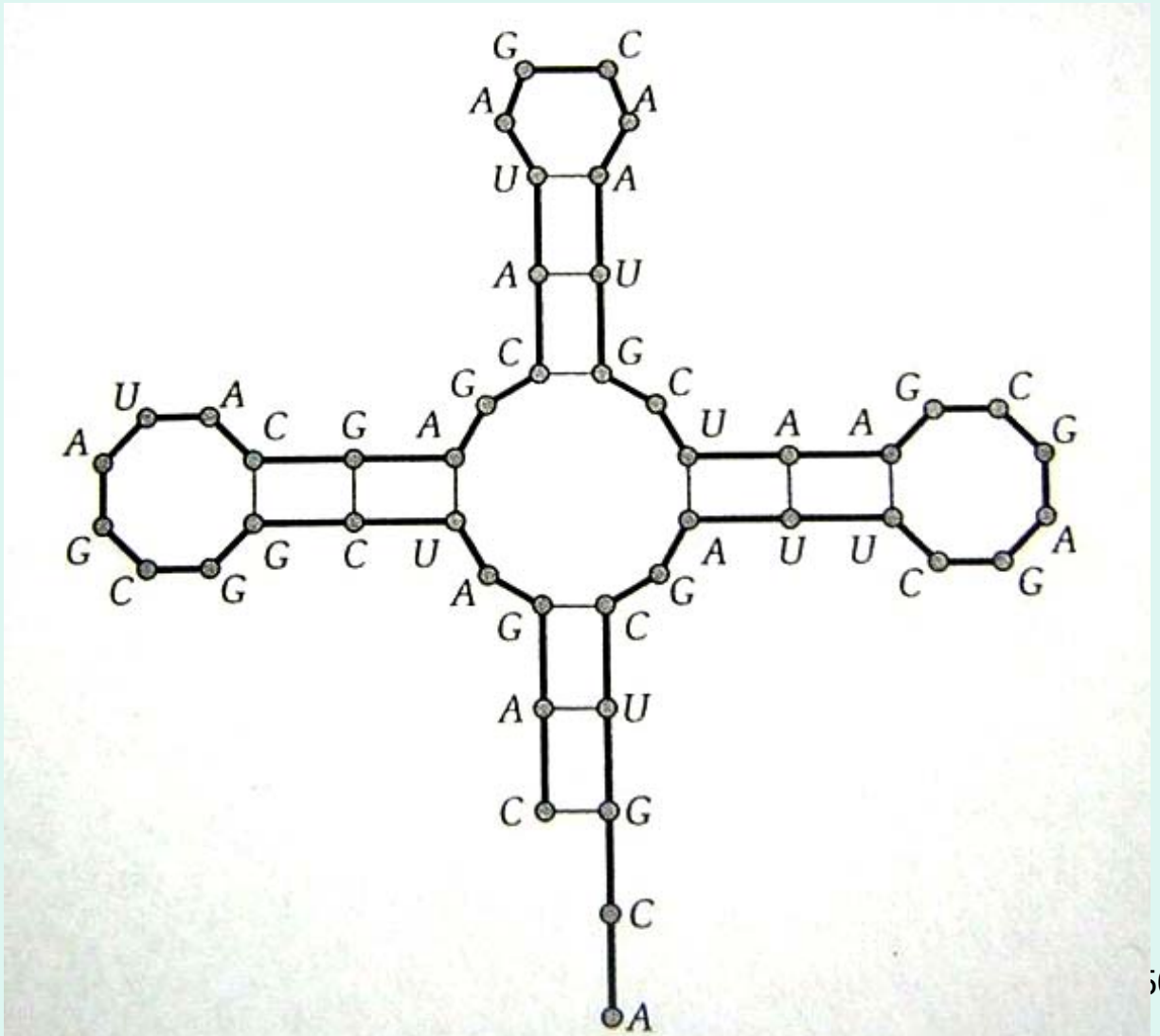
四种碱基：

***A*、*C*、**

G*、*U

一级结构

是一条链



二级结构的构成条件

(1) 没有尖的转弯：
参与配对的元素至少相隔4个碱基，即如果(i,j)是一个配对，那么$i<j-4$.

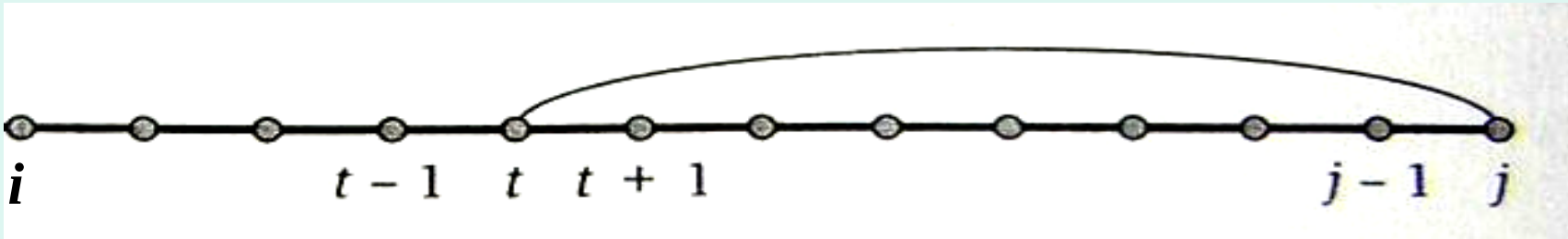
(2) 配对原则：
A-U，*C-G*，顺序不限

(3) *S*是匹配：
没有碱基出现在两个或更多的配对中

(4) 不交叉：
如果(i,j)与(k,l)是*S*中的两个对，那么我们不能有$i<k<j<l$.

问题与算法设计

给定RNA的一级结构：由*A,U,C,G*构成的长为*n*的序列，寻找具有最大匹配对数的二级结构.



令OPT(*i,j*)是序列*L*[*i..j*]的最大匹配对数

OPT(*i,j*)=max[OPT(*i,j*–1),
max(1+OPT(*i,t*–1)+OPT(*t*+1,*j*–1))]

动态规划算法的设计

设计步骤

- 将问题表示成多步判断
- 确定是否满足优化原则——必要条件
- 确定子问题的重叠性——估计算法效率
- 列出关于优化函数的递推方程(或不等式)和边界条件
- 自底向上计算子问题的优化函数值----非递归的算法
- 备忘录方法记录中间结果
- 标记函数追踪问题的解

动态规划算法的问题

- 时间复杂性改进依赖于子问题的重叠程度
- 空间复杂性较高

回溯（backtrack）

回溯算法的基本思想和适用条件

回溯算法的设计步骤

估计回溯算法的效率

改进回溯算法的途径

分支估界

应用实例

基本思想和适用条件

实例

基本思想

搜索问题、搜索空间、搜索策略

判定条件、结点状态、存储结构

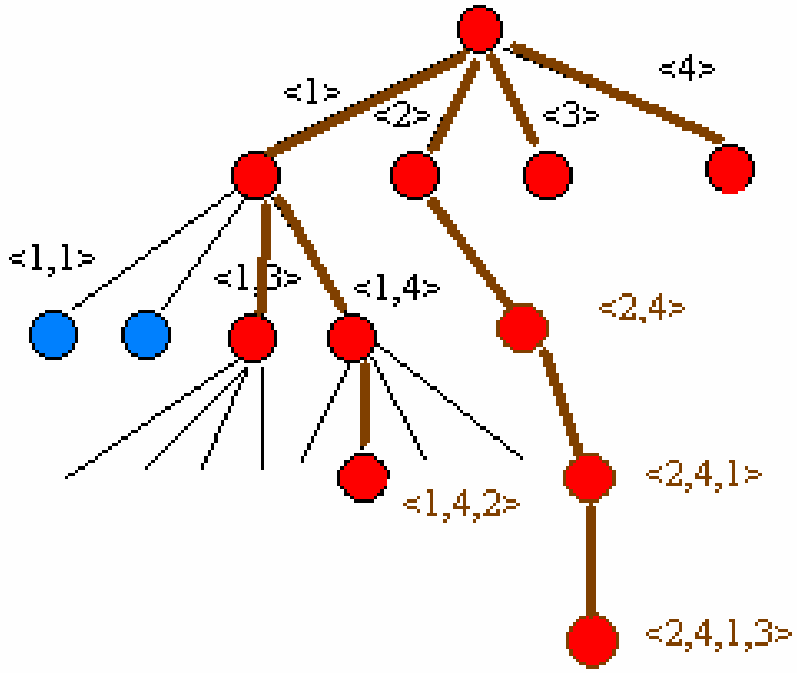
必要条件

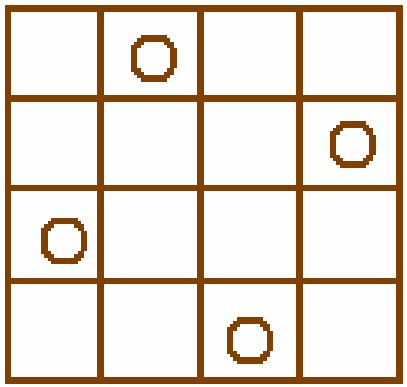
实　例

例1　四后问题

解表示成一个4维向量，$<x_1,x_2,x_3,x_4>$（放置列号）

搜索空间：4叉树



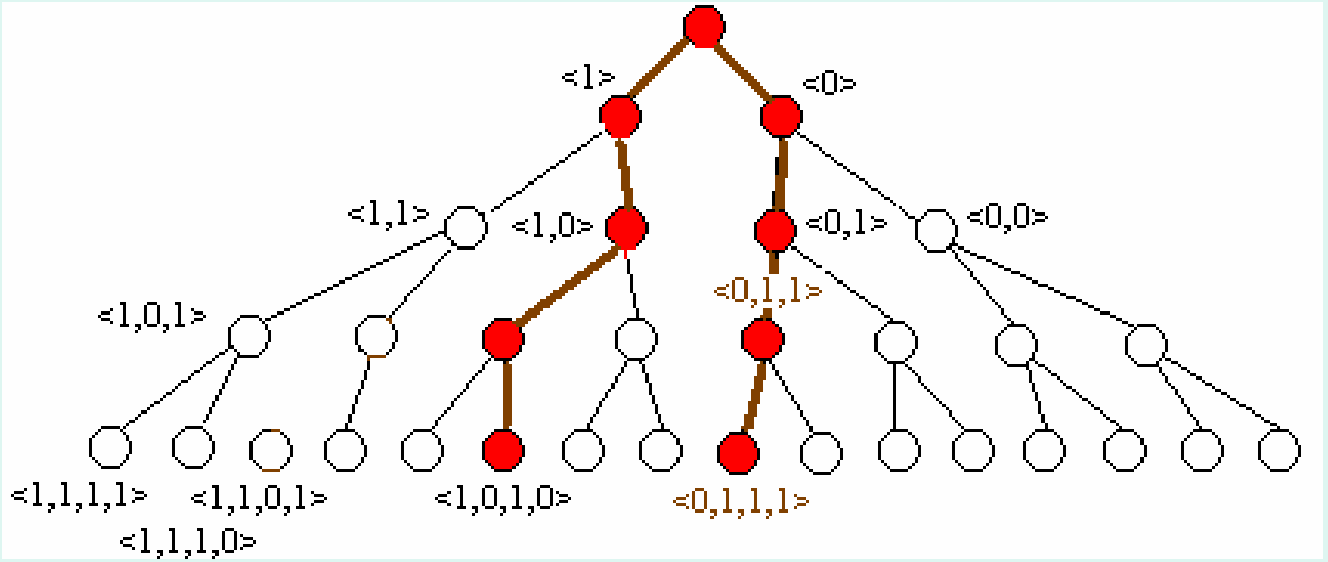


例2　0-1背包问题：

V={12,11,9,8}, W={8,6,4,3}, B=13

结点：向量<x_1,x_2,x_3,x_k>（子集的部分特征向量）

搜索空间：子集树，2^n片树叶



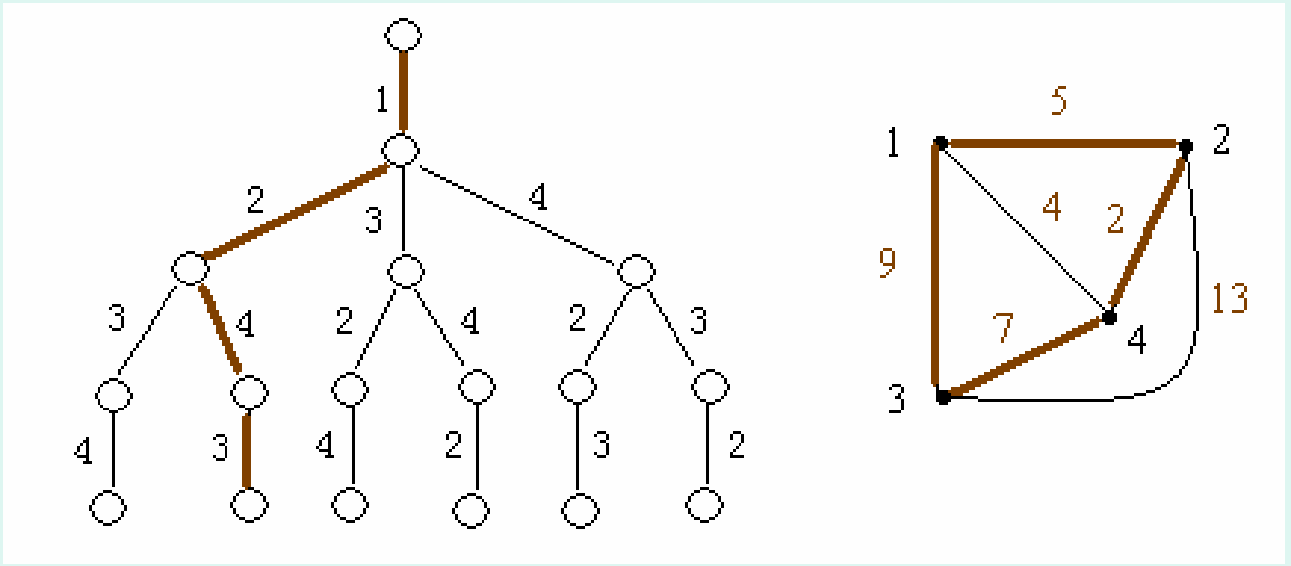
<0,1,1,1> 对应于可行解：x_1=0, x_2=1, x_3=1, x_4=1. 重量：13, 价值：28

<1,0,1,0>对应于可行解：x_1=1, x_2=0, x_3=1, x_4=0. 重量：12，价值：21

例3　巡回售货员问题：

结点：向量<$x_1,x_2,\ldots,x_k$>　（部分巡回路线）

搜索空间：排列树



<1,2,4,3> 对应于巡回路线：

1→2 →4 →3 →1

长度：5+2+7+9=23

基 本 思 想

适用问题：求解搜索问题

搜索空间：一棵树

每个结点对应了部分解向量，

树叶对应了可行解

搜索过程：

采用系统的方法隐含遍历搜索树

搜索策略：

深度优先，宽度优先，函数优先，宽深结合等

结点分支判定条件：

满足约束条件---分支扩张解向量

不满足约束条件，回溯到该结点的父结点

结点状态：

动态生成

结点状态：白结点（尚未访问）；

灰结点（正在访问该结点为根的子树）

黑结点（该结点为根的子树遍历完成）

存储：当前路径

必要条件--多米诺性质

设$P(x_1, x_2, \ldots, x_i)$为真表示向量$<x_1,x_2, \ldots,x_i>$中i个皇后放置在彼此不能攻击的位置

$$P(x_1, x_2, \ldots, x_{k+1}) \rightarrow P(x_1, x_2, \ldots, x_k) \qquad 0 < k < n$$

例4　求不等式的整数解

$$5x_1+4x_2-x_3 \le 10,\ 1 \le x_i \le 3,\ i=1,2,3$$

$P(x_1, \ldots, x_k)$:意味将$x_1,x_2,\ldots,x_k$代入原不等式的相应部分使得左边小于等于10

不满足多米诺性质

变换： 令$x_3=3-x_3'$,

$$5x_1+4x_2+x_3' \le 13\ ,\ \ 1 \le x_1, x_2 \le 3,\ 0 \le x_3' \le 2$$

回溯算法的设计步骤

设计要素

定义搜索问题的解向量和每个分量的取值范围

解向量为$<x_1,x_2,\ldots,x_n>$

x_i的可能取值的集合为X_i，$i=1,2,\ldots,n$.

$x_1,x_2,\ldots,x_{k-1}$确定以后x_k的取值集合为S_k, $S_k \subseteq X_k$

确定结点儿子的排列规则

判断是否满足多米诺性质

搜索策略----深度优先

确定每个结点能够分支的约束条件

确定存储搜索路径的数据结构

递归回溯

算法　ReBack(k)

1. if $k>n$ then $< x_1, x_2, \ldots, x_n>$是解;
2. else while $S_k \neq \varnothing$ do
3. 　　$x_k \leftarrow S_k$中最小值
4. 　　$S_k \leftarrow S_k - \{x_k\}$
5. 　　计算S_{k+1}
6. 　　ReBack(k+1)

算法　ReBacktrack(n)

1. for $i \leftarrow 1$ to n 计算X_k
2. ReBack(1)

迭代回溯

迭代算法Backtrack

1. 对于i=1,2,…,n 确定X_i
2. k=1
3. 计算S_k
4. while $S_k \neq \varnothing$ do
5. $x_k \leftarrow S_k$中最小值; $S_k \leftarrow S_k - \{x_k\}$
6. if $k<n$ then
7. $k \leftarrow k+1$; 计算S_k
8. else $<x_1,x_2,\ldots,x_n>$是解
9. if $k>1$ then $k \leftarrow k-1$； goto 4

估计回溯算法的平均效率

计数搜索树中平均遍历的结点，Monte Carlo方法

Monte Carlo方法

1．从根开始，随机选择一条路经，直到不能分支为止，即从$x_1,x_2,\ldots$, 依次对x_i赋值，每个x_i的值是从当时的S_i中随机选取，直到向量不能扩张为止

2．假定搜索树的其他$|S_i|$-1个分支与以上随机选出的路径一样，计数搜索树的点数。

3．重复步骤1和2，将结点数进行概率平均。

算法

算法Monte Carlo

1. *sum*←0　　//*sum*为 *t* 次结点平均数
2. for *i*←1 to *t* do　　//取样次数 *t*
3. 　　*m*←Estimate(*n*)　　//*m*为本次结点总数
4. 　　*sum*←*sum*+*m*
5. *sum*←*sum*/*t*

计算结点数

m为本次取样结点总数，k 为层数，r_1为本层分支数，r_2为上层分支数，n为树的层数

算法Estimate(n)

1. $m \leftarrow 1$; $r_2 \leftarrow 1$; $k \leftarrow 1$ //m为结点总数
2. While $k \le n$ do
3. if $S_k = \varnothing$ then return m
4. $r_1 \leftarrow |S_k| * r_2$ //r_1为扩张后结点总数
5. $m \leftarrow m + r_1$ // r_2为扩张前结点总数
6. $x_k \leftarrow$ 随机选择 S_k 的元素
7. $r_2 \leftarrow r_1$
8. $k \leftarrow k+1$

例5 Monte Carlo方法估计四后问题的效率

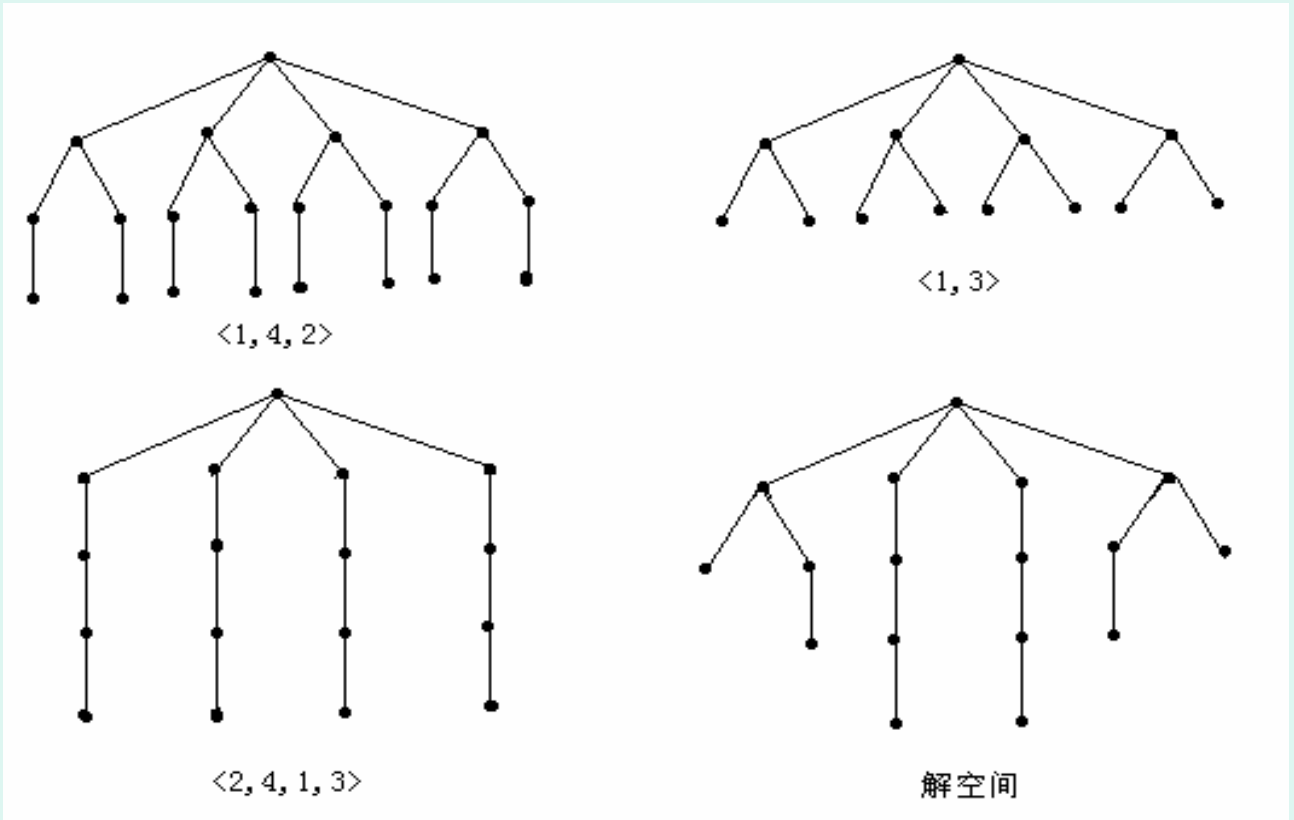
case1．<1,4,2>：1+4+4×2+4×2 = 21

case2．<2,4,1,3>:：4×3+1=17

case3．<1,3>：1+4×1+4×2=13

假设 4次抽样测试：case1 1次，case2 1次，case3 2次，平均为16

解空间的结点数为17



影响算法效率的因素

1. 最坏情况下的时间$W(n)=(p(n)f(n))$

   其中$p(n)$每个结点时间，$f(n)$结点个数

2. 影响回朔算法效率的因素

   搜索树的结构

   分支情况：分支均匀否

   树的深度

   对称程度：对称适合裁减

   解的分布

   在不同子树中分布多少是否均匀

   分布深度

   约束条件的判断：计算简单

改 进 途 径

根据树分支设计优先策略：

结点少的分支优先，解多的分支优先

利用搜索树的对称性剪裁子树

分解为子问题:

求解时间$f(n)=c2^n$，组合时间$T=O(f(n))$

如果分解为k个子问题，每个子问题大小为n/k

则求解时间为 $kc2^{\frac{n}{k}}+T$

组合优化问题的描述

目标函数（极大化或极小化）

约束条件

搜索空间中满足约束条件的解称为可行解

使得目标函数达到极大(或极小)的解称为最优解

例6　背包问题：

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \le 10$$

$$x_i \in N, i = 1,2,3,4$$

分支估界技术

设立代价函数（极大化）

函数值是以该结点为根的搜索树中的所有可行解的目标函数值的上界

父结点的代价大于等于子结点的代价

设立界

代表当时已经得到的可行解的目标函数的最大值

搜索中停止分支的依据

不满足约束条件或者其代价函数小于当时的界

界的设定与更新（目标函数值为正数）

初值可以设为**0**

可行解的目标函数值大于当时的界，进行更新

实　例

背包问题：

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \le 10$$

$$x_i \in N, i = 1,2,3,4$$

对变元重新排序

$$\frac{v_i}{w_i} \ge \frac{v_{i+1}}{w_{i+1}}$$

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10$$

$$x_i \in N, i = 1,2,3,4$$

代价函数与分支策略的设定

$<x_1, x_2, \ldots, x_k>$的代价函数

$$\sum_{i=1}^{k} v_i x_i + (b - \sum_{i=1}^{k} w_i x_i) \frac{v_{k+1}}{w_{k+1}}$$

若对某个$j > k$有$b - \sum_{i=1}^{k} w_i x_i \geq w_j$

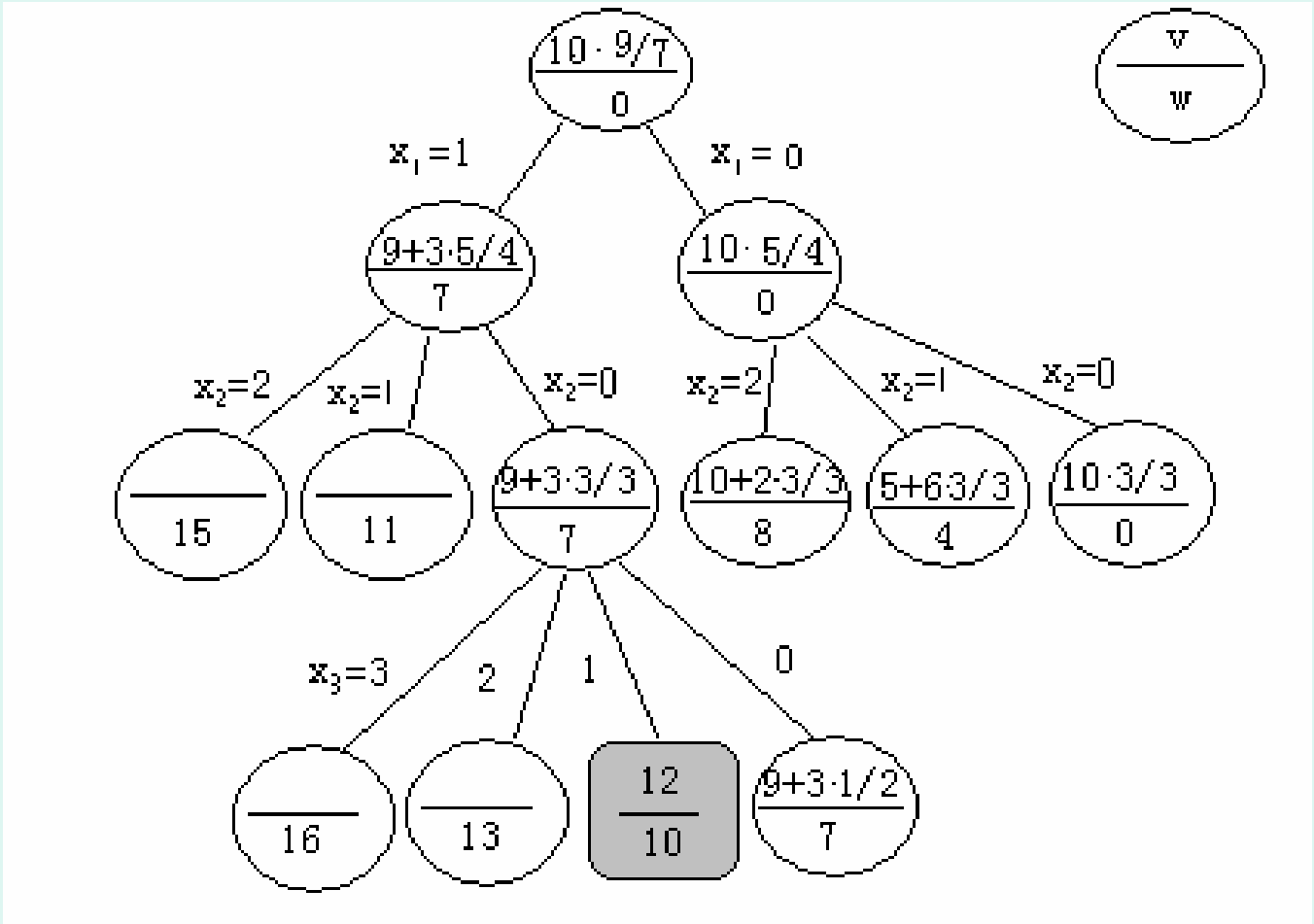
$$\sum_{i=1}^{k} v_i x_i$$ 否则

分支策略----深度优先

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10$$

$$x_i \in N, i = 1,2,3,4$$



实例（续）

例7 最大团问题

给定无向图$G=<V,E>$,求G中的最大团.

算法设计

结点$<x_1,x_2,\ldots,x_k>$的含义：已检索k个顶点，其中$x_i=1$对应的顶点在当前的团内，搜索树为子集树

约束条件：该顶点与当前团内每个顶点都有边相连

界：当前图中已检索到的极大团的顶点数

代价函数：目前的团扩张为极大团的顶点数上界

$F=c_n+n-k$

其中c_n为目前团的顶点数（初始为0），k为结点层数

时间：$O(n2^n)$

例8 图的m着色

给定无向连通图G和m种颜色，给图的顶点着色，每个顶点一种颜色，且每条边的两个顶点不同色。给出所有可能的着色方案（如果不存在，则回答“No”）

搜索空间为m叉完全树. 将颜色编号为1, 2, … , m.

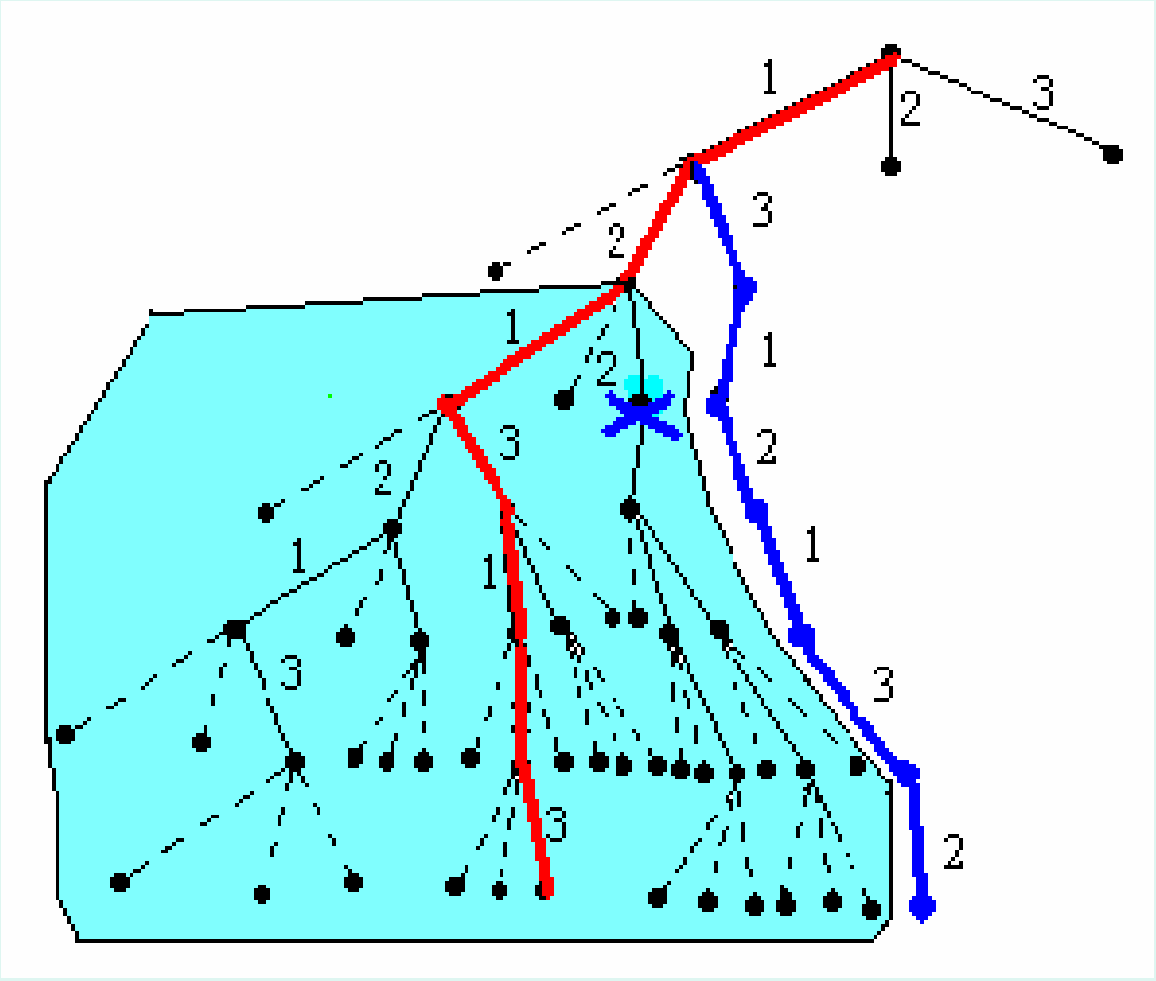
结点$<x_1, x_2, \dots, x_k>$ 表示顶点1着色x_1,顶点2着色x_2, … ,
顶点 k着色x_k

约束条件：该顶点邻接表中已着色的顶点没有同色的

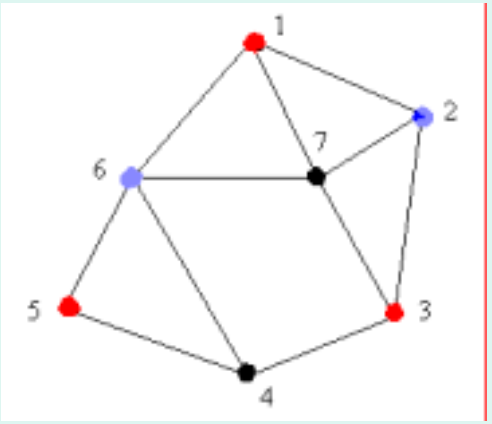
代价函数：没有（不是优化问题）

时间：$O(nm^n)$

回溯算法的图示



<1>
<1,2>
<1,2,1>
<1,2,1,3>
<1,2,1,3,1>
<1,2,1,3,1,2>
<1,2,1,3,1,2,3>



提高效率的途径

根据对称性，只需搜索1/3的解空间即可. 当1和2确定,即<1,2>以后，只有1个解，因此在<1,3>为根的子树中也只有1个解.由于3个子树的对称性，总共有6个解.

进一步分析，在取定<1,2>以后，不可以扩张成<1,2,3>, 因为可以检查是否有和1,2,3都相邻的顶点.如果存在,例如7, 则没有解. 所以可以从打叉的结点回溯，而不必搜索它的子树.

例9　巡回售货员问题

问题：给定n个城市集合$C=\{c_1,c_2,\ldots,c_n\}$, 从一个城市到另一个城市的距离d_{ij}为正整数，求一条最短且每个城市恰好经过一次的巡回路线.

巡回售货员问题的类型：有向图、无向图.

设巡回路线从1开始，解向量为$<i_1,i_2,\ldots,i_{n-1}>$, 其中$i_1,i_2,\ldots,i_{n-1}$为$\{2,3,\ldots,n\}$的排列. 搜索空间为排列树，结点$<i_1,i_2,\ldots,i_k>$表示得到k步巡回路线

约束条件：令$B=\{i_1,i_2,\ldots,i_k\}$, 则

$i_{k+1}\in\{2,\ldots,n\}-B$

界：当前得到的最短巡回路线长度

代价函数：设顶点c_i出发的最短边长度为l_i，d_j为选定的部分巡回路线中第 j 段的长度，则

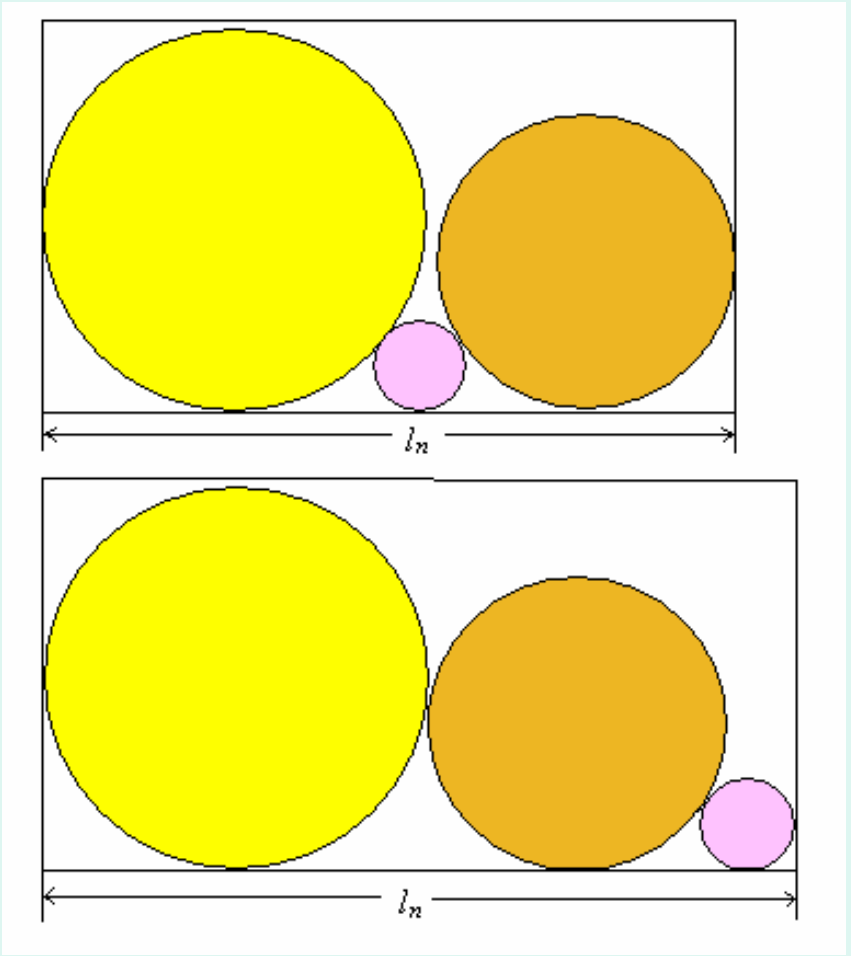
$$L=\sum_{j=1}^{k}d_j+\sum_{i_j\notin B}l_{i_j}$$

为部分巡回路线扩张成全程巡回路线的长度下界

时间 $O(n!)$：计算$O((n-1)!)$次，代价函数计算$O(n)$

例11 圆排列问题

给定n个圆的半径序列$R=\{r_1,r_2,\ldots,r_n\}$，将圆放到矩形框中，各圆与矩形底边相切，求具有最小长度l_n的圆的排列顺序.



算 法 设 计

解为$<i_1,i_2,\ldots,i_n>$：其中$i_1, i_2, \ldots, i_n$为$1, 2, \ldots, n$ 的排列

解空间为排列树.

部分解向量$<i_1,i_2, \ldots,i_k>$：表示前 k 个圆已经排好.

令$B=\{i_1,i_2, \ldots, i_k\}$,下一个园选择 i_{k+1}.

约束条件：$i_{k+1}\in\{1,2,\ldots,n\}-B$

界：当前得到的最小园排列长度

代价函数计算的符号说明

k：算法完成第k步，已经选择了第1—第k个圆，

r_k：第k个圆的半径

d_k：第k-1个圆到第k个圆的圆心距离

x_k：第k个圆的圆心坐标，规定$x_1=0$,

l_k：第1—第k个圆的排列长度

L_k：放好1—第k个圆后，对应结点的代价函数值

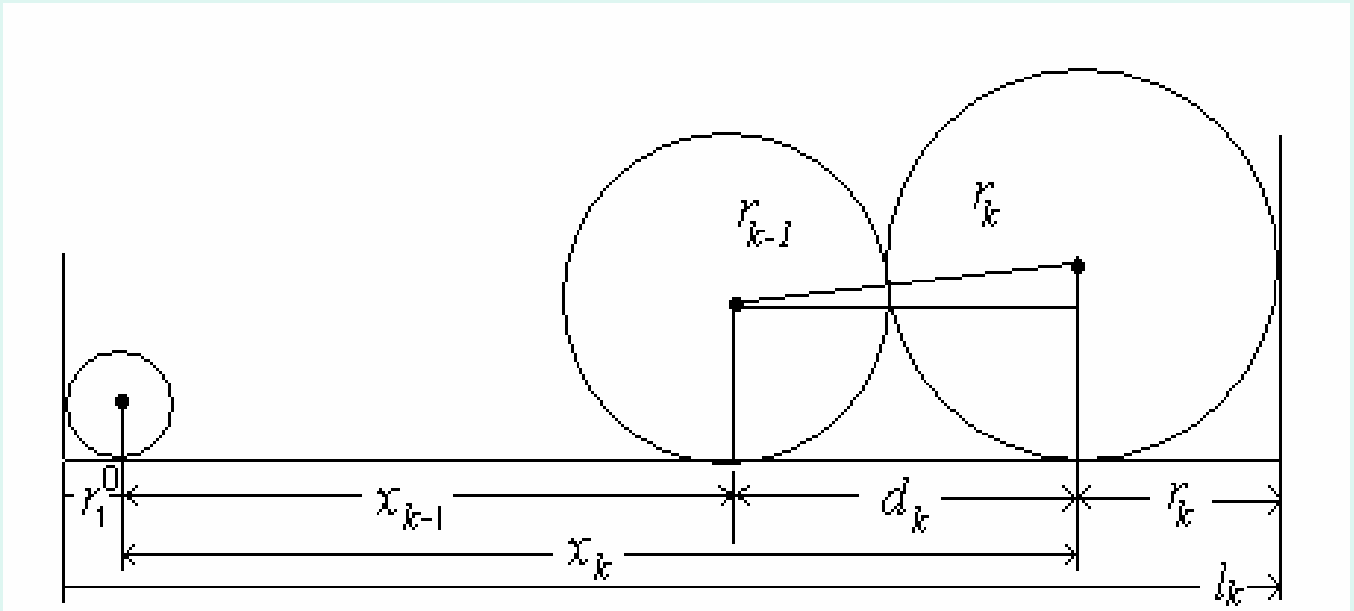
$L_k \leq l_n$

相 关 公 式

$$d_k = \sqrt{(r_{k-1} + r_k)^2 - (r_{k-1} - r_k)^2} = 2\sqrt{r_{k-1} r_k}$$

$$x_k = x_{k-1} + d_k$$

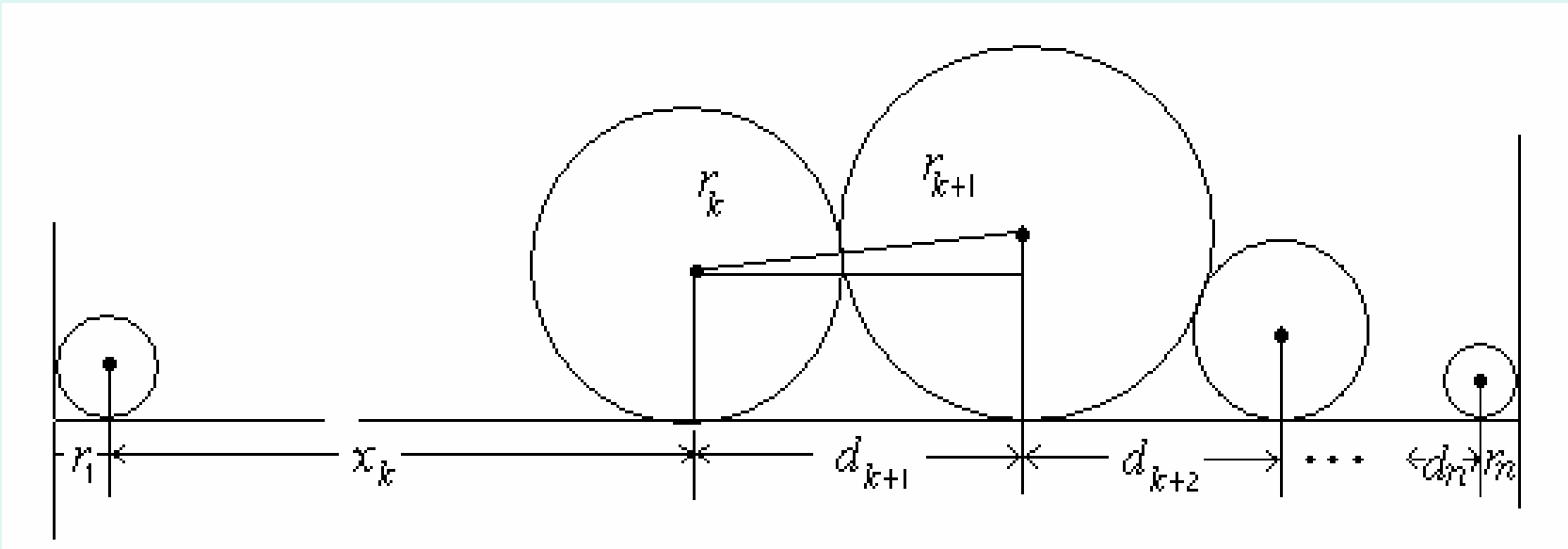
$$l_k = x_k + r_k + r_1$$



相关公式（续）

$$L_k = x_k + d_{k+1} + d_{k+2} + \ldots + d_n + r_n + r_1$$

$$= x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + \ldots + 2\sqrt{r_{n-1} r_n} + r_n + r_1$$



代 价 函 数

$$L_k = x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + \ldots + 2\sqrt{r_{n-1} r_n} + r_n + r_1$$

$$\geq x_k + 2(n-k)r + r + r_1$$

$$= x_k + (2n - 2k + 1)r + r_1$$

$$r = \min(r_{i_j}, r_k) \quad i_j \in \{1,2,\ldots,n\} - B$$

$$B=\{i_1, i_2, \ldots, i_k\},$$

时间：$O(n\,n!)=O((n+1)!)$

实　例

R={1,1,2,2,3,5}

取排列<1,2,3,4,5,6>,

半径排列为：1,1,2,2,3,5，结果见下表和下图

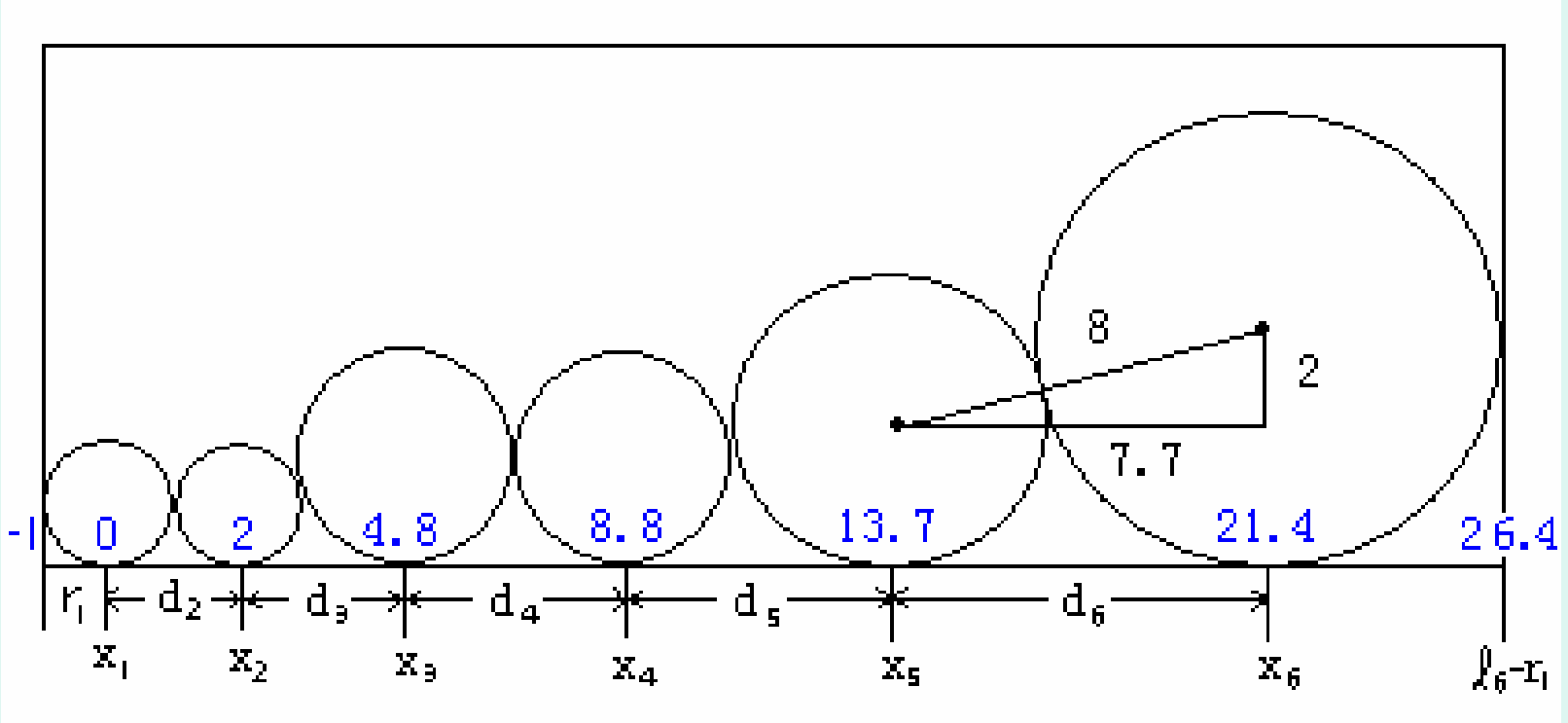
k	r_k	d_k	x_k	l_k	L_k
1	1	0	0	2	12
2	1	2	2	4	12
3	2	2.8	4.8	7.8	19.8
4	2	4	8.8	11.8	19.8
5	3	4.9	13.7	17.7	23.7
6	5	7.7	21.4	27.4	27.4

R={1,1,2,2,3,5}

取排列<1,2,3,4,5,6>, 半径排列为：1,1,2,2,3,5，

最短长度L_6=27.4

取排列<1,3,5,6,4,2>, 半径排列为1,2,3,5,2,1，L_6=26.5



回 溯 算 法 小 结

适应于求解组合搜索问题（含组合优化问题）

求解条件：满足多米诺性质

解的表示：解向量，求解是不断扩充解向量的过程

回溯条件：

　搜索问题---约束条件

　优化问题---约束条件+代价函数

算法复杂性：

　遍历搜索树的时间，最坏情况为指数

　空间代价小

平均时间复杂性的估计：**Monte Carlo**方法

降低时间复杂性的主要途径：

　利用对称性裁减子树

　划分成子问题

分支策略（深度优先、宽度优先、宽深结合、优先函数）

贪心法

(Greedy Approach)

基本思想

算法设计

- 设计要素
- 与动态规划法的比较
- 正确性证明

得不到最优解的处理办法

应用实例

基 本 思 想

实例：最小生成树的 **Kruskal** 算法, 活动选择问题

适用问题：组合优化问题，满足优化原则

设计方法：多步判断，解为判断序列

选择依据：

　　是否满足约束条件

　　局部优化测度

使用贪心法要解决的问题:

　　是否可以得到最优解?

　　不能得到最优解，解与最优解的误差估计

例1　活动选择问题

$S=\{1, 2, \ldots, n\}$为n 项活动的集合

s_i, f_i分别为活动 i 的开始和结束时间

活动 i 与 j 相容 当且仅当 $s_i \geq f_j$ 或 $s_j \geq f_i$

求最大的两两相容的活动集

思路：

按结束时间递增顺序将活动排列为$1,2,\ldots,n$,

使得　$f_1 \leq f_2 \leq \ldots \leq f_n$

按照标号从小到大选择

贪 心 算 法

算法 Greedy Select

1. $n \leftarrow length[S]$;
2. $A \leftarrow \{1\}$;
3. $j \leftarrow 1$;
4. for $i \leftarrow 2$ to n
5. do if $s_i \geq f_j$
6. then $A \leftarrow A \cup \{i\}$;
7. $j \leftarrow i$;
8. return A.

最后完成时间为$\max \{f_k : k \in A\}$.

实　例

输入

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

解为　$A = \{1, 4, 8, 11\}$　　$t=14$

正确性证明

定理1　算法Select 执行到第k步, 选择k项活动 $i_1= 1, i_2, \ldots, i_k$, 那么存在最优解A包含 $i_1=1, i_2, \ldots , i_k$

根据定理：算法至多到第n步得到最优解

对步数进行归纳的证明思路

- 证明存在最优解包含活动 1
- 假设按照算法前k步选择都导致最优解，证明第$k+1$步选择也导致最优解

1. 第k步：选择活动 $i_1=1, i_2, \dots, i_k$
2. 归纳假设：存在最优解

   $$A=\{i_1=1, i_2, \dots, i_k\}\cup B$$

   B是剩下的待选活动集 S'的一个最优解
3. 由归纳基础，存在S'的最优解 B' 包含i_{k+1}
4. 由 $|B'|=|B|$ 知 $A'=\{i_1=1, i_2, \dots, i_k\}\cup B'$ 最优
5. $A'=\{i_1=1, i_2, \dots, i_k, i_{k+1}\}\cup(B' -\{i_{k+1}\})$最优.

归 纳 基 础

设$S=\{1,2,\ldots,n\}$是活动集，

活动按截止时间递增顺序排序.

$k=1$, 证明存在最优解包含活动1.

任取最优解A, A中的活动按照截止时间递增的顺序排列. 如果A的第一个活动为j，$j\neq 1$, 令 $A'=(A-\{j\})\cup\{1\}$, 由于$f_1\leq f_j$, A'也是最优解，且含有1.

归 纳 步 骤

假设命题对k为真,证明对$k+1$也为真.

算法执行到第 k步, 选择了活动 $i_1=1, i_2, \ldots, i_k$, 根据归纳假设存在最优解A包含 $i_1= 1, i_2, \ldots , i_k$, A中剩下的活动选自集合$S'=\{ i \mid i\in S, s_i\geq f_k\}$,且

$$A= \{i_1, i_2, \ldots , i_k\}\cup B$$

B是S'的最优解. 若不然, S'的最优解为B^*, B^*的活动比 B多，那么$B^*\cup\{1, i_2, \ldots , i_k\}$是 S 的最优解，且比 A的活动多,与 A 的最优性矛盾.

根据归纳基础，存在 S'的最优解B'含有S'中的第一个活动，设为i_{k+1}, 且$|B'|=|B|$, 于是

$$\{i_1,i_2,\ldots,i_k\}\cup B'=\{i_1,i_2,\ldots i_k,i_{k+1}\}\cup(B'-\{i_{k+1}\})$$

也是原问题的最优解.

贪心算法的设计

适用：

满足优化原则的组合优化问题

问题求解表示成多步判断

整个判断序列对应问题的最优解

子序列对应子问题的最优解

贪心选择：

确定一个优化测度

不考虑以前的选择，只与当前状态有关

以优化测度的极大(或极小)作为依据

贪心算法的设计（续）

确定是否满足贪心选择性质

具有贪心选择性质得到最优解，否则为近似解

正确性证明：

归纳法：证明贪心法得到最优解

对算法步数归纳、对问题规模归纳

交换论证：在保证最优性不变的前提下，从一个最优解出发进行逐步替换，最终得到贪心法的解

自顶向下计算

通过贪心选择，将原问题规约为子问题

线性表记录选择的结果

动态规划、分支估界和贪心法比较

技术	动态规划	分支估界	贪心法
使用条件	优化原则 多步判断	多米诺性质 多步判断	贪心选择+优化原则 多步判断
选择依据	子问题结果	约束条件和界	局部最优性质
计算过程	看子问题结果选择 自底向上	选择后生成子问题 自顶向下	选择后生成子问题 自顶向下
数据结构	二维表	树、队列	线性表
解	一个最优解	一个和多个最优解	一个最优或近似解
关键问题	递推方程 空间复杂性高	设定代价函数 时间复杂性高	贪心选择性质证明 近似解的误差估计

贪心选择性质的证明

数学归纳法

叙述一个可以归纳证明的命题：

- **对步数k归纳**

  对于任意k，k步贪心选择得到 $i_1,i_2,\ldots,i_k$, 那么存在最优解包含$i_1,i_2,\ldots,i_k$

- **规模k归纳：**

  对于任意k,贪心法得到关于规模为k的问题的最优解

归纳基础：$k=1$, 命题为真

归纳步骤：假设对$<k$的正整数命题为真，证明对k命题也为真

例2 最优装载 Loading

n个集装箱$1,2,\ldots,n$ 装上轮船，集装箱i的重量w_i, 轮船装载重量限制为c， 无体积限制. 问如何装使得上船的集装箱最多？不妨设$w_i \le c$.

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} w_i x_i \le c$$

$$x_i = 0,1 \quad i = 1,2,\ldots,n$$

贪心法：将集装箱按照从轻到重排序，轻者先装

贪心选择性质证明

对规模的归纳

- 设集装箱标号按照从轻到重记为$1,2,\ldots,n$
- $n=1$,贪心选择得到最优解（只有1个箱子）
- 假设对于规模为$n-1$的输入得到最优解，证明对规模为n的输入也得到最优解

归纳步骤

假设对于n-1个集装箱的输入，贪心法都可以得到最优解，考虑n个集装箱的输入$N=\{1,2,\ldots,n\}$, 其中

$$w_1 \le w_2 \le \ldots \le w_n.$$

由归纳假设，对于$N'=\{2,3,\ldots,n\}$，$c'=c-w_1$, 贪心法得到最优解I'. 令$I=\{1\}\cup I'$，则I是关于N的最优解.

若不然，存在包含1的关于N的最优解I^*（如果I^*中没有1，用1替换I^*中的第一个元素得到的解也是最优解），且$|I^*|>|I|$；那么$I^*-\{1\}$是N'的解且

$$|I^*-\{1\}|>|I-\{1\}|=|I'|$$

与I'的最优性矛盾.

说　明

Loading算法

复杂性$T(n)=O(n\log n)$

Loading问题是**0-1**背包问题的特例：

即$v_i=1,\ i=1,2,\ldots,n.$

该问题是$O(n\log n)$时间可解的

0-1背包问题是**NP**难的。

贪心选择性质的证明

交换论证

例3　最小延迟调度问题

任务集合S，$\forall i \in S$，d_i为截止时间，t_i为加工时间，

d_i, t_i为正整数.

一个调度$f: S \to N$，$f(i)$为任务i 的开始时间. 求最大延迟达到最小的调度，即求 f 使得

$$\min_{f}\{\max_{i \in S}\{f(i)+t_i-d_i\}\}$$

$$\forall i, j \in S, i \neq j, f(i)+t_i \leq f(j) \text{ or } f(j)+t_j \leq f(i)$$

贪心策略选择

贪心策略1：按照 t_i 从小到大安排任务

贪心策略2：按照 $d_i - t_i$ 从小到大安排任务

贪心策略3：按照 d_i 从小到大安排任务

策略1 对某些实例得不到最优解.

反例：$t_1=1, d_1=100, t_2=10, d_2=10$

策略2 对某些实例得不到最优解.

反例：$t_1=1, d_1=2, t_2=10, d_2=10$

算法设计

1. **Sort(*S*)使得$d_1 \le d_2 \le \ldots \le d_n$**
2. **$f(1) \leftarrow 0$，$i \leftarrow 2$**
3. **while $i \le n$ do**
4. **$f(i) \leftarrow f(i-1)+t_{i-1}$**
5. **$i \leftarrow i+1$**

按照截止时间d_i从小到大选择

交换论证——正确性证明

算法的解的性质：

没有空闲时间，没有逆序.

逆序(i,j): $f(i)<f(j)$ and $d_i>d_j$

命题1 所有没有逆序、没有空闲时间的调度具有相同的最大延迟.

因为f_1与f_2都没有逆序，具有相同截止时间的任务必须被连续安排. 在这些任务中最大延迟是最后一个任务，被延迟的时间只与已安排任务加工时间之和有关，与任务标号无关.

交换论证：从一个没有空闲时间的最优解出发，在不改变最优性的条件下，转变成没有逆序的解.

(1) 如果一个最优调度存在逆序，那么存在$i<n$使得$(i,i+1)$构成一个逆序.

(2) 存在逆序(i,j)，$j=i+1$，那么交换i和j 得到的解的逆序数减1，后面证明这个新的调度仍旧最优.

(3) 至多经过$n(n-1)/2$次交换得到一个没有逆序的最优调度.

证明：调度 f_1, (i,j) 构成逆序, $j=i+1$, $d_i>d_j$
交换 i与 j 得到 f_2, f_2比 f_1不增加最大延迟时间

(1) 交换 i, j 对其他任务的延迟时间没影响

(2) 交换后不增加 j 的延迟

(3) 任务i 在 f_2 的延迟 L_{2i} 小于任务 j 在 f_1的延迟 L_{1j}, 因此小于f_1的最大延迟

$f_1(i)=s$　　$f_1(j)=s+t_i$　　$f_1(j)+t_j=s+t_i+t_j$

$f_2(j)=s$　　$f_2(i)=s+t_j$　　$f_2(i)=s+t_j+t_i$

i 在 f_2 结束时间 = j 在 f_1 的结束时间 = $s+t_i+t_j$

j 在 f_1 的延迟: $L_{1j}=(s+t_i+t_j)-d_j$

i 在 f_2 的延迟: $L_{2i}=(s+t_i+t_j)-d_i$

$d_j < d_i \Rightarrow L_{2i} < L_{1j}$

贪心法得不到最优解的处理办法

讨论对于哪些输入贪心选择能够得到最优解

输入应该满足的条件

讨论贪心法的解最坏情况下与最优解的误差

绝对误差与相对误差估计

确定得到最优解的输入所满足的条件

例4 找零钱问题

设有n种零钱,

重量分别为 $w_1, w_2, \cdots, w_n$,

价值分别为 $v_1=1, v_2, \cdots, v_n$.

付的总钱数是 y

如何付钱使得所付钱的总重最轻?

动态规划算法

属于整数规划问题

动态规划算法可以得到最优解

设$F_k(y)$表示用前k种零钱，总钱数为y的最小重量

递推方程

$$F_{k+1}(y)=\min_{0\le x_{k+1}\le\left\lfloor \frac{y}{v_{k+1}} \right\rfloor}\{F_k(y-v_{k+1}x_{k+1})+w_{k+1}x_{k+1}\}$$

$$F_1(y)=w_1\left\lfloor \frac{y}{v_1} \right\rfloor=w_1y$$

Greedy算法

假设

$$\frac{w_1}{v_1} \geq \frac{w_2}{v_2} \geq \ldots \geq \frac{w_n}{v_n}$$

使用前k种零钱，总钱数为y
贪心法的总重为$G_k(y)$，则有如下递推方程

$$G_{k+1}(y) = w_{k+1}\left\lfloor \frac{y}{v_{k+1}} \right\rfloor + G_k(y \bmod v_{k+1})$$

$$G_1(y) = w_1\left\lfloor \frac{y}{v_1} \right\rfloor = w_1 y$$

n=2时得到最优解

$F_1(y) = G_1(y)$ ， $F_2(y) = G_2(y)$

$$
\begin{aligned}
&[F_1(y - v_2(x_2 + \delta)) + w_2(x_2 + \delta)] \\
&-[F_1(y - v_2 x_2) + w_2 x_2] \\
=&[w_1(y - v_2 x_2 - v_2\delta) + w_2 x_2 + w_2\delta] \\
&-[w_1(y - v_2 x_2) + w_2 x_2] \\
=&-w_1 v_2\delta + w_2\delta = \delta(-w_1 v_2 + w_2) \le 0
\end{aligned}
$$

$n>2$时得到最优解的判定条件

定理2 假定 $G_k(y)=F_k(y)$,

$v_{k+1}>v_k$ 且 $v_{k+1}=pv_k-\delta$, $0\le\delta<v_k$, $p\in Z^+$,

则以下命题等价.

(1) $G_{k+1}(y)\le G_k(y)$

(2) $G_{k+1}(y)=F_{k+1}(y)$

(3) $G_{k+1}(pv_k)=F_{k+1}(pv_k)$

(4) $w_{k+1}+G_k(\delta)\le pw_k$

用条件(4)需$O(k)$时间验证$G_{k+1}(y)=F_{k+1}(y)$?

对n种零钱作出验证, 可在$O(n^2)$时间内完成

实　例

例　$v_1=1, v_2=5, v_3=14, v_4=18, w_i=1, i=1, 2, 3, 4.$

对一切 y 有$G_1(y)=F_1(y), G_2(y)=F_2(y).$

验证　$G_3(y) = F_3(y)$

$$v_{k+1} = pv_k - \delta,\ 0 \le \delta < v_k,\ p \in Z^+,$$

$v_3=14=3\ v_2-1,$　即 $p=3, \delta=1.$

$$w_{k+1} + G_k(\delta) \le pw_k$$

$w_3+G_2(\delta)=1+ G_2(1) =1+1 = 2$

$pw_2=3\times 1=3$

$w_3+G_2(\delta)\le p\ w_2$

例　$v_1=1, v_2=5, v_3=14, v_4=18, w_i=1, i=1, 2, 3, 4.$

$v_{k+1} = pv_k - \delta, 0 \le \delta < v_k, p \in Z^+,$

$$w_{k+1} + G_k(\delta) \le pw_k$$

$v_4=18=2v_3-10$, 即 $p=2, \delta=10.$

$w_4+G_3(\delta)=1+G_3(10)=1+2=3$

$pw_3=2\times 1=2,$　　$w_4+G_3(\delta)>pw_3$

$G_4(y)$不是最优解. $G_4(pv_3) > F_4(pv_3)$. 即

$G_4(28)=\lfloor 28/18 \rfloor + \lfloor 10/5 \rfloor = 1+2 = 3$

$F_4(28)=28/14 = 2.$

近似解的误差估计

例5　装箱问题

有n个物体, 长度分别为$a_1,a_2,...,a_n$, $a_i \le 1, i=1,2,\ldots,n$.

要把它们装入长为1的箱子(只考虑长度的限制)

问至少需要多少只箱子?

$$\min \ m$$

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} C(B_j), \qquad C(B_j) \le 1, j = 1,2,...,m$$

$C(B_j)$ 称为箱子 B_j 的装入量

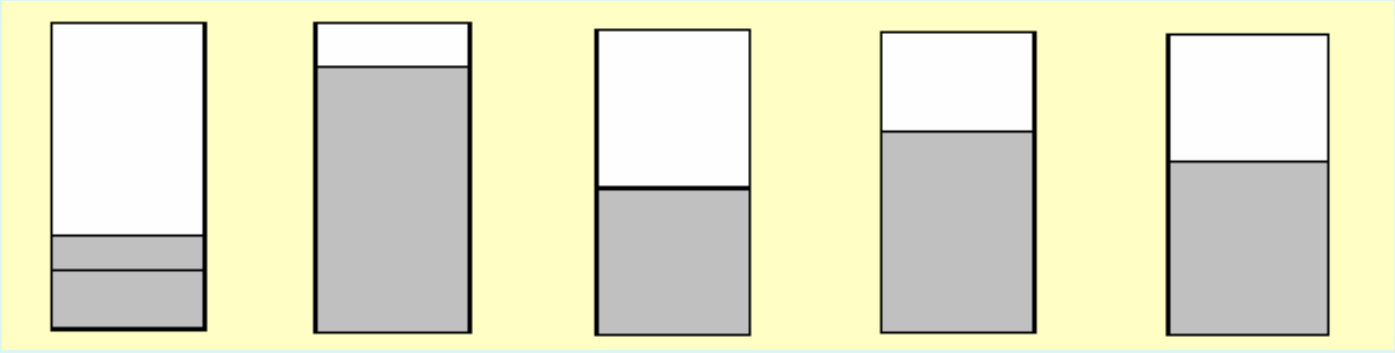
$1—C(B_j)$ 称为箱子 B_j 的空隙

下次适合法*NF*

算法1 *NF*

1. 初始$k \leftarrow 1$; 箱子号$j \leftarrow 1$; $C(B_j) \leftarrow 0$
2. **while** $k \le n$ **do**
3. **if** $a_k \le 1 - C(B_j)$
4. **then** $C(B_j) \leftarrow C(B_j) + a_k$
5. **else** $j \leftarrow j+1$; $C(B_j) \leftarrow a_k$
6. $k \leftarrow k+1$

实例：输入 { 0.2, 0.1, 0.9, 0.4, 0.7, 0.6 }



误差估计

L　　输入

L^*　　根据最优算法所得到的箱子数

$NF(L)$　NF算法所得到的箱子数

$r(NF)$　NF算法相对于最优算法的误差

$$r(NF) = \lim_{k \to \infty} \left[\max_{L^* = k} \frac{NF(L)}{L^*} \right]$$

可以证明 $r(NF) = 2$.

误差估计（续）

上界

任取箱子$B_j, j<m$, 则有　$C(B_j)+C(B_{j+1})>1$

所以必有　$C(B_1) + C(B_2) + ... + C(B_m) >(m-1)/2$

$$\left[\begin{array}{l} m\text{为奇数}\quad C(B_1)+...+C(B_m)>C(B_1)+...+C(B_{m-1})>\frac{m-1}{2} \\ m\text{为偶数}\quad C(B_1)+...+C(B_m)>\frac{m}{2}>\frac{m-1}{2} \end{array}\right]$$

物体总体积　　大于$(m-1)/2$

最优的算法　　$L^* >(m-1)/2$

$NF(L)$　　m

$NF(L)< 2L^* + 1,$

令$L^*\to\infty$得　　$r(NF)\le 2$

下界：找紧的实例

设计某个输入 L，使得 $NF(L)$ 是 L^*的2倍

$$L = \{\underbrace{\frac{1}{2}, \frac{1}{2N}, \frac{1}{2}, \frac{1}{2N}, \cdots, \frac{1}{2}, \frac{1}{2N}, \frac{1}{2}}_{2N\text{个}\frac{1}{2},\ 2N-1\text{个}\frac{1}{2N}}\}$$

$L^*=N+1$

每两个 1/2 的物体装一箱，用N 个箱子

剩下 $2N-1$个$(1/2N)$ 的物体放入1箱

$NF(L)=2N$

对于任意大的 L^*,存在某个输入使得 $2L^*-2 \le NF(L)$

$$r(NF) \ge \lim_{L^* \to \infty} \frac{2L^*-2}{L^*} = 2$$

首次适合法FF

算法2

对于物体 a_k, 依次检查 $B_1, B_2, \ldots$ 的空隙，找到第一个能够放入 a_k 的箱子 B_j, 就把 a_k放入. 即如果 a_k装入 B_j, 则

$$1-C(B_j)\geq a_k,$$

$$1-C(B_i)< a_k,\ \forall i<j$$

对于FF 算法有

$$\frac{17}{10}L^*-2\leq FF(L)\leq\frac{17}{10}L^*+2$$

即 $r(FF)=1.7$

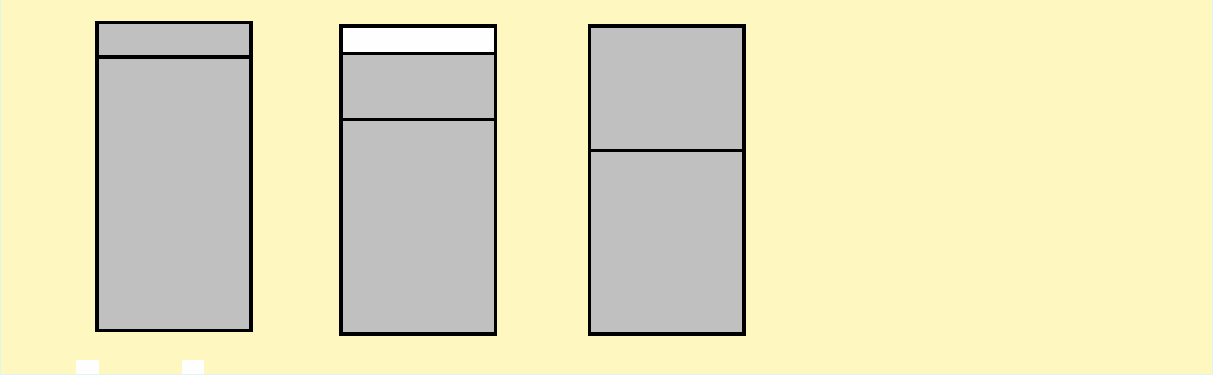
递降首次适合法FFD

算法3 1. 排序L中的物体使得 $a_1 \geq a_2 \geq \ldots \geq a_n$

2. 使用FF 算法.

实例：{ 0.2，0.1，0.9，0.4，0.7，0.6 }

{ 0.9，0.7，0.6，0.4，0.2，0.1}



$$\frac{11}{9}L^* \leq FFD(L) \leq \frac{11}{9}L^* + 4 \quad \text{即} \quad r(FFD) = 1.2$$

例6　最优二元前缀码问题

前缀码：用**0-1**字符串作为代码表示字符，要求任何字符的代码都不能作为其它字符代码的前缀

非前缀码　***a*---001, *b*---00, *c*---010, *d*---01**

0100001:　01, 00, 001　*d, b, a*

010, 00, 01　*c, b, d*

前缀码的存储采用二叉树的结构，每个字符作为树叶，每个前缀码看作根到树叶的路径

输入：***n*个字符$x_1, x_2, \ldots, x_n$,**

每个字符传输概率 $f(x_i), i=1, 2, \ldots, n.$

求：前缀码，使得平均传输一个字符的位数达到最小

算法：**Huffman**树得到最优解

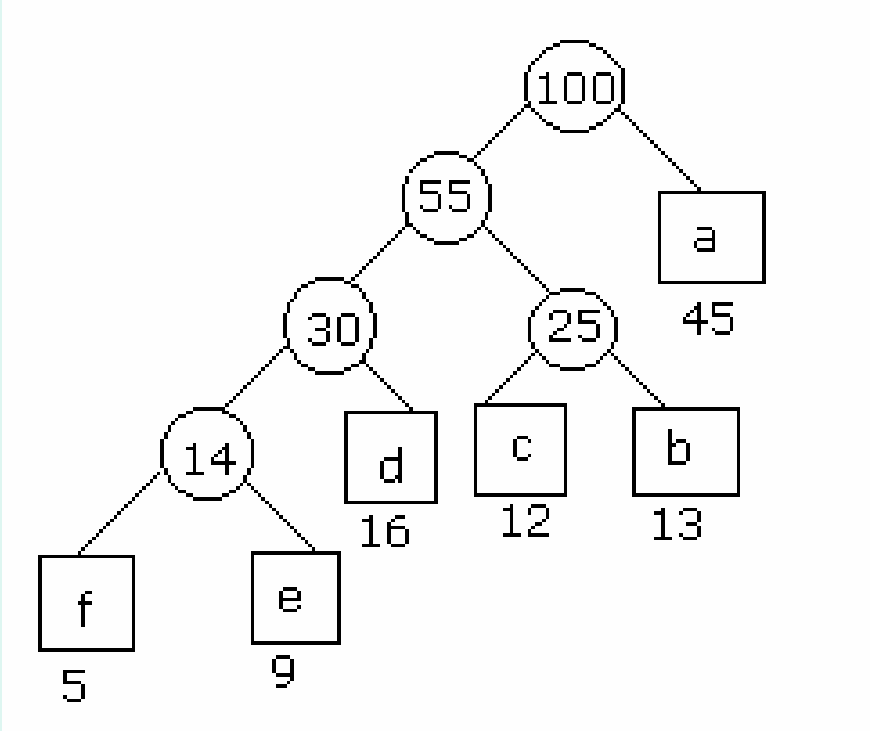
算法 Huffman(*C*)

```
1. n←|C|;
2. Q←C;                          //按频率递增构成队列Q
3. for i←1 to n-1 do
4.    z←Allocate-Node()          //生成结点z
5.    z.left←Q中最小元           //取出Q中最小元作为z的左儿子
6.    z.right←Q中最小元          //取出Q中最小元作为z的右儿子
7.    f[z]←f[x]+f[y]
8.    Insert(Q,z);               //将z插入Q , O(logn)
9. Return Q
```

时间：$O(n\log n)$

例如 ***a*:45, *b*:13; *c*:12; *d*:16; *e*:9; *f*:5,**

Huffman树为



编码:

***f*--0000,**

***e*--0001,**

***d*--001,**

***c*--010,**

***b*—011,**

***a*--1**

平均位数:

4*5%+4*9%+3*16%+3*12%

+3*13%+1*45% = 2.24

正确性证明

- 证明存在一个对应于最优前缀码的二叉树，以最小的频率作为树叶，且这两片树叶是兄弟
- 证明 x, y 是树叶兄弟，z 是 x, y 的父亲，

  z 的频率 $f[z]=f[x]+f[y]$, 令$T' = T-\{x,y\}$,

  则 T'是对应于最优前缀码

  $$C' = (C-\{x,y\})\cup\{z\}$$

  的树

交换论证

引理1

设C是字符集，$\forall c \in C$, $f[c]$为频率，$x,y \in C$, $f[x]$,$f[y]$频率最小，那么存在最优二元前缀码使得 x, y的码字等长，且仅在最后一位不同.

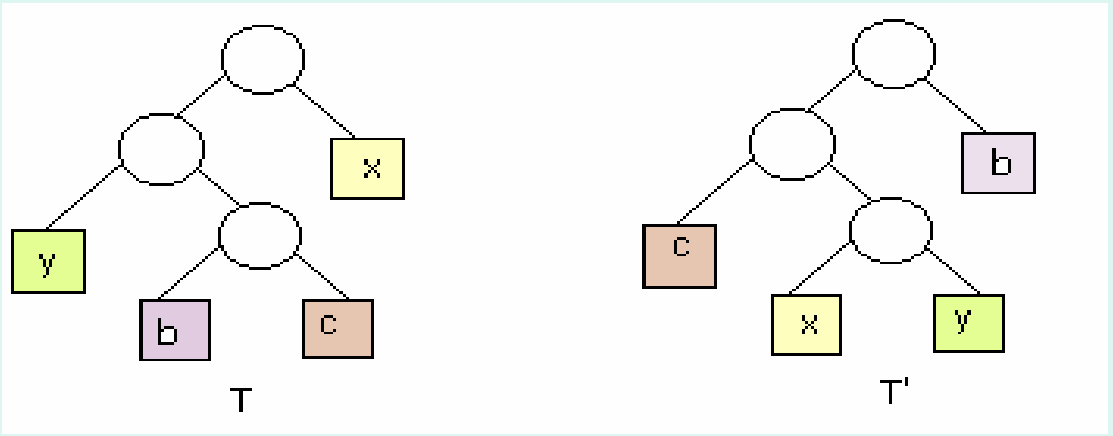
$T \rightarrow T'$

$f[y] \le f[c]$

$f[x] \le f[b]$

b与 x 交换

c与 y 交换



则T与T'的权之差为

$$B(T) - B(T') = \sum_{i \in C} f[i] d_T(i) - \sum_{i \in C} f[i] d_{T'}(i) \ge 0$$

其中$d_T(i)$为i在T中的层数（i到根的距离）

引理2 设T是最优解对应的树，$\forall x,y\in T$, x,y是树叶兄弟，z是x,y的父亲，z的频率$f[z]=f[x]+f[y]$, 令$T' = T-\{x,y\}$, 则T'是对应于最优前缀码$C' =(C-\{x,y\})\cup\{z\}$的树

证明：$\forall c\in C-\{x,y\}$,有 $d_T(c)=d_{T'}(c) \Rightarrow f[c]d_T(c)=f[c]d_{T'}(c)$

由 $d_T(x) = d_T(y) = d_{T'}(z) +1$

得 $f[x]\, d_T(x) + f[y]\, d_T(y) = (f[x] + f[y])\, (d_{T'}(z) + 1)$

$= f[z]d_{T'}(z) + (f[x] + f[y])$

从而 $B(T) = B(T') + f[x] + f[y]$

若T'不是关于C'的最优树，那么存在T^*使得 $B(T^*)<B(T')$, z是T^*中的树叶，将x,y加到z上作为儿子，那么得到

$B(T^*) + f[x] + f[y] < B(T)$

与T 的最优性矛盾.

定理3 Haffman算法得到最优前缀码

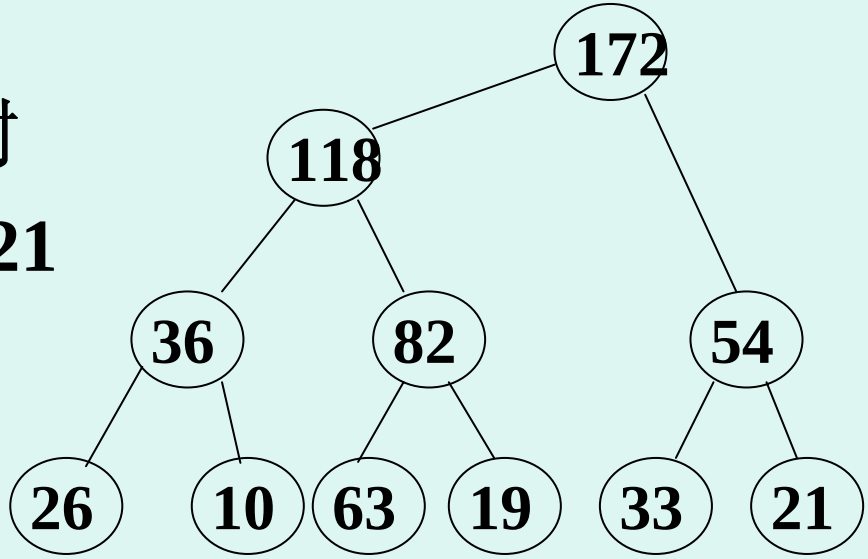
例7　文件归并

给定一组不同长度的排好序文件构成的集合 $S=\{f_1,\ldots,f_n\}$. 其中f_i表示第 i个文件含有的项数.

使用二分归并将这些文件归并成一个有序的文件. 找到一个比较次数最少的归并次序.

归并过程对应于二叉树

实例： **26,10,63,19,33,21**



归并的代价

归并树叶 f_i 和 f_j, 代价是 $|f_i| + |f_j|$.
$C(T)$ 是树的内结点的权之和

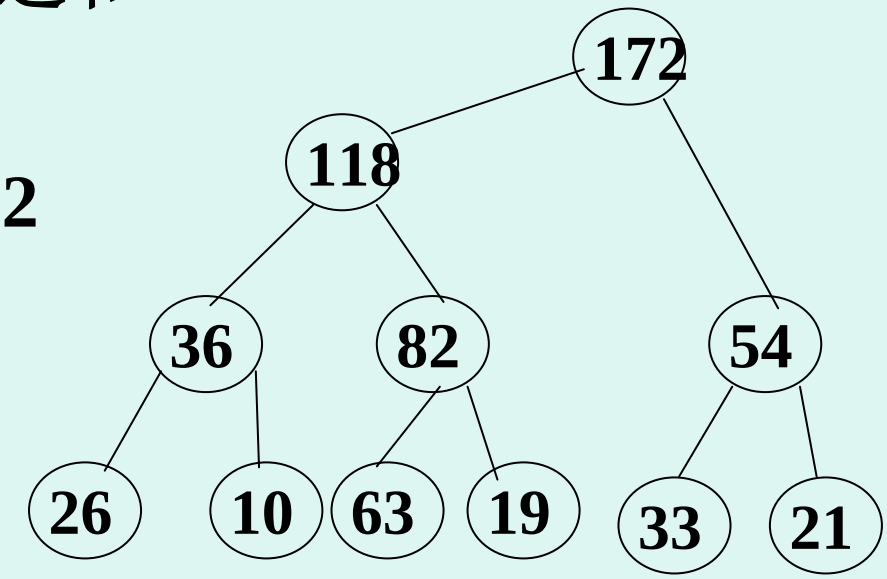
实例：

$C(T)=36+82+54+118+172$

$=(26+10+63+19)\times 3$

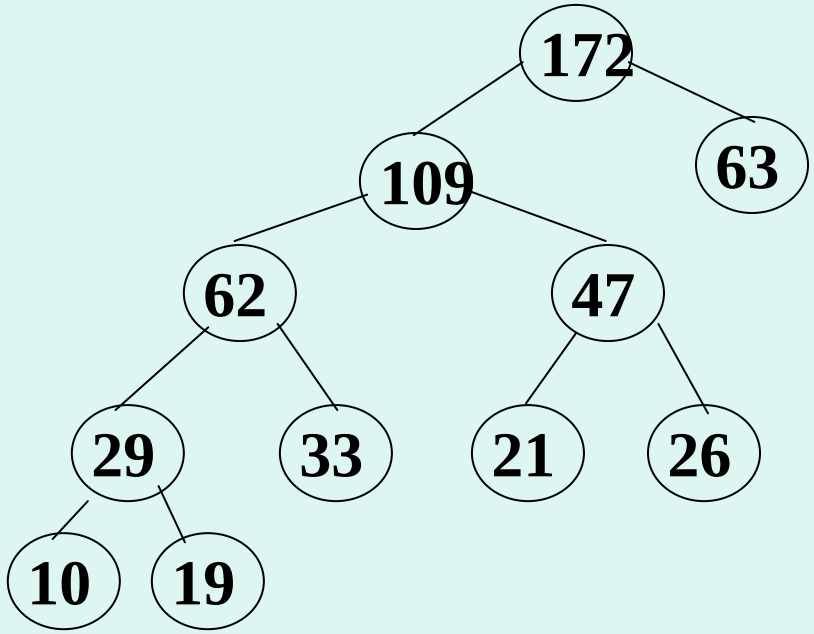
$+(33+21)\times 2$

$=462$



$$C(T)=\sum_{k=1}^{n}|f_k|\times depth(f_k)$$

更好的归并方法



(10+19)×4+(33+21+26)×3+63

=116+240+63=419

归纳证明

归纳基础：存在最优解，首次归并项数最少的两个文件

归纳步骤：

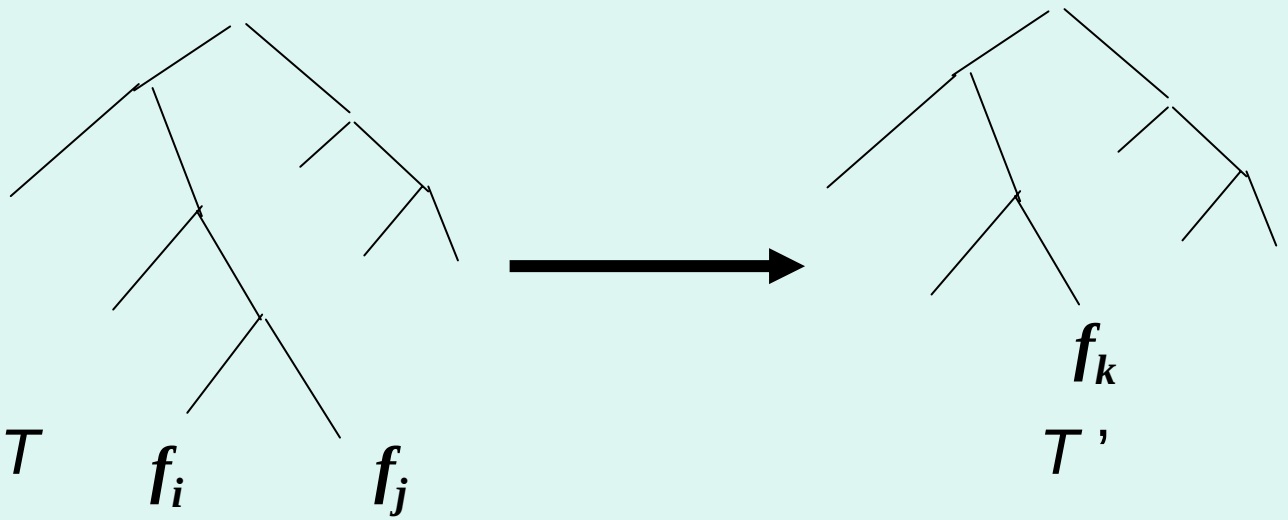
原问题：$\{f_1, \dots, f_n\}$，最优解T

子问题：$\{f_k\} \cup (\{f_1, \dots, f_n\} - \{f_i, f_j\})$，$T$ 的子树T'

$C(T)=C(T')+f_i+f_j$

如果T'不是$\{f_k\} \cup (\{f_1, \dots, f_n\} - \{f_i, f_j\})$ 的最优解，那么用更优的解T^*替代，从而得到原问题的最优解，代价

$C(T^*)+f_i+f_j<C(T)$



算　法

- **Huffman**树的算法
- 时间$O(n\log n)$

例8　最小生成树

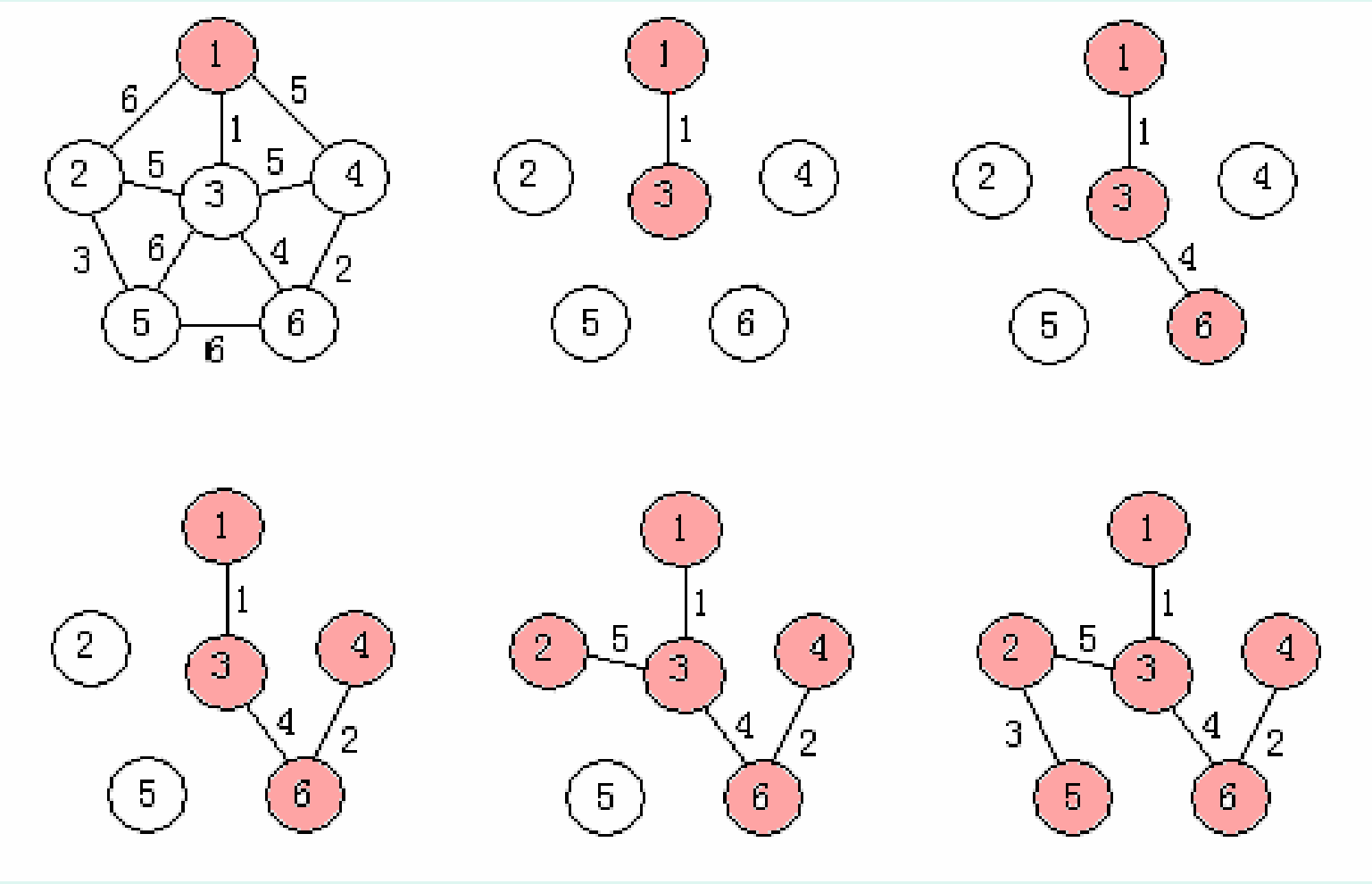
无向连通带权图$G=(V,E,W)$

使得$W(T)$最小的生成树T

算法 Prim(G,E,W)

1. $S\leftarrow\{1\}$
2. while $V-S\neq\varnothing$ do
3. 　从$V-S$中选择 j 使得 j 到S中顶点的边权最小
4. 　$S\leftarrow S\cup\{j\}$

实 例


1
6
5
1
2
5
5
4
3
6
4
2
5
6
6
1
1
2
3
4
5
6
1
1
2
3
4
4
5
6
1
1
2
3
4
4
2
5
6
1
1
2
5
3
4
4
2
5
6
1
1
2
5
3
4
3
4
2
5
6

Prim算法的正确性证明

对步数归纳

命题：对于任意$k<n$, 存在一棵最小生成树包含算法前k步选择的边

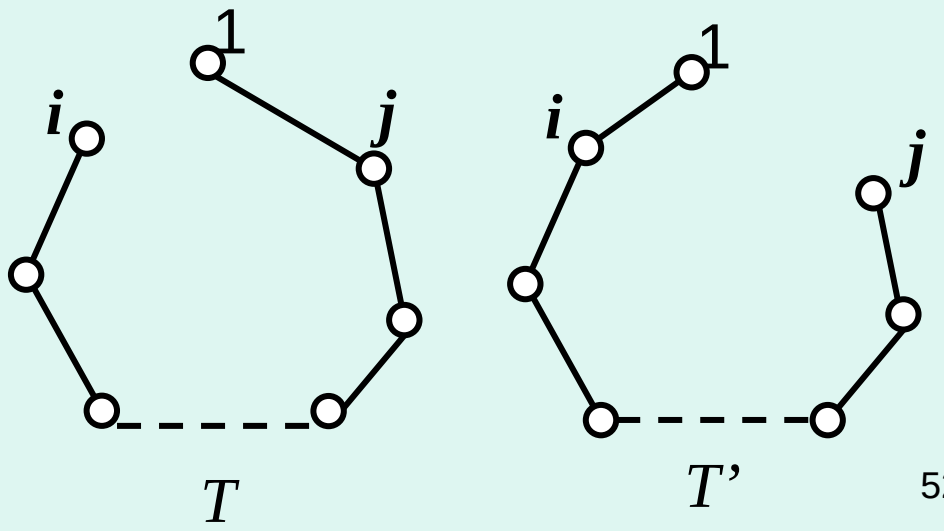
k=1, 存在一棵最小生成树T包含边e={1,i}, 其中{1,i}是所有关联 1 的边中权最小的.

设T 为一棵最小生成树，假设T 不包含{1,i}, 则$T\cup$\{{1,i}\}含有一条回路，回路中关联1的另一条边为{1,j}，

令 $T'=(T-\{\{1,j\}\})\cup\{\{1,i\}\}$，

则T'也是生成树，

且$W(T')\leq W(T)$.



Prim算法的正确性证明（续）

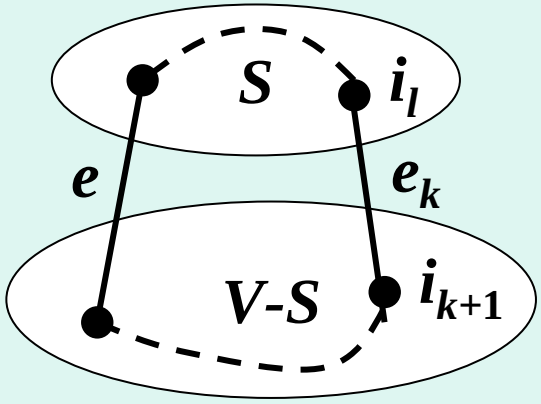
归纳步骤

假设算法进行了k-1步，生成树的边为$e_1,e_2,\ldots,e_{k-1}$，这些边的k个端点构成集合S. 由归纳假设存在G 的一棵最小生成树T 包含这些边.

算法第k 步选择了顶点 i_{k+1}, 则 i_{k+1}到S中顶点的边权最小, 设这条边为 $e_k=\{i_{k+1},i_l\}$. 假设T不含有e_k，则将 e_k 加到T中形成一条回路. 这条回路有另外一条连接S与V-S中顶点的边e, 令$T^*=(T-\{e\})\cup\{e_k\}$,

则T^*是G 的一棵生成树，包含$e_1,e_2, \ldots, e_k$, $W(T^*)\le W(T)$.

时间$T(n)=O(n^2)$

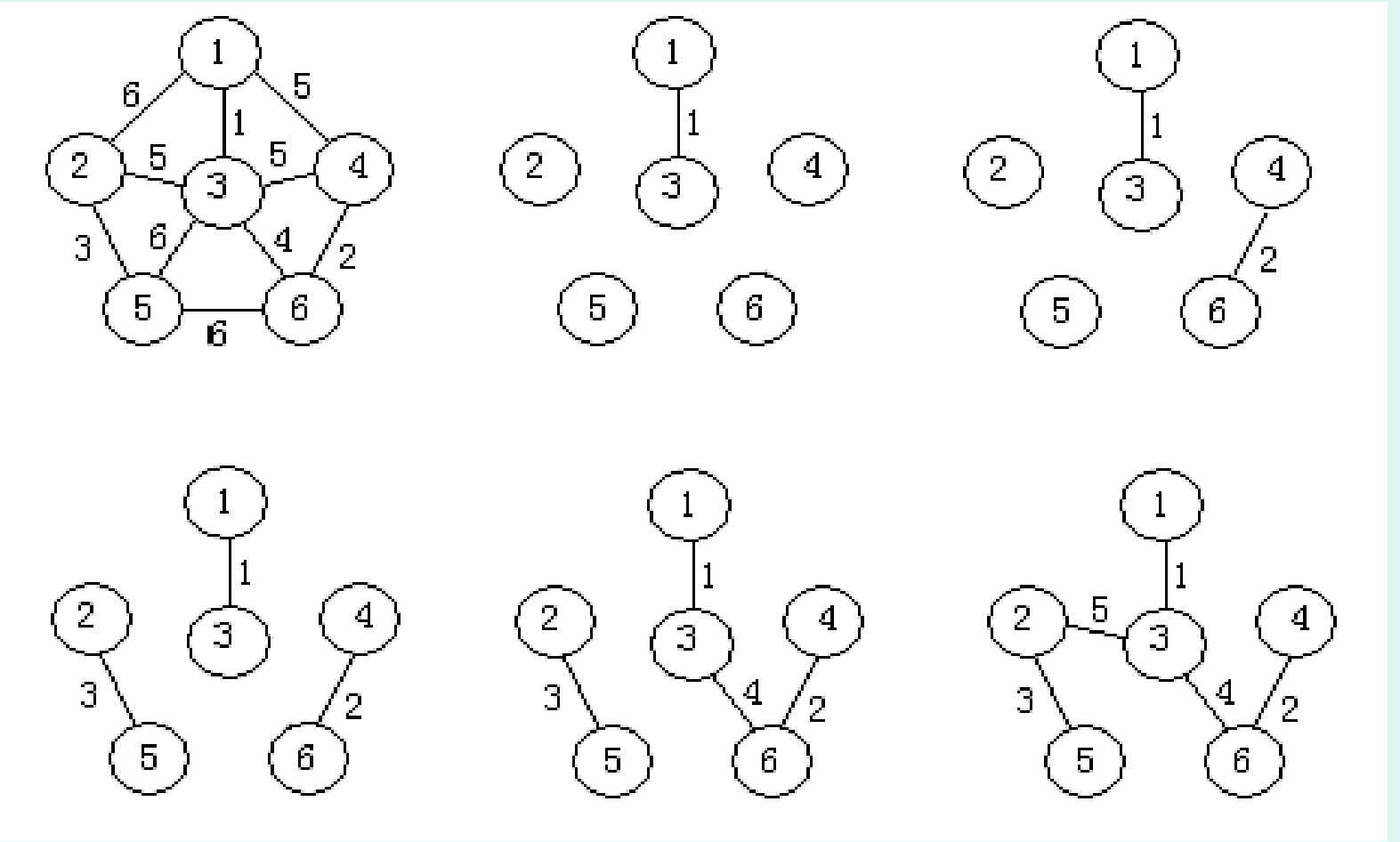


算法 Kruskal

算法 Kruskal（G,W）

1. 按照权从小到大顺序排序 G 中的边 $\{e_1, e_2, \ldots, e_m\}$
2. **for** $i \leftarrow 1$ **to** m **do**
3. 　　如果 e_i 的两个端点不在同一个连通分支，则将 e_i 加到 T 中.

实　例


1
6
5
1
2
5
5
4
3
6
4
2
3
5
6
6
1
1
2
3
4
5
6
1
1
2
3
4
2
5
6
1
1
2
3
4
3
2
5
6
1
1
2
3
4
3
4
2
5
6
1
1
2
5
3
4
3
4
2
5
6

Kruskal算法的正确性证明

对顶点数归纳

命题：对于任意n, 使用Kruskal算法对于n阶图得到一棵最小生成树.

证明 n=2, 只有一条边，命题显然为真。

假设对于n个顶点的图算法正确，考虑n+1个顶点的图G, G中最小权边 $e = \{i, j\}$。

从G中短接 i 和 j，得到图G'为n个顶点的图. 根据归纳假设，由算法存在G'的最小生成树T'.令$T=T'\cup\{e\}$, 则T是关于G的最小生成树.

Kruskal算法正确性证明(续)

若不然，存在一棵G的最小生成树T^*，$W(T^*)<W(T)$. 如果 e 属于T^*, 则短接 e 得到G'的生成树$T^*-\{e\}$, 且$W(T^*-\{e\})<W(T')$, 与T'的最优性矛盾；如果e不属于T^*, 在T^*中加上边e，形成回路.去掉回路中任意别的边所得生成树的权小于$W(T^*)$，与$W(T^*)$的最小性矛盾.

时间：$T(m)=O(m\log m)$

例9 单源最短路径

给定带权图$G=(V,E)$,每条边的权为道路长度（非负实数）.

源$s\in V$，求从s出发到达其它结点的最短路径.

Dijkstra算法：

$x\in S \Leftrightarrow x\in V$ 且从 s 到 x 的最短路径长度已知

初始：$S=\{s\}$，$S=V$ 时算法结束

从s到u相对于S的最短路径：从 s 到 u且仅经过 S 中顶点的最短路径

$dist[u]$：从 s 到 u 的相对于S 的最短路径的长度

Dijkstra算法

Dijkstra（*G*,*E*,*W*）

1. ***S*={*s*}; *dist*[*s*]←0**
2. **for *i*∈*V*–{*s*} do**
3. ***dist*[*i*]←*w*(*s*,*i*)** //如果*s* 到 *i* 没有边,*w*(*s*,*i*)=∞
4. **while *V*-*S*≠∅ do**
5. 从*V*-*S*中取出相对*S*具有最短路径的顶点 *j*
6. ***S*←*S*∪{*j*};**
7. **for *i*∈*V*–*S* do**
8. ***dist*[*i*]←min{*dist*[*i*], *dist*[*j*]+*w*(*j*,*i*)}**

实　例

***S*={1},**

从1到2相对***S***的最短路径：

1-2，*dist*[2]=10

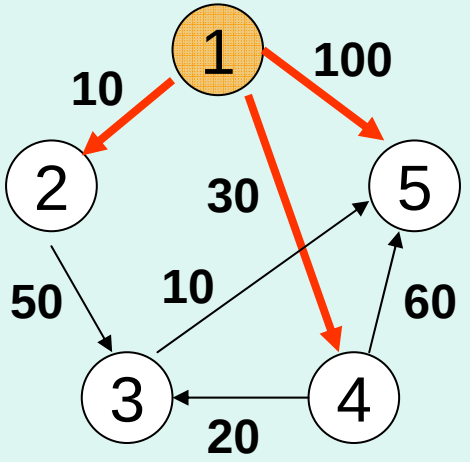
从1到3相对***S***的最短路径：无

从1到4相对***S***的最短路径：

1-4，*dist*[4]=30

从1到5相对于***S***的最短路径：

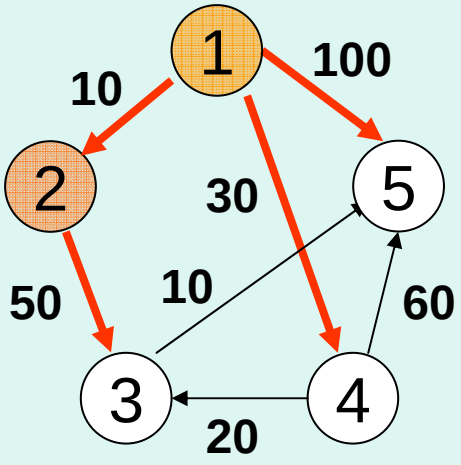
1-5，*dist*[5]=100



S={1,2},

修改从1到3相对***S***的最短路径：

1-2-3，*dist*[3]=60



S={1,2,4},

修改1到3相对S的最短路径：

1-4-3，*dist*[3]=50

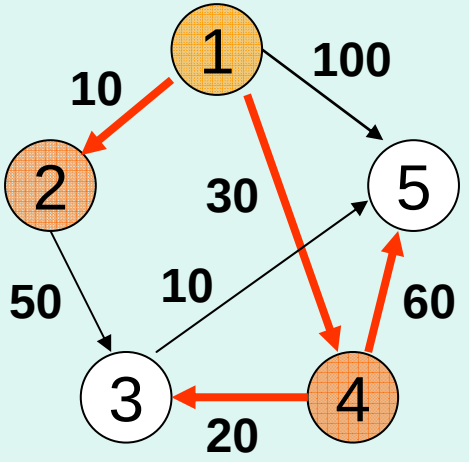
修改1到5相对S的最短路径：

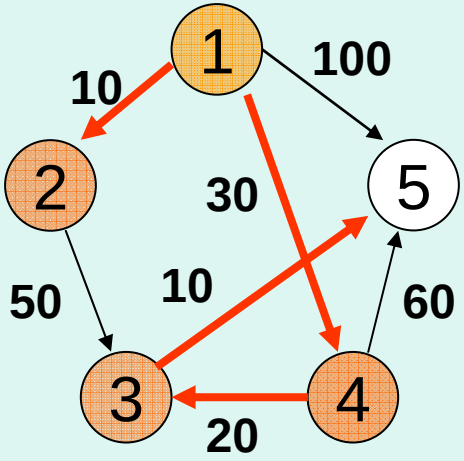
1-4-5，*dist*[5]=90

S={1,2,4,3}

修改1到5的相对S的最短路径：

1-4-3-5，*dist*[5]=60





Dijkstra算法正确性证明

Short[*i*] 表示从 ***s*** 到 ***i*** 的最短路径长度
dist[*i*] 表示相对于***S*** 的从 ***s*** 到 ***i*** 的最短路径长度

命题：当算法进行到第 ***k*** 步时，对于***S*** 中每个结点 ***i***， **dist[*i*] = short[*i*]**

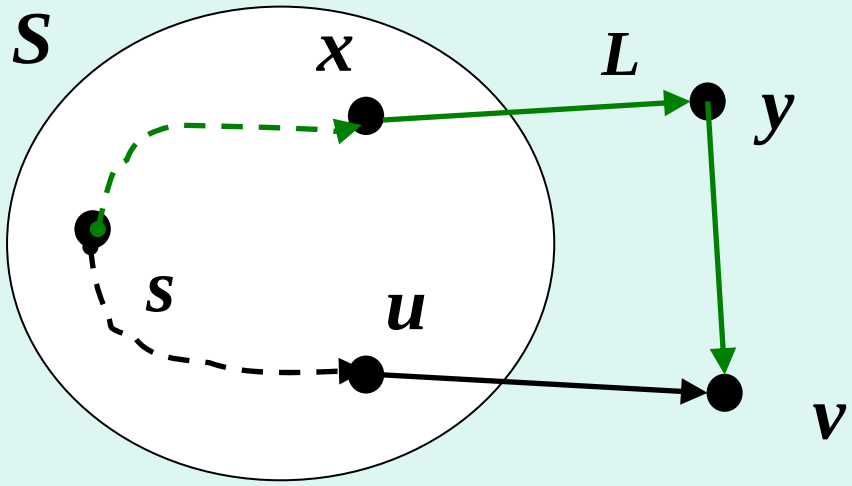
归纳基础
***k*=1, *S*={*s*}, dist[*s*]=short[*s*]=0,** 命题为真.

Dijkstra算法正确性证明（续）

归纳步骤

假设命题对于k 为真. 考虑 k+1步, 选择顶点v（边{u,v}）.

假若存在另一条 s-v路径 L（绿色），最后一次出S 的顶点为 x, 在这次从S 中出来之后经过V–S 中的第一个顶点为 y.



$$\text{dist}[v] \le \text{dist}[y] \quad //v\text{先被选}$$
$$\le \text{dist}[y]+d(y,v)$$
$$\le L$$

$$\text{dist}[v]=\text{short}[v]$$

$$T(n)=O(n^2)$$

贪心法小结

设计要素

- 适用于满足优化原则的组合优化问题
  将问题表示成多步判断
- 确定一个优化测度---贪心选择的依据
- 确定是否满足贪心选择性质
  --每步贪心选择都会导致最优解
- 自顶向下计算

贪心法小结（续）

贪心算法的正确性证明

- 数学归纳法

  叙述一个可以归纳证明的命题：

  (1) 步数k归纳：对于任意k，k步贪心选择得到$i_1,i_2,\ldots,i_k$, 那么存在最优解包含$i_1,i_2,\ldots,i_k$

  (2) 问题规模k归纳：对于任意k,贪心法得到关于规模为k的问题的最优解

- 交换论证

贪心法小结（续）

贪心法得不到最优解的处理办法

讨论对于哪些输入贪心选择能够得到最优解

讨论贪心法的解最坏情况下与最优解的误差

贪心法的时间复杂度和空间复杂度低

概 率 算 法

随机数

概念

伪随机数生成算法

几种主要的概率算法

Sherwood算法

Las Vegas算法

Monte Carlo算法

伪 随 机 数

随机数与伪随机数

概率算法中要进行随机选择，需要大量随机数.

通常根据某种规则生成随机数，这些随机数不是真正随机的，但在一定程度上可以模拟随机数，一般叫做伪随机数

随机变量X的取值范围是$(0,1)$且对任意的$0<a<1$, $P\{0<X\leqslant a\}=a$, 则称X服从$(0,1)$上的均匀分布

伪随机数生成算法

伪随机数生成算法 ---线性同余法

生成伪随机序列为$\{a_i\}$, $i= 0,1,\ldots, n,\ldots,$

$0<a_i<m$

$$\begin{cases} a_0 = d \\ a_n = (ba_{n-1} + c) \bmod m \qquad n = 1,2,... \end{cases}$$

模数：m, 机器最大数.

乘数：b, $2\le b<m$，计算前给定

常数：c, $0\le c<m$，计算前给定

种子：d, $0\le d<m$, 计算时随机给出

乘同余法

$$\begin{cases} a_0 = d \\ a_n = ba_{n-1} \bmod m \qquad n = 1,2,... \end{cases}$$

参数：$c = 0$，$d \neq 0$，$m=2^{31}-1$，$b=7^5$

实例：$d=1$

16,807,
1,622,650,073,
1,144,108,930,
101,027,544,
……

282,475,249,
984,943,658,
470,211,272,
1,457,850,878

离散型伪随机数的产生

输入：$P=\{a_i, a_{i+1}, \ldots, a_j\}$,

输出：伪随机数$a_k \in P$

算法：**Random(*i*, *j*)**　// 产生$a_i,\ldots,a_j$之间随机数

1. 产生伪随机数x　　//乘同余法
2. $n \leftarrow j-i+1$　　//总个数n
3. **for** $k \leftarrow 1$ **to** n **do**　　// 检查第k个区间
4. 　　**if** $x/m \le k/n$ **then** $x \leftarrow a_{k+i-1}$　　// $0<x<m$
5. 　　**return**

确定型算法与随机算法

1. 特点：

确定型算法：　对某个特定输入的每次运行过程是可重复的，运行结果是一样的.

随机算法：对某个特定输入的每次运行过程是随机的，运行结果也可能是随机的.

2. 随机算法的优势：

在运行时间或者空间需求上随机算法比确定型算法往往有较好的改进

随机算法设计简单

随机算法复杂性度量

随机算法A对于规模为n的某个给定的实例I 的一次执行的时间与另一次可能不一样.

随机算法A对于某个规模为n的实例I 的运行时间的期望：算法A反复求解实例I 的平均时间.

一般情况下，许多随机算法对于规模为n的不同的实例，运行时间的期望都一样，因此这个值代表随机算法的平均时间的期望.

Sherwood算法

例1 快速排序的随机算法

```
算法 RandQuicksort(A,p,r)
  1. if  p<r  then
  2.     q←RandPartition(A,p,r)
  3.     RandQuicksort(A,p,q-1)
  4.     RandQuicksort(A,q+1,r)

算法 RandPartition(A,p,r)
  1. i←Random(p,r)
  2. A[i]↔A[p]
  3. return Partition(A,p,r)
```

算 法 比 较

确定型排序算法 **Quicksort**

最坏情况下时间为$O(n^2)$

平均情况下为 $O(n\log n)$

确定型排序+选择算法

如果选用中位数进行划分, 时间为$O(n\log n)$

随机快速排序算法

期望时间 $O(n\log n)$

最坏情况概率非常小，在实际应用中可以忽略

例2 随机选择算法

算法 RandSelect(*A*, *p*, *r*, *k*) //从*A*[*p*..*r*]中选第*k*小

1. if *p*=*r* then return *A*[*p*]

2. *i*←Random(*p*, *r*)

3. 以*A*[*i*]为标准划分*A*

4. *j*←划分后小于等于*A*[*i*]的数构成数组的大小

5. if *k*≤*j*

6. then return RandSelect(*A*, *p*, *p*+*j*-1, *k*)

7. else return RandSelect(*A*, *p*+*j*, *r*, *k*-*j*)

预期时间估计

假设随机选择时，1..n中每个数被选的概率相等，并且假设第k个数总是出现在划分后两个数组中较大的数组（最坏情况的上界）.那么，随机选择算法的预期时间为

$$T(n) \le \frac{1}{n}(T(n-1)+T(n-2)+\ldots+T(\frac{n}{2}+1)$$

$$+T(\frac{n}{2})+T(\frac{n}{2}+1)+\ldots+T(n-1))+O(n)$$

$$\le \frac{2}{n}\sum_{i=n/2}^{n-1}T(i)+O(n)$$

注意n为奇数，上式没有$T(n/2)$项

预期时间估计（续）

$$T(n) \le \frac{2}{n}\sum_{i=n/2}^{n-1} T(i) + tn$$

设$T(k) \le ck$ 对一切 $k < n$为真,则

$$T(n) \le \frac{2}{n}[c\frac{n}{2} + c(\frac{n}{2}+1) + ... + c(n-1)] + tn$$

$$= \frac{2c}{n}\frac{(\frac{n}{2}+n-1)\frac{n}{2}}{2} + tn$$

$$= \frac{c}{2}(\frac{3n}{2}-1) + tn = \frac{3cn}{4} - \frac{c}{2} + tn$$

$$\le \frac{3cn}{4} + tn = (\frac{3}{4}+\frac{t}{c})cn \le cn$$ 取$c \ge 4t$即可

算法比较

拟中位数选择算法

- 最坏情况 $O(n)$
- 平均情况 $O(n)$

随机算法

- 平均时间 $O(n)$
- 最坏情况 $O(n^2)$
  - 每次恰好选到边界元素
  - 与实例无关，只与选择有关
  - 概率很小，可以忽略
- 期望时间 $O(n)$

预期复杂度分析

设A为求解问题π的确定型算法，B为对应的Sherwood算法，对于输入x的计算时间分别记为$t_A(x)$, $t_B(x)$. 令

$$X_n=\{\, x \mid x\text{为}\pi\text{的实例}, |x|=n \,\}$$

输入等概率分布情况下，A的平均时间复杂度

$$\overline{t_A}(n)=\frac{1}{|X_n|}\sum_{x\in X_n} t_A(x)$$

B在实例x的时间期望 $t_B(x)=\overline{t_A}(n)+s(n)$

B 的平均复杂性

$$\overline{t_B}(n)=\frac{1}{|X_n|}\sum_{x\in X_n} t_B(x)=\overline{t_A}(n)+s(n)$$

Sherwood算法总结

算法能得到正确的解

算法比确定型算法简单

算法的平均性能与确定型算法一样

算法改善了最坏情况的期望运行时间

运行时间基本与输入实例无关

确定算法最坏情况在随机选择出现的概率接近0

实现途径

- 通过改进确定型算法得到
- 将确定型选择原则改为随机选择
- 在确定型算法前增加随机洗牌步骤

随机洗牌算法

算法 RandomShuffle(*A*,*n*)

1. for *i*←1 to *n*-1 do

2. *j*←Random(*i*,*n*)

3. *A*[*i*]↔*A*[*j*]

例如数组*A*： ***n*=12**

A	**2, 3, 1, 8, 6, 9, 11, 4, 5, 12, 10, 7**
***i*=1, *j*=4：**	**8, 3, 1, 2, 6, 9, 11, 4, 5, 12, 10, 7**
***i*=2, *j*=9：**	**8, 5, 1, 2, 6, 9, 11, 4, 3, 12, 10, 7**
***i*=3, *j*=8：**	**8, 5, 4, 2, 6, 9, 11, 1, 3, 12, 10, 7**

Las Vegas 算法

例3 *n*后问题

```
算法BoolQueen(n)
1. k←1                          // k放皇后的行号
2. count←0                      // count放好的皇后数
3. while k≤n do
4.     for i←1 to n do          // i为待选列号
5.         检查 i 与前面 k-1个皇后的相容性
6.         如果相容则将 i 加入S
7.     if S≠∅ then
8.         j←Random(1,|S|)
9.         x_k←S[j]
10.        count←count+1
11.        k←k+1
12.    else k←n+1
13. return count
```

*n*后问题的LV算法

```
算法QueenLV(n)  //重复调用随机算法BoolQueen
1. p←BoolQueen(n)
2. while p<n do
3.    p←BoolQueen(n)
```

改进算法---与回朔相结合

设 ***stopVegas*** ≤ ***n***, 表示用**QueenLV**算法放置的皇后数

剩下 ***n-stopVegas*** 个皇后用回朔方法放置

stopVegas = **0** 时是完全的回朔算法

stopVegas = ***n*** 时是完全的**Las Vegas**算法

改进的LV算法比较

对于不同的 ***stopVegas*** 值，设

p 为算法成功概率

s 为一次成功搜索访问的结点数的平均值

e 为一次不成功搜索访问的结点数的平均值

t 为算法找到一个解的平均时间

则 $$t = ps + (1-p)(e+t) \Rightarrow t = s + e\frac{1-p}{p}$$

***n*=12时的统计数据：*stopVegas* = 5时算法效率高**

stopVegas	*p*	*s*	*e*	*t*
0	1.0000	262.00	-	262.00
5	0.5039	33.88	47.23	80.39
12	0.0465	13.00	10.20	222.11

Las Vegas 算法总结

一次运行可能得不到解

得到的解一定是正确的

改进途径：与确定型算法相结合

改进确定型算法的平均情况下复杂度

修改确定性算法得到

Monte Carlo算法

例4 判断是否存在主元素

主元素：出现次数超过一半以上的元素

输入：n个元素的数组T

输出：如果存在输出“true”，否则“false”

算法 Majority(T,n)

1. $i \leftarrow$Random(1,n)
2. $x \leftarrow T(i)$
3. 计数 x 在 T 中出现的个数 k
4. if $k>n/2$ then return true
5. else return false

算法的正确性问题

如果回答**true:**

则***T***中一定存在主元素，算法给出正确的解；

如果回答**false:**

则***T***中不一定不存在主元素，算法可能出错.

回答**true**的概率大于**1/2**

偏真**1/2**正确的 **Monte Carlo**算法

重复调用 **2** 次可以提高算法正确性的概率

```
算法 BoolMajority(T,n)
  1. if Majority(T,n) then return true
  2. else return Majority(T,n)
```

改进途径

设算法 **Majority** 对于存在主元素的输入给出正确回答的概率为p, 算法**BoolMajority**的步1回答**true**的概率为p，进入步2的概率为$1-p$, 在步2回答**true**的概率为p, 因此**BoolMajority**算法回答**true**的概率为

$$p+(1-p)p=2p-p^2=1-(1-p)^2>\frac{3}{4}$$

如果调用k次，则正确概率为

$$p+(1-p)p+(1-p)^2p+\ldots+(1-p)^{k-1}p$$

$$=p[(1-p)^0+(1-p)^1+\ldots+(1-p)^{k-1}]$$

$$=p\frac{1-(1-p)^k}{1-(1-p)}=1-(1-p)^k>1-(\frac{1}{2})^k=1-2^{-k}$$

调用次数k	1	2	3	4	5	6
正确概率大于	0.5	0.75	0.875	0.938	0.969	0.985

改进途径（续）

对于任意给定的ε>0, 如果要使出错的概率不超过ε，则调用次数 k 满足

$$\left(\frac{1}{2}\right)^k \le \varepsilon \Rightarrow k\log\frac{1}{2} \le \log\varepsilon \Rightarrow -k \le \log\varepsilon$$

$$\Rightarrow k \ge -\log\varepsilon \Rightarrow k \ge \left\lceil \log\frac{1}{\varepsilon} \right\rceil$$

算法 MCMajority 调用 $k=\lceil \log(1/\varepsilon) \rceil$ 次 Majority

对于存在主元素的输入出错率不超过ε

```
算法 MCMajority(T,n,ε)
1. k←⌈log(1/ε)⌉
2. for i←1 to k
3.     if Majority(T,n) then return true
4. return false
```

例5 串相等测试

问题：*A*有一个长串*x*, *B*有长串*y*，*A*和*B*希望知道*x*=*y*?

方法一：

A 将 *x* 发送给 *B*，*B* 测试 *x* = *y*?

发送消耗：长串占用信道资源大

方法二：

A 用 *x* 导出一个短串 *f*(*x*)　(fingerprints)

A 将 *f*(*x*) 发送到 *B*

B 使用同样方法导出相对于 *y* 的短串 *f*(*y*)

B 比较 *f*(*x*) 与 *f*(*y*)

如果 $f(x) \neq f(y)$，则 $x \neq y$;

如果 $f(x) = f(y)$，则不确定.

fingerprints的产生方法

设 x 和 y 的二进制表示为正整数 $I(x), I(y)$

选择素数 p, **fingerprint**函数为

$$Ip(x) = I(x) \bmod p$$

A 传送 p 和 $Ip(x)$ 给 B. 当 p 不太大时，传送一个短串.

存在问题：

$$x = y \Rightarrow Ip(x) = Ip(y)$$

$$Ip(x) = Ip(y) \not\Rightarrow x = y$$

出错条件：固定

$$p \mid (I(x) - I(y))$$

改进算法的途径

改进方法： 随机选择素数 p 进行测试

算法 StringEqualityTest

1. 随机选择小于 M 的素数 p //M为正整数
2. A 发送 p 和 $Ip(x)$ 给 B
3. B 测试是否 $Ip(x) = Ip(y)$

出错必要条件：

x 的位数等于 y 的位数

$p \mid (I(x)-I(y))$

关于素数的结果

$\pi(t)$: 小于 t 的不同的素数个数

例　$\pi(20)=8$, 素数：2, 3, 5, 7, 11, 13, 17, 19

两个相关结果:

(1) 素数定理

$$\pi(t) \approx \frac{t}{\ln t}$$

(2) 若 $k<2^n$, n不太小，整除 k 的不同素数个数小于 $\pi(n)$

素 数 性 质

n	10^3	10^4	10^5	10^6	10^7
$\pi(n)$	168	1229	9592	78498	664579
$\frac{n}{\ln n}$	145	1086	8686	72382	620421
$\frac{\pi(n)}{n/\ln n}$	1.159	1.132	1.104	1.085	1.071

$k<2^{10}=1024$, 那么整除 k 的素数个数$<\pi(10)=4$,

例如$k=984$，那么整除984 的只有2, 3, 41

出错概率估计

n: x 和 y 的二进制表示的位数

$x, y \le 2^n$

若选择 $M \ge 2n^2$, 出错概率估计：

$$\frac{|\{p \mid I(x)-I(y)\text{小于}2^n，\text{且 } p \text{ 整除 } I(x)-I(y)\}|}{\pi(M)}$$

$$\le \frac{\pi(n)}{\pi(M)} \approx \frac{n/\ln n}{2n^2/\ln(2n^2)} \approx \frac{n/\ln n}{2n^2/2\ln n} \le \frac{1}{n}$$

改进算法

重复执行 j 次，每次随机选择小于 M 的素数

算法StringTest

输入：x, y，n位二进制数

输出：“Yes”(如果$x=y$); 或者“No” (如果$x \neq y$)

1. for $i \leftarrow 1$ to j
2. 随机选择小于 M 的素数 p // M为正整数
3. A 发送 p 和 $I_p(x)$ 给 B
4. B 测试
5. if $Ip(x) \neq Ip(y)$
6. then return “No”
7. return”Yes”

算法分析

令 $j = \lceil \log\log n \rceil$, 则算法出错的概率

$$(\frac{1}{n})^j \le \frac{1}{n^{\lceil \log\log n \rceil}}$$

实例：x 和 y 是1000000位二进制数

$n = 10^6$

$M = 2 \cdot 10^{12} = 2^{40.8631}$

素数 p 的二进制表示至多 $\lfloor \log M \rfloor + 1 = 41$ 位

$I(x)$ 的位数至多 $\lfloor \log(p-1) \rfloor + 1 \le \lfloor \log M \rfloor + 1 = 41$

总共传送82位

例6 模式匹配

输入：$X = x_1 x_2 \ldots x_n$

$Y = y_1 y_2 \ldots y_m$　　$m \le n$

输出：若Y在X中，Y出现的第一个位置；否则为“0”

不妨假设X, Y为二进制串

算法一：

初始Y与X的首元素对齐

依次从前到后比较X与Y的元素

如果X与Y的所有元素都相等，则输出Y的首位置j

否则将Y的位置向后移动一个单位，重复原来过程.

运行时间：$O(mn)$

其他算法

算法二 **Knuth, Morris, Pratt**

利用有限状态自动机模式匹配算法

Introduction to Algorithms

运行时间：$O(m+n)$

算法三 随机算法

设计思想：设 $X(j) = x_j x_{j+1} \cdots x_{j+m-1}$

改造算法一

不是比较每个 $X(j)$ $(j = 1, 2, \ldots, n-m+1)$ 与 Y，

而是将 $I_p(Y)$ 与 $I_p(X(j))$ 比较

X(j)与X(j+1)的关系

$$X(j)=2^{m-1}x_j+2^{m-2}x_{j+1}+\ldots+2x_{j+m-2}+x_{j+m-1}$$

$$X(j+1)=2^{m-1}x_{j+1}+2^{m-2}x_{j+2}+\ldots+2x_{j+m-1}+x_{j+m}$$

$$X(j+1)=2X(j)-2^m x_j+x_{j+m}$$

算法三的关键技术

由 $I_p(X(j))$ 求 $I_p(X(j+1))$ 的公式：

$$I_p(X(j+1)) = (2I_p(X(j)) - W_p x_j + x_{j+m}) (\mathrm{mod}\ p)$$

$$W_p = 2^m (\mathrm{mod}\ p)$$

导出：

$$X(j+1) = 2X(j) - 2^m x_j + x_{j+m}$$

$$I_p(X(j+1)) = (2X(j) - 2^m x_j + x_{j+m}) (\mathrm{mod}\ p)$$

令$W_p = 2^m (\mathrm{mod}\ p)$

$$I_p(X(j+1)) = (2I_p(X(j)) - W_p x_j + x_{j+m}) (\mathrm{mod}\ p)$$

算　法

算法 PatternMaching

输入：串 X 和 Y, $|X|=n$, $|Y|=m$, $m \le n$

输出：如果 Y 在 X 中，Y 出现的第一位置；否则为“0”

1. 从小于 M 的素数集合中随机选择素数 p
2. $j \leftarrow 1$
3. $W_p \leftarrow 2^m \pmod p$
4. $I_p(X(j)) \leftarrow I(X(j)) \pmod p$
5. $I_p(Y) \leftarrow I(Y) \pmod p$
6. while $j \le n-m+1$ do
7. 　　if $I_p(X(j)) = I_p(Y)$ then return j
8. 　　$I_p(X(j+1)) \leftarrow (2I_p(X(j)) - W_p x_j + x_{j+m}) \pmod p$
9. 　　$j \leftarrow j+1$
10. return 0

算法分析

时间复杂度

W_p, $I_p(Y)$, $I_p(X(1))$ 计算 $O(m)$ 时间

从 $I_p(X(j))$ 计算 $I_p(X(j+1))$ 总共需要$O(n)$时间

总时间为 $O(m+n)$

出错条件：

$$Y \neq X(j) \wedge I_p(Y) = I_p(X(j))$$

$$\Leftrightarrow p \mid \prod_{\{j \mid Y \neq X(j)\}} \mid I(Y) - I(X(j)) \mid$$

出错概率分析

乘积大小不超过 $(2^m)^n$

整除它的素数个数不超过 $\pi(mn)$

选 $M = 2mn^2$，则出错概率不超过

$$\frac{\pi(mn)}{\pi(M)} \approx \frac{mn/\ln(mn)}{2mn^2/\ln(mn^2)}$$

$$= \frac{\ln(mn^2)}{2n\ln(mn)} < \frac{\ln(mn)^2}{2n\ln(mn)} = \frac{1}{n}$$

例7 素数测试

- 求x的m次幂
- 求a的模n的m次幂
- **Fermart小定理**
- 测试算法分析

求 x 的 m 次幂

输入：x为实数

$m = d_k d_{k-1} \ldots d_1 d_0$ 为二进制自然数

输出：x^m

算法 **Exp(*x*,*m*)**

1. $y \leftarrow 1$;
2. for $j \leftarrow k$ downto 0 do
3. $y \leftarrow y^2$;
4. if $d_j = 1$ then $y \leftarrow xy$
5. return y

a 模 n 的 m 次幂

输入：$a, m, n \in Z^+, m \le n,$

$m = b_k b_{k-1} \ldots b_1 b_0$为二进制自然数

输出：$a^m \pmod n$

算法ExpMod(a, m, n)

1. $c \leftarrow 1$
2. for $j \leftarrow k$ downto 0 do
3. $\quad c \leftarrow c^2 \pmod n$
4. $\quad$ if $b_j = 1$ then $c \leftarrow ac \pmod n$
5. return c

$T(n) = O(k\log^2 n) = O(\log^3 n)$

Fermart 小定理

定理： 如果n为素数，则对所有的正整数

$a \neq 0 \pmod n$　　有　　$a^{n-1} \equiv 1 \pmod n$

证明： $a^{n-1} \equiv 1 \pmod n \Leftrightarrow n \mid (a^{n-1}-1) \Leftrightarrow n \mid (a^n - a)$

考虑用 a 种颜色涂色装有 n 颗珠子的手镯，若手镯只能旋转，则$|G|=n$,由于n 为素数，G 为循环群，除了恒等置换之外，G 的其他置换只含有一个轮换．根据Polya定理，不同的手镯数为

$$M = \frac{1}{n}(a^n + a + a + \ldots + a) = \frac{1}{n}[a^n + (n-1)a] = \frac{1}{n}(a^n - a) + a$$

$$\therefore \quad n \mid (a^n - a)$$

素数测试算法

检测 $2^{n-1} \equiv 1 (\mathrm{mod}\ n)$. 是，输出“素数”

否则输出“合数”.

算法 Ptest1(n)

输入： 奇整数n, $n>5$

输出： “prime” 或者 “composite”

1. if ExpMod(2,n-1,n) =1 then return prime
2. else return composite

算法Ptest1只对a=2进行测试. 如果n为合数且算法输出“素数”，则称n为基2的伪素数. 例如341.

改进算法(一)

改进方法是随机选取2~n-1中的数，进行测试.

取a=3，$3^{340} \pmod{341} \equiv 56$，341不是素数.

算法Ptest2(n)

1. $a \leftarrow$Random(2,n–1)
2. if Expmod(a,n–1,n)=1 then return prime
3. else return composite

算法的问题

Fermat小定理的条件是必要条件，不是充分条件，满足这个条件的也可能是合数.

对上述所有与 n 互素的正整数 a，都满足上述条件的合数 n 称为**Carmichael**数，如**561**，**1105**，**1729**，**2465**等。**Carmichael**数非常少，小于10^8的只有**255**个。

如果n为合数，但不是**Carmichael**数，算法**Ptest2** 测试n为合数的正确的概率至少为**1/2.** 但是这个算法不能解决**Carmichael**数的问题。

素数的另一个必要条件

定理2 如果n为素数，则方程$x^2 \equiv 1 \pmod{n}$的根只有两个，即$x = 1$，$x = -1$（或 $x = n-1$）.

证明 $x^2 \pmod{n} \equiv 1$

$\Leftrightarrow x^2 - 1 \equiv 0 \pmod{n}$

$\Leftrightarrow (x+1)(x-1) \equiv 0 \pmod{n}$

$\Leftrightarrow x+1 \equiv 0$ 或 $x-1 \equiv 0$ （域中没有零因子）

$\Leftrightarrow x = n-1$ 或 $x=1$

称 $x \neq \pm 1$ 的根为非平凡的.

定理2: 如果方程有非平凡的根，则n为合数. 例如:

$x^2 \pmod{5} \equiv 1 \Leftrightarrow x = 1$ 或 $x = 4$

$x^2 \pmod{12} \equiv 1 \Leftrightarrow x = 1$ 或 $x = 11$ 或 $x = 5$ 或 $x=7$

5 和 7 是非平凡的根

基于定理2的测试原理

设n为奇素数，存在q, m使得 $n-1=2^q m$, ($q\geq 1$). 序列

$$a^m \pmod n, a^{2m} \pmod n, a^{4m} \pmod n, \ldots, a^{2^q m} \pmod n$$

的最后一项为 $a^{n-1} \pmod n$, 而且每一项是前面一项的平方.

对于任意 i（$i=0,1,\ldots q-1$）, 判断

$$a^{2^i m} \pmod n$$

是否为 1 和 $n-1$, 且它的后一项是否为1.

如果其后项为1，但本项不等于 1 和 $n-1$, 则它就是非平凡的根，从而知道n不是素数.

随机选择 $a\in\{2,3,\ldots, n-1\}$, 进行上述测试.

实　例

例如 $n=561$, $n-1=560=2^4\cdot 35$, 假设 $a=7$, 构造的序列为

$$7^{35}(\mathrm{mod}\,561)=241,$$

$$7^{2^1 35}(\mathrm{mod}\,561)=7^{70}(\mathrm{mod}\,561)=298,$$

$$7^{2^2 35}(\mathrm{mod}\,561)=7^{140}(\mathrm{mod}\,561)=166,$$

$$7^{2^3 35}(\mathrm{mod}\,561)=7^{280}(\mathrm{mod}\,561)=67,$$

$$7^{2^4 35}(\mathrm{mod}\,561)=7^{560}(\mathrm{mod}\,561)=1$$

第 5 项为 1, 但是第 4 项等于 67, 它既不等于 1 也不等于 560, 是个非平凡的根，因此可以判定 n 为合数.

根据这个思想设计的计算机算法称为 Miller-Rabin 算法，它随机选择正整数 $a\in\{2,3,\ldots,n-1\}$, 然后进行上述测试.

算法子过程(一)

算法 findq-m(*n*)　//找*q*,*m*使得$n-1=2^q m$

1. $q \leftarrow 0$; $m \leftarrow n-1$

2. repeat

3.　$m \leftarrow m/2$

4.　$q \leftarrow q+1$

5. until　*m*是奇数

算法子过程（二）

算法 test(n,q,m) //检测序列是否存在非平凡的根

1. $a \leftarrow$Random(2,$n-1$)
2. $x_0 \leftarrow$ExpMod(a,m,n) //$x_0=a^m \pmod n$, $O(\log^3 n)$
3. for $i \leftarrow 1$ to q do //$q=O(\log n)$
4. $\quad x_i \leftarrow x_{i-1}^2 \pmod n$ //$O(\log^2 n)$
5. $\quad$ if $x_i=1$ and $x_{i-1}\neq 1$ and $x_{i-1}\neq n-1$
6. $\quad\quad$ then return composite
7. if $x_q\neq 1$ then return composite
8. return prime

测试算法

```
算法 Witness(n)      //检测素数算法
  1. findq-m(n)      //O(logn)
  2. test(n,q,m)     //O(log³n)
```

$W(n)=O(\log^3 n)$

可以证明算法 **Witness** 出错的概率至多为**1/2.**

重复运行k次，可以将出错概率降到至多2^{-k}.

Miller-Rabin算法

令$k=\lceil \log n \rceil$, 出错的概率小于等于$2^{-k} \le 1/n$. 即算法给出正确答案的概率为$1-1/n$.

换句话说，如果n为素数，则算法输出素数. 如果n为合数，则算法以$1-1/n$的概率输出“合数”.

算法PrimalityTest(n) //$n \ge 5$，奇整数//

1. findq-m(n)
2. $k \leftarrow \lceil \log n \rceil$
3. for $i \leftarrow 1$ to k //重复执行log n次//
4. test(n,q,m)

时间：$T(n)=O(\log^4 n)$ //按位乘统计

几种算法总结

算法	有解	解正确	改进复杂性的优势	设计要点
Sherwood	有	是	可能改善最坏情况	随机选择
Las Vegas	不定	是	可能改善平均情况	与确定算法相结合
Monte Carlo	有	不定	解决目前困难的问题	概率>1/2 多次执行

顺序算法分析的基本方法

算法分析的原则

- 正确性、工作量、占用空间
- 简单性、最优性（问题复杂度）

算法分析的实例

- 搜索有序表
- 排序
- 选择

算法分析的原则

正确性

概念 在给定有效输入后，算法经过有限时间的计算并产生正确的答案，就称算法是正确的.

正确性证明的内容：

方法的正确性证明——算法思路的正确性.
证明一系列与算法的工作对象有关的引理、定理以及公式.

程序的正确性证明——证明所给出的一系列指令确实做了所要求的工作.

程序正确性证明的方法：

大型程序的正确性证明——可以将它分解为小的相互独立的互不相交的模块，分别验证.

小模块程序可以使用以下方法验证：

数学归纳法、软件形式方法等

工作量--时间复杂性分析

计量工作量的标准：对于给定问题，该算法所执行的基本运算的次数.

基本运算的选择：根据问题选择适当的基本运算

问题	基本运算
在表中查找 x	比较
实矩阵相乘	实数乘法
排序	比较
遍历二叉树	置指针

两种时间复杂性：

最坏情况下的复杂性$W(n)$

平均情况下的复杂性$A(n)$

占用空间--空间复杂性分析

两种占用:

存储程序和输入数据的空间

存储中间结果或操作单元所占用空间--额外空间

影响空间的主要因素：

存储程序的空间一般是常数(和输入规模无关)

输入数据空间为输入规模 $O(n)$

空间复杂性考虑的是额外空间的大小

如果额外空间相对于输入规模是常数，称为原地工作的算法.

两种空间复杂性:

最坏情况下的复杂性和平均情况下的复杂性.

简单性

含义：算法简单，程序结构简单.

好处：容易验证正确性

便于程序调试

简单的算法效率不一定高．要在保证一定效率的前提下力求得到简单的算法.

最优性

含义　指求解某类问题中效率最高的算法

两种最优性

最坏情况下最优：设 A 是解某个问题的算法，如果在解这个问题的算法类中没有其它算法在最坏情况下的时间复杂性比 A 在最坏情况下的时间复杂性低，则称 A 是解这个问题在最坏情况下的最优算法.

平均情况下最优：设 A 是解某个问题的算法，如果在解这个问题的算法类中没有其它算法在平均情况下的时间复杂性比 A 在平均情况下的时间复杂性低，则称 A 是解这个问题在平均情况下的最优算法.

寻找最优算法的途径

设计算法A, 求$W(n)$. 相当于给出最坏情况下的一个上界.

寻找函数$F(n)$, 使得对任何算法都存在一个规模为n的输入并且该算法在这个输入下至少要做$F(n)$次基本运算. 相当于对问题给出最坏情况下所需基本运算次数的一个下界.

如果$W(n)=F(n)$, 则A是最优的.

如果$W(n)>F(n)$, A不是最优的或者$F(n)$的下界过低.

改进A或设计新算法A'使得$W'(n)<W(n)$.

重新证明新下界$F'(n)$使得$F'(n)>F(n)$.

重复以上两步,最终得到$W'(n) = F'(n)$为止.

例1 在n个不同的数中找最大的数

基本运算：比较

算法 findmax

输入 数组L, 项数 $n \geq 1$

输出 L中的最大项MAX

1. $MAX \leftarrow L(1)$; $i \leftarrow 2$;
2. **while** $i \leq n$ **do**
3. **if** $MAX < L(i)$ **then** $MAX \leftarrow L(i)$;
4. $i \leftarrow i+1$;

行3的比较进行$n-1$次, 故 $W(n) = n-1$.

问题类复杂度

定理

在 n 个数的数组中找最大的数, 并以比较作为基本运算的算法类中的任何算法最坏情况下至少要做 $n-1$ 次比较.

证 因为MAX是唯一的, 其它的$n-1$个数必须在比较后被淘汰. 一次比较至多淘汰一个数, 所以至少需要 $n-1$ 次比较.

结论: findmax 算法是最优算法.

下界证明法一：判定树

- 判定树：作为某个算法类的模型，根据算法类的性质给出判定树的构造规则
  - 一棵判定树对应一个算法
  - 一片树叶代表一个输入
  - 从根到某片树叶的路径代表算法对这个输入的工作量
  - 树深代表最坏情况的工作量，平均路径长度代表平均工作量
- 算法类的最坏情况下时间复杂度下界：所有的树中深度最浅的树的树深.
- 算法类的平均情况下时间复杂度下界：所有的树中平均深度最浅的树的树深.

下界证明法二：构造最坏输入

任意给定一个算法 A，A对于任意输入 x 都存在一个确定的操作序列 τ

- τ中的操作分成两类：
  - 决定性的：能够对确定输出结果提供有效信息
  - 非决定性的：对确定结果没有帮助的冗余操作
- 根据算法A构造某个输入实例 x，使得A对x 的操作序列 τ 包含尽量多的非决定性操作.
- 给出冗余操作+必要的操作的计数公式

下界证明法三：归约

条件

- 已知问题 Q最坏情况下时间复杂度下界为$F(n)$，即求解Q的任何算法 A 最坏情况的时间$\geq F(n)$
- 需要确定问题 P 的算法类最坏情况下的时间复杂度下界

方法

- 设计一个求解 Q 的算法 A，A调用任意解P的算法B.
  时间为$T_A(n)= t_1(n)+p(n)+t_2(n)$：$t_1$为把$Q$的实例$I$ 转换到P的实例 $f(I)$ 的时间，t_2为把 P 的解s 转换成Q的解的时间
- 若$t_1(n), t_2(n)=O(p(n))$，$|f(I)|=O(n)$，则$T_A(n)=\Theta(p(n))$

  $p(n)=\Theta(T_A(n))=\Omega(F(n))$

结论

P至少与Q一样难

算法分析的实例

搜索有序表

顺序搜索、改进顺序搜索、二分搜索

最优性的分析

排序

起泡排序、快速排序与归并排序、堆排序

最优性的分析

选择

选择最大、选最大和最小、选第二大、选中位数

最优性分析

搜索有序表

问题: 按递增顺序排列的数组 L, 项数 $n \geq 1$. 数 x
如果 x 在 L 中, 输出 x 的下标; 否则输出0

算法1　顺序搜索

输入: L, x

输出: j

1. $j \leftarrow 1$
2. while $j \leq n$ and $L(j) \neq x$ do $j \leftarrow j+1$
3. if $j > n$ then $j \leftarrow 0$

分析：设 x 在L中每个位置和空隙的概率都是$1/(2n+1)$

$W(n)=n$

$A(n)=[(1+2+...+n)+n(n+1)]/(2n+1) \approx 3n/4.$

改进的顺序搜索

算法2　改进的顺序搜索

1. $j \leftarrow 1$
2. while $j \le n$ and $x > L(j)$ do $j \leftarrow j+1$
3. if $x < L(j)$ or $j > n$ then $j \leftarrow 0$

复杂度 $W(n) = n$

$$A(n) = [2(1+2+\ldots+n)+n]/(2n+1) \approx n/2$$

二分法搜索

算法3 二分搜索

1. $k \leftarrow 1$; $m \leftarrow n$
2. **while** $k \le m$ **do**
3. $\quad j \leftarrow \lfloor (k+m)/2 \rfloor$
4. **if** $x = L(j)$ **then return**
5. **if** $x < L(j)$ **then** $m \leftarrow j-1$
6. $\quad$ **else** $k \leftarrow j+1$
7. $j \leftarrow 0$

复杂度 $W(n) = 1 + W(\lfloor n/2 \rfloor)$ $\quad n>1$

$W(1) = 1$

二分法最坏复杂度

定理1 $W(n)=\lfloor \log n \rfloor + 1 \qquad n \geq 1$

证 对n归纳

n=1时, 左= $W(1)$=1, 右 $=\lfloor \log 1 \rfloor$+1 = 1.

假设对一切k, $1 \leq k < n$, 命题为真, 则

$$W(n) = 1 + W\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

$$= 1 + \left\lfloor \log \left\lfloor \frac{n}{2} \right\rfloor \right\rfloor + 1$$

$$= \begin{cases} \lfloor \log n \rfloor + 1 & n\text{为偶数} \\ \lfloor \log(n-1) \rfloor + 1 & n\text{为奇数} \end{cases}$$

$$= \lfloor \log n \rfloor + 1$$

二分法平均复杂度

令 $n=2^k-1$, S_t 是算法做 t 次比较的输入个数,

$1 \le t \le k$

则

$S_1=1=2^0$, $S_2=2=2^1$, $S_3=2^2$, $S_4=2^3$, … ,

$S_t= 2^{t-1}$, $t<k$

$S_k= 2^{k-1} + n + 1$

其中 2^{k-1} 为 x 在表中做 k 次比较的输入个数

$$A(n) = \frac{1}{2n+1}(1S_1 + 2S_2 + \ldots + kS_k)$$

二分法平均复杂度（续）

$$A(n)=\frac{1}{2n+1}(1S_1+2S_2+\ldots+kS_k)$$

$$=\frac{1}{2n+1}[\sum_{t=1}^{k}t2^{t-1}+k(n+1)]$$

$$=\frac{1}{2n+1}[(k-1)2^k+1+k(n+1)]$$

$$=\frac{k-1}{2}+\frac{k}{2}=k-\frac{1}{2}=\lfloor\log n\rfloor+\frac{1}{2}$$

下界估计：判 定 树

设A是一个搜索算法, 对于给定输入规模n, A的一棵判定树是一棵二叉树, 其结点被标记为1, 2, ... , n, 且标记规则是:

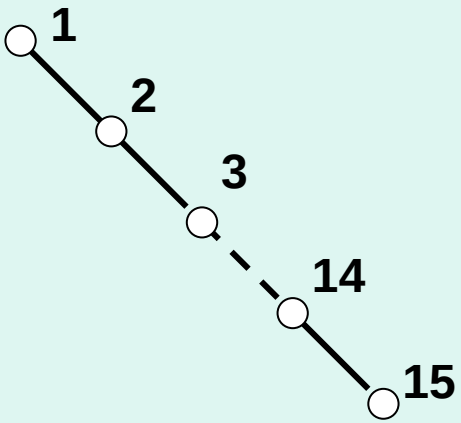
1. 根据算法A，首先与 x 比较的 L 的项的下标标记为树根.
2. 假设某结点被标记为i,

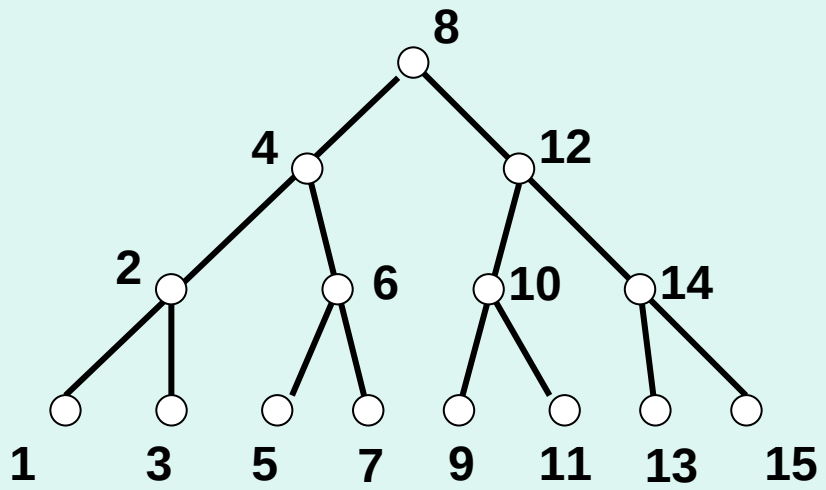
   i 的左儿子是当 $x<L(i)$时，算法A下一步与x比较的项的下标; i 的右儿子是当 $x>L(i)$时，算法A下一步与x比较的项的下标.

   若 $x<L(i)$ 时算法 A 停止, 则 i 没有左儿子. 若 $x>L(i)$ 时算法 A 停止, 则 i 没有右儿子.

实　例

改进顺序搜索算法和二分搜索算法的判定树，$n=15$





给定输入, 算法*A*将从根开始,沿一条路径前进,直到某个结点为止. 所执行的基本运算次数是这条路径的结点个数. 最坏情况下的基本运算次数是树的深度+1.

引 理

引理1　在二叉树的 t 层至多有 2^t 个结点 (根为0层)

证 归纳法.

$t = 0$, 树有1个结点(根). $2^0=1$. 命题为真.

假设 t 层有 2^t 个结点, 则 $t+1$ 层至多有 $2\cdot 2^t$ 个结点, 即 2^{t+1} 个结点.

引　理（续）

引理2　深度为 d 的二叉树至多有$2^{d+1}-1$个结点.

证　归纳法.

$d=0$, 树有1个结点(根). 而$2^{0+1}-1=1$.

假设深度为 d 的二叉树至多有$2^{d+1}-1$个结点, 考虑一棵深度为$d+1$的二叉树. 由引理1, 在第$d+1$层至多有2^{d+1}个结点. 故深度为$d+1$的二叉树至多有

$$2^{d+1}-1+2^{d+1}=2^{d+2}-1$$

个结点

引　理（续）

引理3　n个结点的二叉树深度至少为$\lfloor \log n \rfloor$.

证　假若n个结点的二叉树深度至多为

$$\lfloor \log n \rfloor - 1,$$

则由引理 2 该树的结点数至多为

$$2^{\lfloor \log n \rfloor - 1 + 1} - 1$$

$$= 2^{\lfloor \log n \rfloor} - 1$$

$$\leq 2^{\log n} - 1 = n - 1$$

问题的复杂度分析

定理2 对于任何一个搜索算法存在某个规模为n的输入使得该算法至少要做$\lfloor \log n \rfloor +1$次比较.

证 由引理3, n个结点的判定树的深度d至少为$\lfloor \log n \rfloor$, 故

$$W(n)=d+1=\lfloor \log n \rfloor +1.$$

结论: 对于有序表搜索问题, 在以比较作为基本运算的算法类中, 二分法在最坏情况下是最优的.

排 序

起泡排序

输入: $L, n \geq 1$.

输出: 按非递减顺序排序的L.

算法 bubbleSort

```
1. FLAG ← n   //标记被交换的最后元素位置
2. while FLAG > 1 do
3.    k ← FLAG –1
4.    FLAG ←1
5.    for j=1 to k do
6.      if L(j) > L(j+1) then do
7.         L(j) ↔ L(j+1)
8.         FLAG ← j
```

实　例

5	7	2	6	9	3	4	8	1
5	2	6	7	3	4	8	1	9
2	5	6	3	4	7	1	8	9
2	5	3	4	6	1	7	8	9
2	3	4	5	1	6	7	8	9
2	3	4	1	5	6	7	8	9
2	3	1	4	5	6	7	8	9
2	1	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

特点：交换发生在相邻元素之间

置换与逆序

逆序　令$L=\{1,2,...,n\}$, 排序的任何输入为L上的置换. 在置换$a_1\ a_2...a_n$中若$i<j$ 但$a_i>a_j$, 则称 (a_i,a_j) 为该置换的一个逆序.

逆序序列　在 i 右边，并且小于i 的元素个数记作b_i, $i=1, 2,\ldots,n$. $(b_1, b_2, \ldots, b_n)$ 称为置换的逆序序列.

例如

置换　　3 1 6 5 8 7 2 4

逆序序列为　(0, 0, 2, 0, 2, 3, 2, 3)

置换与逆序（续）

逆序序列满足以下性质：

$b_1=0;\quad b_2=0,1;\quad \ldots\ ;\ b_n=0,1,\ldots,n-1$

总共$n!$个不同的逆序序列

置换与它的逆序序列构成一一对应

逆序数 置换中的逆序总数

$$b_1 + b_2 + \ldots + b_n$$

实例：置换 3 1 6 5 8 7 2 4

逆序序列为 (0, 0, 2, 0, 2, 3, 2, 3)

逆序数 12

复杂度分析

最坏情况分析

起泡排序算法的对换只发生在相邻的元素之间

每次相邻元素的交换只消除1个逆序

交换个数不少于逆序数

比较次数不少于交换次数

最大的逆序数是 $n(n-1)/2$

最坏情况下起泡排序算法所做的交换次数至少是 $n(n-1)/2$, 比较次数至少是$n(n-1)/2$.

复杂度分析（续）

平均情况

设各种输入是等概的

置换α 的逆序序列是 $(b_1, b_2, \dots, b_n)$,

置换α′ 的逆序序列为 $(0-b_1, 1-b_2, \dots, n-1-b_n)$

α 与α′ 的逆序数之和为 $n(n-1)/2$

将$n!$个置换分成 $n!/2$个组，

每组逆序之和为 $n(n-1)/2$.

平均逆序数 $n(n-1)/4$ ，

平均的交换次数为 $n(n-1)/4$

结论　起泡排序的最坏和平均复杂性均为$O(n^2)$

快速排序与二分归并排序

快速排序

最坏情况 $O(n^2)$

平均情况 $O(n\log n)$

二分归并排序

最坏情况 $O(n\log n)$

平均情况 $O(n\log n)$

堆排序

堆的定义

堆的运算

- 堆整理 **HEAPIFY(*A*,*i*)**
  - 复杂度分析
- 建堆 **BUILD-HEAP(*A*)**
  - 复杂度分析

堆排序算法 **HEAPSORT(*A*)**

- 复杂度分析

堆的定义

设T是一棵深度为 d 的二叉树，结点为L中的元素.

若满足：

所有内结点（可能一点除外）的度数为 2

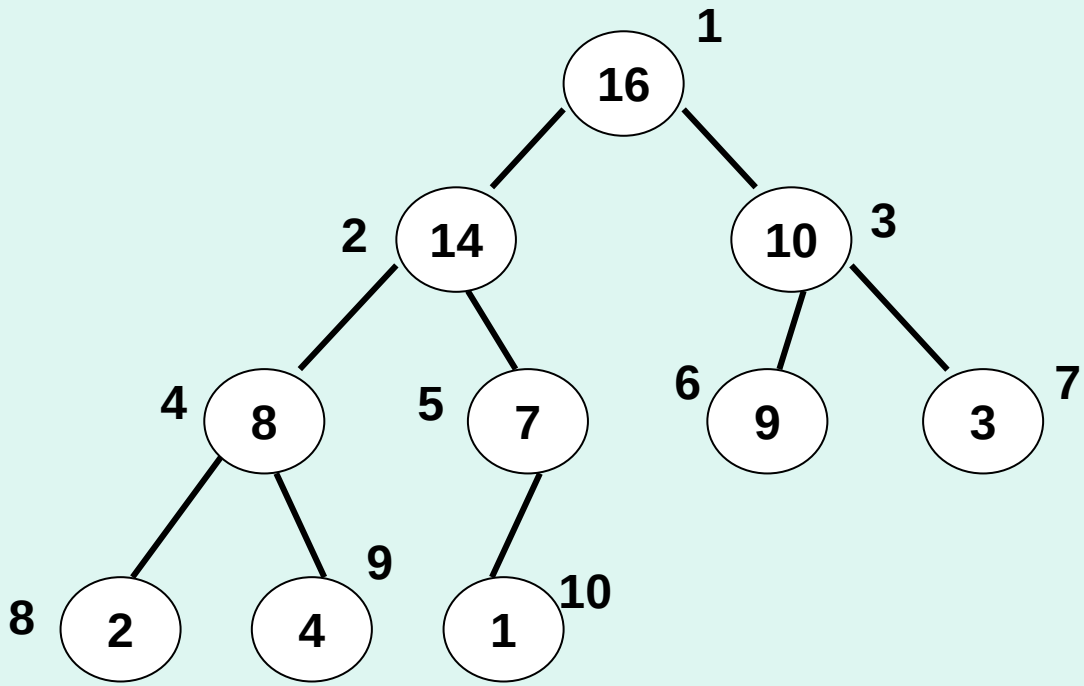
所有树叶至多在相邻的两层

d–1 层的所有树叶在内结点的右边

d–1 层最右边的内结点可能度数为1（没有右儿子）

每个结点的元素不小于儿子的元素

堆的实例



堆存储在数组*A*

A[*i*]：结点 *i* 的元素，例如*A*[2]=14.

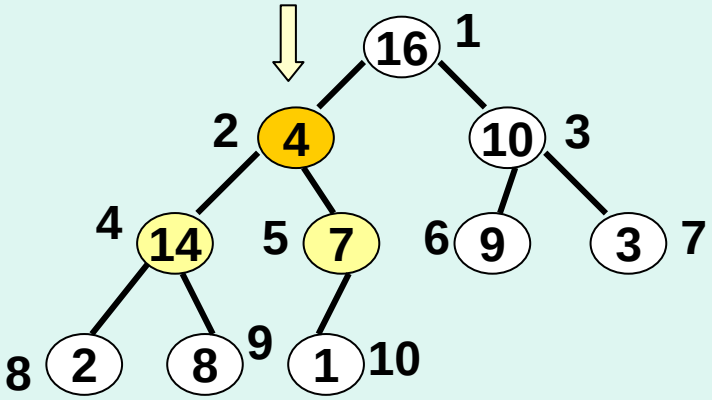
LEFT(*i*), *RIGHT*(*i*) 分别表示 *i* 的左儿子和右儿子

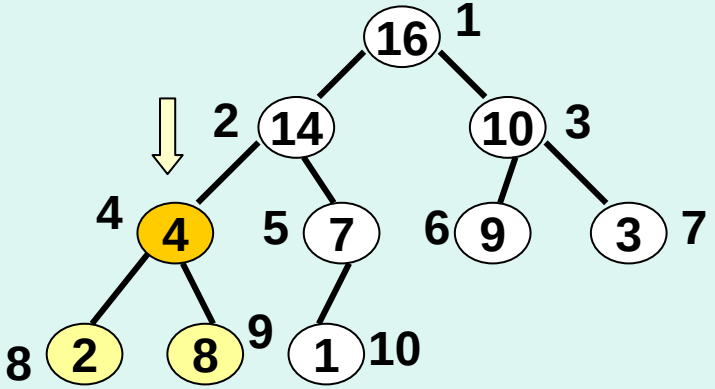
堆的运算——整理

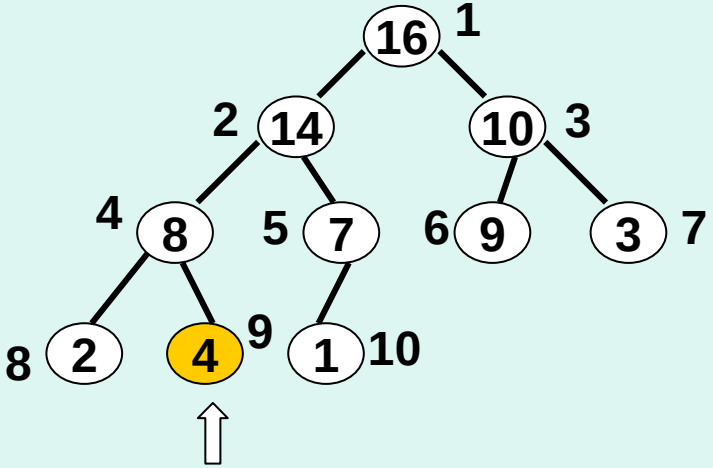
算法 HEAPIFY(A,i)

1. $l \leftarrow LEFT(i)$
2. $r \leftarrow RIGHT(i)$
3. if $l \le heap\text{-}size[A]$ and $A[l] > A[i]$
4. then $largest \leftarrow l$
5. else $largest \leftarrow i$
6. if $r \le heap\text{-}size[A]$ and $A[r] > A[largest]$
7. then $largest \leftarrow r$
8. if $largest \neq i$
9. then exchange $A[i] \leftrightarrow A[largest]$
10. **HEAPIFY(A, $largest$)**

HEAPIFY 实例







HEAPIFY(*A*,2)

复杂度分析

每次调用为$O(1)$

子堆大小至多为原来的 2/3

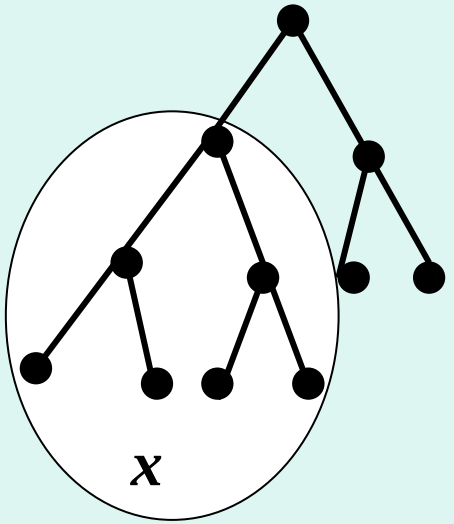
递推不等式

$$T(n) \le T(2n/3) + \Theta(1)$$

解得 $T(n) = \Theta(\log n)$

或者 $T(h) = \Theta(h)$

h为堆的根的高度（距树叶距离）



结点总数

$x+(x-1)/2+1=(3x+1)/2$

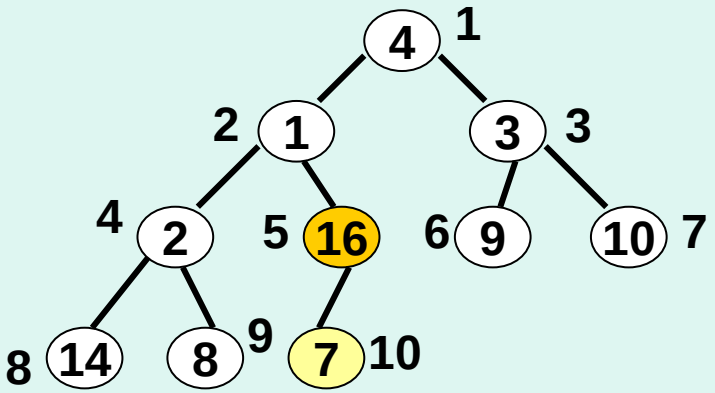
堆的运算——建堆

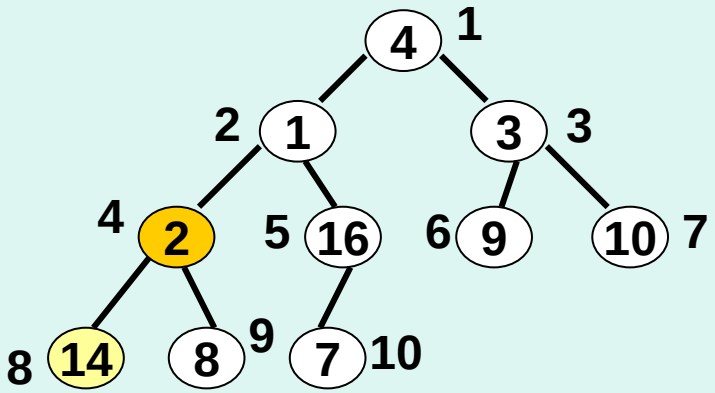
算法 BUILD-HEAP(*A*)

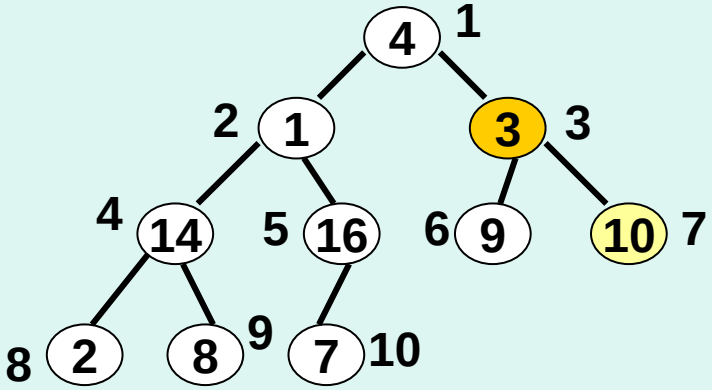
1．*heap-size*[*A*] ← *length*[*A*]

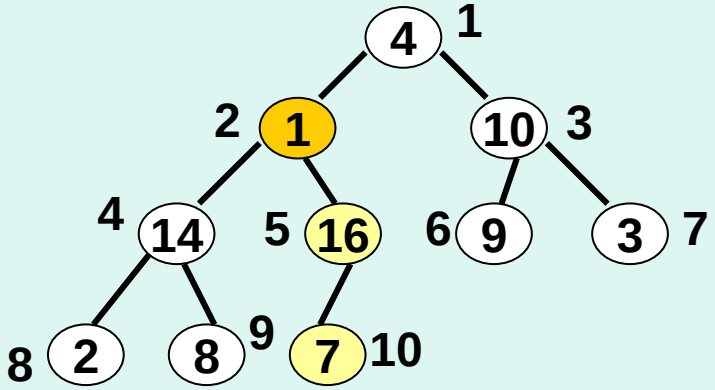
2．for *i* ←⌊*length*[*A*]/2⌋ downto 1

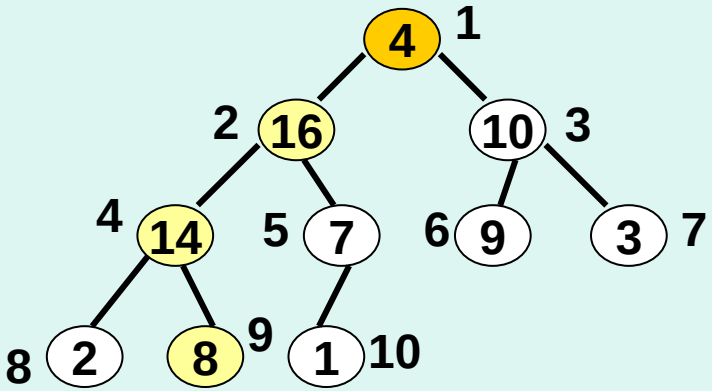
3． do HEAPIFY(*A*, *i*)

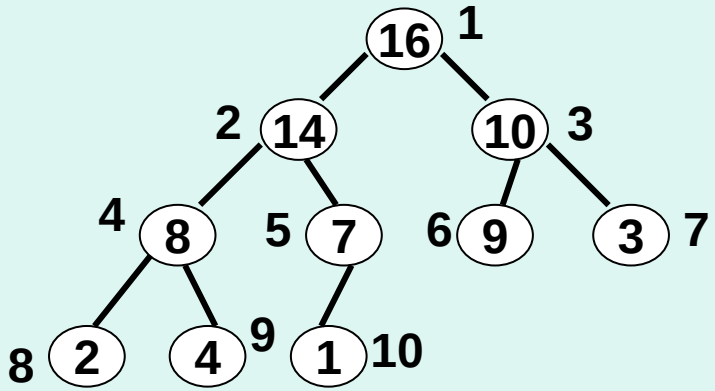

1 4
2 1
3 3
4 2
5 16
6 9
10 7
8 14
8 9
7 10


1 4
2 1
3 3
4 2
5 16
6 9
10 7
8 14
8 9
7 10


1 4
2 1
3 3
4 14
5 16
6 9
10 7
8 2
8 9
7 10


1 4
2 1
10 3
4 14
5 16
6 9
3 7
8 2
8 9
7 10


1 4
2 16
10 3
4 14
5 7
6 9
3 7
8 2
8 9
1 10


1 16
2 14
10 3
4 8
5 7
6 9
3 7
8 2
4 9
1 10

复杂度分析

结点的高度　该结点距树叶的距离

结点的深度　该结点距树根的距离

同一深度结点分布在树的同一层

同一高度结点可以分布在树的不同的层

思路：

HEAPIFY(*i*) 的复杂度依赖于 ***i*** 的高度***h***

按照高度计数结点数，乘以***O*(*h*)**, 再求和

$$T(n)=\sum_{h=0}^{\lfloor \log n \rfloor} \text{高为}h\text{的结点数} \times O(h)$$

引　理

***n*个元素的堆高度 *h* 的层至多存在 $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ 个结点.**

证明: 对*h*进行归纳.

归纳基础

h=0, 命题为真，即证堆中树叶数为 $\left\lceil \frac{n}{2} \right\rceil$

归纳步骤

假设对*h*-1为真，证明对*h*也为真.

归纳基础

h=0, 树叶分布在 d 和 d-1 层，d 层(x个), d-1 层(y个).

Case1: x为偶数

每个内结点有2个儿子，树叶数为

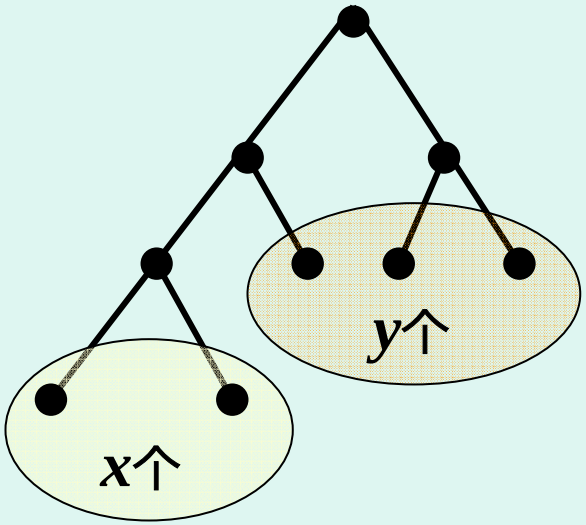
$$x + y = x + 2^{d-1} - \frac{x}{2} = 2^{d-1} + \frac{x}{2}$$

$$= \frac{(2^d + x)}{2}$$

$$= \left\lceil \frac{2^d + x - 1}{2} \right\rceil$$

$$= \left\lceil \frac{n}{2} \right\rceil$$

(x为偶数，d-1层以前各层结点总数2^d-1)



归纳基础 (续)

Case2 x**为奇数，**

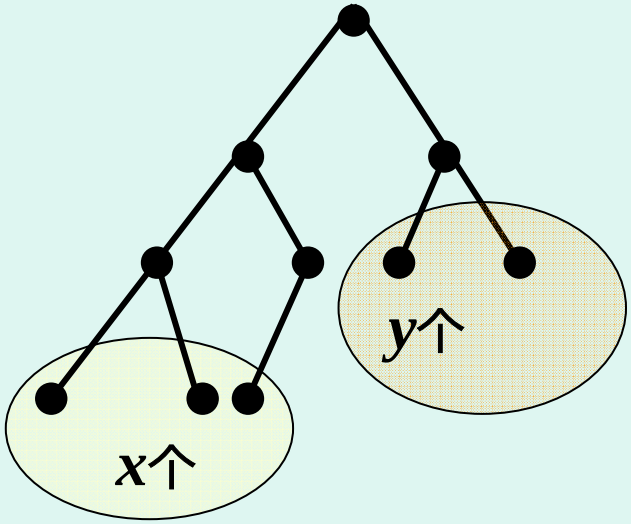
$$x + y = x + 2^{d-1} - \frac{x+1}{2}$$

$$= 2^{d-1} + \frac{x-1}{2}$$

$$= \frac{2^d + x - 1}{2}$$

$$= \frac{n}{2} = \left\lceil \frac{n}{2} \right\rceil$$

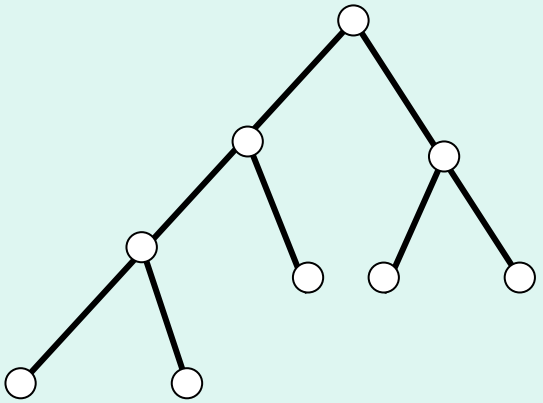
(x为奇数，n为偶数)



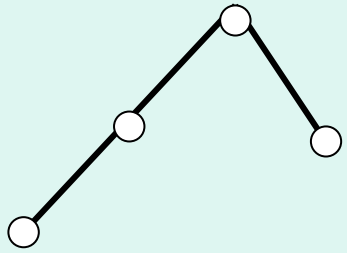
归纳步骤

假设命题对于高度 h-1为真，证明对于高度为 h 也为真.

设T表示n个结点的树，从T中移走所有的树叶得到树T', T'的结点数为n'. T高为h的层恰好是T'的高为h-1的层.



T



T'

归纳步骤（续）

令 n_h 表示 T 中高为 h 的层的结点数

根据归纳基础，$n_0 = \lceil n/2 \rceil$

$$n' = n - n_0 = \left\lfloor \frac{n}{2} \right\rfloor$$

$$n_h = n'_{h-1}$$

$$n_h = n'_{h-1} \le \left\lceil \frac{n'}{2^h} \right\rceil = \left\lceil \frac{\left\lfloor \frac{n}{2} \right\rfloor}{2^h} \right\rceil \le \left\lceil \frac{\frac{n}{2}}{2^h} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

时间复杂度分析

定理3 n 个结点的堆算法 **BUILD-HEAP** 的时间复杂性是 $O(n)$

证明：对高为 h 的结点调用**HEAPIFY**算法时间是 $O(h)$, 高为 h 的结点数至多为 $\left\lceil \frac{n}{2^{h+1}} \right\rceil$, 因此时间复杂性

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{h}{2^{h+1}} \right\rceil\right)$$

$$= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^{h+1}}\right) = O(n)$$

求 和

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = [0 + \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \ldots]$$

$$= [\frac{1}{2} + \frac{1}{2^2} + \ldots] + [\frac{1}{2^2} + \frac{1}{2^3} + \ldots] + [\frac{1}{2^3} + \frac{1}{2^4} + \ldots] + \ldots$$

$$= [1 + \frac{1}{2} + \frac{1}{2^2} + \ldots][\frac{1}{2} + \frac{1}{2^2} + \ldots] = \frac{1}{2}\frac{1}{(1 - \frac{1}{2})^2} = 2$$

堆 排 序

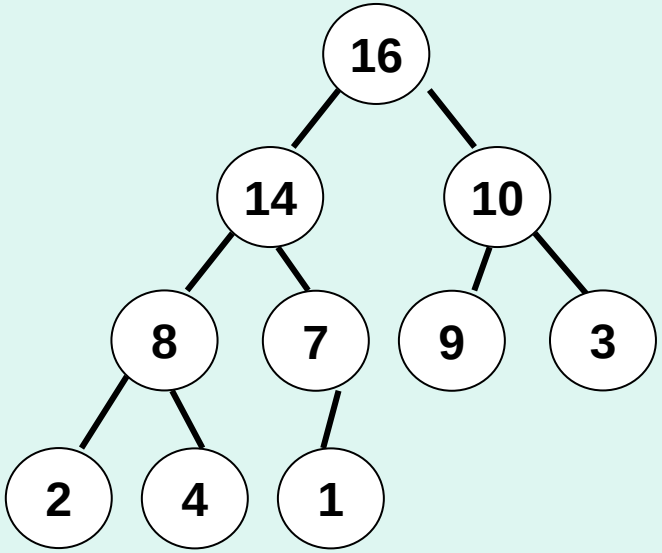
算法 HEAPSORT(*A*)

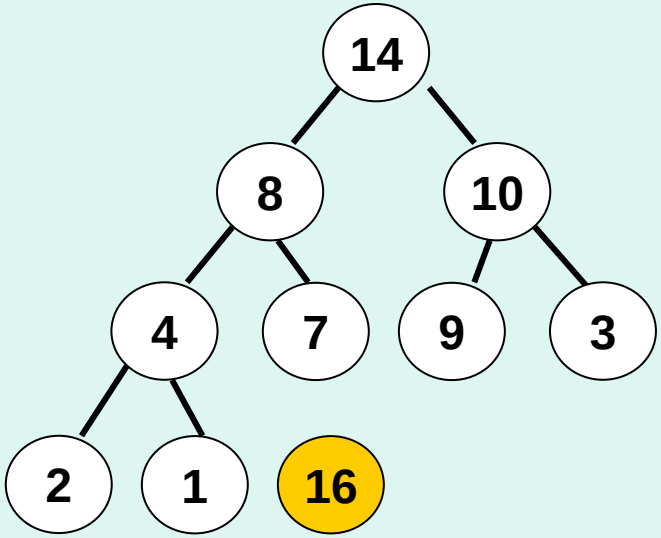
1. BUILD-HEAP(*A*)
2. for *i* ← length[*A*] downto 2
3. do *exchange* *A*[1]↔*A*[*i*]
4. *heap-size*[*A*] ←*heap-size*[*A*]–1
5. HEAPIFY(*A*,1)

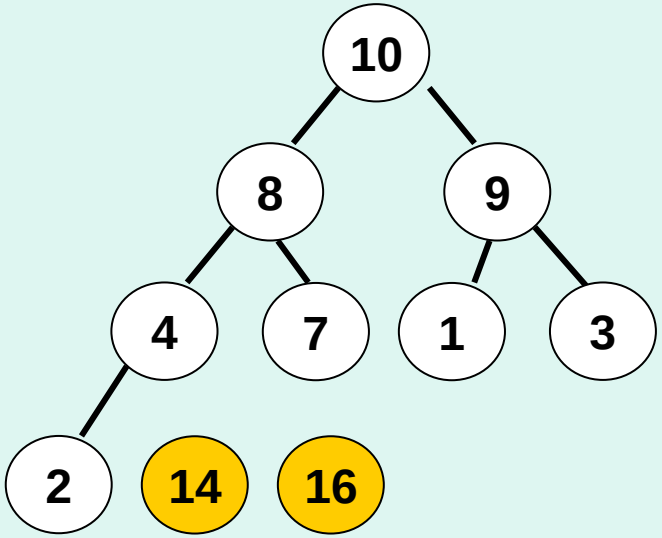
复杂性：*O*(*n*log*n*)

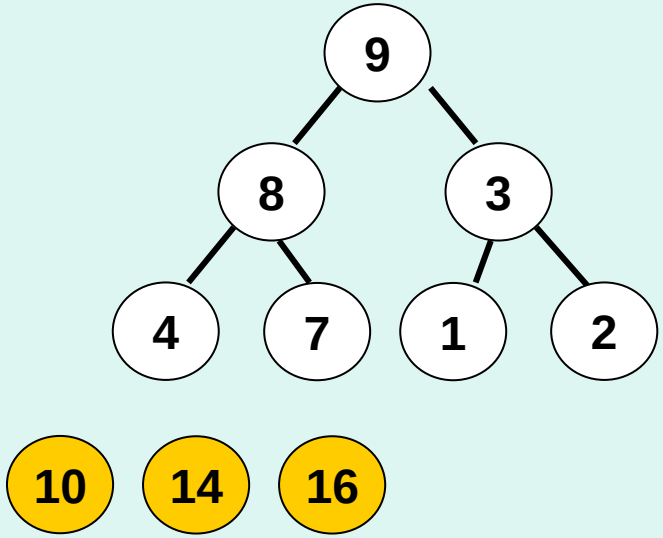
BUILD_HEAP 时间为 *O*(*n*),
从行2-5 调用 HEAPIFY *n*-1次，每次*O*(log*n*)，
时间为O(*n*log*n*)

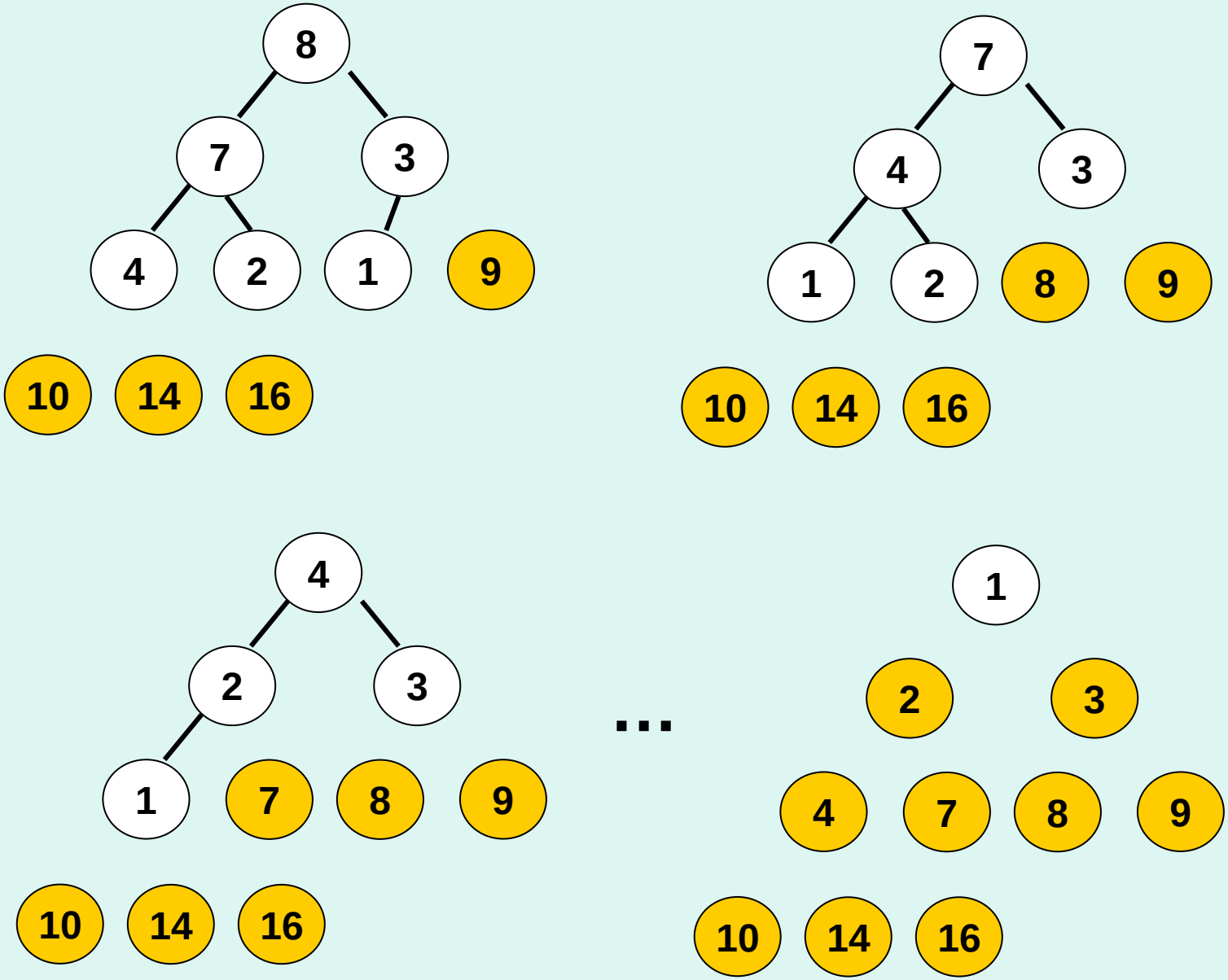
实 例


16
14
10
8
7
9
3
2
4
1


14
8
10
4
7
9
3
2
1
16


10
8
9
4
7
1
3
2
14
16


9
8
3
4
7
1
2
10
14
16


8
7
3
4
2
1
9
10
14
16
7
4
3
1
2
8
9
10
14
16
4
2
3
1
7
8
9
10
14
16
...
1
2
3
4
7
8
9
10
14
16

最优性

考虑以比较运算作为基本算法的排序算法类，
任取算法 A, 输入$L=\{x_1, x_2, \ldots, x_n\}$, 如下定义判定树

1．A第一次比较的元素为x_i, x_j，那么树根标记为 i, j

2．假设结点 k 已经标记为 i, j,

（1）$x_i<x_j$

若算法结束， k的左儿子标记为输入；

若下一步比较元素x_p, x_q，那么 k的左儿子标记为p,q

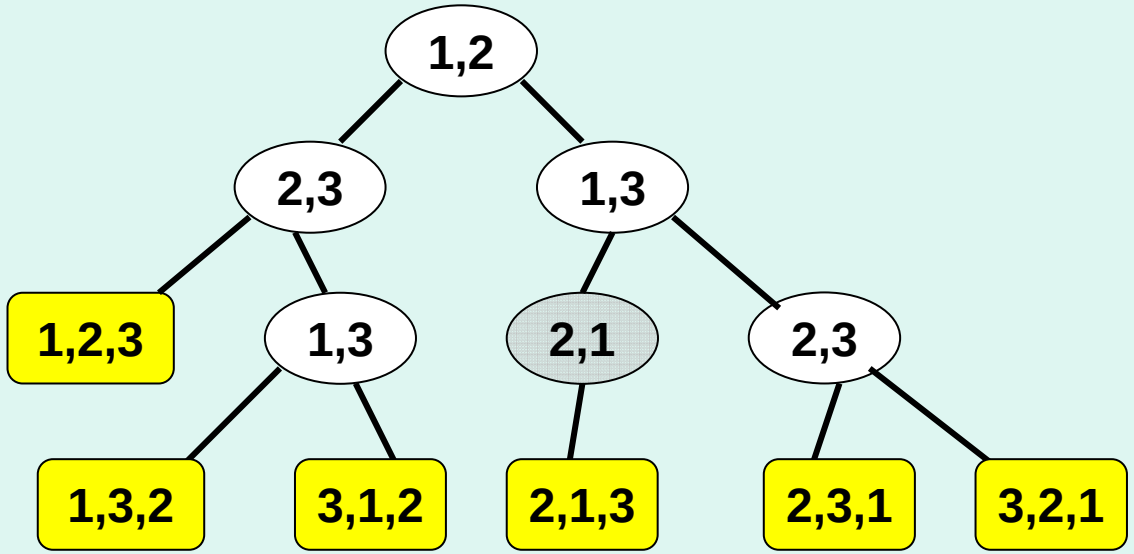
（2）$x_i>x_j$

若算法结束， k的右儿子标记为输入；

若下一步比较元素x_p, x_q，那么 k的右儿子标记为p,q

判定树的实例

设输入为x_1, x_2, x_3，起泡排序的判定树如下



任意输入：对应了判定树中从树根到树叶的一条路经，

最坏情况下的比较次数：树深

去掉非二叉的内结点所连接的树叶（灰色），得到二叉树叫做B-树。B树的深度与判定树至多相差1

引 理

引理1 设 t 为B-树中的树叶数，d 为树深，则 $t \le 2^d$.

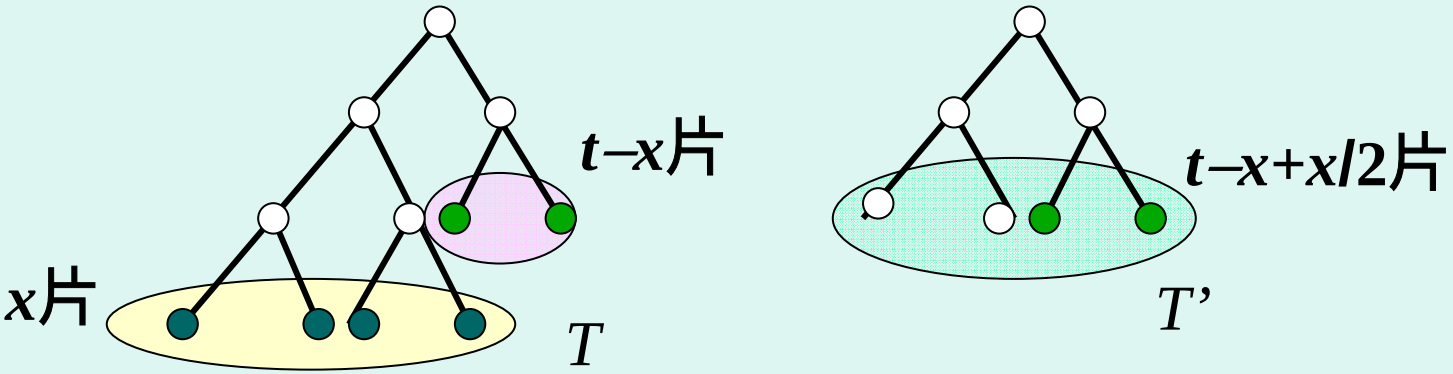
证明 归纳法.

d=0, 树只有1片树叶，深度为0，命题为真.

假设对一切小于d 的深度为真，设 T 是一棵深度为 d 的树，树叶数为 t. 取走 T 的 d 层的 x片树叶，得到树 T'. 则T'的深度为$d-1$，树叶数 t'。那么

$$t' = (t-x) + x/2 = t - x/2, \quad x \le 2^d$$

$$t = t' + x/2 \le 2^{d-1} + 2^{d-1} = 2^d$$



最坏情况复杂度的下界

引理2 对于给定的n，任何通过比较进行排序的算法的判定树的深度至少为$\lceil \log n! \rceil$.

证明：判定树的树叶有$n!$个，由引理1得证。

定理4 任何通过比较对 n 个元素排序的算法在最坏情况下的时间复杂性是$\lceil \log n! \rceil$, 近似为 $n \log n - 1.5n$.

证明：最坏情况的比较次数为树深，由引理2树深至少为

$$\log n! = \sum_{j=1}^{n} \log j \geq \int_1^n \log x dx = \log e \int_1^n \ln x dx$$

$$= \log e(n \ln n - n + 1)$$

$$= n \log n - n \log e + \log e$$

$$\approx n \log n - 1.5n$$

结论：堆排序算法在最坏情况阶达到最优。

平均情况分析

假设所有的输入等概分布，令 epl(T) 表示 B 树中从根到树叶的所有路径长度之和, epl(T)/n! 的最小值对应平均情况复杂性的下界.

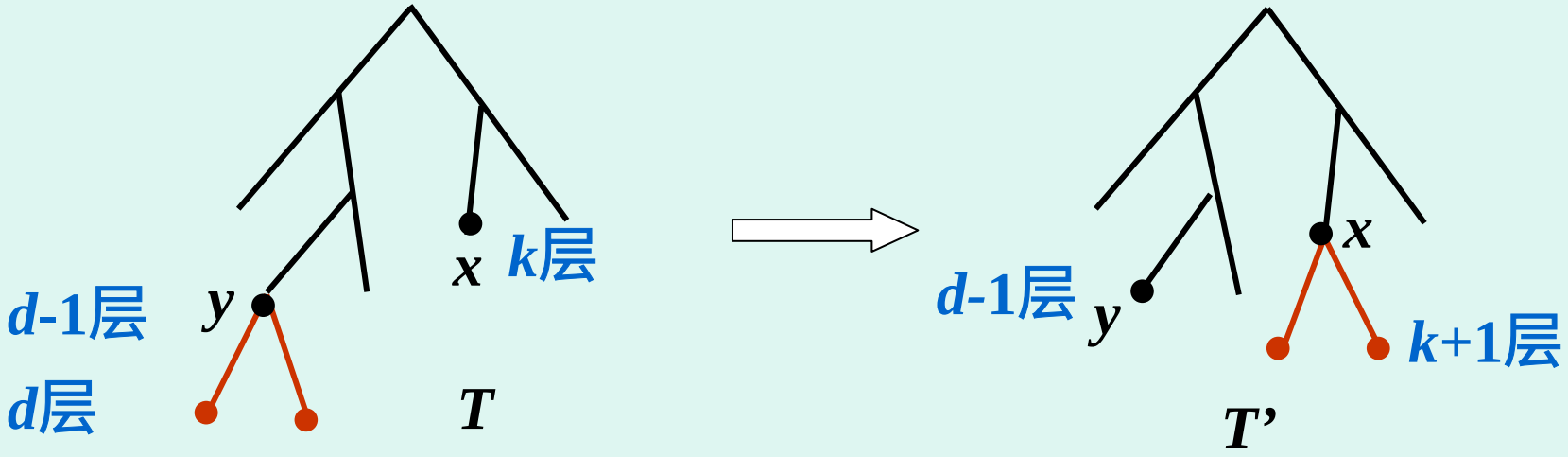
分析具有最小 epl(T) 值的树的结构求得这个最小值.

引理3 在具有 t 片树叶的所有 B-树中,树叶分布在两个相邻层上的树的 epl 值最小

证明：反证法.

设树 T 的深度为 d，假设树叶 x 在第 k 层，$k<d-1$. 取 d-1层的某个结点 y，y 有两个儿子是第 d 层的树叶.将y的两个儿子作为 x 的儿子得到树 T'.

具有最小epl 值的树的结构



$$
\begin{aligned}
&\mathrm{epl}(T) - \mathrm{epl}(T') \\
&= (2d+k) - [(d-1)+2(k+1)] \\
&= 2d+k-d+1-2k-2 \\
&= d-k-1 > 0 \qquad\qquad (d > k+1)
\end{aligned}
$$

T ’的树叶相距层数小于 T 的树叶相距的层数，

而 T ’ 的 epl 值小于 T 的 epl 值

epl值的下界

引理4 具有t片树叶且 epl 最小的 B 树 T 满足

$$\mathrm{epl}(T) = t\lfloor \log t \rfloor + 2(t - 2^{\lfloor \log t \rfloor})$$

证明：由引理1 树 T 的深度 $d \geq \lceil \log t \rceil$,

由引理3，树 T 只有 d 和 d-1层有树叶.

Case1 $t = 2^k$.

必有 $d = k$,

$$\mathrm{epl}(T) = t\,d = t\,k = t\,\lfloor \log t \rfloor$$

epl值的下界（续）

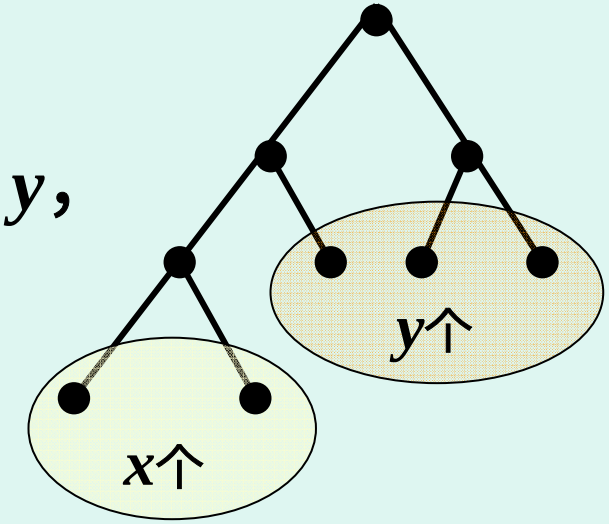
Case2 $t \neq 2^k$.

设 d 层和 $d-1$ 层树叶数分别为 x, y，

$$x + y = t$$

$$x/2 + y = 2^{d-1}$$

解得 $x = 2t - 2^d,\ y = 2^d - t.$



$$\begin{aligned}
\text{epl}(T) &= x\,d + y\,(d-1) \\
&= (2t - 2^d)d + (2^d - t)(d-1) \\
&= td - 2^d + t = t(d-1) + 2(t - 2^{d-1}) \\
&= t\lfloor \log t \rfloor + 2(t - 2^{\lfloor \log t \rfloor}) \quad (\lfloor \log t \rfloor = d-1)
\end{aligned}$$

平均复杂度的下界

定理4 在输入等概分布下任何通过比较对n个项排序的算法平均比较次数至少为$\lfloor \log n! \rfloor$, 近似为 $n\log n - 1.5\, n$.

证明：算法类中任何算法的平均比较次数是该算法判定树T 的 epl(T)/n!, 根据引理4

$$A(n) \geq \frac{1}{n!} epl(T)$$

$$= \frac{1}{n!}[n!\lfloor \log n! \rfloor + 2(n! - 2^{\lfloor \log n! \rfloor})]$$

$$= \lfloor \log n! \rfloor + \varepsilon, \qquad 0 \leq \varepsilon < 1$$

$$\approx n \log n - 1.5n$$

$$0 \leq n! - 2^{\lfloor \log n! \rfloor} < n! - 2^{\log n! - 1} = n! - \frac{n!}{2} = \frac{n!}{2}$$

结论：堆排序在平均情况下阶达到最优.

几种排序算法的比较

算法	最坏情况	平均情况	占用空间	最优性
起泡排序	$O(n^2)$	$O(n^2)$	原地	
快速排序	$O(n^2)$	$O(n\log n)$	$O(\log n)$	平均最优
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n)$	最优
堆排序	$O(n\log n)$	$O(n\log n)$	原地	最优

选择算法的结果

问题	算法	最坏情况	空间
选最大	顺序比较	$n-1$	$O(1)$
选最大和最小	顺序比较	$2n-3$	$O(1)$
	算法FindMaxMin	$\lceil 3n/2 \rceil - 2$	$O(1)$
选第二大	顺序比较	$2n-3$	$O(1)$
	锦标赛方法	$n+\lceil \log n \rceil - 2$	$O(n)$
选中位数	排序后选择	$O(n\log n)$	$O(\log n)$
	算法Select	$O(n)\sim 2.95n$	$O(\log n)$

有关最优性的结果

1. 选最大算法finmax是最优的算法

2. 找最大和最小

定理6　任何通过比较找最大和最小的算法至少需要$\lceil 3n/2 \rceil - 2$次比较.

证明思路：任给算法A，根据算法 A 的比较顺序构造输入T, 使得 A 对 T 至少做$\lceil 3n/2 \rceil - 2$ 次比较.

证：不妨设 n 个数彼此不等，A为任意找最大和最小的算法. max是最大，A必须确定有$n-1$个数比max小，通过比较被淘汰. min是最小，A也必须确定有$n-1$个数比min大，通过比较而淘汰. 总共需要$2n-2$个信息单位.

按下述策略构造输入T，使A对T 至少做$\lceil 3n/2 \rceil - 2$次比较.

选最大、最小问题下界

数的状态及其含义

N：没有参加过比较　　W：　赢

L：输　　WL：赢过且至少输 1 次

比较后状态改变提供信息单位，状态不改变不提供信息单位

每增加1个W增加1个信息单位；每增加1个L也增加1个信息单位

两个变量通过一次比较增加的信息单位个数不同（0, 1, 2)

初始状态 N,N → 比较后状态 W,L：增加 2个信息单位

初始状态 W,N → 比较后状态 W,L：增加 1个信息单位

初始状态 W,L → 比较后状态 W,L：增加 0个信息单位

构造输入的方法

根据参与比较的两个变量的状态，调整对参与比较的两个变量的赋值，控制每次比较后得到的信息单位个数.从而使得为得到 $2n-2$个信息单位，该算法对所构造的输入至少要做$\lceil 3n/2 \rceil - 2$ 次比较.

对输入变量的分配方案

x 与 *y* 的状态	分配输入值的策略	新状态	提供的信息单位
N,N	***x>y***	**W,L**	**2**
W,N；WL,N	***x>y***	**W,L；WL,L**	**1**
L,N	***x<y***	**L,W**	**1**
W,W	***x>y***	**W,WL**	**1**
L,L	***x>y***	**WL,L**	**1**
W,L；WL,L；W,WL	***x>y***	不变	**0**
WL,WL	保持原值	不变	**0**

例如*A*作了8次比较：

x_1,x_2---$x_1>x_2$; x_1,x_5---$x_1>x_5$; x_3,x_4---$x_3>x_4$; x_3,x_6---$x_3>x_6$

x_3,x_1---$x_3>x_1$; x_2,x_4---$x_2>x_4$; x_5,x_6---$x_5>x_6$; x_6,x_4---$x_6>x_4$

比较	x_1	x_2	x_3	x_4	x_5	x_6
	状态 值	状态 值	状态 值	状态 值	状态 值	状态 值
	N *	N *	N *	N *	N *	N *
$x_1>x_2$	W 20	L 10				
$x_1>x_5$	W 20				L 5	
$x_3>x_4$			W 15	L 8		
$x_3>x_6$			W 15			L 12
$x_3>x_1$	WL 20		W 25			
$x_2>x_4$		WL 10		L 8		
$x_5>x_6$					WL 5	L 3
$x_6>x_4$				L 2		WL 3

构造输入为（20，10，25，2，5，3）

问题复杂度的下界

为得到$2n-2$个信息单位，对于上述输入A至少做$\lceil 3n/2\rceil-2$ 次比较.

根据分析，一次比较得到2个信息单位只有情况1. A至多有$\lfloor n/2\rfloor$个情况1，至多得到 $2\lfloor n/2\rfloor \le n$个信息单位. 其它情况, 1次比较至多获得1个信息单位，至少还需要$n-2$次比较.

当 n 为偶数，A至少做

$$\lfloor n/2\rfloor + n-2 = 3n/2-2 = \lceil 3n/2\rceil -2$$

次比较. 当 n 为奇数，A至少做

$$\lfloor n/2\rfloor + n-2 + 1 = (n-1)/2 + 1 + n-2 = \lceil 3n/2\rceil -2$$

次比较.

结论：FindMaxMin是最优算法

找第二大问题复杂度下界

定理7 对于任何从n个数中选第二大、并以比较作为基本运算的算法，都存在某个输入，使得对于这个输入算法至少做 $n+\lceil \log n \rceil -2$次比较.

证 不妨设n个数彼此不同.

任何找第二大的算法必须找最大. 因为要确认第二大，必须要确认它不是最大，即它要在1次比较中失败，而这次比赛中的胜者一定是最大. 确定最大比较至少为$n-1$次.

下面证明，对任何找第二大的算法A，存在某个输入使得max与$\lceil \log n \rceil$个元素比较过，这些元素中有1个第二大，而其余元素要在与第二大的比较中再次被淘汰.

找第二大问题
复杂度下界（续）

元素x的权：$w(x)$, 表示上述构造中以x为根的子树中的结点数

初始，$w(x_i)=1$，$i=1, 2, \ldots, n$;

如果 $x_i>x_j$, 则

$w(x_i)\leftarrow w(x_i)+w(x_j)$;

$w(x_j)\leftarrow 0$;

算法结束，根的权为 n，其他结点的权为0.

找第二大问题
复杂度下界（续）

赋值原则：

在比较的时候进行赋值或者调整赋值.

只对没有失败过的元素（权大于0的元素）进行赋值.

权大者胜，原来胜的次数多的仍旧胜，输入值也大.

1. $w(x), w(y)>0$:

   若 $w(x)>w(y)$, 那么$x>y$, x的值大于y 的值；//权大者胜

   若 $w(x)=w(y)$, 那么 $x>y$, x的值大于y 的值；//权等，任意分配

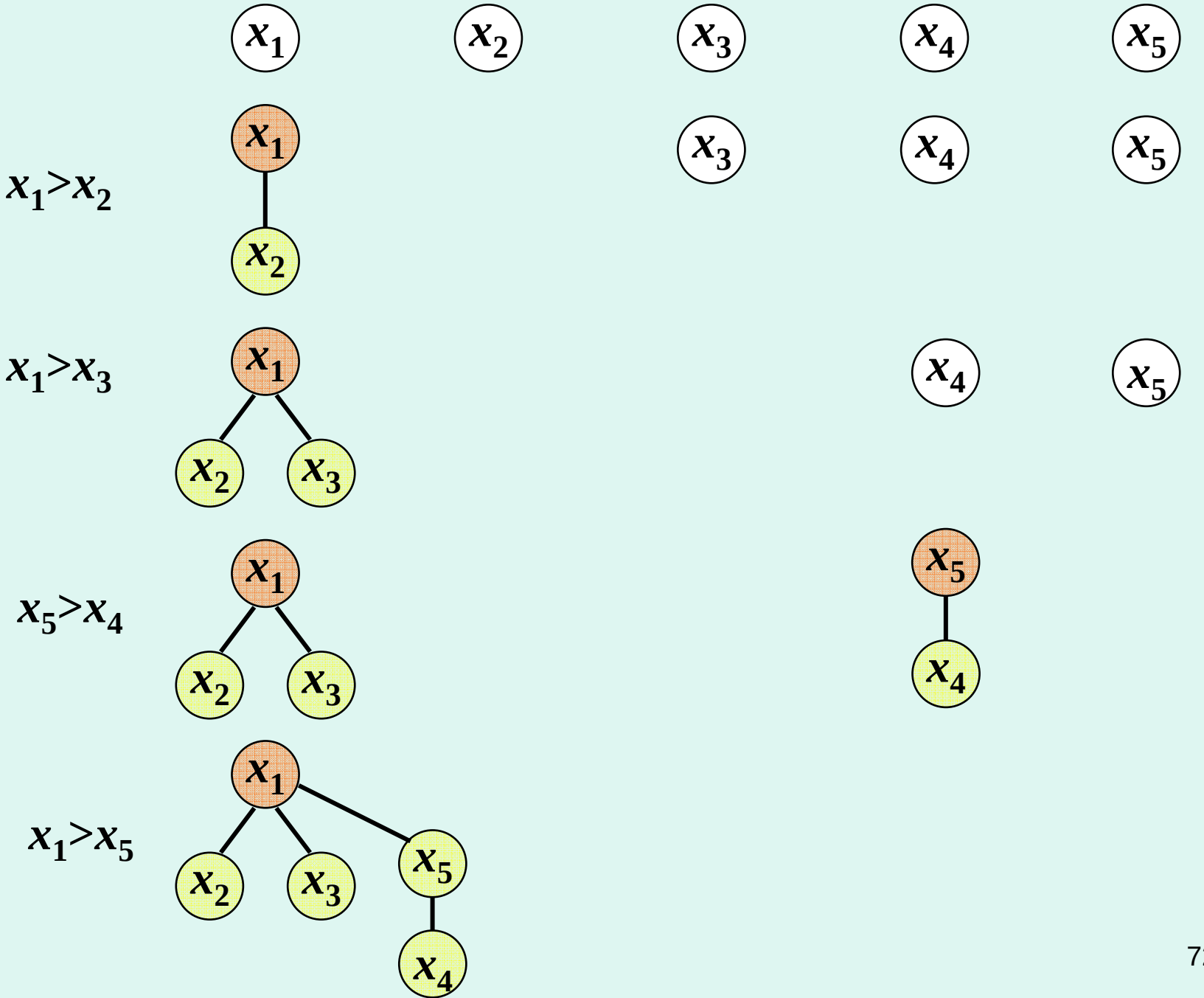
2. $w(x)=w(y)=0$, 那么x, y值不变； // x与 y 比较对于确定第二大无意义

找第二大问题
复杂度下界（续）

比较	$w(x_1)$	$w(x_2)$	$w(x_3)$	$w(x_4)$	$w(x_5)$	值
初始	1	1	1	1	1	*，*，*，*，*
第1步 $x_1>x_2$	2	0	1	1	1	20，10，*，*，*
第2步 $x_1>x_3$	3	0	0	1	1	20，10，15，*，*
第3步 $x_5>x_4$	3	0	0	0	2	20，10，15，30，40
第4步 $x_1>x_5$	5	0	0	0	0	41，10，15，30，40

根据算法 *A* 的比较次序, 在比最大的过程中如下构造树：

1. 初始是森林，含有 *n* 个结点；
2. 如果 *x, y* 是子树的树根，则算法比较 *x, y*；
3. 若 *x, y* 以前没有参加过比较，任意赋值给 *x, y*，比如 *x>y*；那么将 *y* 作为 *x* 的儿子；
4. 若 *x, y* 已经在前面的比较中赋过值，且 *x>y*，那么把 *y* 作为 *x* 的儿子，以 *y* 为根的子树作为 *x* 的子树；


x_1 x_2 x_3 x_4 x_5
$x_1 > x_2$
x_3 x_4 x_5
$x_1 > x_3$
x_4 x_5
$x_5 > x_4$
x_5 x_4
$x_1 > x_5$
x_4

找第二大问题
复杂度下界（续）

针对这个输入，估计与max比较而淘汰的元素数

1. 根max的权为 n，其它结点权为 0
2. w_k 表示 max 在它第 k 次比较后形成以 max 为根子树的结点总数，则 $w_k \le 2w_{k-1}$
3. 设 K 为max与权不为 0 (被max淘汰)结点比较次数，则

$$n = w_K \le 2^K w_0 \le 2^K \Rightarrow K \ge \log n \Rightarrow K \ge \lceil \log n \rceil$$

4. 这 K 个元素彼此不同，因为同一元素不可能 计数 2 次.
5. 为确定第二大，淘汰K–1个元素, 至少用$\lceil \log n \rceil$ –1 次比较.

结论：锦标赛方法是找第二大的最优算法.

找中位数算法复杂度下界

定理8 设n为奇数，任何通过比较运算找 n 个数的中位数(median)的算法在最坏情况下至少做 $3n/2–3/2$ 次比较

证 首先定义决定性的比较与非决定性的比较.

决定性的比较: 建立了x与 median 的关系的比较.

$\exists y$ ($x>y$ 且 $y \geq$ median), x满足上述条件的第一次比较

$\exists y$ ($x<y$ 且 $y \leq$ median), x满足上述条件的第一次比较

（比较时 y 与median的关系可以不知道）

非决定性的比较：当$x>$median, $y<$median, 这时$x>y$的比较不是决定性的.

为找到中位数，必须要做 $n–1$ 次决定性的比较.

针对算法构造输入，使得非决定性比较达到$(n–1)/2$次.

输入构造方法

1. 分配一个值给中位数 median；
2. 如果*A*比较*x*与*y*，且*x*与*y*没有被赋值，那么赋值*x*,*y*使得
   $x>\text{median}, y<\text{median}$;
3. 如果*A*比较*x*与*y*, 且$x>\text{median}$, *y*没被赋值，则赋值 *y* 使得
   $y<\text{median}$;
4. 如果*A*比较*x*与*y*, 且$x<\text{median}$, *y*没被赋值，则赋值*y*使得
   $y>\text{median}$;
5. 如果存在 $(n-1)/2$个元素已得到小于median的值，则对未赋值的全部分配大于median的值；
6. 如果存在 $(n-1)/2$个元素已得到大于median的值，则对未赋值的全部分配小于median的值.
7. 如果剩下1个元素则分配median给它.

例如 A 做如下比较：

x_1,x_2---$x_1>x_2$； x_3,x_4---$x_3>x_4$； x_5,x_6---$x_5>x_6$；

x_1,x_3---$x_1>x_3$；x_3,x_7---$x_3>x_7$； x_7,x_4---$x_7>x_4$； …

1. 初始　　median=4
2. $x_1>x_2$　　$x_1=7, x_2=1$
3. $x_3>x_4$　　$x_3=5, x_4=2$
4. $x_5>x_6$　　$x_5=6, x_6=3$

非决定性比较

5. $x_7=4$
6. $x_1>x_3$
7. $x_3>x_7$

决定性比较

8. …

输入构造方法（续）

元素状态 N：未分配值；S：得到小于median值；
L：得到大于median值

比较前的状态	分配策略
N, N	一个大于median，一个小于median
L, N 或 N, L	分配给状态N的元素的值小于median
S, N 或 N, S	分配给状态N的元素的值大于median

A 对所构造的输入进行的上述比较都是非决定性的. 这种比较至少有$(n-1)/2$个. 因此总比较次数至少为

$$(n-1)+(n-1)/2 = 3n/2-3/2$$

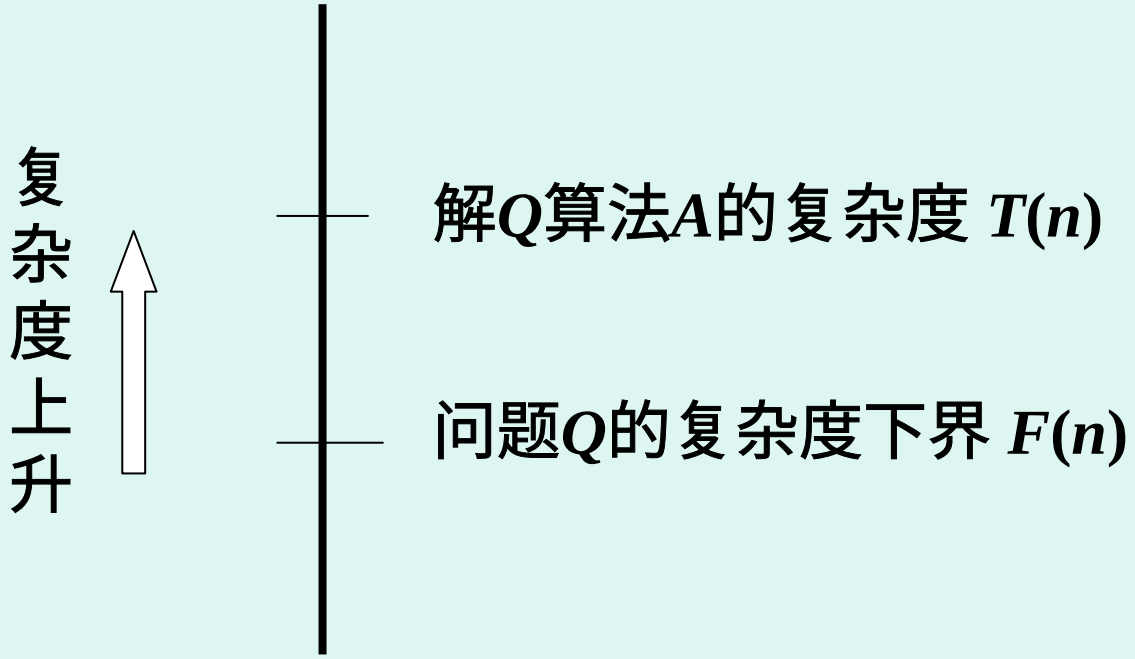
结论：Select算法在阶上达到最优.

几种选择算法的总结

问题	算法	最坏情况	问题下界	最优性
找最大	Findmax	n-1	n-1	最优
找最大最小	finmaxmin	$\lceil 3n/2 \rceil$-2	$\lceil 3n/2 \rceil$-2	最优
找第二大	锦标赛	$n+\lceil \log n \rceil$-2	$n+\lceil \log n \rceil$-2	最优
找中位数	Select	$O(n)$	$3n/2-3/2$	阶最优
找第k小	Select	$O(n)$	$n+\min\{k,n-k+1\}-2$	阶最优

线性时间的归约

- 问题P, 问题Q：问题Q的复杂度下界已知
- 存在变换 f 将Q的任何实例转换成 P 的实例，f 的时间为线性时间
- 解Q的算法 A：
  1. 将Q的实例 I 变成 $f(I)$,
  2. 用解 P 的算法作为子程序解 $f(I)$，时间与解P的时间为同样的阶



$p(n)$为任意解 P 的算法 B 的复杂度

A：将Q的实例 I 转换成 P 的实例 $f(I)$	$t_1(n)$
调用 B 解 $f(I)$，得到解 s	$p(n)$
将 s 转换成对 I 的解	$t_2(n)$

$T(n)=t_1(n)+p(n)+t_2(n)=\Theta(p(n))$

$p(n)=\Theta(T(n))=\Omega(F(n))$

实　例

- P问题与Q问题

  P：平面直角坐标系中 n个点的最小生成树问题
  Q：元素的唯一性问题

- 元素唯一性问题的复杂度为 $\Theta(n\log n)$

  输入：多重集 $S=\{ n_1\cdot a_1, n_1\cdot a_1, \ldots , n_k\cdot a_k \}$
  构造判定树，树叶为 S 的全排列数

$$\frac{n!}{n_1!\, n_2! \ldots n_k!}$$

  最坏情况下树深　$\Theta(\log n!) = \Theta(n \log n)$

实　例

- 变换 f

  Q 的实例：$x_1, x_2, \ldots, x_n$，变成 X 轴上的 n 个点

- 算法

  1. 利用求最小生成树算法构造树T，确定T 的最短边e.

  2. 检测 e 的长度是否为**0**

  3. if $|e|=0$ **then** 不唯一，**else** 是唯一的.

- 结论

  平面直角坐标系中n个点的最小生成树问题是$\Omega(n\log n)$.

第三章 计算复杂性理论

Turing机

计算复杂性理论

NP完全性理论的基本概念

NP完全性证明

用NP完全性理论分析问题

NP难度

Turing机

Turing机的定义

基本模型

基本Turing机的变种

单向带的Turing机

*k*条带的Turing机

非确定型的Turing机

Turing机模型接受语言能力的等价性

单向带Turing机与基本Turing机等价

k 条带的Turing机与基本Turing机等价

非确定型Turing机与基本Turing机等价

Turing机的定义

基本模型　双向无限带的**Turing**机

$M = <Q, \Sigma, \Gamma, \delta, q_0, B, F>$, 其中

Q 有穷状态集

Γ 有穷带字符集

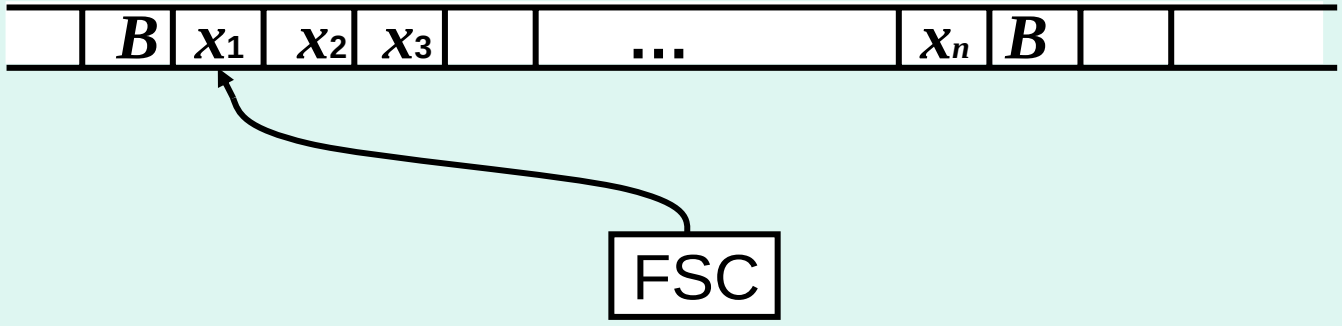
Σ 输入字符集　　$\Sigma \subset \Gamma$

B 空白字符,　　$B \in \Gamma - \Sigma$

q_0 初始状态,　　$q_0 \in Q$

F 终结状态集,　$F \subset Q,\ q_Y, q_N \in F$

$\delta: (Q-F) \times \Gamma \to Q \times \Gamma \times \{L,R\}$　　状态转移函数



瞬时描述（ID）

$\alpha_1 q \alpha_2$ 表示此刻 **Turing**机的 **FSC** 处于状态 q, 读写头指在串 α_2 的第一个字符.

例如**Turing**机 M 的某时刻的状态转移函数是

$$\delta(q, x_i) = (p, Y, L)$$

带上的字符串为 $x_1 x_2 \ldots x_i \ldots x_n$, 读写头指向字符 x_i, 则它的瞬间描述是:

$$x_1 x_2 \ldots x_{i-1} q\, x_i \ldots x_n \vdash x_1 x_2 \ldots x_{i-2}\, p\, x_{i-1} Y x_{i+1} \ldots x_n$$

$\vdash$ 表示由左边的**ID**一步达到右边的**ID**

$\vdash^*$ 表示由左边的**ID**经有限步达到右边的**ID**

Turing机接受语言

被 M 接受的语言记作$L(M)$，是Σ^*上的字的集合.

当这些字左端对齐方格 1 放在带上, M 处于状态 q_0, M 的带头指向方格 1 时, 经过有限步 M 将停机在接受状态 q_Y, 即

$$L(M)=\{\omega \mid \omega\in\Sigma^*, \exists\alpha_1,\alpha_2\in\Gamma^*(q_0\omega \vdash^*\alpha_1 q_Y\alpha_2)\}$$

如果字ω不是$L(M)$中的字, M可以不停机或停机在拒斥状态q_N.

实 例

例1 $L=\{ 0^n1^n \mid n \geq 1 \}$, 设计接受 L 的 Turing机如下:

$M = <Q, \Sigma, \Gamma, \delta, q_0, F>$

$Q = \{ q_0, q_1, q_2, q_3, q_Y \}$,

$\Sigma = \{ 0, 1\}$,

$\Gamma = \{ 0, 1, X, Y, B \}$,

$F = \{ q_Y \}$. 其中 q_Y 代表接受停机状态

初始将字符串放在从 1 到 n 方格中, FSC处在状态 q_0, 读写头指向方格 1. 将第一个0改写成 X, 然后带头向右扫描. 遇到第一个1, 将1改为Y, 然后带头向左扫描. 遇到第一个 X 改为向右扫描. 这时进入下一个巡回. 每个巡回将一对 0 和 1 改为 X 和 Y, 直到接受或拒斥停机.

实例（续）

转移函数如下（其中_代表拒斥停机状态 ）

	0	1	X	Y	B
q_0	(q_1,X,R)	_	_	(q_3,Y,R)	_
q_1	$(q_1,0,R)$	(q_2,Y,L)	_	(q_1,Y,R)	_
q_2	$(q_2,0,L)$	_	(q_0,X,R)	(q_2,Y,L)	_
q_3	_	_	_	(q_3,Y,R)	q_Y

例如输入 **0011**，**Turing** 机动作如下：

$$q_00011 \vdash Xq_1011 \vdash X0q_111 \vdash Xq_20Y1 \vdash q_2X0Y1$$

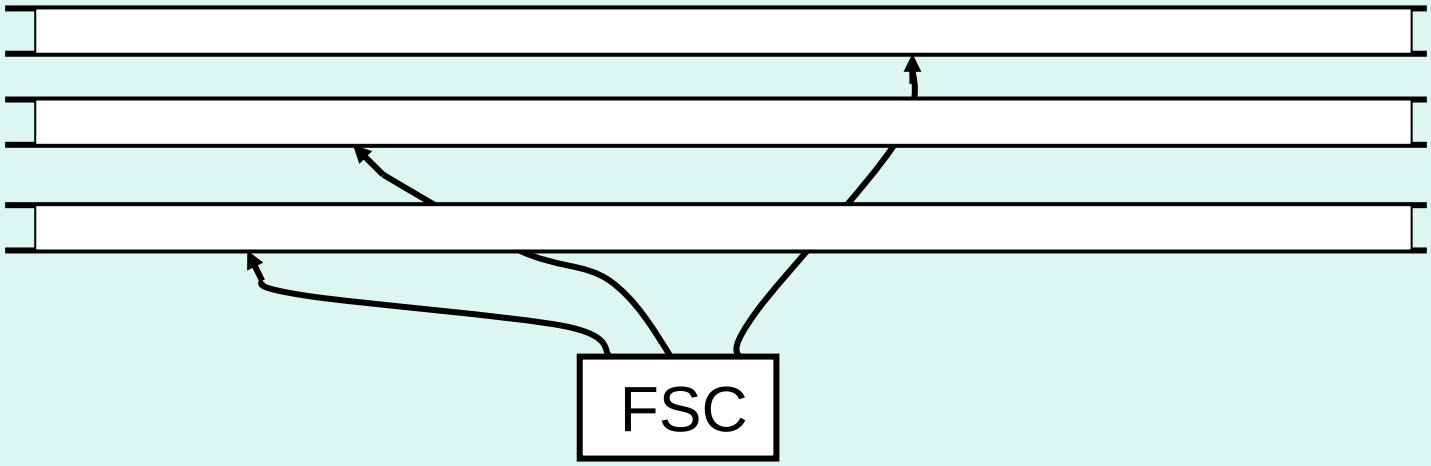
$$\vdash Xq_00Y1 \vdash XXq_1Y1 \vdash XXYq_11 \vdash XXq_2YY \vdash Xq_2XYY$$

$$\vdash XXq_0YY \vdash XXYq_3Y \vdash XXYYq_3 \vdash XXYYq_Y$$

基本Turing机的变种

单向无限带的 Turing 机 带方格从1开始, 向右无限长. 其它与基本 Turing 机相同.

多带的 Turing 机 k 条双向带, k 个读写头, 其中 k 为大于 1 的常数. 初始将输入写在第一条带的方格 1 到 n 内. 其它带为空. 每个读写头扫描一条带,可以改写被扫描方格的字符, 读写头然后向左或向右移动一个方格. 读写头的动作由 FSC 的状态及 k 条带所扫描的 k 个字符来决定.

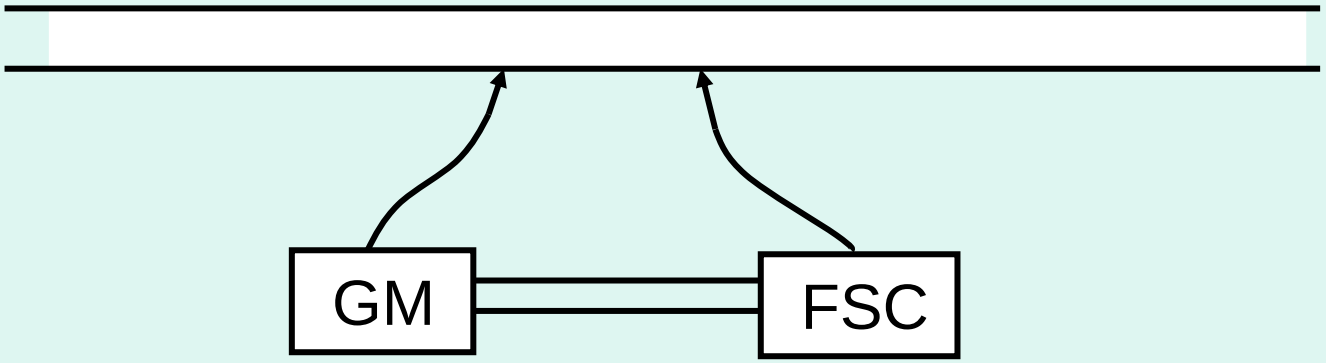


基本Turing机的变种（续）

非确定型的 Turing 机 (NDTM)

一个有限状态控制器 FSC, 猜想模块 GM, 读写头, 只写头, 双向无限带.

初始 FSC 处于q_0状态, 带方格的 1 到 *n* 写上输入 *x*, 其它方格为空白字符*B*. 读写头指在方格1, 只写头指在方格-1. 计算分为猜想阶段和检查阶段.



基本Turing机的变种（续）

猜想阶段

FSC 处于q_0状态下的待用状态.

猜想模块 **GM** 指挥只写头,每次一步在所扫描的方格内写下Γ中的某个字符, 然后左移一个方格或不动.

到某个时刻, 猜想模块进入待用状态, 这时有限状态控制器进入 q_0 状态下的活跃状态. 从此刻起, 计算进入检查阶段， 而猜想模块是否继续动作, 或怎样动作, 完全是任意的.

检查阶段

与确定型的 **Turing** 机完全一样. 如果 **FSC** 进入接受状态, 则计算停止, 接受 x. 如果进入拒斥状态, 则计算停止, 不接受x.

Turing机的等价性

可以证明几种 Turing 机在接受语言的能力上是等价的.

证明思路：相互模拟

双向带模拟单向带：左边做记号

单向带模拟双向带：将双向带从中间剪断、对折

k 条带模拟 1 条带：只用1条

1 条带模拟 k 条带：将1条带分成2k道, 用2个巡回模拟1个动作

非确定型的模拟确定型的：检查阶段模拟

确定型的模拟非确定型的：穷举接受计算所有可能的猜想

单向带Turing 机
与基本Turing 机等价

定理1 语言 L 被一个具有双向无限带的**Turing**机 M_2 接受当且仅当 L 被一个具有单向无限带的**Turing**机 M_1 接受.

证明思路：相互模拟

用 M_2 模拟 M_1.

在读写头的初始位置的左侧单元做记号, 然后模拟 M_1 的动作. 如果在模拟期间到达做过记号的单元, 则 M_2 停机在拒斥状态.

证明单向带模拟双向带

用 M_1 模拟 M_2. 设 $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_2, B, F_2)$,
构造 M_1 如下:

$M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, B, F_1)$. 其中

$Q_1 = \{ [q,U], [q,D] \mid q \in Q_2 \} \cup \{ q_1 \}$

$\Gamma_1 = \{ [X,Y] \mid X \in \Gamma_2 \wedge (Y \in \Gamma_2 \vee Y = \#) \}$

$\Sigma_1 = \{ [a,B] \mid a \in \Sigma_2 \}$

$F_1 = \{ [q,U], [q,D] \mid q \in F_2 \}$

	A_{-2}	A_{-1}	A_0	A_1	A_2	A_3	A_4	

A_1	A_2	A_3	A_4	
#	A_0	A_{-1}	A_{-2}	

证明单向带模拟双向带（续）

初始 M_1处于q_1, 读写头指在第一个方格

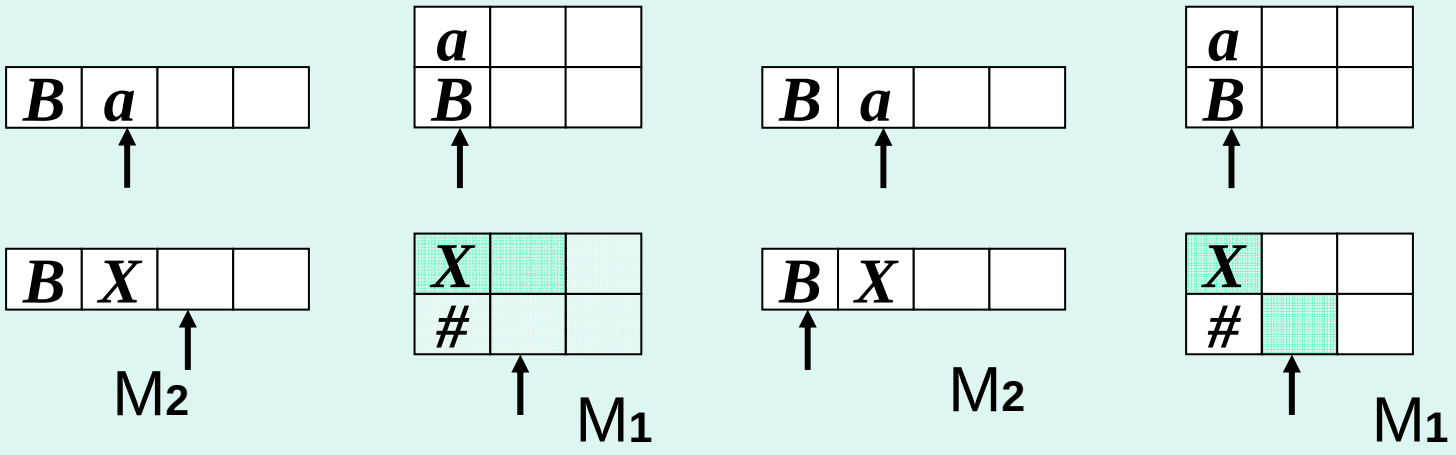
第一步

$\delta_2(q_2,a) = (q,X,R)$

$\delta_1(q_1,[a,B]) = ([q,U],[X,\#],R)$

$\delta_2(q_2,a) = (q,X,L)$

$\delta_1(q_1,[a,B]) = ([q,D],[X,\#],R)$



证明单向带模拟双向带（续）

M_2的动作不经过方格0, 此刻M_1的状态为$[q,U]$或$[q,D]$, 读写头字符为 $[X,Y]$.

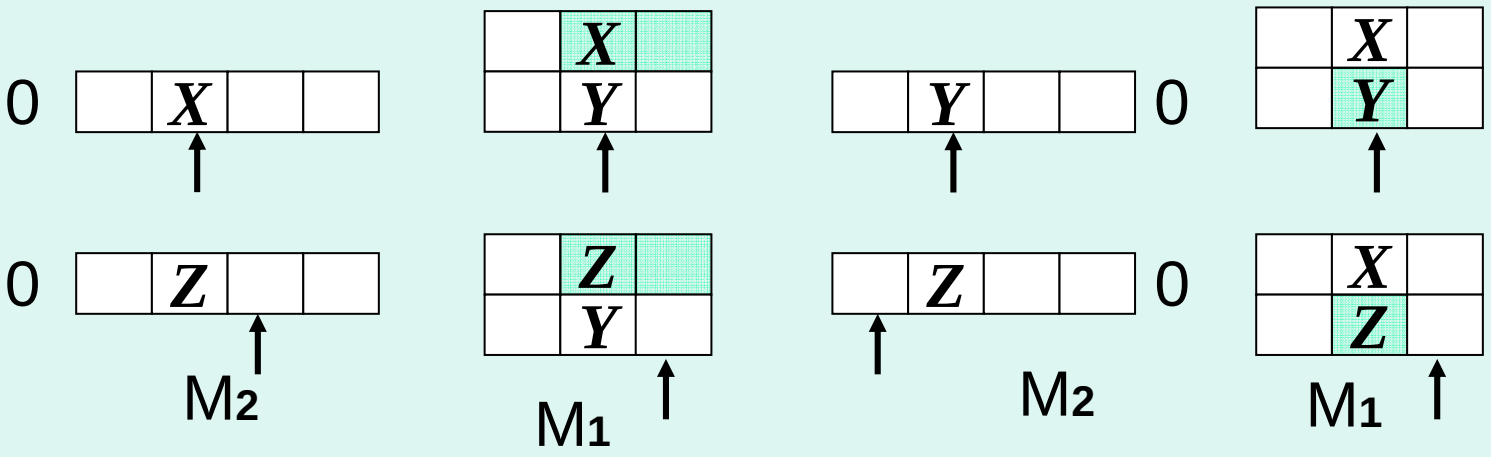
若M_2带头在方格1的右边, 且 $\delta_2(q,X) = (p,Z,A)$, $A = R$ 或 L, 则

$$\delta_1([q,U],[X,Y]) = ([p,U],[Z,Y],A)$$

若M_2带头在方格0的左边, 且 $\delta_2(q,Y) = (p,Z,A)$, $A = R$ 或 L, 则

$$\delta_1([q,D],[X,Y]) = ([p,D],[X,Z],\bar{A}).$$

其中$\bar{A}$与 A 的方向相反.



证明单向带模拟双向带（续）

M_2的读写头指在方格1, 向左移到方格0.

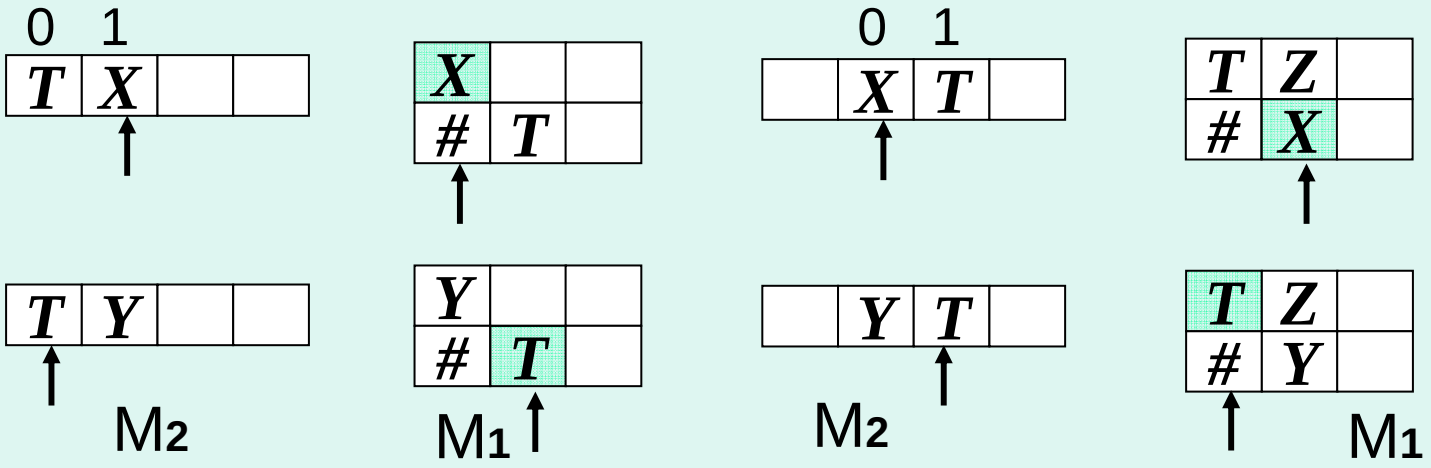
M_1状态为$[q,U]$, 读写头字符为$[X,\#]$，且 $\delta_2(q,X) = (p,Y,L)$，则

$$\delta_1([q,U],[X,\#]) = ([p,D],[Y,\#],R)$$

M_2的读写头指在方格0, 向右移到方格1.

M_1状态为$[q,D]$, 读写头字符为$[Z,X]$，且 $\delta_2(q,X) = (p,Y,R)$，则

$$\delta_1([q,D],[Z,X]) = ([p,U],[Z,Y],L)$$



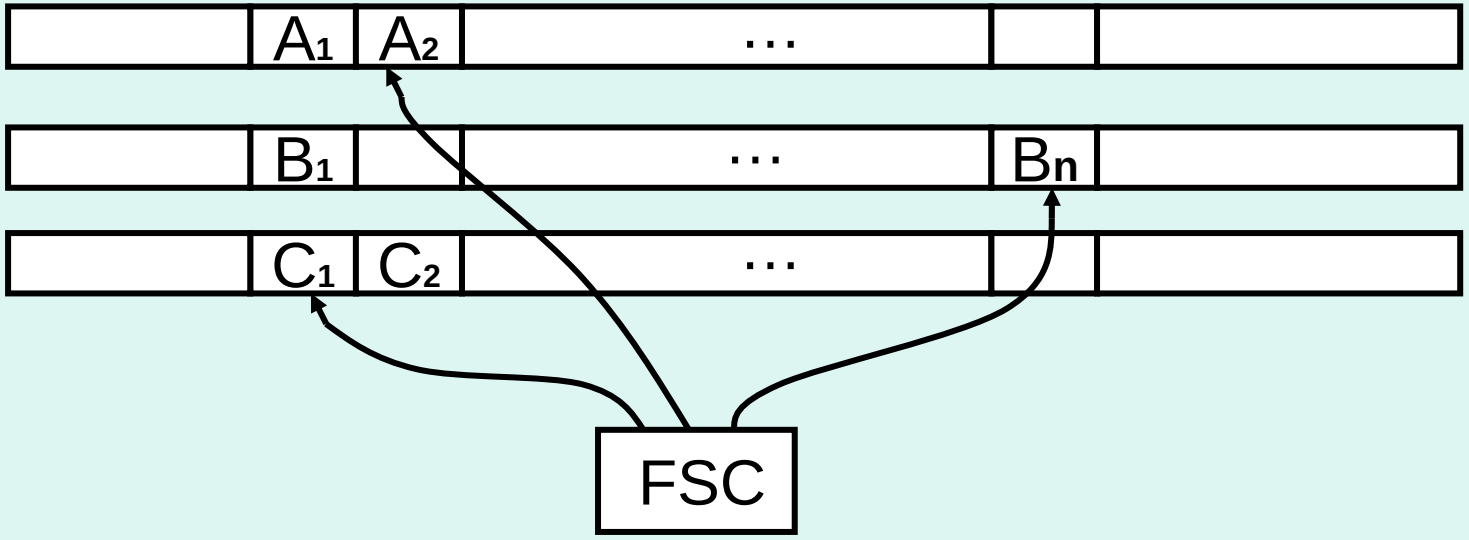
模拟复杂性：一步模拟一步. M_1和M_2的时间复杂性一样.

k条带Turing 机
与基本Turing 机等价

定理2 语言L被一个k 条无限带的 Turing机 M_1接受当且仅当 L 被一个双向无限带的 Turing机 M_2接受.

证 显然 M_1可以模拟 M_2.

设 L可以被 M_1接受. 如下构造 M_2:



k条带Turing 机与基本Turing 机等价（续）

把一条带划分成 **2*k*** 道，分别对应 ***k*** 条带的带头位置和带上的字符串，上面 **3** 条带对应下面**1** 条带

head1		×				
tape1	A_1	A_2				
head2					×	
tape2	B_1	B_2			B_n	
head3	×					
tape3	C_1	C_2				

k条带Turing 机与基本Turing 机等价（续）

为了模拟M_1的一个动作, M_2的读写头要从左至右, 再从右至左做一次巡回.

开始, M_2的读写头在最左边的单元, 然后M_2向右边巡视, 访问每一个具有带头记号的单元, 并记录下M_1的每一个带头所扫描的符号(用状态记录). 当 M_2 越过一个带头记号时, 必须修改右方带头标记的数目.

当右边不存在带头标记时, M_2开始向左巡视. 根据右边带头标记数目的变化, 使得M_2知道何时巡视到头.

M_2向左通过每个带头记号时, 修改M_1的被该带头扫视的符号, 并将该带头向左或向右移动一个单元.

最后, M_2根据M_1状态的变化改变自己的状态, 从而完成对M_1一个动作的模拟. 如果M_1的新状态为接受, 则M_2进入接受状态.

k条带Turing 机与基本Turing 机等价（续）

模拟复杂性:

在 M_1 的第 i 步，带头最远距离至多为 $2i$

M_2 对 M_1 的第 i 步动作的模拟至多 $4i$ 步

如果 M_1 的接受计算有 k 步

则 M_2 的接受计算的步数至多为

$$\sum_{i=1}^{k} 4i = 2k(k+1) = O(k^2)$$

称为2次方减速效应.

非确定型Turing 机与基本Turing 机等价

定理3 语言 L 被一个非确定型的Turing机 M_1接受
当且仅当 L 被一个确定型的Turing机 M_2接受.

证 用 M_1模拟 M_2, 只须在检查阶段模拟 M_2的动作.
考虑用 M_2模拟 M_1.
L中的串 x 可以被 M_1 在有限步接受 (设为$f(n)$步), 因此猜想串的长度至多为 $f(n)$. 易见长度不超过 $f(n)$的猜想串至多有 $|\Gamma|^{f(n)}$ 个.
设 M_2有3条带. 1条用于保存输入串, 1条依次写下每一个猜想串. 对于每个猜想串, M_2 将输入复制到第3条带, 然后根据猜想串在第3条带模拟 M_1 的检查阶段的动作. 如果 M_1进入接受状态, 则 M_2 也进入接受状态.

模拟复杂性: 如果 M_1 接受计算步数为 $f(n)$, M_2的步数为 $f(n)|\Gamma|^{f(n)}$

计算复杂性理论

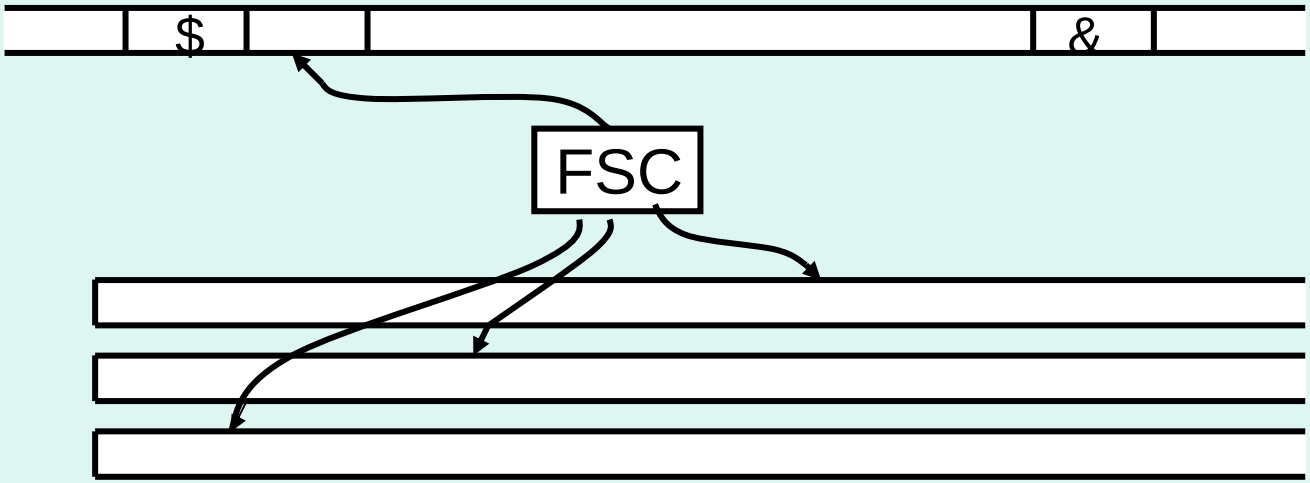
空间和时间复杂度的形式定义

- 确定型**Turing**机
- 非确定型**Turing**机

计算复杂度的有关结果

- 带压缩
- 线性加速
- 带数目的减少

确定型Turing机
空间复杂度的形式定义



离线的**Turing**机 M, **1**条具有端记号的只读输入带, k条半无限存储带. 如果对每个长为n的输入串, M在任一条存储带上都至多扫视 $S(n)$个 单元, 那么称 M 在最坏情况下的空间复杂度为 $S(n)$.

确定型Turing机
时间复杂度的形式定义

k 条双向带的 **Turing**机 ***M***, 一条带包含输入.

如果对于每个长为***n***的输入串, ***M*** 在停机前至多做 ***T*(*n*)** 个动作, 那么称***M***在最坏情况下的时间复杂度为 ***T*(*n*)**.

两条假设:

空间复杂性至少需要 **1**

时间复杂性至少需要读入输入的时间,

因此这里作如下假定:

对一切***n***, ***S*(*n*)≥1**, **log*n*** 是 **max { 1,⌈log*n*⌉ }** 的缩写.

对一切***n***, ***T*(*n*)≥*n*+1**, ***T*(*n*)**是 **max { *n*+1,*T*(*n*) }** 的缩写.

实 例

例1 $L = \{ wcw^R \mid w$为0-1字符串$\}$, 设计接受 L 的Turing机 M_1和 M_2, 使得 M_1的时间复杂度为$O(n)$, M_2的空间复杂度为 $O(\log n)$.

M_1有2条带,把 c 左边的 w 复制到第2条带上. 当发现 c 时第2条带的读写头向左, 输入带的读写头向右. 比较两个带头的符号, 如果符号一样, 字符个数一样, M_1 接受 x. M_1 至多作 $n+1$ 个动作. 时间复杂度为 $n+1$. 空间复杂度为$\lceil (n-1)/2 \rceil + 1$.

M_2有2条带, 第2条带作为二进制的计数器. 首先检查输入是否只有1个 c, 以及 c 左边和右边的符号是否一样多. 然后逐个比较 c 左边和右边的字符, 用上述计数器找到对应的字符. 如果所有的字符都一样, M_2 接受停机. 空间复杂度为二进制的计数器的占用空间, 即 $O(\log n)$. 时间复杂度为 $O(n^2)$.

非确定型Turing机
计算复杂度的形式定义

给定串 x, M 接受 x 的时间: M关于x 的所有接受计算中, 在猜想阶段和检查阶段直到进入接受停机状态为止所发生步数的最小值.

M 的最坏情况下的时间复杂度

$T(n) = \max\{ m \mid$ 存在 $x \in L, |x|=n, M$ 接受 x 的时间为$m \}$

给定串x, M 接受 x 的空间: M关于x 的所有接受计算中, 在猜想阶段和检查阶段直到进入接受停机状态为止在存储带上扫视的最少单元数.

M 的最坏情况下的空间复杂度

$S(n) = \max\{ m \mid$ 存在$x \in L, |x|=n, M$ 接受 x 的空间为$m \}$

计算复杂度的有关结果

带压缩、线性加速和带数目的减少

带压缩：空间占用的线性压缩

由于Turing机的状态数和带字符集的大小可以是任意给定的常数. 可以将若干个带字符编码成一个字符, 所以空间占用量总可以压缩一个常数因子.

线性加速：计算时间加速一个常数因子

这就是在计算复杂性研究中要强调函数阶的原因

带数目的减少：在空间或时间复杂度不变的情况下减少带的数目.

这里对Turing机进行一点修改, 允许带头原地不动.

带压缩

定理1 如果语言 L 被一个具有 k 条存储带 $S(n)$ 空间有界的 Turing机 (TM) 接受, 则对任意 $1>c>0$, L 被一个$cS(n)$空间有界的 TM 接受.

证 设 M_1是接受 L 的 $S(n)$ 空间有界的离线TM, 如下构造 M_2, 使得M_2 是 $cS(n)$ 空间有界的, 并且接受 L.

对于某个常数 r, M_2 的每个存储单元保存一个符号, 代表了M_1 相应带上 r 个相邻单元的内容.

M_2的输入字符集 Σ 与M_1相同， 带字符集为 Σ^r

带压缩 (续)

M_2 的有限状态控制器能够记住在所代表的那些符号中 M_1 的哪个单元是真正被扫视的单元. 如果 M_1 的状态为q, M_2 的状态为 $(q, i_1, i_2, \ldots, i_k)$. 其中 $i_j \in \{1, 2, \ldots, r\}$, 表示 M_1 的第 j 条带读写头的位置. 令 $rc \geq 2$, 在任何带上至多用 $\lceil S(n)/r \rceil$ 个单元, M_2 就可以模拟 M_1.

若$S(n) \geq r$, 则

$$\left\lceil \frac{S(n)}{r} \right\rceil \leq \frac{S(n)}{r} + 1 \leq \frac{2S(n)}{r} \leq cS(n)$$

若$S(n) < r$, 则 M_2 只要一个单元就可将任何一条带的内容存入. 而空间复杂度至少是 1. 也满足要求.

时间线性加速

$f(n)$是n的函数，$\liminf_{n\to\infty} f(n)$是$n\to\infty$时 $f(n)$, $f(n+1)$,...的最大下界的极限

定理2 如果语言 L 被一个 k 带 $T(n)$ 时间有界的TM M_1接受, 那么只要 $k>1$, 且

$$\liminf_{n\to\infty} \frac{T(n)}{n} = \infty$$

那么，L 就可以被一个 k 带 $cT(n)$ 时间有界的TM M_2 接受, 其中 c 是大于 0 的任意常数.

时间线性加速(续)

证明思路：构造 M_2 来模拟 M_1

（1）将 M_1 带字符集上的 m 个字符编码成 M_2 字符集上的 1 个大字符，以使得 M_2 处理1个字符相当于 M_1 处理 m 个字符.

（2）M_1 动作 m 步，M_2 至多通过 8 步来模拟 M_1，两个机器动作后的结果一样.

（3）M_1的计算时间为 $T(n)$，证明 M_2 的模拟时间至多为

$$n+\frac{n}{m}+8\frac{T(n)}{m}+9$$

（4）对于给定的 $c(0<c<1)$ 确定 m 的值，以使得

$$n+\frac{n}{m}+8\frac{T(n)}{m}+9\le cT(n)$$

证明（1）
----编码大字符

M_2 将输入复制到一条存储带上，同时将 m 个符号编码成一个符号(m待定). 从现在起, M_2 用这条存储带作为输入带, 并将原来的输入带作为存储带.

被 M_2 带头扫描的单元称为基地单元. 对于每一条带, M_2 的有限状态控制器都要记下带上基地单元所代表的 M_1 的 m 个符号中的哪一个正在被扫描.

根据 M_1 的转移函数可以确定 M_2 的转移函数.

证明 （2）
----M_2的基本步

M_2在1个基本步模拟 M_1 的 m 个动作.

基本步分成两个过程：获取信息，处理信息

获取信息过程:

M_2 将每个带头向左移1次, 向右移2次, 再左移1次, 同时把基地单元左右相邻单元的符号记录在有限状态控制器中. 这至多需要M_2的4个动作.

在这些动作后, M_2又回到它的基地单元.

证明 （2）

----M_2的基本步

处理过程：

如果 M_1 在某个带头离开由基地单元及其左右邻单元所代表的区域之前接受停机, 那么 M_2 就接受停机.

否则，根据 M_2 获取的基地单元和左右邻单元的内容，M_2 的转移函数将确定 M_2 的下步动作. M_2 在每条带上再次访问基地单元的两个相邻单元. 必要的话,它要改写这些符号以及基地单元的符号. 最后，M_2 把它的带头放到一个单元上, 这个单元代表了在这次模拟动作结束时 M_1 相应的带头正扫视的符号. 这至多需要 M_2 的4个动作.

M_1 要将一个带头移出由基地单元和相邻单元所代表的区域至少需要 m 个动作. 因此 M_2 在8个动作中至少模拟了 M_1 的 m 个动作.

证明（3）

----模拟时间

M_1的时间：$T(n)$

M_2的模拟时间：$n+\frac{n}{m}+8\frac{T(n)}{m}+9$

对输入复制和编码 (m个单元编码成1个字符)：n

将新的输入带的带头移回左端：$\lceil n/m \rceil$

模拟M_1的$T(n)$个动作：最多 $8\lceil T(n)/m \rceil$

$$n+\left\lceil \frac{n}{m} \right\rceil+8\left\lceil \frac{T(n)}{m} \right\rceil \le n+\frac{n}{m}+1+8\frac{T(n)}{m}+8$$

$$\le n+\frac{n}{m}+8\frac{T(n)}{m}+9$$

证明（4）
----m的确定

（4）针对给定的c, $0<c<1$，确定m, 使得 $cm \geq 16$

$$\liminf_{n\to\infty} \frac{T(n)}{n} = \infty$$

即对任意常数 d 都存在n'，使得对所有的 $n \geq n'$都有

$$\frac{T(n)}{n} \geq d \quad \Rightarrow \quad n \leq \frac{T(n)}{d}$$

从而, 当 $n \geq \max\{9, n'\}$有

$$n + \frac{n}{m} + 8\frac{T(n)}{m} + 9 \leq 2n + \frac{n}{m} + 8\frac{T(n)}{m}$$

$$= n(2 + \frac{1}{m}) + 8\frac{T(n)}{m} \leq T(n)(\frac{2}{d} + \frac{1}{md} + \frac{8}{m})$$

证明（4）

----m的确定

要使上式小于等于$cT(n)$, 即

$$\frac{2}{d}+\frac{1}{md}+\frac{8}{m}\le c \quad \Rightarrow \quad \frac{2m}{d}+\frac{1}{d}+8\le cm$$

只要 $$\frac{2m}{d}+\frac{1}{d}+8\le 16 \quad \Rightarrow \quad \frac{2m}{d}+\frac{1}{d}\le 8$$

即 $$d=\frac{m}{4}+\frac{1}{8}$$

结论：对任意 c ($0<c<1$)，存在n', 当 $n \ge \max\{9, n'\}$ 时, M_2 做的动作不超过$cT(n)$.

证明（4）

----m的确定

如果 $n<\max\{9, n'\}$, 只有有限个输入串.

可以在 M_2 的设计时就确定哪个串是可接受的，哪个串不是可接受的.

M_2 的动作：利用它的有限状态控制器, 用 n 个动作读入输入,到达作为输入末端的空白字符，然后用 M_2 的转移函数 1 步就使得有限状态控制器到达接受或拒斥状态.

M_2的时间复杂性仍是$cT(n)$, 因为

$$cT(n) = \max\{ n+1, cT(n) \}$$

推 论

推论1　如果 $\liminf_{n\to\infty} \frac{T(n)}{n} = \infty$

那么对任何 c ($0<c<1$), DTIME($T(n)$) = DTIME($cT(n)$)

证 如果 L 被k ($k>1$) 带TM在 $T(n)$ 时间内接受, 则 L 可以被 k 带 TM在 $cT(n)$ 时间接受.

如果 L 被单带 TM 在 $T(n)$ 时间内接受, 那么它可被 2 带的 TM 在 $T(n)$ 时间内接受. 从而被 2带 TM 在 $cT(n)$ 时间内接受.

推论2 如果 $\liminf_{n\to\infty} \frac{T(n)}{n} = \infty$

那么对任何c($0<c<1$), NTIME($T(n)$) = NTIME($cT(n)$)

时间加速

定理3 如果对于$k>1$和某个常数$c>1$, L被一个k带cn时间有界的TM接受, 则对于每个$\varepsilon>0$, L被一个k带$(1+\varepsilon)n$时间有界的TM接受.

证 取$m=16c/\varepsilon$即可.

$$n+\left\lceil\frac{n}{m}\right\rceil+8\left\lceil\frac{T(n)}{m}\right\rceil=n+\left\lceil\frac{n\varepsilon}{16c}\right\rceil+8\left\lceil\frac{cn\varepsilon}{16c}\right\rceil$$

$$\le n+\frac{n\varepsilon}{16c}+\frac{8cn\varepsilon}{16c}+9=n\left[1+\left(\frac{1}{16c}+\frac{1}{2}\right)\varepsilon\right]+9$$

$$\le n(1+\varepsilon)+9$$

推 论

推论1 如果对于某个$c>1, T(n) = cn$, 则对任何 $\varepsilon>0$,

$$\mathbf{DTIME}(T(n)) = \mathbf{DTIME}((1+\varepsilon)n)$$

推论2 如果对于某个$c>1, T(n) = cn$, 那么对于任何 $\varepsilon>0$,

$$\mathbf{NTIME}(T(n)) = \mathbf{NTIME}((1+\varepsilon)n)$$

带数目的减少

定理4 如果语言 L 被一个具有 k 条存储带 $S(n)$空间有界的 TM 接受, 那么 L 可以被一个具有 1 条存储带的 $S(n)$ 空间有界的 TM接受.

证 设 M_1是接受 L 的具有 k 条带 $S(n)$空间有界的 TM, 如下构造M_2, M_2 的带被划分成 $2k$ 条道, 如上节的定理2, M_2 在 $2k$ 条道上模拟 M_1 的 k 条带. M_2使用的单元数不超过 $S(n)$.

结论: 可以假设任何 $S(n)$ 空间有界的 TM 只有 1 条带, 且如果 $S(n)\geq n$, 那么该 TM是一个单带的 TM, 而不是一个离线的 TM.

带数目的减少（续）

定理5 如果 L 在 DTIME(T(n)) 中, 那么 L 可被一个单带 TM 在时间 $T^2(n)$ 内被接受.

证 M_1 是 $T(n)$ 时间有界的 k (k>1)带 TM.

存在M_1’ (k 带）在 $\frac{T(n)}{\sqrt{2}}$ 时间接受 M_1（线性加速）.

M_2 是单带 TM. 模拟 k 带的 M_1’，根据二次方减速效应时间为

$$2\left(\frac{T(n)}{\sqrt{2}}\right)^2 = \frac{2T^2(n)}{2} = T^2(n)$$

推论 L 在 NTIME($T(n)$) 中, 则 L 可以被一个非确定型 $T^2(n)$ 时间有界的 TM 接受.

带数目的减少（续）

定理6 如果 L 可以被一个 k 带 $T(n)$ 时间有界的TM接受, 则 L 可以被一个2带 TM在 $T(n)\log T(n)$ 时间内接受.

推论 如果 L 被一个 k 条带 $T(n)$ 时间有界的NDTM接受, 则 L 也可以被一个双带 $T(n)\log T(n)$ 时间有界的NDTM接受.

小结：减少带数目，可以保证空间复杂性不变.

减少带数目，时间复杂性增加.

有关复杂度的重要结果

	带数	M_2 模拟 M_1 的复杂度		类型
M_1	k带	$S(n)$		带压缩
M_2	k带	$cS(n)$, $\forall 0<c<1$		
M_1	k带	$T(n)$ $\liminf_{n\to\infty} \frac{T(n)}{n} = \infty$	cn,$\forall c>1$	时间加速
M_2	k带	$cT(n)$, $\forall 0<c<1$	$(1+\varepsilon)n$	
M_1	k带	$S(n)$		带数目减少 空间不变
M_2	1带	$S(n)$		
M_1	k带	$T(n)$		带数目减少 时间增加
M_2	1带	$T^2(n)$		
M_2	2带	$T(n)\log T(n)$		

NP完全理论的基本概念

判定问题和语言

- 判定问题定义与描述
- 判定问题与语言的关系

P类与NP类

- 难解的问题与多项式可解的问题
- P类与NP类定义

多项式变换与NP完全性

- 多项式变换的定义及性质
- NP完全的定义

Cook定理

判定问题和语言

判定问题的定义

陈述判定问题的标准格式

引入判定问题的理由

判定问题的形式描述——语言

算法的形式描述——Turing机

判定问题的定义和描述

定义 一个判定问题 $\pi = (D\pi, Y\pi)$, 其中 $D\pi$ 为实例集, $Y\pi \subseteq D\pi$ 为肯定实例的集合. 任给实例 $I \in D\pi$, 问 $I \in Y\pi$?

陈述一个判定问题的标准格式

对实例中参数的一般描述和一个肯定--否定问题.

子图同构问题

实例: 两个图 $G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$

问: G_1 是否包含与 G_2 同构的子图? 即是否存在子集

$V' \subseteq V_1$, $E' \subseteq E_1$ 使得 $|V'|=|V_2|$, $|E'|=|E_2|$, 且有双射 f: $V_2 \to V'$ 满足以下条件?

$$\{u,v\} \in E_2 \Leftrightarrow \{ f(u), f(v) \} \in E'$$

引入判定问题的理由

- 判定问题的形式化描述简单.
- 许多优化问题的难度与判定问题的难度相关.
  判定问题 $\propto$ 优化问题

  将判定问题转换为优化问题. 时间为 $O(g(n))$
  调用解优化问题的算法求解优化问题. 时间为 $O(f(n))$
  用优化问题的解得到判定问题的解. 时间$O(h(n))$
  总时间：$O(g(n)+f(n)+h(n))$
  $g(n)$ 和 $h(n)$ 为 $o(f(n))$，判定问题时间为$O(f(n))$

实例：巡回售货员问题是优化问题. 如果在实例中加上参数B, 问是否存在长度不超过B的旅行? 得到对应的判定问题.

判定问题的形式描述

语言的定义

∑为有穷字符集

∑*是 ∑ 上所用有穷字符串的集合

∑*的任何子集为 ∑ 上的语言

判定问题与语言的关系

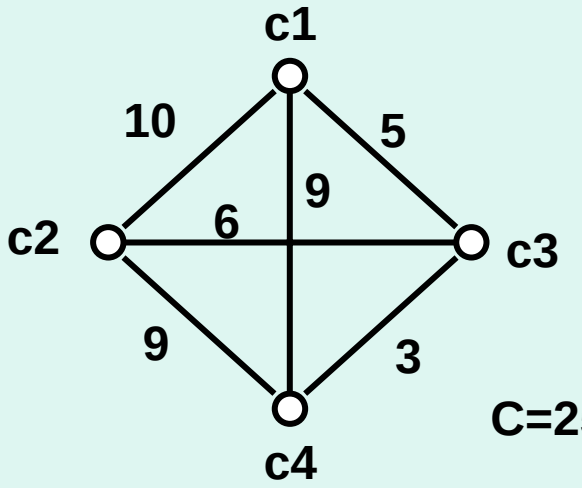
在合理的编码系统 **e**下, 判定问题的任意实例被编码成一个字符串 ***x***

巡回售货员问题的实例

相应的字符串是:

∑ = {c,[,],/,0,1,2,3,4,5,6,7,8,9,#}

c[1]c[2]c[3]c[4]//10/5/9//6/9//3#25



判定问题的形式描述（续）

编码系统 e 将 Σ^* 中的字符串划分成三类:

不是实例中的编码

$D\pi$－$Y\pi$中的实例的编码

$Y\pi$ 中的实例的编码

$Y\pi$ 中的实例的编码构成与判定问题 π 对应的语言 L.

具体定义如下:

与判定问题 π 和编码系统 e 相关的语言 $L[\pi,e]$

设 π 为判定问题, e 是关于π 的编码系统, 其字符表为 Σ,

$L[\pi,e]=\{ x\in\Sigma^*: x$是某个实例 $I\in Y\pi$ 在 e下的编码 $\}$

$I\in Y\pi \Leftrightarrow x\in L[\pi,e]$

算法的形式描述

算法解判定问题 π ⇔ 用 Turing 机识别语言 $L[\pi,e]$

说明：

对判定问题π, 相关的 $L[\pi,e]$ 与编码系统 e 有关.

合理的编码系统:

无冗余的符号和信息.

数字用二进制(或其他进制，不允许用一进制）表示.

可译码

在不同的合理的编码系统下, 实例的编码所对应的输入长度 length: $D\pi \rightarrow Z^+$ 多项式相关. 换句话说, e, e'为合理的编码系统, 实例 I 在e和e'下的编码所对应的输入长度分别为 length[I] 和 length'[I], 则存在多项式 P 和 P' 使得

$$\text{length}'[I] \le P(\text{length}[I]), \quad \text{length}[I] \le P'(\text{length}'[I])$$

P类与NP类

难解的问题

定义

实例

P类与NP类的定义

非形式定义（问题类）

形式定义（语言类）

P类与NP类的关系

$P \subseteq NP$

P=NP?

难解的问题

定义：不存在确定型多项式时间算法的问题

难解的原因

- 问题太难, 在多项式时间不可能找到解
- 解本身太庞大, 表示解的符号串长度不是输入长度的多项式函数

一般只考虑第一种难解的问题

两类难解的问题

不可判定问题——不存在算法

可判定的难解问题——有算法，但不存在多项式时间的算法

实例---不可判定问题

停机问题是不可判定的

定理1 不存在根据任意 Turing机 T 的定义, 能够确定 T 在输入 d_T 上是否停机的算法.

证明思路：证明 T 在输入 T 上停机是不可判定的

反证法：

假设存在 Turing机 H，可以对任何 Turing机 T，判定T在 T 上是否停机.

根据 H，构造 Turing机 L, 使得对于L产生如下悖论：

L 在 L 上停机 $\Rightarrow$ L 在 L 上不停机

L 在 L 上不停机 $\Rightarrow$ L 在 L 上停机

证 明

证 假定H是可判定T在T上是否停机的Turing机. H有两个停机状态:

Yes停机状态 $\Leftrightarrow$ T 在 T 上停机

No 停机状态 $\Leftrightarrow$ T 在 T 上不停机

构造新的Turing机 L. L 比 H 多两个新的状态 q_1', q_2'.

H 进入Yes停机状态当且仅当 L 转移到q_1'. 并且设定转移函数, $\forall u \in \Sigma$ 有

$$\delta(q_1',u) = (q_2',u,R), \quad \delta(q_2',u) = (q_1',u,L)$$

H 进入 No 停机状态, L 也进入 No 停机状态. 从而有

T 在 T 上停机 $\Rightarrow$ L 在 T 上不停机

T 在 T 上不停机 $\Rightarrow$ L 在 T 上停机

令 $T = L$, 则导致

L 在 L 停机 $\Rightarrow$ L 在 L 上不停机

L 在 L 不停机 $\Rightarrow$ L 在 L 上停机

产生矛盾.

实例---可判定的难解问题

非确定型的难解问题

(用非确定型的Turing机在多项式时间不可能解出)

判定半扩展正则表达式是否表示它的字母表上的所有的串.

The Design and Analysis of Computer Algorithms, Aho, Hopcroft, Ullman. 1972.

NP类问题

是判定问题

用非确定型的算法在多项式时间可解

到目前还没有找到多项式时间的确定型算法

是否为难解的问题（不清楚）

P类和NP类非形式定义

P类: 用确定型算法在多项式时间可解的判定问题类

NP类: 用非确定型算法在多项式时间可解的判定问题类

非确定型算法求解判定问题π

任意给定实例 $I \in D\pi$, 如果 $I \in Y\pi$, 则存在结构 s，使得当对输入I 猜想 s 时，检查阶段回答Yes; 如果 $I \notin Y\pi$, 则不存在结构 s，使得当对输入I 猜想 s 时，检查阶段回答 Yes.

多项式时间的非确定型算法

对于解判定问题非确定型算法, 如果存在多项式 P, 对每个实例 $I \in Y\pi$ 存在猜想 s，使得检查阶段在时间 $P(\text{length}[I])$ 内回答 Yes.

几点说明

1. 猜想的结构 s 的规模小于 $P(\text{length}[I])$

2. 对肯定实例 $I \in Y\pi$，至少存在一个猜想 s，使得检查阶段在时间 $P(\text{length}[I])$ 内回答 Yes, 而不管其他猜想.

3. 对否定实例 $I \in D\pi - Y\pi$ 的运算可以不管.

4. 可以看成具有无限并行能力的计算机. 将所有可能的结构猜想出来, 每个结构一道, 如果有一个回答Yes, 则整个机器停机并回答Yes; 使得机器回答Yes的最短时间就是计算时间.

5. 肯定与否定是不对称的

NP类问题肯定与否定不对称

问题 π 与 π 的补问题 π^c.

π: 对于给定的实例 I，X是否成立?

π^c: 对于给定的实例 I，X是否不成立?

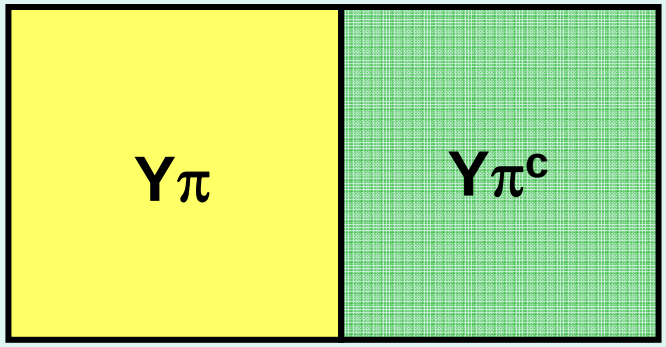
$D\pi = D\pi^c$,

$Y\pi = D\pi^c - Y\pi^c$,

$Y\pi^c = D\pi - Y\pi$.

$\pi \in P \Rightarrow \pi^c \in P$

$\pi \in NP \Rightarrow \pi^c \in NP$? 不一定



Dπ=Dπc

$\pi \in P \Leftrightarrow \pi^c \in P$

证 设 M 是解π的DTM程序, 则对任意 $I \in D\pi$, M 都停机. M 有两个停机状态Yes和No. 设 M' 完全模拟 M 的动作, 只是交换两个停机状态. 则 M'是解 π^c 的DTM程序, 因为

M'停机No $\Leftrightarrow$ M 停机Yes $\Leftrightarrow$ $I \in Y\pi \Leftrightarrow I \in D\pi^c - Y\pi^c$

M'停机Yes $\Leftrightarrow$ M 停机No $\Leftrightarrow$ $I \in D\pi - Y\pi \Leftrightarrow I \in Y\pi^c$

所以 $\pi \in P \Leftrightarrow \pi^c \in P$

$\pi \in NP$, 存在解π的非确定型算法. 要想得到解 π^c的非确定型算法必须对 $I \in Y\pi^c$的实例做猜想. 而 $I \in D\pi - Y\pi \Leftrightarrow I \in Y\pi^c$,对于$I$, 原来解$\pi$的非确定型算法可能根本不停机, 因此不能由这个解$\pi$的非确定型算法得到解$\pi^c$的非确定型算法. $\pi \in NP$ 推不出 $\pi^c \in NP$

P与NP类的形式定义

P = { L:存在多项式时间的DTM程序M使得 $L=L_M$ }

NP ={ L:存在多项式时间的NDTM程序M使得 $L=L_M$ }

非形式描述	形式描述
判定问题 π	语言 L
实例 $I \in D\pi$	符号串 x
肯定实例 $I \in Y\pi$	符号串 $x \in L_M$
实例规模	符号串长度
确定型算法	对所有输入停机的DTM程序
确定型算法对I 回答Yes	DTM程序对符号串x停机接受状态
确定型算法求解判定问题 π	DTM程序识别语言 L
多项式时间的确定型算法	多项式时间的DTM程序
$\pi \in P$(问题类)	$L \in P$ (语言类)

P与NP的关系

$P \subseteq NP$

$P = NP$?

可以证明 DTM 模拟 NDTM 的时间为指数时间

DTM 模拟 NDTM 的时间复杂性的上界

定理2 $P \subseteq NP$

证 任取 L 属于P, 令 M_1 是多项式时间的识别 L 的 DTM 程序. 如下构造识别 L 的 NDTM 程序 M_2.

对于给定的输入, 任意写下猜想串, 然后在检查阶段模拟 M_1 的动作即可. 易见 M_2 是识别 L 的 NDTM 程序, 故 L 属于 NP.

DTM 模拟 NDTM

定理3 如果 $L \in NP$, 那么存在多项式 P 使得 L 能用时间复杂性函数为 $O(2^{P(n)})$ 的 DTM 程序识别.

证 $L \in NP$, 存在识别 L 的NDTM程序 M, 即存在多项式q, 对于任意 $x \in L$, $|x|=n$, 都存在猜想串 s, $|s| \le q(n)$, M 对 x 和 s 的计算在 $q(n)$ 步停机 yes.

长度不超过 $q(n)$ 的猜想串, 至多有 $K^{q(n)}$个, $K=|\Gamma|$.

如下构造 DTM 程序M': 依次写下所有长度不超过$q(n)$的猜想串 s, 对输入 x 和 s 使用 M 的检查阶段的动作进行计算. 如果对某个猜想串 s, M 在 $q(n)$ 步内停机Yes, 则 M' 停机Yes; 如果 M 对所有的猜想串在 $q(n)$ 步或停机 No, 或不停机, 则M'停机No. 易见$L_{M'} = L_M$.

考虑 M'的时间复杂性. 对每个猜想串的计算时间至多为$q(n)$, 对所有猜想串的计算时间至多为

$$O(K^{q(n)}q(n)) = O(2^{P(n)})$$

多项式变换和NP完全性

多项式变换的定义
- 非形式的定义
- 变换实例
- 形式定义

多项式变换的性质

NP完全性

NP完全的有关结果

Cook定理
- 可满足性问题
- Cook定理

多项式变换的非形式定义

设判定问题 π_1, π_2, 若存在函数 $f: D\pi_1 \to D\pi_2$ 满足

I. f 可以用确定型算法在多项式时间计算

II. $\forall I \in D\pi_1,\ I \in Y\pi_1 \Leftrightarrow f(I) \in Y\pi_2$

则称 π_1 可多项式变换到 π_2, 记作 $\pi_1 \propto \pi_2$.

构造多项式变换的步骤：

针对原来问题的参数给出变换规则 $f: D\pi_1 \to D\pi_2$

证明 $I \in Y\pi_1 \Leftrightarrow f(I) \in Y\pi_2$

证明计算 f 的时间为多项式时间

变换实例

HC α TS

π_1: 哈密顿回路问题 HC

π_2: 巡回售货员问题 TS

HC 实例: 图$G=(V,E)$, $V=\{v_1,v_2,\ldots,v_m\}$

问： G 中是否包含一条哈密顿回路?

TS 实例：城市集合 $C=\{c_1, c_2, \ldots, c_m\}$,正整数 $d(c_i,c_j)$,

$1\le i<j\le m$，B为正整数

问：是否存在一条长度不超过 B 的巡回路线？

多项式变换 f 如下：

令城市集合 $C=V$. $B=m$.

$\forall v_i, v_j \in V\ (i\neq j)$, $d(v_i,v_j)=\begin{cases}1 & \{v_i,v_j\}\in E\\ 2 & \{v_i,v_j\}\notin E\end{cases}$

证 明

证明肯定实例当且仅当变换到肯定实例

假设 G 存在一条 HC $<v_1,v_2, \dots ,v_m>$, 则$<v_1,v_2, \dots ,v_m>$ 也是 $f(G)$ 中的旅行路线. 易见该旅行路线中的每条边长度都是 1, 总长为m. 所以 $f(G)$ 为TS的肯定实例.

假设$<v_1,v_2, \dots , v_m>$是 $f(G)$ 中长度不超过 m 的旅行. 由于任何城市间的距离为 1 或 2, 总共有m个距离, 因此每个距离必为 1. 从而证明了G是 HC 的肯定实例.

证明变换的时间为多项式时间

计算 $d(v_i, v_j)$, $i \neq j$. 共计算 $m(m-1)/2$次, 每次要检查 $\{v_i, v_j\}$ 是否属于E (可以用邻接表进行). 显然是多项式时间.

多项式变换的形式定义

设Σ_1, Σ_2为字母表, $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ 是语言, 如果 $f: \Sigma_1^* \to \Sigma_2^*$满足:

I. 存在计算 f 的多项式时间的 DTM 程序

II. $\forall x \in \Sigma_1^*,\ x \in L_1 \Leftrightarrow f(x) \in L_2$

则称 f 是从 L_1到 L_2 的多项式变换, 记作$L_1 \propto L_2$.

所谓多项式时间的 DTM 程序 M 计算函数 $f: \Sigma^* \to \Gamma^*$是指: 输入字符表为 Σ, 带字符表为 Γ. M 对 Σ^*上的所有长度为 n 的 字符串在 n 的多项式步内停机，且 f 对于任何 $x \in \Sigma^*$, $f(x)$ 是输入 x 到 M 停机时带方格 1 到最右边的非空白方格为止的符号串.

若$L_1 \propto L_2$, $L_2 \propto L_1$, 则称 L_1与 L_2是多项式等价的.

多项式变换的性质

（1）若 $L_1 \propto L_2$, 则 $L_2 \in P \Rightarrow L_1 \in P$.

（2）若 $L_1 \propto L_2$, $L_2 \propto L_3$, 则 $L_1 \propto L_3$.

证（1）设 Σ_1, Σ_2是L_1和 L_2的字符表, $f: \Sigma_1^* \to \Sigma_2^*$是从 L_1 到 L_2 的多项式变换, M_f 是计算 f 的多项式时间的 DTM 程序, M_2是识别 L_2 的多项式时间的 DTM 程序.

如下构造识别 L_1 的多项式时间的 DTM 程序 M':

M'首先模拟 M_f, 对 $x \in \Sigma_1^*$计算出 $f(x) \in \Sigma_2^*$. 接着模拟 M_2, 确定 $f(x)$是否属于L_2. M'识别 L_1 是由于

$x \in L_1 \Leftrightarrow f(x) \in L_2 \Leftrightarrow M_2$对 $f(x)$停机$q_Y \Leftrightarrow M'$对 x 停机 q_Y

设 M_f, M_2 的时间复杂性函数分别为多项式 P_f, P_2, 则 $|f(x)| \le P_f(|x|)$, 从而 M' 的时间复杂性为

$$O(P_f(|x|) + P_2(P_f(|x|)))$$

即 $L_1 \in P$.

NP完全性与有关结果

NP 完全性

若$L \in NP$, 且对于任意语言$L' \in NP$ 都有 $L' \propto L$, 则称 L是*NP*完全的, 记作$L \in NPC$, 其中 *NPC* 是*NP*完全的语言类.

类似可以定义 *NPC* 为 *NP* 完全的问题类.

有关 *NP* 完全的一些结果

（1）若 $L_1, L_2 \in NP$, $L_1 \in NPC$, 且$L_1 \propto L_2$, 则 $L_2 \in NPC$.

（2）若 $L \in NPC$, 则 $L \in P \Rightarrow P = NP$.

（3）若 $P \neq NP$, 则 $NPC \cap P = \varnothing$.

证明留作思考.

可满足性问题

几个概念

布尔变量的集合 $U:=\{u_1, u_2, \ldots, u_m\}$, u_i为布尔变量

U上的文字: 对于$u \in U$, u 或 $\bar{u}$ 称为U上的文字

U上的子句c: U上若干文字的集合，表示这些文字的析取.

例如 $\{u_1, u_3, u_5\}$

表示 $u_1 + u_3 + u_5$

U上的子句集C: U上的字句的集合，表示 C 中子句的合取.

例如 $C=\{\{u_1, u_2, \bar{u}_5\}, \{u_1, u_3, \bar{u}_5\}, \{u_5\}\}$

表示 $(u_1+u_2+\bar{u}_5)(u_1+u_3+\bar{u}_5)(u_5)$

可满足性问题（续）

U上的真值赋值： 函数 t: $U \to \{T,F\}$

$t(u) = F$, 称 u 是假值,

$t(u) = T$, 称 u 是真值.

易见 $t(u) = T \Leftrightarrow t(\bar{u}) = F$

t 满足子句 c: c中至少一个文字在这个赋值 t 下取真值

t 满足子句集C: t 满足集合 C 中的每个子句

可满足性问题 (SAT)

实例: 布尔变量集合 U 以及 U 上的子句集 C

问: 是否存在满足 C 的真值赋值?

Cook定理

Cook定理 (1971)

可满足性问题 SAT 是 *NP* 完全的.

证明思路

1. 设计一个解 **SAT** 问题的多项式时间的非确定型算法

2. 证明任何 ***NP*** 类中的语言 ***L*** 可以多项式变换到 **SAT.**

- 设 ***M*** 是接受 ***L*** 的多项式时间的 **NDTM**程序，将***M*** 在 ***x*** 上的计算变换到一个 **SAT** 的实例。
- 证明 ***M*** 在 ***x*** 上是接受计算当且仅当变换后是肯定实例
- 证明变换时间是多项式时间

证 明

1. SAT∈*NP*.

任意猜想一个关于U 中变量的真值赋值, 然后检查这个赋值是否满足C 中所有的子句。这可以在多项式时间完成.

2. 证明任何 *NP* 类中的语言 *L* 可以多项式变换到SAT.

设 M 是接受 L 的多项式时间的 NDTM程序，且整系数多项式 $P(n)$ 是时间复杂性函数 $T_M(n)$的上界. 考虑 M 在 x 上的计算. 如 M 接受 x, $|x|=n$, 则至多在$P(n)$步停机q_Y. 猜想串的长度不超过 $P(n)$，因此

带内容：$-P(n)$ 到 $P(n)+1$ 的带方格

状态：　至多有 $P(n)+1$个

如在 $P(n)$前停机, 则认为计算仍在进行, 只不过状态, 猜想串, 带头, 带符号全不变. 如在 $P(n)$ 时刻状态为 q_Y, 称 M 接受 x; 如状态为 q_N 或其它状态, 则称 M不接受 x.

证明（变量设计）

$M = (\Gamma, Q, \delta)$, 其中

$$Q = \{q_0, q_1 = q_Y, q_2 = q_N, ..., q_r\}$$

$$\Gamma = \{S_0 = B, S_1, S_2, ..., S_v\}$$

$$x = S_{k_1} S_{k_2} ... S_{k_n}$$

变量集合 U

$Q[i, k]$，$0 \le i \le P(n), 0 \le k \le r$,

$(P(n)+1)\ (r+1)$ 个

$Q[i, k]$=T $\Leftrightarrow$ 在时刻 i M 处于状态 q_k,

$H[i, j]$，$0 \le i \le P(n), -P(n) \le j \le P(n)+1$,

$(P(n)+1)\ (2P(n)+2)$ 个

$H[i, j]$=T $\Leftrightarrow$ 在时刻 i 读写头扫描方格 j

$S[i, j, l], 0 \le i \le P(n), -P(n) \le j \le P(n)+1, 0 \le l \le v$,

$(P(n)+1)\ (2P(n)+2)\ (v+1)$ 个

$S[i, j, l]$=T $\Leftrightarrow$ 在时刻 i 方格 j 的内容为 S_l

证明（子句设计）

任何 M 在 x 上的接受计算, 都对应于一组U上变量的赋值 t: $U \to \{F,T\}$.

任给一组 U 上变量的赋值, 不一定对应于 M 在 x 上的计算. 例如 $Q[1,1]=T$, $Q[1,2]=T$, 由于在时刻1 M 不可能同时处在状态1和状态2, 显然不能对应于M 在 x 的接受计算. 必须对赋值给予一定的限制, 使得它恰好对应于M 在 x 上的接受计算. 这些限制构成子句集C.

子句分成 6 种类型 $G_1, G_2, \ldots, G_6$, 即

$$C = G_1 \cup G_2 \cup \ldots \cup G_6$$

证明（子句设计续）

G_1: 在时刻 i, M必处于且只处于r 种状态之一

$$\{Q[i,0], Q[i,1], \ldots, Q[i,r]\} \qquad 0 \le i \le P(n)$$

$$\{\overline{Q[i,k]}, \overline{Q[i,k']}\} \qquad 0 \le i \le P(n), 0 \le k < k' \le r$$

G_2: 在时刻 i, 读写头扫描 $-P(n)$ 到 $P(n)+1$的带方格, 且只扫描一个方格

$$\{H[i,-P(n)], H[i,-P(n)+1], \ldots, H[i,P(n)+1]\} \qquad 0 \le i \le P(n)$$

$$\{\overline{H[i,j]}, \overline{H[i,j']}\} \qquad 0 \le i \le P(n), -P(n) \le j < j' \le P(n)+1$$

G_3: 在时刻 i, 带方格 j 含且只含字符集的一个字符

$$\{S[i,j,0], S[i,j,1], \ldots, S[i,j,v]\} \qquad 0 \le i \le P(n), -P(n) \le j \le P(n)+1$$

$$\{\overline{S[i,j,l]}, \overline{S[i,j,l']}\} \qquad 0 \le i \le P(n), -P(n) \le j \le P(n)+1, 0 \le l < l' \le v$$

证明（子句设计续）

G_4: 检查开始时刻0, 状态是 q_0,带头指在方格1, x放在方格1到方格n.

$\{ Q[0,0] \}$

$\{ H[0,1] \}$

$\{ S[0,0,0] \},\{ S[0,1,k_1] \}, \ldots ,\{ S[0,n,k_n] \},\{ S[0,n+1,0] \},$
$\ldots , \{ S[0,P(n)+1,0] \}$

G_5: 时刻 $P(n)$ M 停机 q_Y

$\{ Q[P(n),1] \}$

G_6: 在时刻 i, 方格 j 的符号为 S_l

如读写头没扫描方格 j, 则方格 j 在时刻 i+1的内容仍旧是S_l

$$\{ H[i,j], \overline{S[i,j,l]}, S[i+1,j,l] \}$$

$$0 \le i \le P(n), \quad -P(n) \le j \le P(n)+1, \quad 0 \le l \le v$$

证明（子句设计续）

如读写头扫描方格 j, j 的内容为 S_l，状态为 q_k，

由 M 的转移函数有

$$\delta(q_k, S_l) = (q_{k'}, S_{l'}, \Delta)$$

则在时刻 i+1, 读写头扫描方格 j+Δ, 状态为 $q_{k'}$，

方格 j 的内容变为$S_{l'}$

如果 $q_k = q_Y$ 或 q_N, 则 $k' = k, l' = l, \Delta = 0$.

$$\left.\begin{array}{l} \{\overline{Q[i,k]}, \overline{H[i,j]}, \overline{S[i,j,l]}, Q[i+1,k']\} \\ \{\overline{Q[i,k]}, \overline{H[i,j]}, \overline{S[i,j,l]}, H[i+1,j+\Delta]\} \\ \{\overline{Q[i,k]}, \overline{H[i,j]}, \overline{S[i,j,l]}, S[i+1,j,l']\} \end{array}\right\} \begin{array}{c} 0 \le i < P(n) \\ -P(n) \le j \le P(n)+1 \\ 0 \le l \le v \\ 0 \le k \le r \end{array}$$

证明（肯定实例）

给定U 上文字赋值, 对应于M接受 x 的计算. 所以

$x \in L$ ⇔ 存在一个M 在 x 上的接受计算

⇔ 检查阶段步数不超过 $P(n)$,

猜想串长度恰好等于 $P(n)$

的 M 在 x 上的接受计算

⇔ 有一个满足$f_L(x)$中子句集合的真值赋值

证明（变换时间）

对于任何 NP 类中的语言L, 根据 x, 能够在不超过 n 的多项式时间内构造出$f_L(x)$. 由于构造过程相当简单, 只须证明 $f_L(x)$ 的长度以 n 的多项式为上界即可. 设可满足性问题的长度函数是$|U||C|$,

变量个数:

$Q[i, k]$ $(P(n)+1)(r+1)$ 个

$H[i, j]$ $(P(n)+1)(2P(n)+2)$ 个

$S[i, j, l]$ $(P(n)+1)(2P(n)+2)(v+1)$ 个

子句个数:

$\|U\|$	$\|G_1\|$	$\|G_2\|$	$\|G_3\|$	$\|G_4\|$	$\|G_5\|$	$\|G_6\|$
$O(P^2(n))$	$O(P(n))$	$O(P^3(n))$	$O(P^2(n))$	$O(P(n))$	$O(1)$	$O(P^2(n))$

NP完全性证明

六个基本的NPC问题

三元可满足性问题 (3SAT)

三维匹配问题 (3DM)

顶点覆盖问题（VC）

团的问题

哈密顿回路 (HC)

均分问题

基本NP完全问题的证明

NP完全问题的证明方法

六个基本NP完全问题

3SAT

实例: 有穷布尔变量集U 和U上的子句集 $C = \{c_1, c_2, \ldots, c_m\}$, 其中$|c_i|=3$, $1 \le i \le m$.

问: 对于U 是否存在满足中所有子句的真值赋值？

3DM

实例: 集合$M \subseteq W \times X \times Y$, 其中$W,X,Y$互不相交, 且 $|W|=|X|=|Y|=q$.

问: M是否包含一个匹配, 即是否存在子集$M' \subseteq M$使得$|M'|= q$且M'中任意两个元素的三个坐标都不相同?

肯定实例：

$W=\{a_1,a_2,a_3,a_4\}$, $X=\{b_1,b_2,b_3,b_4\}$, $Y=\{c_1,c_2,c_3,c_4\}$

$M=\{ (a_1,b_2,c_1)$, (a_1,b_1,c_1), (a_2,b_1,c_2) , (a_2,b_2,c_1), (a_3,b_3,c_3), (a_4,b_4,c_4) , $(a_4,b_2,c_1)\}$

六个基本NP完全问题（续）

VC

实例: 图$G=(V,E)$, 正整数 $K \le |V|$.

问: G中是否存在大小不超过 K 的顶点覆盖, 即是否存在子集 $V' \subseteq V$,使得 $|V'|=K$, 且对每条边 $\{u,v\} \in E$ 都有 $u \in V'$或 $v \in V'$?

团

实例: 图$G=(V,E)$, 正整数 $J \le |V|$.

问: G是否包含大小不小于 J 的团, 即是否存在子集 $V' \subseteq V$,使得 $|V'| \ge J$, 且 V' 中每两个顶点都由 E 中的一条边连接?

（独立集）

实例：图$G=(V,E)$，正整数 $J \le |V|$

问：G 中是否包含大小不小于 J 的独立集，即是否存在$V' \subseteq V$, 使得$|V'| \ge J$，且$\forall\, u,v \in V'$，$\{u, v\} \notin E$?

六个基本NP完全问题（续）

HC

实例: 图 $G=(V,E)$, $|V|=n$.

问: G中是否包含一条哈密顿回路, 即是否有G的顶点排列 $< v_{j_1}, v_{j_2}, \ldots, v_{j_n} >$ 使得

$$\{v_{j_i}, v_{j_{i+1}}\} \in E, \quad 1 \le i < n, \quad \{v_{j_n}, v_{j_1}\} \in E ?$$

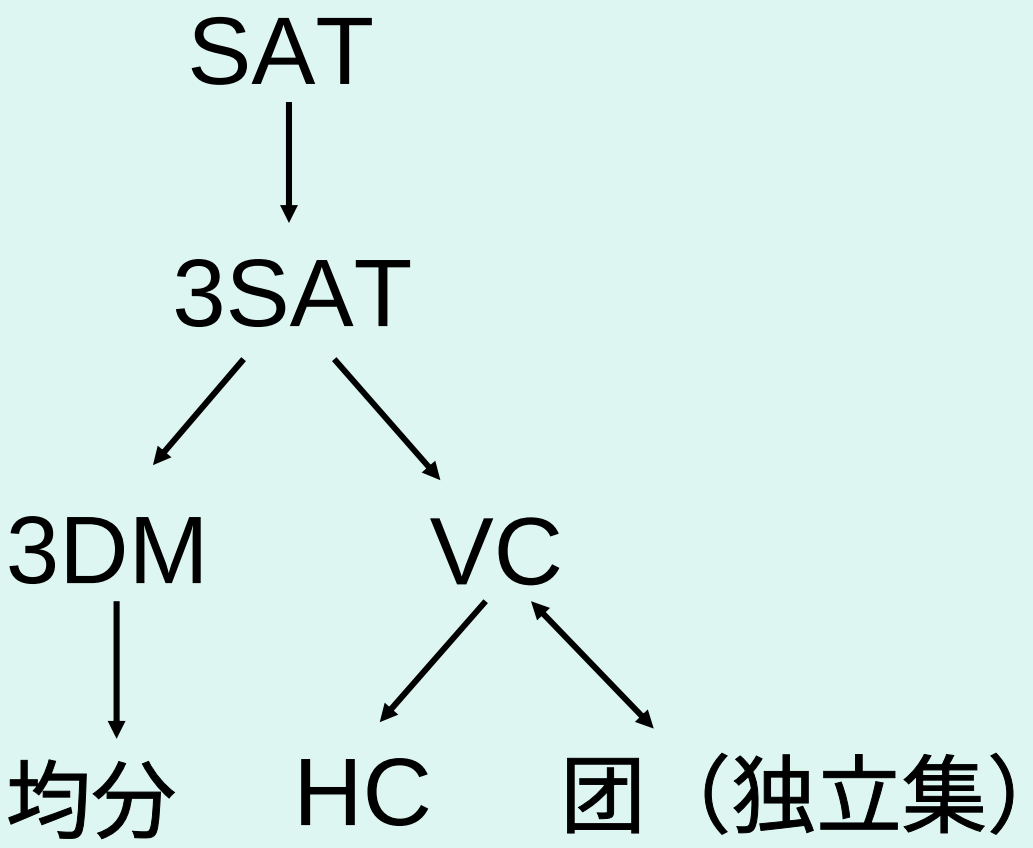
均分

实例: 有穷集合 A, $\forall a \in A$ 有 “大小” $S(a) \in Z^+$

问: 是否存在子集 $A' \subseteq A$ 使得

$$\sum_{a \in A'} S(a) = \sum_{a \in A - A'} S(a)$$

证明顺序



3SAT

1．3SAT∈NPC

证明思路

证 3SAT∈*NP*

将SAT的实例变换成 3SAT 的实例：

根据子句中的文字个数 k 分别处理

1个子句转变成多个子句

保证每个子句含有3个文字

转变前与转变后的真值不变

证 显然3SAT属于*NP*.

设 $U=\{u_1, u_2, \dots, u_n\}$, $C=\{c_1, c_2, \dots, c_m\}$是 SAT的任何实 例.

对于C中任意子句

$$c_j = \{ z_1, z_2, \dots, z_k \},$$

构造新增的变量集U_j’和子句集C_j’.

构造3SAT的实例

若 $k = 1$, $c_j = \{ z_1 \}$, 则

$U_j' = \{ y_j^1, y_j^2 \}$

$C_j' = \{ \{ z_1, y_j^1, y_j^2 \}, \{ z_1, y_j^1, \overline{y_j^2} \}, \{ z_1, \overline{y_j^1}, y_j^2 \}, \{ z_1, \overline{y_j^1}, \overline{y_j^2} \} \}$

C_j'被满足$\Leftrightarrow$ c_j 被满足.

若 $k = 2$, $c_j = \{ z_1, z_2 \}$, 则

$U_j' = \{ y_j^1 \}$

$C_j' = \{ \{ z_1, z_2, y_j^1 \}, \{ z_1, z_2, \overline{y_j^1} \} \}$

C_j'被满足$\Leftrightarrow$ $z_1 + z_2$ 为真 $\Leftrightarrow$ c_j 被满足.

若 $k = 3$, 则

$U_j' = \varnothing$

$C_j' = \{ c_j \}$

构造3SAT的实例（续）

若 $k>3$, 例如$c_j = \{ z_1, z_2, \dots , z_5 \}$, 则

$$U_j' = \{ y_j^1, y_j^2 \}$$

$$C_j' = \{\{ z_1, z_2, y_j^1 \}, \{ \overline{y_j^1}, z_3, y_j^2 \}, \{ \overline{y_j^2}, z_4, z_5 \}\}$$

c_j 可满足 $\Leftrightarrow$ $z_1+z_2+\dots+z_5$为真 $\Rightarrow$ C_j'可满足

如 z_1+z_2为真, 则 $y_j^1 = y_j^2 = \mathrm{F}$;

如 z_3为真, 则 $y_j^1 = \mathrm{T}$, $y_j^2 = \mathrm{F}$;

如 z_4+z_5为真, 则 $y_j^1 = \mathrm{T}$, $y_j^2 = \mathrm{T}$.

反之, 若C_j'可满足而c_j不满足, 则 $z_1+z_2+\dots+z_5$为假, 即 $z_1, z_2, \dots , z_5$全为假, 必须 $y_j^1=\mathrm{T}$, $y_j^2=\mathrm{T}$, 最后的子句不满足.

构造3SAT的实例（续）

一般若$c_j=\{z_1, z_2, \ldots, z_k\}$, $k>3$

$$U_j' = \{y_j^i : 1 \le i \le k-3\}$$

$$C_j' = \{\{z_1, z_2, y_j^1\}\} \cup$$

$$\{\{\overline{y_j^i}, z_{i+2}, y_j^{i+1}\} : 1 \le i \le k-4\} \cup$$

$$\{\{\overline{y_j^{k-3}}, z_{k-1}, z_k\}\}$$

令

$$U' = U \cup U_1' \cup U_2' \cup \ldots \cup U_m'$$

$$C' = C_1' \cup C_2' \cup \ldots \cup C_m'$$

肯定实例变到肯定实例

若 t 满足C', 证明 t 满足C. 假若 t 不满足c_j，则 $z_1, z_2, \ldots, z_k$ 全为假，只能有

$$y_j^{\,1} = y_j^{\,2} = \ldots = y_j^{\,k-3} = \mathrm{T}$$

C'的最后一个字句不满足.

反之，设 $t: U \rightarrow \{\mathrm{T},\mathrm{F}\}$是满足 C 的真值赋值, 将 $U'-U$ 的变量分成 U_1', U_2', ..., U_m', 且 U_j' 的变量只出现于子句 C_j'中。对于$k = 1,2,3$, t 已满足 C_j'的所有子句.

肯定实例变到肯定实例（续）

下面考虑 k >3情况.

k > 3, 设 l 表示使文字 z_l 在 t 下取真的最小正整数 l,

Case1：l = 1 或 2, z_1=T 或 z_2 = T, 则

t ’为:

$$y_j^1 = y_j^2 = \ldots = y_j^{k-3} = \mathrm{F}$$

t’ 满足C_j’的所有子句

$$C_j' = \{\{z_1, z_2, y_j^1\}, \{\overline{y_j^1}, z_3, y_j^2\}, \ldots,$$

$$\{\overline{y_j^{k-4}}, z_{k-2}, y_j^{k-3}\}, \{\overline{y_j^{k-3}}, z_{k-1}, z_k\}\}$$

肯定实例变到肯定实例（续）

$k > 3$, 设 l 表示使文字 z_l 在 t 下取真的最小正整数 l,

Case2: $l = k$ 或 k-1, $z_{k-1} = \mathrm{T}$ 或 $z_k = \mathrm{T}$, 则 t'为:

$$y_j^1 = y_j^2 = \ldots = y_j^{k-3} = \mathrm{T}$$

t' 满足C_j'的所有子句

$$C_j' = \{\{z_1, z_2, y_j^1\}, \{\overline{y_j^1}, z_3, y_j^2\}, \ldots,$$

$$\{\overline{y_j^{k-4}}, z_{k-2}, y_j^{k-3}\}, \{\overline{y_j^{k-3}}, z_{k-1}, z_k\}\}$$

肯定实例变到肯定实例（续）

下面考虑 $k>3$情况.

$k>3$, 设 l 表示使文字 z_l 在 t 下取真的最小正整数 l,

Case3: $3 \le l \le k-2$, z_l 为真，则 t ’为:

$$y_j^1 = \ldots = y_j^{l-2} = \mathrm{T},\ y_j^{l-1} = \ldots = y_j^{k-3} = \mathrm{F}$$

t’ 满足C_j’的所有子句

$$C_j' = \{\{z_1, z_2, y_j^1\}, \{\overline{y_j^1}, z_3, y_j^2\}, \ldots, \{\overline{y_j^{l-3}}, z_{l-1}, y_j^{l-2}\},$$
$$\{\overline{y_j^{l-2}}, z_l, y_j^{l-1}\}, \{\overline{y_j^{l-1}}, z_{l+1}, y_j^l\},$$
$$\ldots, \{\overline{y_j^{k-4}}, z_{k-2}, y_j^{k-3}\}, \{\overline{y_j^{k-3}}, z_{k-1}, z_k\}\}$$

变换的时间代价

k	1	2	3	>3
$\|U_j'\|$	2	1	0	$k-3$
$\|C_j'\|$	4	2	1	$k-2$

因为 $k \le 2n$, 所以

$|C_j'| \le k-2 \le 2n-2$

$|U_j'| \le k-3 \le 2n-3$

即

$|C'| \le O(P(mn))$, $|U'| \le O(P(mn))$

3SAT 的实例规模不超过 SAT 实例规模的多项式, 构造在多项式时间完成.

3DM

3DM∈NPC

实例: 集合 $M \subseteq W \times X \times Y$, 其中 W, X, Y互不相交,

且$|W|=|X|=|Y|=q$.

问: M 是否包含一个匹配, 即是否有子集 $M' \subseteq M$ 使得

$|M'|=q$ 且 M'中任意两个元素的三个坐标都不相同?

证 显然 3DM 属于NP. 下面将 3SAT 变换到 3DM.

设 $U = \{ u_1, u_2, \ldots, u_n \}$, $C = \{ c_1, c_2, \ldots, c_m \}$是 3SAT 的实例, 将 3DM 的三元组划分成三类:

真值安排和扇出, 满足性检验, 废料堆.

构造3DM的实例

真值安排和扇出 对于任意 $u_i \in U$, $i = 1, 2, \ldots, n$

$$T_i^t = \{(\bar{u}_i[j], a_i[j], b_i[j]) : 1 \le j \le m\}$$

$$T_i^f = \{(u_i[j], a_i[j+1], b_i[j]) : 1 \le j < m\} \cup \{(u_i[m], a_i[1], b_i[m])\}$$

$$T_i = T_i^t \cup T_i^f$$

$$a_i[j] \in X, \quad b_i[j] \in Y, \quad u_i[j], \bar{u}_i[j] \in W, \quad 1 \le j \le m$$

$$M' \cap T_i = T_i^t \Leftrightarrow t(u_i) = T$$

T_i^t 为灰色三角形, m个

T_i^f 为浅黄色三角形, m个

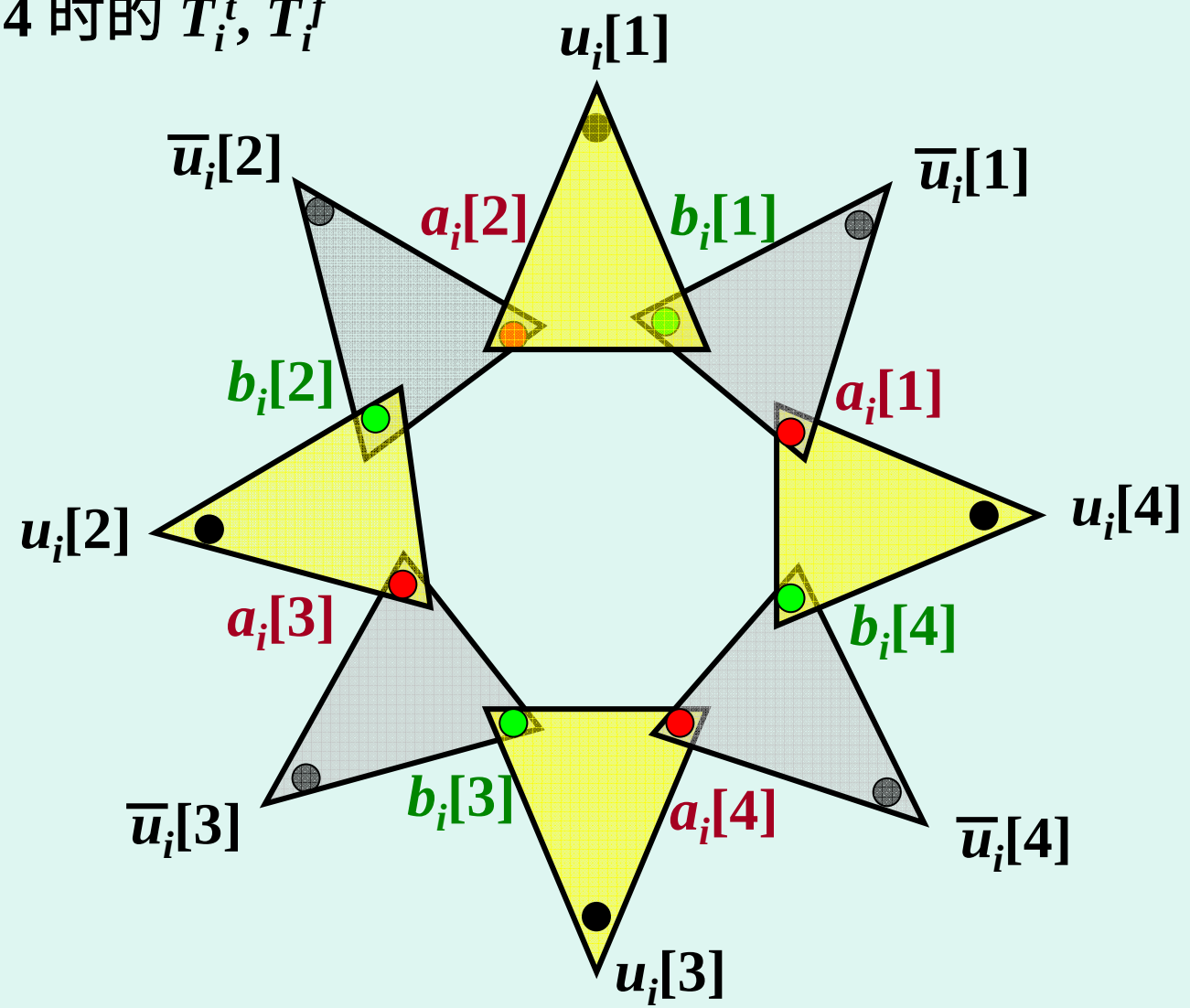
每个 $a_i[j]$, $b_i[j]$ 都在灰色和浅黄色两个三角形中

$u_i[j]$ 只存在于浅黄色三角形中

$\bar{u}_i[j]$ 只存在于灰色三角形中

构造3DM的实例（续）

$m = 4$ 时的 T_i^t, T_i^f



构造3DM的实例（续）

满足性检验分量

对于任何 $c_j \in C$，对应的三元组集合 C_j 定义为

$$C_j = \{(u_i[j], s_1[j], s_2[j]) : u_i \in c_j\}$$

$$\Upsilon \{(\bar{u}_i[j], s_1[j], s_2[j]) : \bar{u}_i \in c_j\}$$

其中 $s_1[j] \in X$, $s_2[j] \in Y$ 为内部元素.

例如 $c_j = \{u_1, \bar{u}_2, u_5\}$

$$C_j = \{(u_1[j], s_1[j], s_2[j]), (\bar{u}_2[j], s_1[j], s_2[j]), (u_5[j], s_1[j], s_2[j])\}$$

任何匹配 M'只能包含 C_j 中的一个三元组, 不妨设为 $(u_1[j], s_1[j], s_2[j])$. 则对应的真值安排和扇出分量只能取 $(\bar{u}_1[j], a_1[j], b_1[j]) \in T_1^t$, 从而 $t(u_1) = T$, c_j 被满足

构造3DM的实例（续）

废料堆

$$G=\{(u_i[j],g_1[k],g_2[k]),(\overline{u}_i[j],g_1[k],g_2[k]):$$
$$1\le i\le n,1\le j\le m,1\le k\le m(n-1)\}$$

令

$$W=\{u_i[j],\overline{u}_i[j]:1\le i\le n,1\le j\le m\}$$

$$X=\{a_i[j],s_1[j],g_1[k]:1\le i\le n,1\le j\le m,1\le k\le m(n-1)\}$$

$$Y=\{b_i[j],s_2[j],g_2[k]:1\le i\le n,1\le j\le m,1\le k\le m(n-1)\}$$

$$M=(T_1\cup T_2\cup\ldots\cup T_n)\cup(C_1\cup C_2\cup\ldots\cup C_m)\cup G$$

实例

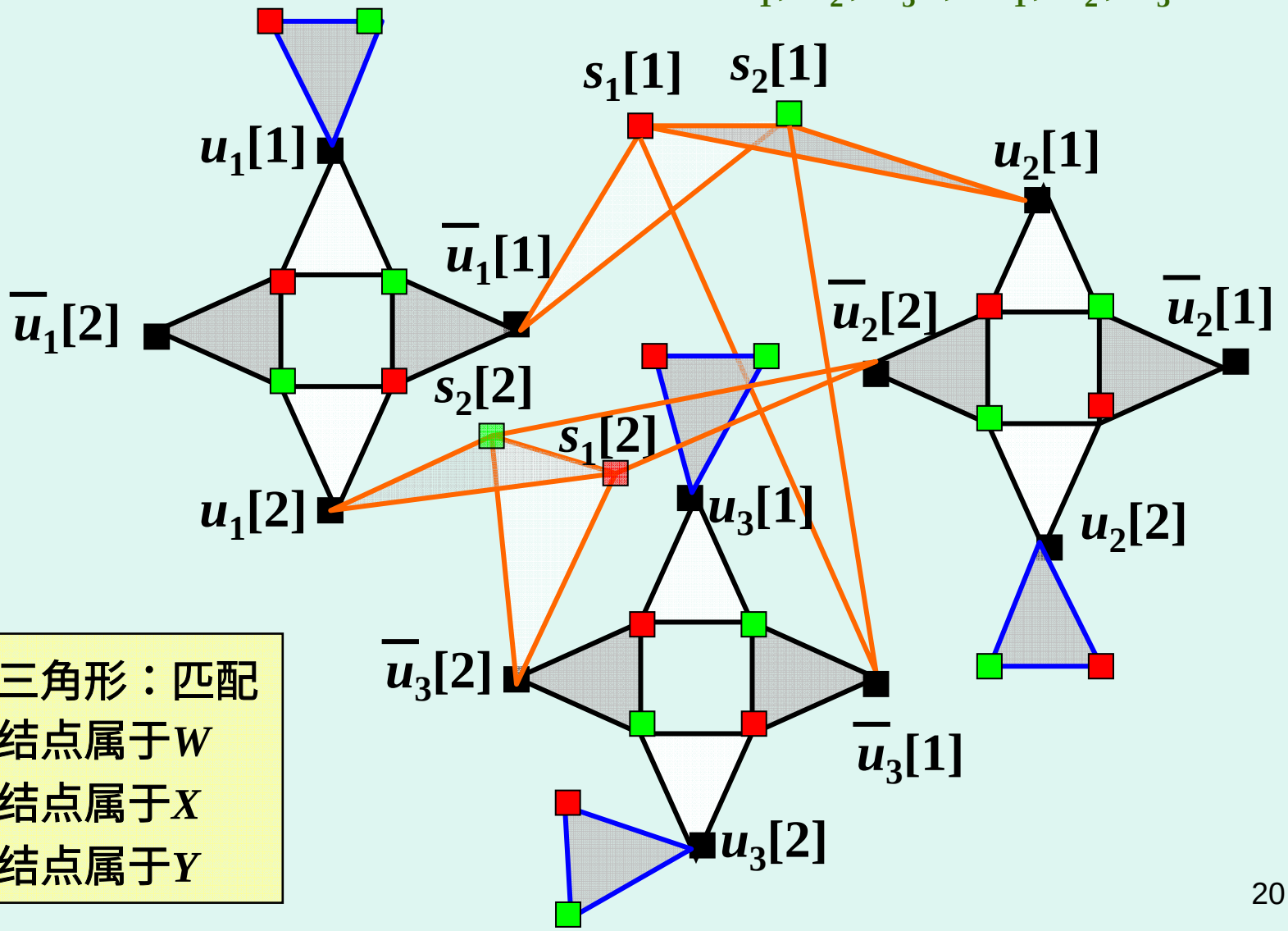
$$U=\{u_1,u_2,u_3\}$$

$$C=\{\{\overline{u}_1,u_2,\overline{u}_3\},\{u_1,\overline{u}_2,\overline{u}_3\}\}$$

变换实例

$U = \{u_1, u_2, u_3\}$

$C = \{\{\overline{u_1}, u_2, \overline{u_3}\}, \{u_1, \overline{u_2}, \overline{u_3}\}\}$



灰三角形：匹配
黑结点属于W
红结点属于X
绿结点属于Y

$M' \subseteq M$ 为匹配. M'必对每个 C_j 含有 C_j 中的一个三元组. 例如这个三元组为($u_i[j], s_1[j], s_2[j]$), 则

$$M' \text{含} \ (\bar{u}_i[j], a_i[j], b_i[j]) \Rightarrow M' \cap T_i = T_i^t$$

$$\Rightarrow t(u_i) = \mathrm{T} \Rightarrow c_j \text{被满足} \Rightarrow C \text{被满足}$$

反之,若 C 被满足, 存在满足C 的赋值 t: $U \to \{\mathrm{T},\mathrm{F}\}$. 对于任何 i, 若 $t(u_i)$为真, 取$T_i^t \subseteq M'$; 否则取 $T_i^f \subseteq M'$. 共取 mn个. 对于任何 j, t 满足 c_j, 存在 u_i 或 $\bar{u}_i$ 属于c_j, 在 t 下为真. 取($u_i[j], s_1[j], s_2[j]$) 或（$\bar{u}_i[j], s_1[j], s_2[j]$) 属于 M'. 共取m个. 在 G 中取剩下的 $m(n-1)$个，总共 $2mn$个. 易见M'是匹配.

变换规模

$$|W| = 2mn,$$

$$|X| = |Y| = mn + m + m(n-1) = 2mn,$$

$$|M| = 2mn + 3m + 2mn\ m(n-1)$$

从而证明变换可在多项式时间完成.

推　论

推论　X3C∈NPC.

X3C 代表三元集合构成的恰当覆盖.

实例: 有穷集 Z, $|Z|=3q$, Z的3元子集的集合 C.

问: C 是否存在一个 X 的恰当覆盖, 即是否存在 $C'\subseteq C$, 使得 X 的每个元素恰好出现在C'的一个成员中?

证 易见 X3C 属于 NP. 任给 3DM 的实例, W, X, Y, M.

如下构造 X3C 的实例. 令

$$Z = W \cup X \cup Y$$

$$C = \{ \{w, x, y\} : (w, x, y) \in M \}$$

三维匹配是三元恰当覆盖的限制形式.

3SAT 到 V C 的变换

VC∈*NPC*.

证 显然 VC∈*NP*. 将3SAT变换到VC.

设$U=\{u_1, u_2, \dots, u_n\}$, $C=\{c_1, c_2, \dots, c_m\}$是 3SAT 的实例.

如下构造图G, 分量设计法.

真值安排分量:

$T_i=(V_i, E_i)$, $1 \le i \le n$, 其中 $V_i=\{u_i, \bar{u}_i\}$, $E_i=\{\{u_i, \bar{u}_i\}\}$

任意覆盖必至少包含 u_i 或 $\bar{u}_i$ 中的一个, 否则不能覆盖边 $\{u_i, \bar{u}_i\}$.

满足性检验分量:

$S_j=(V_j', E_j')$, $1 \le j \le m$, 其中

$V_j'=\{a_1[j], a_2[j], a_3[j]\}$

$E_j'=\{\{a_1[j], a_2[j]\}, \{a_1[j], a_3[j]\}, \{a_2[j], a_3[j]\}\}$

覆盖至少包含V_j'中的两个顶点, 否则不能覆盖E_j' 三角形

3SAT 到 V C 的变换（续）

联络边:

沟通分量之间的关系

对于每个子句 c_j, 设$c_j = \{ x_j , y_j , z_j \}$, 则

$E_j''=\{\{a_1[j], x_j \}, \{a_2[j], y_j\},\{a_3[j], z_j\}\}$

$G = (V,E)$

$V = (V_1 \cup V_2 \cup ... \cup V_n) \cup (V_1' \cup V_2' \cup ... \cup V_m')$

$E = (E_1 \cup E_2 \cup ... \cup E_n) \cup (E_1' \cup E_2' \cup ... \cup E_m')$

$\cup (E_1'' \cup E_2'' \cup ... \cup E_m'')$

$K = n + 2m$

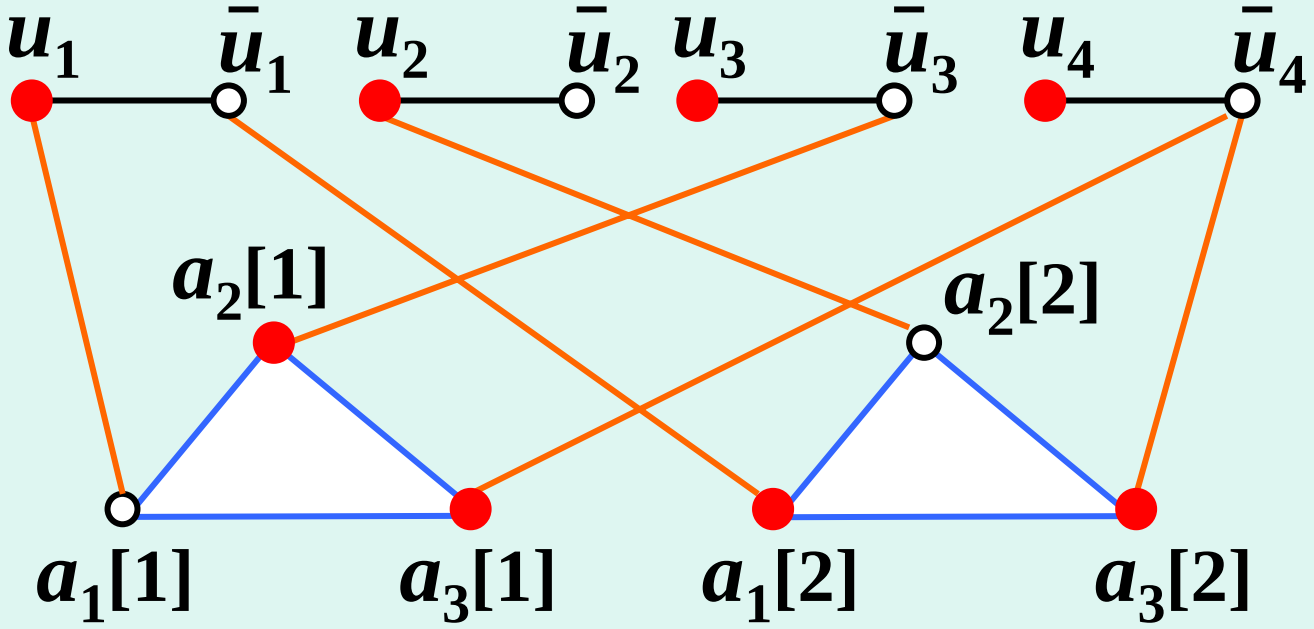
显然构造可在多项式时间完成

变换实例

$U = \{u_1, u_2, u_3, u_4\}$,

$C = \{ \{u_1, \bar{u}_3, \bar{u}_4\}, \{\bar{u}_1, u_2, \bar{u}_4\} \}$,

$K = 4 + 2 \times 2 = 8$



肯定实例变换到肯定实例

设V'是V 中不超过 K 的顶点覆盖, 则 V'中必包含T_i 中的一个顶点和每个E_j'中的两个顶点, 至少要 $n+2m$ 个顶点. 而 $K=n+2m$, 故 V'中一定只包含每个T_i 中的一个顶点和每个 E_j' 中的两个顶点.

如下得到赋值

$$u_i \in V' \Leftrightarrow t(u_i)=T, \quad \bar{u_i} \in V' \Leftrightarrow t(u_i)=F$$

E_j''中的三条边有两条被$V' \cap V_j'$ 中的顶点覆盖, 第三条必被$V' \cap V_i$中的顶点覆盖. 这表示在V_i 中的这个顶点对应的文字取真. 所以子句 c_j 被满足. 从而 C 被满足.

设 $t: U \rightarrow \{T,F\}$是满足 C 的一组赋值. 若 $t(u_i)=T$, 则在T_i 中取顶点 u_i, 否则取 $\bar{u_i}$. 因为 t 满足子句 c_j, 在 E_j''中的三条联络边中至少有一条被覆盖, 那么取 E_j''中的另两条边的端点作为 V' 中的端点即可.

顶点覆盖问题
与团、独立集问题的关系

设 $G = (V, E)$为图，$V' \subseteq V$, 则

V'是 G 的顶点覆盖

$\Leftrightarrow$ $V - V'$是G 的独立集

$\Leftrightarrow$ $V - V'$是$\overline{G}$ 的团

G 含有大小不超过 K 的顶点覆盖

$\Leftrightarrow$ G 中含有大小不小于$|V| - K$ 的独立集

$\Leftrightarrow$ $\overline{G}$ 中含有大小不小于$|V| - K$ 的团

HC

HC∈*NPC*.

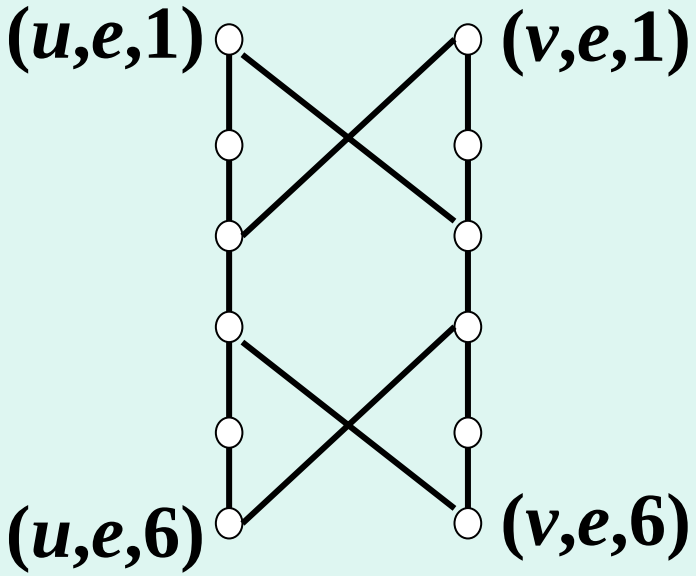
证 显然 HC 属于*NP*.

将 VC 变换到 HC.

设 $G=(V,E)$, K 是VC的实例.

对于任意边 $e=\{u,v\}$,

设计 覆盖检验分量



$V_e'=\{(u,e,i),(v,e,i): 1\le i\le 6\}$

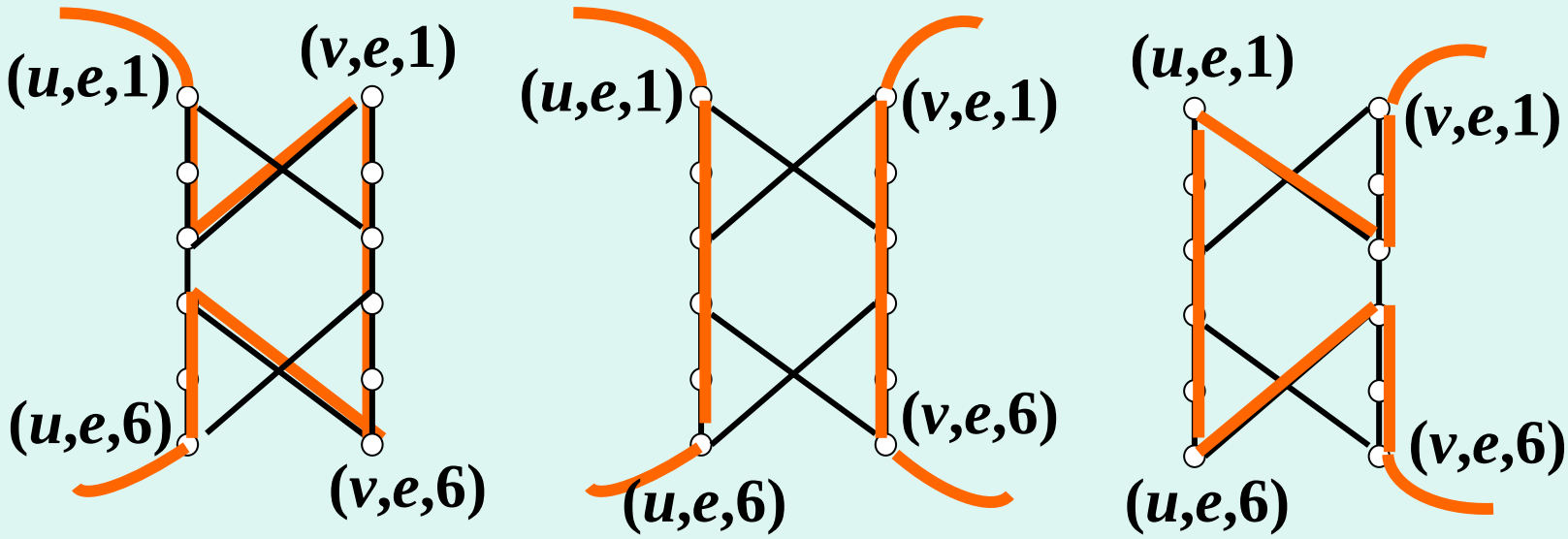
$E_e'=\{\{(u,e,i),(u,e,i+1)\},\{(v,e,i),(v,e,i+1)\}: 1\le i\le 5\}$

$\cup\{\{(u,e,3),(v,e,1)\},\{(v,e,3),(u,e,1)\}\}$

$\cup\{\{(u,e,6),(v,e,4)\},\{(v,e,6),(u,e,4)\}\}$

HC通过覆盖检验分量的方式

在*G'*中的任何 HC, 通过以上分量只有下面的三种方法：

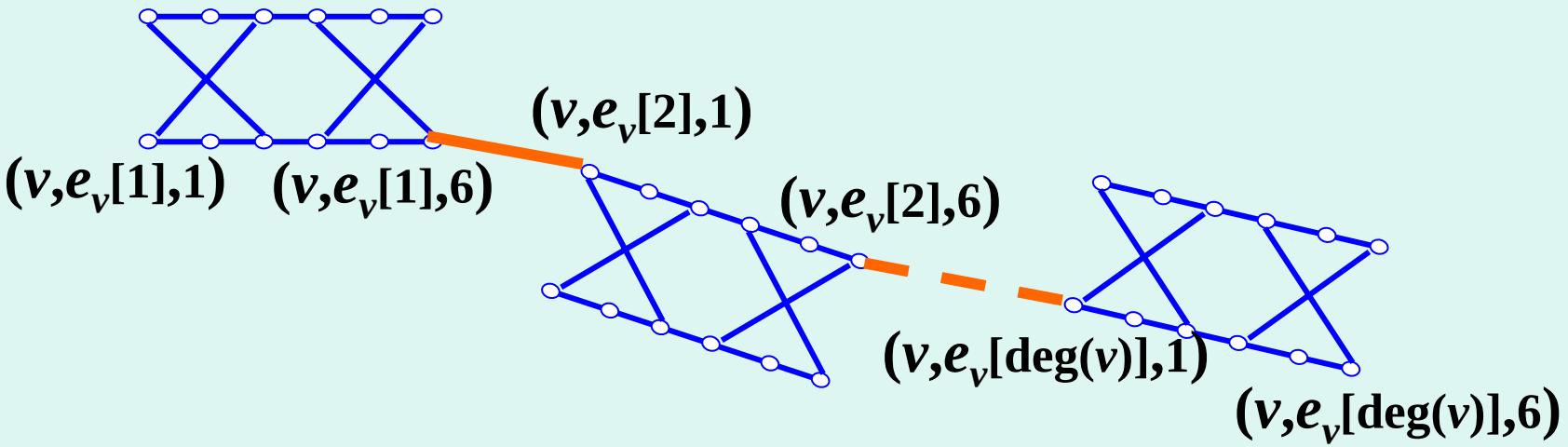


（1）*u*属于覆盖　（2）*u,v*都属于覆盖　（3）*v*属于覆盖

覆盖检验分量连接成路径

对应每个顶点 v，v 所关联的边为 $e_v[1], e_v[2], \ldots, e_v[\deg(v)]$, 将这些边对应的覆盖检验分量用下述边连接在一起：

$$E_v' = \{\{ (v,e_v[i],6), (v,e_v[i+1],1)\} \mid 1 \le i < \deg(v) \}$$



路径之间的连接

选择器 顶点 $a_1, a_2, \ldots, a_K$. 把上述路径的每个端点与选择器顶点相连，即

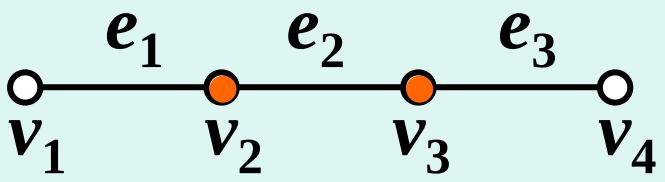
$$E''=\{\{ a_i, (v,e_v[1],1) \},\{ a_i, (v,e_v[\deg(v)],6) \} \mid 1\le i \le K, v\in V \}$$

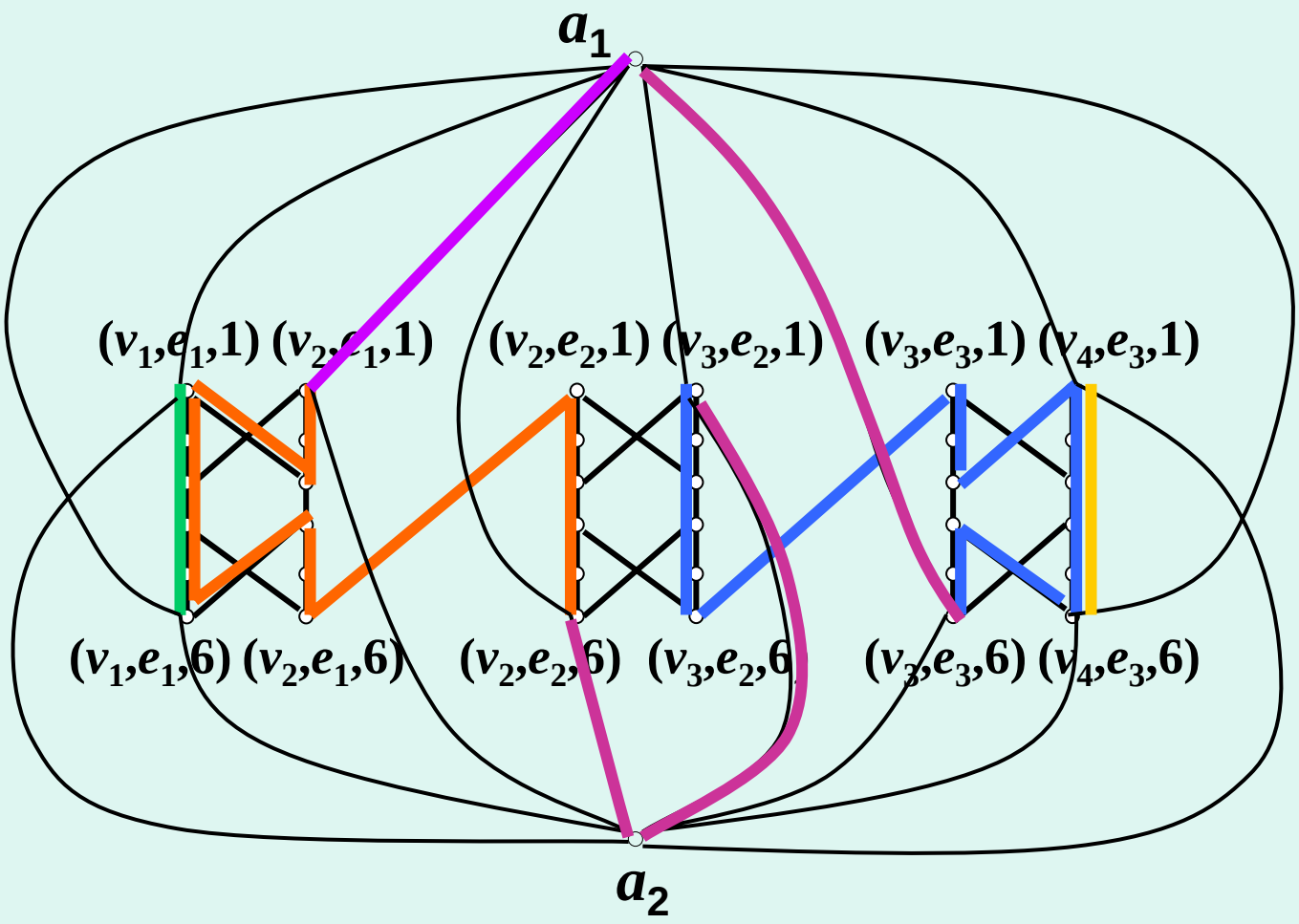
图$G'= (V',E')$, 其中

$$V' = \{a_i \mid 1 \le i \le K\} \cup \left(\bigcup_{e\in E} V_e' \right)$$

$$E' = \left(\bigcup_{e\in E} E_e' \right) \cup \left(\bigcup_{v\in V} E_v' \right) \cup E''$$

实例


e_1
e_2
e_3
v_1
v_2
v_3
v_4


a_1
$(v_1,e_1,1)$ $(v_2,e_1,1)$
$(v_2,e_2,1)$ $(v_3,e_2,1)$
$(v_3,e_3,1)$ $(v_4,e_3,1)$
$(v_1,e_1,6)$ $(v_2,e_1,6)$
$(v_2,e_2,6)$ $(v_3,e_2,6)$
$(v_3,e_3,6)$ $(v_4,e_3,6)$
a_2

肯定实例变换成肯定实例

下面证明G 有大小不超过K 的顶点覆盖当且仅当G'有Hamilton回路. 设V_1是G 的顶点覆盖, $|V_1| \le K$, 不妨设$|V_1| = \{ v_1, v_2, \ldots, v_K \}$. 如下选择$G'$中的边，对于边 $e = \{u,v\}$. 根据覆盖中的顶点是 u，u和v，v 三种情况选择覆盖分量的通过方法(1),(2)和(3). 然后选择 $E_{v'}$中的全体边，从而将覆盖检验分量连成 K 个路段，每一段对应V_1中的一个顶点. 最后，选择边

$$\{a_i, (v_i, e_{v_i}[1], 1)\}, \quad 1 \le i \le K$$

$$\{a_{i+1}, (v_i, e_{v_i}[\deg(v_i)], 6)\}, \quad 1 \le i < K$$

$$\{a_1, (v_K, e_{v_K}[\deg(v_K)], 6)\}$$

把 K个选择器插入这 K 段路间，连成Hamilton回路。

肯定实例变换成肯定实例（续）

反之，设 C 是 G' 的 Hamilton 回路. K 个选择器将回路分成 K 段，每段从一个选择器到另一个选择器. 根据构造规则，这段路对应某一顶点关联的所有边, K个选择器选出 K 个顶点.

任取 G 中的边 e，存在某个 C 中的路段经过 e 所在的覆盖检验分量. 这个路段对应覆盖集某个顶点 v 关联的所有的边. 那么 v 覆盖边 e. 因此，K 个选择器选出的 K 个顶点构成了顶点覆盖.

多项式时间构造 G'.

$$|V'| = 12m + K$$

$$|E'| \le 14m + n(D-1) + 2Kn$$

均分

5. 均分∈NPC

实例：有穷集 A，$\forall a \in A, s(a) \in Z^+$.

问：是否存在 $A' \subseteq A$，使得

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$$

证：显然均分是*NP* 类问题. 下面将 3DM 变换到均分

设$W, X, Y, M \subseteq W \times X \times Y$ 是3DM的实例，

其中 $|W| = |X| = |Y| = q$，

$W = \{ w_1, w_2, \ldots, w_q \}$

$X = \{ x_1, x_2, \ldots, x_q \}$

$Y = \{ y_1, y_2, \ldots, y_q \}$

$M = \{ m_1, m_2, \ldots, m_k \}$

变换设计

构造 A，$|A| = k+2$

对应于每个 $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$ 有a_i，i=1,2,…,k

$s(a_i)$为二进制数，分成 $3q$ 段，每段有 $p = \lceil \log(k+1) \rceil$ 位, 共计 $3pq$ 位，每段对应一个$W \cup X \cup Y$ 中的元素.

$w_{f(i)}, x_{g(i)}, y_{h(i)}$ 所代表的段的最右位为 1，其它为 0.

$$s(a_i) = 2^{p(3q-f(i))} + 2^{p(2q-g(i))} + 2^{p(q-h(i))}$$

注：$p \geq \log(k+1)$，$2^p \geq k+1$，$k \leq 2^p-1$,

当 k个 1 相加时不会产生段之间的进位

变换实例

$W=\{w_1, w_2\}$, $X=\{x_1, x_2\}$, $Y=\{y_1, y_2\}$, $q=2$

$M=\{(w_1, x_2, y_2), (w_1, x_2, y_1), (w_2, x_1, y_1)\}$ $k=3$

$p=\lceil \log(3+1) \rceil=2$

元素 a_1, a_2, a_3 分别对应

$(w_1, x_2, y_2), (w_1, x_2, y_1), (w_2, x_1, y_1)$

$s(a_1) = 01\ 00\ 00\ 01\ 00\ 01 = 2^{10} + 2^4 + 2^0$

$s(a_2) = 01\ 00\ 00\ 01\ 01\ 00 = 2^{10} + 2^4 + 2^2$

$s(a_3) = 00\ 01\ 01\ 00\ 01\ 00 = 2^8 + 2^6 + 2^2$

$B = 01\ 01\ 01\ 01\ 01\ 01$

变换设计 (续)

$$B = \sum_{j=0}^{3q-1} 2^{pj} = 2^{p(3q-1)} + 2^{p(3q-2)} + \ldots + 2^{2p} + 2^{p} + 2^{0}$$

B的段数与 $s(a_i)$ 一样，每段的最右位为 1，其它为 0

子集 $A' = \{ a_i : 1 \le i \le k \}$ 满足

$$\sum_{a \in A'} s(a) = B$$

当且仅当 $M' = \{ m_i : a_i \in A' \}$ 是 M 的匹配

A 的最后两个元素 b_1, b_2

$$s(b_1) = 2\sum_{i=1}^{k} s(a_i) - B, \quad s(b_2) = \sum_{i=1}^{k} s(a_i) + B$$

肯定实例变到肯定实例

$$\sum_{a\in A} s(a) = \sum_{i=1}^{k} s(a_i) + s(b_1) + s(b_2)$$

$$= \sum_{i=1}^{k} s(a_i) + 2\sum_{i=1}^{k} s(a_i) - B + \sum_{i=1}^{k} s(a_i) + B$$

$$= 4\sum_{i=1}^{k} s(a_i)$$

假设 $A' \subseteq A$使得 A'和 $A\text{-}A'$的元素大小之和相等，即

$$\sum_{a\in A'} s(a) = \sum_{a\in A-A'} s(a) = 2\sum_{i=1}^{k} s(a_i)$$

由于 $s(b_1) + s(b_2) = 3\sum_{i=1}^{k} s(a_i)$

b_1, b_2不在同一子集，

肯定实例变到肯定实例（续）

考虑包含 b_1的子集 A', 必有

$$\sum_{a\in A'-\{b_1\}} s(a) = 2\sum_{i=1}^{k} s(a_i) - (2\sum_{i=1}^{k} s(a_i) - B) = B$$

因此 A'-$\{b_1\}$的元素对应的三元组构成 M 的匹配

反之，若子集 M' 构成 M 的匹配，则

$$A' = \{b_1\} \cup \{a_i : m_i \in M'\}$$

满足

$$\sum_{a\in A'} s(a) = (2\sum_{i=1}^{k} s(a_i) - B) + B$$

$$= 2\sum_{i=1}^{k} s(a_i) = \sum_{a\in A-A'} s(a)$$

易见变换时间是多项式时间

NP完全性的证明方法

限制法

- 三元集合的恰当覆盖(X3C)
- 最小覆盖问题
- 击中集
- 子图同构问题
- 有界度的生成树
- 0-1背包 Knapsack
- 多处理机调度

NP完全性的证明方法（续）

局部替换法

3SAT

两点间的 **Hamilton** 通路问题

整体计算

三角划分

区间排序

分量设计法

最小拖延排序

限制法

限制法：通过对问题 π 的实例加以限制得到一个已知 *NP* 完全问题的实例.

例1 三元集合的恰当覆盖(X3C)

实例：有穷集 S, $|S|=3q$, S 的三元子集的集合 C

问： 是否有$C' \subseteq C$, 使得 S 的每个元素恰好出现在 C'的一个成员中?

证明：限制

$S = W \cup X \cup Y$

$|W|=|X|=|Y|=q$

$C = \{ \{w, x, y\} \mid (w, x, y) \in W \times X \times Y \}$

则 $|C'|=q$, 且 C'中任两个元素都不交，成为 3DM问题

限制法实例

例2 最小覆盖问题

实例：集合 S 的子集的集合 C，正整数 K.

问： C是否有 S 的大小不超过 K 的覆盖，即是否包含子集 $C' \subseteq C$ 使得$|C'| = K$ 且$\cup C' = S$？

证明：限制 $\forall c \in C, |c|=3, |S|=3K$, 则为 X3C 问题.

例3 击中集

实例：集合 S 的子集的集合 C，正整数 K

问： S 是否包含 C 的大小不超过 K 的集中集（hitting set）,即是否有子集 $S' \subseteq S$, 使得 $|S'| \leq K$, 且 S' 至 少包含 C 的每个子集的一个元素？

证明： 限制 $\forall c \in C, |c|=2$, 令$V=S, E=C$, 则构成图$G = (V,E)$的顶点覆盖问题.

限制法实例

例4 子图同构问题

实例：图 $G=(V_1, E_1)$, $H=(V_2, E_2)$

问：G中是否有同构于H的子图，即是否有子集$V\subseteq V_1$, $E\subseteq E_1$, 使得$|V|=|V_2|$,$|E|=|E_2|$,且存在双射函数 $f:V_2\to V$,使得

$$\{u, v\}\in E_2 \Leftrightarrow \{f(u), f(v)\}\in E?$$

证明：限制 H 为完全图，且 $|V_2|=J$, 则是团的问题.

例5 有界度的生成树

实例：图$G=(V,E)$, 正整数 $K\leq|V|-1$

问： G是否包含一棵顶点度数不超过 K 的生成树，即是否有子集$E'\subseteq E$, $|E'|=|V|-1$, 图$G'=(V, E')$是连通的，且V中每个顶点至多包含在 E'的 K 条边中？

证明：限制 $K=2$，则为 Hamilton 通路问题

限制法实例

例6 0-1背包 Knapsack

实例：有穷集 U，$\forall u \in U$, 大小 $s(u) \in Z^+$, 价值 $v(u) \in Z^+$, 大小的约束 $B \in Z^+$, 价值目标 $K \in Z^+$.

问：是否有子集 $U' \subseteq U$，使得

$$\sum_{u \in U'} s(u) \le B, \quad \sum_{u \in U'} v(u) \ge K$$

证明：限制$\forall u \in U$,

$$s(u) = v(u)$$

$$B = \left\lfloor \frac{1}{2} \sum_{u \in U} s(u) \right\rfloor, \quad K = \left\lceil \frac{1}{2} \sum_{u \in U} v(u) \right\rceil$$

则成为均分问题

限制法实例

例7 多处理机调度

实例：有穷任务集 A，$\forall a \in A$, 长度 $l(a) \in Z^+$, 处理机台数 $m \in Z^+$, 截止时间 $D \in Z^+$.

问： 是否存在不交的集合 $A_1, A_2, \ldots, A_m$, 使得

$$A = A_1 \cup A_2 \cup \ldots \cup A_m$$

$$\max\{\sum_{a \in A_i} l(a) : 1 \le i \le m\} \le D$$

证明：限制

$$m = 2$$

$$D = \left\lfloor \frac{1}{2} \sum_{a \in A} l(a) \right\rfloor$$

则成为均分问题.

局部替换法

局部替换法：选择已知 *NP* 完全问题的实例中的某些元素作为基本单元，然后把每个基本单元替换成指定结构，从而得到目标问题的对应实例.

例8 3SAT

已知问题：SAT ⇒ 目标问题：3SAT

基本单元：子句 ⇒ 子句集

局部替换法实例

例9 两点间的 Hamilton 通路

实例：$G=(V,E)$，$u, v \in V$.

问：G 中是否存在一条从 u 到 v 的 Hamilton 通路？

已知问题：HC $\Rightarrow$ 目标问题：两点间Hamilton通路

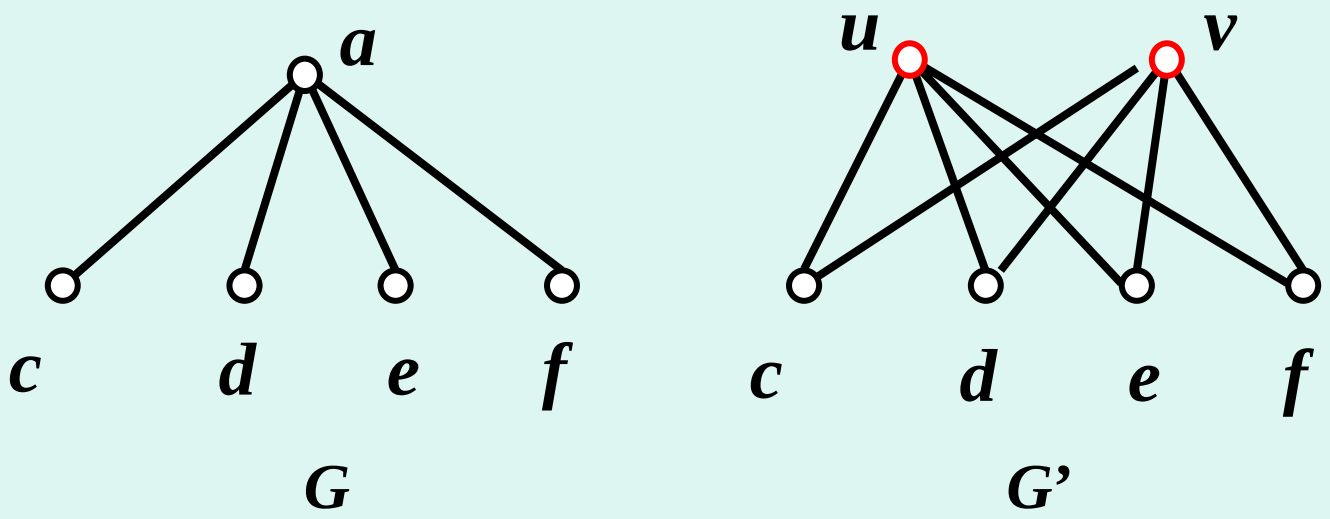
基本单元：顶点 a $\Rightarrow u, v$

证：对于 HC的任一实例，任选顶点 a, 用 u,v 代替a, 即

$$G' = (V',E'), \quad V' = (V-\{a\}) \cup \{u, v\}$$

$$E' = (E-\{\{a,v'\} \mid \{a,v'\} \in E\}) \cup \{\{v,v'\}, \{u,v'\} \mid \{a,v'\} \in E\}$$

替换实例



G 有一条 Hamilton 回路当且仅当

G’有一条从 u 到v 的 Hamilton通路

局部替换法实例

例10　整体计算

实例：有穷集 A 的子集的集合 C，正整数 J

问：是否存在 j ($j \le J$) 个并运算序列

$$< z_1=x_1 \cup y_1,\ z_2=x_2 \cup y_2,\ \dots,\ z_j=x_j \cup y_j >,$$

满足

x_i 和 y_i （i =1, 2, … , j）或是{ a }, $a \in A$, 或是 z_k ($k<i$)

对于 $1 \le i \le j$，$x_i \cap y_i = \varnothing$

对于每个 $c \in C$, 存在某个 z_i ($1 \le i \le j$) 等于 c ?

实例

A = { a, b, c, d }, J=3, C={{ a, b, c },{ b, c, d }}

$< z_1 = \{ b \} \cup \{ c \},\ z_2 = z_1 \cup \{ a \},\ z_3 = z_1 \cup \{ d \} >$

替　换

已知问题：**VC** $\Rightarrow$ 目标问题：整体计算

基本单元：边 $\Rightarrow$ C 中的子集

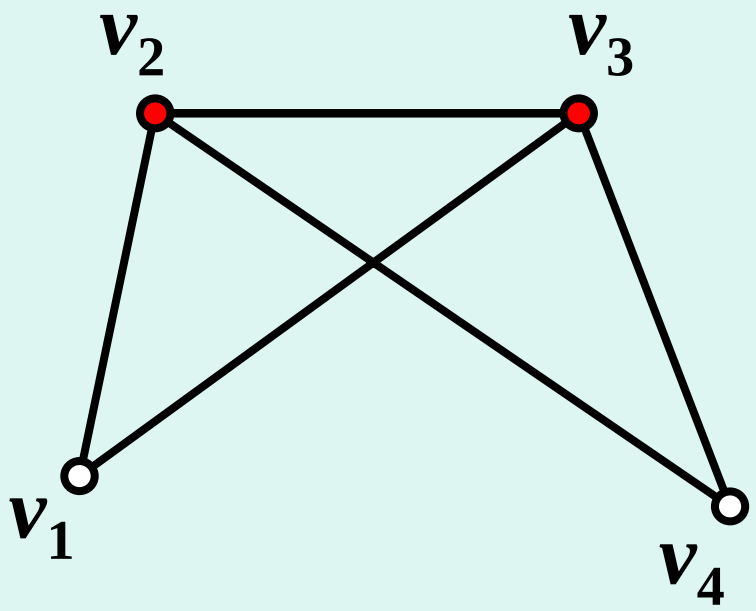
证：取 $a_0 \notin V$, 将 $\{ u, v \}$ 替换成 $\{ a_0, u, v \} \in C$.

$A = \{ a_0 \} \cup V$

$C = \{\{ a_0, u, v \} : \{ u, v \} \in E \}$

$J = K + |E|$

变换实例



例如 $G = (V,E)$,
$K = 2$
是顶点覆盖的肯定实例

变换成：

$A=\{a_0, v_1, v_2, v_3, v_4\}$

$C=\{\{a_0, v_1, v_2\},\{a_0,v_1,v_3\},\{a_0,v_2,v_3\},\{a_0,v_2,v_4\},\{a_0,v_3,v_4\}\}$

$J = 2+5 = 7$

并运算序列如下：

$<z_1=\{a_0\}\cup\{v_2\}, z_2=\{a_0\}\cup\{v_3\}, z_3=\{v_1\}\cup z_1, z_4=\{v_1\}\cup z_2,$

$z_5=\{v_3\}\cup z_1, z_6=\{v_4\}\cup z_1, z_7=\{v_4\}\cup z_2 >$

肯定实例变到肯定实例

设 V'是 G 的顶点覆盖，$|V'|\le K$, 不妨设

$V' = \{v_1, v_2, \dots, v_k\}$. 令$|E|=m$, $E=\{e_1, e_2, \dots, e_m\}$,

$\forall e_j \in E$, $e_j=\{u_j, v_{r[j]}\}$, 且$v_{r[j]} \in V'$, 如下构造并运算序列：

$<z_1=\{a_0\}\cup\{v_1\}, \dots, \ z_k=\{a_0\}\cup\{v_k\},$

$z_{k+1}=\{u_1\}\cup z_{r[1]}, z_{k+2}=\{u_2\}\cup z_{r[2]}, \dots, z_{k+m}=\{u_m\}\cup z_{r[m]}>$

则该序列满足要求

反之，若 $S = < z_1=x_1\cup y_1, z_2=x_2\cup y_2, \dots , z_j=x_j\cup y_j >$ 是整体计算的肯定实例, $j\le J$. 假设 S 是所有满足要求的序列中最短的，且在所有最短序列中包含形如 $z_i=\{u\}\cup\{v\}$ 的运算最少.

先证明 S 中包含0个上述运算. 假如 S 中有 $z_i=\{u\}\cup\{v\}$, $u, v\in V$, $\{u,v\}\notin C$, 因为$\{u,v\}$不是3元集. 由最短性，后边一定用 $\{u,v\}$ 构造 S 中的3元集 $\{a_0, u, v\}$, 即 $\{ u,v \}\cup\{a_0\}$ 或 $\{a_0\}\cup\{u,v\}$. 不妨设为前者. 那么将

$$z_i=\{u\}\cup\{v\}, \{a_0, u, v\}=z_i\cup\{a_0\}$$

替换成

$$z_i=\{a_0\}\cup\{u\}, \{a_0, u, v\}=z_i\cup\{v\}$$

并不增加序列长度，但是减少了形如 $\{u\}\cup\{v\}$ 形式的运算，与假设矛盾.

由上边的分析，S中的运算具有下述形式：

$$z_i = \{a_0\} \cup \{u\} \qquad u \in V$$

$$\{a_0, u, v\} = \{v\} \cup z_i \qquad \{u,v\} \in E$$

因为$|C|=|E|$, C中元素为3元集，故 S 中恰好含有$|E|$个形如 $\{a_0, u, v\}=\{v\}\cup z_i$ 的运算和 $J-|E| \le K$ 个形如$z_i=\{a_0\}\cup\{u\}$形式的运算. 令

$V'=\{\ u \in V :\ z_i=\{a_0\}\cup\{u\}$ 是 S 中的运算 $\}$

则$|V'| \le K$, 且V'为顶点覆盖.

例11 三角划分

实例：$G=(V,E)$, $|V|=3Q$, $Q \in Z^+$.

问：能否把 V 划分成 Q 个不交的三顶点集$V_1, V_2, \ldots, V_Q$, 使得对每个$V_i=\{ v_i[1], v_i[2], v_i[3] \}$, $i=1, 2, \ldots, Q$, $\{ v_i[1], v_i[2] \}, \{ v_i[1], v_i[3] \}, \{ v_i[2], v_i[3] \} \in E$?

已知问题：X3C $\Rightarrow$ 目标问题：三角划分

基本单元：C中3元子集 $\Rightarrow$ G中的子图

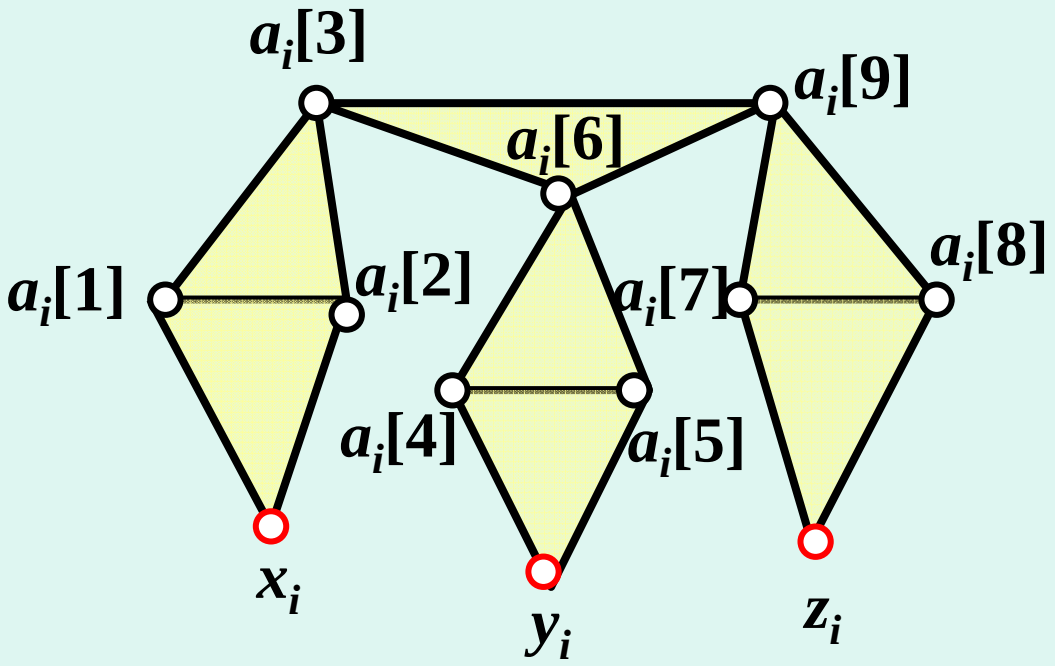
证：设集合X, $|X|=3q$, X 的三元子集的集合 C 是 X3C 的实例

替换

$\forall c_i=\{x_i, y_i, z_i\}\in C$ 替换成下面的子图.

替换后的 $|V|$ 和 Q 满足：$|V|=|X|+9|C|=3q+9|C|$，

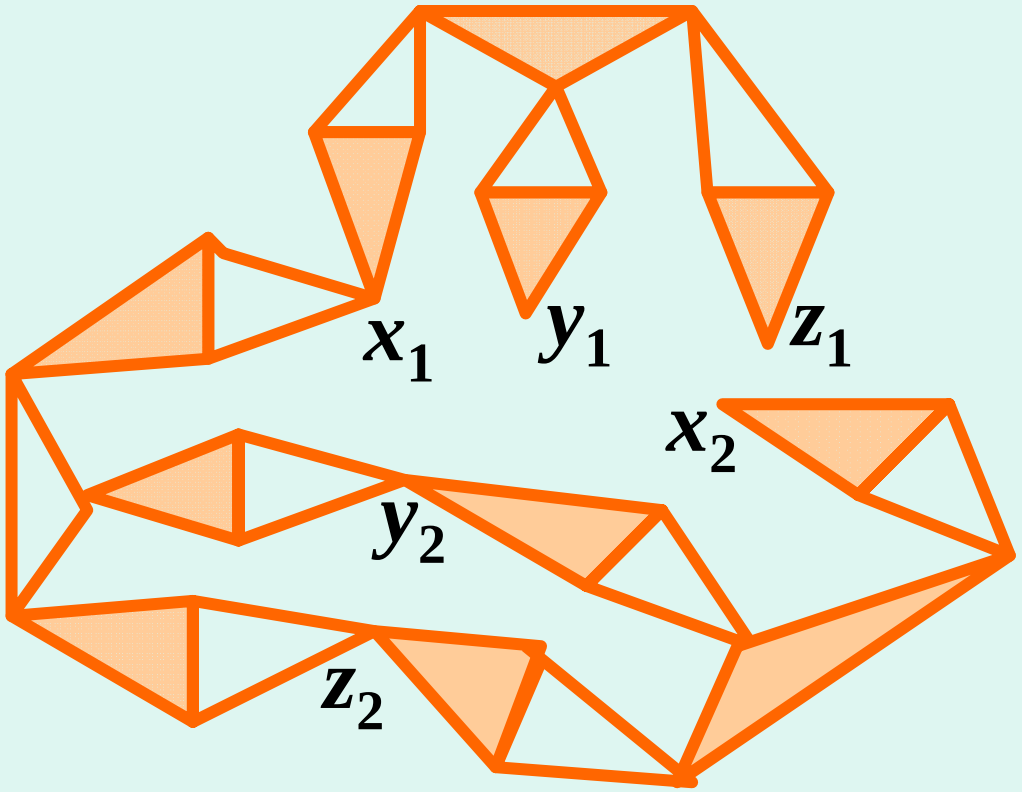
$Q=q+3|C|$



替换实例

$X = \{ x_1, y_1, z_1, x_2, y_2, z_2 \}$

$C = \{\{ x_1, y_1, z_1 \}, \{ x_1, y_2, z_2 \}, \{ x_2, y_2, z_2 \}\}$



肯定实例变换成肯定实例

令$C' \subseteq C$是 X 的恰当覆盖，

对于$c_i \in C'$, $c_i=\{x_i, y_i, z_i\}$, 取底下的三个三角形和顶上的一个三角形（下划分）；对于$c_i \notin C'$, $c_i =\{x_i, y_i, z_i\}$, 取中间的三个三角形（上划分）.

X的每个元素恰在一个三角形中，$3q+9|C|$ 个顶点恰好分在 $q+3|C|$ 个三角形中，构成三角划分.

反之，设$V = V_1 \cup V_2 \cup \ldots \cup V_q$ 是G的三角划分，每个E_i 或是上划分，或是下划分. 取每个下划分对应的 $\{x_i, y_i, z_i\}$ 得到 c_i，所有的 c_i 构成集合C'. C'中每个 X 的元素恰好出现一次，C'构成恰当覆盖.

例12　区间排序

实例：有穷任务集 T, $\forall t \in T$, 开放时间 $r(t)$, 截止时间 $d(t)$, 需要时间 $l(t)$, 其中 $r(t) \in N$, $d(t)$, $l(t) \in Z^+$.

问：是否存在关于 T 的可行调度，即是否存在函数 $\sigma: T \to N$ 使得 $\forall t \in T$ 满足：

$$\sigma(t) \geq r(t)$$

$$\sigma(t) + l(t) \leq d(t)$$

$$\forall t' \in T - \{ t \},$$

$$\sigma(t') + l(t') \leq \sigma(t) \text{ 或 } \sigma(t) + l(t) \leq \sigma(t') \text{ ?}$$

已知问题：均分　$\Rightarrow$ 目标问题：区间排序

基本单元：A中元素 $\Rightarrow$ T中的任务

替换

证 设 A 和 $s(a)\in Z^+$ $(\forall a\in A)$ 为均分的实例.

$\forall a\in A$ 将 a 替换成 $t_a\in T$, $d(t_a)=B+1$，$l(t_a)=s(a)$ ，其中 $B=\sum_{a\in A}s(a)$

实施者 $\bar{t}$ $r(\bar{t})=\left\lceil\frac{B}{2}\right\rceil$, $d(\bar{t})=\left\lceil\frac{B+1}{2}\right\rceil$, $l(\bar{t})=1$

B为奇数，则不能调度

$$r(\bar{t})=d(\bar{t}),\quad r(\bar{t})+l(\bar{t})>d(\bar{t})$$

若B为偶数，存在均分

$$r(\bar{t})=\frac{B}{2},\quad d(\bar{t})=\frac{B}{2}+1=r(\bar{t})+l(\bar{t}),\text{必有}\sigma(\bar{t})=\frac{B}{2}.$$

分量设计法

分量设计法　根据目标问题的实例设计分量(分量的成分与目标问题相关)，实现已知 NPC 问题的实例（分量的功能与已知问题相关）.

3SAT变换到VC，其中分量有真值安排分量、满足性检验分量等，这些分量都是子图，用来实现三元可满足性问题的实例.

例13 最小拖延排序

实例：任务集 T，$\forall t \in T$, 完成时间 $l(t)=1$,

截止时间 $d(t) \in Z^+$.

T 上的偏序 π，非负整数 $K \le |T|$

问：是否存在可行调度 σ 使得被拖延的任务数不超过K，即

$\sigma: T \to \{0, 1, \ldots, |T|-1\}$, 使得

若 $t \neq t'$, 则 $\sigma(t) \neq \sigma(t')$

若 $t \pi t'$, 则 $\sigma(t) < \sigma(t')$

$|\{ t \in T: \sigma(t)+1 > d(t)\}| \le K$?

变换

证 将团变换到最小拖延排序.

设$G = (V,E)$ 是团问题的实例，

顶点分量：任务 t，$d(t) = |V|+|E|$

边分量：任务 t, $d(t) = J(J+1)/2$,

任务集：$T = V \cup E$

$K = |E| - J(J-1)/2$

偏序：$e = \{ u,v \}$，u, v 任务完成后才能开始 e 任务,

即 $t \pi t' \Leftrightarrow t \in V, t' \in E$, t 是 t'的一个端点

设给定最小拖延排序的肯定实例，顶点任务不会拖延，只有边任务可能拖延，在截止时间之后完成的任务是被拖延的任务. 拖延的任务数为 $|E|-J(J-1)/2$，不拖延的任务数为 $J(J-1)/2$. 因此，在截止时间之前安排的边数为 $J(J-1)/2$ （在安排这些边之前至多安排 J 个顶点，而这些顶点构成大小为 J 的团）。

反之，若 G 中含大小为 J 的团，则先安排团的 J 个顶点任务，接着相关的 $J(J-1)/2$ 个边任务，然后是剩下的顶点任务，那么被拖延的任务数为 $|E|-J(J-1)/2$.

NP完全性理论的应用

用NP完全性理论进行子问题分析

- 子问题的定义
- 子问题的计算复杂性
- 子问题的NP完全性证明

NP难度

- 搜索问题
- Turing归约
- NP-hard, NP-easy

子问题的定义及其实例

定义 设判定问题 $\pi=(D\pi,Y\pi)$, 若

$$D\pi'\subseteq D\pi,\ Y\pi'\subseteq Y\pi\cap D\pi',$$

则称 $\pi'=(D\pi',Y\pi')$ 是 π 的子问题.

子问题的问题与原问题一样，定义域缩小.

通过对实例的参数加以限制得到子问题.

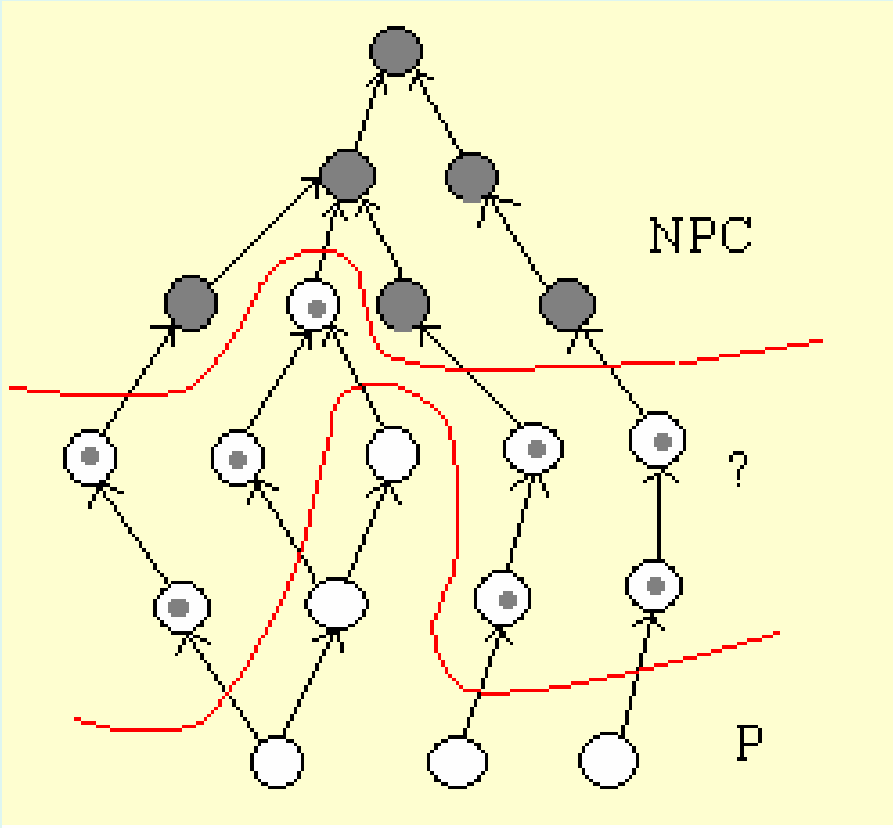
子问题实例

限制子句中文字的个数得到SAT的子问题3SAT,2SAT

限制顶点度数、平面图、二部图、无圈图等得到图论问题的子问题

限制常数 K 的大小，如 $K=2$, $K=3$ 等得到子问题

子问题的计算复杂性



努力扩大已知区域，缩小未知区域

当**P≠NP**时，存在不属于**NPC**也不属于**P**的问题

有先行约束的多处理机调度问题

优化问题:

给定任务集 T, m 台机器,$\forall t\in T$, $l(t)\in Z^+$, T上的偏序$\prec$.

若$\sigma: T\rightarrow\{0, 1, \ldots, D\}$满足下述条件,则称 T 为可行调度.

$\forall t\in T, \sigma(t)+l(t)\leq D$

$\forall i, 0\leq i\leq D, |\{t\in T : \sigma(t)\leq i<\sigma(t)+l(t)\}|\leq m$

$\forall t, t'\in T, t\prec t'\Leftrightarrow \sigma(t)+l(t)\leq\sigma(t')$

求使得 D 最小的可行调度.

条件说明:

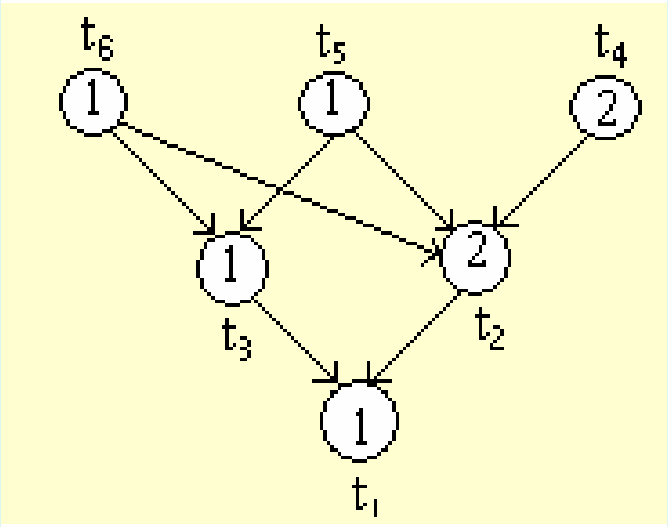
任务在截止时间前完成

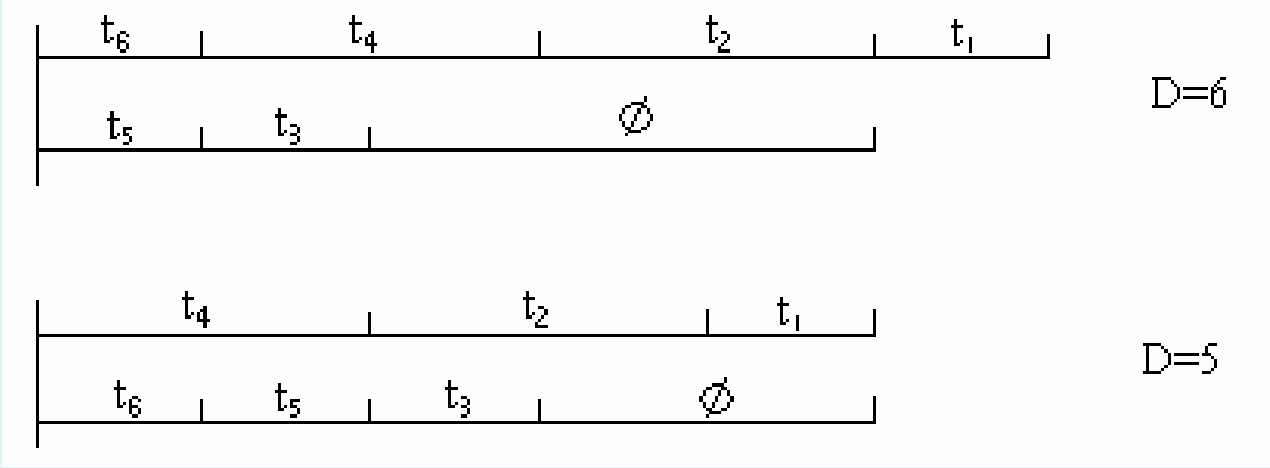
同时工作的台数不超过 m

有偏序约束的任务必须按照约束先行

实例

任务集如图所示， ***m*=2,** 求使得 ***D*** 最小的可行调度.





判定问题

实例：任务集T, m台机器,$\forall t\in T$, $l(t)\in Z^+$,T上的偏序$\prec$,
截止时间 $D\in Z^+$.

问：是否存在小于等于 D 的可行调度？

子问题通过限制参数(机器数、工作时间、偏序)构成

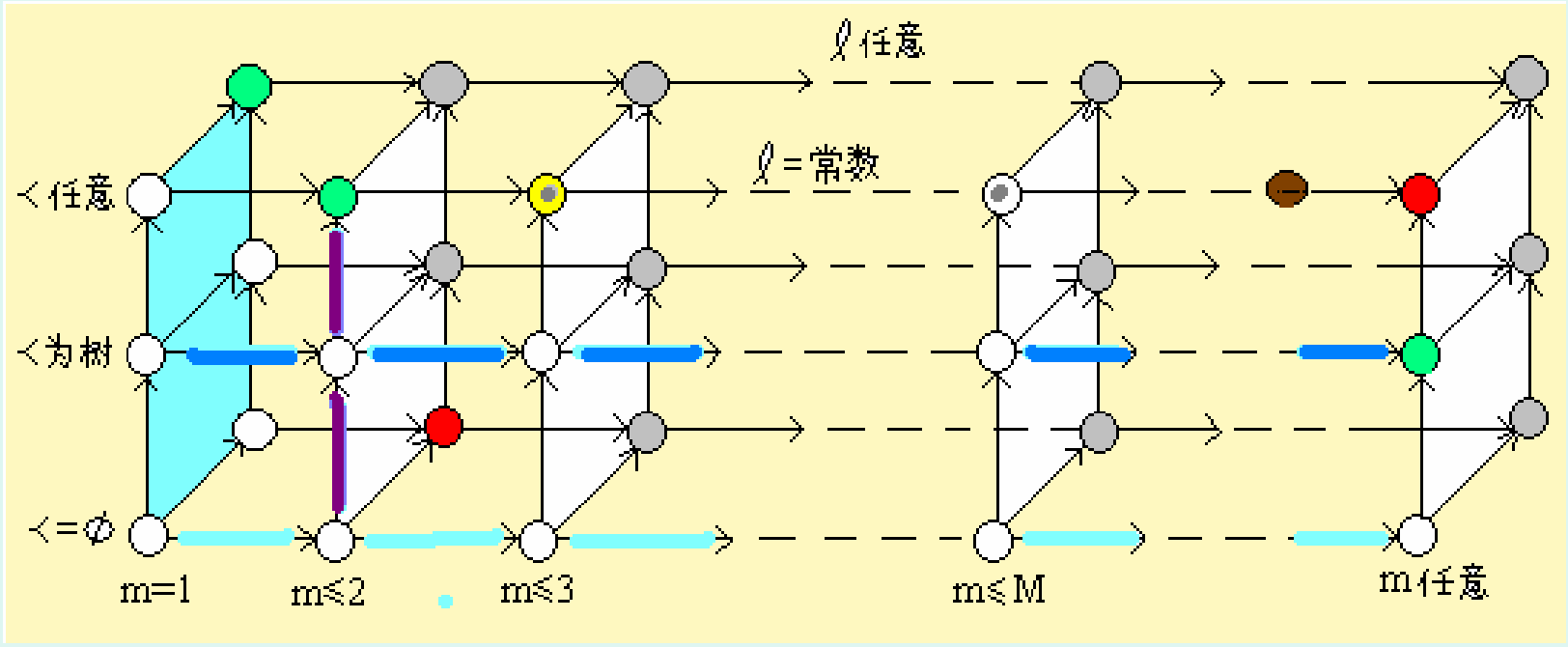
参数	限　制		
偏序	$\varnothing$	树	任意
m大小	$m \le 1, 2, \ldots,$ 某个常数		m 任意
l 大小	l 为常数		l 任意

调度问题的子问题结构

从上到下：偏序任意、树形偏序、无偏序约束

从左到右：处理器台数限制逐步放大

从前到后：各任务等长工作时间、任意工作时间



某些子问题属于P的说明

m =1, l 任意，偏序任意（绿色面积）

算法：

1. 计算所有任务的工作时间之和 **Time**
2. **if Time** $\leq D$,存在可行调度.
3. 按照偏序从高层依次取任务，进行安排即可.

m 任意, l 为常数，偏序为空（浅兰色箭头）

算法：

1. **Time** $\leftarrow \lceil |T|/m \rceil l$
2. **if Time** $\leq D$,存在可行调度.
3. 将任务按顺序依次分配到 m 台机器上.

某些子问题属于P的说明(续)

***m* 任意，*l* 为常数，偏序为树（深兰色箭头）**

关键路径算法:

1.按照偏序关系，从树叶依次拿任务，每次至多下移 ***m*** 个结点.

2.将分配次数与 ***D*** 比较,如果不超过 ***D*,** 回答**Yes.**

分配次数至少为树高,可以证明时间为多项式时间.

***m*=2, 偏序任意, *l* 为常数（紫色箭头）**

算法：**H.N.Gabow,An almost linear algorithm for two processors scheduling, J.Assoc. Comput.Math,29, 1982, 766-780.**

术语简介

极大多项式可解的子问题π：（绿色）

π 是多项式时间可解的,并且不存在其他多项式可解的子问题π',使得π 是π'的子问题.

极小NP完全的子问题π（红色）

π 是NP完全的，并且不存在其他NP完全的子问题π'使得π'是π 的子问题.

极小未解决的子问题π：（黄色）

π 的子问题都属于P, 但不知π 是否属于P，也不知P是否属于NPC.

极大未解决的子问题π: (棕色)

π的父问题都是NP完全的，但不知π 是否属于P，也不知P是否属于NPC.

子问题NP完全性证明

许多子问题的**NP**完全性证明采用局部替换法.

以图论为例，平面性限制，顶点度数限制等形成子问题.

以顶点度数 ***D*** 为例，子问题分析如下：

问题	**P**	**NPC**
VC	$D \leq 2$	$D \geq 3$
HC	**2**	**3**
顶点三着色	**3**	**4**
反馈顶点集	**2**	**3**
团	给定***D***	任意

局部替换法的实例

对于由于顶点度数 D 限制得到子问题的NPC证明，通常采用局部替换法，设计新的保持性质不变，又能降低顶点度数的顶点替换.

例 2　顶点 3 着色问题

原问题：任意无向图的顶点 3 着色问题

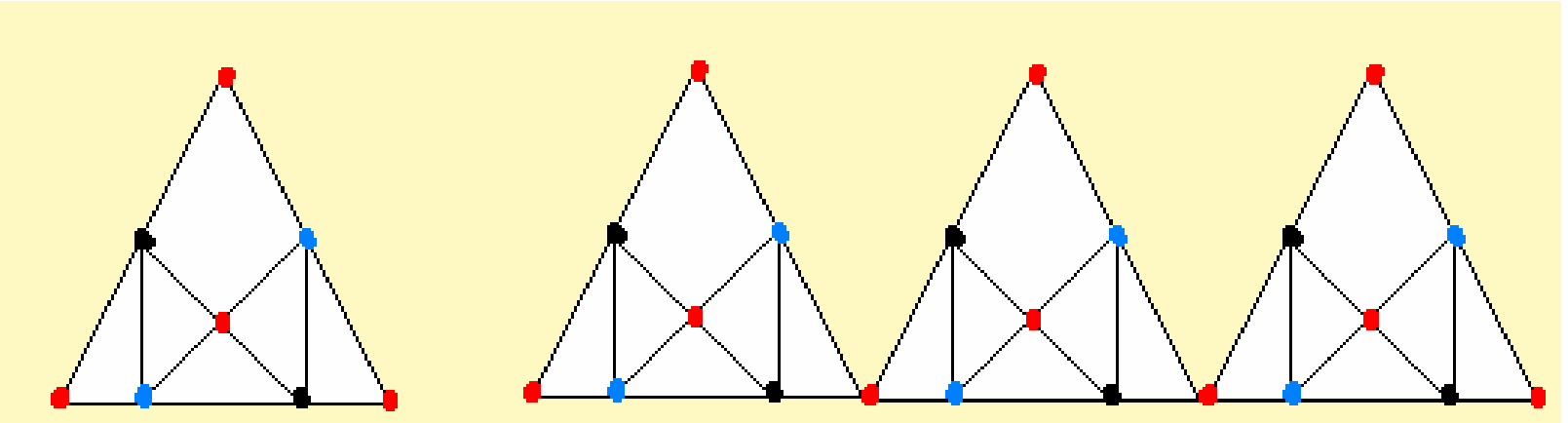
子问题：最大度数 D 不超过 4 的顶点 3 着色问题

基本单元：度数 k 大于等于 5 的顶点

替换成：子结构 H_k（内部顶点度数不超过 4 的连续 $k-2$个三角形）

H_k 的结构

对于G中的 k 度顶点，$k \geq 5$, 替换成 k-2个连续的三角形H_k,



H_k 具有 k 个外顶点（和外部顶点邻接）

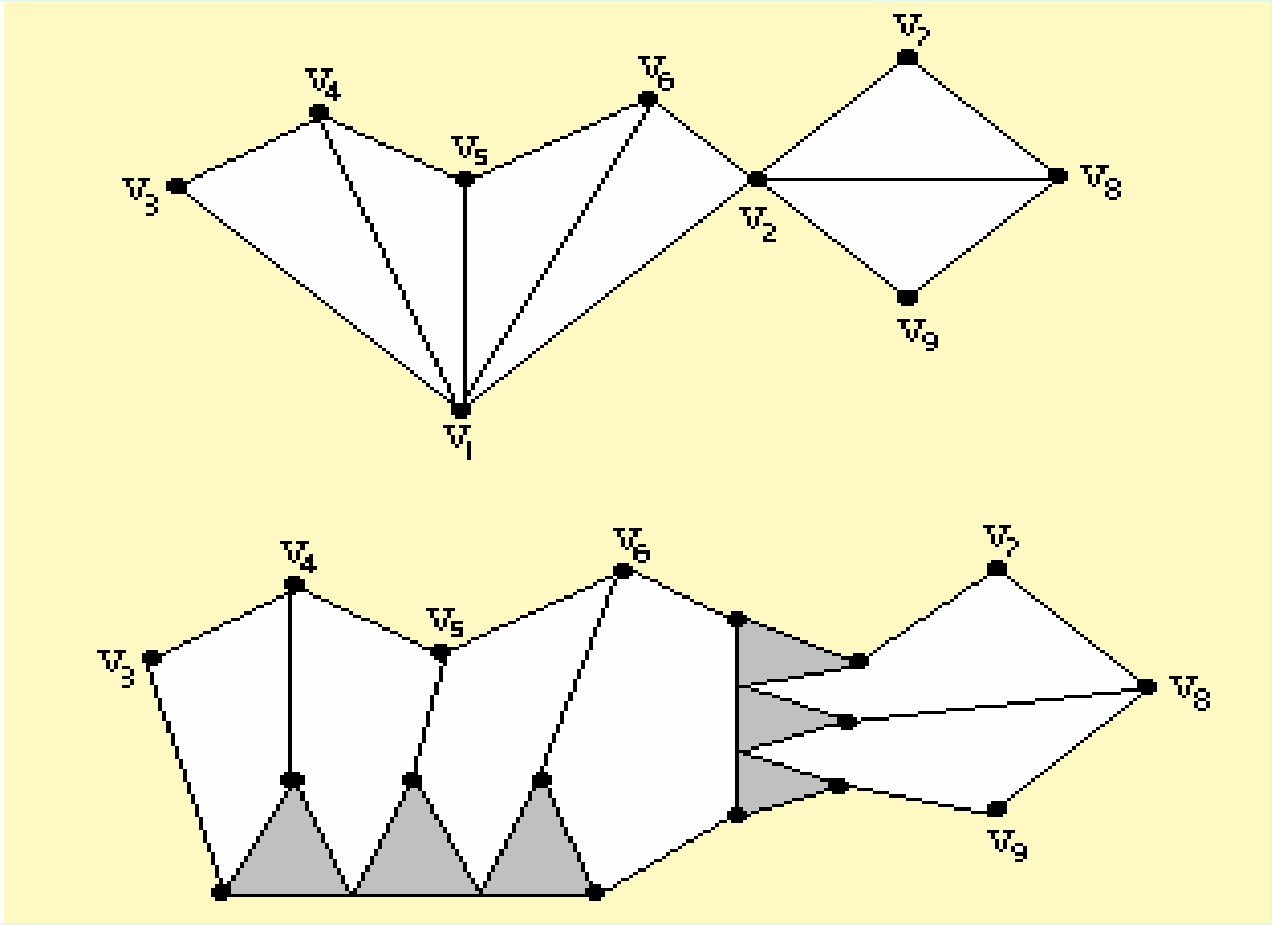
H_k 的顶点度数不超过 4

H_k 的出口（外顶点）度数为 2

H_k 可着 3 色，且每个外顶点着同色

替换实例

下图的 v_1 和 v_2 是5度顶点，被 H_5 替换得到新的图，其中灰色区域表示替换后的两个 H_5.



NP难度

搜索问题

定义

判定问题与搜索问题的关系

搜索问题的形式定义

Turing归约

定义

性质

NP难，NP易与NP等价

Turing归约的应用

搜索问题的定义及实例

搜索问题定义：一个搜索问题 π 有实例集 $D\pi$, 对于π中的任何实例 I，有一个有穷的解集合 $S\pi[I]$.

如果存在算法 A，对于任何实例 $I \in D\pi$, A 都停机， 并且如果 $S\pi[I]=\varnothing$，则回答无解，否则给出 $S\pi[I]$中的一个解，那么称 A 解搜索问题 π.

例如，巡回售货员的优化问题，$S_{TS}[I]$就是实例 I 中所有最短巡回路线的集合. Hamilton 回路的构造性问题，当 G 中不存在Hamilton回路时回答无解，否则给出一条Hamilton回路. $S_{HC}[I]$中是所有的HC的集合.

判定问题与搜索问题之间的关系

判定问题是搜索问题的特例.

对于判定问题 π，若 ***I*** 属于***Y*π**, 则 ***S*π[*I*]={ Yes },**

若 ***I*** 属于***D*π-*Y*π**, 则 ***S*π[*I*]={ No }.**

搜索问题的形式说法

设Σ是有穷字符表，$R \subseteq \Sigma^+ \times \Sigma^+$ 称为Σ上的符号串关系，

$\Sigma^+ = \Sigma^* - \{\varepsilon\}$，长度至少为 1 的有穷串的集合.

搜索问题 π，I 为 π 的任意实例，I 在合理编码系统 e 下的编码为 x, 其编码系统的字符集为 Σ. 令

$R[\pi,e]=\{ (x,y)$: $x \in \Sigma^+$是实例 $I \in D\pi$在e下的编码，

$y \in \Sigma^+$是解$s \in S\pi[I]$在e下的编码 $\}$

则 R 中的有序对由有解的实例的编码及其对应的一个解的编码构成

搜索问题的形式说法(续)

定义函数 $f:\Sigma^+\to\Sigma^*, x\in\Sigma^+$，如果存在 $y\in\Sigma^+$使得 $(x,y)\in R$，则令 $f(x)=y$, 否则 $f(x)=\varepsilon$，那么称函数 f 实现符号串关系 R.

如果存在 DTM 程序 M 计算的函数 f_M 实现符号串关系 R，则称 M 解符号串关系 R.

如果存在多项式时间的 DTM 程序 M 解 $R[\pi,e]$，则称搜索问题 π在编码系统 e下是多项式可解的.

Turing归约
Turing Reduction

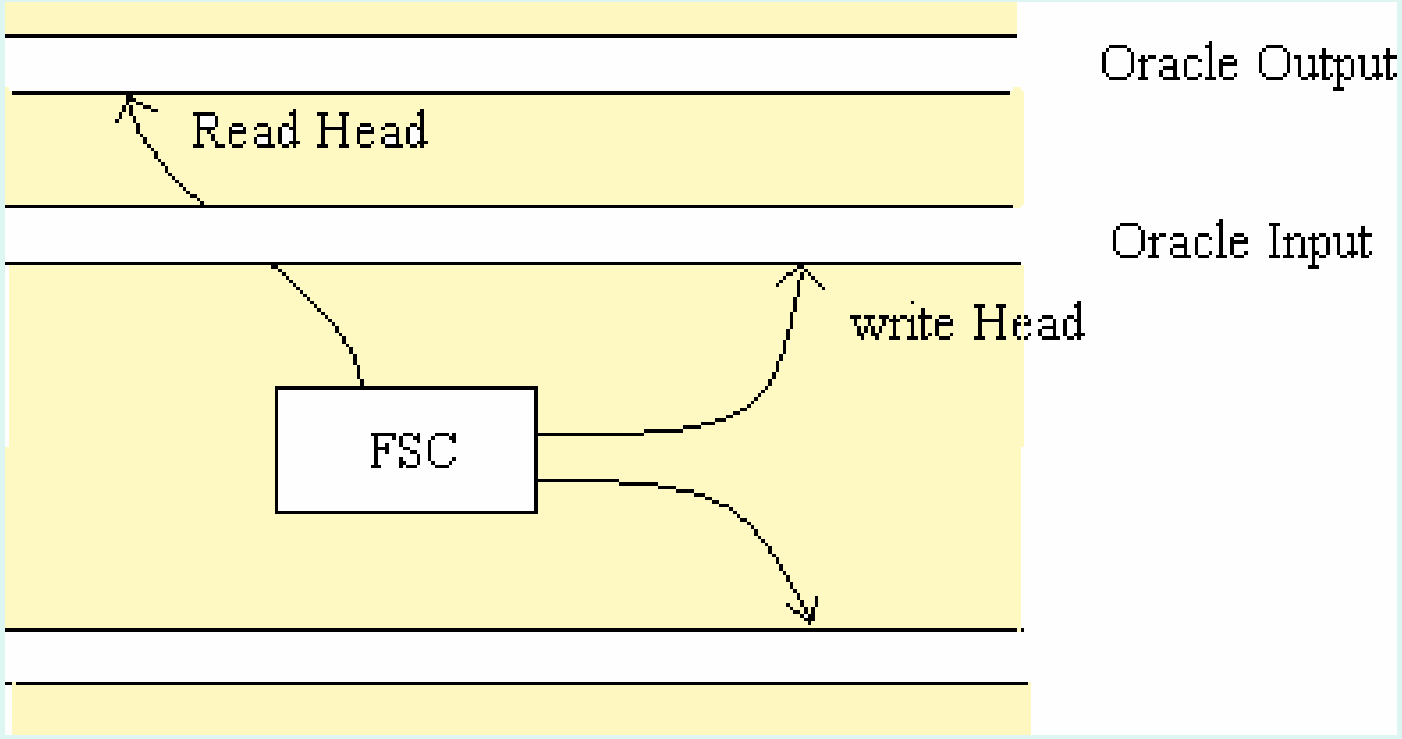
非形式定义

设 π_1, π_2是搜索问题，A 是利用解 π_2 的假想子程序 s 解 π_1的算法，且只要 s 是多项式时间的，A也是多项式时间的，则称算法 A 是从 π_1到 π_2 的多项式时间的 **Turing归约**. 这是也称 π_1 **Turing**归约到 π_2，记作$\pi_1 \propto_T \pi_2$.

Turing归约(续)

形式定义： **带外部信息源的 Turing 机 OTM**

(Oracle Turing Machine)



OTM

有穷带字符集 Γ

输入字符集 $\Sigma \subset \Gamma$，空白字符 $b \in \Gamma - \Sigma$.

有穷状态集 Q, 其中含有q_0, q_h, q_c, q_r分别表示初始状态、停机状态、访问外部信息源状态和恢复计算状态.

转移函数 $\delta: (Q - \{q_h, q_c\}) \times \Gamma \times (\Sigma \cup \{b\})$

$\rightarrow Q \times \Gamma \times (\Sigma \cup \{b\}) \times \{-1,1\} \times \{-1,1\} \times \{-1,1\}$

基本　只写　　基本　只写　只读

OTM的计算

输入 $x \in \Sigma^+$

初始：

x 写在基本带上方格 1 到 $|x|$，其余方格为 b.

Oracle的输入带(只写带)为 b.

每个带的带头置方格 1.

初始状态为 q_0.

OTM的计算(续)

计算：

若 $q=q_h$，则计算结束，基本带方格1 到最右边非空白方格的符号串是输出串.

若$q \in Q-\{q_h, q_c\}$, s_1, s_2分别为基本带和只读带扫描的字符，且

$$\delta(q,s_1,s_2) = (q',s_1',s_2', \Delta_1, \Delta_2, \Delta_3)$$

那么状态变成 q',基本带字符改写为 s_1', 只写带字符改写为 s_2', 基本带、只写带、只读带带头分别移动 Δ_1, Δ_2, Δ_3.

OTM的计算(续)

若 $q = q_c$,

y 为 Oracle 输入带的字符串，

$z = g(y)$为外部信息源函数，

在一步之内将外部信息源 Oracle 的输出带(只读)的 1 到 $|z|$ 方格写上 z，其余为 b；将输入带清为空白；将只读头，只写头置 1；q_c 变成 q_r. 基本带的内容、读写头不变.

相对化的OTM程序

相对化的**OTM**程序**:**

设 Mg 是 M 与外部信息源 g 相结合的相对化**OTM**程序. 其带字符表为 Γ，输入字符表为 Σ.

多项式时间的**OTM**程序**:**

如果存在多项式 P 使得对于一切 $x\in\Sigma^+$，Mg 在$P(|x|)$步内停机，则称 Mg 是多项式时间的 **OTM**程序.

OTM程序计算函数 f**:**

设 Mg 对一切 $x\in\Sigma^+$停机, 且停机时方格**1**到最右边非空白方格的字符串是$f_{Mg}(x)$,那么它计算函数f_{Mg}: $\Sigma^+ \to \Sigma^*$

Turing归约的形式定义

多项式时间的Turing归约：

设R_1,R_2是Σ上的两个符号串关系，M是输入字符集为Σ的OTM程序，如果对于每个实现 R_2 的函数 g:$\Sigma^+ \to \Sigma^*$，相对化的OTM程序 Mg 都是多项式时间的OTM程序，且 Mg 计算的函数实现 R_1，那么称 M 是从 R_1到 R_2的多项式时间的Turing归约，记作 $R_1 \propto_T R_2$

两种定义的对比

非形式定义	形式定义
搜索问题 π	符号串关系 R
实例 I	符号串 x
解集合$S\pi[I]$	$\{ y \in \Sigma^+ \mid (x,y) \in R \}$
回答无解	ε
解π的算法 A	解 R 的 DTM程序
子程序 s	外部信息源 g
调用子程序 s 解π的算法 A	相对化的OTM程序M^g
从π_1到π_2的 Turing归约	从 R_1到 R_2的Turing归约

Turing归约的性质

传递性

$\pi_1 \propto_T \pi_2$，$\pi_2 \propto_T \pi_3 \Rightarrow \pi_1 \propto_T \pi_3$

多项式变换是 **Turing归约** 的特例.

设 π_1多项式变换到 π_2, 如下设计解 π_1 的算法 *A*：

1. 对于π_1的任何实例 *I*，先将 *I* 变换成 π_2实例 *I'*,
2. 利用解 π_2 的假想子程序 *s* 对 *I'* 识别.
3. 如果 *I'*是肯定实例，则 *I* 也是肯定实例；
4. 如果 *I'*是否定实例，则 *I* 也是否定实例.

易见只要 *s* 是多项式时间的，则 *A* 也是多项式时间的.

因此$\pi_1 \propto_T \pi_2$.

NP难度

定义

设 π 是搜索问题，如果存在 NP 完全问题 π' 使得 $\pi' \propto_T \pi$，则称 π 是**NP-hard**. 这意味着在多项式可计算的角度看，π 至少 像 NPC 问题一样难.

设 π 是搜索问题，如果存在 NP 问题 π' 使得 $\pi \propto_T \pi'$，则称 π 是**NP-easy**.

设 π 是搜索问题，如果 π 是 NP-hard, 同时也是 NP-easy, 则称 π 是**NP等价**的.

NP-easy ⩽ NP等价(包含NPC问题) ⩽ NP-hard

相关结果

所有的 **NPC** 问题都是 **NP-hard.**

因为多项式变换是**Turing**归约的特例.

若 $\pi \in$ **NPC,** 则 π 多项式可解 $\Leftrightarrow$ **P = NP**

若 π 是 **NP-hard,** 则 π 多项式可解 $\Rightarrow$ **P = NP**

若 π 是 **NP-easy,** 则 π 多项式可解 $\Leftarrow$ **P = NP**

若 π 是 **NP**等价, 则 π 多项式可解 $\Leftrightarrow$ **P = NP**

相关结果(续)

设π是判定问题，

π为NPC $\not\Rightarrow$ π^c为NPC

π为 NP-hard $\Rightarrow$ π^c为 NP-hard

证明：设 π为 NP-hard，任取实例 $I \in D\pi$, I 也是 π^c 的实例，且

$$I \in Y\pi \Leftrightarrow I \in D\pi^c - Y\pi^c$$

设 s 是解π^c的算法，如下得到解 π 的算法：先调用 s 对于实例 I 求解. 如果回答"Yes", 则回答"No", 否则回答"Yes". 从而证明了π Turing归约到 π^c. 据Turing归约的性质，π^c 为NP-hard.

Turing归约的应用

1. 证明许多 **NPC** 问题所对应的优化或构造问题为**NP-hard.**
2. 证明某些非 **NP** 类的判定问题是 **NP-hard**

说明:

1. 将解对应优化问题或构造问题的算法作为解判定问题算法的子程序，得到解判定问题的算法，从而构造了从判定问题到对应优化问题或构造问题的**Turing**归约.

实例—证明NP等价

例1 巡回售货员问题（TSO）是 NP等价的.

证：易证 TSO是 NP-hard. 下面证明 TSO 是NP-easy.

引入中间问题：巡回售货员的延伸问题（TSE）

TSE

实例：有穷城市的集合 $C = \{c_1, c_2, \cdots, c_m\}$

距离 $\forall c_i, c_j \in C, d(c_i, c_j) \in Z^+$,

长度限制 $B \in Z^+$,

部分旅行路线 $\vartheta = < c_{\pi(1)}, \cdots, c_{\pi(k)} >$

问：ϑ是否可以延伸成全长不超过 B 的全程旅行

$< c_{\pi(1)}, \cdots, c_{\pi(k)}, c_{\pi(k+1)}, \cdots, c_{\pi(m)} >$?

易证 TSE属于NP.

TSO 到 TSE 的 Turing归约

设 $s(C,d,\vartheta,B)$是解 TSE 的子程序，其中 C 为城市集，d为距离函数，ϑ为部分旅行，B为长度限制.

下面构造解TSO的算法.

思路:

用二分法确定最短路旅行长度 B^*

旅行长度界于$m \to m \times d,\ d = \max\{ d(c_i, c_j)\}$

每次取中点值验证是否存在能延伸到此长度的旅行

根据最小长度值 B^*确定旅行路线

从c_1开始，依次检查$<c_1,c_2>,<c_1,c_3>\ldots$是否能延伸到 B^*长度的旅行，选择第一个可延伸的顶点c_i.

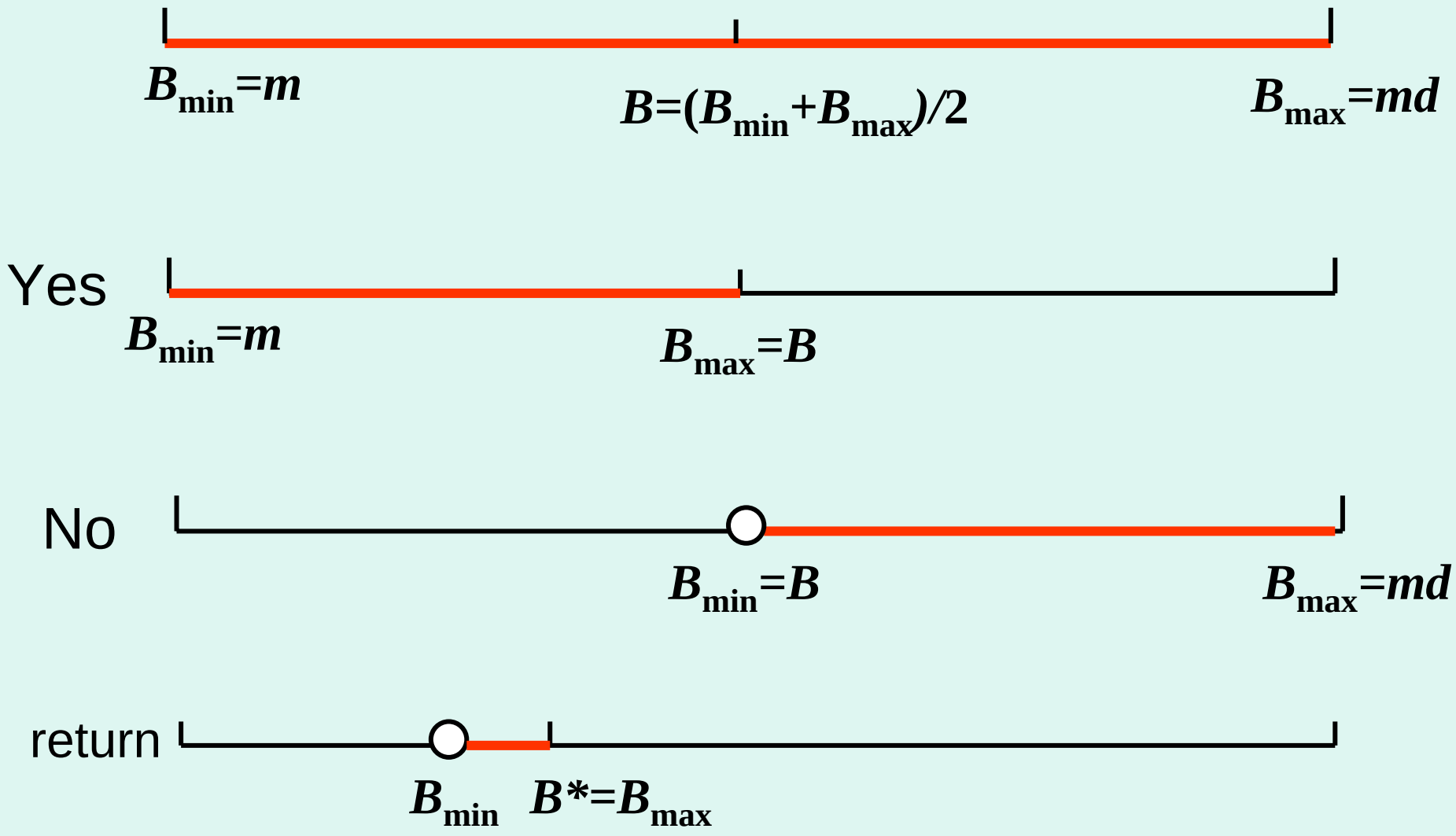
按照上面方法确定后面的其他顶点.

算法 MinLength

设 $s(C,d,\vartheta,B)$是解 TSE 的子程序，其中 C 为城市集，d 为距离函数，ϑ为部分旅行，B为长度限制.

算法 Minlength（二分法确定最短旅行长度 B^*）

1. 令$Bmin \leftarrow m$, $Bmax \leftarrow m \cdot \max\{ d(c_i,c_j): c_i, c_j \in C \}$
2. 若$Bmax - Bmin = 1$, 则 $B^* \leftarrow Bmax$, 结束
3. $B \leftarrow [(Bmin+Bmax)/2]$
4. $s(C,d,<c_1>,B)$
5. 若回答"Yes", $Bmax \leftarrow B$, 否则$Bmin \leftarrow B$
6. 转2.


$B_{min}=m$
$B=(B_{min}+B_{max})/2$
$B_{max}=md$
Yes
$B_{min}=m$
$B_{max}=B$
No
$B_{min}=B$
$B_{max}=md$
return
B_{min}
$B^*=B_{max}$

算法 FindSolution

算法 FindSolution (找解)

1. $i \leftarrow 2, j \leftarrow 2$； $M \leftarrow \{2, 3, \ldots, m\}$
2. $j \leftarrow M$ 中的最小值
3. $\vartheta = <c_1, c_j>$
4. $s(C, d, \vartheta, B^*)$
5. if 回答"Yes", 则
6. $\quad i \leftarrow i+1$, $M \leftarrow M-\{j\}$
7. else
8. $\quad$ 从ϑ中去掉 c_j
9. $\quad$ 从 M 中选择大于 j 的最小值 k
10. $\quad$ 将 c_k 加入到 ϑ 的最后项
11. $\quad M \leftarrow M \cup \{j\}$
12. 如果 $i \le m$, 转4； 否则停机

复杂度分析

至多$m-2$次调用s可以找到第2个城市，至多$m-3$次调用s可以找到第3个城市，…，至多1次调用s可以找到第$m-1$个城市。调用s的总次数至多为

$$(m-2)+(m-3)+...+1=\frac{(m-1)(m-2)}{2}$$

为 m 的多项式，而 TSO 的实例规模为 $m+\log B\max$，所以是从 TSO 到 TSE 的 Turing归约.

而 TSE 是 NPC 问题，因为判定问题 TS 是 TSE的子问题，相当 $\vartheta = <c_1>$的情况. 因此，TSO Turing归约到 NP问题 TSE，从而证明了TSO 是 NP-easy.

类似地可以证明六个基本NPC问题对应的优化问题都是NP-hard.

实例--分析非NP类问题的难度

例2　第 k 个最大子集

实例：有穷集 A, $\forall a \in A$, $S(a) \in Z^+$, 正整数 $B \le S(A)$, $K \le 2^{|A|}$, 其中 $S(A) = \sum_{a \in A} S(a)$

问：是否至少存在 K 个不同的子集 A',满足$S(A') \le B$?

这个问题不是 NP 问题. 要猜想 A 的 K 个子集，K 不是 $|A|$ 的多项式, 输入规模为

$$|A| \lceil \log K \rceil \lceil \log S(A) \rceil$$

没办法用它的多项式个符号写下猜想.

可以证明这个问题是 NP-hard.

Turing归约

命题：第 k 个最大子集问题是 NP-hard.

下面构造从均分问题到这个问题的 Turing归约.

设 $s[A,S,B,K]$是解第 K 个最大子集的子程序，其中

$A = \{ a_1, a_2, \dots , a_n\}$

$S: A \rightarrow Z^+$

$B \le S(A)$

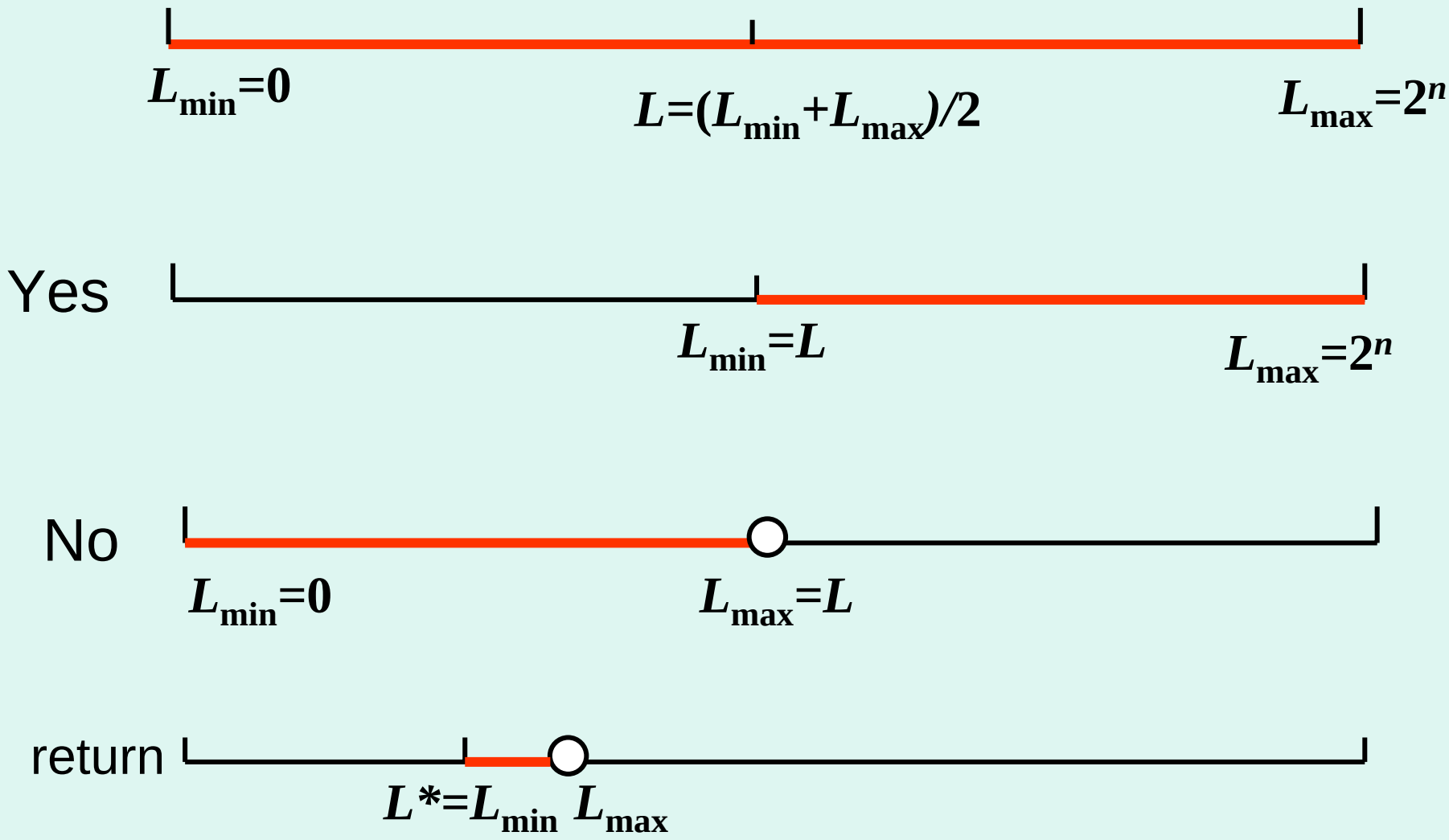
$K \le 2^n$

设 $A = \{ a_1, a_2, \dots , a_n \}$，$S:A \rightarrow Z^+$是均分的实例，如下构造解均分问题的算法.

算法

算法 Partition

1. **if** $S(A)$为奇数 **then return** “No”, 结束；
2. $b \leftarrow S(A)/2$, $Lmin \leftarrow 0$, $Lmax \leftarrow 2^n$;
3. **if** $Lmax - Lmin = 1$ **then** $L^* \leftarrow Lmin$, 转8；
4. $L \leftarrow (Lmax + Lmin)/2$；
5. $s[A, S, b, L]$;
6. **if** 回答 “Yes” **then** $Lmin \leftarrow L$ **else** $Lmax \leftarrow L$；
7. **goto** 3；
8. $s[A, S, b-1, L^*]$;
9. **if** 回答“No” **then return** “Yes” **else return** “No”


$L_{min}=0$
$L=(L_{min}+L_{max})/2$
$L_{max}=2^n$
Yes
$L_{min}=L$
$L_{max}=2^n$
No
$L_{min}=0$
$L_{max}=L$
return
$L^*=L_{min}$ L_{max}

算法说明

步1到步7　二分法找 L^*.

L^* 的含义：

至少 $Lmin$ 个子集其和不超过 $S(A)/2$.

至多 $Lmax-1$ 个子集其和不超过 $S(A)/2$.

恰好有 L^*个子集其和不超过 $S(A)/2$.

步8 检查是否有一个子集其和等于 $S(A)/2$.

如果至少有 L^* 个子集其和 $\le b-1<b$,

那么没有一个子集其和恰好等于 $S(A)/2$.

复杂性估计

步1到步7

二分法查找 L^*

调用 $s[A, S, b, L]$子程序　$\log 2^n = O(n)$次

步8

调用 $s[A, S, b, L]$子程序　1 次

如果 s 是多项式时间的算法，则算法也是多项式时间的.

复杂性类的谱系

NP之外

- **Co-NP**
- **PSPACE**

P之内

- **NC**
- **P**完全

Co-NP

定义

Co-NP = { L| $\overline{L}$∈NP }

Co-NP = { π^c | π∈NP }

HC的补问题

实例：图$G=(V,E)$

问：G 中是否不存在一条**Hamilton**回路？

Open Problem：HC^c∈NP ?

Co-NP完全

定义 $L \in$ **Co-NP**，如果存在多项式变换$\propto$使得$\forall L' \in$ **Co-NP** 都有 $L' \propto L$, 则称 L 是 **Co-NP完全的（Co-NPC）**

结论

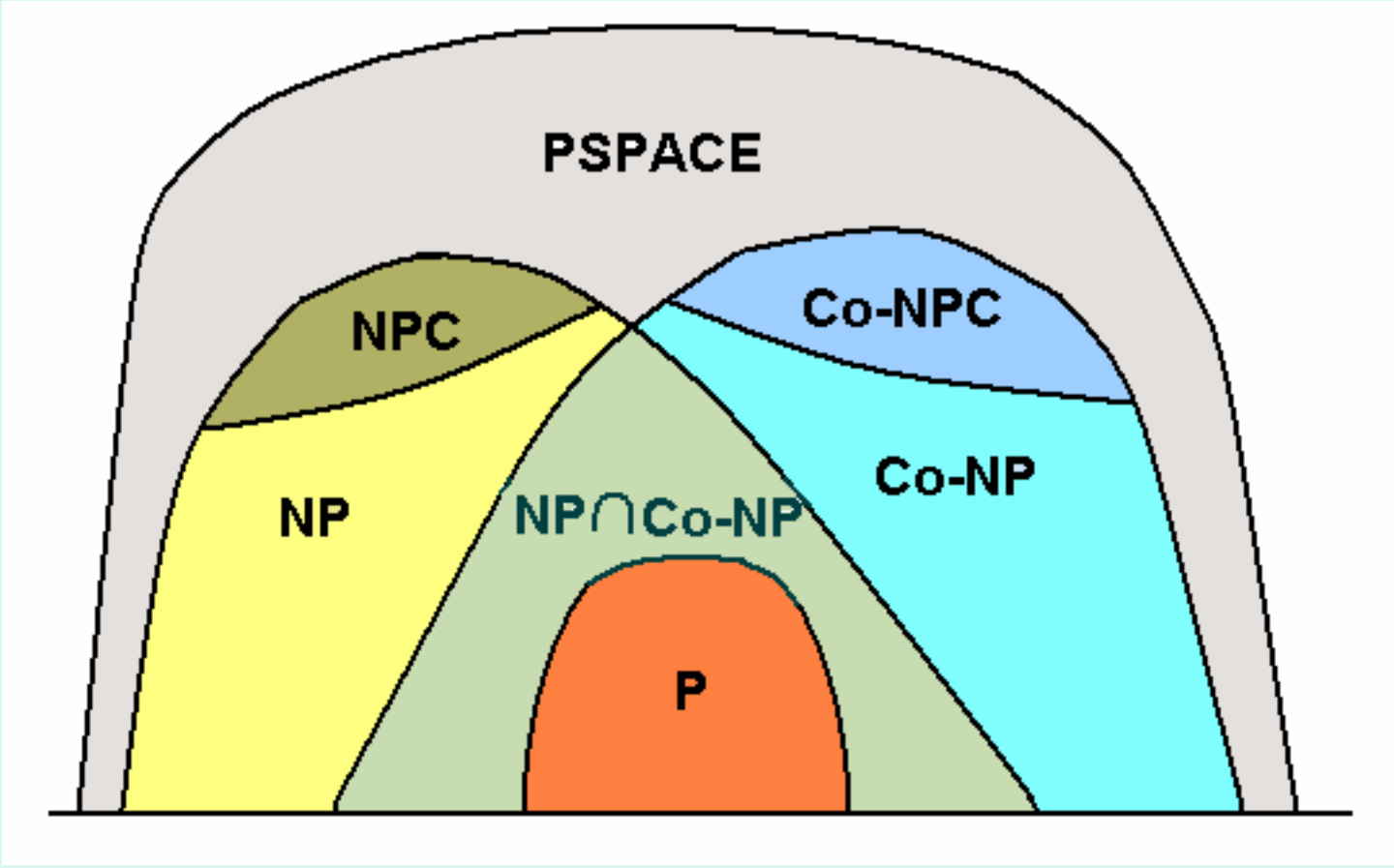
1. $L \propto L' \Rightarrow \overline{L} \propto \overline{L'}$
2. $L \in$ **NPC** $\Rightarrow \overline{L} \in$ **Co-NPC**
3. **P=NP** $\Rightarrow$ **NP = Co-NP**
4. $L \in$ **Co-NP** $\wedge L \in$ **NPC** $\Rightarrow$ **NP = Co-NP**

PSPACE 多项式空间有界的判定问题类

NP$\subseteq$**PSAPCE**

NP与Co-NP的结构

前提：$NP \neq Co\text{-}NP \Rightarrow P \neq NP$



并行计算的PRAM模型

PRAM（Parallel Random Access Machine）

特征：

1. 无限大共享存储器(用于处理器之间交换数据)
2. 有限或无限个(*n*的多项式个)功能相同的处理器
3. 指令集相同：逻辑运算、算术运算、访问内存、输入输出、转移
4. 执行每条指令的时间相同，且与处理器个数无关

并行计算的复杂性分类

定义1 设π是判定问题，使用PRAM并行计算模型，

(1) 如果π可以在多项式时间，用多项式个处理器求解，则称π是并行可解的。

(2) 如果π可以在$\log n$的多项式时间，用多项式个处理器求解，则称π是高度并行可解的。

(3) 如果π是并行可解的，但不是高度并行可解的，则称π是固有顺序的。

并行可解的问题类就是**P类.** 高度并行可解的问题类是P的子类 **NC类**，但至今不知道是否为真子类.

将$(\log n)^k$ 简记为 $\log^k n$

P类：$n^{O(1)}$时间(多项式时间)顺序求解的判定问题类

NC^k类：PRAM并行计算模型, $O(\log^k n)$时间，$n^{O(1)}$个处理器，并行可解的判定问题类

NC类：NC= $\bigcup_{k\in N} NC^k$

Nick Pippenger 最早研究NC类
1978年在多伦多大学访问时提出这个问题
Cook后来称这个类为NC类

$NC \subseteq P$

Open Problem: NC=P ?

NC多一归约

定义2 设判定问题 π_1, π_2, 如果存在函数 f 使得

$$I \in Y\pi_1 \Leftrightarrow f(I) \in Y\pi_2$$

则称 f 是π_1到π_2的多一归约，也称π_1多一归约到π_2，记作 $\pi_1 \alpha_m \pi_2$。注意计算模型为**PRAM**，(当**PRAM**只有一个处理器时就是一般确定型计算机)

如果 f 是π_1到π_2的多一归约，且计算 f 的时间是 $O(\log^{O(1)} n)$，则称π_1**NC-多一归约**到π_2, 记作

$$\pi_1 \alpha_m^{NC} \pi_2$$

P完全性

定义3 设π∈P, 如果对于任意的 π′∈P, 都存在NC多一归约使得 $\pi' \alpha_m^{NC} \pi$ ，则称 π 是P完全的。

定理 如果任何 P 完全的问题属于NC, 则 NC = P.

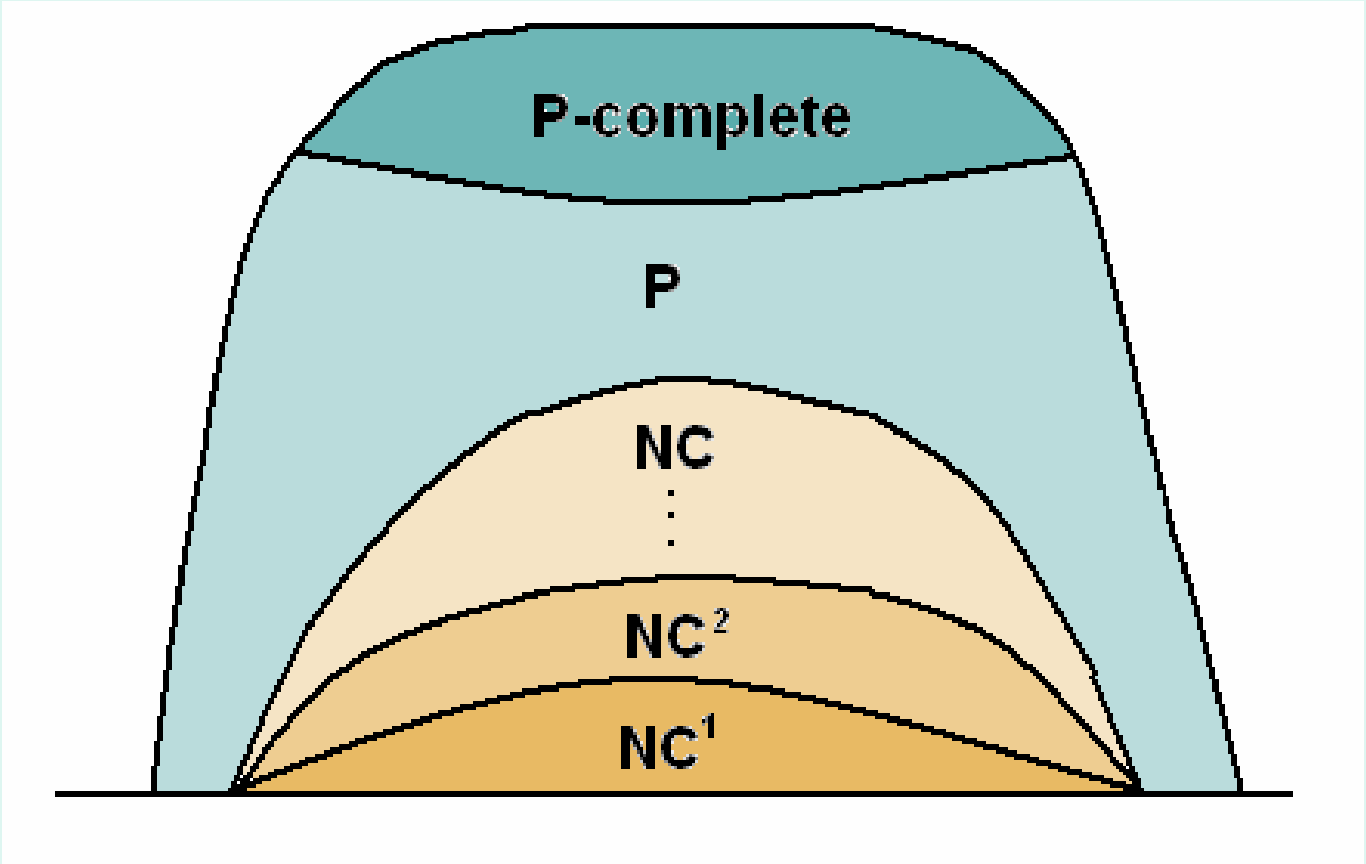
推广到搜索问题——NC图灵归约(调用子程序)

设π是搜索问题，如果存在P完全问题 π’使得 $\pi' \alpha_T^{NC} \pi$ ，则称 π 是P难的（P-hard）.

P内部的结构

$$NC^1 \subseteq NC^2 \subseteq ... \subseteq NC \subseteq P$$

$$P - complete \subseteq P, \quad P = NC ?$$



P完全问题

生成问题 (Generability)

实例：集合X, X上的二元运算 $\bullet$, X的子集T，X中的元素x

问： x是否属于T 的关于 $\bullet$ 运算的闭包？

闭包算法

输入：T

输出：T 关于 $\bullet$ 运算的闭包 **closure**(T)

1. $T' \leftarrow T$
2. **While** $T' \neq T' \cup T' \bullet T'$ **do** $T' = T' \cup T' \bullet T'$;
3. **closure**(T) $\leftarrow T'$.

其中 $T' \bullet T' = \{a \bullet b \mid a, b \in T\}$.

While循环不超过 $|X-T|$ 次，因为每次至少增加**1**个元素，而闭包大小以 $|X|$ 为界。因此生成问题属于**P**。

其他P完全问题

1. 电路值问题（CVP）

   实例：布尔电路由∧，∨，¬ 门构成，一组输入

   问：电路输出是否为1？

2. 深度优先搜索问题（DFS）

   实例：$G=(V,E)$，由邻接表表示，$s,u,v\in V$

   问：在以s为始点的深度优先遍历中，顶点u是否在v之前被访问？

其他P完全问题

3. 最大流问题（MAXFLOW）

实例：给定一个具有整数容量及源结点s, 汇结点 t 的有向网络D, 正整数 f.

问：D中是否存在大于等于f 的流值？

4. 线性不等式问题（LI）

实例：n 阶整数矩阵A, n 维向量 b,

问：是否存在有理向量 X 使得$AX \leq b$?

其他复杂性类

概率复杂性

- 概率图灵机
- 最坏复杂性与期望复杂性
- 概率复杂性类

近似解的复杂性

- 不可近似性、难近似的问题

通讯复杂性

参数化的复杂性

参考书

堵丁柱等，《计算复杂性导论》，高教出版社，2002.
Ingo Wegener，Complexity Theory, 科学出版社，2006.

算法设计与分析

- 一般算法的精细分析技术
  - 与实现的数据结构相结合
  - 均摊的分析技术
- 其他设计与分析技术
  - 在线算法
  - 近似算法
  - 并行算法
  - 分布式算法

考 试 范 围

数学基础：估计函数的阶、递推方程求解、求和技术

算法设计技术：

分治策略、动态规划、贪心法、回溯与分支估界
能进行简单应用

算法分析技术：

对算法进行最坏和平均时间复杂度分析
证明某些简单问题的复杂度的下界

计算复杂性理论

图灵机的定义
理解计算复杂性理论的基本概念和重要结果
简单的 NPC 和 NP-hard 问题的证明方法

考试时间、地点

- 时间：**1月9日上午8:00**
- 地点：二教**405，406，407**
- 答疑时间：**1月8日**，地点：**1622/理一**
  上午 **9:00-11:30**
  下午 **2:00-5:00**
- 小论文提交：**1月10日**

预祝大家考试取得好成绩！