算法分析与设计

屈 婉 玲

qwl@pku.edu.cn

计算思维与人才培养

- 2006年3月周以真(Jeannette M. Wing，卡内基·梅隆大学计算机系系主任)首次提出Coputational Thinking的概念：运用计算机科学的基础概念去求解问题、设计系统和理解人类的行为，它包括了涵盖计算机科学之广度的一系列思维活动。

数学思维与工程思维的互补与融合：抽象与实现

编程 ⇨ 评价程序 优化程序 ⇨ 问题分析 方法确定

技能：会做　　能力：做得好　　思维：认知、方法论

三种基本的思维

- 三种思维的共同特点：
  用语言文字表达、有语法与语义规则、推理逻辑

	实验思维	理论思维	计算思维
起源	物理学	数学	计算机科学
过程步骤	**1.**实验观察归纳建立简单数学公式 **2.**导出数量关系 **3.**实验验证	**1.**定义概念 **2.**提出定理 **3.**给出证明	**1.**建模(约简、嵌入、转化、仿真、…) **2.**抽象与分解,控制系统复杂性 **3.**自动化实现…
特点	解释以往现象 无矛盾 预见新的现象	公理集 可靠协调推演规则 正确性依赖于公理	结论表示有限性 语义确定性 实现机械性

算法与计算思维

- 算法课程是训练计算思维的重要课程；涉及到对问题的抽象，建模，设计好的求解方法，复杂性的控制，...
- 可计算性与计算复杂性： 形式化、确定性、有限性，抽象与逻辑证明
- 算法设计与分析：抽象建模、归纳、正确性证明、效率分析、…

课程简介

课程名称：

算法分析与设计

Design and Analysis of Algorithms

课号：

0A002

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

了解计算复杂性理论的基本概念及其应用

课程主要内容

近似算法；NP 完全理论简介；随机算法	问题处理策略 计算复杂性理论
算法分析与问题的计算复杂性	算法分析方法
分治策略；动态规划；贪心算法；回溯与分支限界	算法设计技术
数学基础、数据结构	基础知识

教 材

书名：

《算法设计与分析》

作者：

屈婉玲, 刘田, 张立昂, 王捍贫

出版社：

清华大学出版社

出版时间：

2011，2013重印

参考书

1. Jon Kleinberg, Eva Tardos, Algorithm Design, Addison-Wesley, 清华大学出版社影印版，2006.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest, Introduction to Algorithms(Second Edtion), The MIT Press 2009.
3. 张立昂，可计算性与计算复杂性导引（第**3**版），北京大学出版社，2011.

4*. 堵丁柱，葛可一，王洁，计算复杂性导论，高教出版社 2002.

5*. Sanjeev Arora, Boaz Barak, Computational Complexity: A Modern Approach, Cambridge University Press，2009.

北京大學

学习安排

教学方式：

以课上讲授为主

视频答疑

书面作业

成绩评定：

平时成绩：50%

期末笔试：50%

视频答疑

http://vclassroom.pku.edu.cn/qwl

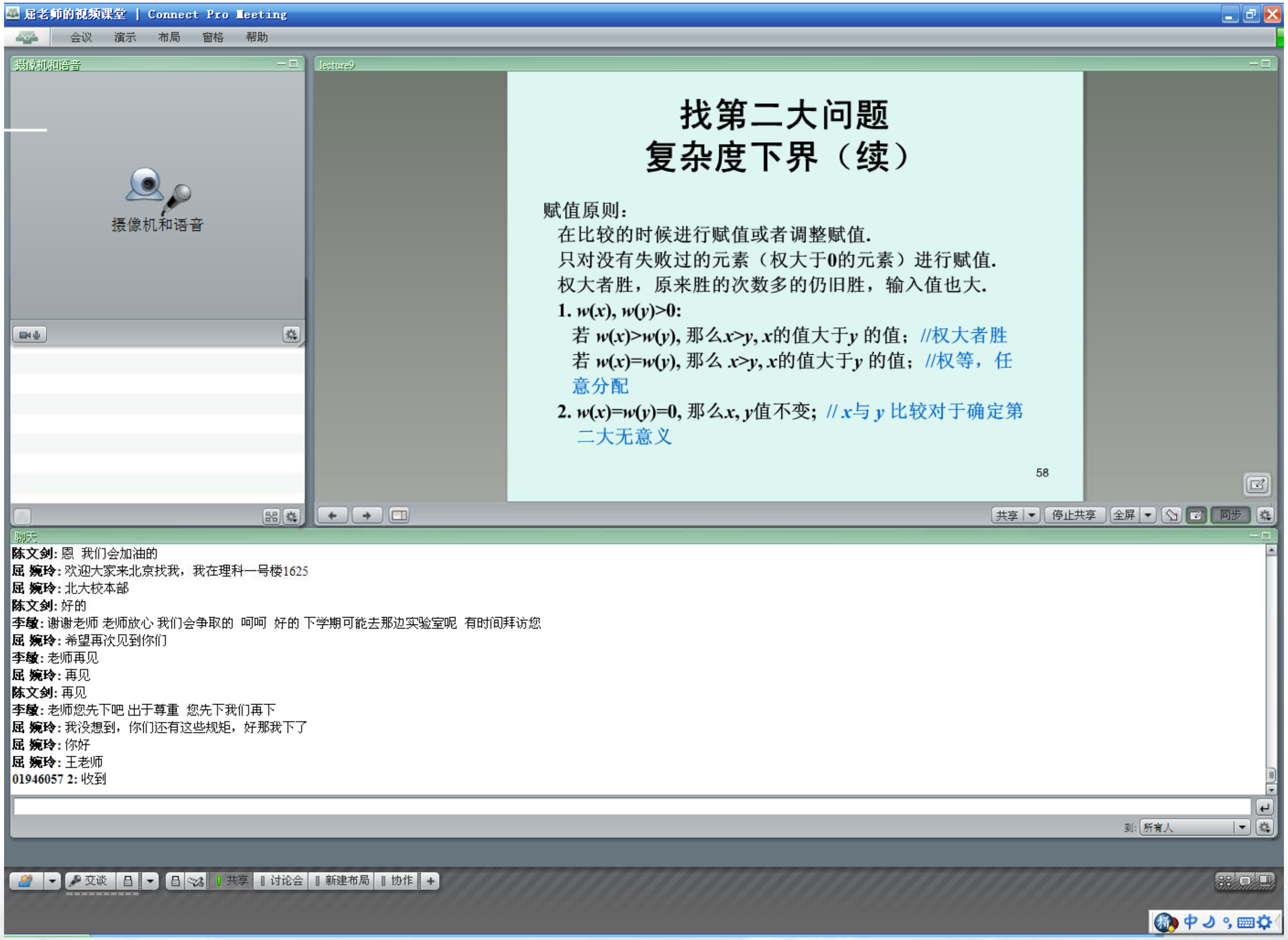
定期安排时间

以客人身份用真实姓名进入

在线答疑：

- 语音
- 打字
- 白板
- 上传文件
- **PPT**播放

视频教室界面


屈老师的视频课堂 | Connect Pro Meeting
会议 演示 布局 窗格 帮助
摄像机和语音
lecture9
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
58
共享 停止共享 全屏 同步
聊天
陈文剑: 恩 我们会加油的
屈 婉玲: 欢迎大家来北京找我，我在理科一号楼1625
屈 婉玲: 北大校本部
陈文剑: 好的
李敏: 谢谢老师 老师放心 我们会争取的 呵呵 好的 下学期可能去那边实验室呢 有时间拜访您
屈 婉玲: 希望再次见到你们
李敏: 老师再见
屈 婉玲: 再见
陈文剑: 再见
李敏: 老师您先下吧 出于尊重 您先下我们再下
屈 婉玲: 我没想到，你们还有这些规矩，好那我下了
屈 婉玲: 你好
屈 婉玲: 王老师
01946057 2: 收到
所有人
交谈 共享 讨论会 新建布局 协作

北京大學

引言 为什么要学算法

- 几个例子
  - 调度问题
  - 排序问题
  - 货郎问题
- 算法研究的内容和目标
  - 算法设计技术
  - 算法分析方法
  - 问题复杂度的界定
  - 计算复杂性理论
- 算法研究的重要性

几个例子

例1：求解调度问题

任务集 $S=\{1, 2, \ldots, n\}$,

第 j 项加工时间：$t_j, \in Z^+, j=1, 2, \ldots, n$

一个可行调度方案：$1, 2, \ldots, n$ 的排列 $i_1, i_2, \ldots, i_n$

求：总等待时间最少的调度方案，即求

S的排列$i_1, i_2, \ldots, i_n$使得 $\min\{\sum_{k=1}^{n}(n-k+1)t_{i_k}\}$

求解方法

贪心策略：加工时间短的先做

如何描述这个方法？这个方法是否对所有的实例都能得到最优解？如何证明？这个方法的效率如何？

例2：投资问题

问题：

m元钱，投资给n个项目，效益函数 $f_i(x)$，表示第 i 个项目投入x元钱的效益，i=1,2,…,n. 如何分配每个项目的钱数使得总效益最大？

令x_i是第 i 个项目的钱数

$$\max \sum_{i=1}^{n} f_i(x_i),$$

$$\sum_{i=1}^{n} x_i = m, \quad x_i \in \mathrm{N}$$

采用什么算法？效率怎样？

蛮力算法的代价

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

非负整数解 $<x_1, x_2, \ldots, x_n>$ 的个数估计：

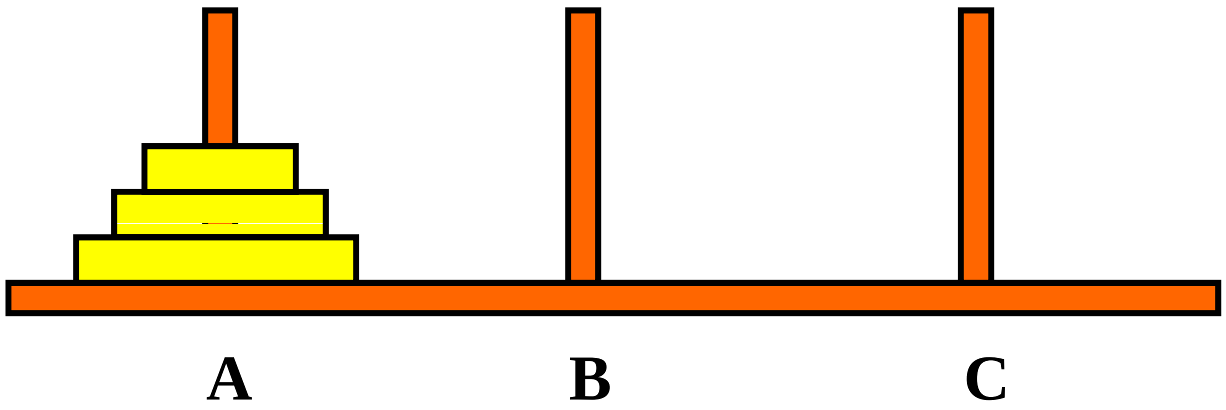
$$C_{m+n-1}^{m}=\frac{(m+n-1)!}{m!(n-1)!}=\Omega((1+\varepsilon)^{m+n-1})$$

蛮力算法——穷举法代价太大

能否利用解之间的依赖关系找到更好的算法?

结论：需要算法设计技术

例3 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），千万亿次/秒，4个多小时.

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

例4 排序算法的评价

已有的排序算法：考察元素比较次数

插入排序、冒泡排序：最坏和平均状况下都为$O(n^2)$

快速排序：最坏状况为$O(n^2)$，平均状况下为$O(n\log n)$

堆排序、二分归并排序：最坏和平均状况下都为$O(n\log n)$

…

问题

那个排序算法效率最高？

是否可以找到更好的算法？排序问题的计算难度如何估计？

例5 货郎问题

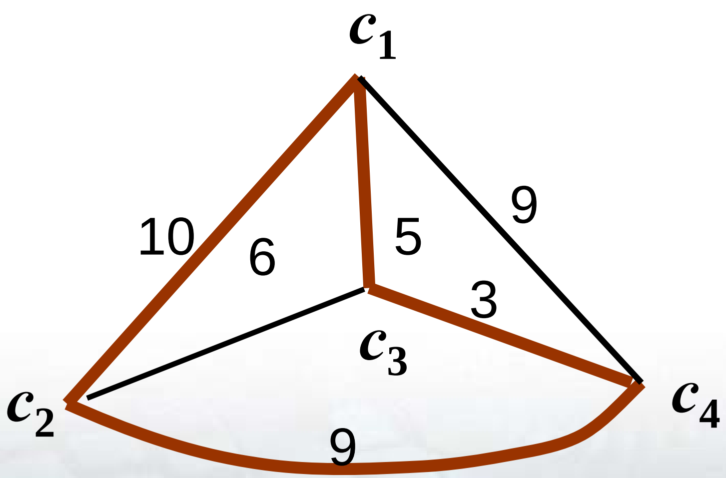
货郎问题：

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

$$d(c_i,c_j) = d(c_j,c_i) \in Z^+,\quad 1 \le i < j \le m$$

求 $1, 2 \ldots, m$ 的排列 $k_1, k_2, \ldots, k_m$ 使得

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



现状：至今没有找到有效的算法，
　　　存在大量问题与它难度等价

问题：是否存在有效算法？
　　　如何处理这类问题？

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→寻找求解算法	算法设计技术
算法→算法的评价	算法分析方法
算法类→问题复杂度的评价	问题复杂性分析
问题类→能够求解的边界	计算复杂性理论

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，即存在求解算法

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

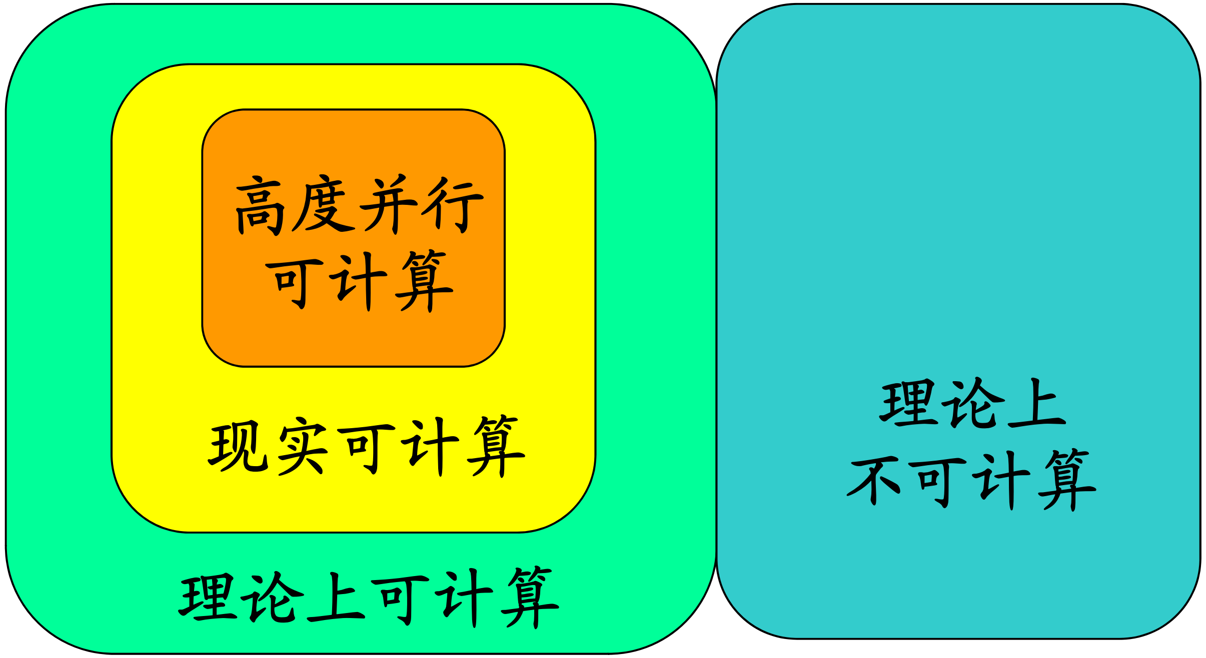
指令执行的过程是确定的

核心论题：**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

理论上与现实上可计算性



算法至少具有指数时间：理论上可计算——难解的

多项式时间的算法：现实上可计算——多项式时间可解的

对数多项式时间的算法：高度并行可解的

多项式时间的算法

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

不是多项式时间的算法

不存在多项式 $p(n)$使得该算法的时间复杂度为 $O(p(n))$

包含指数时间甚至更高阶的算法

多项式函数与指数函数

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

多项式函数与指数函数

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倍的计算机	快1000倍的计算机
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	$N_6 + 4.19$	$N_6 + 6.29$

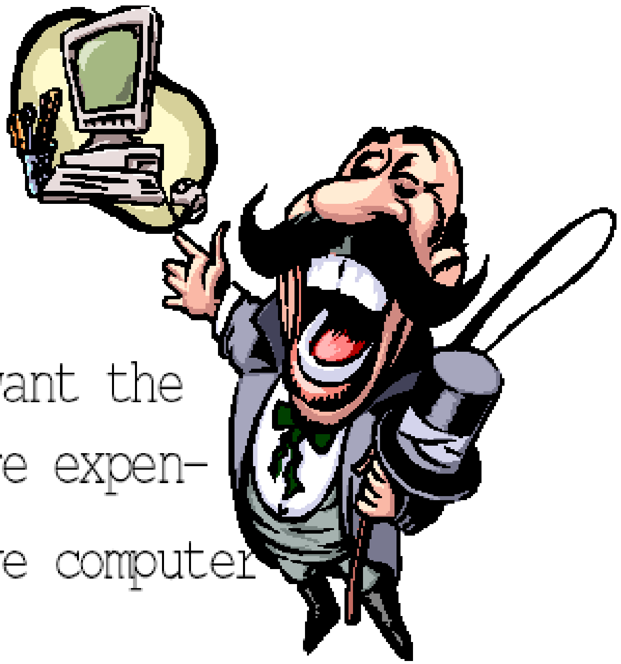
10亿次/秒机器求解的问题

- 快速排序算法给10万个数据排序, 运算量约为 $10^5 \times \log_2 10^5 \approx 1.7\times10^6$, 仅需$1.7\times10^6/10^9=1.7\times10^{-3}$秒.
- Dijkstra算法求解1万个顶点的图的单源最短路径问题, 运算量约为$(10^4)^2=10^8$, 约需$10^8/10^9=0.1$秒.
- 回溯法解100个顶点的图的最大团问题, 运算量为 $100\times2^{100}\approx1.8\times10^{32}$, 需要$1.8\times10^{32}/10^9=1.8\times10^{21}$秒 $=5.7\times10^{15}$年, 即5千7百万亿年!
- 1 分钟能解多大的问题. 1分钟60秒, 这台计算机可做用快速排序算法可给2×10^9(即, 20亿)个数据排序, 用Dijkstra算法可解2.4×10^5个顶点的图的单源最短路径问题. 而用回溯法一天只能解41个顶点的图的最大团问题

北京大学

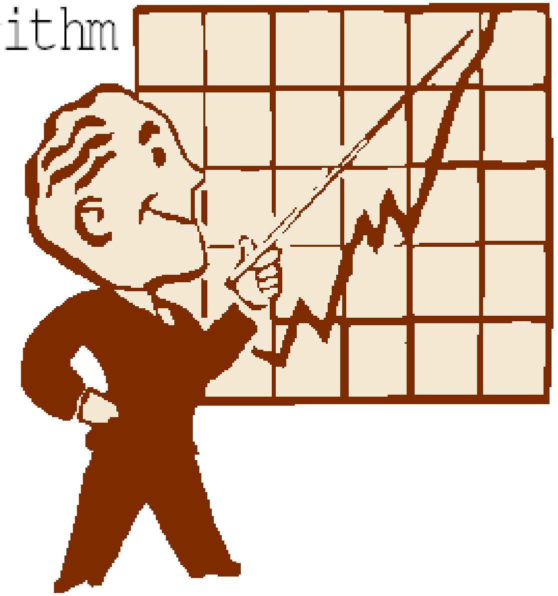
两种选择

I want the more expen-sive computer



rich man

I want the better algorithm



smart mathematician

问题的复杂度分析

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

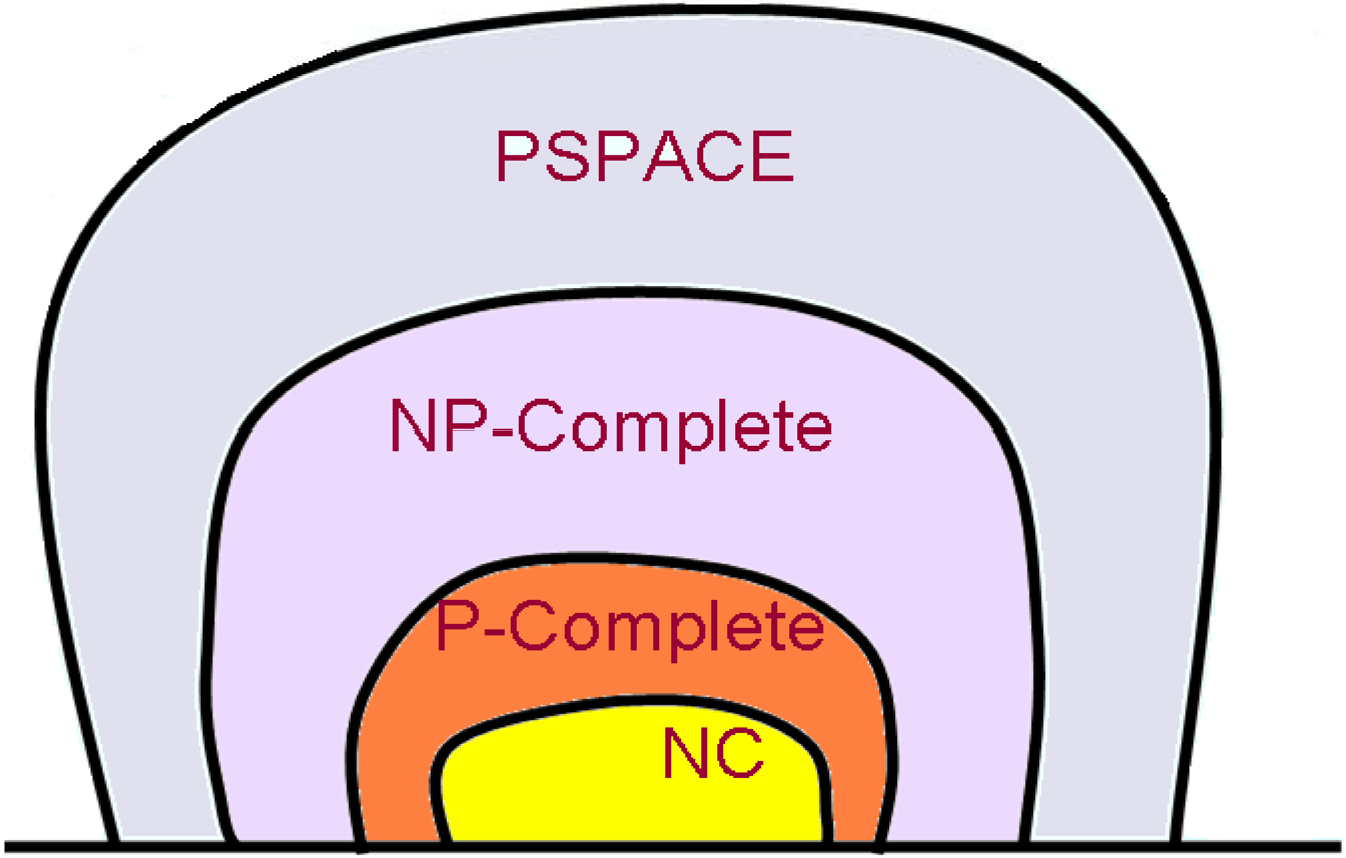
难解的问题***P***

不存在解***P*** 的多项式时间的算法

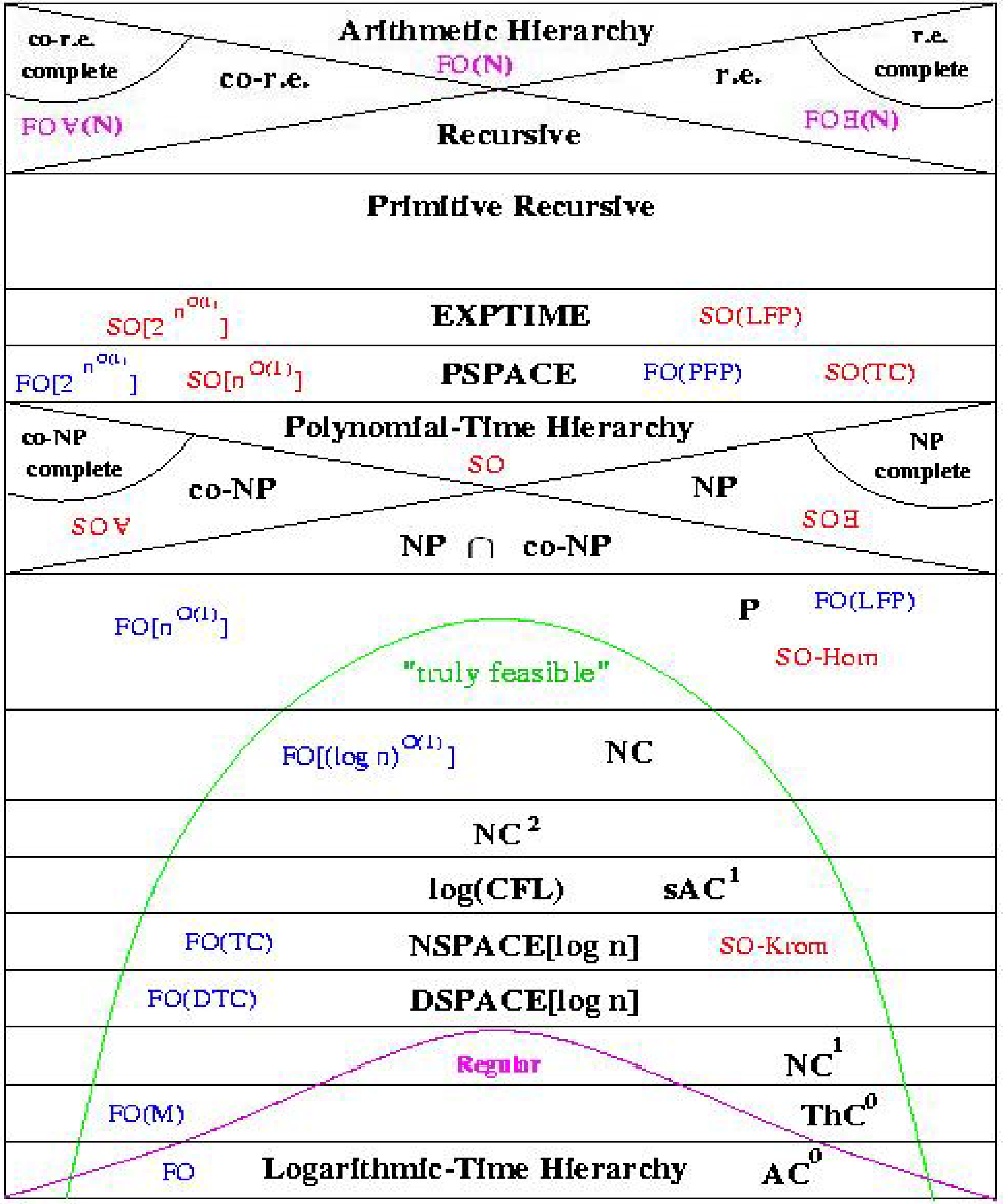
实际上可计算的问题

多项式时间可解的问题

计算复杂性类的谱系



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)
Recursive
FO∃(N)
Primitive Recursive
SO[$2^{n^{O(1)}}$]
EXPTIME
SO(LFP)
FO[$2^{n^{O(1)}}$]
SO[$n^{O(1)}$]
PSPACE
FO(PFP)
SO(TC)
Polynomial-Time Hierarchy
co-NP complete
co-NP
SO
NP
NP complete
SO∀
SO∃
NP ∩ co-NP
FO[$n^{O(1)}$]
P
FO(LFP)
"truly feasible"
SO-Horn
FO[$(\log n)^{O(1)}$]
NC
NC^2
log(CFL)
sAC^1
FO(TC)
NSPACE[log n]
SO-Krom
FO(DTC)
DSPACE[log n]
Regular
NC^1
FO(M)
ThC^0
FO
Logarithmic-Time Hierarchy
AC^0

算法与计算复杂性理论进展

- 算法
  - 概率算法
  - 近似算法
  - 在线算法
  - 分布式算法
- 计算复杂性
  - 概率Turing机与概率复杂性
  - 近似求解的复杂性
  - 参数复杂性
  - 计数复杂性
  - 通信复杂性

算法研究的重要性

- 算法设计与分析技术在计算机科学与技术领域有着重要的应用背景
- 算法设计分析与计算复杂性理论的研究是计算机科学技术的核心研究领域

  1966-2005期间，**Turing**奖获奖**50**人，其中**10**人 以算法设计，**7**人以计算理论、自动机和复杂性研究领域的杰出贡献获奖

  计算复杂性理论的核心课题 “**P=NP?**” 是本世纪 **7**个最重要的数学问题之一

- 通过算法设计与分析课程的训练对提高学生的素质和分析问题解决问题的能力以及计算思维有着重要的作用

第1章 基础知识

1.1 算法的基本概念

问题
算法
算法的时间复杂度

1.2 算法的伪码表示

1.3 数学基础知识

函数的渐近的界
序列求和
递推方程求解

1.1算法的基本概念

问题：需要回答的一般性提问，通常含有若干参数，对参数的一组赋值就得到问题的一个实例.

算法：是有限条指令的序列，这个指令序列确定了解决某个问题的一系列运算或操作.

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.

算法的时间复杂度：针对问题指定基本运算，计数算法所做的基本运算次数

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：在指定输入的概率分布下，算法求解输入规模为n的实例所需要的平均时间 $A(n)$.

检索问题的时间估计

检索问题

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

最坏情况下时间：$W(n)=n$

平均情况：假设 $x\in L$ 的概率为 p，x 在 L 不同位置等概分布.

实例集 S，实例$I\in S$的概率是 p_I, 算法对I 的基本运算次数为 t_I，平均情况下的时间复杂度为

$$A(n)=\sum_{I\in S} t_I p_I$$

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

北京大学

1.2 算法的伪码描述

赋值语句：←

分支语句：if …then … [else…]

循环语句：while, for， repeat until

转向语句：goto

输出语句：return

调用：直接写过程的名字

注释：//…

实例：求最大公约数

算法1.1 Euclid(m,n)

输入：非负整数m,n，其中m与n不全为0

输出：m与n的最大公约数

while $m>0$ **do**

  $r \leftarrow n$ **mod** m

  $n \leftarrow m$

  $m \leftarrow r$

return n

实例：改进的顺序检索

算法1.2 Search(L,x)

输入：数组 $L[1..n]$，其元素按照从小到大排列，数 x.

输出：若 x 在 L中，输出 x 的位置下标 j；否则输出0.

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** $j \leftarrow 0$
4. **return** j

1.3 数学基础

1.3.1 函数的渐近的界

定义1.1 设 f 和g是定义域为自然数集 N上的函数.

(1) 若存在正数 c 和 n_0使得对一切 $n \ge n_0$有$0 \le f(n) \le cg(n)$ 成立, 则称 $f(n)$ 的渐近的上界是 $g(n)$，记作

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

(2) 若存在正数 c 和 n_0，使得对一切 $n \ge n_0$有 $0 \le cg(n) \le f(n)$ 成立, 则称 $f(n)$的渐近的下界是 $g(n)$，记作

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

(3) 若对于任意正数 c 都存在 n_0，使得当 $n \ge n_0$ 时有 $0 \le f(n) < cg(n)$ 成立, 则记作

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

函数的渐近的界（续）

(4) 若对于任意正数 c 都存在 n_0，使得当 $n \geq n_0$ 时有 $0 \leq cg(n) < f(n)$成立, 则记作

$$f(n)=\omega(g(n)).$$

(5) 若 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$, 则记作

$$f(n)=\Theta(g(n)).$$

例 $f(n)=n^2+n$，则

$f(n)=O(n^2)$，$f(n)=O(n^3)$,

$f(n)=o(n^3)$,

$f(n)=\Theta(n^2)$

有关定理

定理1.1 设 f 和 g 是定义域为自然数集合的函数.

(1) 如果 $\lim\limits_{n\to\infty} f(n)/g(n)$ 存在且等于某个常数 $c>0$, 那么

$$f(n)=\Theta(g(n)).$$

(2) 如果 $\lim\limits_{n\to\infty} f(n)/g(n)=0$，那么

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

(3) 如果 $\lim\limits_{n\to\infty} f(n)/g(n)=+\infty$，那么

$$f(n)=\omega(g(n)).$$

证明定理1.1 (1)

证　根据极限定义，对于给定的正数 ε =$c/2$, 存在某个n_0，只要$n \geq n_0$，就有

$$\left|\frac{f(n)}{g(n)} - c\right| < \varepsilon \Rightarrow c - \varepsilon < \frac{f(n)}{g(n)} < c + \varepsilon$$

$$\Rightarrow \frac{c}{2} < \frac{f(n)}{g(n)} < \frac{3c}{2} < 2c$$

对所有的$n \geq n_0$，$f(n) \leq 2cg(n)$. 从而推出 $f(n)=O(g(n))$
对所有的$n \geq n_0$，$f(n) \geq (c/2)g(n)$，从而推出 $f(n)=\Omega(g(n))$,
于是 $f(n)=\Theta(g(n))$

定理1.2-1.3

定理1.2 设 f, g, h是定义域为自然数集合的函数，

(1) 如果 $f=O(g)$ 且 $g=O(h)$，那么 $f=O(h)$.

(2) 如果 $f=\Omega(g)$ 且 $g=\Omega(h)$，那么 $f=\Omega(h)$.

(3) 如果 $f=\Theta(g)$ 和 $g=\Theta(h)$，那么 $f=\Theta(h)$.

定理1.3 假设f和g是定义域为自然数集合的函数，若对某个其它的函数 h, 我们有 $f=O(h)$ 和 $g=O(h)$，那么

$$f+g=O(h).$$

例：函数的阶

例1 设 $f(n) = n^2 - 3n$， 证明 $f(n) = \Theta(n^2)$.

证 因为

$$\lim_{n \to +\infty} \frac{f(n)}{n^2} = \lim_{n \to +\infty} \frac{\frac{1}{2}n^2 - 3n}{n^2} = \frac{1}{2}$$

根据定理1.1有 $f(n) = \Theta(n^2)$.

可以证明：多项式函数，幂函数的阶低于指数函数

$n^d = o(r^n)$，$r>1$，$d>0$

基本函数类

阶的高低

至少指数级：$2^n, 3^n, n!, \ldots$

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

对数多项式级：$\log n, \log^2 n, \ldots$

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

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

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

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

对数函数

符号：

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

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

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

性质：

$$\log_b n = o(n^\alpha) \quad \alpha > 0$$

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

$$\log_k n = \Theta(\log_l n)$$

阶乘

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

$n! = o(n^n)$， $n! = \Omega(2^n)$

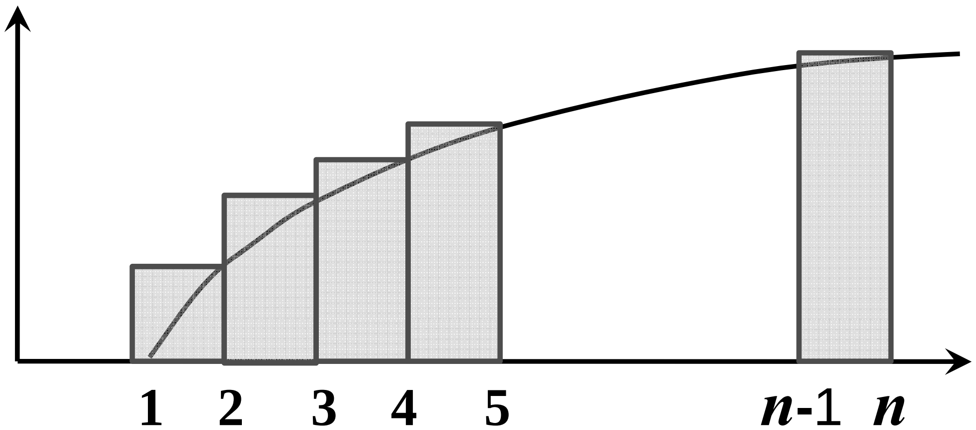
$\log(n!) = \Theta(n\log n)$

$$\log(n!) = \sum_{k=1}^{n} \log k$$

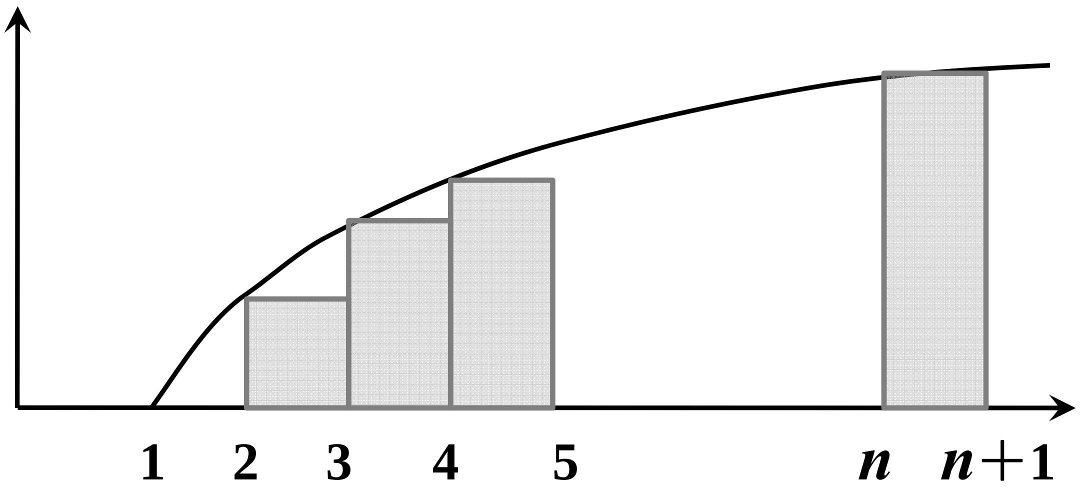
$$\geq \int_1^n \log x dx$$

$$= \log e(n\ln n - n + 1)$$

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



阶乘（续）



$$\log(n!) = \sum_{k=1}^{n} \log k \le \int_{2}^{n+1} \log x dx = O(n \log n)$$

例2：函数的阶

按照阶从高到低对以下函数排序：

$$\log^2 n,\quad 1,\quad n!,\quad n2^n,\quad n^{1/\log n},\quad (3/2)^n,\quad \sqrt{\log n},\quad (\log n)^{\log n},$$

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

结果：

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

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

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

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

函数阶的渐近性质

例3 **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$\Theta(\sqrt{n})$

北京大学

取整函数

$\lfloor x \rfloor$：表示小于等于x的最大的整数

$\lceil x \rceil$：表示大于等于x的最小的整数

取整函数具有下述性质：

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

(2) $\lfloor x+n \rfloor = \lfloor x \rfloor + n$, $\lceil x+n \rceil = \lceil x \rceil + n$，其中$n$为整数

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

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

求和公式

基本求和公式

$$\sum_{k=1}^{n} a_k = \frac{n(a_1 + a_n)}{2}, \quad \{a_k\}\text{为等差数列}$$

$$\sum_{k=0}^{n} aq^k = \frac{a(1-q^{n+1})}{1-q}, \quad \sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x},$$

$$\sum_{k=0}^{\infty} aq^k = \frac{a}{1-q} (q < 1)$$

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

估计和式上界的方法

放大法：

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

2. 假设存在常数 $r<1$，使得 对一切 $k \ge 0$有 $a_{k+1}/a_k \le r$ 成立

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

求和实例

例4 求和

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

(2) $$\sum_{t=1}^{k}t2^{t-1}$$

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

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

求和实例

$$(2)\quad \sum_{t=1}^{k} t2^{t-1} = \sum_{t=1}^{k} t(2^{t} - 2^{t-1})$$

$$= \sum_{t=1}^{k} t2^{t} - \sum_{t=1}^{k} t2^{t-1}$$

$$= \sum_{t=1}^{k} t2^{t} - \sum_{t=0}^{k-1} (t+1)2^{t}$$

$$= \sum_{t=1}^{k} t2^{t} - \sum_{t=0}^{k-1} t2^{t} - \sum_{t=0}^{k-1} 2^{t}$$

$$= k2^{k} - (2^{k} - 1)$$

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

实例

例5 估计$\sum_{k=1}^{n}\frac{k}{3^k}$的上界.

解 由 $a_k=\frac{k}{3^k}$, $a_{k+1}=\frac{k+1}{3^{k+1}}$

得
$$\frac{a_{k+1}}{a_k}=\frac{1}{3}\frac{k+1}{k}\le\frac{2}{3}$$

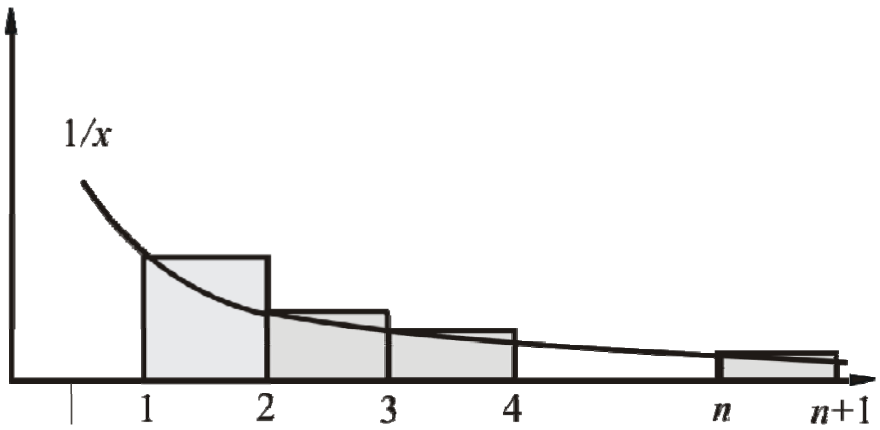
$$\sum_{k=1}^{n}\frac{k}{3^k}\le\sum_{k=1}^{\infty}\frac{1}{3}\left(\frac{2}{3}\right)^{k-1}=\frac{1}{3}\frac{1}{1-\frac{2}{3}}=1$$

估计和式渐近的界

估计 $\sum_{k=1}^{n}\frac{1}{k}$ 的渐近的界.

$$\sum_{k=1}^{n}\frac{1}{k} \ge \int_{1}^{n+1}\frac{dx}{x}$$

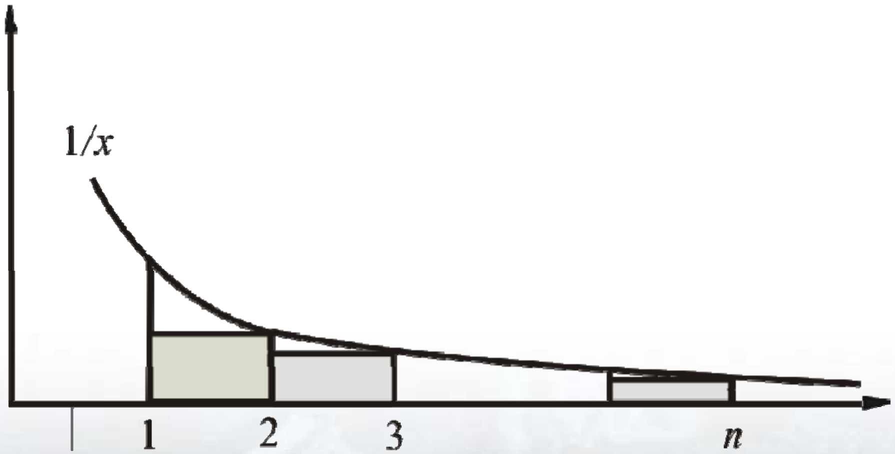
$$= \ln(n+1)$$



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

$$\le 1 + \int_{1}^{n}\frac{dx}{x}$$

$$= \ln n + 1$$



递推方程的求解

设序列$a_0, a_1, \ldots, a_n, \ldots$, 简记为$\{a_n\}$, 一个把$a_n$与某些个$a_i(i<n)$联系起来的等式叫做关于序列 $\{a_n\}$ 的递推方程

求解方法:

迭代法

  直接迭代：插入排序最坏情况下时间分析

  换元迭代：二分归并排序最坏情况下时间分析

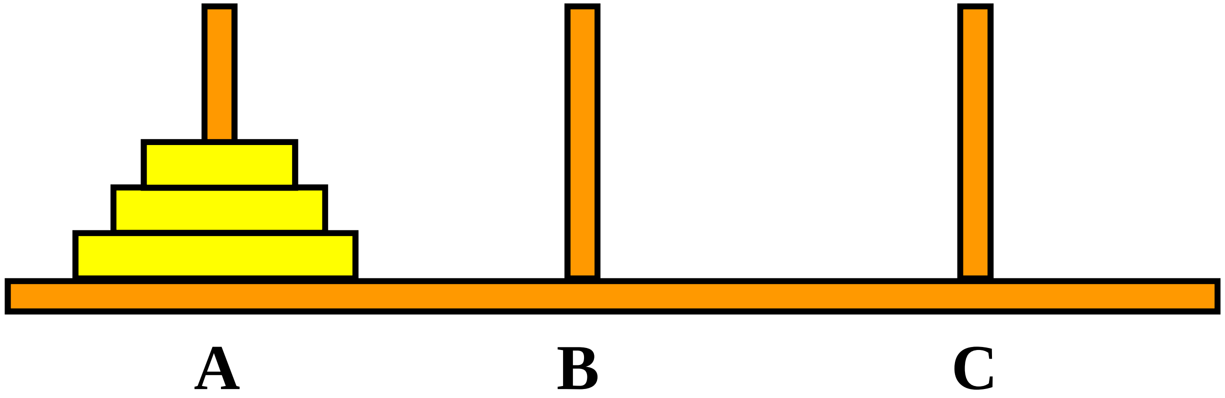
  差消迭代：快速排序平均情况下的时间分析

迭代模型：递归树

尝试法：快速排序平均情况下的时间分析

主定理：递归算法的分析

例6：Hanoi塔



算法1.3 Hanoi(A,C,n) // 将A的n个盘子按要求移到C

1. if n=1 then move (A,C) // 将A的1个盘子移到C
2. else Hanoi(A,B,n–1)
3. move(A,C)
4. Hanoi(B,C,n–1)

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

1秒移1个，64个盘子要多少时间？（5000亿年）

直接迭代：插入排序

算法1.4 InsertSort(A,n) // A为n个数的数组

1. for $j \leftarrow 2$ to n
2. $x \leftarrow A[j]$
3. $i \leftarrow j-1$ // 行3到行7把$A[j]$插入$A[1..j-1]$之中
4. while $i > 0$ and $x < A[i]$ do
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow x$

$$\begin{cases} W(n) = W(n-1) + n - 1 \\ W(1) = 0 \end{cases}$$

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

$=[W(n-2)+n-2]+n-1=W(n-2)+(n-2)+(n-1)$

$=[W(n-3)+n-3]+(n-2)+(n-1=\ldots$

$=W(1)+1+2+\ldots+(n-2)+(n-1)$

$=1+2+\ldots+(n-2)+(n-1)=n(n-1)/2$

二分归并排序

算法1.5 **MergeSort(A,p,r)** // 归并排序数组$A[p..r]$

1. if $p<r$
2. then $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort(A,p,q)
4. MergeSort($A,q+1,r$)
5. Merge(A,p,q,r)

$$\begin{cases} W(n) = 2W(n/2) + n - 1 \\ W(1) = 0 \end{cases}$$

归并过程

算法1.6 Merge(A,p,q,r)　　// 将排序数组$A[p..q]$与$A[q+1,r]$合并

1. $x \leftarrow q-p+1$, $y \leftarrow r-q$　　// x, y分别为两个子数组的元素数
2. 将$A[p..q]$复制到$B[1..x]$, 将$A[q+1..r]$复制到$C[1..y]$
3. $i \leftarrow 1$, $j \leftarrow 1$, $k \leftarrow p$
4. while $i \le x$ and $j \le y$ do
5. 　　if $B[i] \le C[j]$　　// B的首元素不大于C的首元素
6. 　　　　$A[k] \leftarrow B[i]$　　// 将B的首元素放到A中
7. 　　　　$i \leftarrow i+1$
8. 　　else
9. 　　　　$A[k] \leftarrow C[j]$
10. 　　　　$j \leftarrow j+1$
11. 　　$k \leftarrow k+1$
12. if $i > x$ then 将$C[j..y]$复制到$A[k..r]$　　// B已经是空数组
13. else 将$B[i..x]$复制到$A[k..r]$　　// C已经是空数组

换元迭代

$$\begin{cases} W(n) = 2W(n/2) + n - 1, \quad n = 2^k \\ W(1) = 0 \end{cases}$$

$$W(n) = 2W(2^{k-1}) + 2^k - 1$$
$$= 2[2W(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1$$
$$= 2^2 W(2^{k-2}) + 2^k - 2 + 2^k - 1$$
$$= 2^2[2W(2^{k-3}) + 2^{k-2} - 1] + 2^k - 2 + 2^k - 1$$
$$= \ldots$$
$$= 2^k W(1) + k2^k - (2^{k-1} + 2^{k-2} + \ldots + 2 + 1)$$
$$= k2^k - 2^k + 1$$
$$= n\log n - n + 1$$

使用迭代法，对解可以通过数学归纳法验证

北京大学

差消化简后迭代

$$\begin{cases} T(n) = \dfrac{2}{n}\sum_{i=1}^{n-1} T(i) + O(n), & n \ge 2 \\ T(1) = 0 \end{cases}$$

快速排序平均时间分析

$$nT(n) = 2\sum_{i=1}^{n-1} T(i) + cn^2,\quad c\text{为某个常数}$$

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

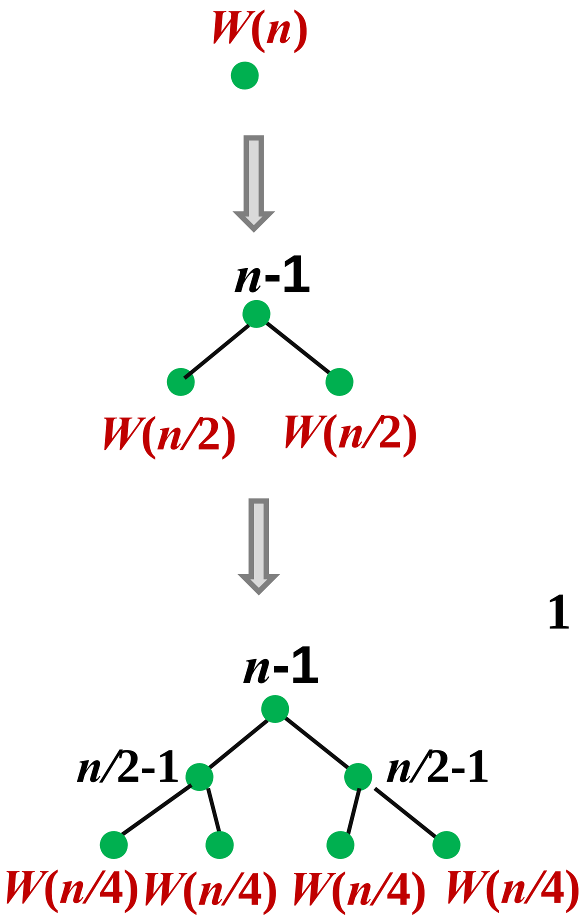
$$nT(n) = (n+1)T(n-1) + O(n)$$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c_1}{n+1} = \cdots = c_1[\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3} + \frac{T(1)}{2}]$$

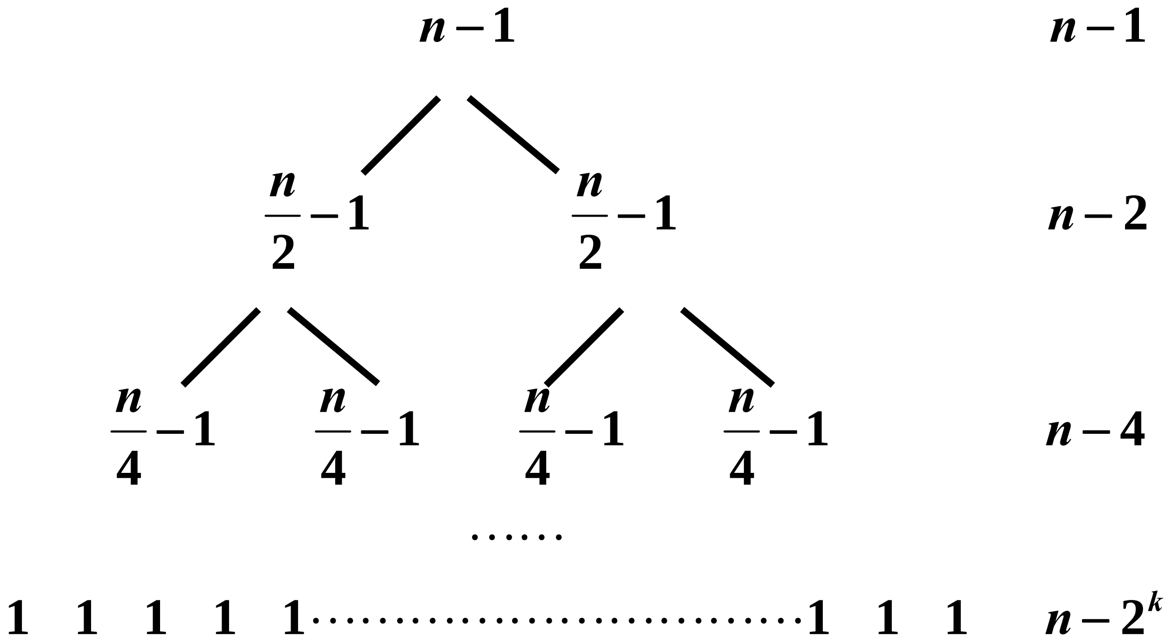
$$= c_1\left[\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right] = \Theta(\log n),\quad T(n) = \Theta(n\log n)$$

北京大学

迭代模型：递归树



生成过程



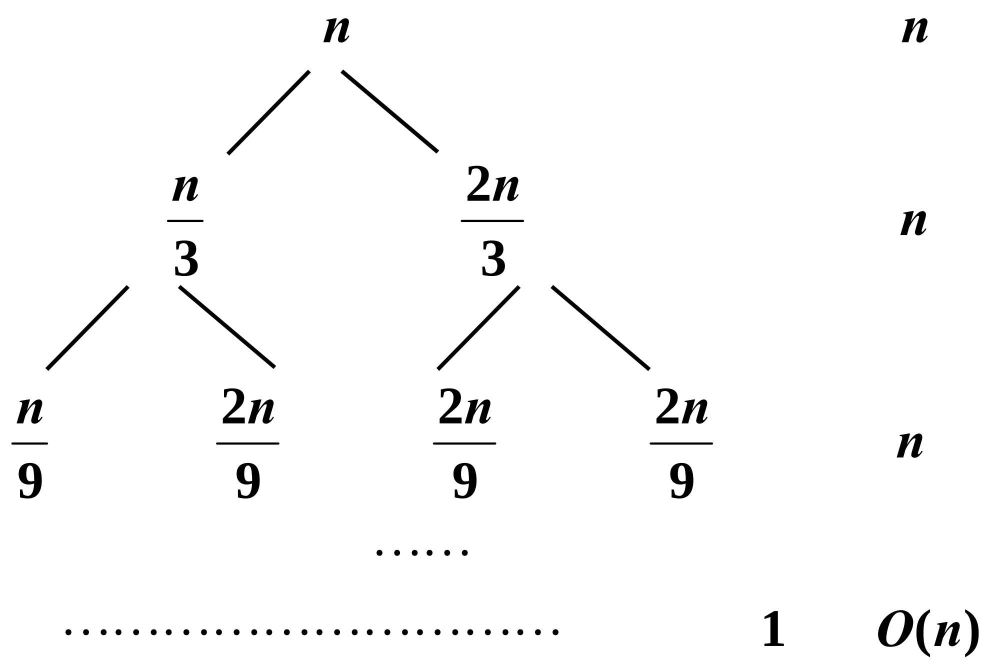
$$W(n)=2W(n/2)+n-1,\ n=2^k,\quad W(1)=0$$

$$W(n)=n-1+n-2+\ldots+n-2^k$$

$$=kn-(2^k-1)=n\log n-n+1$$

递归树的应用实例

$T(n)=T(n/3)+T(2n/3)+n$



层数 k:　　$n(2/3)^k=1 \Rightarrow (3/2)^k=n \Rightarrow k=O(\log_{3/2}n)$

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

尝试法：快速排序

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

(1) $T(n)=C$为常函数，左边$=O(1)$

右边$=\dfrac{2}{n}C(n-1)+O(n)$

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

(2) $T(n)=cn$，左边$=cn$

右边$=\dfrac{2}{n}\sum_{i=1}^{n-1}ci+O(n)$

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

$$=cn-c+O(n)$$

尝试法：快速排序

$$\begin{cases} T(n) = \dfrac{2}{n}\sum\limits_{i=1}^{n-1} T(i) + O(n), & n \ge 2 \\ T(1) = 0 \end{cases}$$

(3) $T(n)=cn^2$，左边$=cn^2$

$$右边= \frac{2}{n}\sum_{i=1}^{n-1} ci^2 + O(n)$$

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

(4) $T(n)=cn\log n$，左边$=cn\log n$

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

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

$$= cn\log n + O(n) + O(\log n)$$

北京大学

以积分作为求和的近似

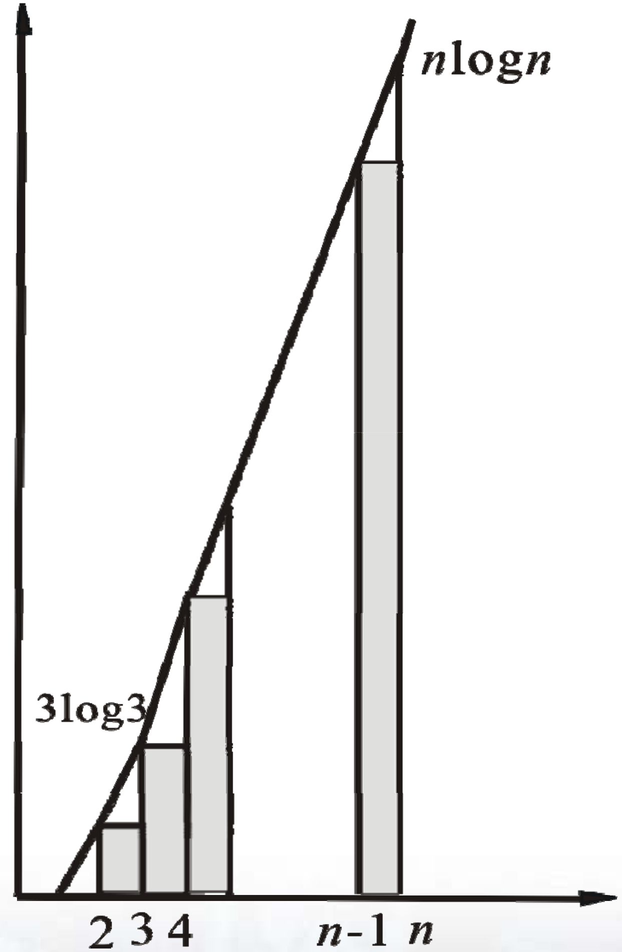
$$\int_2^n x\log x dx = \int_2^n \frac{x}{\ln 2}\ln x dx$$

$$= \frac{1}{\ln 2}\left[\frac{x^2}{2}\ln x - \frac{x^2}{4}\right]\Bigg|_2^n$$

$$= \frac{1}{\ln 2}\left(\frac{n^2}{2}\ln n - \frac{n^2}{4}\right) - \frac{1}{\ln 2}\left(\frac{4}{2}\ln 2 - \frac{4}{4}\right)$$

$$\sum_{i=1}^{n-1} i\log i$$

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



主定理

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

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

则有以下结果：

1. 若$f(n)=O(n^{\log_b a-\varepsilon}), \varepsilon>0$, 那么$T(n)=\Theta(n^{\log_b a})$
2. 若$f(n)=\Theta(n^{\log_b a})$, 那么$T(n)=\Theta(n^{\log_b a}\log n)$
3. 若$f(n)=\Omega(n^{\log_b a+\varepsilon}), \varepsilon>0$, 且对于某个常数$c<1$和充分大的$n$有 $af(n/b) \leq cf(n)$，那么$T(n)=\Theta(f(n))$

迭代

设$n=b^k$

$$T(n)=aT(\frac{n}{b})+f(n)$$

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

$$=\ldots$$

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

$$=a^kT(1)+\sum_{j=0}^{k-1}a^jf(\frac{n}{b^j})$$

$$=c_1n^{\log_b a}+\sum_{j=0}^{k-1}a^jf(\frac{n}{b^j}) \qquad T(1)=c_1$$

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

$$T(n)=c_1n^{\log_b a}+\sum_{j=0}^{k-1}a^j f(\frac{n}{b^j})$$

$$=c_1n^{\log_b a}+O(\sum_{j=0}^{\log_b n-1}a^j(\frac{n}{b^j})^{\log_b a-\varepsilon})$$

$$=c_1n^{\log_b a}+O(n^{\log_b a-\varepsilon}\sum_{j=0}^{\log_b n-1}\frac{a^j}{(b^{\log_b a-\varepsilon})^j})$$

$$=c_1n^{\log_b a}+O(n^{\log_b a-\varepsilon}\sum_{j=0}^{\log_b n-1}(b^{\varepsilon})^j)$$

$$=c_1n^{\log_b a}+O(n^{\log_b a-\varepsilon}\frac{b^{\varepsilon\log_b n}-1}{b^{\varepsilon}-1})$$

$$=c_1n^{\log_b a}+O(n^{\log_b a-\varepsilon}n^{\varepsilon})=O(n^{\log_b a})$$

情况2 $f(n)=\Theta(n^{\log_b a})$

$$T(n)=c_1 n^{\log_b a}+\sum_{j=0}^{k-1} a^j f(\frac{n}{b^j})$$

$$=c_1 n^{\log_b a}+\Theta(\sum_{j=0}^{\log_b n-1} a^j (\frac{n}{b^j})^{\log_b a})$$

$$=c_1 n^{\log_b a}+\Theta(n^{\log_b a}\sum_{j=0}^{\log_b n-1} \frac{a^j}{a^j})$$

$$=c_1 n^{\log_b a}+\Theta(n^{\log_b a}\log n)$$

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

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

$$T(n)=c_1 n^{\log_b a}+\sum_{j=0}^{k-1}a^j f(\frac{n}{b^j})$$

$$\le c_1 n^{\log_b a}+\sum_{j=0}^{\log_b n-1}c^j f(n) \qquad (af(\frac{n}{b})\le cf(n))$$

$$=c_1 n^{\log_b a}+f(n)\frac{c^{\log_b n}-1}{c-1}$$

$$=c_1 n^{\log_b a}+\Theta(f(n)) \qquad （c<1）$$

$$=\Theta(f(n)) \qquad (f(n)=\Omega(n^{\log_b a+\varepsilon}))$$

主定理的应用

例7 求解递推方程 $T(n)=9T(n/3)+n$

解 上述递推方程中的 $a=9, b=3, f(n)=n$，那么

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

相当于主定理的第一种情况，其中ε=1. 根据定理得到

$$T(n)=\Theta(n^2)$$

例8 求解递推方程 $T(n)=T(2n/3)+1$

解 上述递推方程中的 a=1, b=3/2, $f(n)$=1，那么

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

相当于主定理的第二种情况. 根据定理得到 .

$$T(n)=\Theta(n^0\log n)=\Theta(\log n)$$

实例

例9 求解递推方程

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

解 上述递推方程中的a=3, b=4, $f(n)$=$n$log$n$，那么

$$n\log n=\Omega(n^{\log_4 3+\varepsilon})=\Omega(n^{0.793+\varepsilon})，\ \varepsilon\approx 0.2$$

此外，要使 $a f(n/b)\le cf(n)$成立，代入 $f(n)$=$n$log$n$，得到

$$\frac{3n}{4}\log\frac{n}{4}\le cn\log n$$

显然只要 $c\ge 3/4$，上述不等式就可以对充分大的n成立.

相当于主定理的第三种情况. 因此有

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

$n^{\log_2 2} = n$

不能使用主定理的例子

例10 求解 $T(n) = 2T(n/2) + n\log n$

$a=b=2$, $n^{\log_2 2} = n$ ， $f(n)=n\log n$

不存在$\varepsilon > 0$ 使 $n\log n = \Omega(n^{1+\varepsilon})$， 不存在$c<1$使$2(n/2)f(n) \le cf(n)$

$$
\begin{array}{ccccccc|c}
 & & & n\log n & & & & n\log n \\
 & \frac{n(\log n-1)}{2} & & & & \frac{n(\log n-1)}{2} & & n(\log n-1) \\
\frac{n(\log n-2)}{4} & & \frac{n(\log n-2)}{4} & & \frac{n(\log n-2)}{4} & & \frac{n(\log n-2)}{4} & n(\log n-2)
\end{array}
$$

....

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

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

$$= n\log^2 n - nk(k-1)/2 = O(n\log^2 n)$$

北京大學

第2章 分治策略
（Divide and Conquer）

2.1 分治策略的基本思想

- 2.1.1 两个熟悉的例子
- 2.1.2 分治算法的一般性描述

2.2 分治算法的分析技术

2.3 改进分治算法的途径

- 2.3.1 通过代数变换减少子问题个数
- 2.3.2 利用预处理减少递归内部的计算量

2.4 典型实例

- 2.4.1 快速排序算法
- 2.4.2 选择问题
- 2.4.3 $n-1$次多项式在全体$2n$次方根上的求值

北京大學

2.1.1 两个熟悉的例子

二分检索

算法2.1 BinarySearch(T, l, r, x)

输入：数组T，下标从 l 到 r；数 x

输出：j // 如果 x 在 T 中，j为下标；否则为**0**

1. $l \leftarrow 1$; $r \leftarrow n$
2. **while** $l \leq r$ **do**
3. $\quad m \leftarrow \lfloor (l+r)/2 \rfloor$
4. $\quad$ **if** $T[m]=x$ **then return** m $\quad$ // x恰好等于中位元素
5. $\quad$ **else if** $T[m]>m$ **then** $r \leftarrow m-1$
6. $\quad\quad$ **else** $l \leftarrow m+1$
7. **return 0**

二分归并排序 **MergeSort**（见第1章）

时间复杂度分析

二分检索最坏情况下时间复杂度$W(n)$满足

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

可以解出$W(n)=\lfloor \log n \rfloor + 1$.

二分归并排序最坏情况下时间复杂度$W(n)$满足

$$\begin{cases} W(n) = 2W(\frac{n}{2}) + n - 1 \\ W(1) = 0 \end{cases}$$

可以解出$W(n)=n\log n - n + 1$.

2.1.2 分治算法的一般性描述

分治算法 **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. $\quad y_i$ = **Divide-and-Conquer**(P_i)
5. **Return Merge**($y_1, y_2, \ldots, y_k$)

算法时间复杂度的递推方程

$$\begin{cases} W(n) = W(|P_1|) + W(|P_2|) + \ldots + W(|P_k|) + f(n) \\ W(c) = C \end{cases}$$

一般原则：子问题均匀划分、递归处理

2.2 分治算法的分析技术

分治策略的算法分析工具：递推方程

两类递推方程

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

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

求解方法

第一类方程：迭代法、换元法、递归树、尝试法

第二类方程：迭代法、递归树、主定理

递推方程的解

方程 $T(n) = aT(n/b) + d(n)$

$d(n)$为常数

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

$d(n) = cn$

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

北京大学

例2.1 芯片测试

条件：有 n 片芯片，（好芯片至少比坏芯片多1片).
问题：使用最少测试次数，从中挑出1片好芯片.

对芯片 A 与 B 测试，结果分析如下：

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

算法思想：两两一组测试，淘汰后芯片进入下一轮.
如果测试结果是情况1，那么 A、B 中留1片，丢1片；
如果是后三种情况，则把 A 和 B 全部丢掉.

分治算法

命题2.1 当n是偶数时，在上述规则下，经过一轮淘汰，剩下的好芯片比坏芯片至少多1片.

证　设A与B都是好芯片有 i 组，A与B一好一坏有 j 组，A与B都坏有 k 组，淘汰后，好芯片数 i，坏芯片数 k

$$2i + 2j + 2k = n$$

$$2i+j > 2k+j \Rightarrow i > k$$

注：当n是奇数时，用其他芯片测试轮空芯片，如果轮空芯片是好的，算法结束；否则淘汰轮空芯片.

每轮淘汰后，芯片数至少减半，时间复杂度是：

$$\begin{cases} W(n) = W(\frac{n}{2}) + O(n) \\ W(3) = 1 \end{cases} \Rightarrow \quad W(n) = O(n)$$

伪码描述

算法2.3 Test(n)

1．$k \leftarrow n$

2．while k >3 do

3．　将芯片分成$\lfloor k/2 \rfloor$组　// 如有轮空芯片，特殊处理

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

　　　if 2片好 then，则任取1片留下

　　　else 2片同时丢掉

5．　$k \leftarrow$剩下的芯片数

6．if k =3

　then 任取2片芯片测试

　　if 1好1坏 then 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

例2.2 幂乘计算

问题：设a是给定实数，计算 a^n，n为自然数

传统算法： $\Theta(n)$

分治法

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

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

计算 Fibonacci 数

Fibonacci 数

1, 1, 2, 3, 5, 8, 13, 21, …

满足 $F_n = F_{n-1} + F_{n-2}$

增加F_0=0，得到数列　0, **1, 1, 2, 3, 5, 8, 13, 21, …**

通常算法：从 $F_0, F_1, \ldots$, 根据定义陆续相加，时间为$\Theta(n)$

定理2.1 设 $\{F_n\}$为 Fibonacci 数构成的数列，那么

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

算法：令矩阵 $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$，用分治法计算 M^n

时间 $T(n) = \Theta(\log n)$.

北京大学

2.3 改进分治算法的途径

2.3.1 通过代数变换 减少子问题个数

例2. 3 位乘问题

设X ,Y 是 n 位二进制数， $n = 2^k$, 求 XY.

一般分治法 令$X = A2^{n/2} + B$, $Y = C2^{n/2} + D$.

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

$$W(n) = 4W(n/2) + cn，\ W(1) = 1$$

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

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

$$W(n) = 3\, W(n/2) + cn，\ W(1) = 1$$

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

矩阵乘法

例2.4　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

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

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

递推方程 $W(n) = 8\,W(n/2) + cn^2$

$W(1) = 1$

解　$W(n) = O(n^3)$.

Strassen 矩阵乘法

变换方法：

$M_1 = A_{11}(B_{12} - B_{22})$
$M_2 = (A_{11} + A_{12})B_{22}$
$M_3 = (A_{21} + A_{22})B_{11}$
$M_4 = A_{22}(B_{21} - B_{11})$
$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$
$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$
$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$

$C_{11} = M_5 + M_4 - M_2 + M_6$
$C_{12} = M_1 + M_2$
$C_{21} = M_3 + M_4$
$C_{22} = M_5 + M_1 - M_3 - M_7$

时间复杂度：

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

$$W(1) = 1$$

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

$$= O(n^{2.8075})$$

2.3.2 利用预处理减少递归操作

算法中的处理尽可能提到递归外面作为预处理

例2.5 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，比较最少需$O(n^2)$时间

分治策略：子集P 中的点划分成两个子 集 P_L和 P_R

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

平面最邻近点对算法

MinDistance(P,X,Y)

输入：n 个点的集合P, X 和Y 分别为横、纵坐标数组

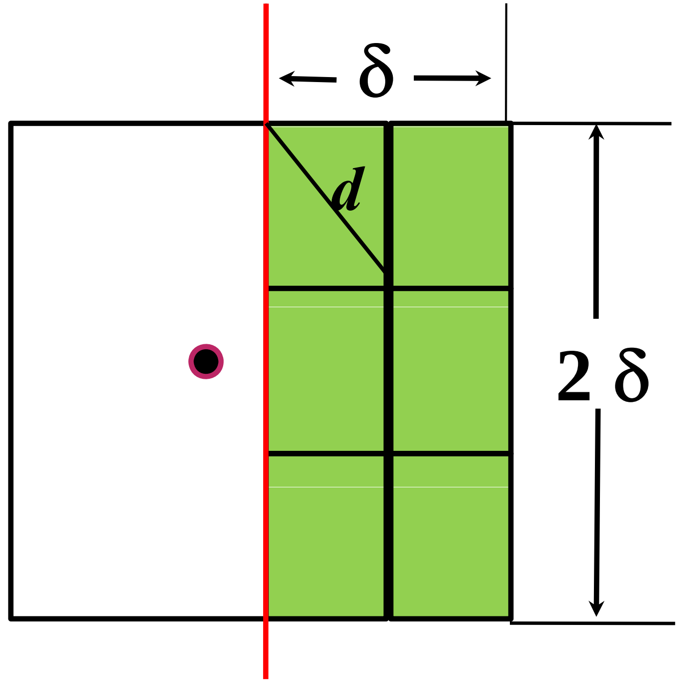
输出：最近的两个点及距离

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); δ_L = P_L中的最小距离
5. MinDistance(P_R,X_R,Y_R); δ_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

北京大学

跨边界的最邻近点

$$d=\sqrt{(\delta/2)^2+(2\delta/3)^2}$$
$$=\sqrt{\delta^2/4+4\delta^2/9}$$
$$=\sqrt{25\delta^2/36}=5\delta/6$$



右边每个小方格至多1个点，每个点至多比较对面的6个点，

距离≤δ的2个点(左右各1个)其纵坐标位置相差不超过12，

检查1个点是常数时间，$O(n)$ 个点需要$O(n)$时间

算法分析

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$O(n)$

$W(n)$总时间，$T(n)$算法递归过程，$O(n\log n)$预处理排序

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

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

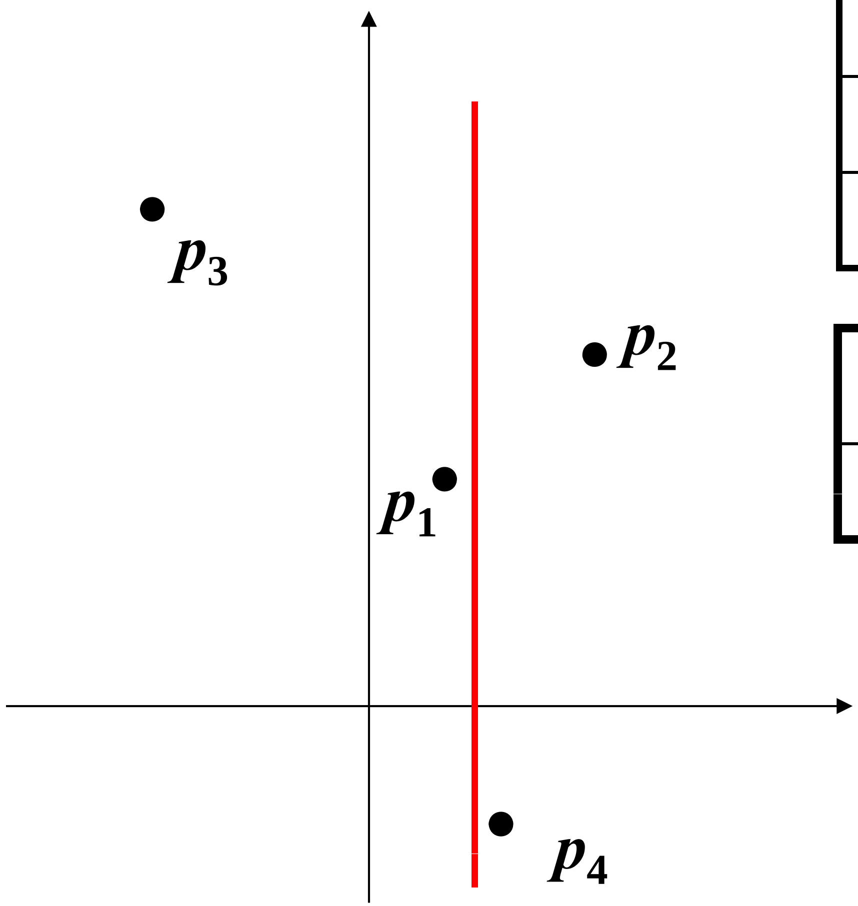
$$T(n) = O(1) \qquad n \le 3$$

解得

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

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

实例：递归中的拆分



P	1	2	3	4
X	0.5	2	-2	1
Y	2	3	4	-1

X	-2(3)	0.5(1)	1(4)	2(2)
Y	-1(4)	2(1)	3(2)	4(3)

X_L	-2(3)	0.5(1)
X_R	1(4)	2(2)
Y_L	2(1)	4(3)
Y_R	-1(4)	3(2)

北京大学

典型实例分析

2.4.1 快速排序

算法 Quicksort(A,p,r)

输入：数组$A[p..r]$

输出：排好序的数组 A

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$,r$)

初始置p=1, r=n，然后调用上述算法

划分过程

Partition(A,p,r)

输入：数组$A[p,r]$

输出：j，A的首元素在排好序的数组中的位置

$x \leftarrow A[p]$

$i \leftarrow p$

$j \leftarrow r+1$

while true do

    repeat $j \leftarrow j-1$

    until $A[j] \le x$

    repeat $i \leftarrow i+1$

    until $A[i] \ge x$

    if $i < j$

    then $A[i] \leftrightarrow A[j]$

    else return j

划分实例

27	99	0	8	13	64	86	16	7	10	88	25	90
	i										*j*	
27	25	0	8	13	64	86	16	7	10	88	99	90
					i				*j*			
27	25	0	8	13	10	86	16	7	64	88	99	90
						i		*j*				
27	25	0	8	13	10	7	16	86	64	88	99	90
							j	*i*				
16	25	0	8	13	10	7	27	86	64	88	99	90

最坏情况下的时间复杂度

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

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

$$T(1) = 0$$

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

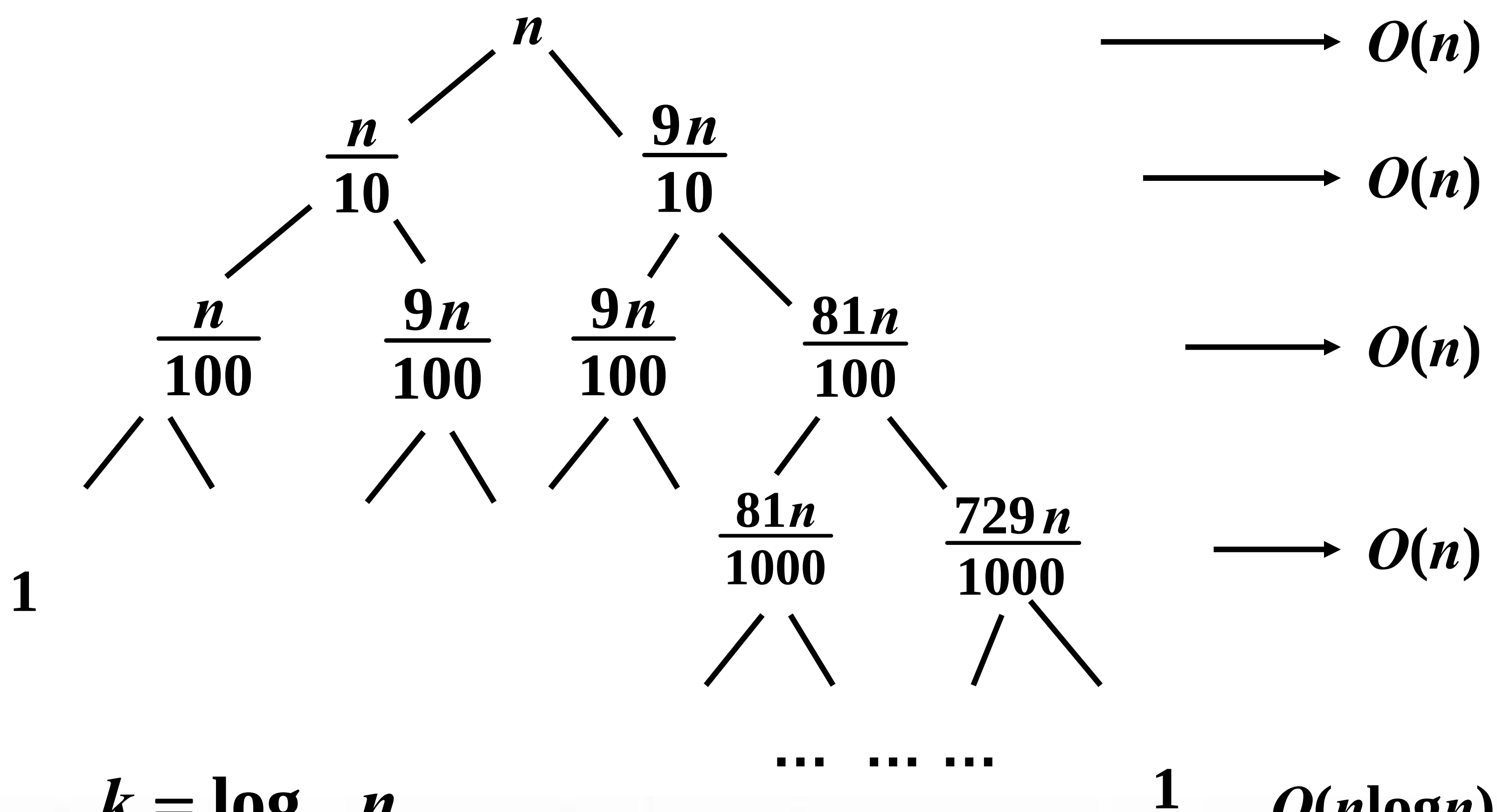
均衡划分

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

$$T(1) = 0$$

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

均衡划分



$$k = \log_{\frac{10}{9}} n$$

$O(n\log n)$

平均情况下时间复杂度

假设输入数组首元素排好序后的正确位置处在**1,2,…,*n*** 各种情况是等可能的，概率为**1/*n*.**

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

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

$$T(1) = 0$$

利用差消法求得 $T(n)=O(n\log n)$

2.4.2 选择问题

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，$i=n/2,\ n/2+1$.

选最大

算法 Findmax

输入：n 个数的数组L

输出：max

1. $max \leftarrow L[1]$;
2. for $i \leftarrow 2$ to n do
3. if $max < L[i]$
4. then $max \leftarrow L[i]$
5. k$\leftarrow i$
5. return max

算法最坏情况下的时间复杂度 $W(n)=n-1$

选最大和最小

通常算法：顺序比较

复杂性：$W(n)=2n-3$

算法 FindMaxMin

输入：n个数的数组L

输出：max，min

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，

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

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：$W(n)=n-1+n-2=2n-3$

锦标赛算法：

两两分组比较，大者进入下一轮

每个元素用数表记录每次比较时小于自己的元素

锦标赛算法

算法 FindSecond

输入：n个数的数组L

输出：*Second*

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．$max \leftarrow$最大数

9．$second \leftarrow max$ 的链表中的最大

时间复杂度分析

命题2.2 *max*在第一阶段的分组比较中总计进行了$\lceil \log n \rceil$次比较.

证 设本轮参与比较的有 t 个元素，经过分组淘汰后进入下一轮的元素数至多是$\lceil t/2 \rceil$. 假设 k 轮淘汰后只剩下一个元素 *max*，利用

$$\lceil \lceil t/2 \rceil /2 \rceil = \lceil t/2^2 \rceil$$

的结果并对 k 归纳，可得到 $\lceil n/2^k \rceil=1$.

若 $n=2^d$，那么有 $k=d=\log n=\lceil \log n \rceil$

若 $2^d<n<2^{d+1}$，那么 $k=d+1=\lceil \log n \rceil$

算法时间复杂度是

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

北京大学

一般性选择问题

问题：选第 k 小.

输入：数组 S, S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：$O(n\log n)$

分治选择算法

算法 Select(S,k)

输入：数组 S，正整数 k

输出：S中的第 k 小元素

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划分成A,B,C,D四个集合

4．把A和D的每个元素与m^*比较，小的构成S_1, 大的构成S_2

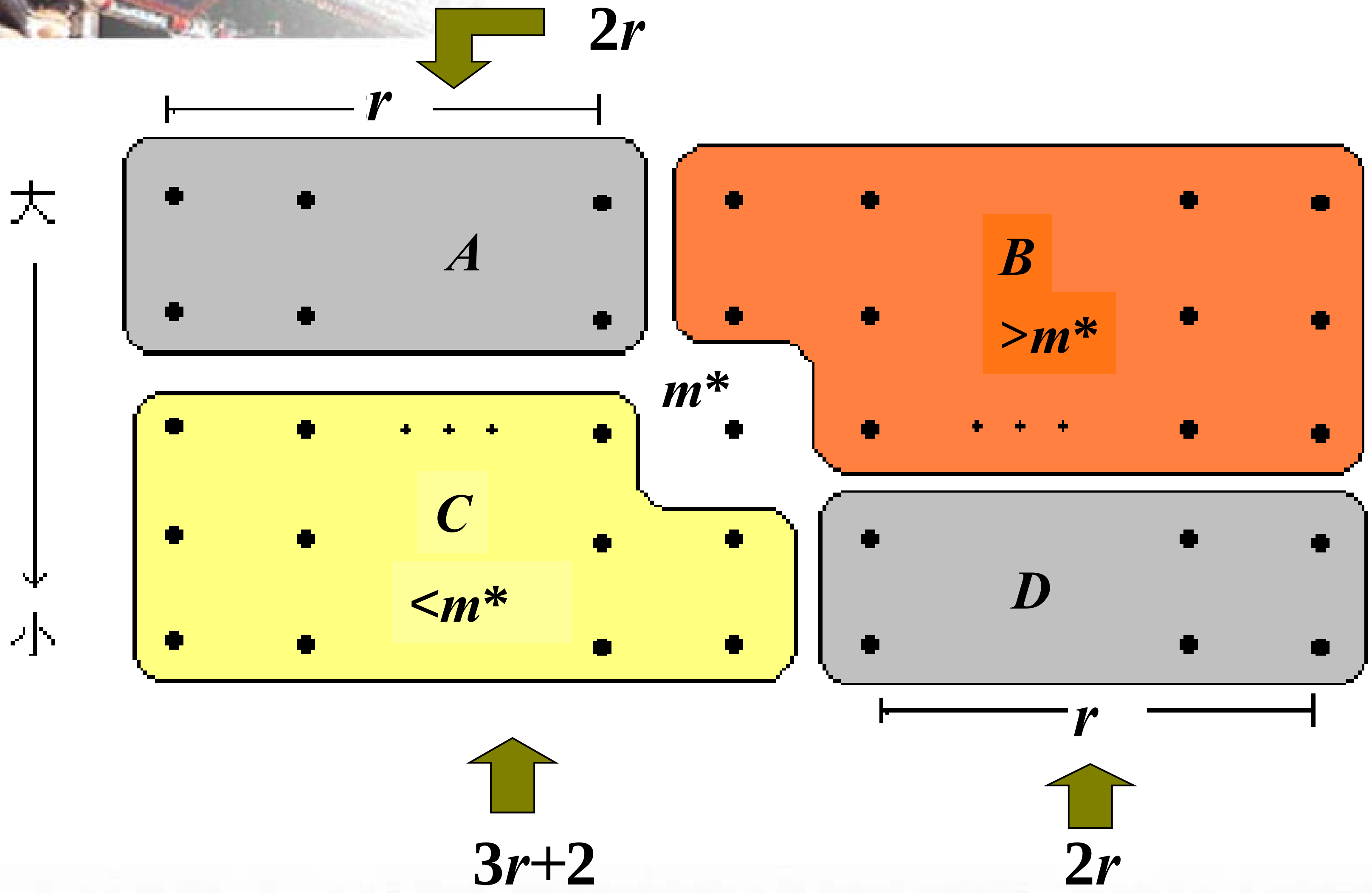
5．$S_1 \leftarrow S_1 \cup C$; $S_2 \leftarrow S_2 \cup B$

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 $2r + 2r + 3r + 2 = 7r + 2$

复杂度估计 $W(n)=O(n)$

不妨设 $n=5(2r+1)$, $|A|=|D|=2r$, $r=\dfrac{\frac{n}{5}-1}{2}=\dfrac{n}{10}-\dfrac{1}{2}$

算法工作量

行2: $O(n)$

行3: $W(n/5)$

行4: $O(n)$

行8-9: $W(7r+2)$

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

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

用递归树做复杂度估计

$$w(n)\le W(\frac{n}{5})+W(\frac{7n}{10})+cn\le cn+\frac{9}{10}cn+\frac{81}{100}cn+\ldots=O(n)$$

递归树

$$tn \qquad\qquad tn$$

$$\frac{tn}{5} \quad \frac{7tn}{10} \qquad\qquad 0.9tn$$

$$\frac{tn}{25} \quad \frac{7tn}{50} \quad \frac{7tn}{50} \quad \frac{49tn}{100} \qquad\qquad 0.81tn$$

……

1 的$2n$次根

$$\omega_j = e^{\frac{2\pi j}{2n}i} = e^{\frac{\pi j}{n}i} = \cos\frac{\pi j}{n} + i\sin\frac{\pi j}{n} \qquad j=0,1,\ldots,2n-1$$

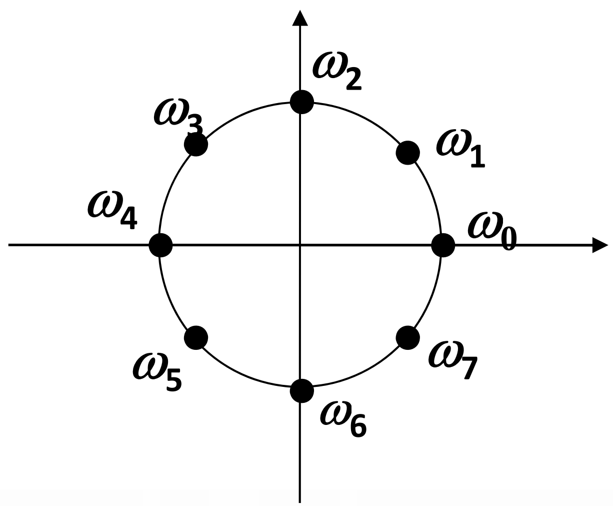
例如n=4, 1的8次方根是：

$$\omega_0 = 1, \qquad \omega_1 = e^{\frac{\pi}{4}i} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i,$$

$$\omega_2 = e^{\frac{\pi}{2}i} = i, \quad \omega_3 = e^{\frac{3\pi}{4}i} = -\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i,$$

$$\omega_4 = e^{\pi i} = -1, \ \omega_5 = e^{\frac{5\pi}{4}i} = -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i,$$

$$\omega_6 = e^{\frac{3\pi}{2}i} = -i, \ \omega_7 = e^{\frac{7\pi}{4}i} = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i$$



北京大学

多项式求值

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

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：$A_1(x)= a_{n-1}$

$A_2(x)= a_{n-2}+ xA_1(x)$

$A_3(x)= a_{n-3}+ xA_2(x)$

…

$A_n(x)= a_0+ xA_{n-1}(x)=A(x)$

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

分治算法

原理：

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

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

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

算法3：

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

复杂度分析：$T(n)=T_1(n)+f(n)$，$f(n)=O(n)$计算$2n$次根时间

$T_1(n)=2T_1(n/2)+g(n)$，$g(n)=O(n)$，

$T_1(1)=O(1)$

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

第3章 动态规划
（Dynamic Programming）

3.1 动态规划的设计思想

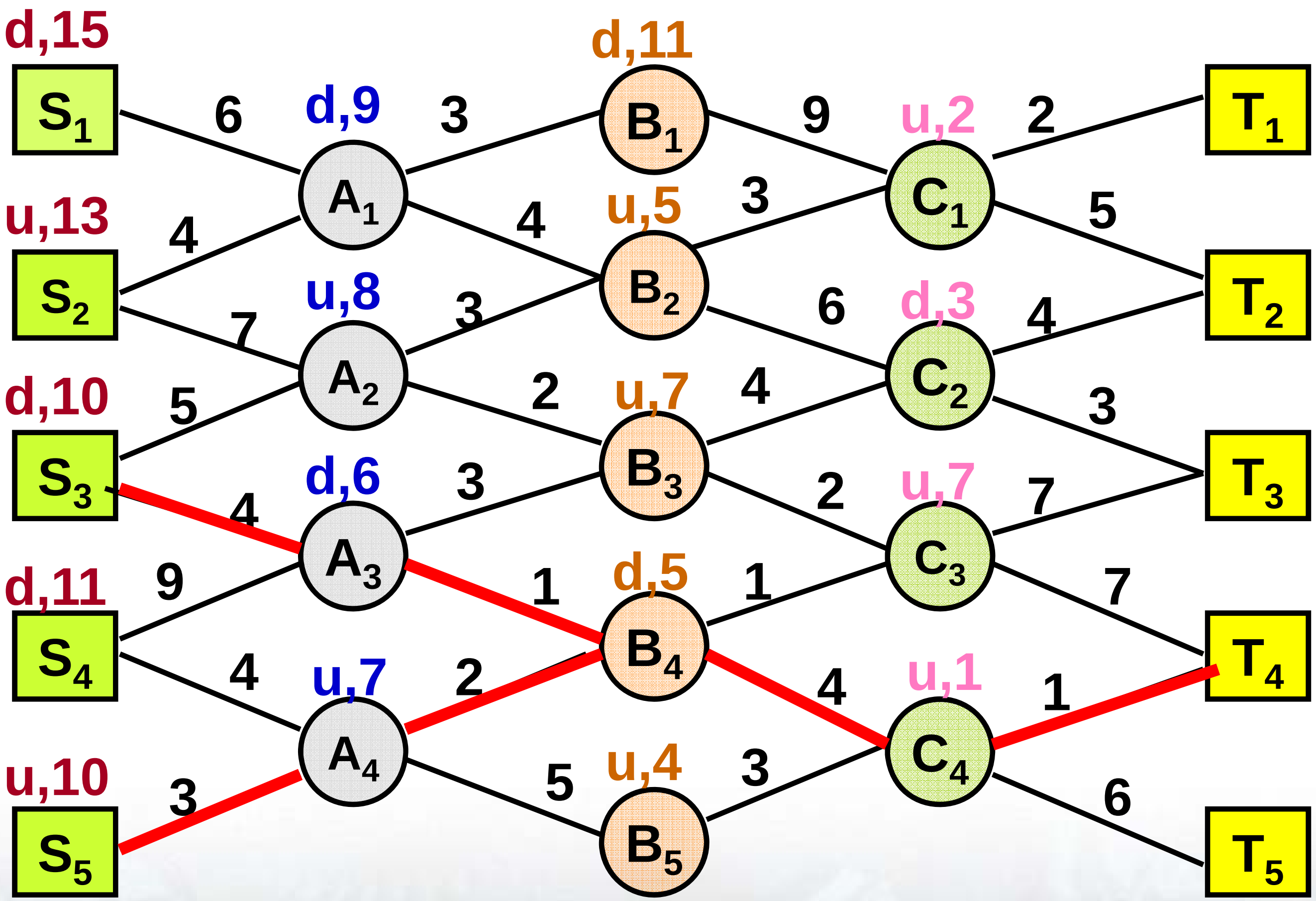
3.2 动态规划的设计要素

3.3 动态规划算法的典型应用

- 投资问题
- 背包问题
- 最长公共子序列LCS
- 图像压缩
- 最大子段和
- 最优二分检索树
- 生物信息学中的动态规划算法

3.1 基本思想和使用条件

例1 求从始点到终点的最短路径



动态规划的基本思想

解：判断序列

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

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

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

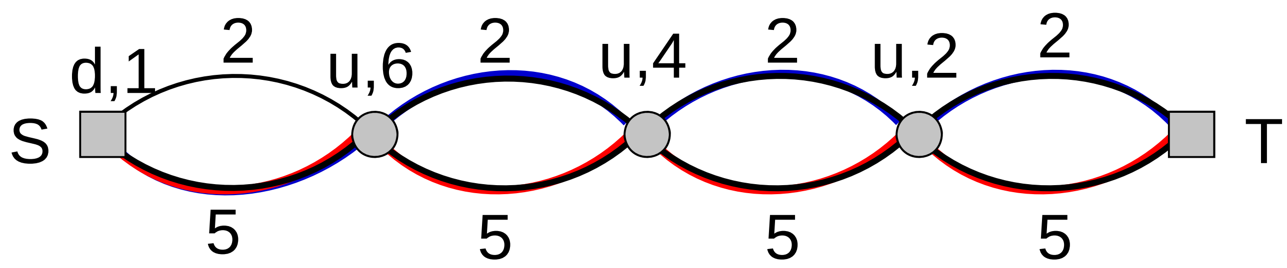
$$F(S_i) = \min_j \{S_i A_j + F(A_j)\}$$

- 优化函数的特点：任何最短路径的子路径都是相对于子路径始点和终点的最短路径
- 求解步骤：确定子问题的边界、从最小的子问题开始进行多步判断

使用条件:优化原则

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

动态规划算法的解：下、上、上、上

不满足优化原则，不能使用动态规划设计技术

3.2 算法设计要素

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 $P = \langle P_0, P_1, \ldots, P_n \rangle$，$n$个矩阵的行数、列数

实例： $P = \langle 10, 100, 5, 50 \rangle$

A_1: 10 × 100, A_2: 100 × 5, A_3: 5 × 50,

乘法次序

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

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

搜索空间的规模

枚举算法：加n个括号的方法有 $\frac{1}{n+1}\binom{2n}{n}$ 种，是一个Catalan数，是指数级别

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

$$=\Omega\left(\frac{1}{n+1}\frac{n^{\frac{1}{2}}2^{2n}n^{2n}e^{n}e^{n}}{e^{2n}n^{\frac{1}{2}}n^{n}n^{\frac{1}{2}}n^{n}}\right)=\Omega(2^{2n}/n^{\frac{3}{2}})$$

动态规划算法

由 i 和 j 确定子问题的边界：输入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

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

确定优化函数和递推方程：

$m[i,j]$ 表示得到 $A_{i..j}$ 的最少的相乘次数，则递推方程和初值

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

设立标记函数：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：递归实现

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. $\quad q \leftarrow$ RecurMatrixChain(P,i,k) + RecurMatrixChain($P,k+1,j$) + $p_{i-1}\, p_k\, p_j$
5. $\quad$ if $q<m[i,j]$
6. $\qquad$ then $m[i,j] \leftarrow q$
7. $\qquad\qquad s[i,j] \leftarrow k$
8. Return $m[i,j]$

这里没有写出算法的全部描述（递归调用的初值等）

递归实现的复杂性

复杂性满足递推关系

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

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

数学归纳法证明 $T(n) \geq 2^{n-1}$

n=2, 显然为真． 假设对于任何小于n 的 k 命题为真， 则

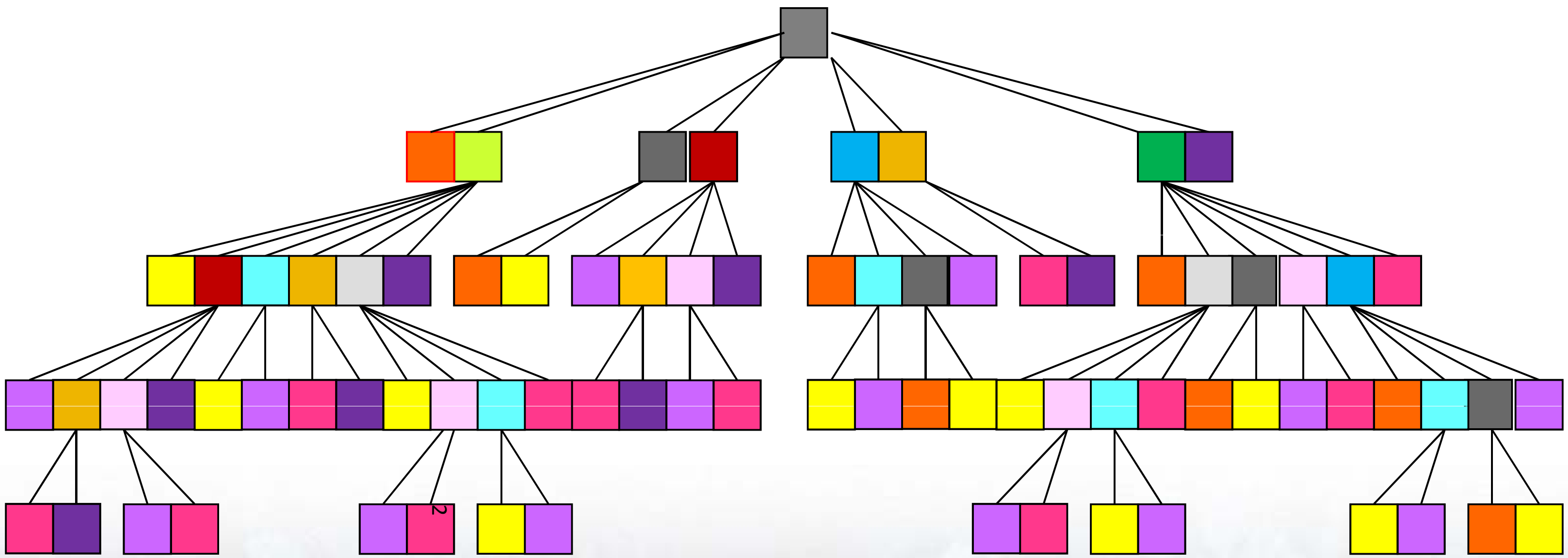
$$T(n) \geq O(n) + 2\sum_{k=1}^{n-1} T(k) \geq O(n) + 2\sum_{k=1}^{n-1} 2^{k-1}$$

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

子问题重复计算

n=5，计算子问题：81个；不同的子问题：15个

子问题	1-1	2-2	3-3	4-4	5-5	1-2	2-3	3-4	4-5	1-3	2-4	3-5	1-4	2-5	1-5
数	8	12	14	12	8	4	5	5	4	2	2	2	1	1	1



算法2：迭代实现

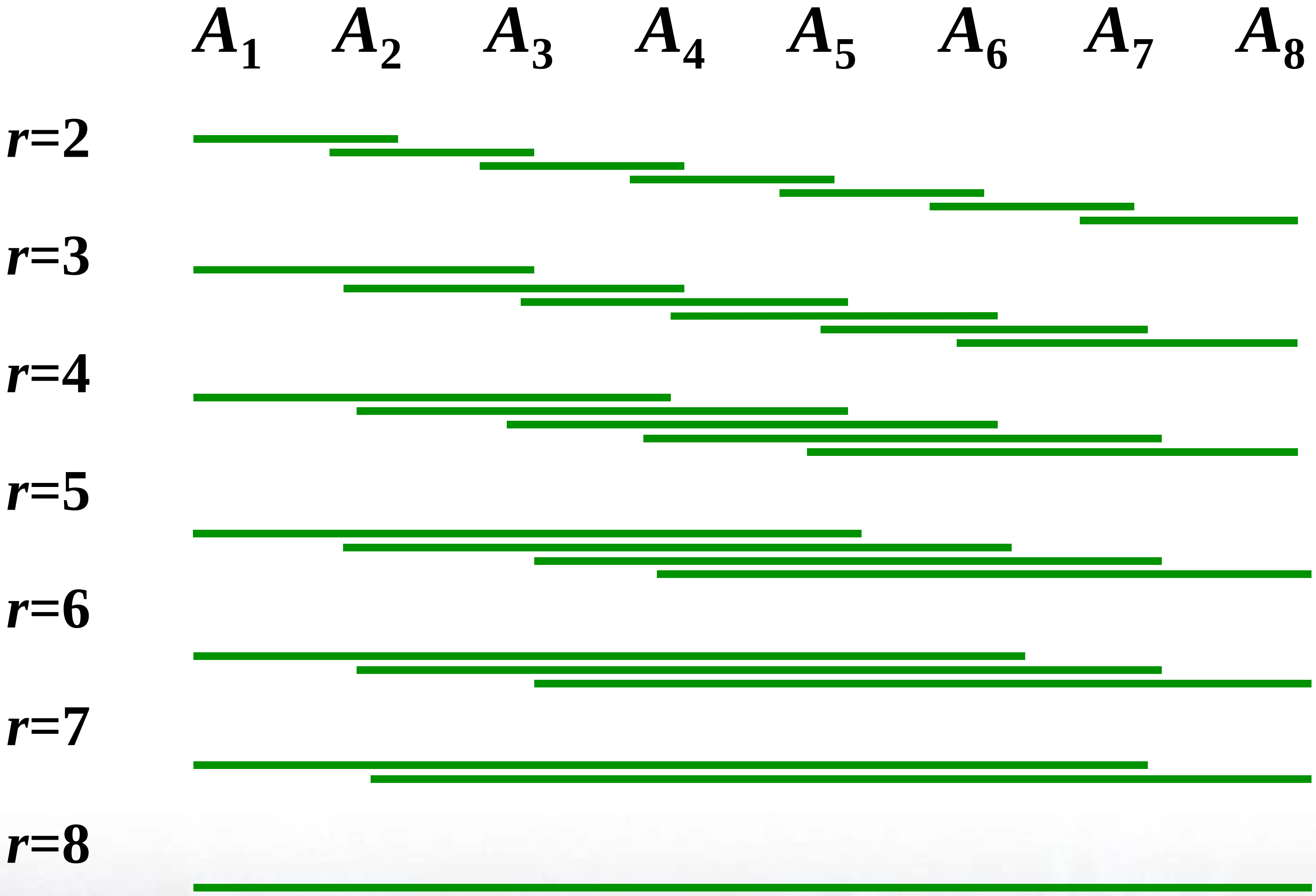
算法2 MatrixChain(P,n)

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都是$O(n)$，循环内为$O(1)$，$W(n)=O(n^3)$

北京大學

n=8 的迭代过程



实例

输入 P= <30, 35, 15, 5, 10, 20>, n=5，

矩阵链：$A_1\,A_2\,A_3\,A_4\,A_5$，其中

A_1:30×35，A_2:35×15，A_3:15×5，A_4:5×10，A_5:10×20

备忘录

r=1	m[1,1]=0	m[2,2]=0	m[3,3]=0	m[4,4]=0	m[5,5]=0
r=2	m[1,2]=15750	m[2,3]=2625	m[3,4]=750	m[4,5]=1000	
r=3	m[1,3]=7875	m[2,4]=4375	m[3,5]=2500		
r=4	m[1,4]=9375	m[2,5]=7125			
r=5	m[1,5]=11875				

r=2	s[1,2]=1	s[2,3]=2	s[3,4]=3	s[4,5]=4	
r=3	s[1,3]=1	s[2,4]=3	s[3,5]=3		
r=4	s[1,4]=3	s[2,5]=3			
r=5	s[1,5]=3				

解： $(A_1\,(A_2A_3))\,(A_4A_5)$

两种实现的比较

递归算法：时间复杂性高，空间较小

非递归算法：时间复杂性较低，空间消耗多

时间复杂性不同的原因：

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

迭代动态规划算法每个子问题只计算一次，时间复杂度等于备忘录中各项的计算量之和（对于需要追踪解的问题，还要加上追踪工作量，一般不超过备忘录计算工作量），通常是问题规模的多项式函数

迭代动态规划算法提高效率的原因：以空间换时间

小结

(1) 划分子问题

用参数表达子问题的边界，将问题求解转 变成多步判断的过程.

(2) 确定优化函数

以该函数的极大(或极小)作为判断的依据，确定是否满足优化原则.

(3) 列出关于优化函数的递推方程 (或不等式)和边界条件

(4) 自底向上计算，以备忘录方法 (表格)存储中间结果

(5) 考虑是否需要设立标记函数

3.3.1 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　$x_1+x_2+\ldots+x_n=m$，$x_i \in \mathrm{N}$

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

子问题划分和优化函数

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

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

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

实例的计算

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

算法的复杂度分析

表中有 m 行 n 列, 共计 mn 项

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

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

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

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

3.3.2 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 $$\sum_{j=1}^{n} w_j x_j \le b, \quad x_j \in \mathbf{N}$$

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

子问题划分、优化函数、标记函数

$F_k(y)$：装前 k 种物品，总重不超过 y，背包的最大价值

$i(k,y)$：装前 k 种物品，总重不超过 y，背包达最大价值时装入物品的最大标号

递推方程、边界条件、标记函数

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

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

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

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

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

实例计算

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

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

$b = 10$

$F_k(y)$ 的计算表如下:

$k \backslash y$	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

追踪问题的解

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

$i_4(10)=4 \Rightarrow x_4 \geq 1$

$i_4(10-w_4)=i_4(3)=2 \Rightarrow x_2 \geq 1$, $x_4=1$, $x_3=0$

$i_2(3-w_2)=i_2(0)=0 \Rightarrow x_2 = 1$, $x_1=0$

解 $x_1=0$, $x_2=1$, $x_3=0$, $x_4=1$

追踪解 x_j 的算法

算法 TrackSolution

输入：$i_k(y)$表，其中$k=1,2,\ldots,n$，$y=1,2,\ldots,b$

输出：$x_1, x_2, \ldots, x_n$，n种物品的装入量

1. for j=1 to n do
2. $\quad x_j \leftarrow 0$
3. $y \leftarrow b$, $j \leftarrow n$
4. $j \leftarrow i_j(y)$，
5. $x_j \leftarrow 1$
6. $y \leftarrow y - w_j$
7. while $i_j(y)=j$ do
8. $\quad y \leftarrow y - w_j$
9. $\quad x_j \leftarrow x_j+1$
10. if $i_j(y) \neq 0$ goto 4

3.3.3 最长公共子序列 LCS

相关概念

X 的子序列 Z ：设序列 $X, Z,$

$$X = < x_1, x_2, \ldots, x_m >$$

$$Z = < z_1, z_2, \ldots, z_k >$$

若存在 X 的元素构成的严格递增序列 $< x_{i_1}, x_{i_2}, \ldots, x_{i_k} >$

使得 $z_j = x_{i_j}, j = 1,2,\ldots,k,$ 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

问题描述

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

实例

X: A B C B D A B

Y: B D C A B A

最长公共子序列: **B C B A**

蛮力算法：检查 X 的每个子序列在Y 中出现

每个子序列 $O(n)$ 时间，X 有 2^m 个 子序列，最坏情况下

时间复杂度：$O(n2^m)$

北京大学

子问题划分及依赖关系

子问题边界：X和Y起始位置为1，X的终止位置是 i，Y的终止位置是j，记作

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

依赖关系：

$$X=<x_1,x_2,\ldots,x_m>, Y=<y_1,y_2,\ldots,y_n>, Z=<z_1,z_2,\ldots,z_k>,$$

Z 为 X 和 Y 的 LCS，那么

(1) 若$x_m=y_n \Rightarrow z_k=x_m=y_n$,且$Z_{k-1}$是$X_{m-1}$与$Y_{n-1}$的 LCS；

(2) 若$x_m \neq y_n$, $z_k \neq x_m \Rightarrow Z$是$X_{m-1}$与$Y$ 的 LCS；

(3) 若$x_m \neq y_n$, $z_k \neq y_n \Rightarrow Z$是$X$与$Y_{n-1}$的 LCS.

满足优化原则和子问题重叠性

递推方程、标记函数

令 X 与 Y 的子序列

$$X_i = \langle x_1, x_2, \ldots , x_i \rangle, \quad Y_j = \langle y_1, y_2, \ldots , y_j \rangle$$

$C[i,j]$：X_i 与 Y_j 的 LCS 的长度

递推方程

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

标记函数：$B[i,j]$，其值为字符↖、←、↑，分别表示$C[i,j]$取得最大值时的三种情况

算法3.4 LCS(*X*,*Y*,*m*,*n*)

```
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]←'←'
```

利用标记函数构造解

算法 Structure Sequence(B, i, j)

输入：$B[i,j]$

输出：X与Y的最长公共子序列

1. **if** i=0 **or** j=0 **then return** //一个序列为空
2. **if** $B[i,j]$ = "↖"
3. **then** 输出$X[i]$
4. Structure Sequence(B, $i-1$, $j-1$)
5. **else if** $B[i,j]$= "↑" **then** Structure Sequence (B, $i-1$, j)
6. **else** Structure Sequence (B, i, $j-1$)

算法的计算复杂度

计算优化函数和标记函数：时间为$O(mn)$

构造解：每一步至少缩小X 或 Y 的长度，时间$\Theta(m+n)$

空间：$\Theta(mn)$

实例

输入：$X=\langle A,B,C,B,D,A,B\rangle$, $Y=\langle B,D,C,A,B,A\rangle$，

标记函数：

	1	2	3	4	5	6
1	*B*[1,1]=↑	*B*[1,2]=↑	*B*[1,3]=↑	*B*[1,4]=↖	*B*[1,5]=←	*B*[1,6]=↖
2	*B*[2,1]=↖	*B*[2,2]=←	*B*[2,3]=←	*B*[2,4]=↑	*B*[2,5]=↖	*B*[2,6]=←
3	*B*[3,1]=↑	*B*[3,2]=↑	*B*[3,3]=↖	*B*[3,4]=←	*B*[3,5]=↑	*B*[3,6]=↑
4	*B*[4,1]=↑	*B*[4,2]=↑	*B*[4,3]=↑	*B*[4,4]=↑	*B*[4,5]=↖	*B*[4,6]=←
5	*B*[5,1]=↑	*B*[5,2]=↑	*B*[5,3]=↑	*B*[5,4]=↑	*B*[5,5]=↑	*B*[5,6]=↑
6	*B*[6,1]=↑	*B*[6,2]=↑	*B*[6,3]=↑	*B*[6,4]=↖	*B*[6,5]=↑	*B*[6,6]=↖
7	*B*[7,1]=↑	*B*[7,2]=↑	*B*[7,3]=↑	*B*[7,4]=↑	*B*[7,5]=↑	*B*[7,6]=↑

解：*X*[2],*X*[3], *X*[4], *X*[6], 即 B, C, B, A

3.3.4 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$i$个像素点灰度值

变位压缩存储格式：将$\{p_1, p_2, \ldots, p_n\}$分割 m 段 $S_1, S_2, \ldots, S_m$

i 段有$l[i]$个像素，每个像素 $b[i]$位，h_i 为 该段最大像素的位数

$$h_i \le b[i] \le 8, \quad h_i = \left\lceil \log(\max_{p_k \in S_i} p_k + 1) \right\rceil$$

约束条件：每段像素个数 $l[i] \le 256$

段头11位：$b[i]$的二进制表示(3 位) + $l[i]$的二进制表示(8位)

i 段占用空间：$b[i] \times l[i] + 11$

问题：给定像素序列$\{p_1, p_2, \ldots, p_n\}$，确定最优分段，即

$$\min_T \{\sum_{i=1}^{j} (b[i] \times l[i] + 11)\}, \quad T = \{S_1, S_2, \ldots, S_j\}\text{为分段}$$

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：$11\times3+4\times3+8\times1+2\times8=69$

分法2：$11\times1+8\times12=107$

分法3：$11\times12+4\times3+8\times1+1\times5+2\times3=163$

结论：分法1是其中最优的分法

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

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

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

P_1 P_2 … P_{i-k}	P_{i-k+1} … P_i
$S[i-k]$位	k 个灰度 $k*b\max(i-k+1,i)$位

算法

```
Compress (P,n)                //计算最小位数S[n]
1. Lmax←256; header←11; S[0]←0   //最大段长Lmax, 头header
2. for i←1 to n do
3.     b[i]←length(P[i])      //b[i]是第i个灰度P[i]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
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
```

时间复杂度 $T(n)=O(n)$

P = <10, 12, 15, 255, 1, 2>.

$S[1]=15$, $S[2]=19$, $S[3]=23$, $S[4]=42$, $S[5]=50$

$l[1]=1$, $l[2]=2$, $l[3]=3$, $l[4]=1$, $l[5]=2$

实例

10	12	15	255	1	2

$S[5]=50$ $1\times2+11$

10	12	15	255	1	2

$S[4]=42$ $2\times2+11$

10	12	15	255	1	2

$S[3]=23$ $3\times8+11$

10	12	15	255	1	2

$S[2]=19$ $4\times8+11$

10	12	15	255	1	2

$S[1]=15$ $5\times8+11$

10	12	15	255	1	2

$6\times8+11$

追踪解

算法 Traceback(n,l)

输入：数组l

输出：数组C // $C[j]$是从后向前追踪的第 j段的长度

1. $j \leftarrow 1$ // j为正在追踪的段数
2. **while** $n \neq 0$ **do**
3. $C[j] \leftarrow l[n]$
4. $n \leftarrow n - l[n]$
5. $j \leftarrow j+1$

时间复杂度：$O(n)$

3.3.5 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

算法 Enumerate

输入：数组$A[1..n]$

输出：*sum, first, last*

```
sum←0
for i←1 to n do               // i为当前和的首位置
    for j←i to n do           // j为当前和的末位置
        thissum←0             // thissum为A[i]到A[j]之和
        for k←i to j do
            thissum←thissum+A[k]
        if thissum > sum
        then sum←thissum
             first←i      // 记录最大和的首位置
             last←j       // 记录最大和的末位置
```

时间复杂度：$O(n^3)$

算法2 分治策略

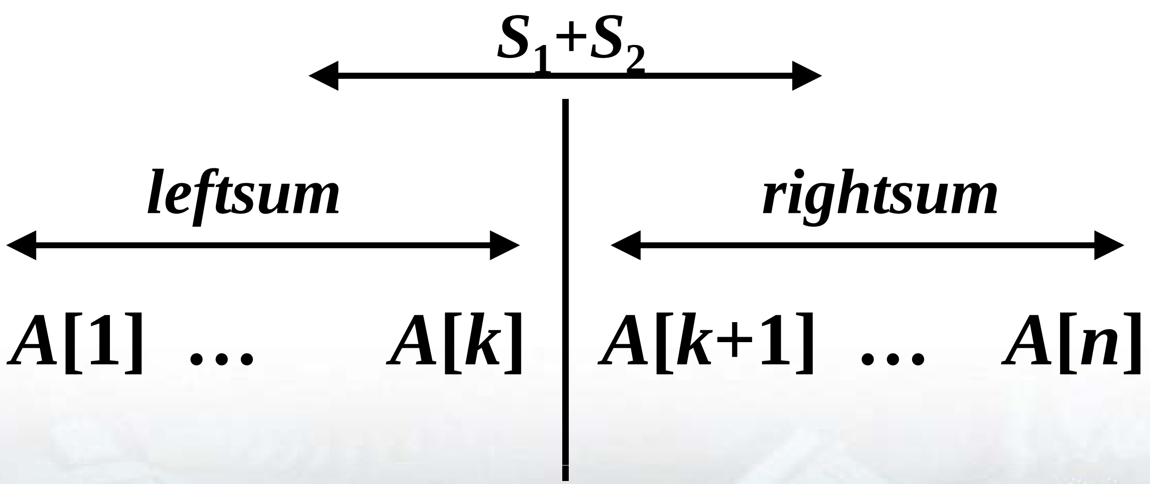
将序列分成左右两半，中间分点*center*

递归计算左段最大子段和 *leftsum*

递归计算右段最大子段和 *rightsum*

$a_{center} \to a_1$的最大和S_1，$a_{center+1} \to a_n$的最大和S_2

max { *leftsum*, *rightsum*, S_1+S_2}



分治算法

算法 MaxSubSum(A, $left$, $right$）

输入：数组A，$left$，$right$分别是A的左、右边界

输出：A的最大子段和sum及其子段的前后边界

1. if $|A|=1$ then 输出元素值（当值为负时输出0）
2. $center \leftarrow \lfloor (left+right)/2 \rfloor$
3. $leftsum \leftarrow$ MaxSubSum(A,$left$,$center$)　　//子问题A_1
4. $righsum \leftarrow$ MaxSubSum(A,$center$+1,$right$)　//子问题A_2
5. $S_1 \leftarrow A_1[center]$　　　//从$center$向左的最大和
6. $S_2 \leftarrow A_2[center+1]$　　　//从$center$+1向右的最大和
7. $sum \leftarrow S_1+S_2$
8. if $leftsum > sum$ then $sum \leftarrow leftsum$
9. if $rightsum > sum$ then $sum \leftarrow rightsum$

时间：$T(n)=2T(n/2)+O(n)$, $T(c)=O(1)$

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

算法3：动态规划

令$C[i]$ 是 $A[1..i]$中必须包含元素$A[i]$的最大子段和

$$C[i]=\max_{1\le k\le i}\{\sum_{j=k}^{i}A[j]\}$$

$A[1]$ $A[2]$ $A[3]$ … $A[j-1]$ | $A[j]$ | … $A[n]$

最大子段和 $C[j-1]$

最大子段和 $C[j]$

递推方程: $C[i]=\max\{C[i-1]+A[i], A[i]\}$ $i=2,\ldots,n$

$C[1]=A[1]$ 若$A[1]>0$, 否则$C[1]=0$

解: $$\mathrm{OPT}(A)=\max_{1\le i\le n}\{C[i]\}$$

北京大学

算法 MaxSum

算法3.10 MaxSum(*A*, *n*)

输入：数组*A*

输出：最大子段和*sum*, 子段的最后位置*c*

1. *sum*←0
2. *b*←0　　// *b*是前一个最大子段和
3. for *i*←1 to *n* do
4. 　if *b*>0
5. 　then *b*←*b*+*A*[*i*]
6. 　else *b*←*A*[*i*]
7. 　if *b*>*sum*
8. 　then *sum*←*b*
9. 　　*c*←*i*　　//记录最大和的末项标号
10. return *sum*，*c*

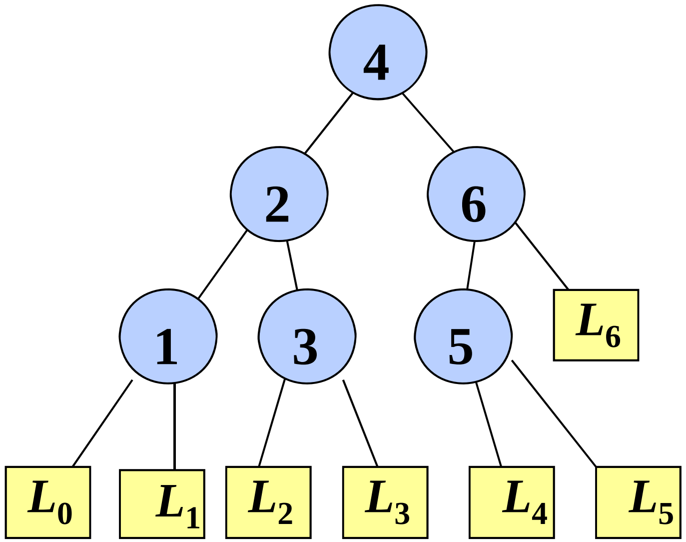
时间复杂度：*O*(*n*), 空间复杂度：*O*(*n*)

3.3.6 最优二叉检索树

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.

检索方法:

1. 初始，x与根元素比较;
2. x<根元素，递归进入左子树;
3. x>根元素，递归进入右子树;
4. x=根元素，算法停止，输出x;
5. x到达叶结点，算法停止，输出x不在数组中.



S={ 1, 2, 3, 4, 5, 6}

存取概率不等情况

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

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

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 ,

S的存取概率分布如下：

$$P = <a_0, b_1, a_1, b_2, a_2, \ldots, b_n, a_n>$$

实例

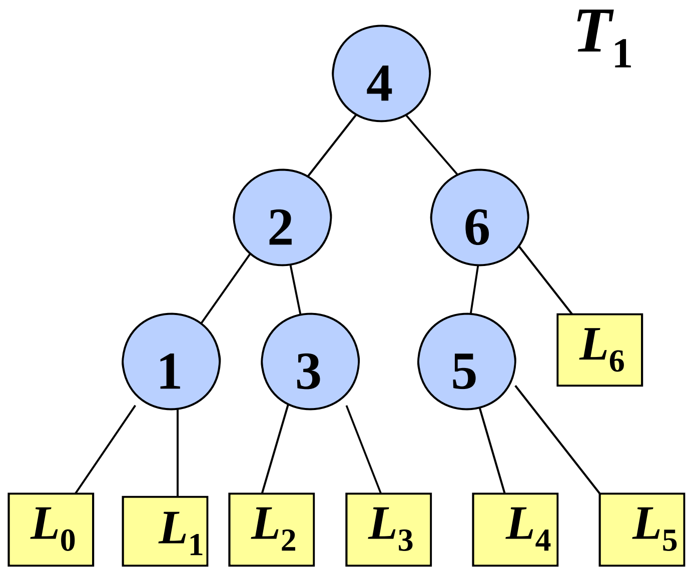
$S=\{1,2,3,4,5,6\}$

$P=<0.04, 0.1, 0.01, 0.2, 0.05, 0.2, 0.02, 0.1, 0.02, 0.1, 0.07, 0.05, 0.04>$

实例

$S = \{\ 1, 2, 3, 4, 5, 6\ \}$

$P = <\ 0.04, 0.1, 0.01, 0.2, 0.05, 0.2,$
$0.02, 0.1, 0.02, 0.1, 0.07, 0.05,$
$0.04\ >$

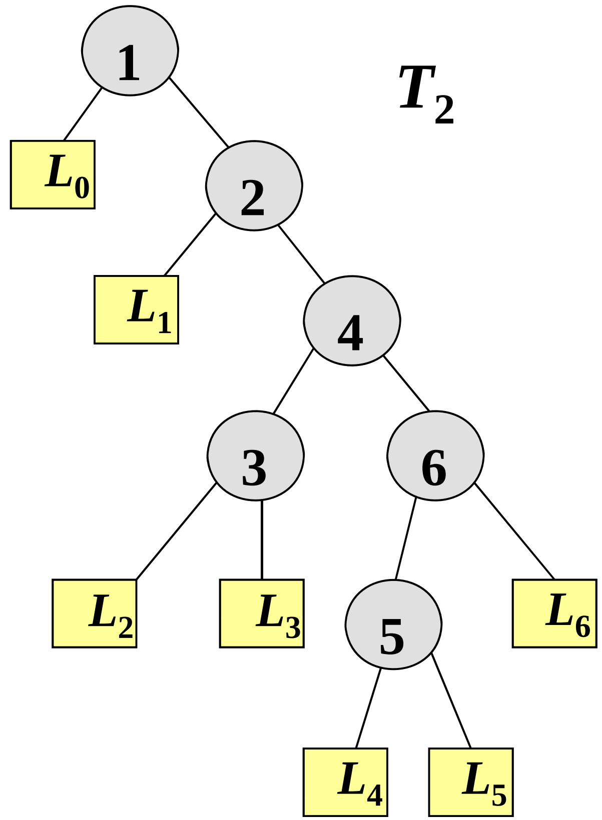


$$m(T_1)=[1*0.1+2*(0.2+0.05)+3*(0.1+0.2+0.1)]$$
$$+[3*(0.04+0.01+0.05+0.02+0.02+0.07)+2*0.04]$$
$$= 1.8+0.71=2.51$$

实例

S = { 1, 2, 3, 4, 5, 6 }

P = < 0.04, 0.1, 0.01, 0.2, 0.05, 0.2, 0.02, 0.1, 0.02, 0.1, 0.07, 0.05, 0.04 >



$$
\begin{aligned}
m(T_2) &= [\,1*0.1+2*0.2 + 3*0.1 \\
&\quad + 4*(0.2+0.05) + 5*0.1\,] \\
&\quad + [\,1*0.04 + 2*0.01 \\
&\quad + 4*(0.05 + 0.02 + 0.04) \\
&\quad + 5*(0.02 + 0.07)] \\
&= 2.3 + 0.95 = 3.25
\end{aligned}
$$

问题

数据集 $S=<x_1,x_2,\ldots,x_n>$

存取概率分布 $P=<a_0, b_1, a_1, b_2, \ldots, a_i, b_{i+1}, \ldots, b_n, a_n>$

结点 x_i 在 T 中的深度是 $d(x_i)$, $i=1,2,\ldots,n$,

空隙 L_j 的深度为 $d(L_j)$, $j=0,1,\ldots,n$,

平均比较次数为:

$$t=\sum_{i=1}^{n} b_i(1+d(x_i))+\sum_{j=0}^{n} a_j d(L_j)$$

问题：给定数据集 S 和相关存取概率分布 P，求一棵最优的(即平均比较次数最少的) 二分检索树.

算法设计:子问题划分

$S[i,j] = \langle x_i, x_{i+1}, \ldots, x_j \rangle$ 是S 以 i 和 j 作为边界的子数据集

$P[i,j] = \langle a_{i-1}, b_i, a_i, b_{i+1}, \ldots, b_j, a_j \rangle$是对应$S[i,j]$存取概率分布

例：$S=\langle A, B, C, D, E \rangle$

$P=\langle 0.04, 0.1, 0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06, 0.1, 0.01 \rangle$

$S[2,4]=\langle B, C, D \rangle$

$P[2,4]=\langle 0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06 \rangle$

子问题划分：以 x_k 作为根，划分成两个子问题：

$S[i,k-1], P[i,k-1]$

$S[k+1,j], P[k+1,j]$

例：以B为根，划分成以下子问题：

$S[1,1]=\langle A \rangle$，$P[1,1]=\langle 0.04, 0.1, 0.02 \rangle$

$S[3,5]=\langle C,D,E \rangle$, $P[3,5]=\langle 0.02, 0.1, 0.05, 0.2, 0.06, 0.1, 0.01 \rangle$

递推方程

设 *m*[*i*,*j*] 是相对于输入 *S*[*i*,*j*] 和 *P*[*i*,*j*] 的最优二叉搜索树的平均比较次数，令

$$w[i,j]=\sum_{p=i-1}^{j}a_p+\sum_{q=i}^{j}b_q$$

是 *P*[*i*,*j*] 中所有概率（包括数据元素与空隙）之和

递推方程：

$$m[i,j]=\min_{i\le k\le j}\{m[i,k-1]+m[k+1,j]+w[i,j]\},\quad 1\le i\le j\le n$$

$$m[i,i-1]=0,\quad i=1,2,\ldots,n$$

证明

$m[i,j]_k$：根为 x_k 时的二分检索树平均比较次数的最小值

$$
\begin{aligned}
& m[i,j]_k \\
&= (m[i,k-1] + w[i,k-1]) + (m[k+1,j] + w[k+1,j]) + 1 \times b_k \\
&= (m[i,k-1] + m[k+1,j]) + (w[i,k-1] + b_k + w[k+1,j]) \\
&= (m[i,k-1] + m[k+1,j]) + (\sum_{p=i-1}^{k-1} a_p + \sum_{q=i}^{k-1} b_q) + b_k + (\sum_{p=k}^{j} a_p + \sum_{q=k+1}^{j} b_q) \\
&= (m[i,k-1] + m[k+1,j]) + \sum_{p=i-1}^{j} a_p + \sum_{q=i}^{j} b_q \\
&= m[i,k-1] + m[k+1,j] + w[i,j]
\end{aligned}
$$

平均比较次数：在所有 k 的情况下 $m[i,j]_k$ 的最小值，

$$m[i,j]=\min\{m[i,j]_k \mid i \le k \le j\}$$

实例

$$m[i,j]=\min_{i\le k\le j}\{m[i,k-1]+m[k+1,j]+w[i,j]\}\quad 1\le i\le j\le n$$

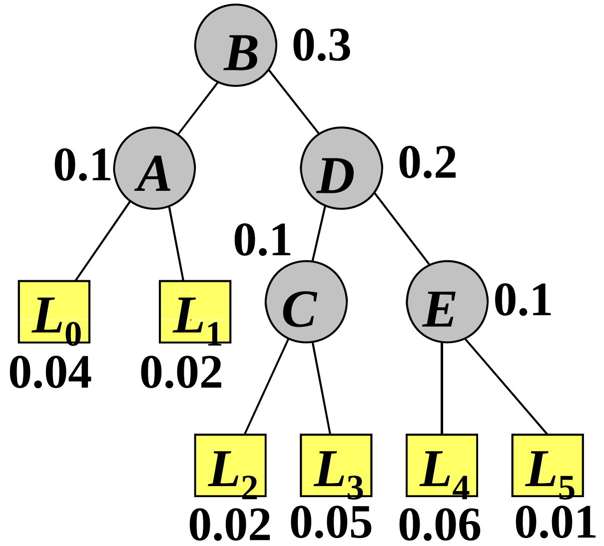
$$m[i,i-1]=0$$

$$m[1,1]=0.16,\quad m(3,5)=0.88$$

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

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

$$=1+\{0.16+0.88\}=2.04$$



复杂性估计:

$T(n)=O(n^3)\qquad S(n)=O(n^2)$

3.3.7生物信息学中的动态规划算法

RNA二级结构预测

一级结构：由字母 *A,C,G,U* 标记的核苷酸构成的一条链

二级结构：核苷酸相互匹配构成二级结构（平面图）

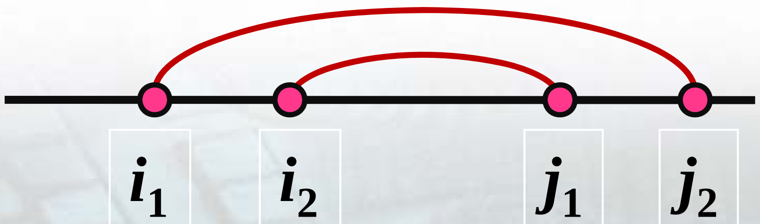
匹配原则：

(1) 配对*U-A,C-G*；

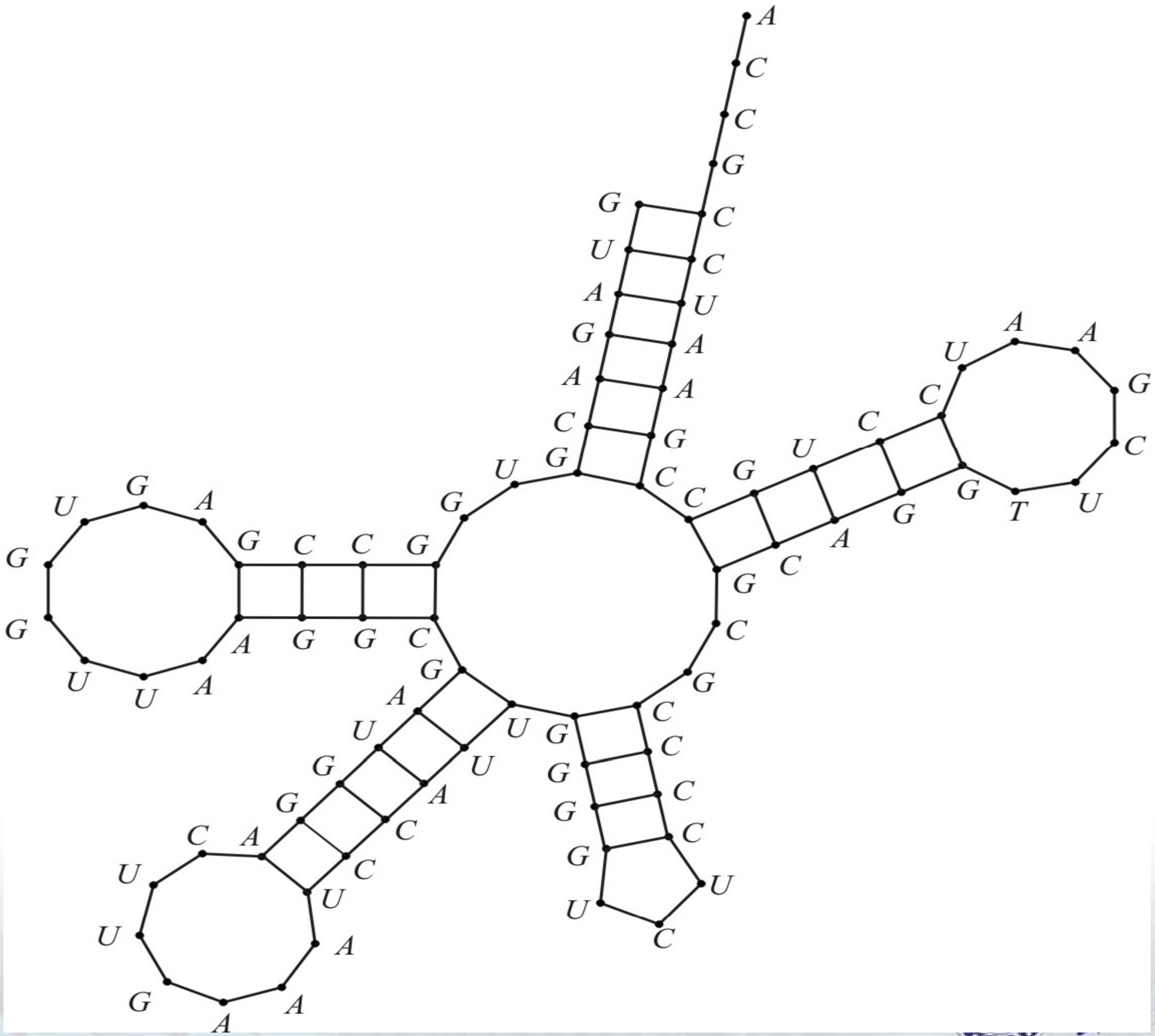
(2) 末端不出现“尖角”，位置i - j 配对，则 $i \le j-4$；

(3) 每个核苷酸只能参加一个配对；

(4) 不允许交叉，即如果位置 i_1, i_2, j_1, j_2 满足$i_1<i_2<j_1<j_2$，不允许 i_1-j_1, i_2-j_2配对. 但可以允许i_1-j_2, i_2-j_1配对.

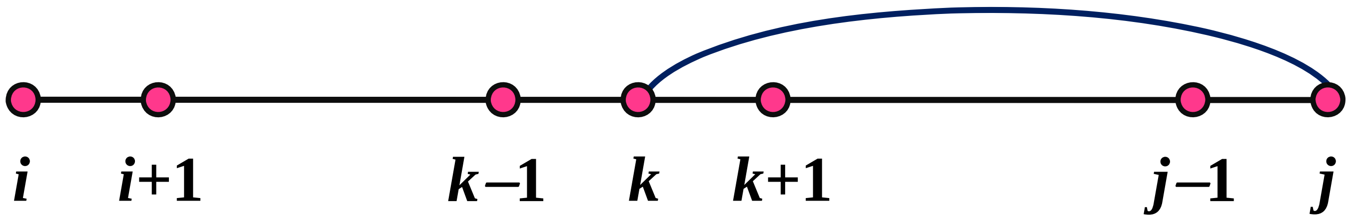


实例：4sRNA的二级结构



问题与算法设计

问题：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**C**[*i,j*]是序列*S*[*i..j*]的最大匹配对数

$$C[i,j]=\max\{C[i,j-1],\ \max_{i\le k<j-4}\{1+C[i,k-1]+C[k+1,j]\}\}$$

$$C[i,j]=0 \quad j-i<5$$

算法时间复杂度是$O(n^3)$

序列比对

编辑距离：给定两个序列S_1和S_2，通过一系列字符编辑（插入、删除、替换）操作，将S_1转变成S_2. 完成这种转换所需要的最少的编辑操作个数称为S_1和S_2的编辑距离.

实例：**vintner** 转变成 **writers**，编辑距离≤6：

vintner

删除v：**-intner**

插入w：**wintner**

插入r：**wrintner**

删除n：**wri-tner**

删除n：**writ-er**

插入s：**writers**

算法设计

$S_1[1..n]$ 和 $S_2[1..m]$ 表示两个子序列

子问题划分：$S_1[1..i]$ 和 $S_2[1..j]$

$C[i,j]$： $S_1[1..i]$ 和 $S_2[1..j]$ 的编辑距离

$$C[i,j] = \min\{C[i-1,j]+1, C[i,j-1]+1, C[i-1,j-1]+t[i,j]\}$$

$$t[i,j] = \begin{cases} 0 & S_1[i] = S_2[j] \\ 1 & S_1[i] \neq S_2[j] \end{cases}$$

$$C[0,j] = j,$$

$$C[i,0] = i$$

算法的时间复杂度是$O(nm)$

小结

(1) 引入参数来界定子问题的边界.

(2) 判断该优化问题是否满足优化原则.

(3) 注意子问题的重叠程度.

(4) 给出带边界参数的优化函数定义与优化函数的递推关系考虑标记函数. 找到递推关系的初值.

(5) 采用自底向上的实现技术，从最小的子问题开始迭代计算，计算中用备忘录保留优化函数和标记函数的值.

(6) 动态规划算法的时间复杂度是对所有子问题(备忘录)的计算工作量求和(可能需要追踪解的工作量)

(7) 动态规划算法一般使用较多的存储空间，这往往成为限制动态规划算法使用的瓶颈因素.

第4章 贪心法

（Greedy Approach）

4.1 贪心法的设计思想

4.2 贪心法的正确性证明

4.3 对贪心法得不到最优解情况的处理

4.4 贪心法的典型应用

  4.4.1 最优前缀码

  4.4.2 最小生成树

  4.4.3 单源最短路径

4.1贪心法的设计思想

活动选择问题

输入：$S=\{1, 2, \ldots, n\}$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 $\Leftrightarrow$ $s_i \geq f_j$ 或 $s_j \geq f_i$.

求：最大的两两相容的活动集 A

实例

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

策略1：排序使得 $s_1 \leq s_2 \leq \ldots \leq s_n$，从前向后挑选

策略2：排序使得 $f_1 - s_1 \leq f_2 - s_2 \leq \ldots \leq f_n - s_n$，从前向后挑选

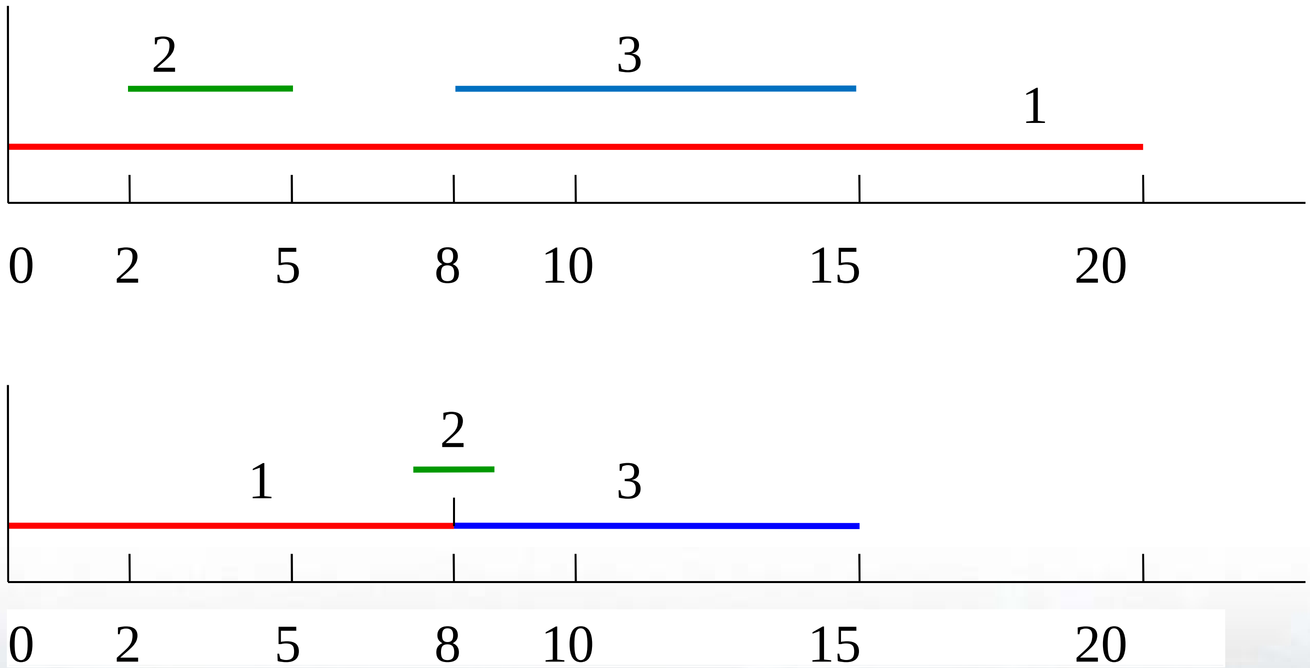
策略3：排序使得 $f_1 \leq f_2 \leq \ldots \leq f_n$，从前向后挑选

以上策略中的挑选都要注意满足相容性条件

两个反例

策略1：$S=\{1,2,3\}$， $s_1=0$, $f_1=20$, $s_2=2$, $f_2=5$, $s_3=8$, $f_3=15$

策略2：$S=\{1,2,3\}$， $s_1=0$, $f_1=8$, $s_2=7$, $f_2=9$, $s_3=8$, $f_3=15$



贪心算法

算法 Greedy Select

输入：活动集S，s_i，f_i，i=1,2,…,n，且 $f_1 \le \ldots \le f_n$

输出：$A \subseteq S$，选中的活动子集

1. $n \leftarrow length[S]$ // 活动个数
2. $A \leftarrow \{1\}$
3. $j \leftarrow 1$ //已选入的最后一个活动的标号
4. for $i \leftarrow 2$ to n do
5. if $s_i \ge f_j$ //判断相容性
6. then $A \leftarrow A \cup \{i\}$
7. $j \leftarrow i$
8. return A

最后完成时间 $t = \max\{ f_k : k \in A \}$

算法运行实例

输入：$S=\{1, 2, \ldots, 10\}$

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

解：$A = \{1, 4, 8\}$，$t = 11$

时间复杂度：排序+活动选择=$O(n\log n)+O(n)=O(n\log n)$

问题：如何证明该算法对所有的实例都能得到正确的解？

算法的正确性证明

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

证：$S=\{1,2,\ldots,n\}$是活动集，且 $f_1\le \ldots \le f_n$

归纳基础：$k=1$, 证明存在最优解包含活动1

任取最优解A, A中的活动按截止时间递增排列. 如果A的第一个活动为 j，$j\neq 1$, 令

$$A' = (A-\{ j \})\cup\{1\},$$

由于 $f_1 \le f_j$, A'也是最优解，且含有1.

算法正确性证明（续）

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，即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.

北京大学

贪心算法的特点

设计要素：

(1) 贪心法适用于组合优化问题，该问题满足优化原则.

(2) 求解过程是多步判断过程，最终的判断序列对应于问题的最优解.

(3) 判断依据某种“短视的”贪心选择性质，性质的好坏决定了算法的成败.

(4) 贪心法必须进行正确性证明

贪心法的优势：

算法简单，时间和空间复杂性低

4.2 贪心法的正确性证明

数学归纳法

1. 叙述一个描述算法正确性的命题$P(n)$，n为算法步数或者问题规模
2. 归纳基础：$P(1)$ 或 $P(n_0)$为真，n_0为某个自然数
3. 归纳步骤：$P(k) \Rightarrow P(k+1)$　　第一数学归纳法

   $\forall k(k<n)P(k) \Rightarrow P(n)$　　第二数学归纳法

交换论证

1. 分析算法的解的结构特征
2. 从一个最优解逐步进行结构变换(替换成分、交换次序等)
3. 证明上述变换最终得到算法的解、变换有限步结束、变换保持最优性不降低、

最优装载 Loading

问题：

n 个集装箱1,2,…,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

北京大学

正确性证明

命题：对装载问题任何规模为n 的输入，算法得到最优解.

对问题规模归纳，设集装箱从轻到重记为1, 2, … , n.

证：k=1, 只有1个箱子，算法显然正确.

假设对于 k 个集装箱的输入，贪心法都可以得到最优解，考虑 输入 $N = \{1, 2, \ldots, k+1\}$, 其中 $w_1 \le w_2 \le \ldots \le w_{k+1}$.

由归纳假设，对于$N' = \{2,3,\ldots,k+1\}$，$C' = C - w_1$, 贪心法 得到最优解 I'. 令 $I = \{1\} \cup I'$，则 I (算法解)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.

最小延迟调度

问题：

给定客户集合A，$\forall i\in A$，t_i 为服务时间，d_i 为完成时间，t_i, d_i 为正整数．一个调度 $f: A\to \mathrm{N}$，$f(i)$为客户 i 的开始时间．求最大延迟达到最小的调度，即求 f 使得

$$\min_f\{\max_{i\in A}\{f(i)+t_i-d_i\}\}$$

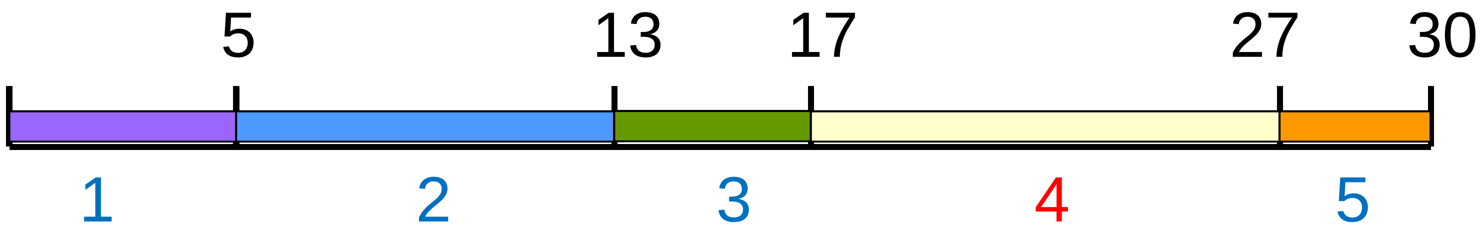
$$\forall i,j\in A, i\neq j, f(i)+t_i\le f(j) \text{ or } f(j)+t_j\le f(i)$$

实例

$A=\{1, 2, 3, 4, 5\}$, $T=<5, 8, 4, 10, 3>$, $D=<10, 12, 15, 11, 20>$

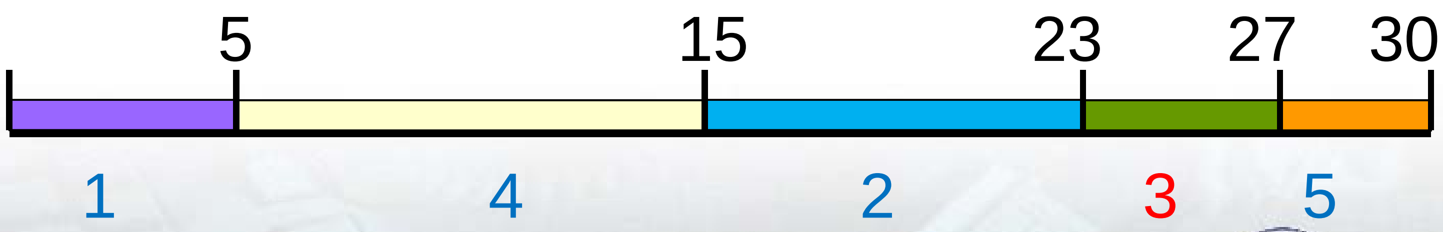
调度1： $f_1(1)=0$, $f_1(2)=5$, $f_1(3)=13$, $f_1(4)=17$, $f_1(5)=27$

各任务延迟：0, 1, 2, 16, 10； 最大延迟：16



调度2： $f_2(1)=0$, $f_2(2)=15$, $f_2(3)=23$, $f_2(4)=5$, $f_2(5)=27$

各任务延迟：0, 11, 12, 4, 10；最大延迟：12



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

算法 **Schedule**

输入：A, T, D

输出：f

1. 排序A使得 $d_1 \le d_2 \le \ldots \le d_n$
2. $f(1) \leftarrow 0$
3. $\mathrm{i} \leftarrow 2$
3. **while** $i \le n$ **do**
4. $\quad f(i) \leftarrow f(i-1)+t_{i-1}$ //任务i-1结束时刻是任务i开始时刻
5. $\quad i \leftarrow i+1$

设计思想：按完成时间从早到晚安排任务，没有空闲

交换论证：正确性证明

算法的解的性质：

(1) 没有空闲时间，没有逆序.

(2) 逆序 (i, j)：$f(i) < f(j)$ 且 $d_i > d_j$

引理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.

证：设 f 没有逆序，在 f 中具有相同完成时间的客户 $i_1, i_2, \ldots, i_k$ 必被连续安排. 在这 k 个客户中最大延迟是最后一个客户，被延迟的时间是

$$t_0 + \sum_{j=1}^{k} t_{i_j} - d$$

与 $i_1, i_2, \ldots, i_k$ 的排列次序无关.

交换论证

证明思想：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. 根据引理 1，这个解和算法的解具有相同的最大延迟.

证明要点

(1) 相邻逆序的存在性：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) 交换相邻的逆序 i 和 j，得到的解的调度仍旧最优.

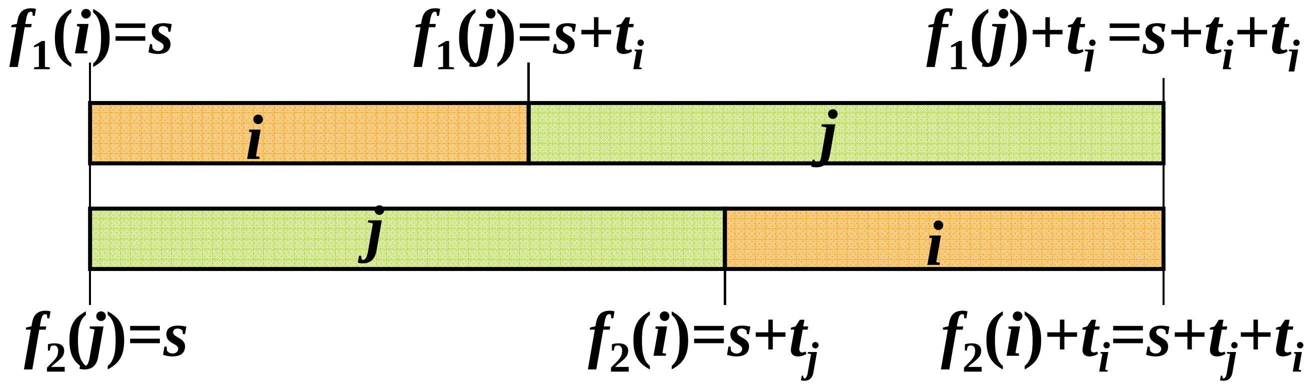
(3) 每次交换后逆序数减1，至多经过 $n(n-1)/2$ 次交换得到一个没有逆序的最 优调度.

交换相邻逆序 (i, j)不影响最优性

(1) 交换 i, j 对其他客户的延迟时间没影响

(2) 交换后不增加 j 的延迟

(3) i 在 f' 的延迟delay(f',i)小于 j 在 f 的延迟 delay(f,j), 因此小于f 的最大延迟 r



delay(f',i)$=s+t_j+t_i-d_i<$delya(f,j)$\leq r$

delay(f, j)$=s+t_i+t_j-d_j$

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

4.3 得不到最优解的处理方法

讨论对于哪些输入贪心法能得到最优解：输入条件
讨论贪心法的解最坏情况下与最优解的误差（见第8章）

找零钱问题

设有 n 种零钱， 重量分别为$w_1, w_2, \ldots, w_n$, 价值分别为 $v_1=1, v_2, \ldots, v_n$. 需要付的总钱数是 y .不妨设币值和钱数都为正整数. 问：如何付钱使得所付钱的总重最轻?

令选用第 i 种硬币的数目是 x_i，$i=1,2,\ldots,n$

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

$$\sum_{i=1}^{n} v_i x_i = y, \quad x_i \in \mathrm{N}, \quad i = 1,2,\ldots,n$$

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

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

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

Greedy算法

假设 $\frac{w_1}{v_1} \ge \frac{w_2}{v_2} \ge \dots \ge \frac{w_n}{v_n}$

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

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

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

n=1, 2 贪心法得到最优解

$n = 1$ 只有一种零钱，$F_1(y) = G_1(y)$， $F_2(y) = G_2(y)$

$n = 2$, 使用价值大的钱越多(x_2越大)，得到的解越好

$$\boxed{F_2(y) = \min_{0 \le x_2 \le \lfloor y/v_2 \rfloor} \{F_1(y - v_2 x_2) + w_2 x_2\}}$$

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

$n>2$得到最优解的判定条件

定理4.5　对每个正整数 k，假设对所有非负整数 y 有$G_k(y)=F_k(y)$，那么　$G_{k+1}(y)\le G_k(y) \Leftrightarrow F_{k+1}(y)=G_{k+1}(y)$

定理4.6　对每个正整数k，假设对所有非负整数 y 有$G_k(y)=F_k(y)$ 且存在 p 和 δ 满足

$$v_{k+1}= pv_k-\delta,\ \text{其中}\ 0\le\delta<v_k,\ v_k\le v_{k+1},\ p\text{为正整数},$$

则下面的命题等价：

(1) $G_{k+1}(y)=F_{k+1}(y)$　　对一切正整数y；

(2) $G_{k+1}(pv_k)=F_{k+1}(pv_k)$；

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

条件(3)需 $O(k)$ 时间验证 $G_{k+1}(y)=F_{k+1}(y)$，整个验证时间$O(n^2)$

实例

$$v_{k+1} = pv_k - \delta,\quad 0 \le \delta < v_k,\quad p \in Z^+$$
$$w_{k+1} + G_k(\delta) \le pw_k$$

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 $G_3(y) = F_3(y)$

$v_3=pv_2-\delta \Rightarrow p=3, \delta=1.$

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

$pw_2=3\times 1=3$

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

$v_4= pv_3-\delta \Rightarrow p=2, \delta=10$

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

$pw_3=2\times 1=2$

$w_4+G_3(\delta) > pw_3$

结论：$G_3(y)=F_3(y)$， 对于 $y=pv_3=28$，$G_4(y)>F_4(y)$

4.4 贪心法的典型应用

4.4.1 最优前缀码

二元前缀码

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

非前缀码的例子

a: **001**, b: **00**, c: **010**, d: **01**

解码的歧义，例如字符串 **0100001**

解码1: **01, 00, 001**   ***d, b, a***

解码2: **010, 00, 01**   ***c, b, d***

前缀码的二叉树及权值

前缀码： { 00000，00001，0001，001，01, 100, 101, 11}

频率： 00000: 5%， 000001: 5%， 0001: 10%， 001: 15%,

01: 25%，100: 10%，101: 10%，11: 20%

平均的二进制位数

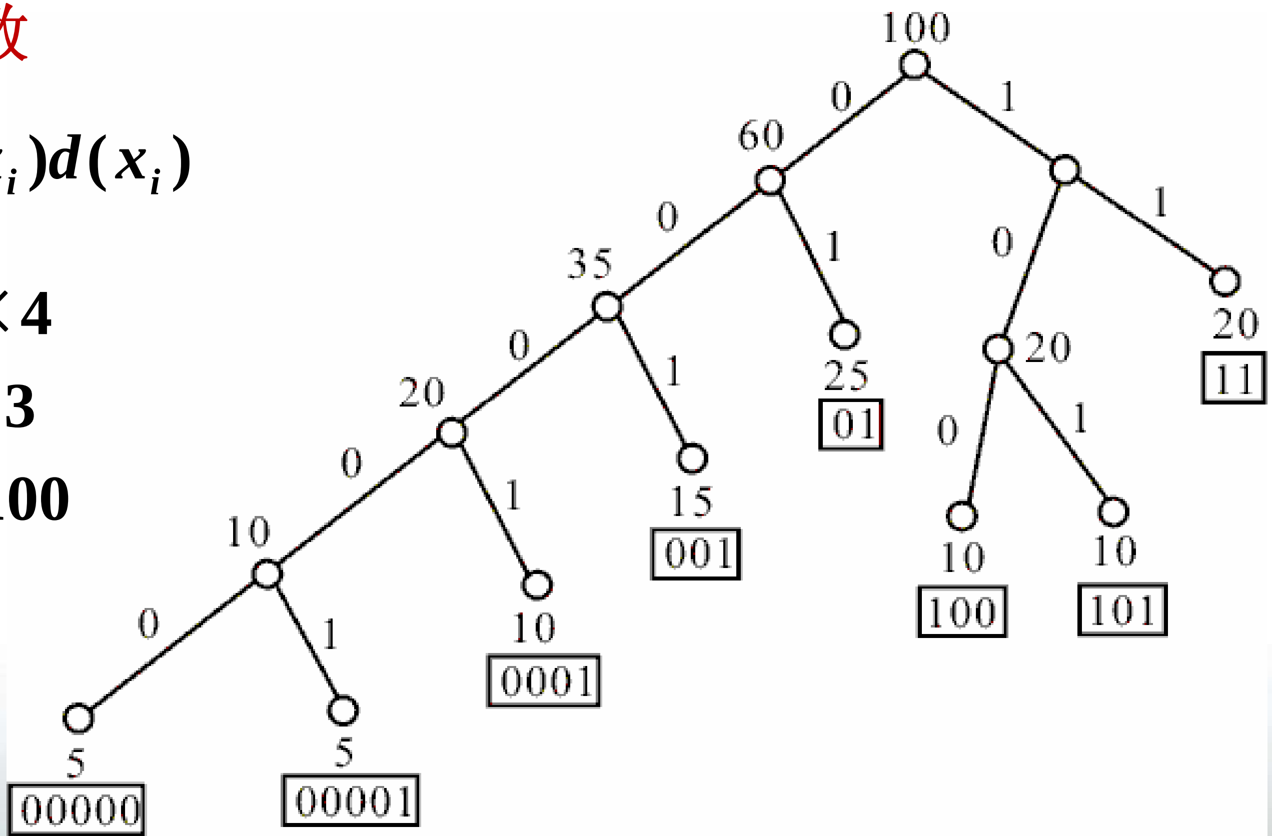
$$B=\sum_{i=1}^{n} f(x_i)d(x_i)$$

B=[(5+5)×5+10×4

+(15+10+10)×3

+(25+20)×2]/100

=2.85



最优前缀码

权值B最小

最优前缀码问题

问题：给定字符集 $C=\{x_1, x_2, \ldots , x_n\}$和每个字符的频率 $f(x_i)$, $i=1,2,\ldots,n$, 求关于C 的一个最优前缀码.

算法 Huffman(C)

输入：$C=\{x_1, x_2, \ldots , x_n\}$， $f(x_i)$, $i=1, 2, \ldots , n$.

输出：Q / /队列

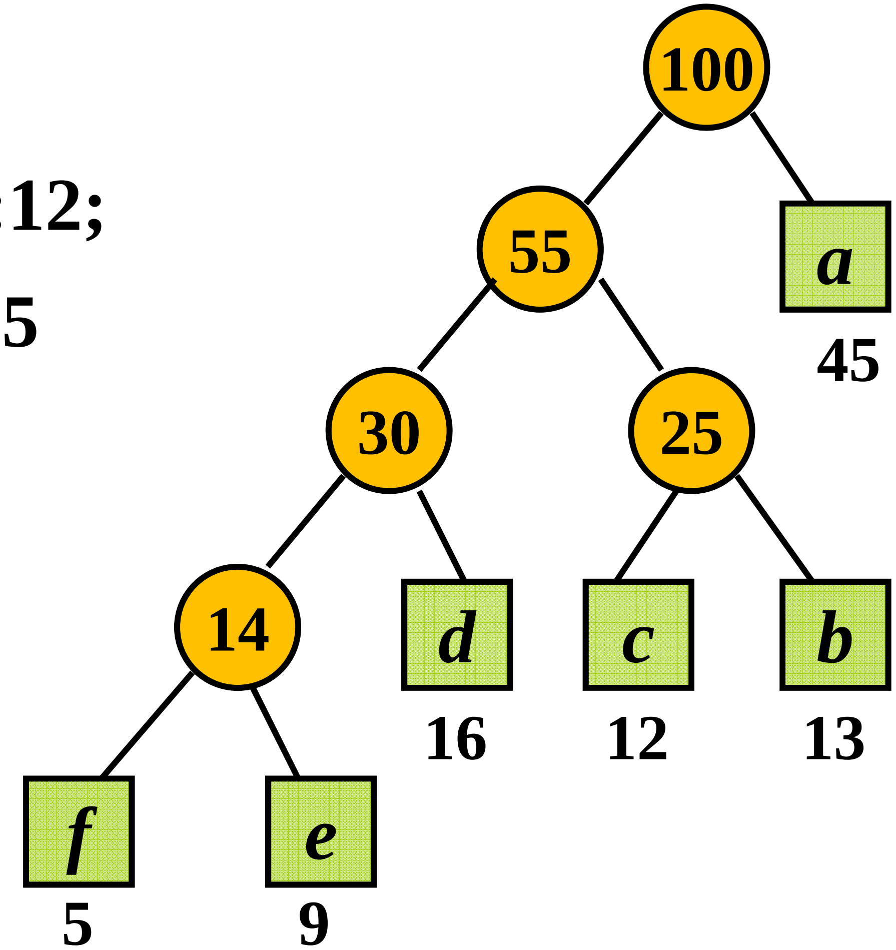
1. $n \leftarrow |C|$
2. $Q \leftarrow C$　　//按频率递增构成队列Q
3. for $i \leftarrow 1$ to $n-1$ do
4. 　$z \leftarrow$Allocate-Node()　　// 生成结点 z
5. 　$z.left \leftarrow Q$中最小元　　// 取出Q最小元作z的左儿子
6. 　$z.right \leftarrow Q$中最小元　// 取出Q最小元作z的右儿子
7. 　$f(z) \leftarrow f(x)+f(y)$
8. 　Insert(Q,z)　　　// 将 z 插入Q
9. return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

北京大学

实例

例如 *a*:45, *b*:13; *c*:12;

d:16; *e*:9; *f*:5



编码:

f--0000, *e*--0001,

d--001, *c*--010,

b—011, *a*--1

平均位数:

$4\times(0.05+0.09)$

$+3\times(0.16+0.12+0.13)+1\times0.45= 2.24$

算法正确性证明:引理1

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.

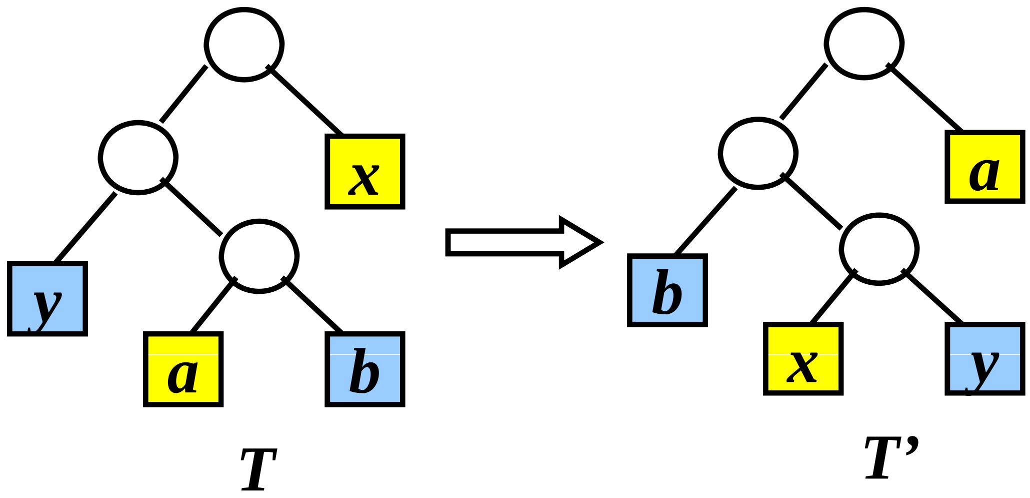
$T\rightarrow T'$

$f[x] \le f[a]$

$f[y] \le f[b]$

a与 x 交换

b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

引理 设 T 是二元前缀码所对应的二叉树，$\forall\ x, y \in T$, x, y是树叶兄弟，z 是 x, y 的父亲，令$T' = T - \{x, y\}$, 且令 z 的频率 $f(z) = f(x)+f(y)$，T'是对应于二元前缀码 $C' = (C-\{x, y\}) \cup \{z\}$ 的二叉树，那么

$$B(T)=B(T')+f(x)+f(y).$$

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

$$d_T(x) = d_T(y) = d_{T'}(z) +1.$$

$$B(T) = \sum_{i \in T} f(i) d_T(i) = \sum_{i \in T, i \neq x,y} f(i) d_T(i) + f(x) d_T(x) + f(y) d_T(y)$$

$$= \sum_{i \in T', i \neq z} f(i) d_{T'}(i) + f(z) d_{T'}(z) + (f(x) + f(y))$$

$$= B(T') + f(x) + f(y)$$

证明：归纳法

定理 **Haffman** 算法对任意规模为n（$n \geq 2$）的字符集C 都得到关于C 的最优前缀码的二叉树.

归纳基础 $n=2$，字符集$C=\{x_1,x_2\}$，**Huffman**算法得到的代码是0和1，是最优前缀码.

归纳步骤 假设**Huffman**算法对于规模为k 的字符集都得到最优前缀码. 考虑规模为$k+1$的字符集$C=\{x_1, x_2, ..., x_{k+1}\}$, 其中$x_1, x_2 \in C$是频率最小的两个字符. 令

$$C' = (C - \{x_1,x_2\}) \cup \{z\}, \quad f(z)=f(x_1)+f(x_2)$$

根据归纳假设，**Huffman**算法得到一棵关于字符集C'、频率$f(z)$和$f(x_i)$（$i=3,4,...,k+1$）的最优前缀码的二叉树T'.

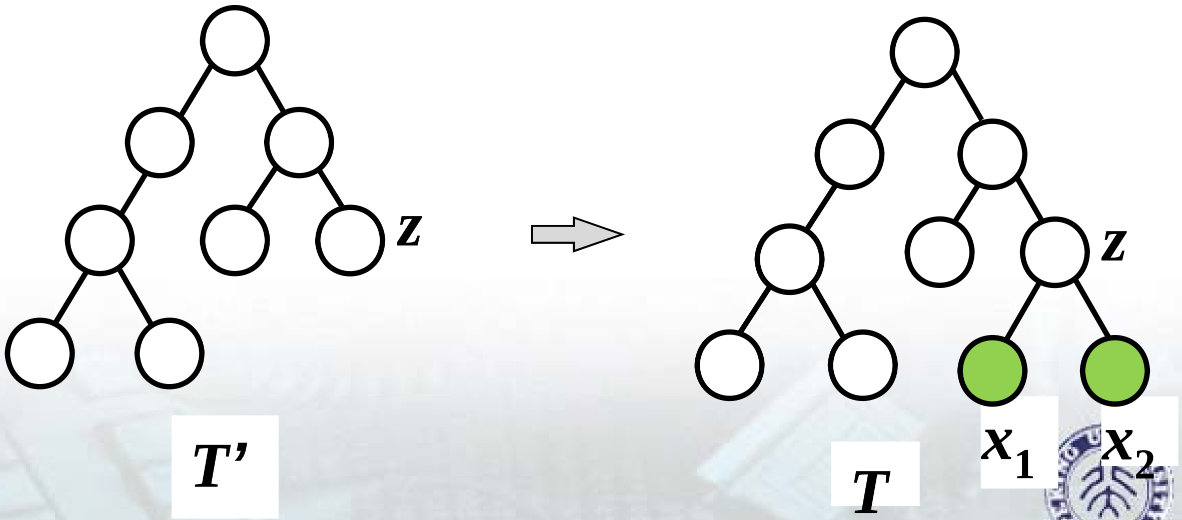
证明：归纳法(续)

把x_1和 x_2作为 z 的儿子附加到T'上，得到树T，那么T是关于字符集$C=(C'-\{z\})\cup\{x_1,x_2\}$ 的最优前缀码的二叉树.

如若不然，存在更优的树T^*. 根据引理1，其最深层树叶是x_1, x_2， 且$B(T^*)<B(T)$. 去掉T^*中的x_1和x_2，根据引理2，所得二叉树$T^{*\prime}$满足

$$B(T^{*\prime})=B(T^*)-(f(x_1)+f(x_2))<B(T)-(f(x_1)+f(x_2))=B(T')$$

与T'是一棵关于C'的最优前缀码的二叉树矛盾.



Huffman树应用:文件归并

问题：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.

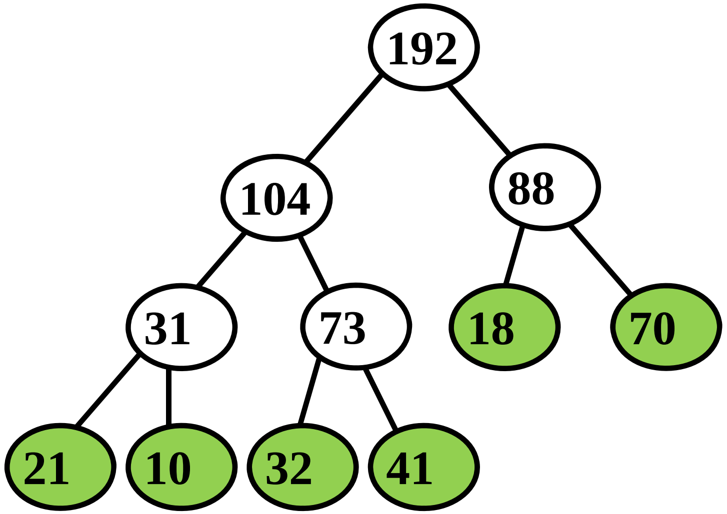
归并过程对应于二叉树：文件为树叶. f_i与f_j 归并的文件是它们的父结点.

归并代价(最多的比较次数)：结点 f_i与 f_j 归并代价为 f_i+f_j-1.
总的代价：每个文件(树叶)的深度乘以文件大小之和再减掉归并次数 $n-1$

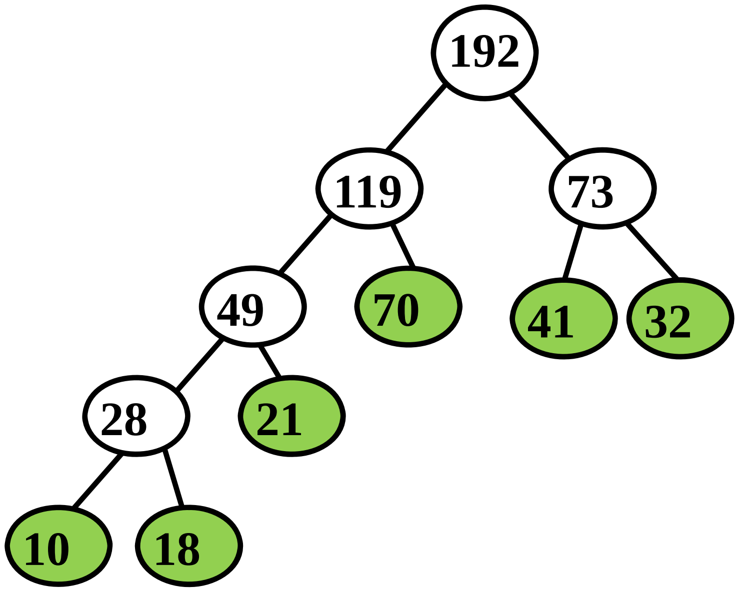
$$\sum_{i \in S} d(i) f_i - (n-1)$$

实例

实例：$S = \{\ 21,10,32,41,18,70\ \}$



顺序归并



Huffman树归并

代价

顺序归并： $(21+10+32+41)\times3+(18+70)\times2-5=483$

Huffman树归并：$(10+18)\times4+21\times3+(70+41+32)\times2-5=456$

4.4.2 最小生成树

无向连通带权图$G=(V,E,W)$，$w(e)\in W$是边e的权. G的一棵生成树是包含了G的所有顶点的树, 树中各边的权之和称为树的权，具有最小权的生成树称为G的最小生成树.

命题4.1 设G是 n阶连通图，那么

(1) T是G 的生成树当且仅当 T 有$n-1$条边.

(2) 如果T是G的生成树，$e\notin T$，那么$T\cup\{e\}$含有一个圈 (回路).

问题：给定连通带权图G，求G的一棵最小生成树.

算法：**Prim算法**和**Kruskal算法**

Prim算法

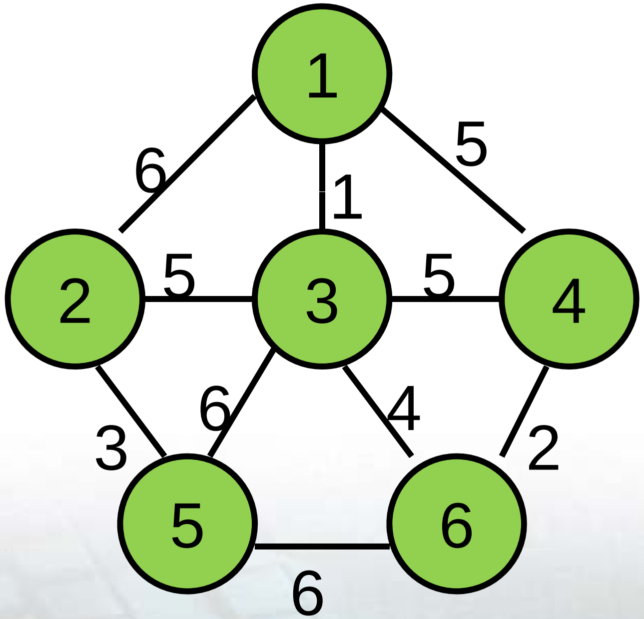
算法 Prim(*G*,*E*,*W*)

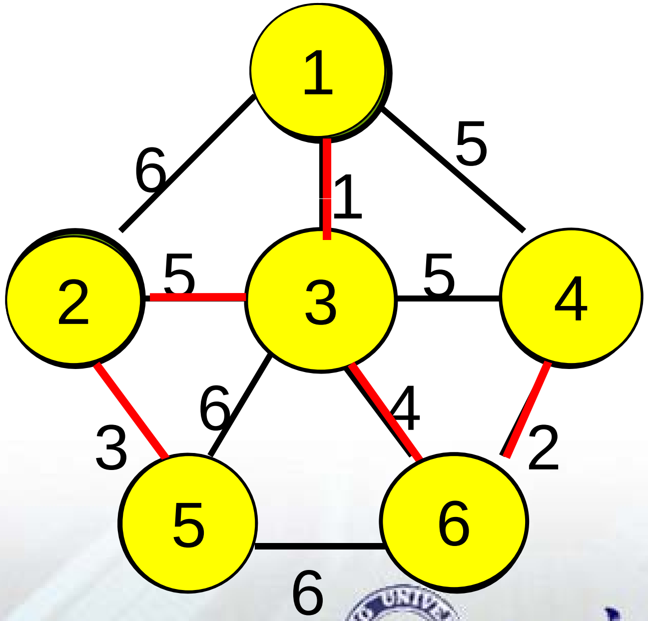
1． *S*←{1}

2． while *V*–*S* ≠ ∅ do

3． 　从*V*–*S*中选择 *j* 使得 *j* 到 *S* 中顶点的边权最小

4． 　*S*←*S*∪{*j*}





北京大學

正确性证明

对步数归纳

定理：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

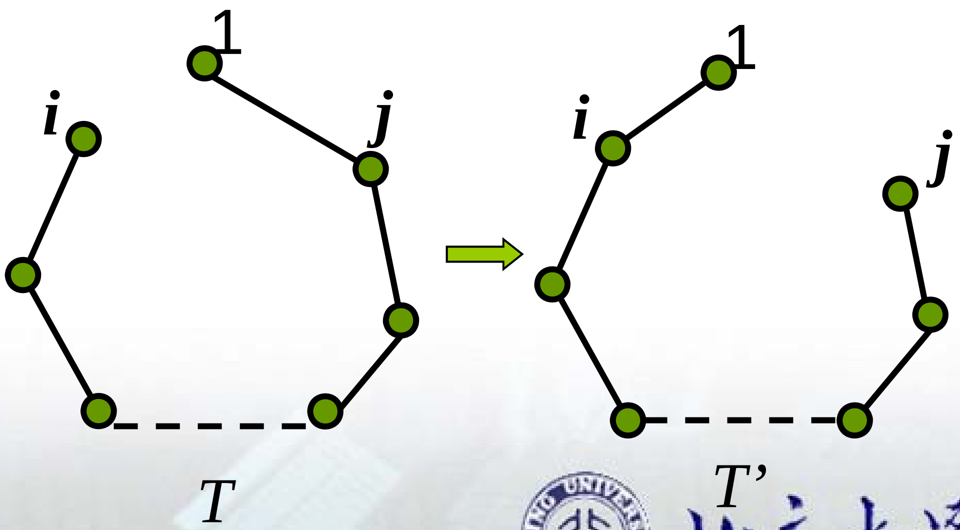
归纳基础：k=1, 存在一棵最小生成树 T 包含边e={1,i}, 其中{1,i}是所有关联 1 的边中权最小的.

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$W(T')\leq W(T)$.



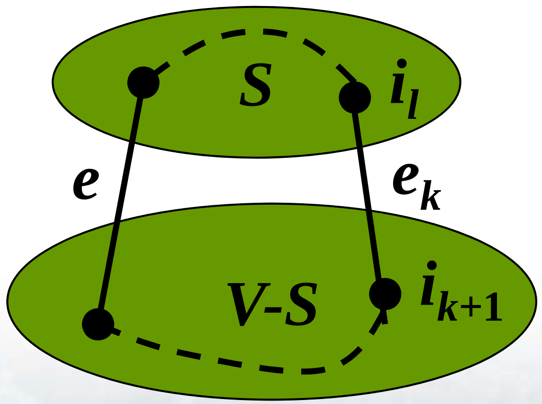
正确性证明(续)

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$e_1,e_2,\ldots,e_k$, $W(T^*)\le W(T)$.

算法时间：$T(n)=O(n^2)$

Kruskal算法

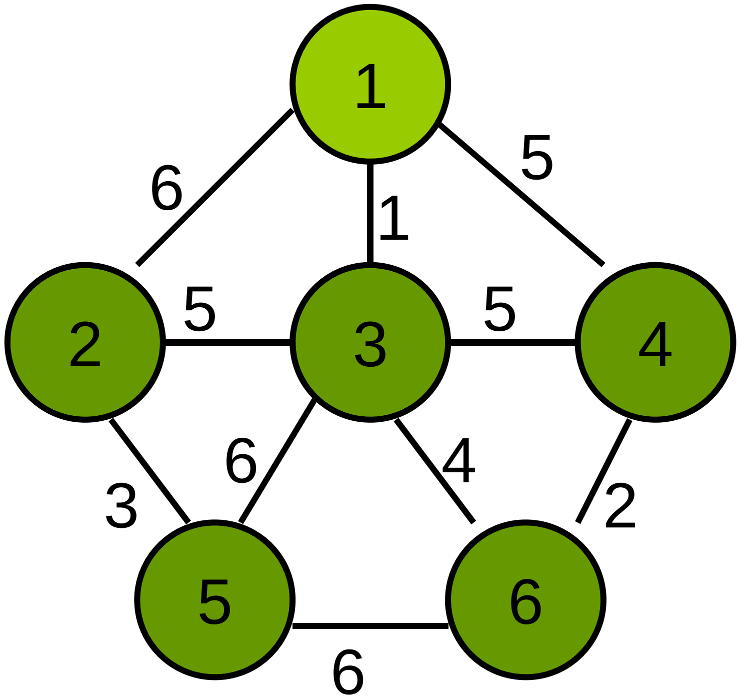
算法4.6 Kruskal

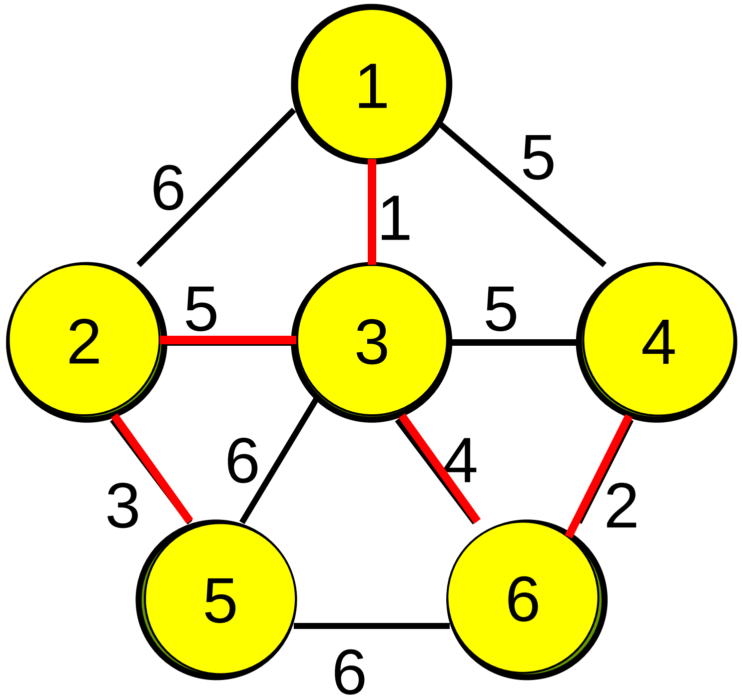
输入：连通图G // 顶点数n，边数m

输出：G的最小生成树

1. 按权从小到大排序G中的边，使得$E=\{e_1, e_2, \ldots, e_m\}$
2. $T \leftarrow \varnothing$
3. **repeat**
4. $e \leftarrow E$中的最短边
5. **if** e的两端点不在同一个连通分支
6. **then** $T \leftarrow T \cup \{e\}$
7. $E \leftarrow E - \{e\}$
8. **until** T包含了$n-1$条边

实例





Kruskal算法正确性证明

命题：对于任意 $n>1$, 算法对 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'. 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.

否则存在G 的含边e 的最小生成树T^*， $W(T^*)<W(T)$. (如果 $e\notin T^*$, 在T^*中加边e，形成回路. 去掉回路中任意别的边所得生成树的权仍旧最小). 在T^*中短接 e 得到G' 的生成树$T^*-\{e\}$, 且

$$W(T^*-\{e\})=W(T^*)-w(e)<W(T)-w(e)=W(T'),$$

与T' 的最优性矛盾.

算法的实现与时间复杂度

数据结构：

建立**FIND**数组，**IND[*i*]** 是结点 ***i*** 的连通分支标记.

(1) 初始**FIND[*i*]=*i*.**

(2) 两个连通分支合并，则将较小分支结点的**FIND**值更新为较大分支的标记

时间复杂度：

(1) 每个结点至多更新**log*n***次，建立和更新**FIND**数组的总时间为$O(n\log n)$

(2) 算法时间为

$$O(m\log m) + O(n\log n) + O(m) = O(m\log n)$$

边排序　　FIND数组　其他

4.4.3 单源最短路径

给定带权有向网络$G=(V,E,W)$,每条边$e=\langle i,j\rangle$的权$w(e)$为非负实数，表示从i到j 的距离. 源点$s\in V$，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

$short[u]$：从 s 到 u 的最短路径的长

$dist[u] \geq short[u]$

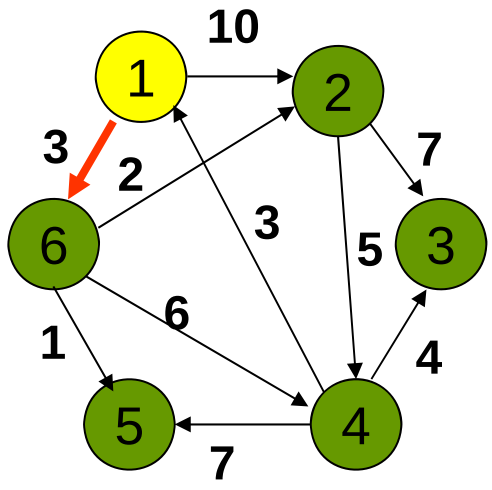
Dijkstra算法

算法 Dijkstra

1. $S \leftarrow \{s\}$
2. $dist[s] \leftarrow 0$
3. **for** $i \in V-\{s\}$ **do**
4. $\quad dist[i] \leftarrow w(s,i)$ // 如果s到i没有边, $w(s,i)=\infty$
5. **while** $V-S \neq \varnothing$ **do**
6. $\quad$从$V-S$中取出具有相对S的最短路径的顶点j
7. $\quad S \leftarrow S \cup \{j\}$;
8. $\quad$**for** $i \in V-S$ **do**
9. $\quad\quad$**if** $dist[j]+w(j,i)<dist[i]$
10. $\quad\quad$**then** $dist[i] \leftarrow dist[j]+w(j,i)$ // 更新$dist[i]$

实例

输入：$G=<V,E,W>$，源点 1
$V=\{1, 2, 3, 4, 5, 6\}$

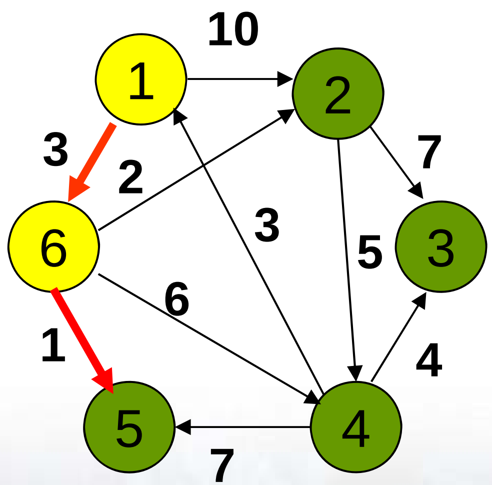


$S=\{1\}$,

$dist[1]=0$

$dist[2]=10$, $dist[6]=3$

$dist[3]=dist[4]=dist[5]=\infty$



$S=\{1,6\}$,

$dist[1]=0$, $dist[6]=3$

$dist[2]=5$, $dist[4]=9$, $dist[5]=4$

$dist[3]=\infty$

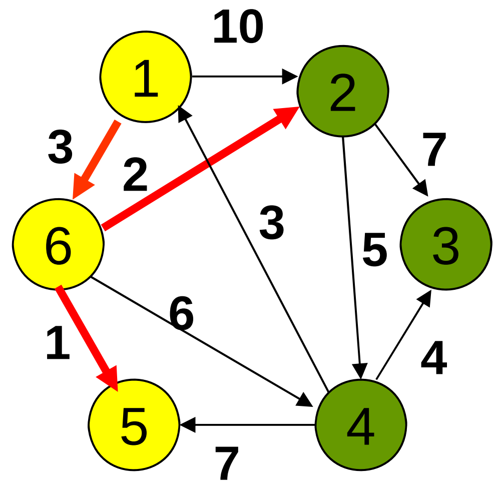
实例（续）

S={1,6,5},

dist[1]=0, *dist*[6]=3, *dist*[5]=4

dist[2]=5, *dist*[4]=9,

dist[3]=∞



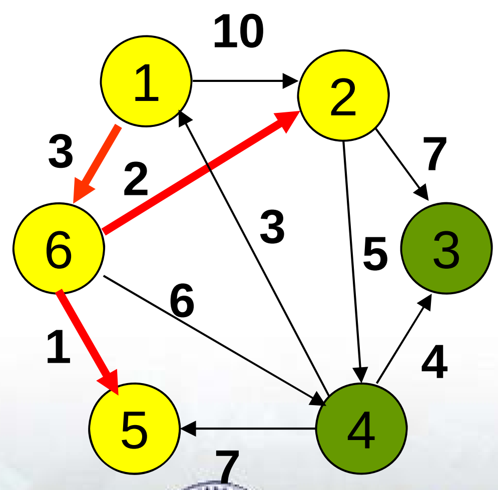
S={1,6,5,2},

dist[1]=0, *dist*[6]=3, *dist*[5]=4

dist[2]=5

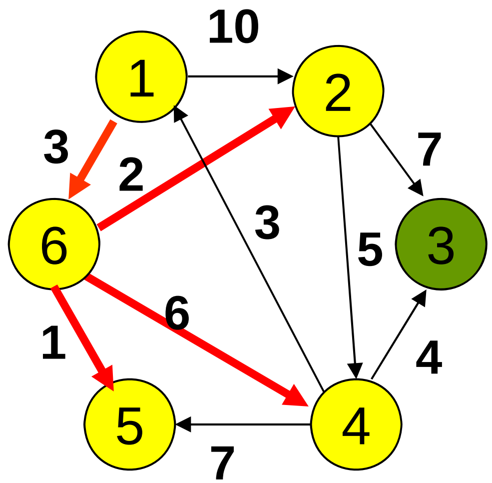
dist[3]=12

dist[4]=9

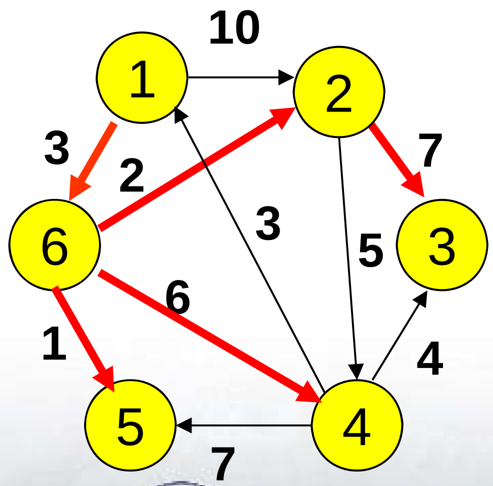


实 例（续）

S={1,6,5,2,4},
dist[1]=0, *dist*[6]=3, *dist*[5]=4
dist[2]=5, *dist*[4]=9,
dist[3]=12



S={1,6,5,2,4,3},
dist[1]=0, *dist*[6]=3, *dist*[5]=4
dist[2]=5, *dist*[4]=9
dist[3]=12



解：
short[1]=0, short[2]=5,
short[3]=12, short[4]=9,
short[5]=4, short[6]=3.

算法正确性证明

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.

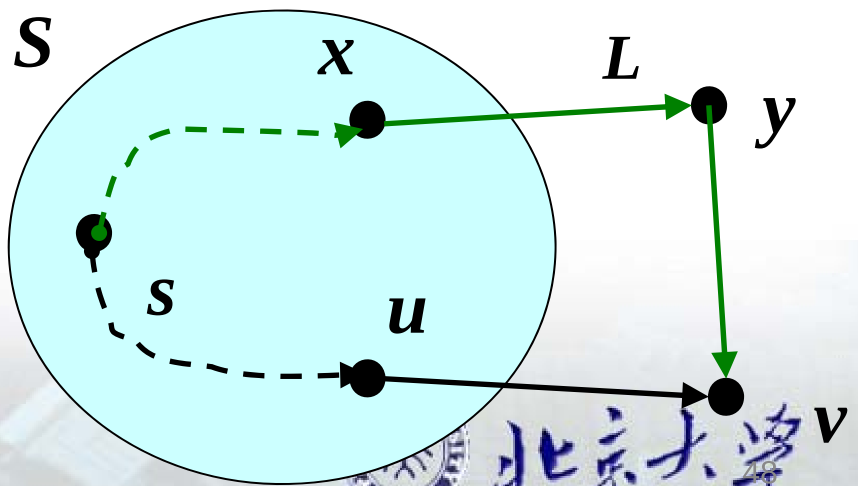
归纳步骤　假设命题对于k 为真. 考虑 k+1步, 选择顶点v (边{u,v}). 假若存在另一条 s-v 路径 L (绿色)，最后一次出S 的顶点为 x, 在这次从S 中出来后
经过V–S 的第一个顶点为 y.

$dist[v] \le dist[y]$　　//v先被选

$\le dist[y]+d(y,v) \le L$

$dist[v]=short[v]$

时间复杂度 $T(n)=O(n^2)$



贪心法小结

(1) 适用于组合优化问题. 求解过程是多步判 断. 判断的依据是局部最优策略，使目标值达到最大(或最小)，与前面的子问题计算结果无关.

(2) 局部最优策略的选择是算法正确性的关键.

(3) 正确性证明方法：数学归纳法、交换论证. 使用数学归纳法主要通过对算法步数或者问题规模进行归纳. 如果要证明贪心策略是错误的，只需举出反例.

(4) 自顶向下求解，通过选择将问题归约为小的子问题.

(5) 如果贪心法得不到最优解，可以对问题的输入进行分析或者估计算法的近似比.

(6) 如果对原始数据排序之后，贪心法往往是一轮处理，时间复杂度和空间复杂度低.

第5章 回溯与分支限界

5.1 回溯算法的基本思想和适用条件

5.2 回溯算法的设计步骤

5.3 回溯算法的效率估计和改进途径

5.4 分支限界

- 5.4.1 背包问题
- 5.4.2 最大团问题
- 5.4.3 货郎问题
- 5.4.4 圆排列问题
- 5.4.5 连续邮资问题

5.1 回溯算法的基本思想和适用条件

5.1.1 几个典型例子

四后问题

0-1背包问题

货郎问题（TSP）

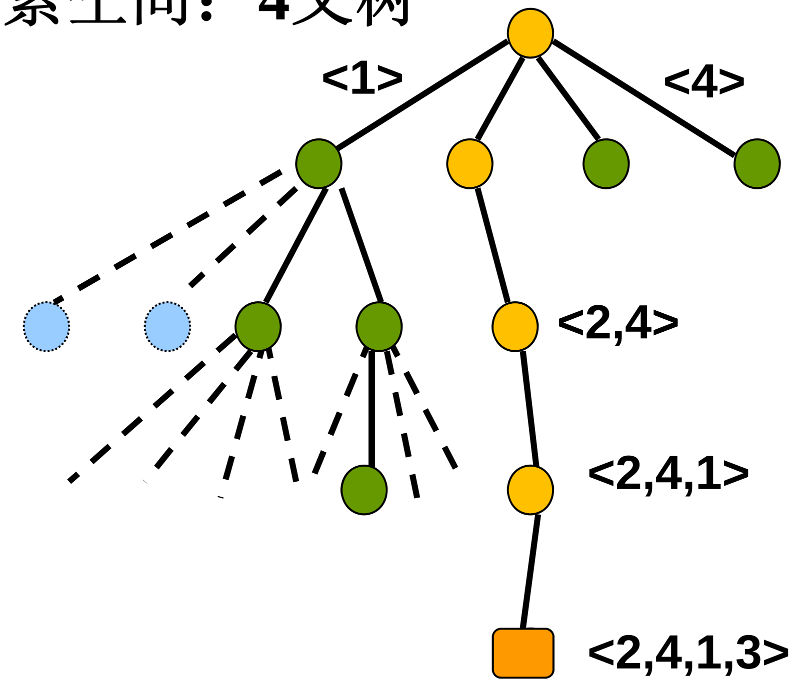
5.1.2 回溯算法的适用条件

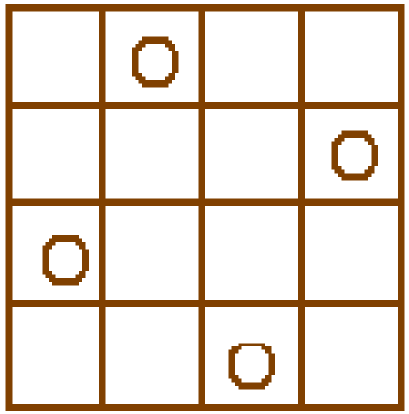
5.1.1几个典型例子

例1　*n*后问题

4后问题：解是一个4维向量，$<x_1, x_2, x_3, x_4>$（放置列号）

搜索空间：4叉树





8后问题：解是一个8维向量，$<x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8>$

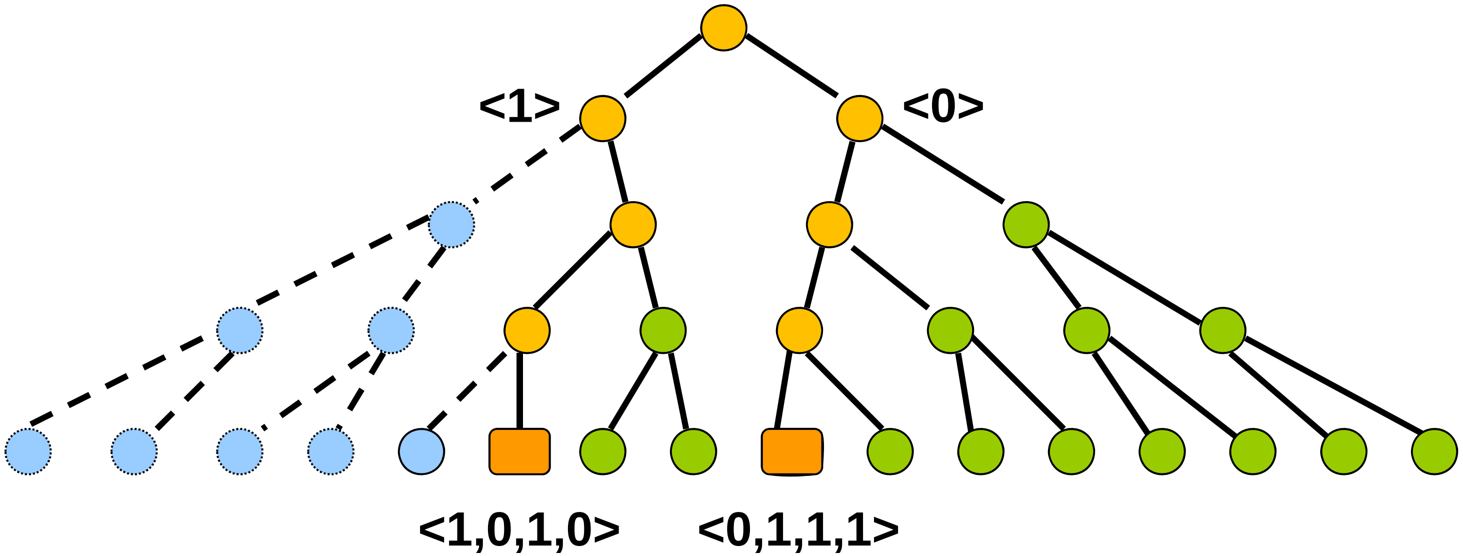
搜索空间：8叉树，　一个解：<1,3,5,2,4,6,8,7>

例2：0-1背包问题

实例：V={12,11,9,8}, W={8,6,4,3}, B=13

结点：向量$\langle x_1, x_2, x_3, \ldots, x_k \rangle$（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> 可行解： x_1=0, x_2=1, x_3=1, x_4=1. 价值:28，重量:13

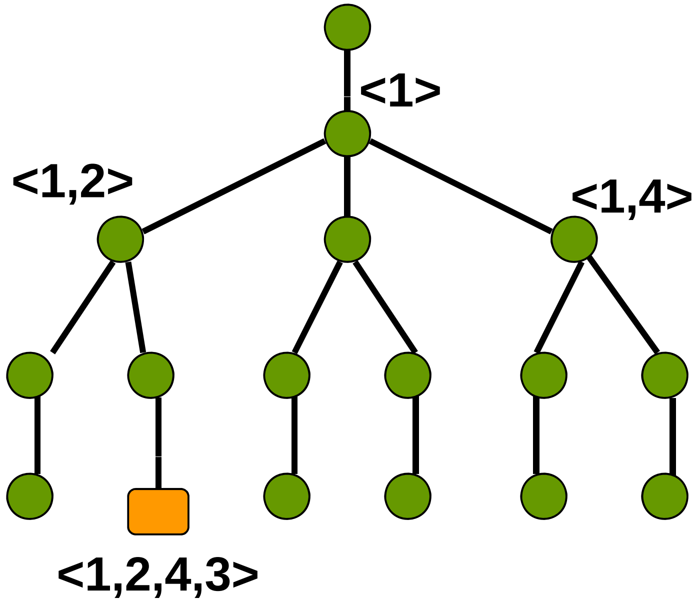
<1,0,1,0> 可行解： x_1=1, x_2=0, x_3=1, x_4=0. 价值:21，重量:12

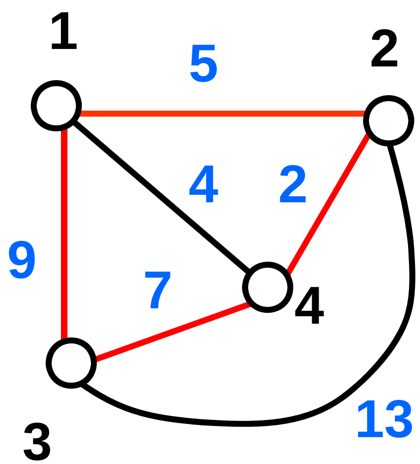
例3：货郎问题

$<i_1,i_2,\ldots,i_n>$为巡回路线

搜索空间：排列树, $(n\text{-}1)!$片树叶

实例：





<1,2,4,3> 对应于巡回路线：1→2 →4 →3 →1

长度：5+2+7+9=23

回溯算法的基本思想

(1) 适用问题：求解搜索问题和优化问题

(2) 搜索空间：树，结点对应部分解向量，树叶对应可行解

(3) 搜索过程：采用系统的方法隐含遍历搜索树

(4) 搜索策略：深度优先，宽度优先，函数优先，宽深结合等

(5) 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

(6) 结点状态：动态生成

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)

(7) 存储：当前路径

5.1.2 回溯算法的适用条件

设 $P(x_1, x_2, \ldots, x_i)$ 为真表示向量 $<x_1, x_2, \ldots, x_i>$ 满足某个性质 (n后问题中i 个皇后放置在彼此不能攻击的位置）

多米诺性质：

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

例4　求不等式的整数解

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

$P(x_1, \ldots, x_k)$: 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 $x_3=3-x_3'$,

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

5.2 回溯算法的设计步骤

(1) 定义搜索问题的解向量和每个分量的取值范围

解向量为 $<x_1, x_2, \ldots, x_n>$

确定 x_i 的可能取值的集合为 X_i , $i = 1, 2, \ldots, n$.

(2) 当 $x_1, x_2, \ldots, x_{k-1}$ 确定以后计算 x_k 取值集合S_k, $S_k \subseteq X_k$

(3) 确定结点儿子的排列规则

(4) 判断是否满足多米诺性质

(5) 搜索策略----深度优先、宽度优先等

(6) 确定每个结点分支约束条件

(7) 确定存储搜索路径的数据结构

回溯算法的递归实现

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)**

输入：n

输出：所有的解

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

回溯算法的迭代实现

迭代算法 Backtrack

输入：n

输出：所有的解

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

例5 装载问题

n个集装箱装上2艘载重分别为c_1和c_2的轮船，w_i为集装箱i的重量，且

$$\sum_{i=1}^{n} w_i \le c_1 + c_2$$

问是否存在一种合理的装载方案将n个集装箱装上轮船？如果有，给出一种方案.

求解思路：令第一船装载量为W_1，用回溯算法求使W_1-c_1达到最小的装载方案<$x_1, x_2, \ldots, x_n$>，如果 $\sum_{i=1}^{n} w_i - W_1 \le c_2$ 回答Yes, 否则回答No.

问题满足多米诺性质，搜索策略：深度优先

算法

算法5.4 Loading (W,c_1),

输入：集装箱重量W=<w_1,w_2,…,w_n>, c_1是第一条船的载重

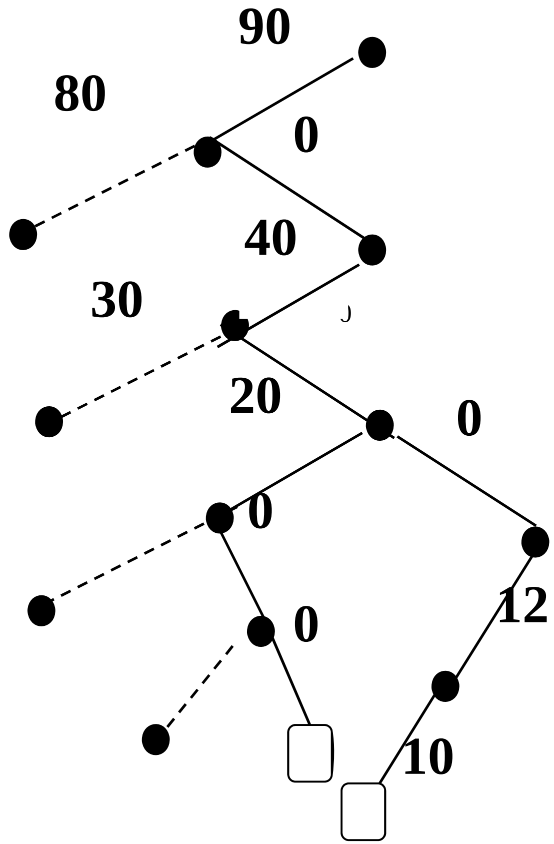
输出：使船1装载量最大的方案<x_1,x_2,…,x_n>，其中x_i=0,1,…,n

1. Sort(W);　　　//对w_1,w_2,…,w_n按照从大到小排序
2. $B \leftarrow \infty$; $best \leftarrow \infty$; $i \leftarrow 1$;
3. while $i \le n$ do
4. 　　if 装入 i 后重量不超过c_1
5. 　　then $B \leftarrow B - w_i$; $x[i] \leftarrow 1$; $i \leftarrow i+1$;
6. 　　else $x[i] \leftarrow 0$; $i \leftarrow i+1$;
7. if　$B<best$ then 记录解；$Best \leftarrow B$;
8. Backtrack(i)；
9. if i=1 then return 最优解
10. else goto 3 .

实例

算法5.5 Backtrack(i)

1. while $i>1$ and $x[i]=0$ do
2. $\quad i \leftarrow i-1$;
3. if $x[i]=1$
4. then $x[i]\leftarrow 0$; $B\leftarrow B+w_i$; $i\leftarrow i+1$.



$W=<90,80,40,30,20,12,10>$,

$c_1=152$, $c_2=130$

复杂性：$W(n)=O(2^n)$

例6 图的着色问题

问题：给定无向连通图G和m种颜色，用这些颜色给图的顶点着色，每个顶点一种颜色. 要求是：G 的每条边的两个顶点着不同颜色. 给出所有可能的着色方案；如果不存在着这样的方案，则回答 "**No**".

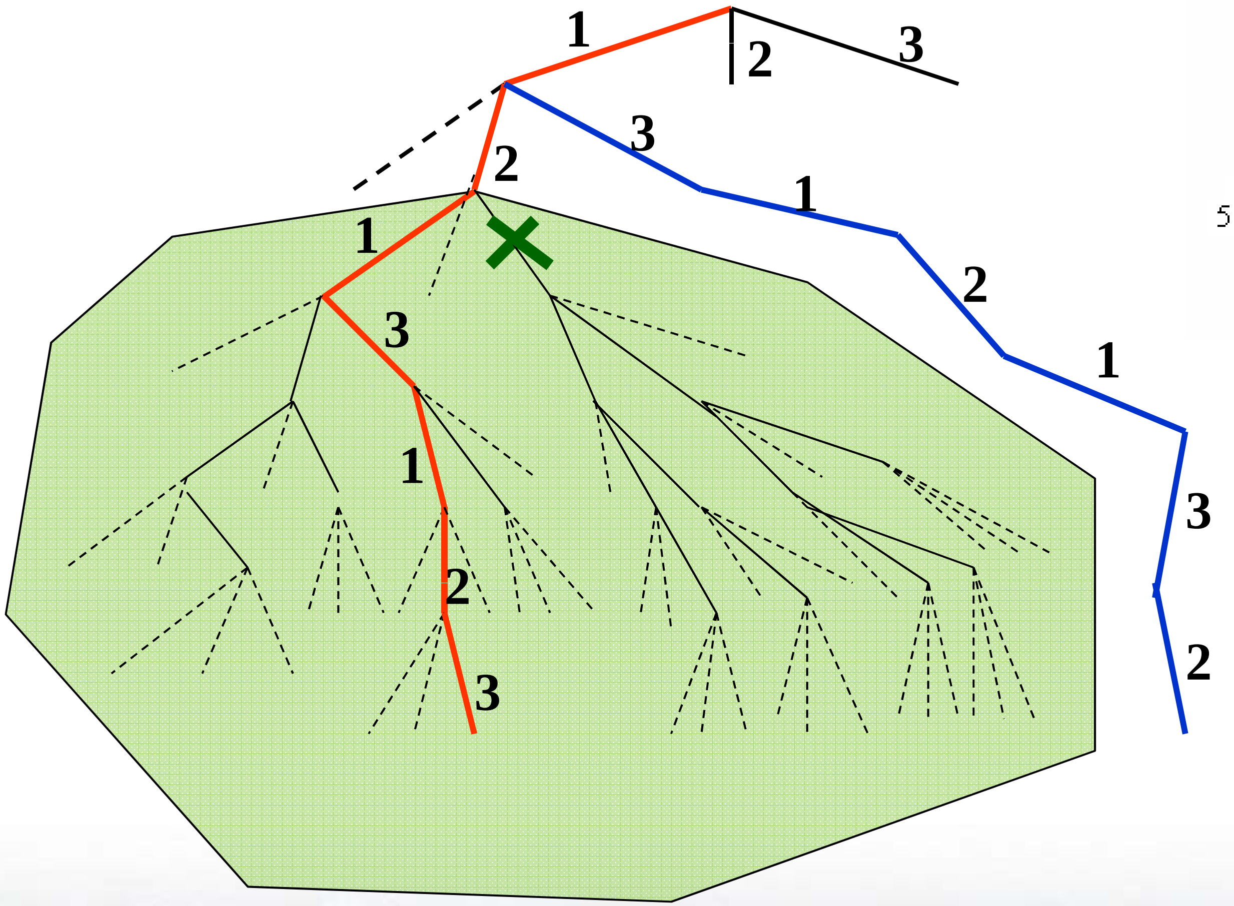
则搜索空间为深度n 的m叉完全树. 将颜色编号为$1,2,\ldots,m$，结点$<x_1, x_2, \ldots, x_k>$：$x_1, x_2, \ldots, x_k \in \{1, 2, .., m\}, 1 \le k \le n$，表示顶点1着颜色 x_1，顶点2着颜色 x_2，…，顶点 k 着颜色 x_k.

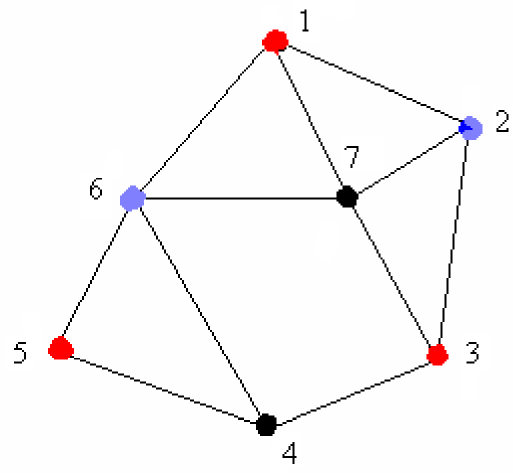
约束条件：该顶点邻接表中的顶点与该顶点没有同色；

搜索策略：深度优先

时间：$O(nm^n)$

实例





<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.

5.3搜索树结点数估计

计数搜索树中的结点，**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.

算法实现

Monte Carlo

输入：n,t 为正整数，n为皇后数，t为抽样次数

输出：sum, 即t 次抽样路径长度的平均值

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

子过程

m为输出——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 $k \leq n$ 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. $k \leftarrow k+1$

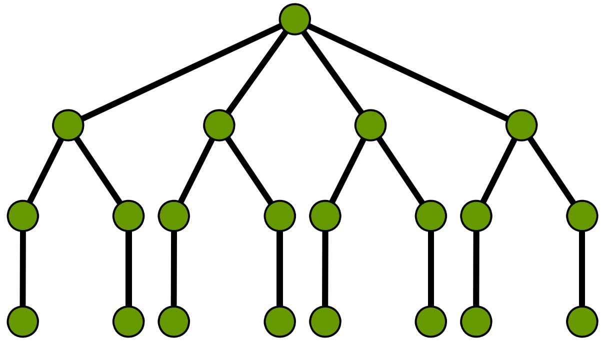
实例

估计四后搜索树的结点数

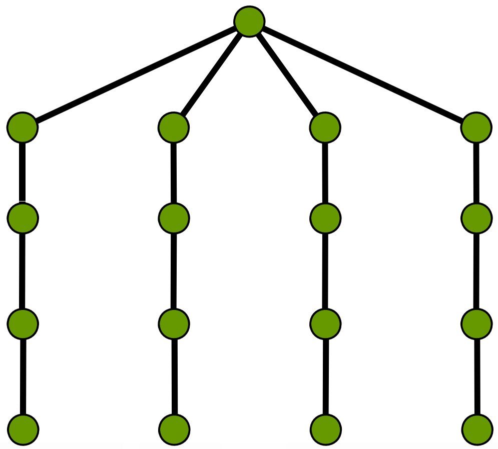
case1．<1,4,2>：1+4+4×2+4×2=21

case2．<2,4,1,3>：4×4+1=17

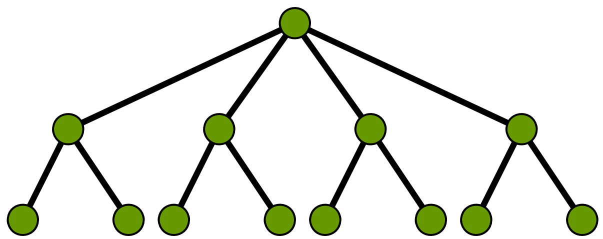
case3．<1,3>：1+4×1+4×2=13



Case1: <1,4,2>



Case2: <2,4,1,3>



Case3: <1,3>

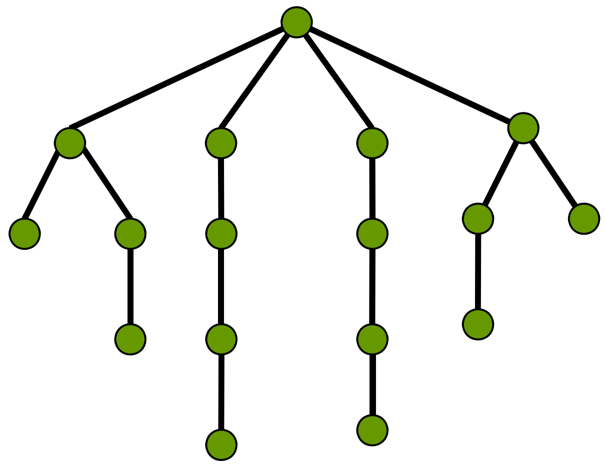
估计结果

假设 4 次抽样测试：

case1:1次，case2:1次，case3:2次，

平均结点数＝(21×1+17×1+13×2)/4=16

搜索空间访问的结点数为17



搜索空间

影响算法效率的因素

最坏情况下的时间$W(n)=(p(n)f(n))$

其中 $p(n)$ 为每个结点时间，$f(n)$为结点个数

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

求解时间为

$$kc2^{\frac{n}{k}}+T$$

5.4 分支限界

组合优化问题的相关概念

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

5.4. 1 背包问题

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

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

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

分支限界技术(极大化)

- 设立代价函数
  - 函数值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
  - 父结点的代价不小于子结点的代价
- 设立界
  - 代表当时已经得到的可行解的目标函数的最大值
  - 界的设定初值可以设为0
  - 可行解的目标函数值大于当时的界，进行更新
- 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

实例：背包问题

背包问题的实例：

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$
$$2x_1 + 3x_2 + 4x_3 + 7x_4 \le 10$$
$$x_i \in \mathrm{N}, i = 1,2,3,4$$

对变元重新排序使得

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

排序后实例 $\max 9x_1 + 5x_2 + 3x_3 + x_4$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10$$
$$x_i \in \mathrm{N}, i = 1,2,3,4$$

代价函数与分支策略确定

结点$<x_1, x_2, \ldots, x_k>$ 的代价函数

$$\sum_{i=1}^{k} v_i x_i + \left(b - \sum_{i=1}^{k} w_i x_i\right)\frac{v_{k+1}}{w_{k+1}} \qquad \text{若对某个 } j > k \text{ 有 } b - \sum_{i=1}^{k} w_i x_i \ge w_j$$

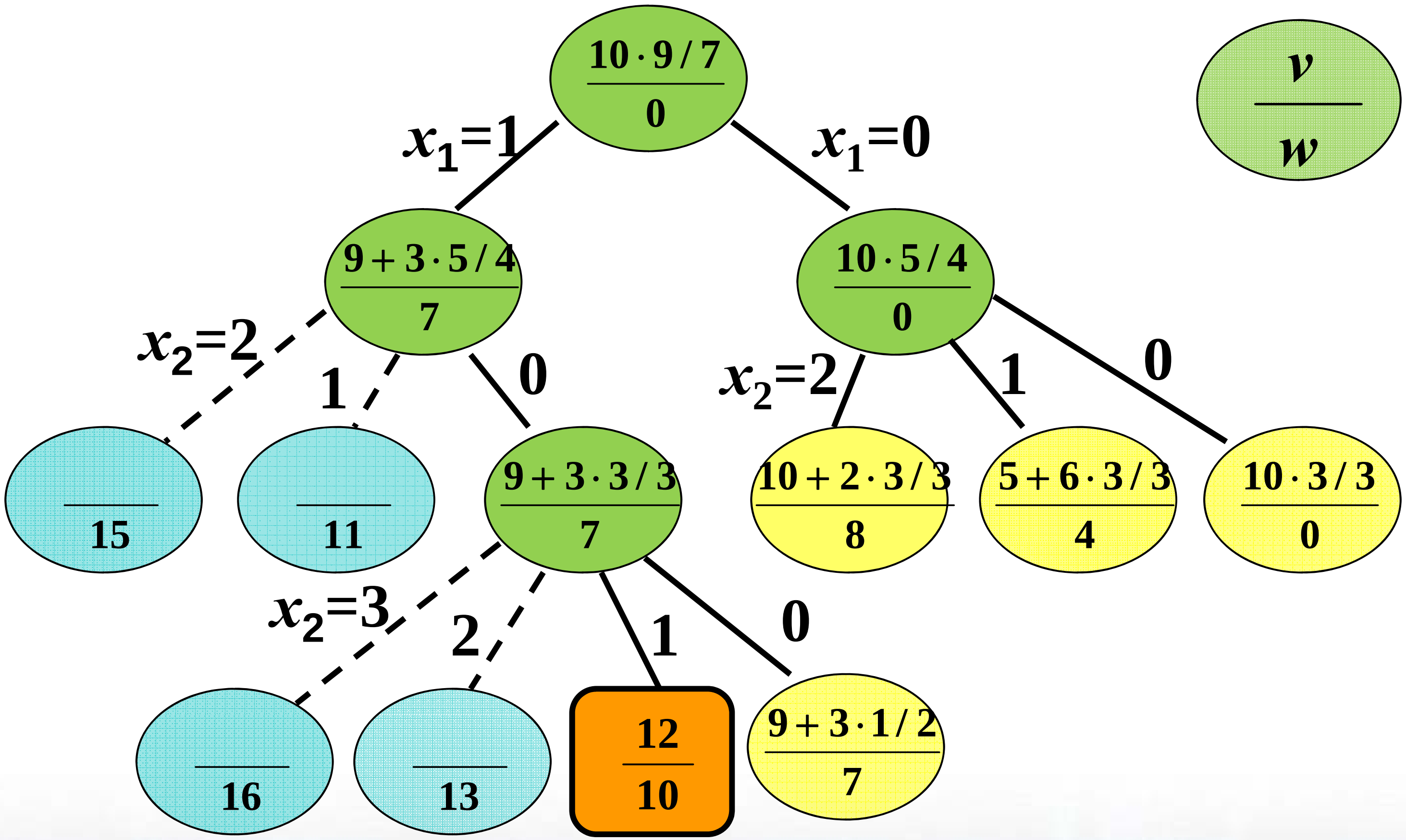
$$\sum_{i=1}^{k} v_i x_i \qquad \text{否则}$$

分支策略----深度优先

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

$$7x_1+4x_2+3x_3+2x_4\le 10,\quad x_i\in \mathrm{N}, i=1,2,3,4$$

实例



5.4.2 最大团问题

问题：给定无向图G=<V, E>, 求G中的最大团.

相关知识：

无向图G = <V, E>,

G的子图：G'= <V',E'>, 其中$V'\subseteq V$, $E'\subseteq E$,

G的补图：$\breve{G}$ =<V, E'>, E'是E关于完全图边集的补集

G中的团：G 的完全子图

G 的点独立集：G 的顶点子集A，且$\forall u,v\in A$, $\{u,v\}\notin E$.

最大团：顶点数最多的团

最大点独立集：顶点数最多的点独立集

命题：U是G 的最大团当且仅当U是$\breve{G}$ 的最大点独立集

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

$$F = C_n + n - k$$

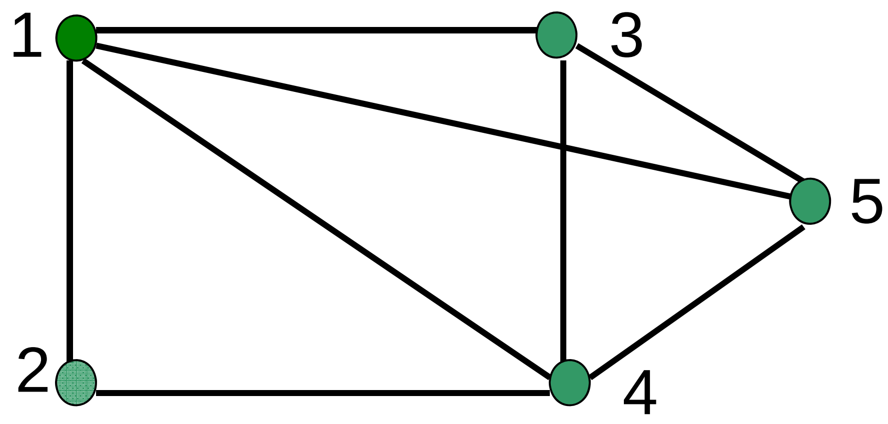
其中 C_n 为目前团的顶点数（初始为0），

k 为结点层数

时间：$O(n2^n)$

北京大学

最大团的实例



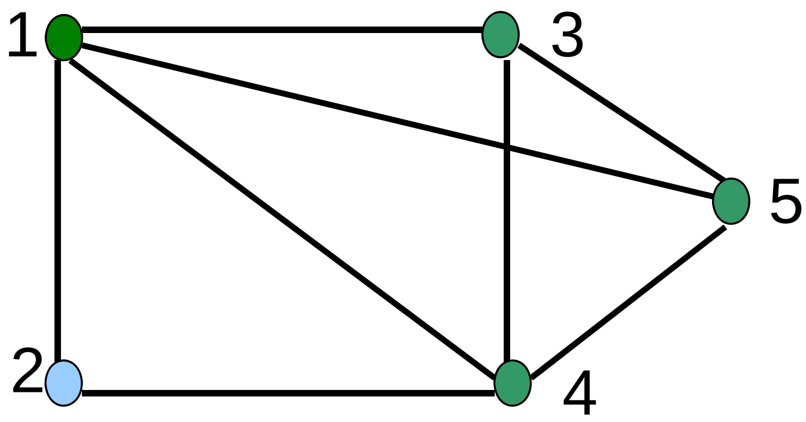
顶点编号顺序为 **1, 2, 3, 4, 5,**

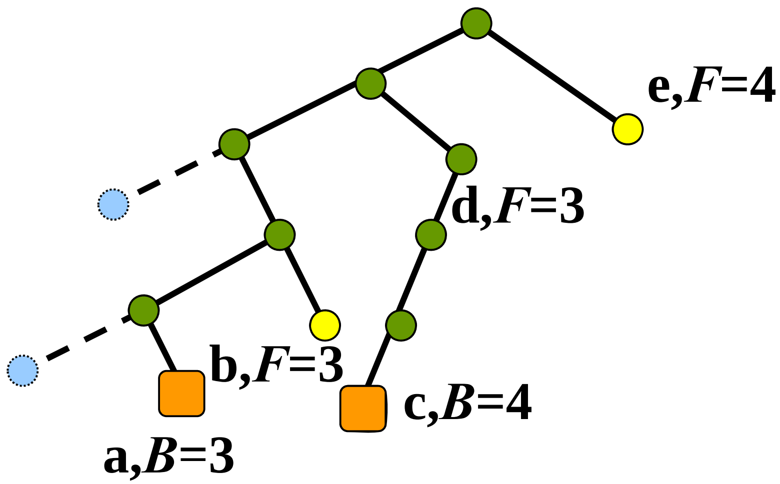
对应 x_1, x_2, x_3, x_4, x_5, x_i=**1** 当且仅当 i 在团内

分支规定左子树为**1**，右子树为**0.**

B 为界，***F*** 为代价函数值.

实例求解





a：得第一个极大团 **{ 1, 2, 4 }**, 顶点数为**3**, 界为**3**;

b：代价函数值 ***F*** **= 3**, 回溯;

c：得第二个极大团**{1, 3, 4, 5}**, 顶点数为**4**, 修改界为**4**;

d：不必搜索其它分支, 因为***F*** **= 4**, 不超过界;

e：***F*** **= 4**, 不必搜索**.**

最大团为 **{1, 3, 4, 5}**, 顶点数为 **4.**

5.4.3 货郎问题

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.

货郎问题的类型：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 步路线

算法设计

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

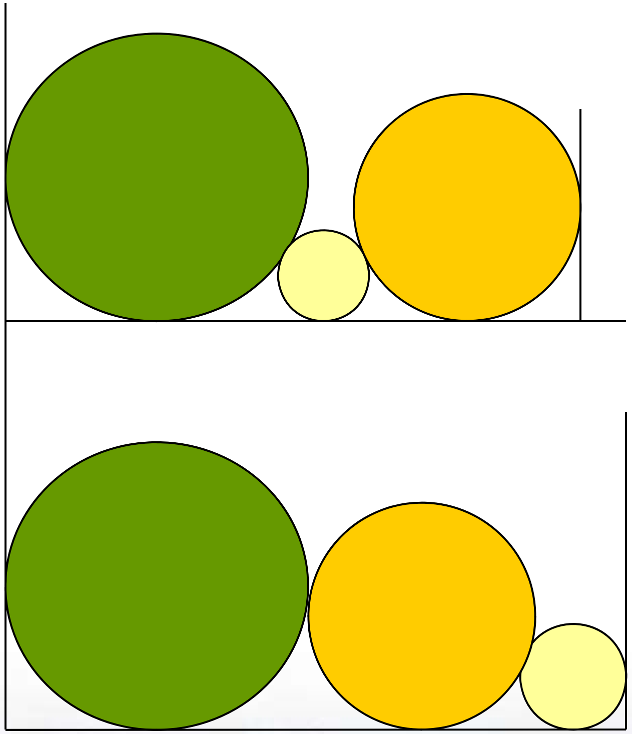
$$L = \sum_{j=1}^{k} d_j + \sum_{i_j \notin B} l_{i_j}$$

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$O(n)$

5.4.4 圆排列问题

问题：给定n个圆的半径序列, 将各圆与矩形底边相切排列, 求具有最小长度 l_n 的圆的排列顺序.



解为$<i_1, i_2, \ldots, i_n>$为1, 2, … , 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

代价函数符号说明

k：算法完成第 k 步，已经选择了第1— k 个圆

r_k：第 k 个圆的半径

d_k：第 k–1 个圆到第 k 个圆的圆心水平距离，k>1

x_k：第 k 个圆的圆心坐标，规定 x_1=0,

l_k：第 1— k 个圆的排列长度

L_k：放好 1—k 个圆以后，对应结点的代价函数值

$L_k \leq l_n$

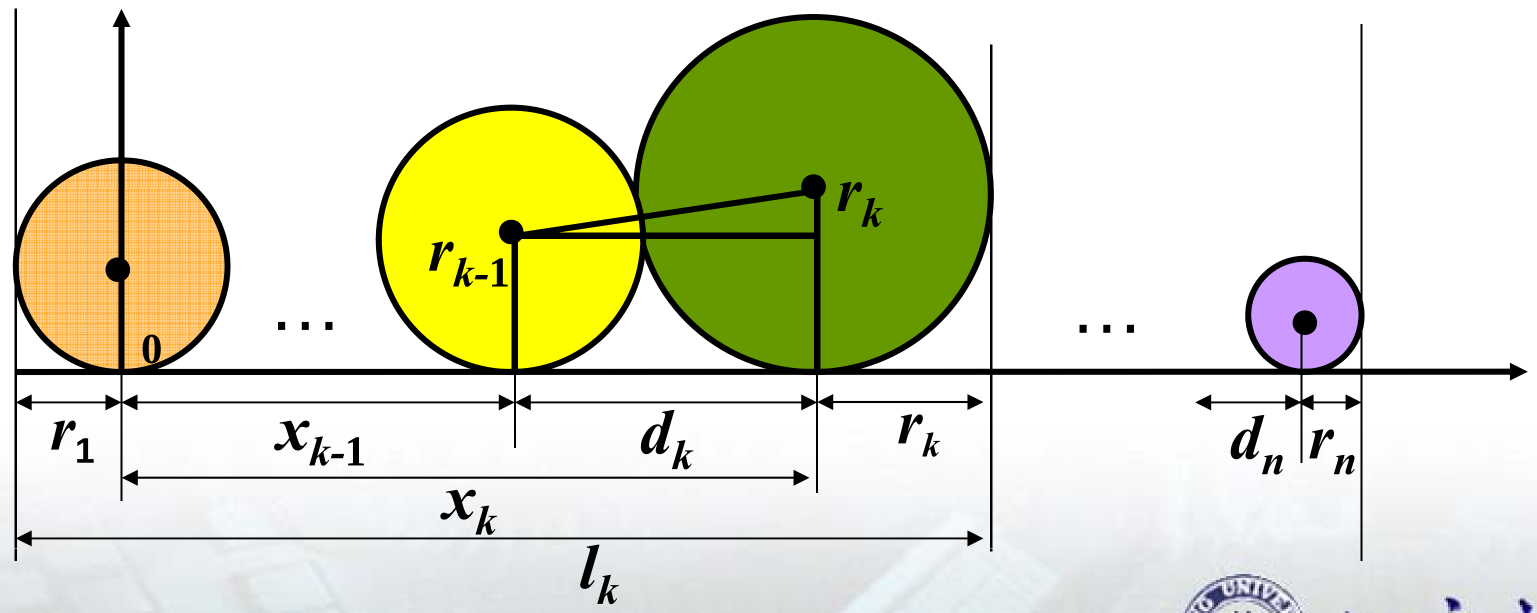
有关量的计算

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

$$x_k = x_{k-1}+d_k, \qquad l_k = x_k + r_k + r_1$$

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

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



代价函数

排列长度是l_n，L是代价函数：

$$
\begin{aligned}
l_n &= x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + \ldots + 2\sqrt{r_{n-1} r_n} + r_n + r_1 \\
&\geq x_k + 2(n-k)r + r + r_1
\end{aligned}
$$

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

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

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

时间：$O(n\, n!) = O((n+1)!)$

实例：计算过程

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

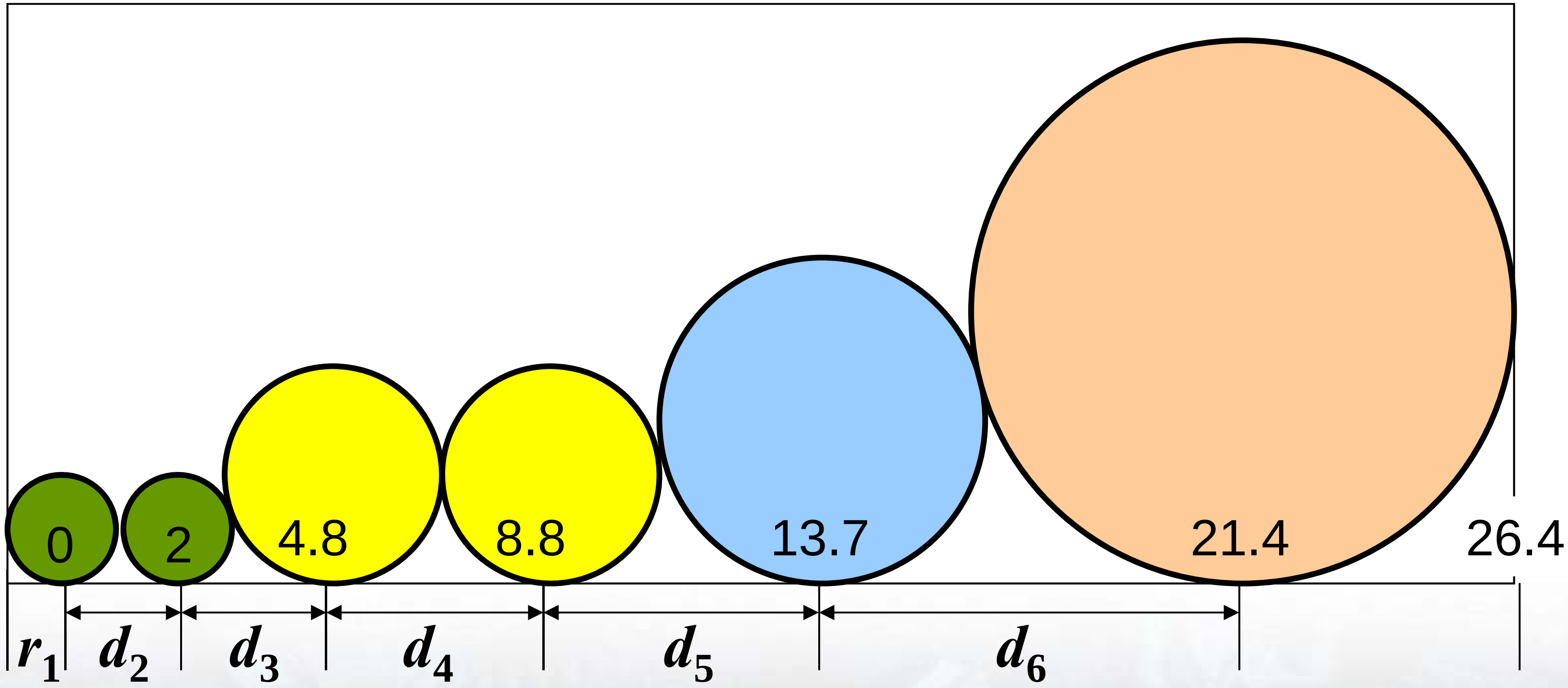
k	r_k	d_k	x_k	l_k	L_k
1	1	0	0	2	12
2	1	2	2	4	12
3	2	2.8	4.8	7.8	19.8
4	2	4	8.8	11.8	19.8
5	3	4.9	13.7	17.7	23.7
6	5	7.7	21.4	27.4	27.4

实例：图示

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

取排列 <1, 2, 3, 4, 5, 6>, 半径排列为：1, 1, 2, 2, 3, 5，

最短长度 $l_6 = 27.4$



5.4.5 连续邮资问题

问题：给定n种不同面值的邮票，每个信封至多m张， 试给出邮票的最佳设计，使得从1开始，增量为1的连续邮资区间达到最大？

实例：n=5，m=4，

面值 $X_1=<1,3,11,15,32>$，邮资连续区间为{ 1, 2, …,70 }

面值 $X_2=<1,6,10,20,30>$，邮资连续区间为{1, 2, 3, 4}

可行解 ：$<x_1, x_2, \ldots, x_n>$，$x_1=1$， $x_1< x_2< \ldots< x_n$

约束条件：在结点$<x_1, x_2, \ldots, x_r>$处，邮资最大连续区间为{ 1, … , r_i }，x_{i+1}的取值范围是{ x_i+1, … , r_i+1}

r_i 的计算

$y_i(j)$：用至多m张面值x_i的邮票加上$x_1, x_2, \ldots, x_{i-1}$面值的邮票贴 j 邮资时的最少邮票数，则

$$y_i(j) = \min_{1 \le t \le m}\{t + y_{i-1}(j - tx_i)\}$$

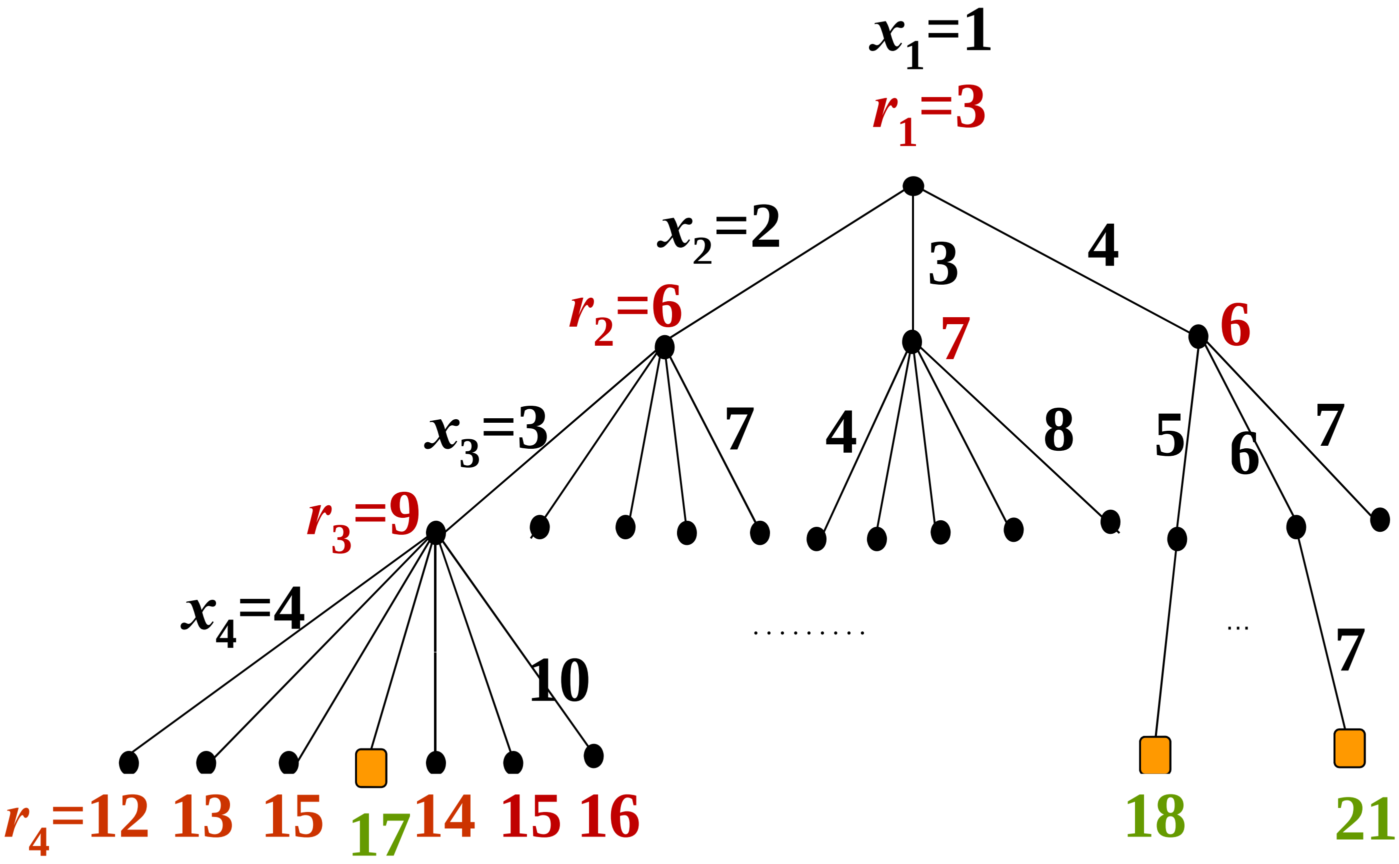
$$y_1(j) = j$$

$$r_i = \min\{j \mid y_i(j) \le m, y_i(j+1) > m\}$$

搜索策略：深度优先

界：max，m张邮票可付的连续区间的最大邮资

实例：n=4,m=3



解：X=<1,4,6,7>，最大连续区间为{1, …, 21}

回溯算法小结

(1) 适应于求解组合搜索问题（含组合优化问题）
(2) 求解条件：满足多米诺性质
(3) 解的表示：解向量，求解是不断扩充解向量的过程
(4) 回溯条件：
   搜索问题-约束条件
   优化问题-约束条件+代价函数
(5) 算法复杂性：最坏情况为指数，空间代价小
(6) 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
(7) 分支策略（深度优先、宽度优先、宽深结合、优先函数）

第6章算法分析与问题的计算复杂度

6.1 平凡下界

6.2 直接计数求解该问题所需要的最少运算

6.3 决策树

6.4 检索算法的时间复杂度分析

6.5 排序算法的时间复杂度分析

6.6 选择算法的时间复杂度分析

算法正确性

- 正确性 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.

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
- 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
  - 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

基于时间的最优性

- 含义：指求解某问题算法类中效率最高的算法
- 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

(1) 设计算法A, 求$W(n)$，得到算法类最坏情况下时间复杂度的一个上界

(2) 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，得到该算法类最坏情况下时间复杂度的一个下界

(3) 如果$W(n)=F(n)$或$W(n)=\Theta(F(n))$, 则A是最优的.

(4) 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$W'(n) = F'(n)$ 或者$W'(n) = \Theta(F'(n))$

6.1 平凡下界

算法的输入规模和输出规模是它的平凡下界

例1

问题：写出所有的n阶置换

求解的时间复杂度下界为$\Omega(n!)$

例2

问题：求n次实系数多项式多项式在给定x的值

求解的时间复杂度下界为$\Omega(n)$

例3

问题：求两个$n\times n$ 矩阵的乘积

求解的时间复杂度下界是$\Omega(n^2)$

6.2 直接计数最少运算数

例4 找最大

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. $i \leftarrow i+1$;

$W(n)=n-1$

以比较作为基本运算的算法类的上界：n-1

找最大问题的复杂度

下界：在 n 个数的数组中找最大的数, 以比较做基本运算的算法类中的任何算法在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.

6.3 决策树
（Decision Tree）

二叉树的性质

命题1 在二叉树的 t 层至多 2^t 个结点

命题2 深度为 d 的二叉树至多 $2^{d+1}-1$ 个结点.

命题3 n个结点的二叉树的深度至少为$\lfloor \log n \rfloor$.

命题4 设 t 为二叉树的树叶个数，d 为树深，如果树的每个内结点都有2个儿子，则 $t \leq 2^d$.

6.4 检索算法时间复杂度分析

检索问题：给定按递增顺序排列的数组 L (项数 $n \geq 1$)和数 x，如果 x 在 L 中, 输出 x 的下标; 否则输出0.

算法1 顺序检索

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$1/(2n+1)$

$$W(n)=n$$

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

北京大学

二分检索最坏时间复杂度

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

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, $1 \le k < n$, 命题为真, 则

$$
\begin{aligned}
W(n) &= 1 + W\left(\left\lfloor \tfrac{n}{2} \right\rfloor\right) \\
&= 1 + \left\lfloor \log \left\lfloor \tfrac{n}{2} \right\rfloor \right\rfloor + 1 \\
&= \begin{cases} \lfloor \log n \rfloor + 1 & n\text{为偶数} \\ \lfloor \log(n-1) \rfloor + 1 & n\text{为奇数} \end{cases} \\
&= \lfloor \log n \rfloor + 1
\end{aligned}
$$

二分检索的平均时间复杂度

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

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

求和

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

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

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

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

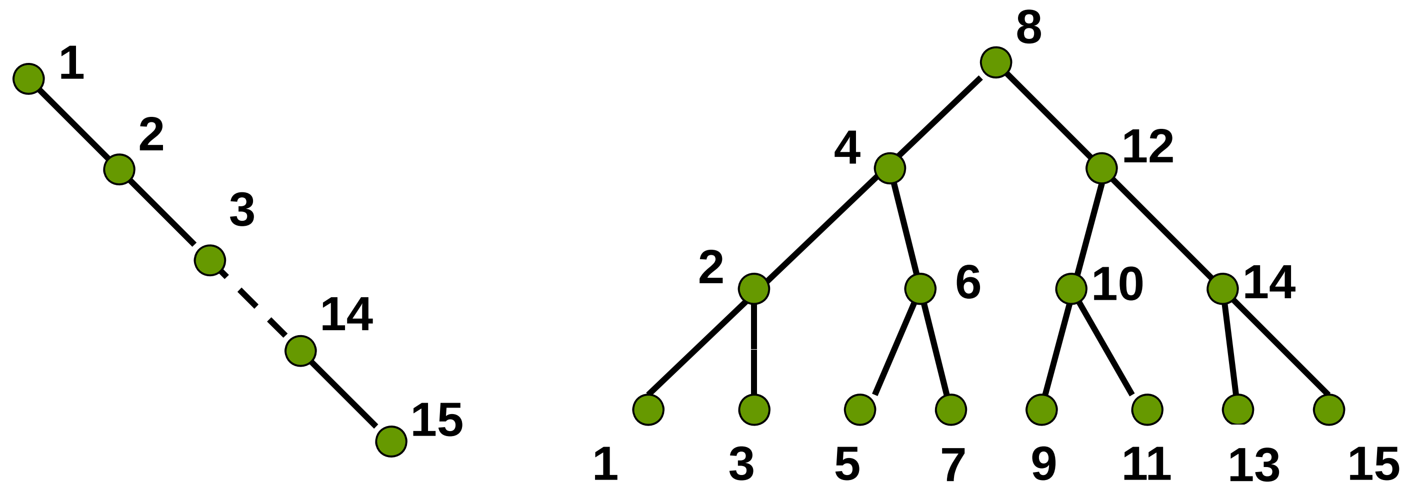
检索问题的决策树

设*A*是一个检索算法, 对于给定输入规模 ***n***, ***A***的一棵决策树是一棵二叉树, 其结点被标记为1, 2, ... , ***n***, 且标记规则是:

- 根据算法*A*，首先与 ***x*** 比较的 ***L*** 的项的下标标记为树根.
- 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

改进顺序检索算法和二分检索算法的决策树，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.**

检索问题的复杂度分析

定理 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 由命题3, n 个结点的决策树的深度 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.

6.5 排序算法时间复杂度分析

- 冒泡排序
- 快速排序与二分归并排序
- 堆排序
- 排序算法的复杂度下界

冒泡排序

输入: $L, n \geq 1$.

输出: 按非递减顺序排序的L.

算法 bubbleSort

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

实例

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

逆序序列的性质

- b_1=0; b_2=0,1; … ; b_n= 0,1, … , 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

冒泡排序算法复杂度分析

- 最坏情况分析：$W(n)=O(n^2)$，至多巡回$O(n)$次，每次$O(n)$.
- 对换只发生在相邻元素之间，每次相邻元素交换只消除1个逆序，比较次数不少于逆序数，最大逆序数 $n(n-1)/2$，于是$W(n)=\Theta(n^2)$.
- 平均情况：设各种输入是等可能的，置换α 的逆序序列是$(b_1, b_2, \ldots, b_n)$，置换α′ 的逆序序列为 $(0-b_1, 1-b_2, \ldots, n-1-b_n)$，α 与α′ 的逆序数之和为 $n(n-1)/2$. $n!$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 $n(n-1)/4$.
- 冒泡排序的最坏和平均复杂性均为$\Theta(n^2)$

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 **Heap-sort(*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.

若满足以下条件，称作堆.

(1) 所有内结点（可能一点除外）的度数为 2

(2) 所有树叶至多在相邻的两层

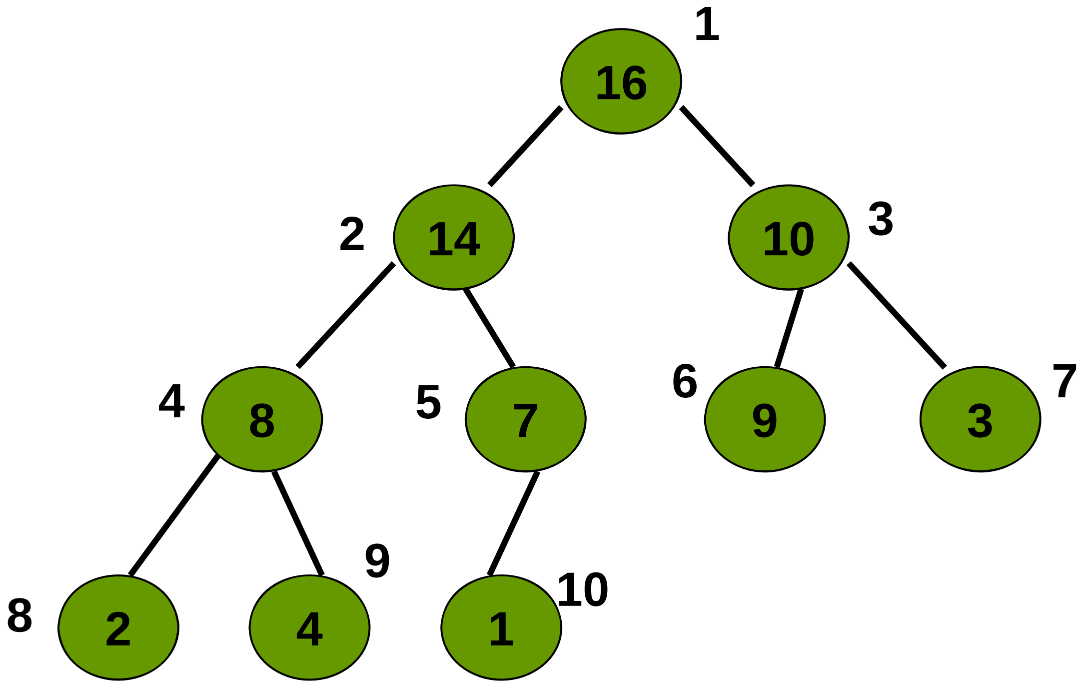
(3) d–1 层的所有树叶在内结点的右边

(4) d–1 层最右边的内结点可能度数为1（没有右儿子）

(5) 每个结点的元素不小于儿子的元素

若只满足前(4)条，不满足第(5)条，称作堆结构

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)**

$l \leftarrow left(i)$

$r \leftarrow right(i)$

if $l \le heap\text{-}size[A]$ **and** $A[l] > A[i]$

    then $largest \leftarrow l$

    else $largest \leftarrow i$

if $r \le heap\text{-}size[A]$ **and** $A[r] > A[largest]$

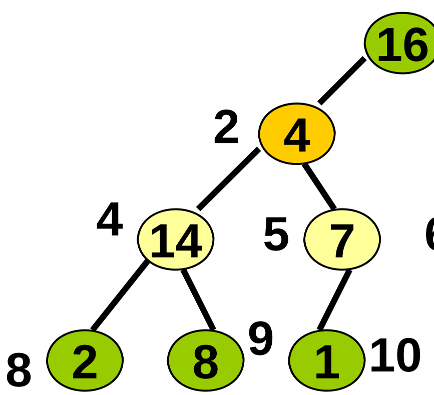
    then $largest \leftarrow r$

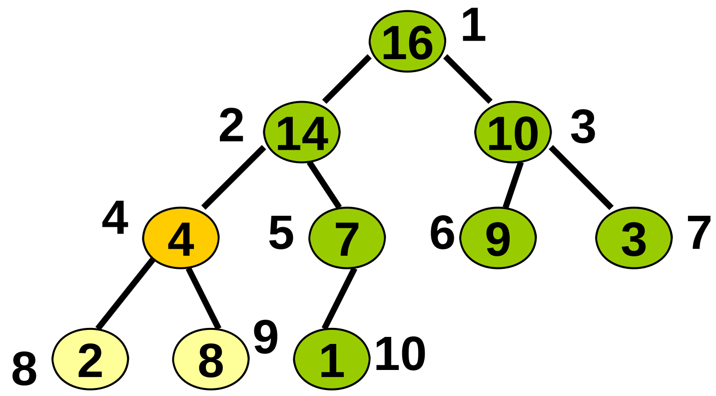
if $largest \ne i$

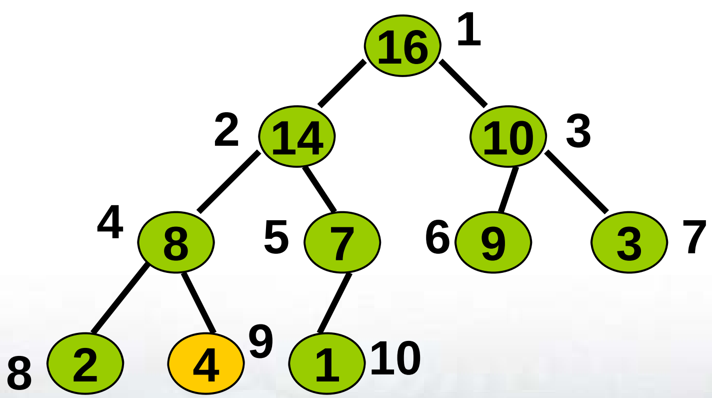
    then *exchange* $A[i] \leftrightarrow A[largest]$

        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

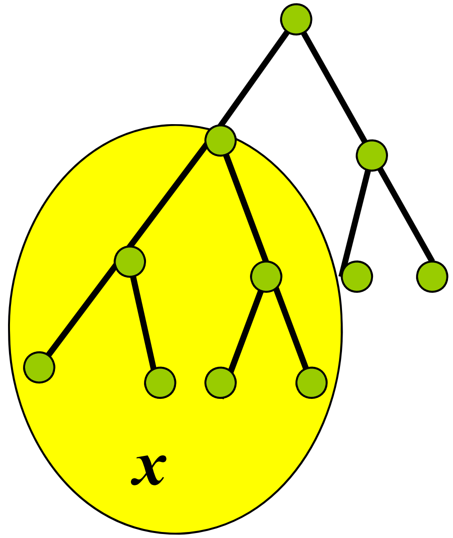
$$T(n) \le T(2n/3) + \Theta(1)$$

解得 $T(n) = \Theta(\log n)$

或者 $T(h) = \Theta(h)$

h为堆的根的高度

（距树叶最大距离）



结点总数

$x+(x-1)/2+1=(3x+1)/2$

北京大学

堆的运算：建堆

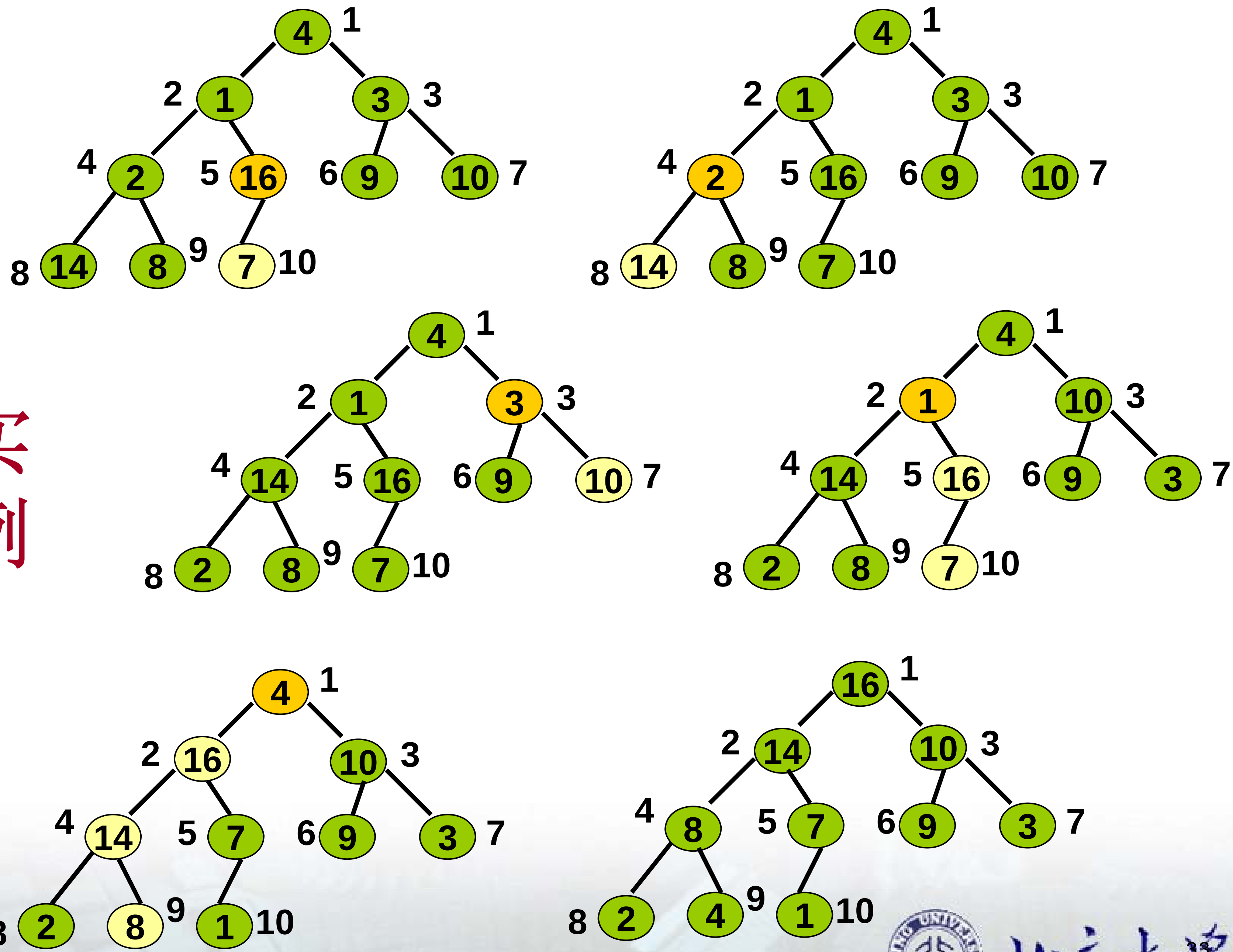
算法 **Build-Heap(A)**

1． $heap\text{-}size[A] \leftarrow length[A]$

2． **for** $i \leftarrow \lfloor length[A]/2 \rfloor$ **downto 1**

3． **do Heapify(A, i)**

实例



时间复杂度分析

- 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：
  按照高度计数结点数，乘以$O(h)$, 再求和
  Heapify(i) 的复杂度依赖于 i 的高度

$$T(n)=\sum_{h=0}^{\lfloor \log n \rfloor} \text{高为}h\text{的结点数} \times O(h)$$

计数高度为h的结点数

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.

证明思路：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.

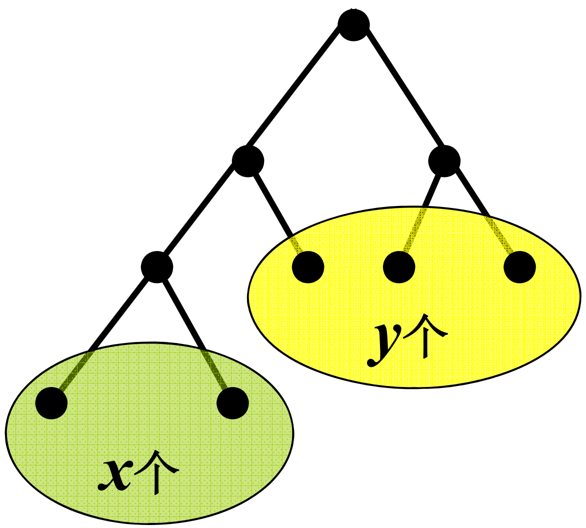
归纳基础

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

$$x+y=x+2^{d-1}-\frac{x}{2}=2^{d-1}+\frac{x}{2}$$

$$=\frac{(2^d+x)}{2}=\left\lceil\frac{2^d+x-1}{2}\right\rceil=\left\lceil\frac{n}{2}\right\rceil$$



每个内结点有 2 个儿子，树叶数为

(x为偶数，d-1 层以前各层结点总数 2^d–1)

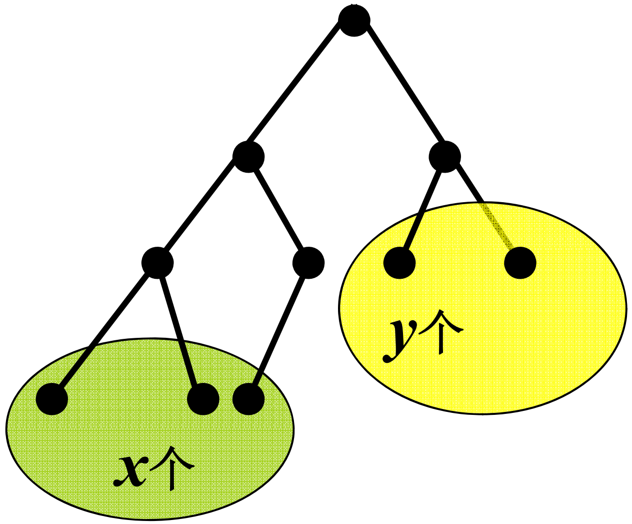
归纳基础（续）

Case2 x为奇数 (x为奇数，n为偶数)

$$x+y = x + 2^{d-1} - \frac{x+1}{2}$$

$$= 2^{d-1} + \frac{x-1}{2}$$

$$= \frac{2^d + x - 1}{2} = \frac{n}{2} = \left\lceil \frac{n}{2} \right\rceil$$



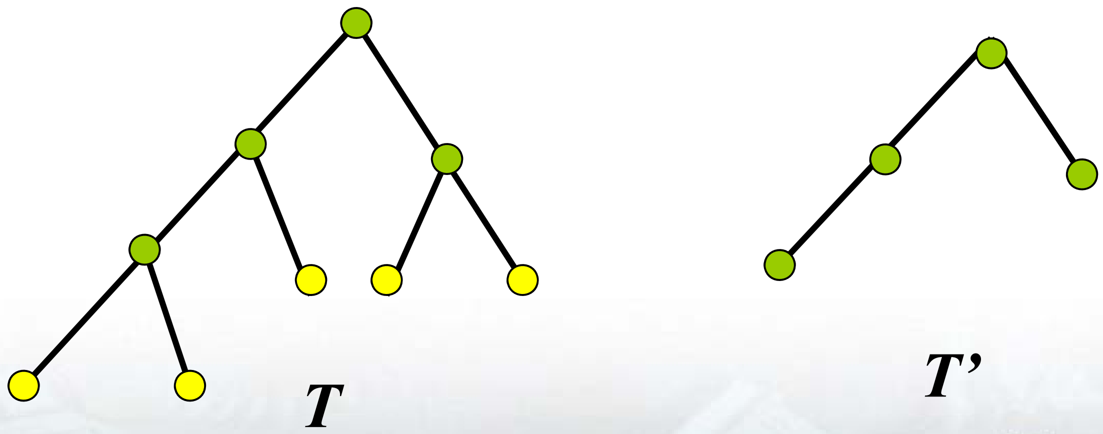
归纳步骤

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 与 T' 的关系：

T 为 h 的层恰好是 T' 的高为 h-1 的层.

$n=n'+n_0$　(*T'* 的结点数为 n', n_0为*T*的树叶数）



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，$n_0 = \left\lceil \frac{n}{2} \right\rceil$

$$n' = n - n_0 = \left\lfloor \frac{n}{2} \right\rfloor$$

$$n_h = n'_{h-1}$$

$$n_h = n'_{h-1} \le \left\lceil \frac{n'}{2^h} \right\rceil = \left\lceil \frac{\left\lfloor \frac{n}{2} \right\rfloor}{2^h} \right\rceil \le \left\lceil \frac{\frac{n}{2}}{2^h} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

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$O(h)$,

根据引理，高为 h 的结点数至多为 $\left\lceil \frac{n}{2^{h+1}} \right\rceil$, 因此时间复杂度

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$$

$$= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^{h+1}}\right) = O(n)$$

求和

$$\sum_{h=0}^{\infty}\frac{h}{2^h}=[0+\frac{1}{2}+\frac{2}{2^2}+\frac{3}{2^3}+\dots]$$

$$=[\frac{1}{2}+\frac{1}{2^2}+\dots]+[\frac{1}{2^2}+\frac{1}{2^3}+\dots]+[\frac{1}{2^3}+\frac{1}{2^4}+\dots]+\dots$$

$$=[1+\frac{1}{2}+\frac{1}{2^2}+\dots][\frac{1}{2}+\frac{1}{2^2}+\dots]$$

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

堆排序算法

算法 **Heap-sort(A)**

1. Build-Heap(A)

2. for $i \leftarrow length[A]$ downto 2

3. do *exchange* $A[1] \leftrightarrow A[i]$

4. $heap\text{-}size[A] \leftarrow heap\text{-}size[A]-1$

5. Heapify(A,1)

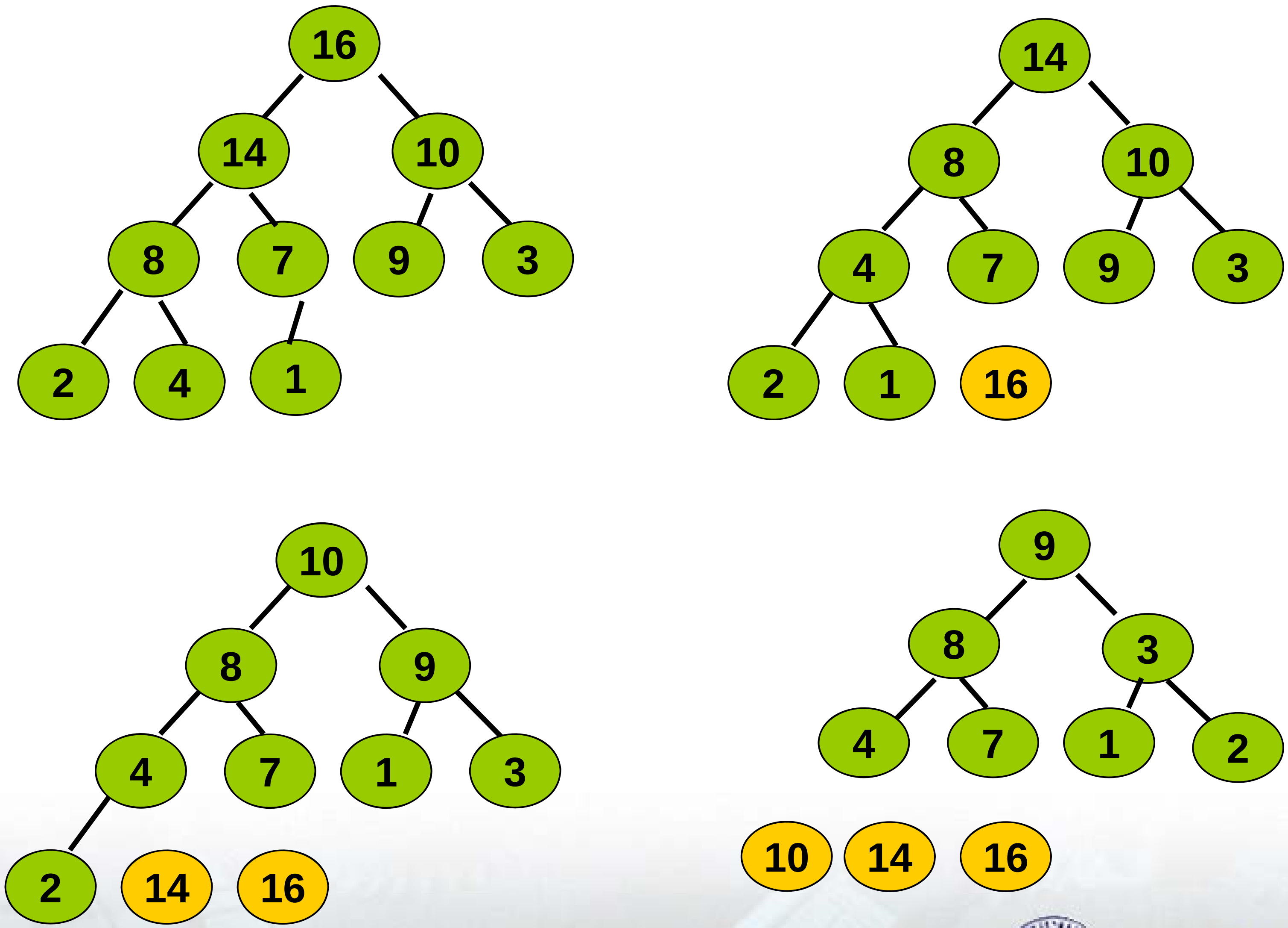
复杂性：$O(n\log n)$

Build-Heap 时间为 $O(n)$,

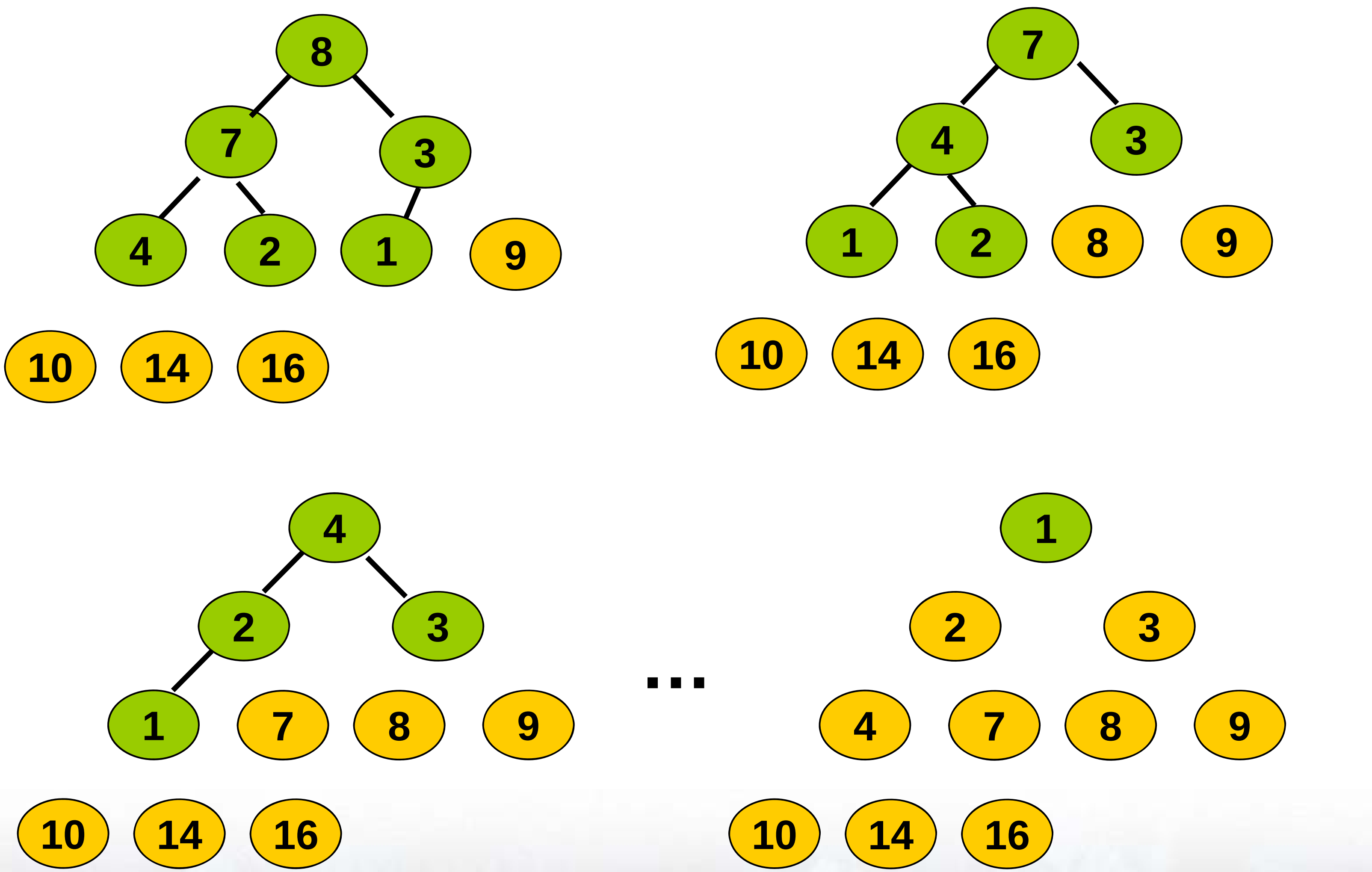
从行2-5 调用 **Heapify** n-1次，每次$O(\log n)$，

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

实例


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

排序问题的决策树

考虑以比较运算作为基本运算的排序算法类，

任取算法 A, 输入 $L=\{x_1, x_2, \ldots, x_n\}$, 如下定义决策树：

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

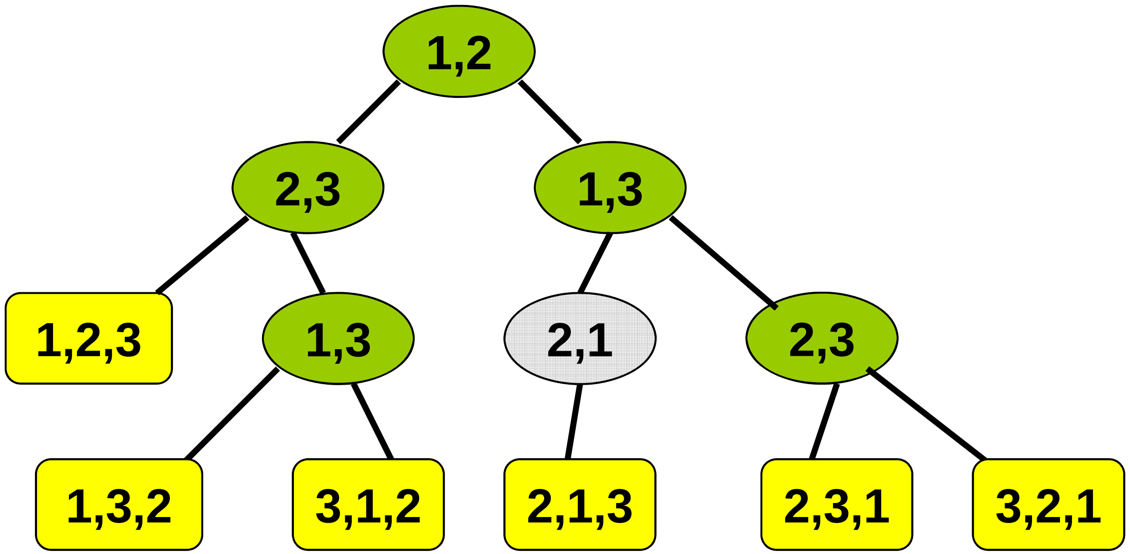
(2) $x_i > x_j$

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

一棵冒泡排序的决策树

设输入为 x_1, x_2, x_3，冒泡排序的决策树如下



任意输入：对应了决策树树中从树根到树叶的一条路经，

算法最坏情况下的比较次数：树深

删除非二叉的内结点（灰色结点），得到二叉树叫做 **B-树**

B-树深度不超过决策树深度，**B**-树有 $n!$ 片树叶

引理

引理1　设 t 为B-树中的树叶数，d 为树深，则 $t \le 2^d$.

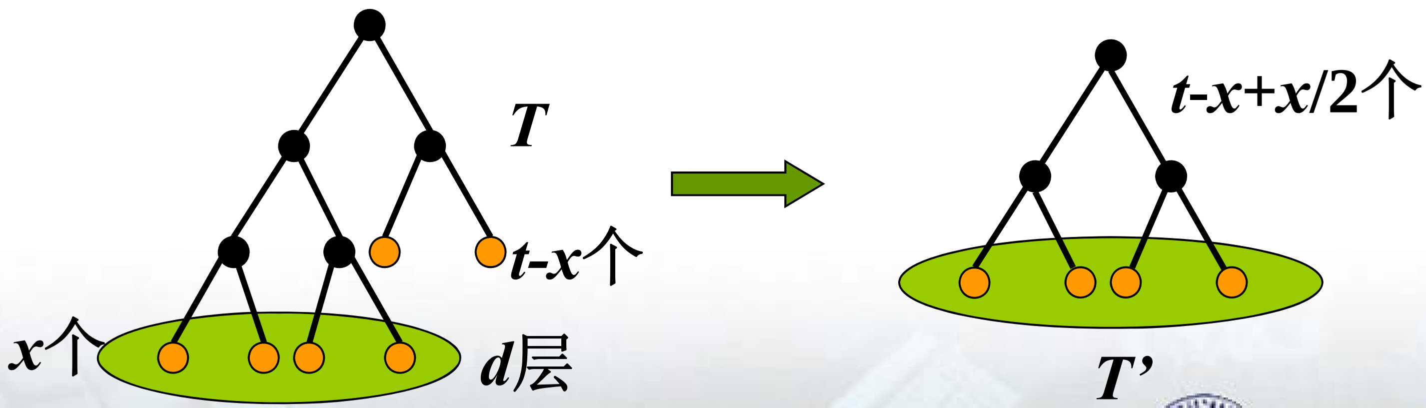
证明 归纳法.

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，任何通过比较对 n 个元素排序的算法的决策树的深度至少为$\lceil \log n! \rceil$.

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

epl(T)：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.

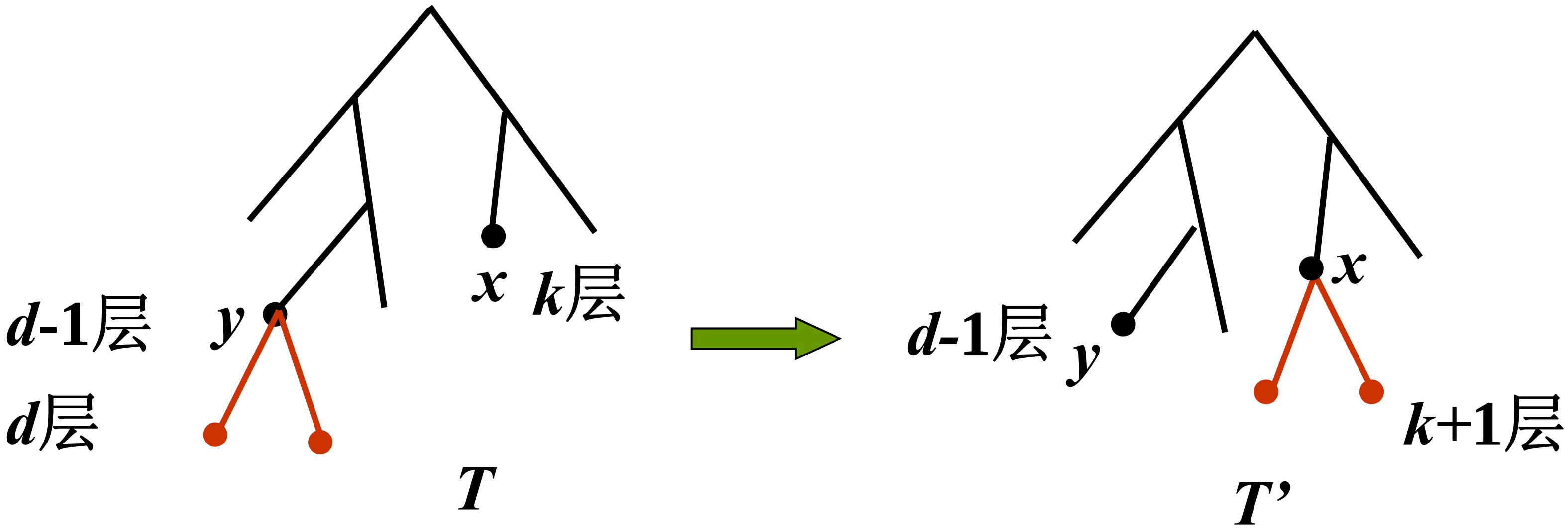
思路：分析具有最小 **epl(T)** 值的树的结构求得这个最小值.

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 值的树结构



$$\mathrm{epl}(T) - \mathrm{epl}(T') = (2d+k) - [(d-1)+2(k+1)]$$
$$= 2d+k - d+1-2k-2 = d-k-1 > 0 \qquad (d > k+1)$$

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 值最小的 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）

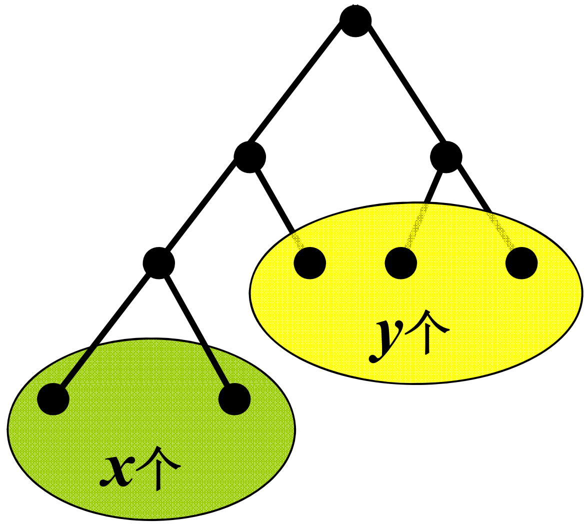
Case2 $t \neq 2^k$.

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 $n\log n - 1.5n$.

证明：算法类中任何算法的平均比较次数是该算法决策树T的 $\text{epl}(T)/n!$，根据引理4

$$
\begin{aligned}
A(n) &\geq \frac{1}{n!} epl(T) \\
&= \frac{1}{n!}\left(n!\lfloor \log n! \rfloor + 2\left(n! - 2^{\lfloor \log n! \rfloor}\right)\right) \\
&= \lfloor \log n! \rfloor + \varepsilon, \qquad 0 \leq \varepsilon < 1 \\
&\approx n \log n - 1.5n
\end{aligned}
$$

$$
0 \leq n! - 2^{\lfloor \log n! \rfloor} < n! - 2^{\log n! - 1} = n! - \frac{n!}{2} = \frac{n!}{2}
$$

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
冒泡排序	$O(n^2)$	$O(n^2)$	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

6.6 选择算法的时间复杂度分析

下界证明方法：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

选择算法的有关结果

	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	$O(n)\sim 2.95n$	$O(\log n)$

选最大算法 **Findmax**是最优的算法

北京大学

6.6.1 选最大与最小算法

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 的比较结果构造输入 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，通过与max的比较被淘汰. min是最小，A也必须确定有n-1个数比min大，通过与min的比较而淘汰.总共需要2n-2个信息单位.

基本运算与信息单位

数的状态标记及其含义：

N：没有参加过比较　　　W：赢

L：输　　　　　　　　　WL：赢过且至少输1次

如果比较后数的状态改变，则提供信息单位，状态不变不提供信息单位，每增加 1 个W 提供 1个信息单位每增加 1 个L 提供 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

case1： N,N → W,L：增加2个信息单位

case2： W,N → W,L：增加1个信息单位

case3： W,L → W,L：增加0个信息单位

算法输出与信息单位

算法输出的条件：

n-2 个数带有 **W** 和 **L** 标记，最大数只带 **W** 标记，最小数只带 **L** 标记，总计 2n-2个信息单位

对于任意给定的算法，构造输入的原则是：

根据算法的比较次序，针对每一步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，使得每次比较后得到的信息单位数达到最小. 从而使得为得到 输出所需要的 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.

对输入变量的赋值原则

x 与 *y* 的状态	赋值策略	新状态	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**

一个赋值的实例

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---$x_6>x_4$ …

	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

构造的输入为（20，10，25，2，5，3）

北京大学

问题复杂度的下界

为得到$2n$-2个信息单位，对上述输入A至少做$\lceil 3n/2 \rceil - 2$ 次比较.

一次比较得到2个信息单位只有case1. A至多有$\lfloor n/2 \rfloor$个case1，至多得到 $2\lfloor n/2 \rfloor \le n$个信息单位. 其它case, 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做的比较次数至少为

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

当 n 为奇数，A做的比较次数至少为

$$\lfloor n/2 \rfloor + n - 2 + 1 = (n-1)/2 + 1 + n - 2 = \lceil 3n/2 \rceil - 2$$

结论：FindMaxMin是最优算法

6.6.2 找第二大问题

元素x的权：$w(x)$, 表示以x为根的子树中的结点数

初始，$w(x_i)=1$，$i=1, 2, \ldots, n$;

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的值大于y 的值；//权大者胜

   若 $w(x)=w(y)$, 那么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

实例

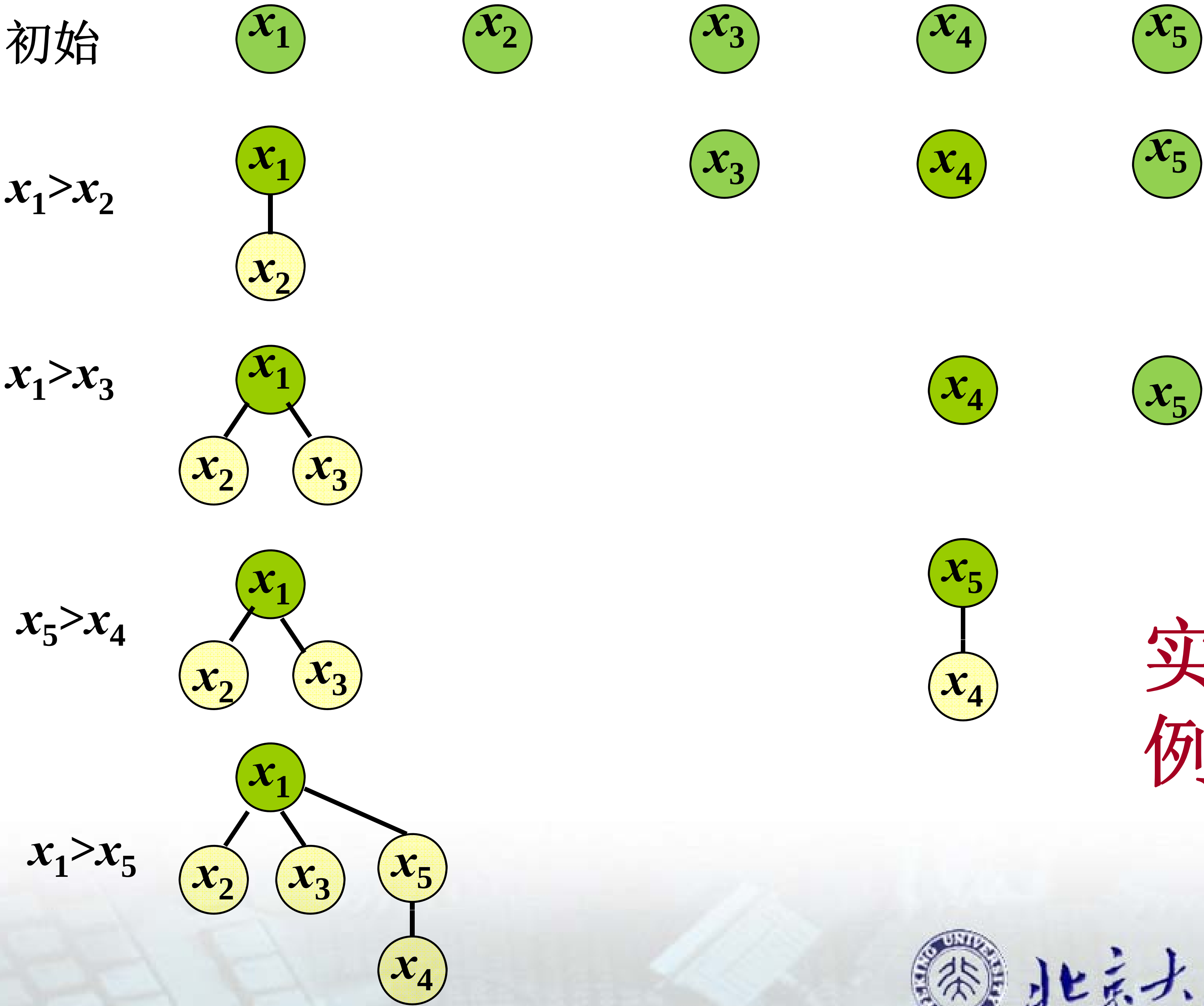
	$w(x_1)$	$w(x_2)$	$w(x_3)$	$w(x_4)$	$w(x_5)$	值
初始	1	1	1	1	1	*, *, *, *, *
第1步 $x_1>x_2$	2	0	1	1	1	20, 10, *, *, *
第2步 $x_1>x_3$	3	0	0	1	1	20, 10, 15, *, *
第3步 $x_5>x_4$	3	0	0	0	2	20, 10, 15, 30, 40
第4步 $x_1>x_5$	5	0	0	0	0	41, 10, 15, 30, 40

构造树

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；

实例



找第二大问题复杂度下界

针对这个输入，估计与max比较而淘汰的元素数

根的权为n，其它的结点权为0，根为max

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 $w_k \le 2w_{k-1}$，设 K 为max最终与权不为0的结点的比较次数，则

$$n = w_K \le 2^K w_0 \le 2^K \Rightarrow K \ge \log n \Rightarrow K \ge \lceil \log n \rceil$$

这 K 个元素彼此不同，因为同一个元素不可能被 计数 2 次. 其中为确定第二大，要淘汰$K-1$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.

6.6.3 找中位数问题

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 $y \ge$ median), x满足上述条件的第一次比较

$\exists y$ ($x<y$ 且 $y \le$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>median, y<median;
3. 如果A比较x与y, 且x>median, y没被赋值，则赋值 y 使得
   y<median;
4. 如果A比较x与y, 且x<median, y没被赋值，则赋值y 使得
   y>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.

构造实例

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. $x_5>x_6$　　$x_5=6, x_6=3$
5. $x_7=4$
6. $x_1>x_3$
7. $x_3>x_7$
8. …

非决定性比较

决定性比较

复杂性分析

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**

这样赋值的输入使得A在这个输入下所进行的上述比较都是非决定性的. 这样的比较至少有$(n-1)/2$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	**FindMaxMin**	$\lceil 3n/2 \rceil-2$	$\lceil 3n/2 \rceil-2$	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

6.7 通过归约确认问题计算复杂度的下界

问题P，问题Q

问题Q的复杂度已知（至少线性） $\Omega(g(n))$

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 $f(n)=O(n)$，解的反变换 $s(n)$ 也是线性时间

解Q的算法：$T_Q(n)= f(n)+T_p(n)+s(n)$

1. 将Q的实例 I 变成 $f(I)$， $f(n)$
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 $T_p(n)$
3. 将解变换成原问题的解 $s(n)$

解P的算法可以解Q. 且时间的阶一样，因此 P至少与Q一样难. $Q \le^l P$

$$f(n)+T_p(n)+s(n)= T_Q(n) =\Omega(g(n))$$

因子分解与素数测试

- 问题：因子分解 factor

  素数测试 testp
- 假设testp问题的难度是$W(n)$
- 素数测试算法 $A(n)$

  1. $p \leftarrow$ factor(n)
  2. if p=“true” then return”No”
  3. else return”Yes”

- 结论： $\Omega(W(n))=T_{\text{testp}}(n)\leq T_{\text{factor}}(n)$

元素唯一性问题

- 问题：给定 n 个数的集合S，判断 S 中的元素是否存在相同元素.
- 元素唯一性问题的复杂度为 $\Theta(n\log n)$

  输入：多重集 $S=\{ n_1\cdot a_1, n_2\cdot a_2, \ldots , n_k\cdot a_k \}$

  构造决策树，树叶为 S 的全排列数

$$\frac{n!}{n_1!\, n_2! \ldots n_k!}$$

最坏情况下树深为

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

最邻近点对与唯一性问题

- **P问题与Q问题：**
  P: 平面直角坐标系中n个点的最邻近点对问题**Close**
  Q: 元素的唯一性问题**Uniqueness**　　$\Omega(n\log n)$
- 变换 f：
  Q的实例：$x_1, x_2, \ldots, x_n$，变成点$(x_1,0),(x_2,0),\ldots,(x_n,0)$
- 解Q算法：
  1. 利用求最邻近点对算法 P 计算最短距离 d.
  2. if d=0 then return "No"
  3. else return "Yes"
- 结论：计算平面直角坐标系中n个点的最邻近点对问题的时间是 $\Omega(n\log n)$，其中算法以比较为基本运算

最小生成树与唯一性问题

- **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**Uniqueness**　$\Omega(n\log n)$

- 变换 f：

  Q的实例：$x_1, x_2, \ldots, x_n$，变成**X** 轴上的n个点,

- 解**Q**算法：

  1. 利用求最小生成树算法**P**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：计算平面直角坐标系 n点最小生成树时间是 $\Omega(n\log n)$，其中算法以比较为基本运算

第7章 NP完全性

主要内容

Turing机

计算复杂性理论

NP完全性理论的基本概念

用NP完全性理论分析问题

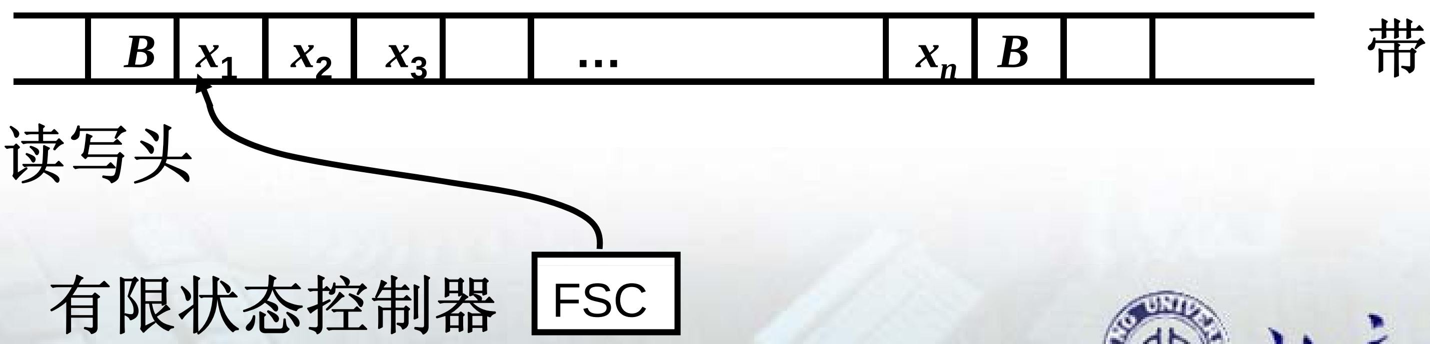
NP难度

Turing机的定义

基本模型　双向无限带的Turing机

$M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$, 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-格局(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 $x_1x_2 \ldots x_i \ldots x_n$, 读写头指向字符 x_i, 则它的瞬间描述是:

$$x_1x_2 \ldots x_{i-1} q\, x_i \ldots x_n \vdash x_1x_2 \ldots x_{i-2}\, p\, x_{i-1} Y x_{i+1} \ldots x_n$$

$\vdash$ 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

实例

设 $L=\{\,0^n1^n \mid n\geq 1\,\}$, 设计接受 L 的 Turing机如下:

$M = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$

$Q = \{\, q_0, q_1, q_2, q_3, q_Y, q_N \,\}$,

$\Sigma = \{\, 0, 1\}$,

$\Gamma = \{\, 0, 1, X, Y, B \,\}$,

$F = \{\, q_Y, q_N\}$

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

	0	1	X	Y	B
q_0	(q_1,X,R)	q_N	q_N	(q_3,Y,R)	q_N
q_1	$(q_1,0,R)$	(q_2,Y,L)	q_N	(q_1,Y,R)	q_N
q_2	$(q_2,0,L)$	q_N	(q_0,X,R)	(q_2,Y,L)	q_N
q_3	q_N	q_N	q_N	(q_3,Y,R)	q_Y

例如输入 **0011**，**Turing** 机动作如下：

$$q_0 0011 \vdash Xq_1 011 \vdash X0q_1 11 \vdash Xq_2 0Y1 \vdash q_2 X0Y1$$
$$\vdash Xq_0 0Y1 \vdash XXq_1 Y1 \vdash XXYq_1 1 \vdash XXq_2 YY \vdash Xq_2 XYY$$
$$\vdash XXq_0 YY \vdash XXYq_3 Y \vdash XXYYq_3 \vdash XXYYq_Y$$

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.

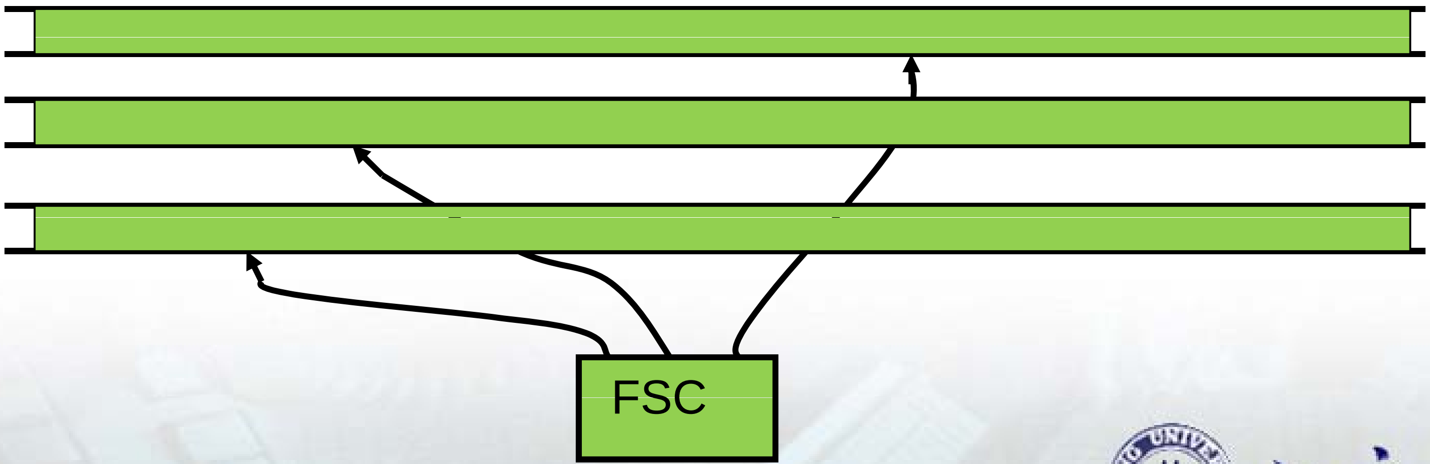
基本 Turing 机可以推广为 ***k* 条带的 Turing 机** DTM.

确定型 Turing机可以推广到**非确定型 Turing机** NDTM.

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.



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$O(\log n)$.

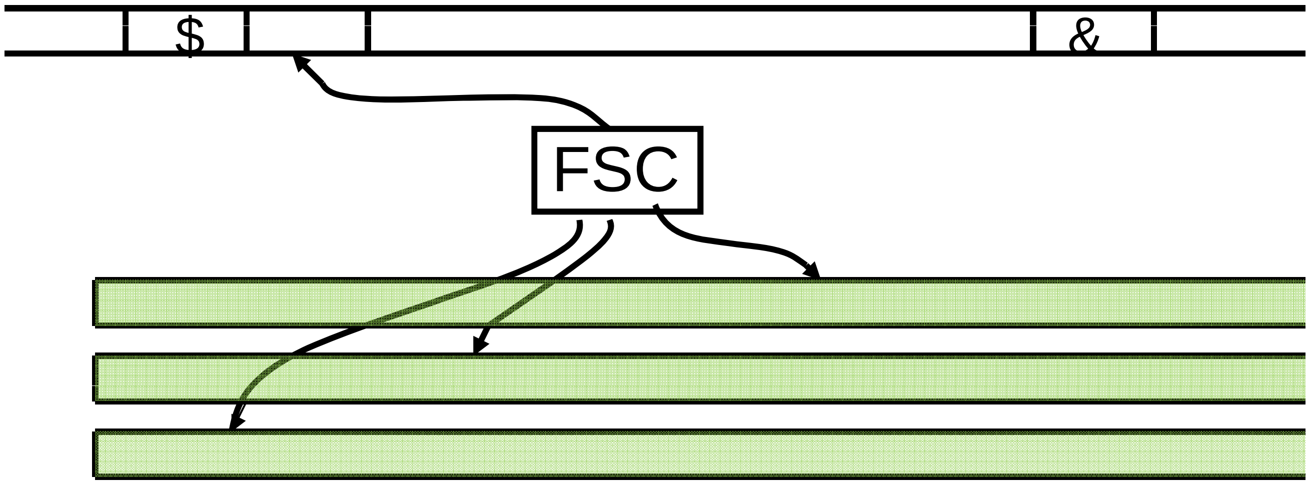
M_1有2条带,把 c 左边的 w 复制到第2条带上. 当发现 c 时第2条带的读写头向左, 输入带的读写头向右. 比较两个带头的符号, 如果符号一样, 字符个数一样, M_1 接受 x. M_1 至多作 $n+1$ 个动作. 时间复杂度为 $n+1$. 空间复杂度为$\lceil (n-1)/2 \rceil +1$.

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 $O(n^2)$.

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
- 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

因此这里作如下规定:

对一切n, $S(n)\geq 1$, $\log n$ 是 $\max\{1, \lceil \log n \rceil\}$ 的缩写.

对一切n, $T(n)\geq n+1$, $T(n)$是 $\max\{n+1, T(n)\}$ 的缩写.

有关计算复杂度的结果

	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	$T(n)$ $\liminf\limits_{n\to\infty}\frac{T(n)}{n}=\infty$	$cn, \forall c>1$	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	$T(n)\log T(n)$		

7.1 P类与NP类

7.1.1 易解的问题与难解的问题

排序$O(n\log n)$，Dijkstra算法 $O(n^2)$，最大团回溯法 $O(n2^n)$

用一台每秒 10亿 (10^9) 次的超大型计算机，上述算法的时间：

10万个数据排序，$10^5\times\log_2 10^5\approx 1.7\times 10^6$，$t=1.7\times 10^{-3}$秒

1万个顶点的图的单源最短路径，$(10^4)^2=10^8$，t=0.1秒

100个顶点的图的最大团，$100\times 2^{100}\approx 1.8\times 10^{32}$，

$t=1.8\times 10^{32}/10^9=1.8\times 10^{21}$秒$=5.7\times 10^{15}$年，即5千7百万亿年!

1分钟能解多大的问题. 1分钟 6×10^{10} 次运算

可给2×10^9 =20亿个数据排序

用Dijkstra算法可解2.4×10^5个顶点的图的单源最短路径问题.

回溯法1天只能解41个顶点的图的最大团问题.

北京大学

算法的时间复杂度

函数 f 和 g 是多项式相关的: 如果存在多项式 p 和 q 使得对任意的 $n\in\mathrm{N}$, $f(n)\le p(q(n))$ 和 $g(n)\le q(f(n))$.

例如　$n\log n$ 与 n^2, n^2+2n+5 与 n^{10} 都是多项式相关的,
$\log n$ 与 n, n^5 与 2^n 不是多项式相关的.

问题Π 的实例 I 的规模: I 的二进制编码的长度, 记作$|I|$.

定义7.1　如果存在函数 f:N→N使得 对任意的规模为 n 的实例 I, 算法 A 对 I 的运算在 $f(n)$ 步内停止, 则称算法 A的时间复杂度为 $f(n)$.

多项式时间算法: 以多项式为时间复杂度.

易解的问题: 有多项式时间算法.

难解的问题: 不存在多项式时间算法.

北京大学

几点说明

1. 当采用合理的编码时, 输入的规模都是多项式相关的.

“合理的”是指在编码中不故意使用许多冗余的字符.

例如, 设实例 I 是一个无向简单图 $G = <V,E>$,

V={ a. b, c, d }, E={ (a,b), (a,d), (b,c), (b,d), (c,d) }

用邻接矩阵表示, 编码 e_1=0101/1011/0101/1110/, 长度 20.

用关联矩阵表示, 编码 e_2=11000/10110/00101/01011/, 长度 24.

G 有n个顶点m条边,

用邻接矩阵时$|I|=n(n+1)$, 用关联矩阵时$|I|=n(m+1)$.

两者多项式相关.

2. 自然数应采用k ($k\geq 2$)进制编码, 不能采用一进制编码.

n 的二进制编码有$\lceil \log_2(n+1) \rceil$位, 一进制编码有$n$位, 两者不是多项式相关的.

北京大学

几点说明

3. 时间复杂度常表成计算对象的某些自然参数的函数，如图的顶点数或边数的函数. 实例的二进制编码的长度与这些自然参数通常是多项式相关的.

4. 运行时间通常是计算执行的 操作指令数，执行的指令数与实际运行时间是多项式相关的.

(1) 要求每一条指令的执行时间是固定的常数.

(2) 规定一个基本操作指令集，可由位逻辑运算与、或、非组成，任何可用这个基本操作指令集中常数条指令实现的操作都是合理的指令，由有限种合理的指令构成的操作指令集是合理的操作指令集.

在上述约定下,算法是否是多项式时间的与采用的编码和操作指令集无关， 从而一个问题是易解的、还是难解的也与采用的编码和操作指令集无关.

易解的问题与难解的问题

易解的问题.

如排序、最小生成树、单源最短路径等

已证明的难解问题.

(1) 不可计算的, 即根本不存在求解的算法, 如希尔伯特第十问题——丢番图方程是否有整数解.

(2) 有算法 但至少需要指数或更多的时间或空间，如带幂运算的正则表达式的全体性, 即任给字母表A上的带幂运算的正则表达式R, 问:$\langle R\rangle = A^*$?这个问题至少需要指数空间.

既没有找到多项式时间算法、又没能证明是难解的问题.

如哈密顿回路问题、货郎问题、背包问题等

7.1.2 判定问题

判定问题: 答案只有两个——是, 否.

判定问题 $\Pi = <D_\Pi, Y_\Pi>$, 其中D_Π是实例集合, $Y_\Pi \subseteq D_\Pi$是所有答案为“Yes”的实例.

哈密顿回路 (HC): 任给无向图G, 问G有哈密顿回路吗?

货郎问题 (TSP): 任给 n 个城市, 城市 i 与城市 j 之间的正整数距离 $d(i,j)$, $i \neq j$, $1 \leq i, j \leq n$, 以及正整数 D, 问有一条每一个城市恰好经过一次最后回到出发点且长度不超过 D 的巡回路线吗? 即, 存在 1, 2, …, n 的排列 σ 使得

$$\sum_{i=1}^{n-1} d(\sigma(i), \sigma(i+1)) + d(\sigma(n), \sigma(1)) \leq D?$$

北京大学

0-1背包的判定问题与优化问题

0-1背包: 任给 n 件物品和一个背包, 物品 i 的重量为 w_i, 价值为 v_i, $1 \le i \le n$, 以及背包的重量限制 B 和价值目标 K, 其中 w_i, v_i, B, K 均为正整数, 问能在背包中装入总价值不少于 K 且总重量不超过 B 的物品吗? 即, 存在子集 $T \subseteq \{1,2,\ldots,n\}$使得

$$\sum_{i \in T} w_i \le B \quad 且 \quad \sum_{i \in T} v_i \ge K?$$

搜索问题, 组合优化问题与判定问题的对应.

如果搜索问题, 组合优化问题有多项式时间算法, 则对应的判定问题也有多项式时间算法; 通常反之亦真.

组合优化问题与判定问题

组合优化问题 Π^* 由3部分组成:

(1) 实例集D_{Π^*}

(2) $\forall I \in D_{\Pi^*}$, 有一个有穷非空集 $S(I)$, 其元素称作 I 的可行解

(3) $\forall s \in S(I)$, 有一个正整数 $c(s)$, 称作 s 的值

如果 $s^* \in S(I)$, 对所有的 $s \in S(I)$, 当Π^*是最小(大)化问题时,

$$c(s^*) \le c(s) \qquad (c(s^*) \ge c(s))$$

则称 s^*是 I 的最优解, $c(s^*)$是 I 的最优值, 记作OPT(I).

Π^*对应的判定问题 Π $= <D_{\Pi}, Y_{\Pi}>$定义如下:

$$D_{\Pi} = \{ (I, K) \mid I \in D_{\Pi^*}, K \in Z^* \}$$, 其中Z*是非负整数集合.

当Π^*是最小化问题时, $Y_{\Pi} = \{ (I, K) \mid \mathrm{OPT}(I) \le K \}$;

当Π^*是最大化问题时, $Y_{\Pi} = \{ (I, K) \mid \mathrm{OPT}(I) \ge K \}$.

7.1.3 NP类 (nondeterministic polymial)

定义**7.2** 所有多项式时间可解的判定问题组成的问题类称作 **P类**.

定义**7.3** 设判定问题 $\Pi = <D,Y>$, 如果存在两个输入变量的多项式时间算法 A 和多项式 p，对每一个实例 $I\in D$, $I\in Y$ 当且仅当存在 t, $|t| \le p(|I|)$, 且 A 对输入 I 和 t 输出 "**Yes**"，则称Π是多项式时间可验证的, A是Π的多项式时间验证算法, 而当 $I\in Y$ 时, 称 t 是 $I\in Y$ 的证据.

由所有多项式时间可验证的判定问题组成的问题类称作 **NP类**.

HC(哈密顿回路), TSP(货郎), 0-1背包$\in$NP

非确定型多项式时间算法

非确定型多项式时间算法

(1) 对给定的实例 I, 首先"猜想"一个 t, $|t| \le p(|I|)$

(2) 然后检查 t 是否是证明 $I \in Y$ 的证据

(3) 猜想和检查可以在多项式时间内完成

(4) 当且仅当 $I \in Y$ 时能够正确地猜想到一个证据 t

*注：非确定型多项式时间算法不是真正的算法

定理7.1　$\mathrm{P} \subseteq \mathrm{NP}$

问题：P=NP?

7.2 多项式时间变换与NP完全性

7.2.1 多项式时间变换

如何比较两个问题的难度?

定义7.4　设判定问题$\Pi_1 = <D_1, Y_1>$, $\Pi_2 = <D_2, Y_2>$. 如果函数 $f: D_1 \to D_2$ 满足条件:

(1) f 是多项式时间可计算的,

(2) 对所有的 $I \in D_1$, $I \in Y_1 \Leftrightarrow f(I) \in Y_2$,

则称 f 是 Π_1 到 Π_2 的多项式时间变换.

如果存在 Π_1 到 Π_2 的多项式时间变换, 则称 Π_1 可多项式时间变换到 Π_2, 记作 $\Pi_1 \le_p \Pi_2$.

北京大学

例

例7.1　HC$\leq_p$TSP.

证　对 HC 的每一个实例 I: 无向图 $G=\langle V,E\rangle$，TSP 对应的实例 $f(I)$ 为: 城市集 V, 任意两个不同的城市 u 和 v 之间的距离

$$d(u,v)=\begin{cases}1, & 若(u,v)\in E, \\ 2, & 否则,\end{cases}$$

以及界限 $D=|V|$.

例

最小生成树: 任给连通的无向赋权图 $G=<V,E,W>$ 以及正整数 B, 其中权 $W:E\to Z^+$, 问有权不超过 B 的生成树吗?

最大生成树: 任给连通的无向赋权图 $G=<V,E,W>$ 以及正整数 D, 其中权 $W:E\to Z^+$, 问 G 有权不小于 D 的生成树吗?

例7.2 最大生成树$\le_p$最小生成树.

证 任给最大生成树的实例 I:连通的无向赋权图 $G=<V,E,W>$ 和正整数 D, 最小生成树的对应实例 $f(I)$: 图 $G'=<V,E,W'>$ 和正整数 $B=(n-1)M-D$, 其中

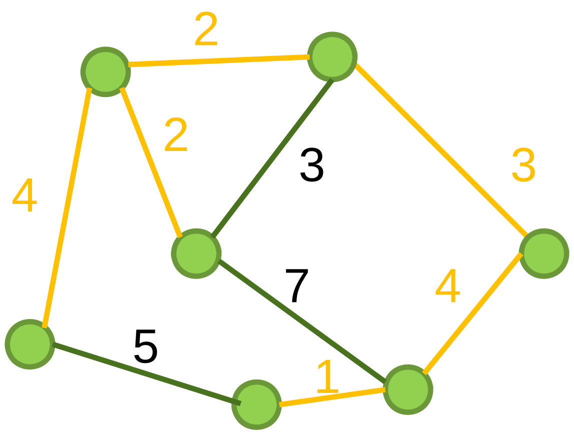
$$n=|V|,\quad M=\max\{\ W(e)\mid e\in E\ \}+1,\quad W'(e)=M-W(e)$$

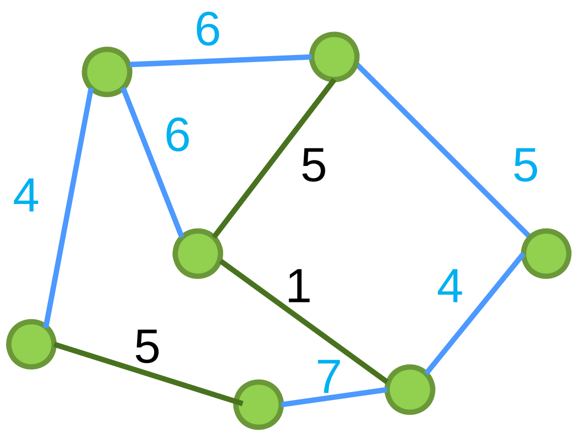
如果存在 G 的生成树 T, 使得 $\sum_{e\in T} W(e)\ge D$, 则

$$\sum_{e\in T} W'(e)=(n-1)M-\sum_{e\in T} W(e)\le (n-1)M-D=B.$$

反之亦然. f 多项式时间可计算.

变换实例





D=12

B=6×8–12=36

最大生成树 ***T***
的实例 ***G***

最小生成树 ***T'***
的实例 ***G'***

$$M = 8,\quad W(T) = 16 \geq 12,\quad W(T') = 32 \leq 36$$

$$\sum_{e \in T} W'(e) = (n-1)M - \sum_{e \in T'} W(e)$$

$\le_p$的性质

定理7.2 $\le_p$具有传递性. 即, 设 $\Pi_1\le_p\Pi_2$, $\Pi_2\le_p\Pi_3$, 则$\Pi_1\le_p\Pi_3$.
证　设 Π_i=<D_i,Y_i>, i=1,2,3, f 和 g是 Π_1 到 Π_2 和 Π_2 到 Π_3 的多项式时间变换. 对每一个$I\in D_1$, 令$h(I)=g(f(I))$.

计算 f 和 g 的时间上界分别为多项式 p 和 q, 不妨设 p 和 q 是单调递增的. 计算 h 的步数不超过 $p(|I|) + q(|f(I)|)$. 输出作为合理的指令, 一步只能输出长度不超过固定值 k 的字符串, 因而 $|f(I)| \le k\,p(|I|)$. 于是,

$$p(|I|) + q(|f(I)|) \le p(|I|) + q(kp(|I|)),$$

得证h是多项式时间可计算的.

对每一个$I\in D_1$,

$$I\in Y_1 \Leftrightarrow f(I)\in Y_2 \Leftrightarrow h(I) = g(f(I))\in Y_3,$$

得证 h是 Π_1 到 Π_3 的多项式时间变换.

$\leq_p$的性质

定理7.3　设 $\Pi_1 \leq_p \Pi_2$, 则 $\Pi_2 \in$ P 蕴涵 $\Pi_1 \in$ P.

证　设 $\Pi_1 = \langle D_1, Y_1 \rangle$, $\Pi_2 = \langle D_2, Y_2 \rangle$, f 是 Π_1 到 Π_2 的多项式时间变换, A 是计算 f 的多项式时间算法. 又设 B 是 Π_2 的多项式时间算法. 如下构造 Π_1 的算法 C:

(1) 对每一个 $I \in D_1$, 应用 A 得到 $f(I)$,

(2) 对 $f(I)$ 应用 B,

(3) C 输出"Yes"当且仅当 B 输出"Yes".

推论7.1　设 $\Pi_1 \leq_p \Pi_2$, 则 Π_1 是难解的蕴涵 Π_2 是难解的.

由例7.2 及最小生成树∈P, 得知最大生成树∈P.

由例7.1, 如果 TSP∈P, 则 HC∈P. 反过来, 如果 HC 是难解的, 则 TSP 也是难解的.

7.2.2 NP完全性

定义7.5 如果对所有的 $\Pi' \in$ NP, $\Pi' \le_p \Pi$, 则称 Π 是NP难的. 如果 Π 是 NP 难的且 $\Pi \in$ NP, 则称 Π 是 NP完全的.

定理7.4 如果存在NP难的问题 $\Pi \in$ P, 则 P=NP.

推论7.2 假设P≠NP, 那么, 如果 Π 是NP难的, 则 $\Pi \notin$ P.

定理7.5 如果存在NP难的问题 Π' 使得 $\Pi' \le_p \Pi$, 则 Π 是NP难的.

推论7.3 如果 $\Pi \in$ NP并且存在 NP完全问题 Π' 使得 $\Pi' \le_p \Pi$, 则Π是NP完全的.

证明NP完全性的“捷径”

(1) 证明$\Pi \in$ NP;

(2) 找到一个已知的NP完全问题Π', 并证明$\Pi' \le_p \Pi$.

7.2.3 Cook-Levin定理

合式公式：由变元、逻辑运算符以及圆括号按照一定的规则组成的表达式.

变元和它的否定称作文字.

有限个文字的析取称作简单析取式.

有限个简单析取式的合取称作合取范式.

给定每一个变元的真假值称作一个赋值. 如果赋值 t 使得合式公式 F 为真, 则称 t 是 F 的成真赋值.

如果 F 存在成真赋值, 则称 F 是可满足的.

例如 $F_1=(x_1\vee x_2)\wedge(\neg x_1\vee x_2\vee x_3)\wedge\neg x_2$ 是一个合取范式.

令$t(x_1)=1, t(x_2)=0, t(x_3)=1$是$F_1$的成真赋值, F_1是可满足的.

$F_2=(x_1\vee\neg x_2\vee x_3)\wedge(\neg x_1\vee\neg x_2\vee x_3)\wedge x_2\wedge\neg x_3$不是可满足的.

Cook-Levin定理

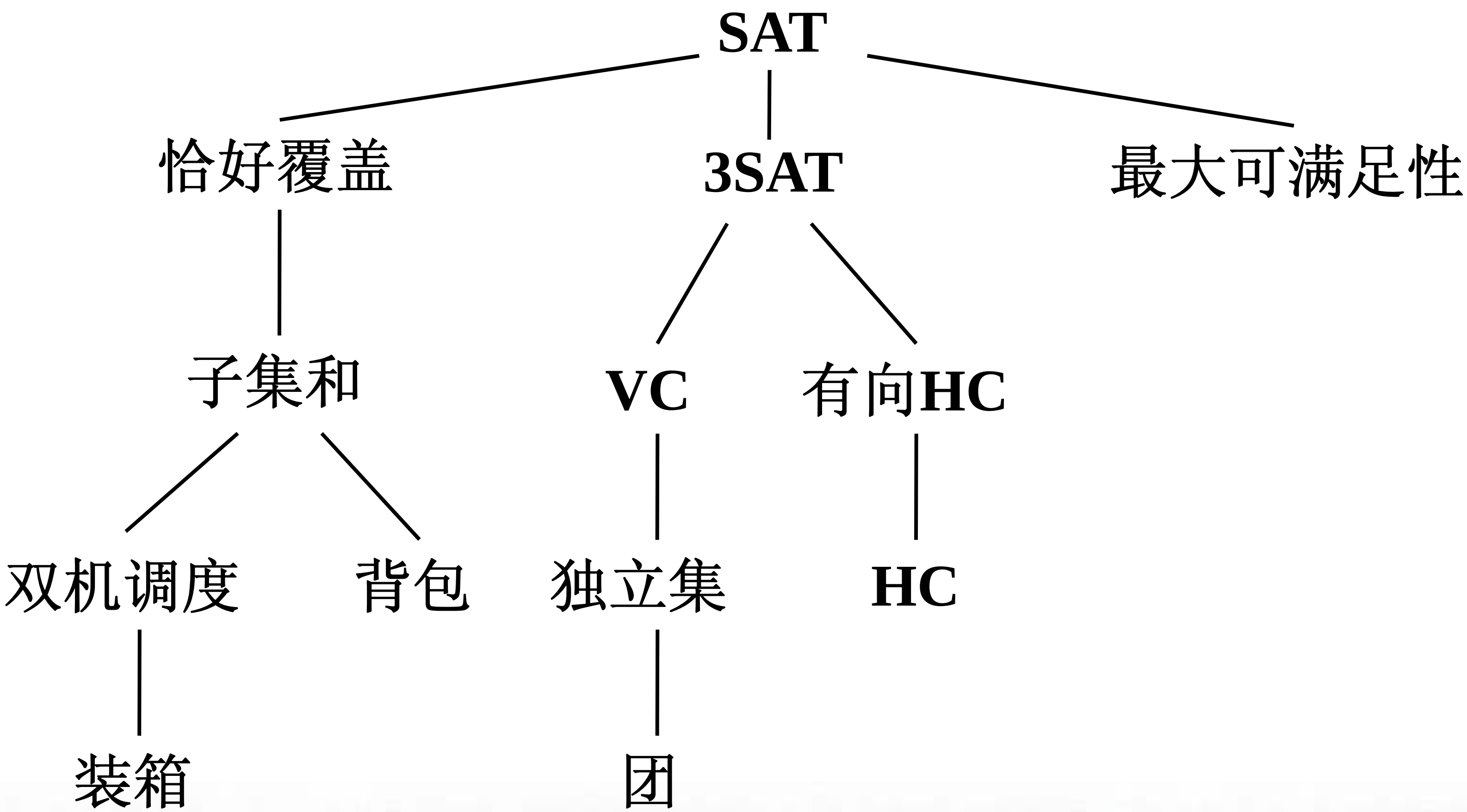
可满足性 (SAT): 任给一个合取范式 F, 问 F 是可满足的吗?

定理7.6 (Cook-Levin定理) SAT 是NP完全的.

证明思想：对于任意一个NP类中语言L，存在一个接受它的非确定型图灵机M. 构造L到SAT的多项式变换. 对于L中的任意串x，M对x的接受计算变换成一个SAT问题的肯定实例

定理7.7 P=NP 的充分必要条件是存在 NP 完全问题 $\Pi\in$P.

7.3 几个NP完全问题



7.3.1 最大可满足性与三元可满足性

最大可满足性(MAX-SAT): 任给关于变元 $x_1, x_2,\ldots, x_n$ 的简单析取式 $C_1,C_2,\ldots,C_m$ 及正整数 K, 问存在关于变元 $x_1, x_2,\ldots, x_n$ 的赋值使得 $C_1,C_2,\ldots,C_m$ 中至少有 K 个为真吗?

3元合取范式: 每一个简单析取式恰好有3个文字的合取范式.
三元可满足性(3SAT): 任给一个3元合取范式F, 问F是可满足的吗?

子问题: 设判定问题$\Pi=<D,Y>$, $\Pi'=<D',Y'>$, 如果$D'\subseteq D$, $Y'=D'\cap Y$, 则 Π'是Π 的特殊情况, 称作 Π 的子问题.

SAT是MAX-SAT的子问题(取$K=m$), 3SAT是SAT的子问题.

定理7.8 MAX-SAT是NP完全的
定理7.9 3SAT是NP完全的.

北京大学

7.3.2 顶点覆盖、团与独立集

设无向图$G=\langle V,E\rangle$, $V'\subseteq V$. V'是G的一个

顶点覆盖: G的每一条边都至少有一个顶点在V'中.

团: 对任意的$u,v\in V'$且$u\neq v$, 都有$(u,v)\in E$.

独立集: 对任意的$u,v\in V'$, 都有$(u,v)\notin E$.

引理7.1 对任意的无向图$G=\langle V,E\rangle$和子集$V'\subseteq V$, 下述命题是等价的:

(1) V'是G的顶点覆盖,

(2) $V-V'$是G的独立集,

(3) $V-V'$是补图 $G^c=\langle V,E^c\rangle$的团.

顶点覆盖、团与独立集

顶点覆盖(VC): 任给一个无向图G=<V,E >和非负整数 $K \le |V|$, 问G 有顶点数不超过 K 的顶点覆盖吗?

团: 任给一个无向图G=<V,E>和非负整数 $J \le |V|$, 问G 有顶点数不小于J 的团吗?

独立集: 任给一个无向图 G=<V,E>和非负整数 $J \le |V|$, 问G 有顶点数不小于J 的独立集吗?

定理7.10 顶点覆盖是 NP 完全的.

定理7.11 独立集和团是 NP 完全的.

7.3.3 -7.34 哈密顿回路、货郎问题与恰好覆盖

有向哈密顿回路: 任给有向图D, 问:D中有哈密顿回路吗?

定理7.12 有向HC是NP完全的.

定理7.13 HC是NP完全的.

定理7.14 TSP是NP完全的.

恰好覆盖: 给定有穷集 $A=\{a_1, a_2,\ldots, a_n\}$和 A的子集的集合$W=\{S_1, S_2,\ldots, S_m\}$, 问: 存在子集 $U\subseteq W$ 使得 U 中子集彼此不交且它们的并集等于A? 称U是 A的恰好覆盖.

例如, 设$A=\{1,2,3,4,5\}$, $S_1=\{1,2\}$, $S_2=\{1,3,4\}$, $S_3=\{2,4\}$, $S_4=\{2,5\}$, 则$\{S_2, S_4\}$是A的恰好覆盖.

定理7.15 恰好覆盖是NP完全的.

7.3.5 子集和、背包、装箱与双机调度

子集和: 给定正整数集合 $X=\{x_1, x_2,\ldots, x_n\}$及正整数$N$, 问存在$X$的子集 T, 使得T 中的元素之和等于N 吗?

装箱: 给定 n 件物品, 物品j的重量为正整数 w_j, $1\le j\le n$, 以及箱子数 K. 规定每只箱子装入物品的总重量不超过正整数 B, 问能用 K 只箱子装入所有的物品吗?

双机调度: 有2台机器和 n 项作业 $J_1, J_2,\ldots, J_n$, 这2台机器完全相同, 每一项作业可以在任一台机器上进行, 没有先后顺序, 作业 J_i 的处理时间为 t_i, $1 \le i \le n$, 截止时间为D, 所有 t_i和 D 都是正整数, 问能把 n 项作业分配给这2台机器, 在截止时间D内完成所有的作业吗?

NP完全性

定理7.16 子集和是NP完全的.

定理7.17 0-1背包是NP完全的

定理7.18 双机调度是NP完全的.

定理7.19 装箱是NP完全的.

注意: 0-1背包问题优化形式的动态规划算法, 其时间复杂度为$O(nB)$, 其中n是物品的个数, B是重量限制. 这不是多项式时间算法, 而是指数时间算法.

伪多项式时间算法: 算法的时间复杂度以 $|I|$ 和 $\max(I)$ 的某个二元多项式 $p(|I|, \max(I))$ 为上界, 其中 $\max(I)$ 是实例 I 中数的最大绝对值.

NP完全性理论的应用

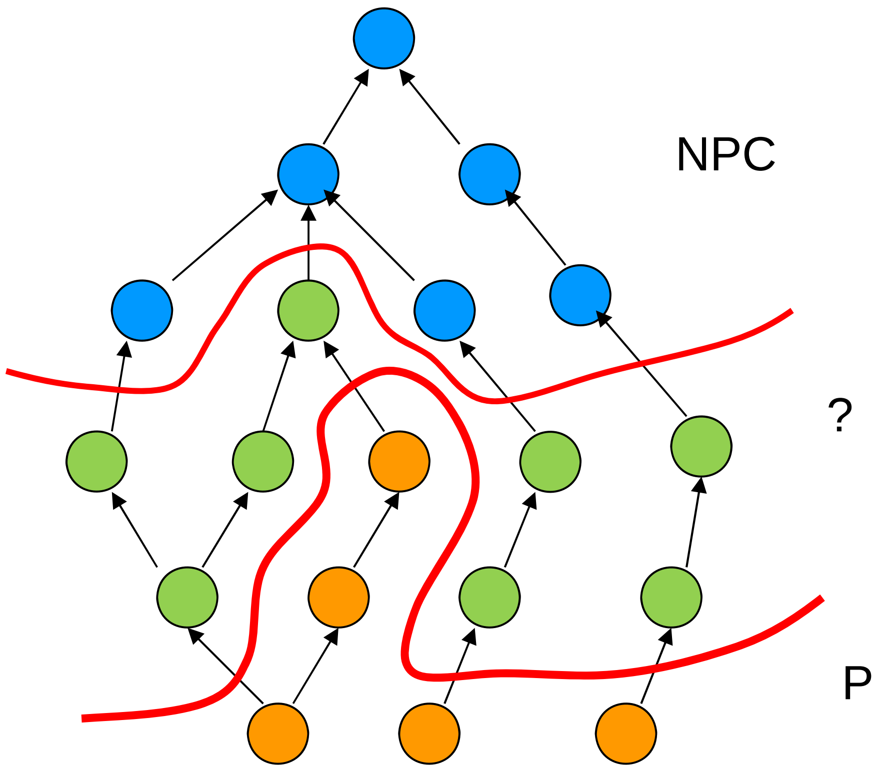
用NP完全性理论进行子问题分析

- 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 $\sigma: T \to \{0, 1, \ldots, D\}$满足下述条件, 则称 T 为可行调度.

(1) $\forall t \in T,\ \sigma(t)+l(t) \le D$

(2) $\forall i,\ 0 \le i \le D,\ |\{ t \in T : \sigma(t) \le i < \sigma(t)+l(t) \}| \le m$

(3) $\forall t, t' \in T,\ t \prec t' \Leftrightarrow \sigma(t)+l(t) \le \sigma(t')$

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

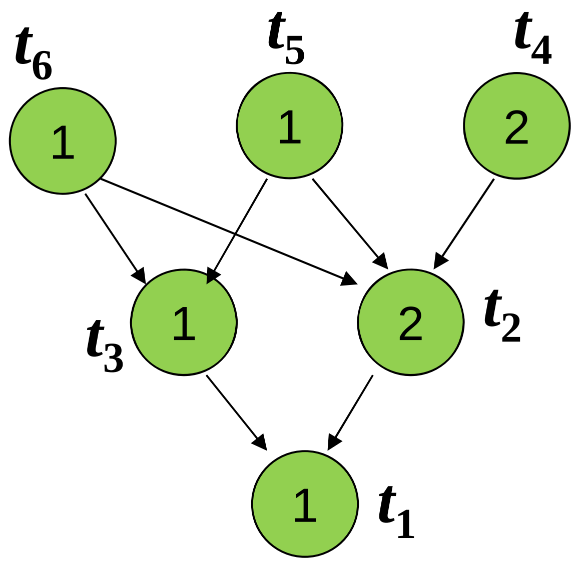
有偏序约束的任务必须按照约束先行

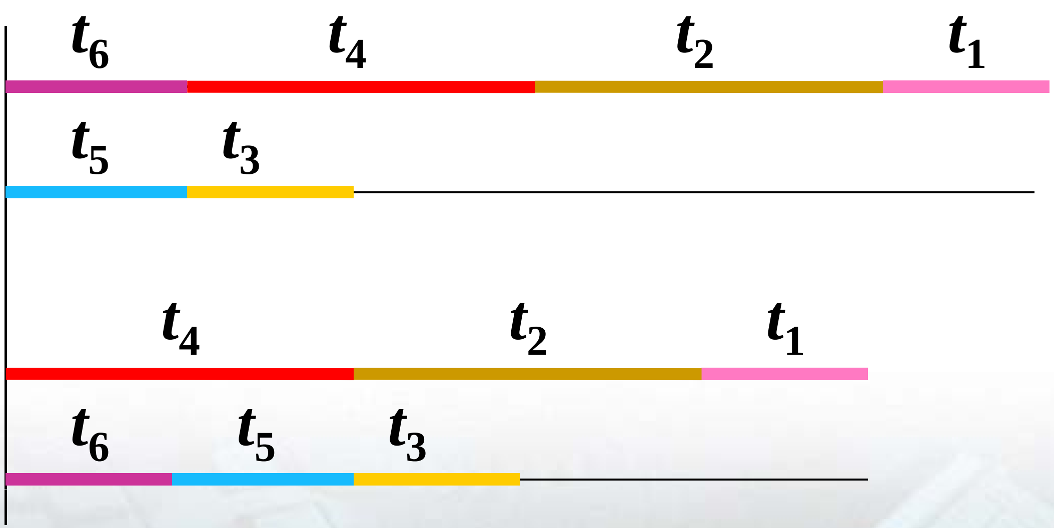
实例

任务集如图所示

$m=2$

求使得 D 最小的可行调度.





$D=6$

$D=5$

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

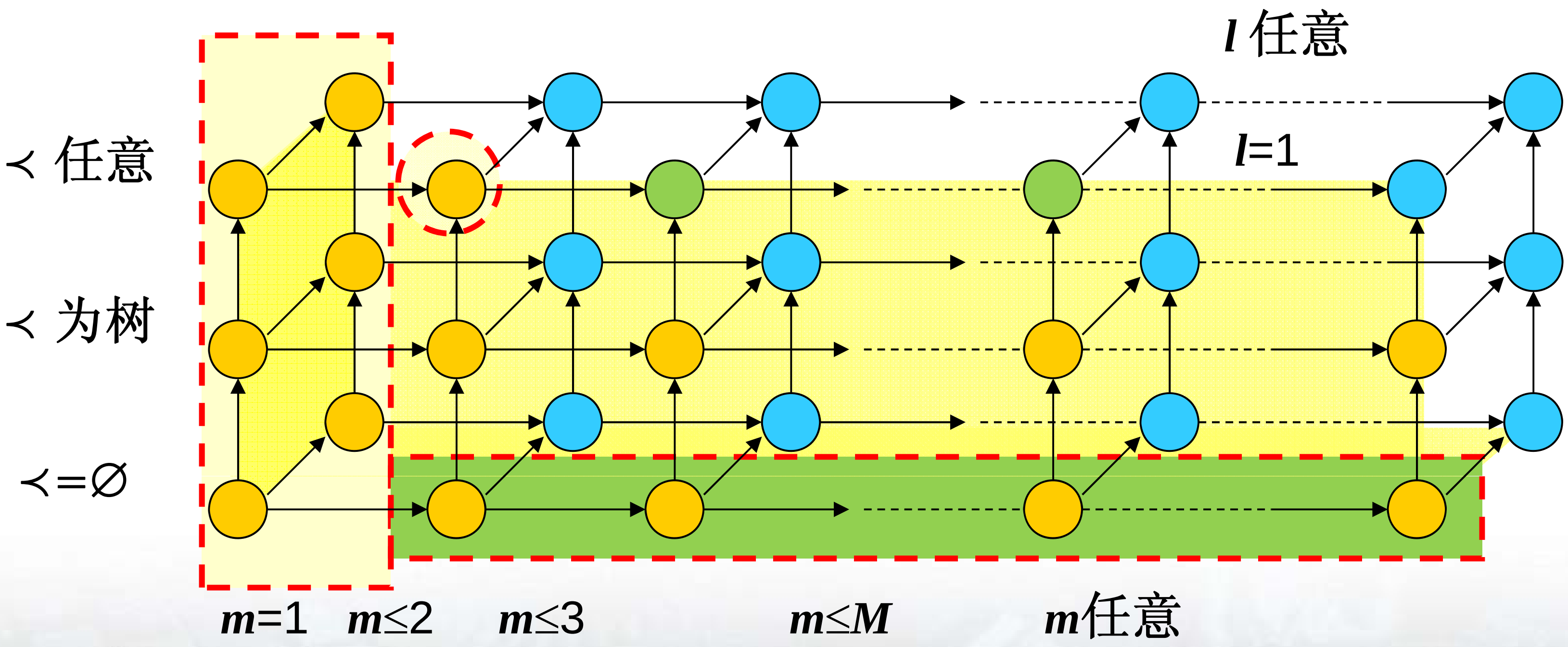
参数	限 制		
偏序	$\varnothing$	树	任意
m大小	$m \le 1, 2, \ldots,$ 某个常数		m 任意
l 大小	l 为常数		l 任意

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



搜索问题与优化问题的难度

Turing归约

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**. 这时也称π_1 Turing归约到 π_2，记作 $\pi_1 \propto_T \pi_2$.

NP难度

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至少和 NPC问题一样难.

许多NP完全问题对应的优化问题都是NP-hard

北京大學

实例—证明NP等价

例1 货郎问题（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 $C = \{ c_1, c_2, \ldots , c_m\}$

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 $\vartheta = < c_{\pi(1)}, \ldots ,c_{\pi(k)} >$

问：ϑ是否可以延伸成全长不超过 B 的全程旅行

$< c_{\pi(1)}, \ldots , c_{\pi(k)}, c_{\pi(k+1)}, \ldots , c_{\pi(m)} >$?

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$m \rightarrow m \times d$, $d = \max\{ d(c_i, c_j)\}$

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.

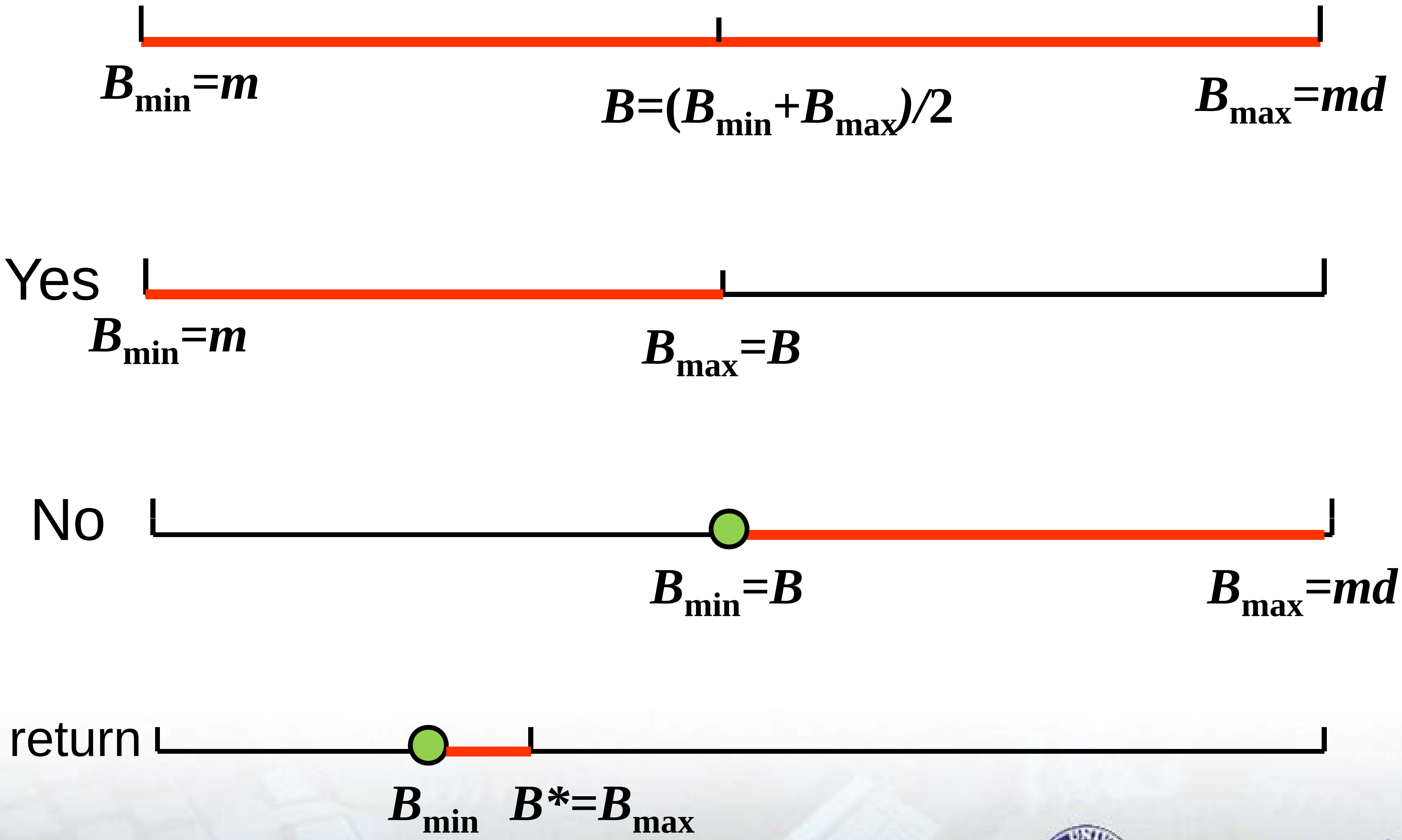
按照上面方法确定后面的其他顶点.

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. $i \leftarrow 2,\ M \leftarrow \{\, 2, 3, \ldots, m \,\}$
2. $j \leftarrow M$ 中的最小值
3. $\vartheta = \langle c_1, c_j \rangle$
4. $s(C, d, \vartheta, B^*)$
5. if 回答"Yes"
6. then $i \leftarrow i+1,\ M \leftarrow M - \{\, j \,\}$
7. else
8. 从ϑ中去掉 c_j
9. 从 M 中选择大于 j 的最小值 k
10. 将 c_k 加入到 ϑ 的最后项
11. $M \leftarrow M \cup \{\, j \,\}$
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

$$(m-2)+(m-3)+\ldots+1=\frac{(m-1)(m-2)}{2}$$

为 m 的多项式，而 **TSO** 的实例规模为 $m+\log B\mathrm{max}$，所以是从 **TSO** 到 **TSE** 的 **Turing**归约.

而 **TSE** 是 **NPC** 问题，因为判定问题 **TSP** 是 **TSE**的子问题，相当 $\vartheta = \langle c_1 \rangle$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.**

第8章 近似算法

8.1 近似算法及其近似比

8.2 多机调度问题

- 8.2.1 贪心的近似算法
- 8.2.2 改进的贪心近似算法

8.3 货郎问题

- 8.3.1 最邻近法
- 8.3.2 最小生成树法
- 8.3.3 最小权匹配法

8.4 背包问题

- 8.4.1 一个简单的贪心算法
- 8.4.2 多项式时间近似方案

8.1 近似算法及其近似比

近似算法: A是一个多项式时间算法且对组合优化问题Π的每一个实例I 输出一个可行解σ. 记A(I)=$c(\sigma)$, $c(\sigma)$是σ的值

最优化算法: 恒有A(I)=OPT(I), 即A总是输出 I 的最优解.

当Π是最小化问题时, 记 $r_A(I)=A(I)/OPT(I)$;

当Π是最大化问题时, 记 $r_A(I)=OPT(I)/A(I)$.

A的近似比为 r (A是r–近似算法): 对每一个实例 I, $r_A(I) \leq r$.

A具有常数比: r是一个常数.

可近似性分类

假设P≠NP , NP难的组合优化问题按可近似性可分成 3 类:

(1) 完全可近似的:对任意小的$\varepsilon>0$, 存在$(1+\varepsilon)$-近似算法.

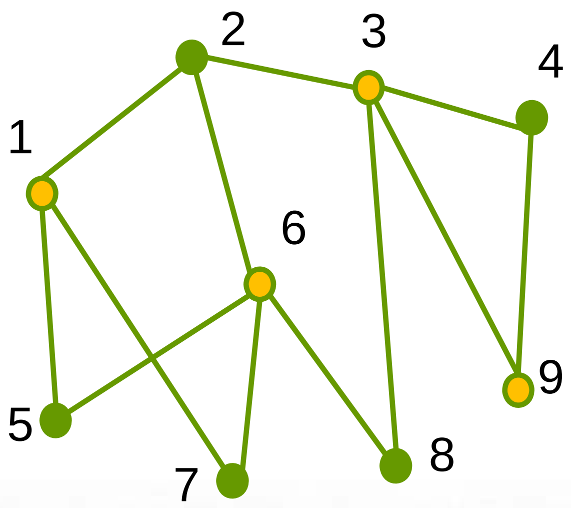
(2) 可近似的: 存在具有常数比的近似算法.

(3) 不可近似的: 不存在具有常数比的近似算法,

最小顶点覆盖问题

问题: 任给图$G=<V,E>$, 求G的顶点数最少的顶点覆盖.

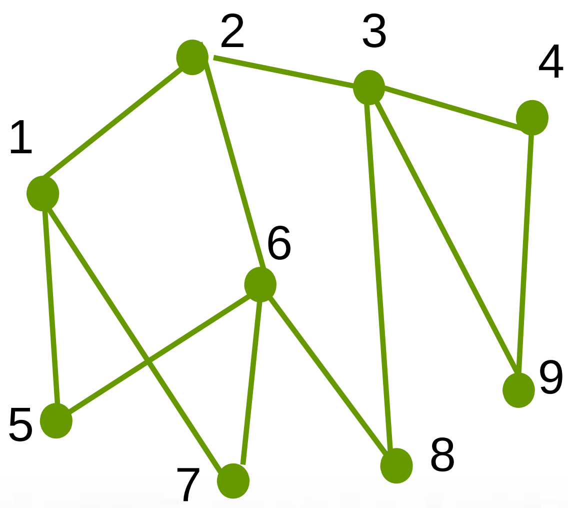
算法MVC: 开始时令$V'=\varnothing$. 任取一条边(u,v), 把u和v加入V'并删去u和v及其关联的边. 重复上述过程, 直至删去所有的边为止. V'为所求的顶点覆盖.



{1,2}

{1,2,3,4}

{1,2,3,4,5,6}



一个最优解：{1,3,6,9}， MVC的解：{1,2,3,4,5,6}

最小顶点覆盖问题

分析: 算法时间复杂度为$O(m)$, $m=|E|$.

记 $|V'| = 2k$, V'由 k 条互不关联的边的端点组成. 为了覆盖这 k 条边需要 k 个顶点, 从而 $\mathrm{OPT}(I) \geq k$. 于是,

$$\mathrm{MVC}(I)/\mathrm{OPT}(I) \leq 2k/k = 2.$$

又, 设图G由 k 条互不关联的边构成, 显然

$$\mathrm{MVC}(I) = 2k, \quad \mathrm{OPT}(I) = k,$$

这表明 MVC 的近似比不会小于2, 上面估计的MVC的近似比已不可能再进一步改进.

近似算法的分析

研究近似算法的两个基本方面——设计算法和分析算法的运行时间与近似比.

分析近似比

(1) 关键是估计最优解的值.

(2) 构造使算法产生最坏的解的实例. 如果这个解的值与最优值的比达到或者可以任意的接近得到的近似比(这样的实例称作紧实例), 那么说明这个近似比已经是最好的、不可改进的了; 否则说明还有进一步的研究余地.

(3) 研究问题本身的可近似性, 即在P≠NP(或其他更强)的假设下, 该问题近似算法的近似比的下界.

8.2 多机调度问题

多机调度问题:

任给有穷的作业集 A 和 m 台相同的机器, 作业 a 的处理时间为正整数 $t(a)$,每一项作业可以在任一台机器上处理. 如何把作业分配给机器才能使完成所有作业的时间最短? 即, 如何把 A 划分成 m 个不相交的子集 A_i 使得

$$\max\left\{\sum_{a\in A_i} t(a) \mid i=1,2,\cdots,m\right\}$$

最小?

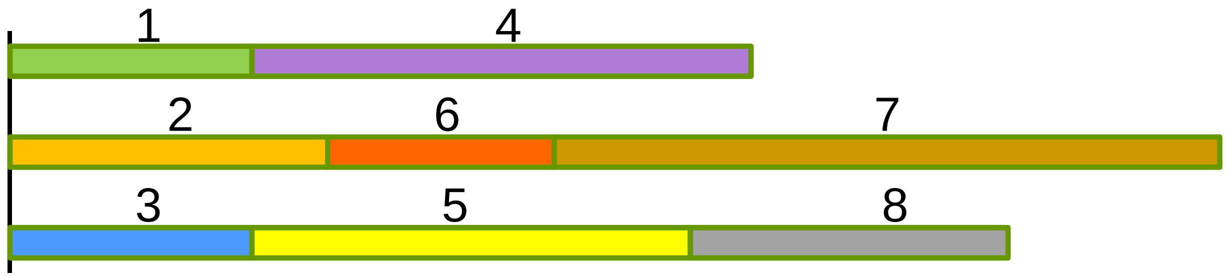
负载: 分配给一台机器的作业的处理时间之和.

贪心法 G-MPS: 按输入的顺序分配作业, 把每一项作业分配给当前负载最小的机器. 如果当前负载最小的机器有2台或 2台以上, 则分配给其中的任意一台.

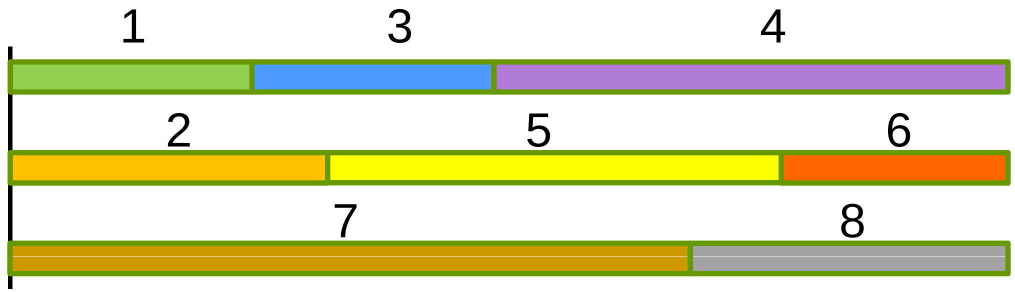
北京大学

实例

例如， **3** 台机器， **8** 项作业，处理时间为 **3, 4, 3, 6, 5, 3, 8, 4**



G-MPS的解
完成时间 15



最优解
完成时间 12

算法给出的分配方案是 **{1,4}, {2,6,7}, {3,5,8},**
负载分别为 **3+6=9，4+3+8=15，3+5+4=12**
最优的分配方案是 **{1,3,4}, {2,5,6}, {7,8},**
负载分别为 **3+3+6=12，4+5+3=12，8+4=12**

贪心法的性能

定理8.1 对多机调度问题的每一个有 m 台机器的实例 I，

$$\text{G-MPS}(I) \le \left(2-\frac{1}{m}\right)\text{OPT}(I).$$

证 显然，(1) $\text{OPT}(I) \ge \frac{1}{m}\sum_{a\in A} t(a)$， (2) $\text{OPT}(I) \ge \max_{a\in A} t(a)$.

设机器 M_j 的负载最大, 记作 $t(M_j)$. 又设 b 是最后被分配给机器 M_j 的作业. 根据算法, 在考虑分配 b 时 M_j 的负载最小, 故

$$t(M_j) - t(b) \le \frac{1}{m}\left(\sum_{a\in A} t(a) - t(b)\right).$$

北京大学

证 明

于是

$$
\begin{aligned}
&\mathrm{G-MPS}(I) = t(M_j) \\
&\le \frac{1}{m}\left(\sum_{a\in A} t(a) - t(b)\right) + t(b) \\
&= \frac{1}{m}\sum_{a\in A} t(a) + \left(1-\frac{1}{m}\right) t(b) \\
&\le \mathrm{OPT}(I) + \left(1-\frac{1}{m}\right)\mathrm{OPT}(I) \\
&= \left(2-\frac{1}{m}\right)\mathrm{OPT}(I).
\end{aligned}
$$

紧实例

M 台机器， $m(m-1)+1$ 项作业,

前 $m(m-1)$ 项作业的处理时间都为 1，最后一项作业的处理时间为m.

算法把前 $m(m-1)$项 作业平均地分配给 m 台机器, 每台 $m-1$ 项，最后一项任意地分配给一台机器.

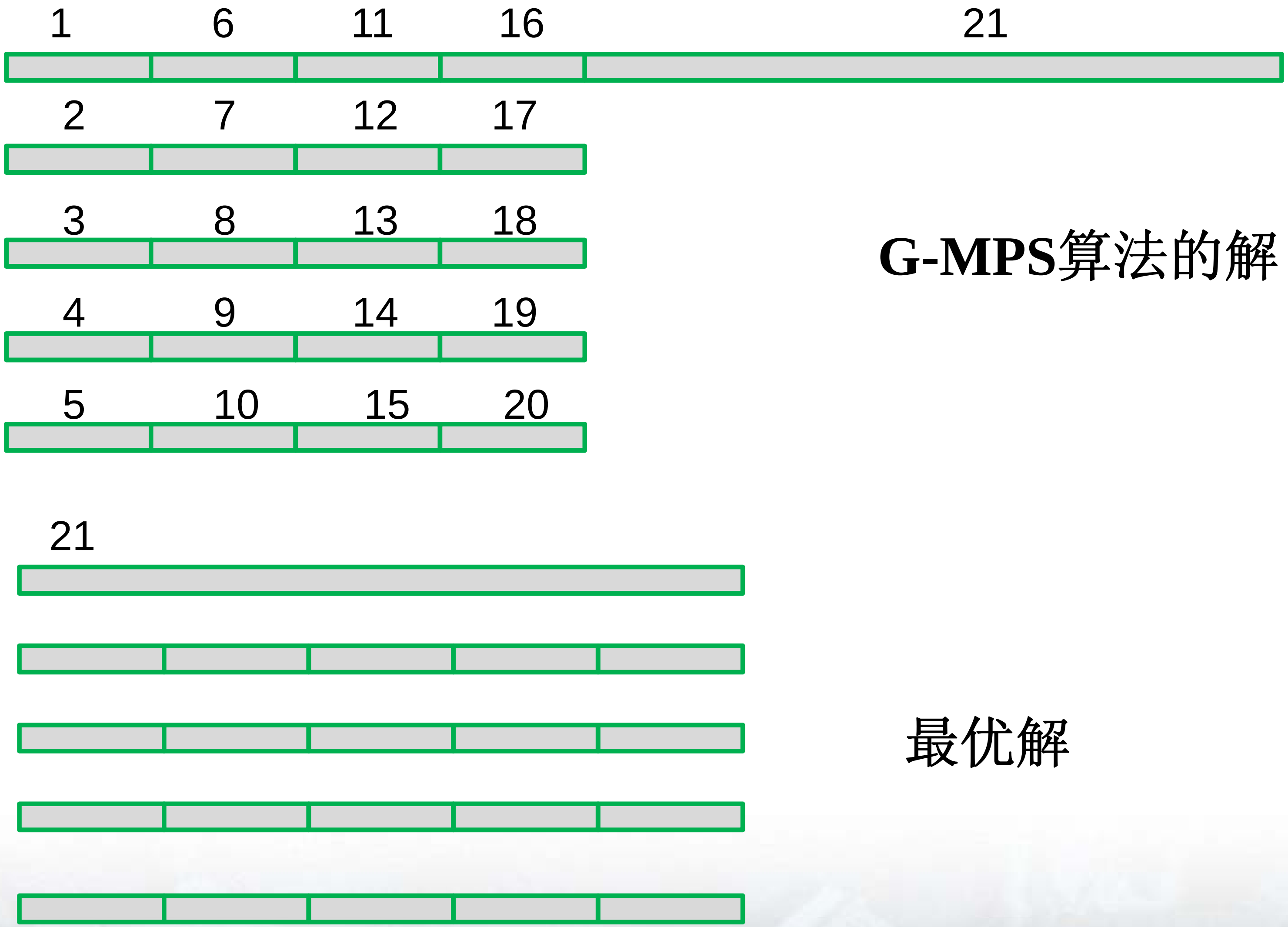
$$\text{G-MPS}(I) = 2m-1.$$

最优分配方案是把前 $m(m-1)$ 项作业平均地分配给 $m-1$ 台机器, 每台 m 项, 最后一项分配给留下的机器,

$$\text{OPT}(I) = m.$$

G-MPS是2-近似算法

m=5的紧实例



G-MPS算法的解

最优解

8.2.2 改进的贪心近似算法

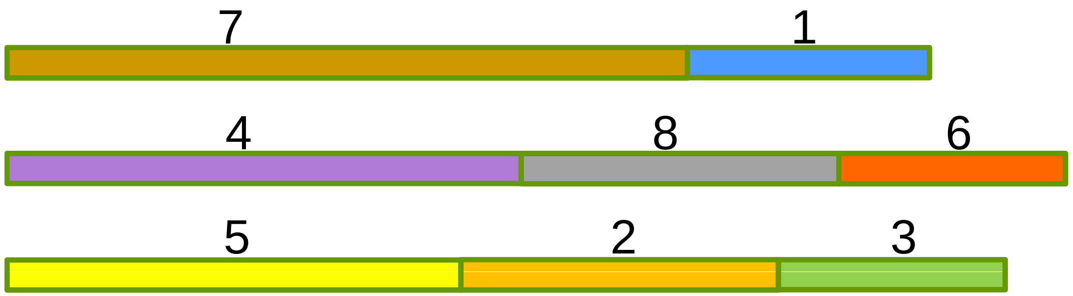
递降贪心法**DG-MPS**：首先按处理时间从大到小重新排列作业, 然后运用**G-MPS.**

例如对上一小节的紧实例得到最优解.

对另一个实例：先重新排序 **8, 6, 5, 4, 4, 3, 3, 3;**

3台机器的负载分别为 **8+3=11, 6+4+3=13, 5+4+3=12.**

比**G-MPS**的结果好.



DG-MPS的解
完成时间**13**

分析：**DG-MPS**增加排序时间$O(n\log n)$, 仍然是多项式时间

近似比

定理8.2 对多机调度问题的每一个有 m 台机器的实例 I,

$$\mathrm{DG-PMS}(I) \le \frac{1}{m}\left(\frac{3}{2}-\frac{1}{2m}\right)\mathrm{OPT}(I)$$

证 设作业按处理时间从大到小排列为 $a_1, a_2,\ldots, a_n$, 仍考虑负载最大的机器 M_j 和最后分配给 M_j 的作业 a_i.

(1) M_j 只有一个作业, 则 $i = 1$, 必为最优解.

(2) M_j 有 2 个或 2个以上作业, 则 $i \ge m+1$, $\mathrm{OPT}(I) \ge 2t(a_i)$

$$\mathrm{DG-MPS}(I) = t(M_j) \le \frac{1}{m}\left(\sum_{k=1}^{n} t(a_k) - t(a_i)\right) + t(a_i)$$

$$= \frac{1}{m}\sum_{k=1}^{n} t(a_k) + \left(1-\frac{1}{m}\right)t(a_i) \le \mathrm{OPT}(I) + \left(1-\frac{1}{m}\right)\cdot\frac{1}{2}\mathrm{OPT}(I)$$

$$= \left(\frac{3}{2}-\frac{1}{2m}\right)\mathrm{OPT}(I)$$

8.3 货郎问题

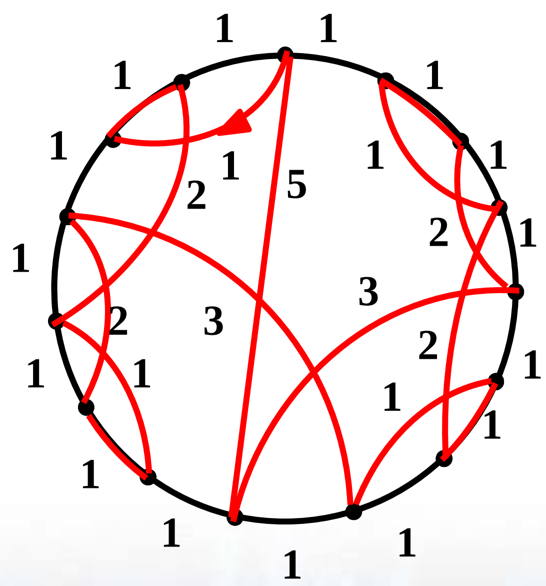
本节考虑满足三角不等式的货郎问题

8.3.1 最邻近法

最邻近法NN：从任意一个城市开始，在每一步取离当前所在城市最近的尚未到过的城市作为下一个城市. 若这样的城市不止一个，则任取其中的一个. 直至走遍所有城市，最后回到开始出发的城市.

一个NN性能很坏的实例 I，

$NN(I)=21$，$OPT(I)=15$



最邻近法的性能

定理8.3 对于货郎问题所有满足三角不等式的 n 个城市的实例 I, 总有

$$\mathrm{NN}(I) \leq \frac{1}{2}\left(\lceil \log_2 n \rceil + 1\right)\mathrm{OPT}(I).$$

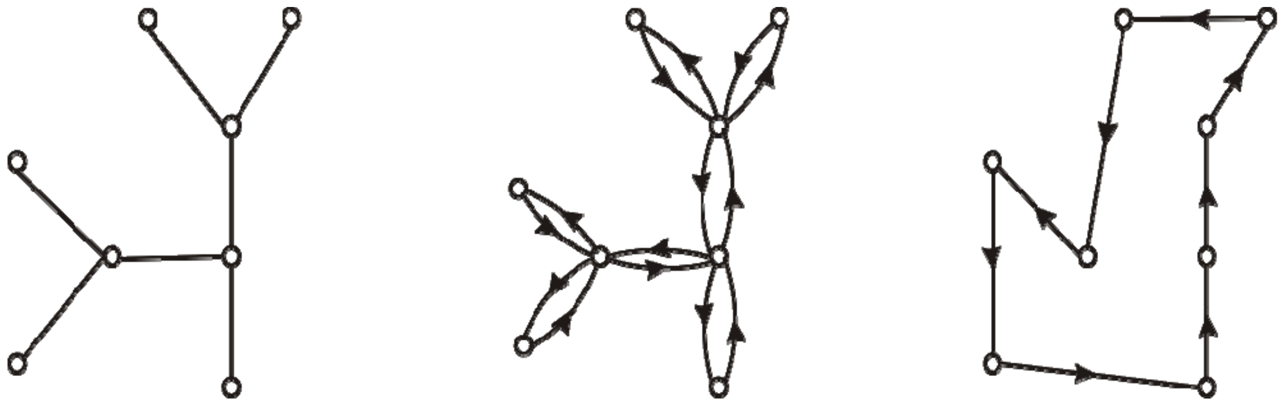
而且, 对于每一个充分大的 n, 存在满足三角不等式的 n 个城市的实例 I 使得

$$\mathrm{NN}(I) > \frac{1}{3}\left(\log_2(n+1) + \frac{4}{3}\right)\mathrm{OPT}(I).$$

8.3.2 最小生成树法

最小生成树法MST: 首先, 求图的一棵最小生成树T. 然后, 沿着 T 走两遍得到图的一条欧拉回路. 最后, 顺着这条欧拉回路, 跳过已走过的顶点, 抄近路得到一条哈密顿回路.

例



求最小生成树和欧拉回路都可以在多项式时间内完成, 故算法是多项式时间的.

最小生成树法的性能

定理8.4 对货郎问题的所有满足三角不等式的实例 ***I*****,**

$$\mathbf{MST}(I) < \mathbf{2OPT}(I).$$

证 因为从哈密顿回路中删去一条边就得到一棵生成树**,** 故 ***T*** 的权小于**OPT(*I*).** 沿 ***T*** 走两遍的长小于**2OPT(*I*).** 因为满足三角不等式**,** 抄近路不会增加长度**,** 故

$$\mathbf{MST}(I) < \mathbf{2OPT}(I).$$

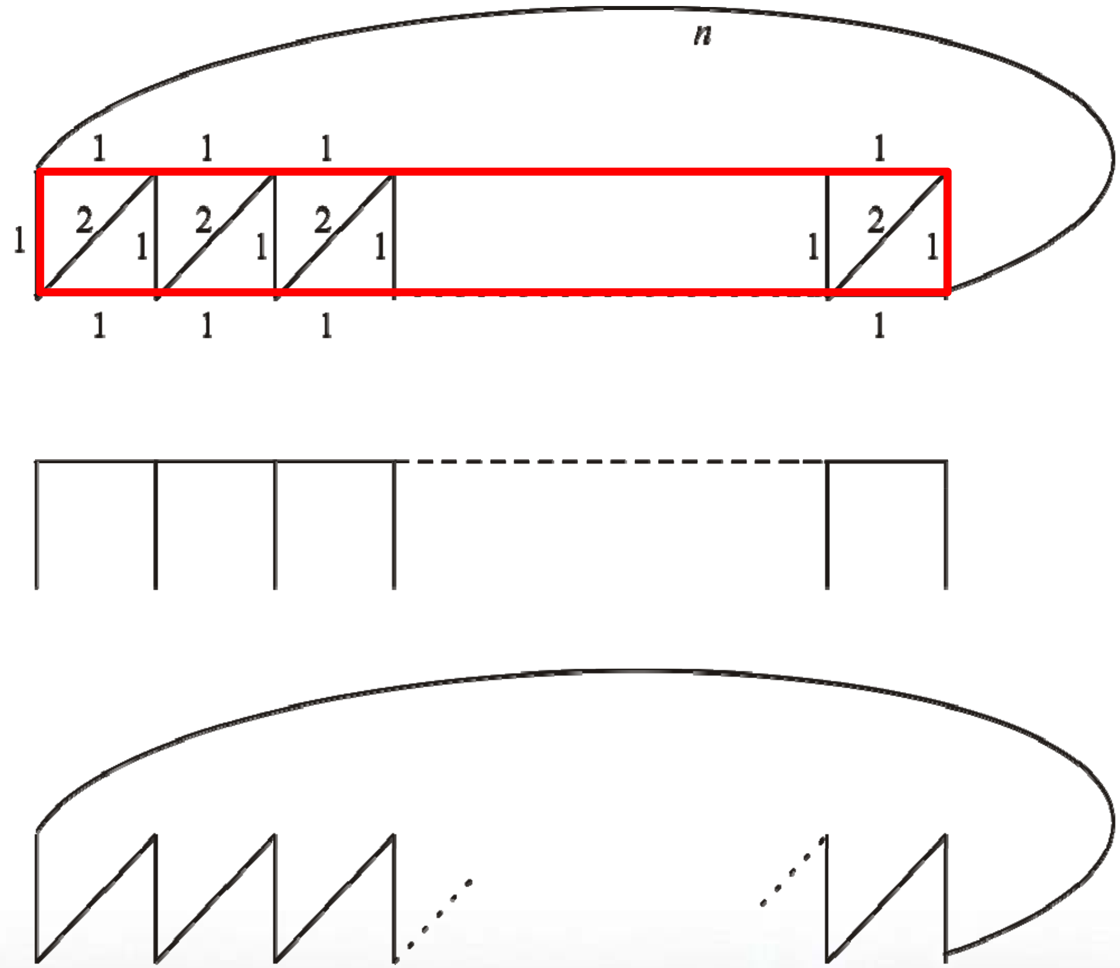
MST是**2-**近似算法**.**

紧实例

$\mathrm{OPT}(I) = 2n$

$\mathrm{MST}(I) = 4n\text{-}2$

$$= \left(2 - \frac{1}{n}\right)\mathrm{OPT}(I)$$

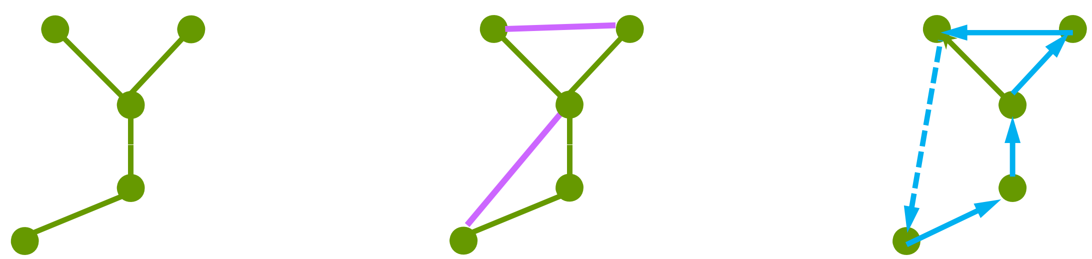


8.3.3 最小权匹配法

最小权匹配法 **MM**:

首先求图的一棵最小生成树 T.

记 T 的所有奇度顶点在原图中的导出子图为 H, H 有偶数个顶点, 求 H 的最小匹配 M. 把 M 加入 T 得到一个欧拉图, 求这个欧拉图的欧拉回路; 最后, 沿着这条欧拉回路, 跳过已走过的顶点, 抄近路得到一条哈密顿回路.



求任意图最小权匹配的算法是多项式时间的, 因此 MM 是多项式时间的.

最小权匹配法的性能

定理8.5 对货郎问题的所有满足三角不等式的实例 I,

$$\mathbf{MM}(I) < \frac{3}{2}\mathbf{OPT}(I)$$

证 由于满足三角不等式, 导出子图 H 中的最短哈密顿回路 C 的长度不超过原图中最短哈密顿回路的长度 OPT(I). 沿着 C 隔一条边取一条边, 得到 H 的一个匹配. 总可以使这个匹配的权不超过 C 长的一半. 因此, H 的最小匹配 M 的权不超过 OPT(I)/2, 求得的欧拉回路的长小于 (3/2)OPT(I). 抄近路不会增加长度, 得证

$$\mathbf{MM}(I) < (3/2)\mathbf{OPT}(I).$$

MM是3/2 -近似算法

货郎问题的难度

定理8.6 货郎问题(不要求满足三角不等式) 是不可近似的, 除非 P = NP.

证 假设不然, 设A是货郎问题的近似算法, 其近似比 $r \le K$, K是常数. 任给图 $G = \langle V,E \rangle$, 如下构造货郎问题的实例 I_G:
城市集V, $\forall u,v \in V$,

若 $(u,v) \in E$, 则令 $d(u,v)=1$; 否则令 $d(u,v)=Kn$, 其中 $|V| = n$. 若G 有哈密顿回路, 则

$$\mathrm{OPT}(I_G) = n, \quad \mathrm{A}(I_G) \le r\,\mathrm{OPT}(I_G) \le Kn$$

否则

$$\mathrm{OPT}(I_G) > Kn, \quad \mathrm{A}(I_G) \ge \mathrm{OPT}(I_G) > Kn$$

所以, G 有哈密顿回路当且仅当 $\mathrm{A}(I_G) \le Kn$

北京大学

证明

于是, 下述算法可以判断图 G 是否有哈密顿回路:

首先构造货郎问题的实例 I_G, 然后对I_G 运用算法 **A.** 若 **A**(I_G) $\le Kn$, 则输出“**Yes**”; 否则输出“**No**”.

由于K是固定的常数, 构造 I_G 可在 $O(n^2)$ 时间内完成且 $|I_G| = O(n^2)$. **A**是多项式时间的, **A** 对 I_G 可在 n 的多项式时间内完成计算, 所以上述算法是 **HC** 的多项式时间算法. 而 **HC** 是 **NP** 完全的, 推得 **P=NP.**

8.4 背包问题

0-1背包问题的优化形式:

任给 n 件物品和一个背包，物品 i 的重量为 w_i, 价值为 v_i, $1\le i \le n$, 背包的重量限制为 B, 其中 w_i, v_i 以及 B 都是正整数.

把哪些物品装入背包才能在不超过重量限制的条件下使得价值最大? 即, 求子集 $S^*\subseteq\{1,2,\ldots,n\}$ 使得

$$\sum_{i\in S^*} v_i = \max\left\{\sum_{i\in S} v_i \mid \sum_{i\in S} w_i \le B, S \subseteq \{1,2.\cdots,n\}\right\}.$$

8.4.1 一个简单的贪心算法

贪心算法G-KK

1. 按单位重量的价值从大到小排列物品. 设

$$v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n.$$

2. 顺序检查每一件物品, 只要能装得下就将它装入背包, 设装入背包的总价值为 V.

3. 求 $v_k = \max\{ v_i \mid i = 1, 2, \ldots, n \}$.

若$v_k > V$, 则将背包内的物品换成物品 k.

实例 (w_i,v_i): (3,7), (4,9), (5,9), (2,2); B=6.

G-KK给出的解是装入(3,7)和(2,2), 总价值为9. 若把第3件物品改为(5,10), 则装入第3件, 总价值为10.

这两个实例的最优解都是装入(4,9)和(2,2), 总价值为11.

G-KK的性能

定理8.7 对**0-1**背包问题的任何实例 I, 有

$$\mathrm{OPT}(I) < 2\text{G-KK}(I).$$

证 设物品 l 是第一件未装入背包的物品, 由于物品按单位重量的价值从大到小排列, 故有

$$\begin{aligned}\mathrm{OPT}(I) &< \text{G-KK}(I) + v_l \\ &\le \text{G-KK}(I) + v_{\max} \\ &\le 2\,\text{G-KK}(I).\end{aligned}$$

G-KK是2-近似算法.

8.4.2 多项式时间近似方案

算法 PTAS 输入$\varepsilon>0$和实例 I.

1. 令 $m = \lceil 1/\varepsilon \rceil$.
2. 按单位重量的价值从大到小排列物品. 设

$$v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n.$$

3. 对每一个 $t=1,2,\ldots,m$ 和 t 件物品, 检查这 t 件物品的重量之和. 若它们的重量之和不超过B, 则接着用G-KK把剩余的物品装入背包.
4. 比较得到的所有装法, 取其中价值最大的作为近似解.

PTAS是一簇算法. 对每一个固定的 $\varepsilon > 0$, PTAS是一个算法, 记作 PTAS_ε .

PTAS的性能

定理8.8 对每一个ε>0和0-1背包问题的实例I,

$$\mathrm{OPT}(I) < (1+\varepsilon)\ \mathrm{PTAS}_{\varepsilon}(I),$$

且 $\mathrm{PTAS}_{\varepsilon}$的时间复杂度为$O(n^{1/\varepsilon+2})$.

证 设最优解为S^*. 若 $|S^*| \le m$, 则算法必得到 S^*. 设$|S^*| > m$. 考虑计算中以 S^*中 m 件价值最大的物品为基础, 用G-KK得到的结果 S. 设物品 l 是 S^* 中第一件不在 S 中的物品, 在此之前 G-KK装入的不属于 S^* 的物品 (肯定有这样的物品, 否则应该装入物品 l) 的单位重量的价值都不小于 v_l/w_l, 当然也不小于 S^* 中所有没有装入的物品的单位重量的价值, 故有 $\mathrm{OPT}(I) < \sum_{i\in S} v_i + v_l$. 又, S 包括 S^* 中 m 件价值最大的物品, 它们的价值都不小于 v_l, 故又有 $v_l \le \sum_{i\in S} v_i/m$.

多项式时间近似方案

$$\mathrm{OPT}(I) < \sum_{i\in S} v_i + v_l \le \sum_{i\in S} v_i + \sum_{i\in S} v_i/m$$
$$\le (1+1/m)\mathrm{PTAS}_\varepsilon(I) \le (1+\varepsilon)\mathrm{PTAS}_\varepsilon(I)$$

时间复杂度. 从 n 件物品中取 t 件(t =1,2,…,m), 所有可能取法的个数为

$$c_n^1 + c_n^2 + \cdots + c_n^m \le m \cdot \frac{n^m}{m!} \le n^m .$$

对每一种取法，G-KK的运行时间 为$O(n)$，故算法的时间复杂度为 $O(n^{m+1}) = O(n^{1/\varepsilon+2})$.

多项式时间近似方案: 以ε>0和问题的实例作为输入 I, 对每一个固定的 $\varepsilon > 0$, 算法是 $1+\varepsilon$ - 近似的.

第9章 随机算法

Las Vegas 型随机算法

- 随机快速排序
- 随机选择
- 随机 *n*后放置

Monte Carlo型随机算法

- 主元素测试
- 串相等测试
- 模式匹配
- 素数测试

随机算法的分类与局限性

随机快速排序算法

算法9.1 随机快速排序算法

输入：包含 ***n*** 个元素的数组

输出：经过排序的 ***n*** 个元素的数组

1. 若数组包含 **0** 或 **1** 个元素则返回

2. 从数组中随机选择一个元素作为枢轴元素

3. 把数组元素分为三个子数组，并且按照 ***A*, *B*, *C*** 顺序排列

A：包含比枢轴元素小的元素；

B：包含与枢轴元素相等的元素；

C：包含比枢轴元素大的元素.

4．对 ***A*** 和 ***C*** 递归地执行上述步骤.

算法分析

定理9.1 设数组含n个不同元素，对任意常数$\varepsilon>0$，随机快速排序算法的期望比较次数

$$T(n) \le 2\, n \ln n.$$

证明一：求解递推式

随机选取枢轴元素，其位于排序后第 i 位置 (i=1,2,…,n)的概率是1/n，A和C的元素数分别是 i 个和 n–i–1个, 得

$$T(n) = (n-1) + \frac{1}{n}\sum_{i=1}^{n-1}[T(i) + T(n-i-1)]$$

解为Θ(nlog$n$). 可归纳证明精确上界是 $T(n) \le 2n \ln n$.

$$T(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} 2i\ln i \le (n-1) + \frac{2}{n}\int_1^n 2x\ln x dx$$

$$\le (n-1) + \frac{2}{n}(n^2\ln n - \frac{n^2}{2} + \frac{1}{2}) \le 2n\ln n$$

随机选择算法

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

```
if  p=r  then  return A[p]
i ←Random(p, r)
以 A[i] 为标准划分 A
j ←划分后小于等于 A[i] 的数构成数组的大小
if  k≤j
    then return  RandSelect (A, p, p+j-1, k)
    else return  RandSelect (A, p+j, r, k-j)
```

时间期望值估计

假设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，算法的期望时间为

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

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

归纳证明 $T(n)\le cn$. 归纳步骤如下：假设对 $k<n$ 命题为真，

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

$$=\frac{3cn}{4}-\frac{c}{2}+tn \le \frac{3cn}{4}+tn = (\frac{3}{4}+\frac{t}{c})cn \le cn,$$ 取 $c \ge 4t$ 即可

n后放置的随机选择

算法BoolQueen(n)

1. $k \leftarrow 1$ // k 放皇后的行号
2. $count \leftarrow 0$ // $count$ 放好的皇后数
3. while $k \le n$ do
4.   for $i \leftarrow 1$ 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 $S \neq \varnothing$ then
8. $j \leftarrow \text{Random}(1,|S|)$
9. $x_k \leftarrow S[j]$
10. $count \leftarrow count+1$
11. $k \leftarrow k+1$
12.   else $k \leftarrow n+1$
13. return $count$

*n*后问题的随机算法

算法QueenLV(*n*)　　//重复调用随机算法BoolQueen

1. $p \leftarrow$ BoolQueen(n)
2. while $p<n$ do
3. 　$p \leftarrow$ BoolQueen(n)

改进算法---与回朔相结合

设 $stopVegas \le n$, 表示用QueenLV算法放置的皇后数

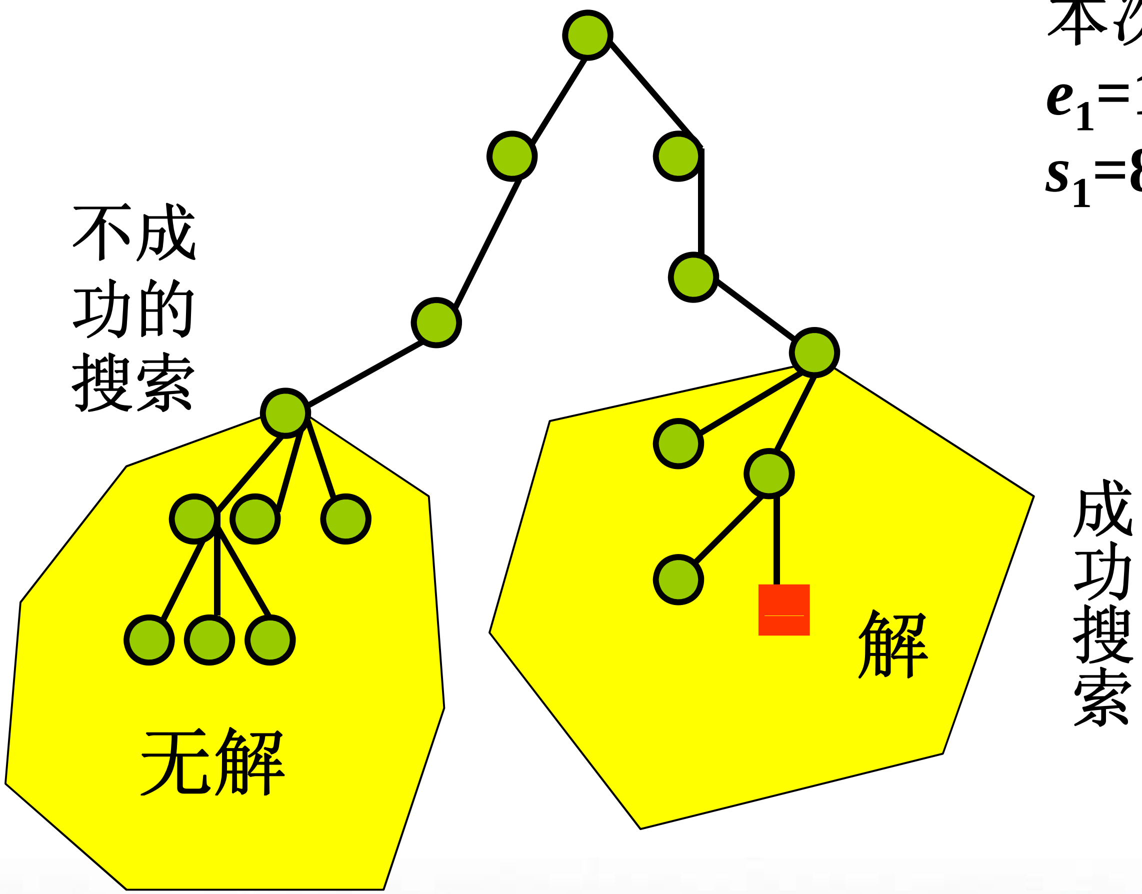
剩下 n - $stopVegas$ 个皇后用回朔方法放置

$stopVegas = 0$ 时是完全的回朔算法

$stopVegas = n$ 时是完全的Las Vegas算法

成功搜索与不成功搜索

本次抽样：
e_1=10
s_1=8



改进算法的分析

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

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

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

LasVegas型随机算法总结

- 特点
  - 通过修改确定性算法得到，一般将算法的某步的确定型选择变成随机选择
  - 一次运行可能得不到解；若得到解，则解一定是正确的
  - 改进途径：与确定型算法相结合
  - 有可能改进确定型算法平均情况下的时间复杂度
- 有效的 LasVegas 算法
  - 运行时间是随机变量，期望运行时间是输入的多项式且总能给出正确答案的随机算法

主元素测试

问题

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：如果存在主元素则输出 "**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

若回答true：则T 存在主元素，算法正确；若回答false，T 仍可能存在主元素，算法可能出错. 回答正确概率大于1/2.

算法 BoolMajority(T,n)

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

BoolMajority 算法正确的概率为

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

调用 k 次Majority算法正确的概率为

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

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

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

$$(\tfrac{1}{2})^k \le \varepsilon \Rightarrow k\log\tfrac{1}{2} \le \log\varepsilon \Rightarrow -k \le \log\varepsilon$$
$$\Rightarrow k \ge -\log\varepsilon \Rightarrow k \ge \left\lceil \log\tfrac{1}{\varepsilon} \right\rceil$$

出错概率不超过ε 的算法——**MCMajority**

算法 MCMajority(*T*,*n*,ε)

1. $k \leftarrow \lceil \log(1/\varepsilon) \rceil$
2. **for** $i \leftarrow 1$ **to** k
3.   **if** Majority(T,n) **then return true**
4. **return false**

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.

北京大学

指纹产生

设 x 和 y 的二进制表示为正整数 $I(x), I(y)$

选择素数 p，指纹函数为

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

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

存在问题：

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

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

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 $I_p(x)$ 给 B
3. B 测试是否 $I_p(x) = I_p(y)$

出错的必要条件：

x 的位数等于 y 的位数

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

有关素数的性质

函数 $\pi(t)$：小于 t 的不同的素数个数，

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

两个相关结果：

(1) 素数定理 $\pi(t) \approx t/\ln t$

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

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

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

例如 $k = 984$，整除 984 的素数只有 2, 3, 41

北京大学

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 $M \ge 2n^2$, 一次测试的出错概率估计：

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

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

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** $I_p(x) \neq I_p(y)$
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

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

总共传送 82 位

模式匹配

问题：输入二进制串 $X = x_1 x_2 \ldots x_n$，$Y = y_1 y_2 \ldots y_m$，$m \le n$

输出：若Y在 X 中，Y 出现的第一个位置；否则为 **"0"**

算法一：顺序比较

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，输出 Y 的首位置 j；否则将 Y 的位置向后移动一个字符，重复原来过程.

运行时间：$O(mn)$

算法二：利用有限状态自动机的模式匹配算法

(Knuth,Morris,Pratt), Introduction to Algorithms

运行时间：$O(m+n)$

随机算法

算法三 利用串比较的随机算法

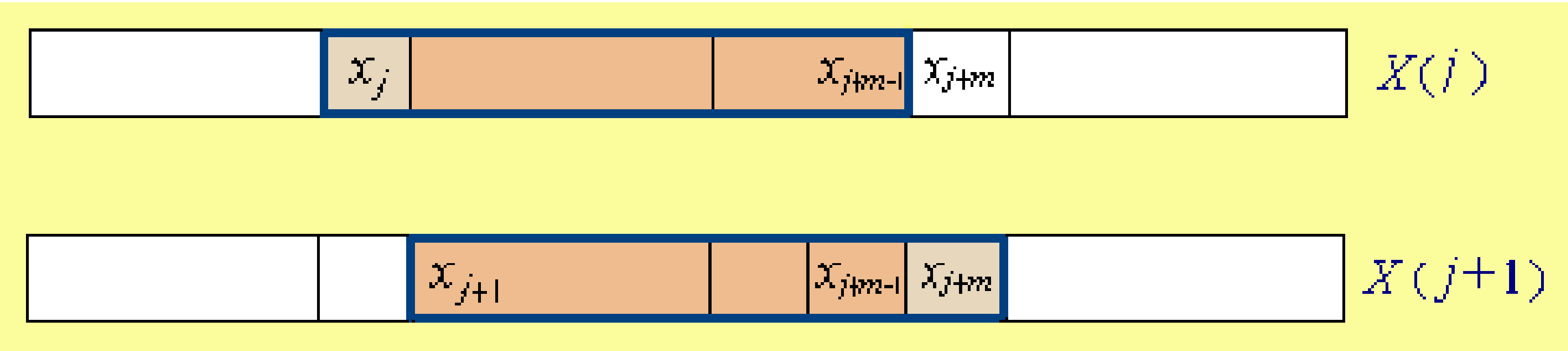
设计思想：设 $X(j) = x_j x_{j+1} \ldots x_{j+m-1}$，把 $X(j)$ $(j=1,2, \ldots, n-m+1)$ 与 Y 逐个字符的比较，改成对指纹 $I_p(X(j))$ 与 $I_p(Y)$ 的比较

$X(j)$与$X(j+1)$的关系

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

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

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



算法三的关键技术

由 $I_p(X(j))$ 求 $I_p(X(j+1))$ 的公式

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

$$W_p = 2^m (\bmod p)$$

该公式说明：由 $I_p(X(j))$ 计算 $I_p(X(j+1))$ 仅需要常数时间

公式的证明：

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

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

令 $W_p = 2^m (\bmod p)$

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

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

时间复杂度为 $O(m+n)$

$W_p, I_p(Y), I_p(X(1))$ 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

出错条件：
$$Y \neq X(j) \wedge I_p(Y) = I_p(X(j))$$
$$\Leftrightarrow p \,\Big|\, \prod_{\{j \mid Y \neq X(j)\}} |I(Y) - I(X(j))|$$

出错概率：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

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

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. $\quad y \leftarrow y^2$;
4. $\quad$ if $d_j=1$ then $y \leftarrow xy$
5. return y

实例

x^{101}: $d_2=1, d_1=0, d_0=1$

$y=1$

$j=2$	$j=1$	$j=0$
$y=1$	$y=x^2$	$y=x^4$
$y=x$		$y=x^5$

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

$T(n) = O(k\log^2 n) = O(\log^3 n)$ 以位乘作为基本运算

Fermart小定理:测试原理

定理1： 如果 n为素数，则对所有的正整数 $a \neq 0 \pmod{n}$有

$$a^{n-1} \equiv 1 \pmod{n}$$

素数测试原理：检测 $2^{n-1} \equiv 1 \pmod{n}$. 如是，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

问题：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)

改进算法：随机选取2--n-1中的数，进行测试. 如取a=3，$3^{340}(\bmod 341) \equiv 56$，341不是素数.

算法Ptest2(n)

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

- Fermat 小定理是必要条件，不是充分条件，满足该条件的也可能是合数. 对所有与 n 互素的正整数 a 都满足条件的合数 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 为合数的概率至少为1/2.

素数的另一个必要条件

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

证明　$x^2(\mathrm{mod}\ n) \equiv 1$

$\Leftrightarrow x^2-1 \equiv 0(\mathrm{mod} n)$

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

$\Leftrightarrow x+1 \equiv 0$ 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.

判别方法：如果方程有非平凡的根，则 n 为合数.

例如：$x^2(\mathrm{mod}\ 12) \equiv 1 \Leftrightarrow x = 1$ 或 $x = 11$ 或 $x = 5$ 或 $x=7$

结论：由于5 和 7 是非平凡的根，12是合数

测试方法

设n为奇素数，存在q, m使得 $n-1=2^q m$, ($q\geq1$).

构造序列：

$$a^m(\bmod n), a^{2m}(\bmod n), a^{4m}(\bmod n), \ldots, a^{2^q m}(\bmod n)$$

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

测试方法：

1. 对于任意 i（$i = 0,1,\ldots, q-1$）, 判断

$$a^{2^i m}(\bmod n)$$

   是否为 1 和 n-1, 且它的后一项是否为1.
2. 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.
3. 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. $\quad m \leftarrow m/2$
4. $\quad q \leftarrow q+1$
5. until m是奇数

运行时间：$O(\log n)$

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

性能分析：

- 1 次测试运行时间 $O(\log^3 n)$
- 可证明1次测试出错的概率至多 1/2. 重复运行k次，可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 **PremalityTest(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

Las Vegas型与 Monte Carlo型随机算法

- **LasVegas型随机算法**
  - 如果得到解，总是给出正确的结果，区别只在于运行时间的长短.
  - 拉斯维加斯型随机算法的运行时间本身是一个随机变量
  - 期望运行时间是输入规模的多项式且总是给出正确答案的随机算法称为有效的拉斯维加斯型算法.
- **Monte Carlo型随机算法**
  - 这种算法有时会给出错误的答案.
  - 其运行时间和出错概率都是随机变量，通常需要分析算法的出错概率.
  - 多项式时间内运行且出错概率不超过1/3的随机算法称为有效的蒙特卡洛型算法

单侧错误和双侧错误

- 弃真型单侧错误
  - 当算法宣布接受时，结果一定是对的
  - 当算法宣布拒绝时，结果有可能是错的.
  - 例如主元素测试算法
- 取伪型单侧错误
  - 当算法宣布拒绝时，结果一定是对的
  - 而当算法宣布接受时，结果有可能是错的
  - 例如素数测试
- 双侧错误
  - 在所有的输入上同时出现上述两种不同的错误

随机算法的分类与局限性

- 拉斯维加斯型随机算法
  - 零错误概率多项式时间算法(有效的)，**ZPP**
- 蒙特卡洛型随机算法
  - 错误概率有界的有效算法(多项式时间)，**BPP**
  - 弃真型单侧错误概率有界的有效算法，**RP**
  - 取伪型单侧错误概率有界的有效算法，**coRP**

$$\mathbf{P} \subseteq \mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP} \subseteq \mathbf{BPP}$$

- 随机算法的局限性
  - 错误概率有界的多项式时间随机算法不太可能解决NP完全问题

第10章 处理难解问题的策略

- 对问题施加限制
  固定参数算法
- 改进指数时间算法
  3SAT、指数时间假设
- 启发式方法
  启发式方法、随机化策略、重启策略、模拟退火
- 平均情形的复杂性
  ***G*(*n*,*p*)**、哈密顿回路、**DistNP**完全
- 难解算例的生成

对问题施加限制

- **SAT问题**

  二元可满足性（**2SAT**）属于**P**

  HornSAT：输入限制为霍恩公式（析取式中正文字，即不带否定号的变量）至多出现一次，属于**P**

- 图的问题

问题	**P**	**NPC**
VC	$D \le 2$	$D \ge 3$
HC	**2**	**3**
顶点三着色	**3**	**4**
反馈顶点集	**2**	**3**
团	给定D	任意

固定参数算法

- 通常把优化问题转化为判定问题时，都会在输入中引入一个参数，这是固定参数算法中参数的来源之一.
- 输入中带有一个参数 k，当输入规模为 n时运行时间为 $O(f(k)n^c)$ 的算法，这里的 $f(k)$是与 n 无关的函数，c 是与 n 和 k 都无关的常数.
- 例

  VC: 给定图G, 正整数 K(不超过G的顶点数)，问是否存在不超过 K 的顶点覆盖？

  固定常数 k，输入为(G, k). 穷举所有 k元 顶点子集，看看是否存在顶点覆盖. 算法复杂度大约是

$$O(knC_n^k)=O(kn^{k+1})$$

  存在$O(2^k kn)$的算法.

北京大學

改进的指数时间算法

- O^*：表示忽略了多项式因子. 如$O^*(2^n)=O(n^{O(1)}2^n)$
- 当一个问题的蛮力算法为$O^*(2^n)$时间时，对任何满足$1 < c < 2$的常数c，时间复杂度为$O^*(c^n)$的指数时间算法称为非平凡的指数时间算法，或改进的指数时间算法.
- 可证明在$O^*(1.8393^n)$时间内正确求解3SAT，截止到2010年底 最好结果：$O^*(1.321^n)$时间的随机算法和$O^*(1.439^n)$时间的确定型算法.
- 任意色数的图的顶点着色问题都有$O^*(3^n)$的算法. 背包问题有比$O^*(2^{n/2})$更好的算法. 货郎问题也有比$O^*(2^n)$更好的算法.

其他处理难解问题的策略

- 启发式方法（Heuristics）：目前无法从理论上给出任何性能保证，但在实践中效果良好，就把这类方法统称为启发式方法（Heuristics）.

  常用的启发式方法主要包括：回溯与分支限界法、局部搜索法（随机化策略、重启策略、模拟退火）、遗传算法等.

- 平均情况下的复杂度

  有些NP完全问题在平均复杂性度量下是易解的，哈密顿回路问题的平均情况下对图$G(n,1/2)$有$O(n^3)$时间的算法.

- 难解算例生成: 确定紧的实例
- 基于统计物理的消息传递算法