

本周教学内容

算法课程主要内容及有关概念

算法研究内容

计算复杂性理论:
货郎问题
背包问题
双机调度问题

问题复杂度概念:
排序问题

算法设计与分析:
调度问题
投资问题

算法的有关概念

算法的伪码表示

几类重要函数的性质

有关函数渐近的界的定理

时间复杂度函数的表示: 函数渐近的界

算法及其时间复杂度的定义

两个例子：调度 问题与投资问题

例1：调度问题

问题 有 n 项任务，每项任务加工时间已知。
从 0 时刻开始陆续安排到一台机器上加工。
每个任务的完成时间是从 0 时刻到任务加工截止的时间。
求：总完成时间（所有任务完成时间之和）
最短的安排方案。

实例

任务集 $S = \{1, 2, 3, 4, 5\}$,

加工时间： $t_1=3$, $t_2=8$, $t_3=5$, $t_4=10$, $t_5=15$

贪心法的解

算法：按加工时间 (3,8,5,10,15) 从小到大安排

解： 1, 3, 2, 4, 5



总完成时间：

$$\begin{aligned} t &= 3 + (3+5) + (3+5+8) + (3+5+8+10) + (3+5+8+10+15) \\ &= 3 \times 5 + 5 \times 4 + 8 \times 3 + 10 \times 2 + 15 \\ &= 94 \end{aligned}$$

问题建模

输入：任务集： $S=\{1, 2, \dots, n\}$,

第 j 项任务加工时间： $t_j \in \mathbb{Z}^+$, $j=1, 2, \dots, n$.

输出：调度 I , S 的排列 i_1, i_2, \dots, i_n ,

目标函数： I 的完成时间, $t(I) = \sum_{k=1}^n (n-k+1)t_{i_k}$

解 I^* ：使得 $t(I^*)$ 达到最小，即

$$t(I^*) = \min \{ t(I) \mid I \text{ 为 } S \text{ 的排列} \}$$

贪心算法

设计策略：加工时间短的先做

算法：根据加工时间从小到大排序,依次加工

算法正确性：对所有输入实例都得到最优解

证：假如调度 f 第 i, j 项任务相邻且有逆序，
即 $t_i > t_j$ 。交换任务 i 和 j 得到调度 g ，



总完成时间 $t(g) - t(f) = t_j - t_i < 0$

直觉不一定是正确的

反例

有4 件物品要装入背包, 物品重量和价值如下:

标号	1	2	3	4
重量 w_i	3	4	5	2
价值 v_i	7	9	9	2

背包重量限制是 6, 问如何选择物品, 使得不超重的情况下装入背包的物品价值达到最大?

实例的解

贪心法： 单位重量价值大的优先，总重不超 6

按照 $\frac{v_i}{w_i}$ 从大到小排序：1, 2, 3, 4

$$\frac{7}{3} > \boxed{\frac{9}{4}} > \frac{9}{5} > \boxed{\frac{2}{2}}$$

贪心法的解：{ 1, 4 }, 重量 5, 价值为 9.

更好的解：{ 2, 4 }, 重量 6, 价值 11.

算法设计

1. 问题建模
2. 选择什么算法？如何描述这个方法？
3. 这个方法是否对所有实例都得到最优解？
如何证明？
4. 如果不是，能否找到反例？

例2：投资问题

问题： m 元钱，投资 n 个项目. 效益函数 $f_i(x)$, 表示第 i 个项目投 x 元的效益, $i=1, 2, \dots, n$. 求如何分配每个项目的钱数使得总效益最大?

实例： 5 万元，投资给 4 个项目，效益函数：

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

建模

输入: $n, m, f_i(x), i = 1, 2, \dots, n, x = 1, 2, \dots, m$

解: n 维向量 $\langle x_1, x_2, \dots, x_n \rangle$, x_i 是第 i 个项目的钱数, 使得下述条件满足:

目标函数 $\max \sum_{i=1}^n f_i(x_i),$

约束条件 $\sum_{i=1}^n x_i = m, \quad x_i \in \mathbb{N}$

蛮力算法

对所有满足下述条件的向量 $\langle x_1, x_2, \dots, x_n \rangle$

$$x_1 + x_2 + \dots + x_n = m$$

x_i 为非负整数, $i = 1, 2, \dots, n$

计算相应的效益

$$f_1(x_1) + f_2(x_2) + \dots + f_n(x_n)$$

从中确认效益最大的向量.

实例计算

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

$$x_1 + x_2 + x_3 + x_4 = 5$$

$$s_1 = \langle 0, 0, 0, 5 \rangle, v(s_1) = 24$$

$$s_2 = \langle 0, 0, 1, 4 \rangle, v(s_2) = 25$$

$$s_3 = \langle 0, 0, 2, 3 \rangle, v(s_3) = 32$$

...

$$s_{56} = \langle 5, 0, 0, 0 \rangle, v(s_{56}) = 15$$

解: $s = \langle 1, 0, 3, 1 \rangle$,

最大效益: $11 + 30 + 20 = 61$

蛮力算法的效率

方程 $x_1 + x_2 + \dots + x_n = m$ 的非负整数解
 $\langle x_1, x_2, \dots, x_n \rangle$ 的个数估计:

可行解表示成 0-1 序列: m 个1, $n-1$ 个 0

$\underbrace{1 \dots 1}_{x_1 \uparrow} 0 \underbrace{1 \dots 1}_{x_2 \uparrow} 0 \dots 0 \underbrace{1 \dots 1}_{x_n \uparrow}$

$n=4, m=7$

可行解 $\langle 1, 2, 3, 1 \rangle$

\Leftrightarrow 序列 $1 0 1 1 0 1 1 1 0 1$

蛮力算法的效率

序列个数是输入规模的指数函数

$$\begin{aligned} & C(m+n-1, m) \\ &= \frac{(m+n-1)!}{m!(n-1)!} \\ &= \Omega((1+\varepsilon)^{m+n-1}) \end{aligned}$$



有没有更好的算法？

小结

问题求解的关键

- 建模：对输入参数和解给出形式化或半形式化的描述
- 设计算法：
采用什么算法设计技术
正确性——是否对所有的实例都得到正确的解
- 分析算法——效率

问题计算复杂度 的界定:排序问题

例3 排序算法的效率

以元素比较作基本运算

算法	最坏情况下	平均情况下
插入排序	$O(n^2)$	$O(n^2)$
冒泡排序	$O(n^2)$	$O(n^2)$
快速排序	$O(n^2)$	$O(n\log n)$
堆排序	$O(n\log n)$	$O(n\log n)$
二分归并排序	$O(n\log n)$	$O(n\log n)$

插入排序的插入操作

前面已经排好，插入2

输入

5	7	1	3	6	2	4
---	---	---	---	---	---	---

插入2

1	3	5	6	7	2	4
---	---	---	---	---	---	---

插入后

1	2	3	5	6	7	4
---	---	---	---	---	---	---

插入排序运行实例

输入	5	7	1	3	6	2	4
初始	5	7	1	3	6	2	4
插入7	5	7	1	3	6	2	4
插入1	1	5	7	3	6	2	4
插入3	1	3	5	7	6	2	4
插入6	1	3	5	6	7	2	4
插入2	1	2	3	5	6	7	4
插入4	1	2	3	4	5	6	7

冒泡排序的一次巡回

巡回前

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

巡回

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

巡回后

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

冒泡排序运行实例

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

巡回1	5	1	3	6	2	4	7	8
-----	---	---	---	---	---	---	---	---

巡回2	1	3	5	2	4	6	7	8
-----	---	---	---	---	---	---	---	---

巡回3	1	3	2	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

巡回4	1	2	3	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

巡回5	1	2	3	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

快速排序一次递归运行

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

交换1

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

交换2

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

划分

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

子问题

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

二分归并排序运行实例

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

划分

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

递归
排序

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

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

合并后的输出

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

问题的计算复杂度分析

问题:

哪个排序算法效率最高?

是否可找到更好的排序算法?

排序问题计算难度如何?

其他问题的计算复杂度

问题计算复杂度估计方法

n^2

插入排序
冒泡排序
快速排序

$n\log n$

堆排序
归并排序

?

更好的算
法下界



那个排序算法效率最高?

排序问题的难度?

小结

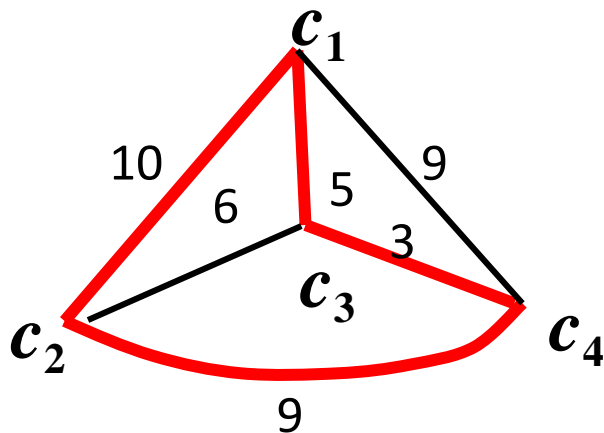
- 几种排序算法简介
 - 插入排序
 - 冒泡排序
 - 快速排序
 - 归并排序
- 排序问题的难度估计——界定什么是最好的排序算法

货郎问题与计算 复杂性理论

例4 货郎问题

问题:

有 n 个城市, 已知任两个城市之间的距离. 求一条每个城市恰好经过1次的回路, 使得总长度最小.



建模与算法

- 输入

有穷个城市的集合 $C = \{c_1, c_2, \dots, c_n\}$,

距离 $d(c_i, c_j) = d(c_j, c_i) \in \mathbb{Z}^+$, $1 \leq i < j \leq n$

- 解: $1, 2, \dots, n$ 的排列 k_1, k_2, \dots, k_n 使得:

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

- 现状: 至今没找到有效的算法

0-1背包问题

0-1背包问题：有 n 个件物品要装入背包，第 i 件物品的重量 w_i ，价值 v_i ， $i=1,2,\dots,n$. 背包最多允许装入的重量为 B ，问如何选择装入背包的物品，使得总价值达到最大？

实例： $n=4$ ， $B=6$ ，物品的重量和价值如下

标号	1	2	3	4
重量 w_i	3	4	5	2
价值 v_i	7	9	9	2

0-1背包问题建模

问题的解: 0-1向量 $\langle x_1, x_2, \dots, x_n \rangle$

$x_i=1 \Leftrightarrow$ 物品 i 装入背包

目标函数 $\max \sum_{i=1}^n v_i x_i$

约束条件 $\sum_{i=1}^n w_i x_i \leq B$

$x_i = 0, 1, i = 1, 2, \dots, n$

双机调度

双机调度问题：有 n 项任务，任务 i 的加工时间为 $t_i, t_i \in \mathbb{Z}^+, i=1,2,\dots,n$. 用两台相同的机器加工，从0时刻开始计时，完成时间是后停止加工机器的停机时间. 问如何把这些任务分配到两台机器上，使得完成时间达到最小？

实例：任务集 $S = \{1,2,3,4,5,6\}$

$$t_1=3, t_2=10, t_3=6, \underline{t_4=2}, t_5=1, t_6=7$$

解：机器 1 的任务：1, 2, 4

机器 2 的任务：3, 5, 6

完成时间： $\max\{3+10+2, 6+1+7\}=15$

双机调度建模

解: 0-1向量 $\langle x_1, x_2, \dots, x_n \rangle$, $x_i=1$ 表示任务 i 分配到第一台机器, $i=1,2,\dots,n$.

不妨设机器1的加工时间 \leq 机器2的加工时间令 $T=t_1+t_2+\dots+t_n$, $D=\lfloor T/2 \rfloor$, 机器1的加工时间不超过 D , 且达到最大.



如何对该问题建模? 目标函数与约束条件是什么?

NP-hard问题

- 这样的问题有数千个，大量存在于各个应用领域.
- 至今没找到有效算法：现有的算法的运行时间是输入规模的指数或更高阶函数.
- 至今没有人能够证明对于这类问题不存在多项式时间的算法.
- 从是否存在多项式时间算法的角度看，这些问题彼此是等价的. 这些问题的难度处于可有效计算的边界.

Algorithm + Data Structure = Programming

好的算法

提高求解问题的效率

节省存储空间

算法的研究目标

问题→建模并寻找算法

算法→算法的评价

算法类→问题复杂度估计

问题类→能够求解的边界

算法设计技术

算法分析方法

问题复杂度分析

计算复杂性理论

课程主要内容

近似算法

随机算法

NP 完全理论简介

算法分析与问题的计算复杂性

分治
策略

动态
规划

贪心
算法

回溯与
分支限界

数学基础、数据结构

计算复杂性理论:

NP完全理论

其他算法

算法设计:

算法分析方法

算法设计技术

基础知识

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

提高学生素质和分析问题解决问题的能力，
培养计算思维

小结

- NP-hard问题的几个例子：货郎问题
0-1背包问题、双机调度问题等
- NP-hard问题的计算现状
- 计算复杂性理论的核心——NP完全理论
- 算法研究的主要内容及重要意义

算法及其 时间复杂度

问题及实例

- 问题

需要回答的一般性提问，通常含若干参数

- 问题描述

定义问题参数(集合,变量,函数,序列等)

说明每个参数的取值范围及参数间的关系

定义问题的解

说明解满足的条件(优化目标或约束条件)

- 问题实例

参数的一组赋值可得到问题的一个实例

算法

- 算法

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 解决问题 P

把问题 P 的任何实例作为算法 A 的输入

每步计算是确定性的

A 能够在有限步停机

输出该实例的正确的解

基本运算与输入规模

- **算法时间复杂度**: 针对指定**基本运算**, 计数算法所做运算次数
- **基本运算**: 比较, 加法, 乘法, 置指针, 交换...
- **输入规模**: 输入串编码长度
通常用下述参数度量: 数组元素多少, 调度问题的任务个数, 图的顶点数与边数等.
- **算法基本运算次数可表为输入规模的函数**
- **给定问题和基本运算就决定了一个算法类**

输入规模

- 排序：数组中元素个数 n
- 检索：被检索数组的元素个数 n
- 整数乘法：两个整数的位数 m, n
- 矩阵相乘：矩阵的行列数 i, j, k
- 图的遍历：图的顶点数 n , 边数 m
- ...

基本运算

- 排序: 元素之间的比较
- 检索: 被检索元素 x 与数组元素的比较
- 整数乘法: 每位数字相乘(位乘) 1 次
 m 位和 n 位整数相乘要做 mn 次位乘
- 矩阵相乘: 每对元素乘 1 次
 $i \times j$ 矩阵与 $j \times k$ 矩阵相乘要做 ijk 次乘法
- 图的遍历: 置指针
- ...

算法的两种时间复杂度

对于相同输入规模的不同实例，算法的基本运算次数也不一样，可定义两种时间复杂度

最坏情况下的时间复杂度 $W(n)$

算法求解输入规模为 n 的实例所需要的最长时间

平均情况下的时间复杂度 $A(n)$

在给定同样规模为 n 的输入实例的概率分布下，算法求解这些实例所需要的平均时间

$A(n)$ 计算公式

平均情况下的时间复杂度 $A(n)$

设 S 是规模为 n 的实例集

实例 $I \in S$ 的概率是 P_I

算法对实例 I 执行的基本运算次数是 t_I

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

在某些情况下可以假定每个输入实例概率相等

例子：检索

检索问题

输入：非降顺序排列的数组 L , 元素数 n ,
数 x

输出： j

若 x 在 L 中, j 是 x 首次出现的下标;

否则 $j = 0$

基本运算： x 与 L 中元素的比较

顺序检索算法

$j=1$, 将 x 与 $L[j]$ 比较. 如果 $x=L[j]$, 则算法停止, 输出 j ; 如果不等, 则把 j 加1, 继续 x 与 $L[j]$ 的比较, 如果 $j>n$, 则停机并输出0.

实例

1	2	3	4	5
---	---	---	---	---

$x = 4$, 需要比较 4 次

$x = 2.5$, 需要比较 5 次

最坏情况的时间估计

不同的输入有 $2n + 1$ 个，分别对应：

$$x = L[1], x = L[2], \dots, x = L[n]$$

$$x < L[1], L[1] < x < L[2], L[2] < x < L[3], \dots, L[n] < x$$

最坏情况下时间： $W(n) = n$

最坏的输入： x 不在 L 中或 $x = L[n]$
要做 n 次比较

平均情况的时间估计

输入实例的概率分布：

假设 x 在 L 中概率是 p ,且每个位置概率相等

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

等差数
列求和

当 $p=1/2$ 时,

$$A(n) = \frac{n+1}{4} + \frac{n}{2} \approx \underline{\frac{3n}{4}}$$

改进顺序检索算法

$j=1$, 将 x 与 $L[j]$ 比较. 如果 $x=L[j]$, 则算法停止, 输出 j ; 如果 $x > L[j]$, 则把 j 加1, 继续 x 与 $L[j]$ 的比较; 如果 $x < L[j]$, 则停机并输出0. 如果 $j > n$, 则停机并输出 0.

实例

1	2	3	4	5
---	---	---	---	---

$x = 4$, 需要比较 4 次

$x = 2.5$, 需要比较 3 次

时间估计

最坏情况下: $W(n) = n$

平均情况下

输入实例的概率分布: 假设 x 在 L 中每个位置与空隙的概率都相等



改进检索算法平均时间复杂度是多少?

小结:

- 算法最坏和平均情况下的时间复杂度定义
- 如何计算上述时间复杂度



算法的伪码表示

算法的伪码描述

赋值语句: \leftarrow

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: //...

例：求最大公约数

算法 **Euclid** (m, n)

输入：非负整数 m, n , 其中 m 与 n 不全为 0

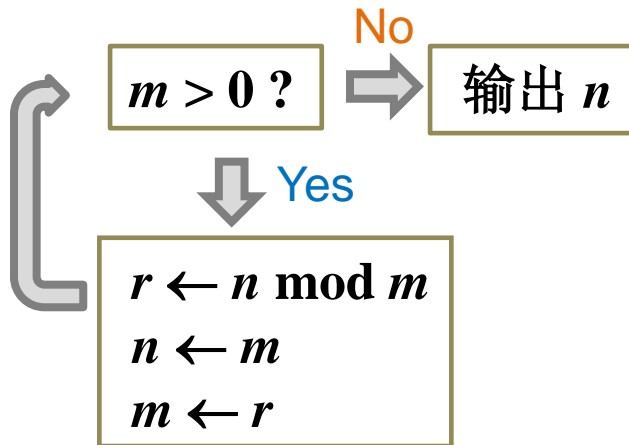
输出： m 与 n 的最大公约数

1. while $m > 0$ do
2. $r \leftarrow n \bmod m$
3. $n \leftarrow m$
4. $m \leftarrow r$
5. return n

运行实例: $n=36, m=15$

while	n	m	r
第1次	36	15	6
第2次	15	6	3
第3次	6	3	0
	3	0	0

↓
输出3



例：改进的顺序检索

算法 **Search (L, x)**

输入：数组 $L[1..n]$, 元素从小到大排列, 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 $j \leq n$ and $x > L[j]$ do $j \leftarrow j+1$
3. if $x < L[j]$ or $j > n$ then $j \leftarrow 0$
4. return j

例：插入排序

算法 **Insert Sort** (A, n)

输入： n 个数的数组 A

输出： 按照递增顺序排好序的数组 A

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

运行实例

2	4	1	5	3
---	---	---	---	---

$j = 3, x = A[3] = 1$

$i = 2, A[2] = 4$

$i > 0, x < A[2] \quad \checkmark$

2	4	4	5	3
---	---	---	---	---

$A[3] = 4, i = 1, x = 1$

$i > 0, x < A[1] \quad \checkmark$

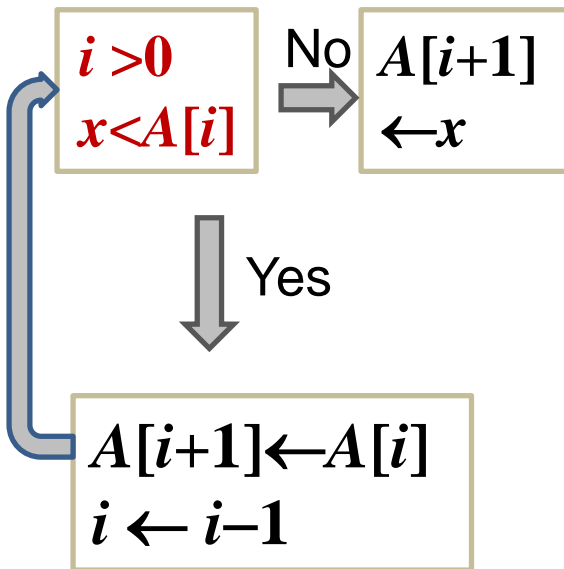
2	2	4	5	3
---	---	---	---	---

$A[2] = 2, i = 0, x = 1$

$i > 0 \quad \times$

1	2	4	5	3
---	---	---	---	---

4. while $i > 0$ and $x < A[i]$ do
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i - 1$
7. $A[i+1] \leftarrow x$



例：二分归并排序

MergeSort (A, p, r)

输入：数组 $A[p..r]$

输出：按递增顺序排序的数组 A

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)

MergeSort有递归调用,也调用Merge过程

例：算法A的伪码

算法 A

输入：实数的数组 $P[0..n]$ ，实数 x

输出： y

1. $y \leftarrow P[0]; \text{ power} \leftarrow 1$
2. for $i \leftarrow 1$ to n do
3. $\text{power} \leftarrow \text{power} * x$
4. $y \leftarrow y + P[i] * \text{power}$
5. return y



算法
A计
算什么值
？

对 $i = 1, 2, \dots, n$ \Rightarrow

$power \leftarrow power * x$
 $y \leftarrow y + P[i] * power$

	i	$power$	y
初值		1	$P[0]$
循环	1	x	$P[0] + P[1]*x$
	2	x^2	$P[0] + P[1]*x + P[2]*x^2$
	3	x^3	$P[0] + P[1]*x + P[2]*x^2 + P[3]*x^3$
			...

输入 $P[0..n]$ 是 n 次多项式 $P(x)$ 的系数
算法 A 计算该多项式在 x 的值

小结

用伪码表示算法

- 伪码不是程序代码，只是给出算法的主要步骤
- 伪码中有哪些关键字？
- 伪码中允许过程调用

函数的渐近的界

大O符号

定义： 设 f 和 g 是定义域为自然数集 \mathbf{N} 上的函数. 若存在正数 c 和 n_0 , 使得对一切 $n \geq n_0$ 有

$$0 \leq f(n) \leq c g(n)$$

成立, 则称 $f(n)$ 的渐近的上界是 $g(n)$, 记作

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

例子

设 $f(n) = n^2 + n$, 则

$f(n) = O(n^2)$, 取 $c = 2$, $n_0 = 1$ 即可

$f(n) = O(n^3)$, 取 $c = 1$, $n_0 = 2$ 即可

1. $f(n) = O(g(n))$, $f(n)$ 的阶不高于 $g(n)$ 的阶.
2. 可能存在多个正数 c , 只要指出一个即可.
3. 对前面有限个值可以不满足不等式.
4. 常函数可以写作 $O(1)$.

大 Ω 符号

定义： 设 f 和 g 是定义域为自然数集 \mathbb{N} 上的函数. 若存在正数 c 和 n_0 , 使得对一切 $n \geq n_0$ 有

$$\underline{0 \leq cg(n) \leq f(n)}$$

成立, 则称 $f(n)$ 的渐近的下界是 $g(n)$, 记作

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

例子

设 $f(n) = n^2 + n$, 则

$f(n) = \Omega(n^2)$, 取 $c = 1, n_0 = 1$ 即可

$f(n) = \Omega(100n)$, 取 $c = 1/100, n_0 = 1$ 即可

1. $f(n) = \Omega(g(n))$, $f(n)$ 的阶不低于 $g(n)$ 的阶.
2. 可能存在多个正数 c , 指出一个即可.
3. 对前面有限个 n 值可以不满足上述不等式.

小o符号

定义 设 f 和 g 是定义域为自然数集 \mathbf{N} 上的函数. 若对于任意正数 c 都存在 n_0 , 使得对一切 $n \geq n_0$ 有

$$\underline{0 \leq f(n) < c g(n)}$$

成立, 则记作

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

例子

例子: $f(n)=n^2+n$, 则

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

$c \geq 1$ 显然成立, 因为 $n^2+n < cn^3$ ($n_0=2$)

任给 $1 > c > 0$, 取 $n_0 > \lceil 2/c \rceil$ 即可. 因为

$$cn \geq \underline{cn_0} > 2 \quad (\text{当 } n \geq n_0)$$

$$n^2+n < 2n^2 < cn^3$$

1. $f(n) = o(g(n))$, $f(n)$ 的阶低于 $g(n)$ 的阶
2. 对不同正数 c , n_0 不一样. c 越小 n_0 越大.
3. 对前面有限个 n 值可以不满足不等式.

小 ω 符号

定义： 设 f 和 g 是定义域为自然数集 \mathbf{N} 上的函数. 若对于 任意正数 c 都存在 n_0 , 使得对一切 $n \geq n_0$ 有

$$\underline{0 \leq cg(n) < f(n)}$$

成立, 则记作

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

例子

设 $f(n) = n^2 + n$, 则

$$f(n) = \omega(n),$$

不能写 $f(n) = \omega(n^2)$, 因为取 $c = 2$, 不存在 n_0 使得对一切 $n \geq n_0$ 有下式成立

$$c n^2 = 2n^2 < n^2 + n \quad \times$$

1. $f(n) = \omega(g(n))$, $f(n)$ 的阶高于 $g(n)$ 的阶.
2. 对不同的正数 c , n_0 不等, c 越大 n_0 越大.
3. 对前面有限个 n 值可以不满足不等式.

Θ 符号

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

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

例子: $f(n) = n^2 + n$, $g(n) = 100n^2$, 那么有

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

1. $f(n)$ 的阶与 $g(n)$ 的阶相等.
2. 对前面有限个 n 值可以不满足条件.

例子：素数测试

算法 **PrimalityTest**(n)

输入： n , 大于2的奇整数

输出： true 或者 false

1. $s \leftarrow \lfloor n^{1/2} \rfloor$

2. for $j \leftarrow 2$ to s

3. if j 整除 n

4. then return false

5. return true

问题：

若 $n^{1/2}$ 可在 $O(1)$

计算, 基本运算是整除, 以下表示是否正确？

$W(n) = O(n^{1/2})$ ✓

$W(n) = \Theta(n^{1/2})$ ✗



为什么？

小结

- 五种表示函数的阶的符号
 $O, \Omega, o, \omega, \Theta$
- 定义
- 如何用定义证明函数的阶？

有关函数渐 近的界的定理

定理1

定理 设 f 和 g 是定义域为自然数集合的函数.

(1) 如果 $\lim_{n \rightarrow \infty} f(n)/g(n)$ 存在, 并且等于某个

常数 $c > 0$, 那么 $f(n) = \Theta(g(n))$.

(2) 如果 $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$, 那么

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

(3) 如果 $\lim_{n \rightarrow \infty} f(n)/g(n) = +\infty$, 那么

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

证明用到 Θ, o, ω 定义

证明定理1(1)

根据极限定义, 对于给定正数 ε 存在某个 n_0 , 只要 $n \geq n_0$, 就有

$$|f(n)/g(n) - c| < \varepsilon$$

$$c - \varepsilon < f(n)/g(n) < c + \varepsilon$$

取 $\varepsilon = c/2$,

$$c/2 < f(n)/g(n) < 3c/2 < 2c$$

对所有 $n \geq n_0$, $f(n) \leq 2cg(n)$, 于是 $f(n) = O(g(n))$;

对所有 $n \geq n_0$, $f(n) \geq (c/2)g(n)$, 于是 $f(n) = \Omega(g(n))$.

从而 $f(n) = \Theta(g(n))$.

例：估计函数的阶

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

证 因为

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

根据定理1, 有 $f(n) = \Theta(n^2)$.

一些重要结果

可证明：多项式函数的阶低于指数函数的阶

$$n^d = o(r^n), \quad r > 1, \quad d > 0$$

证 不妨设 d 为正整数,

$$\lim_{n \rightarrow \infty} \frac{n^d}{r^n} = \lim_{n \rightarrow \infty} \frac{dn^{d-1}}{r^n \ln r} = \lim_{n \rightarrow \infty} \frac{d(d-1)n^{d-2}}{r^n (\ln r)^2}$$

$$= \dots = \lim_{n \rightarrow \infty} \frac{d!}{r^n (\ln r)^d} = 0$$

分子分母
求导数

一些重要结果(续)

可证明：对数函数的阶低于幂函数的阶

$$\ln n = o(n^d), \quad d > 0$$

证

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\ln n}{n^d} &= \lim_{n \rightarrow \infty} \frac{\frac{1}{n}}{dn^{d-1}} \\ &= \lim_{n \rightarrow \infty} \frac{1}{dn^d} = 0 \end{aligned}$$

定理 2

定理 设函数 f, g, h 的定义域为自然数集合,

- (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 $f=\Theta(h)$

函数的阶之间的关系具有传递性

例子

按照阶从高到低排序以下函数：

$$f(n)=(n^2+n)/2, \quad g(n)=10n$$

$$h(n)=1.5^n, \quad t(n)=n^{1/2}$$

$$h(n) = \omega(f(n)),$$

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

$$g(n) = \omega(t(n)),$$

排序 $h(n), f(n), g(n), t(n)$

定理3

定理 假设函数 f 和 g 的定义域为自然数集,
若对某个其它函数 h , 有 $f=O(h)$ 和 $g=O(h)$,
那么 $f + g = O(h)$.

该性质可以推广到有限个函数.

算法由有限步骤构成. 若每一步的时间复杂度函数的上界都是 $h(n)$, 那么该算法的时间复杂度函数可以写作 $O(h(n))$.

小结

- 估计函数的阶的方法：
计算极限
阶具有传递性
- 对数函数的阶低于幂函数的阶，多项式函数的阶低于指数函数的阶.
- 算法的时间复杂度是各步操作时间之和，在常数步的情况下取最高阶的函数即可.

几类重要的函数

基本函数类

阶的高低

至少指数级: $2^n, 3^n, n!, \dots$

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

对数多项式级: $\log n, \log^2 n, \log \log n, \dots$

对数函数

符号:

$$\log n = \log_2 n$$

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

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

性质:

$$(1) \log_2 n = \Theta(\log_l n)$$

$$(2) \log_b n = o(n^\alpha) \quad \alpha > 0$$

$$(3) a^{\log_b n} = n^{\log_b a}$$

有关性质 (1) 的证明

$$\log_k n = \frac{\log_l n}{\log_l k} \quad \log_l k \text{ 为常数}$$

$$\lim_{n \rightarrow \infty} \frac{\log_k n}{\log_l n} = \lim_{n \rightarrow \infty} \frac{\log_l n}{\log_l k \cdot \log_l n} = \frac{1}{\log_l k}$$

根据定理, $\log_k n = \Theta(\log_l n)$

有关性质 (2) (3) 的说明

$$\log_b n = \Theta(\ln n)$$

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

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



$$\log_b n \log_b a = \log_b a \log_b n$$

指数函数与阶乘

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

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

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

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

应用：估计搜索空间大小

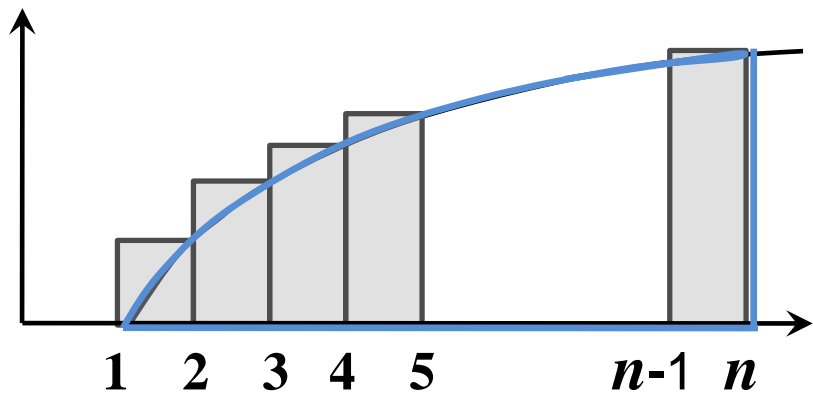
$$C_{m+n-1}^m = \frac{(m+n-1)!}{m!(n-1)!}$$

m 元钱、投资
 n 个项目的分配方
案数（视频001）

$$= \frac{\sqrt{2\pi(m+n-1)}(m+n-1)^{m+n-1}(1+\Theta(\frac{1}{m+n-1}))}{\sqrt{2\pi m}m^m(1+\Theta(\frac{1}{m}))\sqrt{2\pi(n-1)}(n-1)^{n-1}(1+\Theta(\frac{1}{n-1}))}$$

$$= \Theta((1+\varepsilon)^{m+n-1})$$

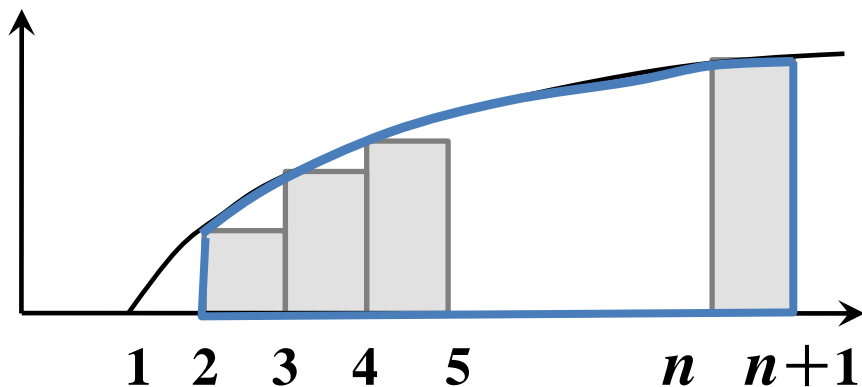
$\log(n!) = \Omega(n \log n)$ 的证明



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

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

$\log(n!) = O(n \log n)$ 的证明



$$\log(n!) = \sum_{k=1}^n \log k \leq \int_2^{n+1} \log x dx = O(n \log n)$$

取整函数

取整函数的定义

$\lfloor x \rfloor$: 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

实例

$$\lfloor 2.6 \rfloor = 2$$

$$\lceil 2.6 \rceil = 3$$

$$\lfloor 2 \rfloor = \lceil 2 \rceil = 2$$

应用: 二分检索

输入数组长度: n

中位数的位置: $\lfloor n/2 \rfloor$

与中位数比较后子问

题大小: $\lfloor n/2 \rfloor$

取整函数的性质

$$(1) \quad \underline{x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1}$$

$$(2) \quad \underline{\lfloor x+n \rfloor = \lfloor x \rfloor + n, \lceil x+n \rceil = \lceil x \rceil + n, n \text{ 为整数}}$$

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

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

证明(1)

(1) 如果 x 是整数 n , 根据定义 $\lfloor x \rfloor = \lceil x \rceil = n$,

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

如果 $n < x < n+1$, n 为整数, 那么

$$\lfloor x \rfloor = n, \lceil x \rceil = n+1,$$

从而有

$$x-1 < n = \lfloor x \rfloor, \quad n < x < n+1 = \lceil x \rceil$$

$$\Rightarrow x-1 < n = \lfloor x \rfloor < x < \lceil x \rceil = n+1 < x+1$$

例：按照阶排序

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

$(3/2)^n, \sqrt{\log n}, (\log n)^{\log n}, 2^{2^n},$

$n^{\log \log n}, n^3, \log \log n, n \log n, n,$

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

例：按照阶排序

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

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

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

$$n^{1/\log n} = 1$$

小结

- 几类常用函数的阶的性质
 - 对数函数
 - 指数函数
 - 阶乘函数
 - 取整函数
- 如何利用上述性质估计函数的阶？

本周教学内容

算法的数学基础

序列求和
方法：

求和公式
估计和式的
上界

算法迭代与
序列求和

差消
法化
简

递归
树模
型

主定理的
应用

迭代法：
基本迭代
换元迭代

主定理及
其证明

递推方程与算法分析的关系

序列求和的方法

数列求和公式

等差、等比数列与调和级数

$$\sum_{k=1}^n a_k = \frac{n(a_1 + a_n)}{2}$$

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

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

求和的例子

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

拆项

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

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

变限

拆项

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

二分检索算法

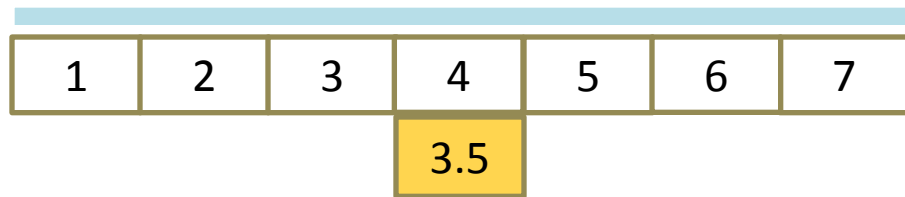
算法 BinarySearch (T, l, r, x)

输入：数组 T ，下标从 l 到 r ；数 x

输出： j

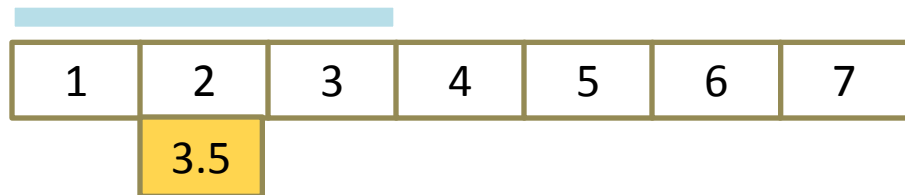
1. $l \leftarrow 1; r \leftarrow n$
2. while $l \leq r$ do
3. $m \leftarrow \lfloor (l+r)/2 \rfloor$
4. if $T[m]=x$ then return m // x 中位元素
5. else if $T[m] > x$ then $r \leftarrow m-1$
6. else $l \leftarrow m+1$
7. return 0

二分检索运行实例



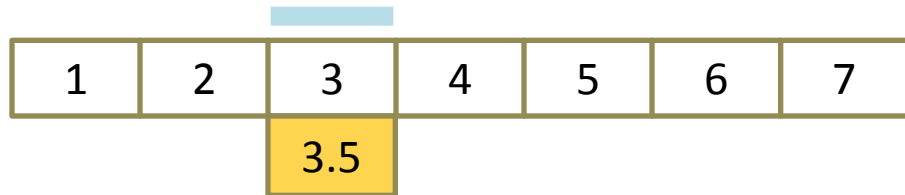
1	2	3	4	5	6	7
			3.5			

第1次
 $3.5 < 4$



1	2	3	4	5	6	7
	3.5					

第2次
 $3.5 > 2$



1	2	3	4	5	6	7
		3.5				

第3次
 $3.5 > 3$

$2n+1$ 个输入

假设 $n = 2^k - 1$, 输入有 $2n + 1$ 种:

$$x = T[1]$$

$$x = T[2]$$

...

$$x = T[n-1]$$

$$x = T[n]$$

x 在 T 中

$$x < T[1]$$

$$T[1] < x < T[2]$$

...

$$T[n-1] < x < T[n]$$

$$T[n] < x$$

x 不在 T 中

比较 t 次的输入个数



比较1次:1个



比较2次:2个



比较3次:4个

对 $t = 1, 2, \dots, k-1$, 比较 t 次: 2^{t-1} 个

比较 k 次的输入有 $2^{k-1} + n + 1$ 个

总次数: 对每个输入乘以次数并求和


二分检索平均时间复杂度

假设 $n = 2^k - 1$, 各种输入概率相等

$$\begin{aligned} A(n) &= \frac{1}{2n+1} \left[\sum_{t=1}^{k-1} t 2^{t-1} + k(2^{k-1} + n + 1) \right] \\ &= \frac{1}{2n+1} \left[\sum_{t=1}^k t 2^{t-1} + k(n+1) \right] \\ &= \frac{1}{2n+1} \left[(k-1)2^k + 1 + k(n+1) \right] \\ &= \frac{k2^k - 2^k + 1 + k2^k}{2^{k+1} - 1} \approx k - \frac{1}{2} = \lfloor \log n \rfloor + \frac{1}{2} \end{aligned}$$

估计和式上界的放大法

放大法:

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

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

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



$a_1 \leq a_0 r, a_2 \leq a_1 r \leq a_0 r^2, \dots$

放大法的例子

估计 $\sum_{k=1}^n \frac{k}{3^k}$ 的上界.

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

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

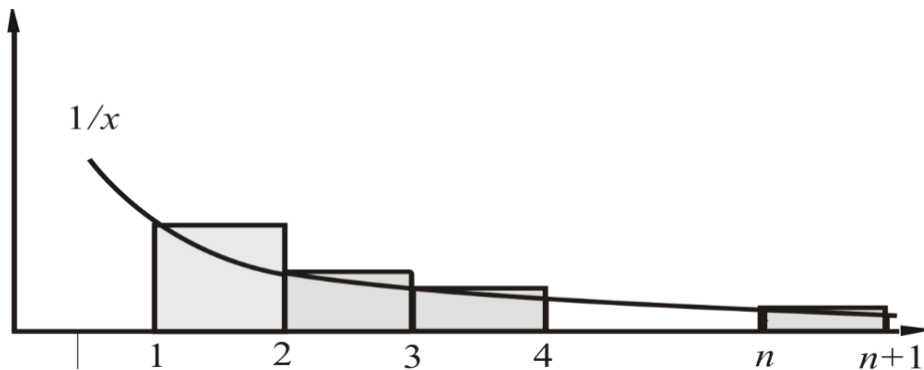
得

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

估计和式渐近的界

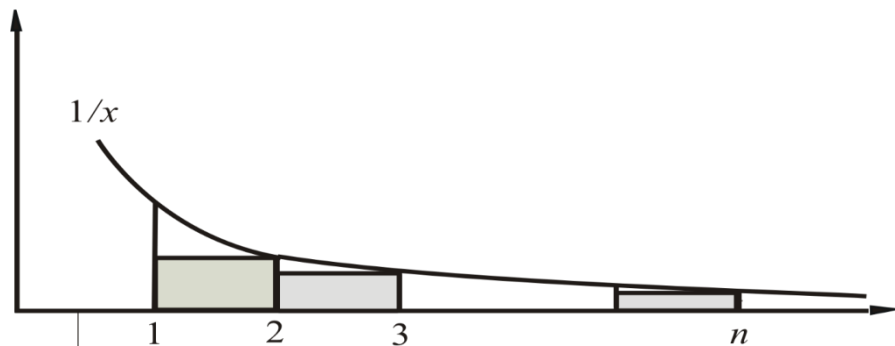
估计 $\sum_{k=1}^n \frac{1}{k}$ 的渐近的界.

$$\sum_{k=1}^n \frac{1}{k} \geq \int_1^{n+1} \frac{dx}{x} = \ln(n+1)$$



估计和式渐近的界

$$\sum_{k=1}^n \frac{1}{k} = \frac{1}{1} + \sum_{k=2}^n \frac{1}{k} \leq 1 + \int_1^n \frac{dx}{x} \\ = \ln n + 1$$



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

小结

- 序列求和基本公式：
等差数列
等比数列
调和级数
- 估计序列和：
放大法求上界
用积分做和式的渐近的界
- 应用：计数循环过程的基本运算次数

递推方程与 算法分析

递推方程

设序列 $a_0, a_1, \dots, a_n, \dots$, 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

递推方程的求解:

给定关于序列 $\{a_n\}$ 的递推方程和若干初值, 计算 a_n

递推方程的例子

Fibonacci数

1, 1, 2, 3, 5, 8, 13, 21,
34, 55, ...



数学家Fibonacci
意大利1170-1240

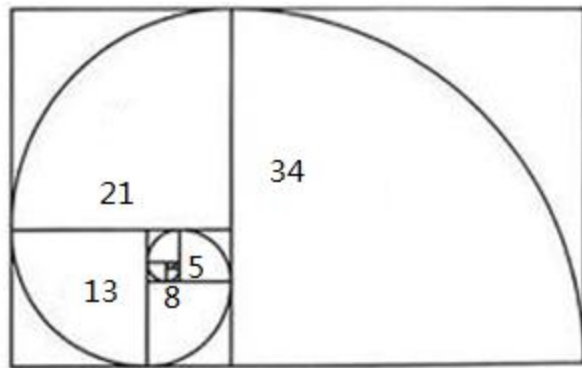
递推方程: $f_n = f_{n-1} + f_{n-2}$

初值: $f_0 = 1, f_1 = 1$

解:

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^{n+1} - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^{n+1}$$

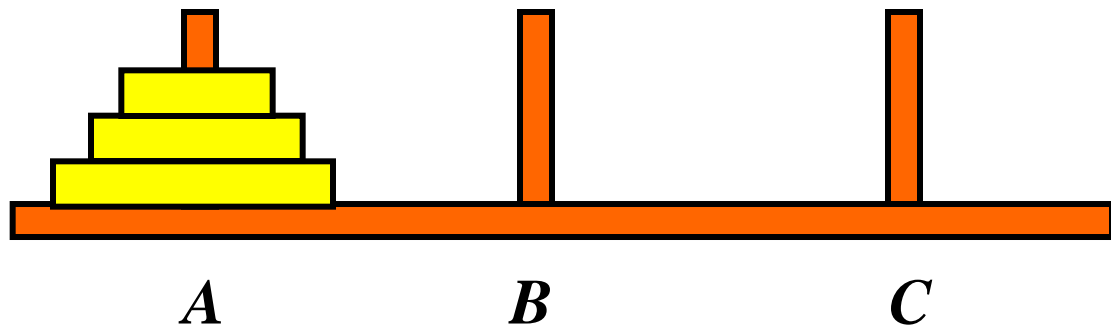
Fibonacci数的存在



55



Hanoi塔问题



n 个盘子从大到小顺序放在A 柱上，
要把它们从 A 移到 C，每次移动 1个
盘子，移动时不允许大盘压在小盘上。
设计一种移动方法。

递归算法

算法 Hanoi (A, C, n) // n 个盘子 A 到 C

1. if $n=1$ then move (A, C) // 1个盘子 A 到 C
2. else Hanoi ($A, B, n-1$)
3. move (A, C)
4. Hanoi ($B, C, n-1$)

设 n 个盘子的移动次数为 $T(n)$

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

$$T(1) = 1,$$

分析算法

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

解

$$T(n) = 2^n - 1$$

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



有没有更好的算法？

没有！这是一个难解的问题，不存在多项式时间的算法！

插入排序

算法 Insert Sort (A, n)

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

最坏情况下时间复杂度

插入排序：

设基本运算是元素比较，对规模为 n 的输入最坏情况下的时间复杂度 $W(n)$

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

$$W(1) = 0$$

解为

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

小结

- 递推方程的定义及初值
- 递推方程与算法时间复杂度的关系

Hanoi塔的递归算法

插入排序的迭代算法

迭代法求解 递推方程

迭代法

- 不断用递推方程的右部替换左部
- 每次替换，随着 n 的降低在和式中多出一项
- 直到出现初值停止迭代
- 将初值代入并对和式求和
- 可用数学归纳法验证解的正确性

Hanoi 塔算法

$$\begin{aligned}T(n) &= 2 T(n-1) + 1 \\T(1) &= 1\end{aligned}$$

$$\begin{aligned}T(n) &= 2 T(\underline{n-1}) + 1 \\&= 2 [\underline{2T(n-2)} + 1] + 1 \\&= 2^2 T(n-2) + 2 + 1 \\&= \dots \\&= 2^{n-1}T(1) + 2^{n-2} + 2^{n-3} + \dots + 2 + 1 \\&= 2^{n-1} + 2^{n-1} - 1 \\&= 2^n - 1\end{aligned}$$

代入初值
求和

插入排序算法

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

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

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

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

$$= \dots$$

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

$$= 1 + 2 + \dots + (n-2) + (n-1)$$

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

换元迭代

- 将对 n 的递推式换成对其他变元 k 的递推式
- 对 k 直接迭代
- 将解 (关于 k 的函数) 转换成关于 n 的函数

二分归并排序

MergeSort (A, p, r)

输入：数组 $A[p..r]$

输出：按递增顺序排序的数组 A

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)

换元

假设 $n=2^k$, 递推方程如下:

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

$$W(1)=0$$

换元:

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

$$W(0) = 0$$

迭代求解

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

$$\begin{aligned} W(n) &= 2W(2^{k-1}) + 2^k - 1 \\ &= 2[2W(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1 \end{aligned}$$

换元，
对 k 迭代

$$\begin{aligned} &= 2^2 W(2^{k-2}) + 2^k - 2 + 2^k - 1 \\ &= 2^2 [2W(2^{k-3}) + 2^{k-2} - 1] + 2^k - 2 + 2^k - 1 \end{aligned}$$

$= \dots$

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

$$= k 2^k - 2^k + 1$$

$$= n \log n - n + 1$$

解的正确性-归纳验证

证明:下述递推方程的解是 $W(n)=n(n-1)/2$

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

$$W(1)=0$$

方法: 数学归纳法

证 $n=1$, $W(1)=1\times(1-1)/2 = 0$

假设对于 n , 解满足方程, 则

$$\begin{aligned} & W(n+1) \\ &= \underline{W(n)+n} = \underline{n(n-1)/2} + n \\ &= n[(n-1)/2+1] = n(n+1)/2 \end{aligned}$$

小结

迭代法求解递推方程

- 直接迭代，代入初值，然后求和
- 对递推方程和初值进行换元，然后求和，求和后进行相反换元，得到原始递推方程的解
- 验证方法——数学归纳法

差消法化简 高阶递推方程

快速排序

- 假设 $A[p..r]$ 的元素彼此不等

以首元素 $A[p]$ 对数组 $A[p..r]$ 划分,使得:

小于 x 的元素放在 $A[p..q-1]$

大于 x 的元素放在 $A[q+1..r]$

- 递归对 $A[p..q-1]$ 和 $A[q+1..r]$ 排序

工作量: 子问题工作量+划分工作量

输入情况

- 有 n 种可能的输入

x 排好序位置	子问题 1 规模	子问题 2 规模
1	0	$n-1$
2	1	$n-2$
3	2	$n-3$
...
$n-1$	$n-2$	1
n	$n-1$	0

对每个输入，划分的比较次数都是 $n-1$

工作量总和

$$T(0) + T(n-1) + n-1$$

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

$$T(2) + T(n-3) + n-1$$

...

$$+ T(n-1) + T(0) + n-1$$

$$2[T(1)+...+T(n-1)]+n(n-1)$$

快速排序平均工作量

假设首元素排好序在每个位置是等概率的

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

$$T(1) = 0$$

全部历史递推方程

对于高阶方程应该先化简，然后迭代

差消化简

利用两个方程相减，将右边的项尽可能消去，以达到降阶的目的

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

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

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

差消化简

$$\begin{aligned} & nT(n) - \underline{(n-1)T(n-1)} \\ = & \underline{2T(n-1)} + cn^2 - c(n-1)^2 \end{aligned}$$



$$nT(n) = (n+1)T(n-1) + c_1n$$



$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c_1}{n+1}$$

迭代求解

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \boxed{\frac{c_1}{n+1}} = \dots$$

$$= c_1 \left[\frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right] + \frac{T(1)}{2}$$

$$= c_1 \left[\frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} \right]$$

代入
初值

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

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

小结

- 对于高阶递推方程先用差消法化简为一阶方程
- 迭代求解

递归树

递归树的概念

- 递归树是迭代计算的模型.
- 递归树的生成过程与迭代过程一致.
- 递归树上所有项恰好是迭代之后产生和式中的项.
- 对递归树上的项求和就是迭代后方程的解.

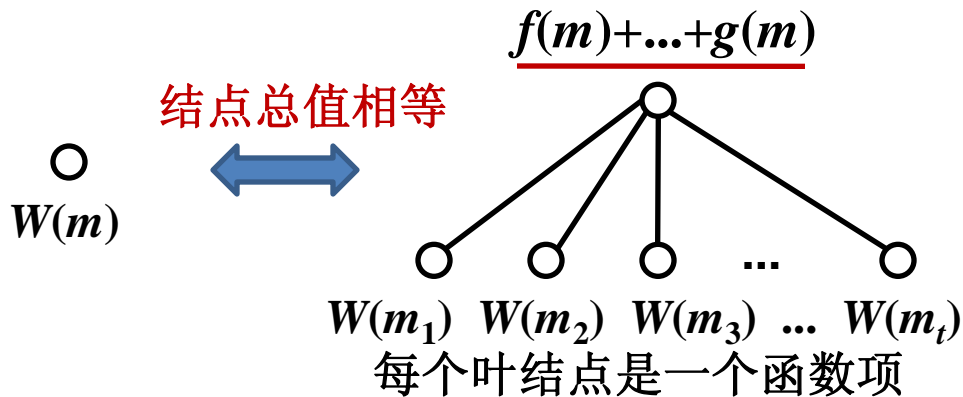
迭代在递归树中的表示

如果递归树上某结点标记为 $W(m)$

$$W(m) = W(m_1) + \dots + W(m_t)$$

$$+ \underline{f(m) + \dots + g(m)}, \quad m_1, \dots, m_t < m$$

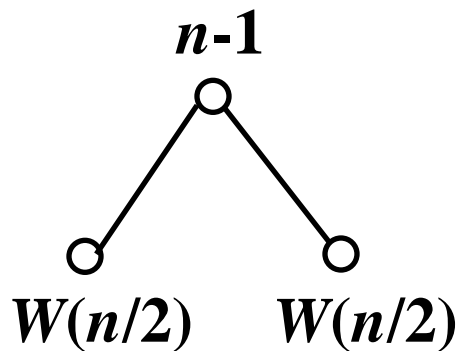
其中 $W(m_1), \dots, W(m_t)$ 称为函数项。



二层子树的例子

二分归并排序

$$W(n) = 2W(n/2) + \underline{n-1}$$

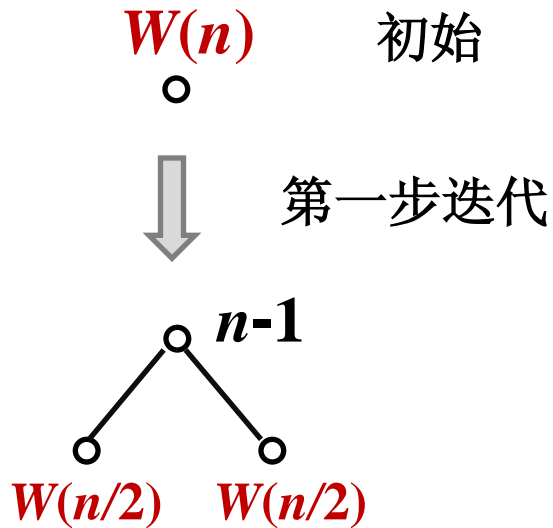


递归树的生成规则

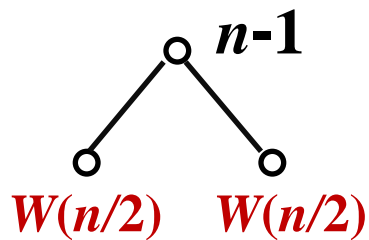
- 初始，递归树只有根结点，其值为 $W(n)$
- 不断继续下述过程：
将函数项叶结点的迭代式 $W(m)$ 表示成二层子树
用该子树替换该叶结点
- 继续递归树的生成，直到树中无函数项(只有初值)为止.

递归树生成实例

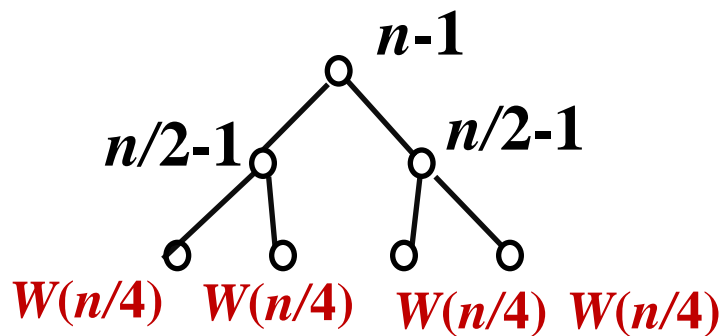
$$W(n) = 2W(n/2) + n-1$$



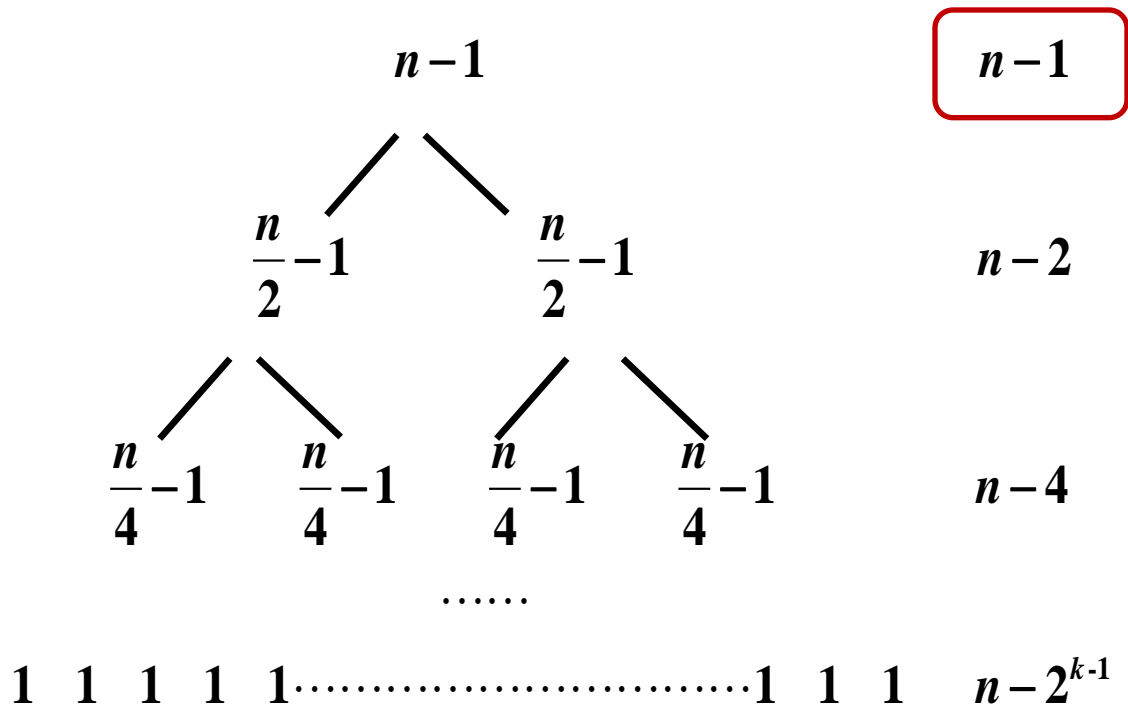
递归树生成实例



第二步迭代



递归树



$$n-1$$

$$n-2$$

$$n-4$$

对递归树上的量求和

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

$$W(1) = 0$$

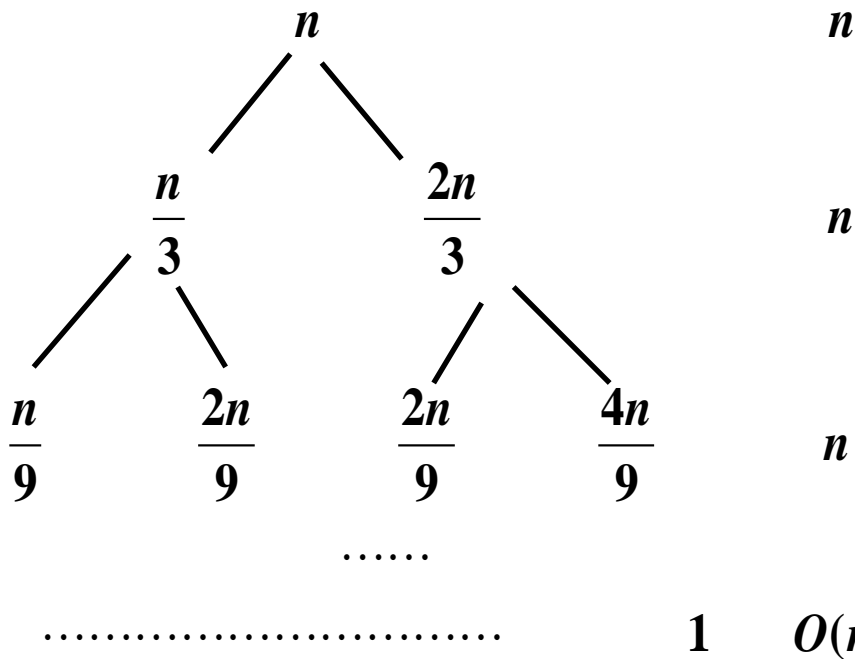
$$W(n) = n - 1 + n - 2 + \dots + n - 2^{k-1}$$

$$= kn - (2^k - 1)$$

$$= n \log n - n + 1$$

递归树应用实例

求解方程: $T(n) = T(n/3) + T(2n/3) + n$



求和

方程: $T(n)=T(n/3)+T(2n/3)+n$

递归树层数 k , 每层 $O(n)$

$$n(2/3)^k = 1$$

$$\Rightarrow (3/2)^k = n$$

$$\Rightarrow k = O(\log_{3/2} n)$$

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

小结

- 递归树是迭代的图形表述
- 递归树的生成规则
- 如何利用递归树求解递推方程?

主定理及其证明

主定理的应用背景

求解递推方程

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

a : 归约后的子问题个数

n/b : 归约后子问题的规模

$f(n)$: 归约过程及组合子问题的解的工作量

二分检索: $T(n) = T(n/2) + 1$

二分归并排序: $T(n) = 2T(n/2) + n - 1$

主定理

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

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

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

存在 ε

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

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

存在 ε

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

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

存在 c
和 n_0

迭代

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

设 $n=b^k$

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

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

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

$= \dots$

迭代结果

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

$$= a^k T\left(\frac{n}{b^k}\right) + a^{k-1} f\left(\frac{n}{b^{k-1}}\right) + \dots + a f\left(\frac{n}{b}\right) + f(n)$$

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

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

- 第一项为所有最小子问题的计算工作量
- 第二项为迭代过程归约到子问题及综合解的工作量

Case1

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

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

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

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

Case1(续)

$$\frac{1}{(b^{\log_b a - \varepsilon})^j} = \frac{b^{\varepsilon j}}{(b^{\log_b a})^j} = \frac{b^{\varepsilon j}}{a^j}$$

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

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

$$= c_1 n^{\log_b a} + O(n^{\log_b a - \varepsilon} \frac{b^{\varepsilon \log_b n} - 1}{b^{\varepsilon} - 1})$$

$$= c_1 n^{\log_b a} + O(n^{\log_b a - \varepsilon} \underline{n^{\varepsilon}}) = \Theta(n^{\log_b a})$$

Case2

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

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

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

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

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

Case3

$$f(n) = \Omega(n^{\log_b a + \varepsilon}) \quad (1)$$

$$af(n/b) \leq cf(n) \quad (2)$$

$$\begin{aligned} T(n) &= c_1 n^{\log_b a} + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right) \\ &\leq c_1 n^{\log_b a} + \sum_{j=0}^{\log_b n - 1} c^j f(n) \end{aligned}$$

$$a^j f\left(\frac{n}{b^j}\right) \leq a^{j-1} cf\left(\frac{n}{b^{j-1}}\right)$$

$$\leq ca^{j-1} f\left(\frac{n}{b^{j-1}}\right) \leq \dots \leq c^j f(n)$$

Case3 (续)

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

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

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

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

小结

- 主定理的应用背景
- 主定理的内容
- 主定理的证明

主定理的应用

求解递推方程：例1

例1 求解递推方程

$$T(n) = 9T(n/3) + n$$

解 上述递推方程中的

$$a = 9, \quad b = 3, \quad f(n) = n$$

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

相当于主定理的**case1**，其中 $\varepsilon=1$ 。

根据定理得到 $T(n) = \Theta(n^2)$

求解递推方程：例2

例2 求解递推方程

$$T(n) = T(2n/3) + 1$$

解 上述递推方程中的

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

$$n^{\log_{3/2} 1} = n^0 = 1$$

相当于主定理的**Case2** .

根据定理得到 $T(n) = \Theta(\log n)$

求解递推方程：例3

例3 求解递推方程

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

解 上述递推方程中的

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

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

取 $\varepsilon = 0.2$ 即可.

条件验证

要使 $a f(n/b) \leq c f(n)$ 成立,

代入 $f(n) = n \log n$, 得到

$$\underline{3 (n/4) \log (n/4) \leq c n \log n}$$

只要 $\underline{c \geq 3/4}$, 上述不等式可以对所有充分大的 n 成立. 相当于主定理的 **Case3**.

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

递归算法分析

二分检索: $W(n)=W(n/2)+1, W(1)=1$

$$a=1, b=2, \underline{n^{\log_2 1}=1}, f(n)=1,$$

属于Case2,

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

二分归并排序:

$$W(n)=2W(n/2)+n-1, W(1)=0$$

$$a=2, b=2, \underline{n^{\log_2 2}=n}, f(n)=n-1$$

属于Case2,

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

不能使用主定理的例子

例4 求解 $T(n)=2T(n/2)+n\log n$

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

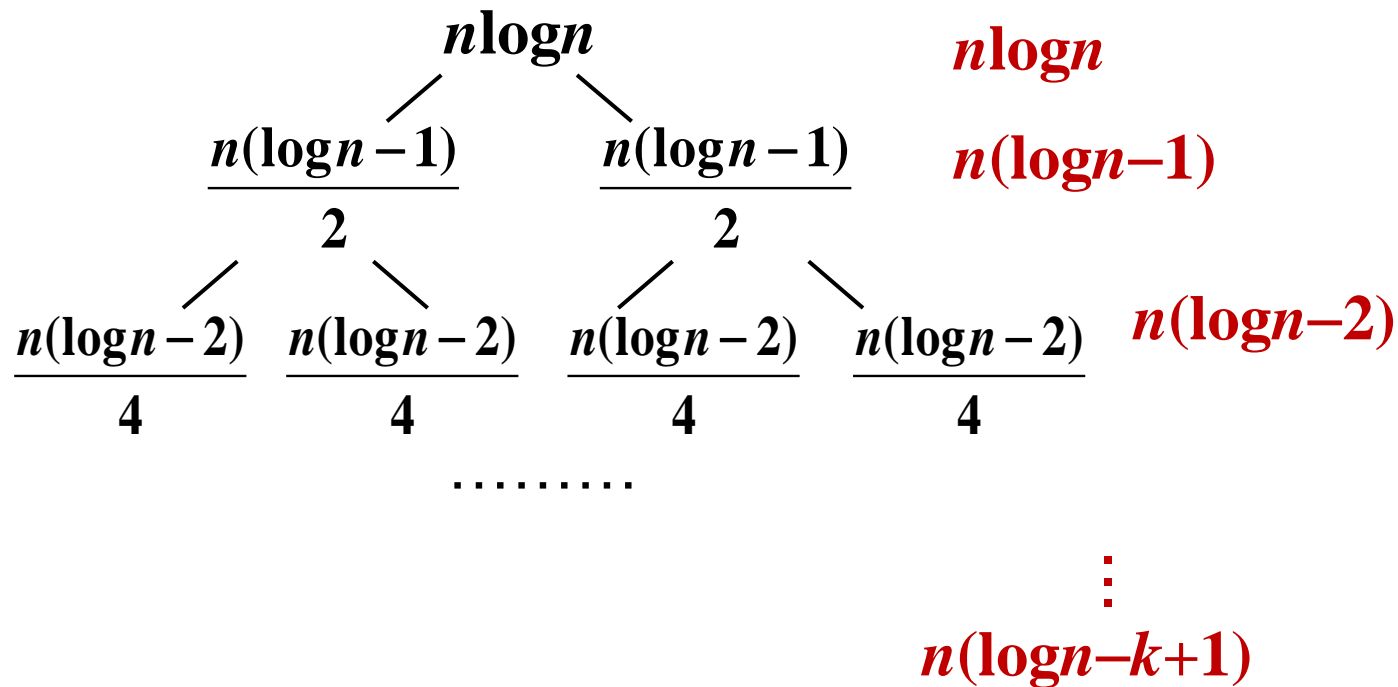
不存在 $\varepsilon > 0$ 使下式成立

$$n \log n = \Omega(n^{1+\varepsilon})$$

不存在 $c < 1$ 使 $af(n/b) \leq cf(n)$ 对所有充分大的 n 成立

$$2(n/2)\log(n/2) = \underline{n(\log n - 1)} \leq cn \log n$$

递归树求解



求和

$$T(n)$$

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

$$+ \dots + n(\log n - k + 1)$$

$$= (n \log n) \log n - n \underline{(1 + 2 + \dots + k - 1)}$$

$$= n \log^2 n - n \underline{k(k - 1) / 2} = O(n \log^2 n)$$

小结

- 使用主定理求解递推方程需要满足什么条件？
- 主定理怎样用于算法复杂度分析？