

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. $m \leftarrow \lfloor (l+r)/2 \rfloor$

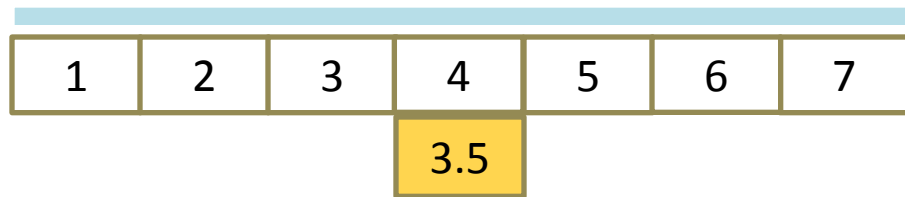
4. if $T[m]=x$ then return m // x 中位元素

5. else if $T[m] > x$ then $r \leftarrow m-1$

6. else $l \leftarrow m+1$

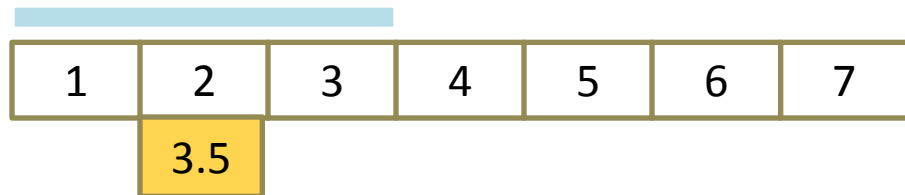
7. return 0

二分检索运行实例



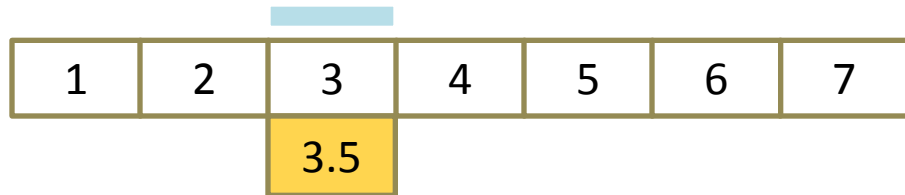
1	2	3	4	5	6	7
			3.5			

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

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

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

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

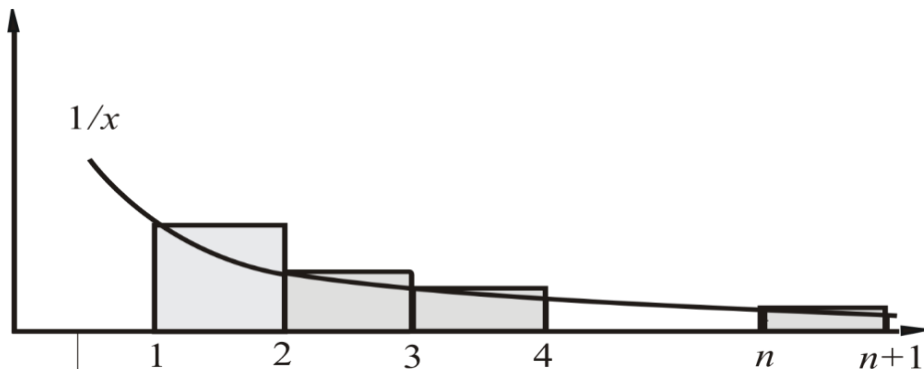
得

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

估计和式渐近的界

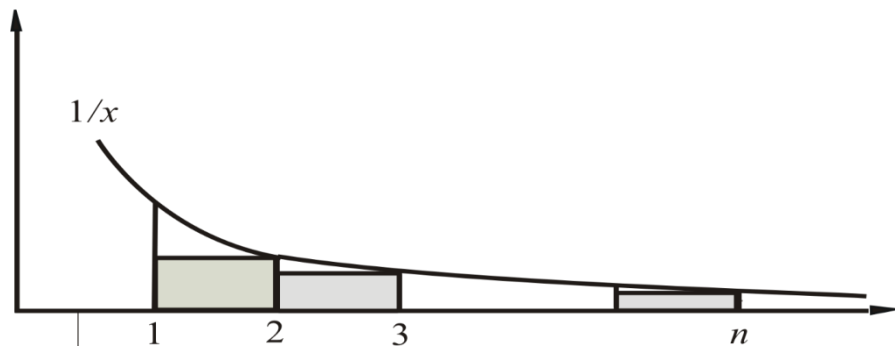
估计 $\sum_{k=1}^n \frac{1}{k}$ 的渐近的界.

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



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

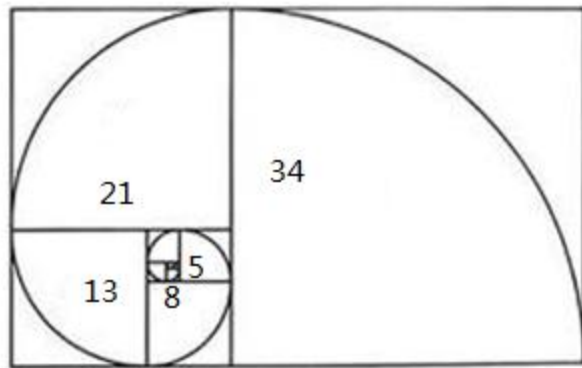
递推方程: $f_n = f_{n-1} + f_{n-2}$

初值: $f_0 = 1, f_1 = 1$

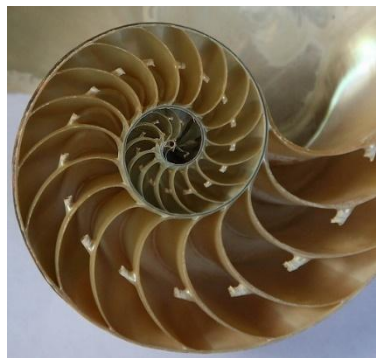
解:

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

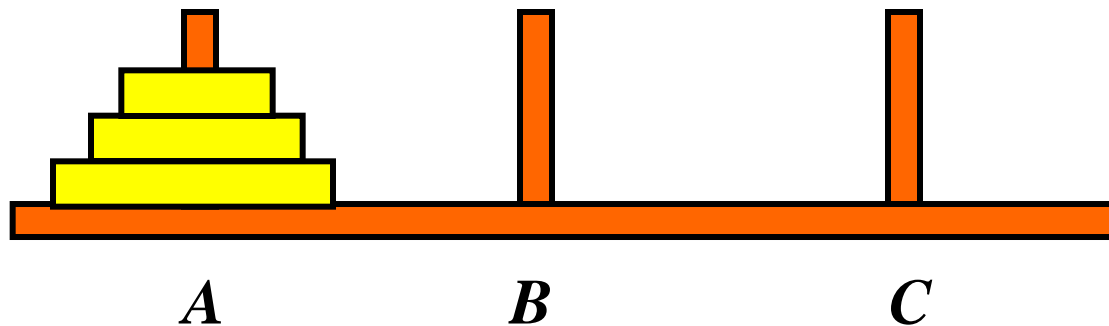
Fibonacci数的存在



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

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

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

$$= 2 [\underline{2T(n-2)} + 1] + 1$$

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

$$= \dots$$

$$= 2^{n-1}T(1) + 2^{n-2} + 2^{n-3} + \dots + 2 + 1$$

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

$$= 2^n - 1$$

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.

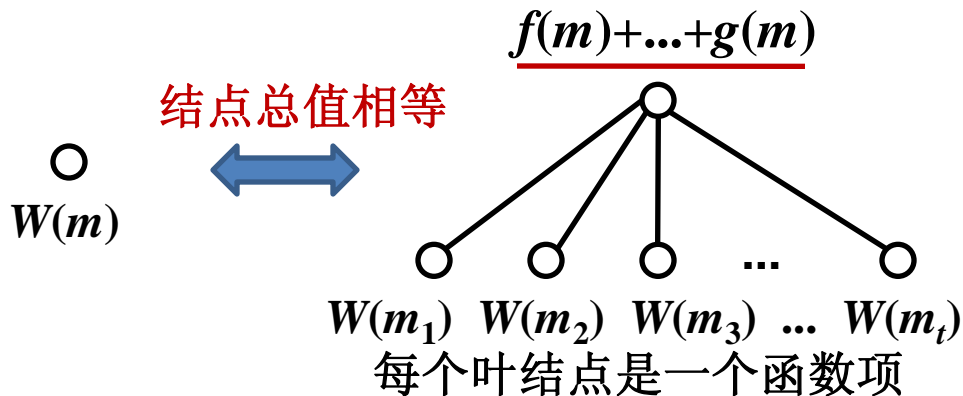
迭代在递归树中的表示

如果递归树上某结点标记为 $W(m)$

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

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

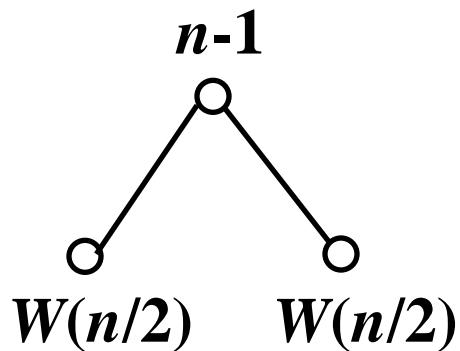
其中 $W(m_1), \dots, W(m_t)$ 称为函数项。



二层子树的例子

二分归并排序

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

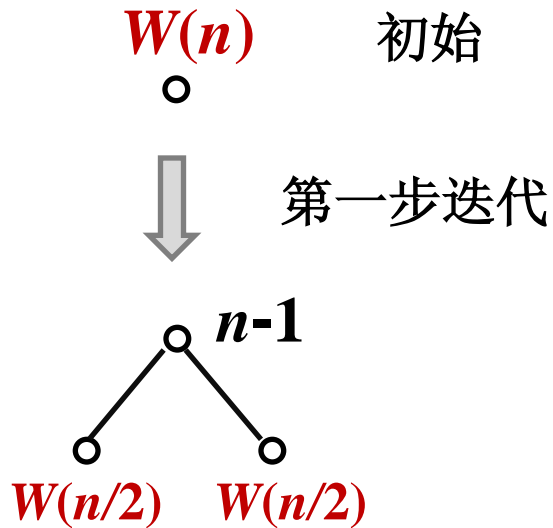


递归树的生成规则

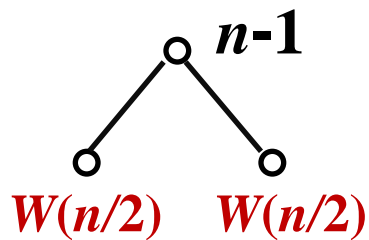
- 初始，递归树只有根结点，其值为 $W(n)$
- 不断继续下述过程：
将函数项叶结点的迭代式 $W(m)$ 表示成二层子树
用该子树替换该叶结点
- 继续递归树的生成，直到树中无函数项(只有初值)为止.

递归树生成实例

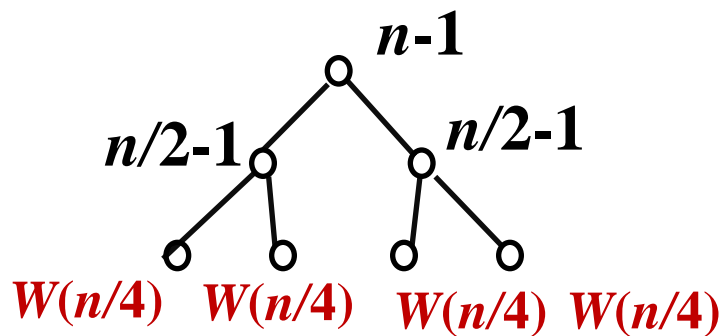
$$W(n) = 2W(n/2) + n-1$$



递归树生成实例



第二步迭代



递归树

$$n-1$$

$$n-2$$

$$n-4$$

.....

$$1 \quad 1 \quad 1 \quad 1 \quad 1 \dots\dots\dots 1 \quad 1 \quad 1 \quad n-2^{k-1}$$

对递归树上的量求和

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

$$W(1) = 0$$

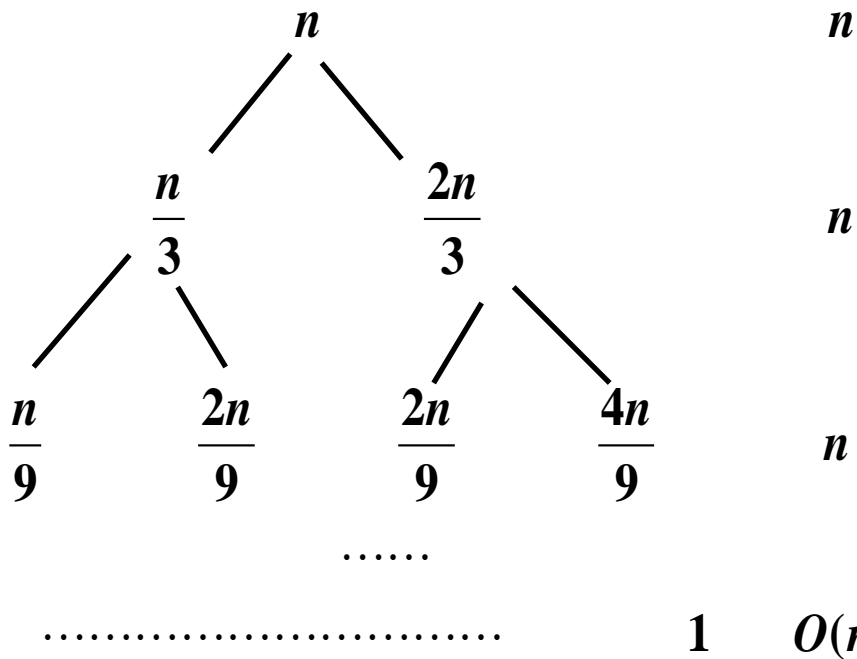
$$W(n) = n - 1 + n - 2 + \dots + n - 2^{k-1}$$

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

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

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)

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

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

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

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

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

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

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

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

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

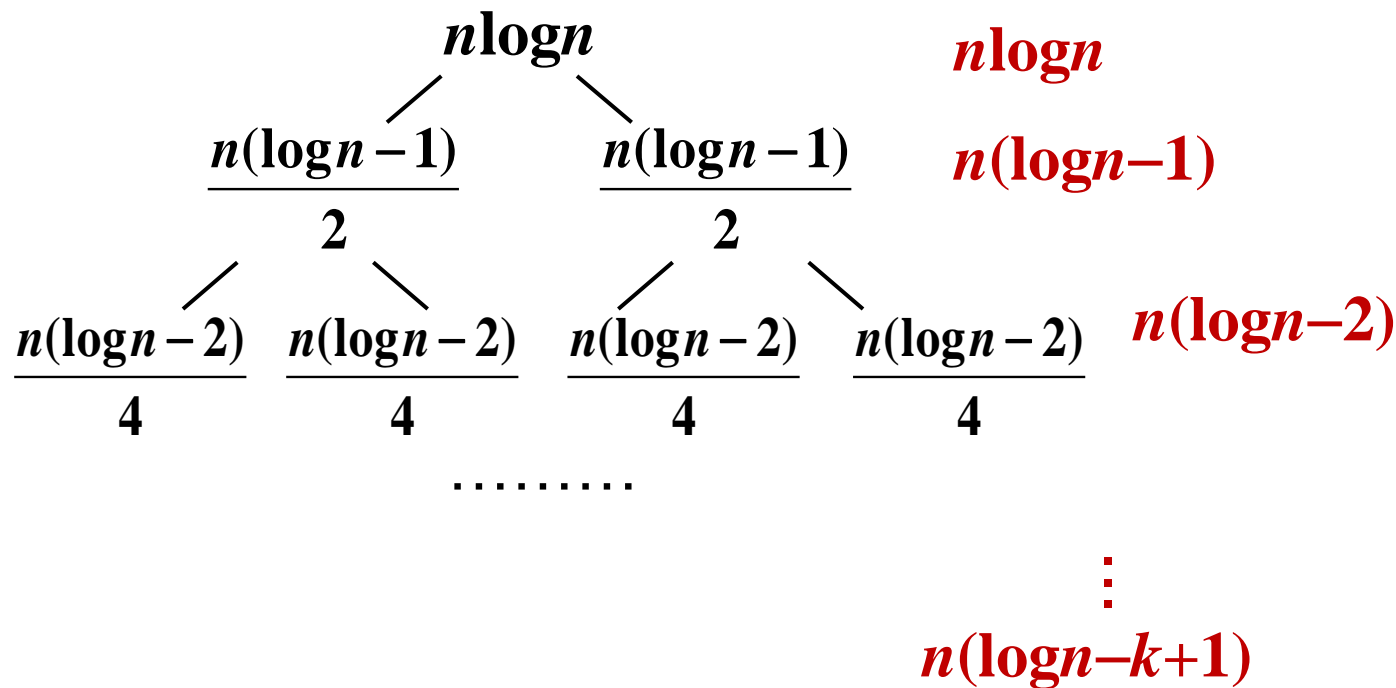
不存在 $\varepsilon > 0$ 使下式成立

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

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？