算法基础

Jun Wu

mail: wujun@yzu.edu.cn, mobile: 18952730176

March 5, 2018

- **W**hat are algorithms?
- **W**hy is the study of algorithms worthwhile?
- **W**hat is the role of algorithms relative to other technologies used in computers?

算法的概念

- **算法(Algorithm)**：一个由计算步骤构成的序列，可以将一组输入值转换成相应地输出值。或，
- 算法：一个由计算步骤构成的序列，可以用来解决一个明确定义的问题。
- 问题：输入及相应输出的描述。例如，
    - Input：A sequence of n numbers $< a_1, \cdots, a_n >$;
    - Output: A permutation (reordering) $< a'_1, \cdots, a'_n >$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.
    - 排序问题。
- 算法的特点：确定性，可行性，输入和输出及有穷性。
- 正确的算法：停机并给出符合问题的定义的输出。

研究算法的意义

- 算法涉及到解决问题的：正确性、可行性以及效率。
- 以效率为例：对1M数据进行排序
- 插入排序$c_1 n^2 = O(n^2)$，$c_1 = 2$;
- 归并排序$c_2 n \log n = O(n \log n)$，$c_2 = 50$.

10^3MIPS Insert sort $2n^2$	10MIPS Merge sort $50n \log n$
$2 \times (10^6)^2/10^9 \approx 2000$s	$50 \times 10^6 \log 10^6/10^7 \approx 100$s

算法与其它技术的关系

- 硬件设计
    - 布线算法
- 图形界面
    - 图形生成、处理等
- 面向对象
    - 编译器设计
- 局域网和广域网
    - 冲突避免、路由、交换

算法几乎无处不在。

Dijstra的告诫

编程序不是计算机科学，就像看望远镜不是天文学一样。

–*Dijstra*

课程概况

- 课程性质：专业选修课
- 先修课程：离散数学，数据结构，程序设计
- 学分：2.5
- 考试：闭卷
- 教材：计算机算法基础，沈孝均编著
- 参考书：
  1. Introduction to Algorithms, C.L.R.S.
  2. 算法设计，Jon Kleinberg, Eva Tardos

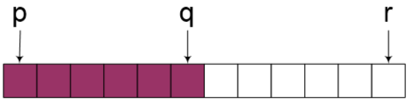
伪码

INSERTION-SORT(A)

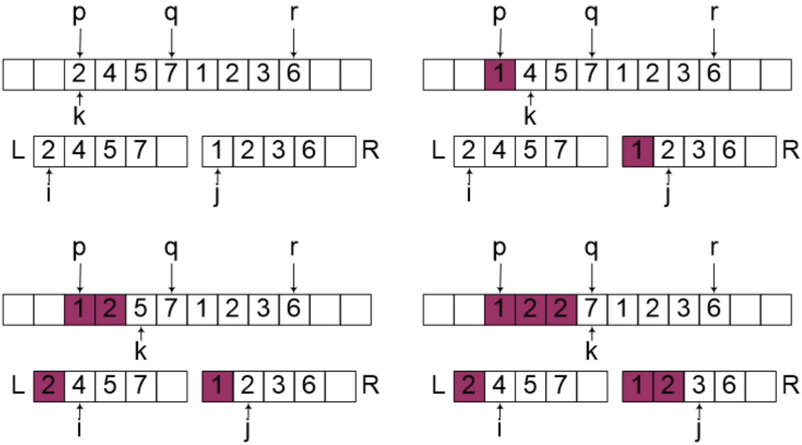
for $j \leftarrow 2$ **to** $length(A)$ **do**
    $key \leftarrow A[j]$
    $\triangleright$ Insert $A[j]$ into the sorted sequence $A[1, j-1]$.
    $i \leftarrow j - 1$
    while $i > 0 \wedge A[i] > key$ **do**
        $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    end while
    $A[i+1] \leftarrow key$
end for

- 采用类Pascal风格的语言
- 忽略了编程的细节，着重关注方法本身
- 在不影响理解的情况下，可以采用部分自然语言描述。

- 增量式
  - 逐步求解的过程，如插入排序
- 分治策略：以归并排序为例
  1. Divide：将n个待排序的元素划分成两个分别有$n/2$个元素的子序列；
  2. Conquer：对两个子序列递归地调用merge-sort进行排序；
  3. Combine：将两个有序的子序列合并成n个元素的有序序列；



归并过程示意



p q r
2 4 5 7 1 2 3 6
k
L 2 4 5 7
1 2 3 6 R
i j
p q r
1 4 5 7 1 2 3 6
k
L 2 4 5 7
1 2 3 6 R
i j
p q r
1 2 5 7 1 2 3 6
k
L 2 4 5 7
1 2 3 6 R
i j
p q r
1 2 2 7 1 2 3 6
k
L 2 4 5 7
1 2 3 6 R
i j

MERGE(A, p, q, r)

$n_1 \leftarrow q - p + 1$
$n_2 \leftarrow r - q$
create arrays $L[1 \cdots n_1 + 1]$ and $R[1 \cdots n_2 + 1]$
for $i \leftarrow 1$ **to** n_1 **do**
    $L[i] \leftarrow A[p + i - 1]$
end for
for $j \leftarrow 1$ **to** n_2 **do**
    $R[j] \leftarrow A[q + j]$
end for
$L[n_1 + 1] \leftarrow \infty$
$R[n_2 + 1] \leftarrow \infty$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow p$ **to** r **do**
    if $L[i] \leq R[j]$ **then**
        $A[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $A[k] \leftarrow R[j]$
        $j \leftarrow j + 1$
    end if
end for

MERGE-SORT(A, p, r)

```
if p < r then
    q ← ⌊(p+r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
end if
```

算法设计方法简介(con't)

- 动态规划
  - 一种强有力的技术
  - 从小的子问题的解逐步计算出大问题的解
  - 计算过程通过动态填表实现(规划)
- 贪心策略
  - 最常用的启发策略
  - 但常常“贪心”不能保证得到好的结果
  - 有些问题满足一些特定性质，可以保证“贪心”一定可以产生最好的结果
- 搜索
  - 几乎是通用的办法
  - 不过开销巨大

- 算法正确性分析循环是关键
- 以插入排序正确性分析为例：

Lemma 1

在每次循环开始时（第j步），$A[1, j-1]$是有序的。

Proof.

- 初始时：$A[1, j-1]$中只有一个数，显然成立。
- 保持性：在循环迭代开始前成立，迭代中，while循环找到正确位置插入$A[j]$。所以，这步迭代后命题仍成立。

□

- 终止时：将循环退出时的j值带入引理，得到插入排序的正确性。

循环不变性法要点

- 分析算法，归纳出循环不变性(循环迭代过程，始终成立的命题）。
- **Initialization**: 在第一次循环开始前不变性是满足的。
- **Maintenance**: 如果在某次迭代前不变性满足，在这次迭代后也满足不变性。
- **Termination**: 循环结束后，不变性可以提供程序正确性的有用信息。
- 例2，归并过程的正确性

Lemma 2

在*MERGE*中，第*14*步循环每次开始时，有

1. $A[p, k-1]$是有序的；
2. $A[p, k-1]$是最小的$k-p$个元素；
3. $L[i]$是L中待合并的元素中最小的；
4. $R[j]$是R中待合并的元素中最小的。

- **Initialization**: $k = p, i = 1, j = 1$，显然成立。
- **Maintenance**:若$L[i] \le R[j]$，则
  - $L[i]$是所有剩下的元素中最小的；
  - 而$A[p \cdots k-1]$中的元素均大于等于$L[i]$，因此
  - $A[p \cdots k]$是所有元素中最小的$k-p+1$个元素；
  - 由于L中的数是排好序的，因此$L[i+1]$将是L中剩下元素中最小的；
  - $R[j]$仍然是R中最小的。
  - 若$L[i] > R[j]$，可以进行类似讨论。
- **Termination**: 循环结束后，$k = r+1$，代入不变性1，得$A[p \cdots r]$有序。

算法复杂性度量

- 如何判断一个算法的好坏？
  - 算法占用资源的多少：处理时间和存储空间
  - 时间复杂性
  - 空间复杂性
- 不同的机器资源的性质不同，如何给定统一的标准？
- RAM机器模型:
  1. 单处理器
  2. 有限字长
  3. 基本的指令集（算术指令、数据移动指令、控制指令等），每种指令的执行时间为常量
  4. 无分级存储

算法复杂度度量

- 算法的运行时间可以用算法基本运算的步数度量
- 同一个算法，对不同的输入实例，运行时间不同
- 算法的复杂度是关于输入规模的函数
- 输入规模：输入数据的长度，通常可用输入元素的个数代替
    - 排序：输入数据的个数；
    - 最短路：图中顶点或边的数目；
    - 两个整数相乘：？
- 常用的复杂度：最好、最坏和平均

复杂度分析的基本方法

INSERTION-SORT(A)	cost	times
for $j \leftarrow 2$ **to** $length(A)$ **do**	c_1	n
$\quad key \leftarrow A[j]$	c_2	$n-1$
$\quad \triangleright$ Insert $A[j]$ into the sorted sequence $A[1, j-1]$.	0	$n-1$
$\quad i \leftarrow j - 1$	c_4	$n-1$
$\quad$ **while** $i > 0 \wedge A[i] > key$ **do**	c_5	$\sum_{j=2}^{n} t_j$
$\quad\quad A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
$\quad\quad i \leftarrow i - 1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
$\quad$ **end while**		
$\quad A[i+1] \leftarrow key$	c_9	$n-1$
end for		

最好情况

- 插入排序的复杂度为:

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j \\
&\quad + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_9(n-1)
\end{aligned}
$$

- 最好情况:$t_j = 1$带入上式:

$$
\begin{aligned}
T(n) &= (c_1 + c_2 + c_4 + c_5 + c_9)n - (c_2 + c_4 + c_5 + c_9) \\
&= an + b
\end{aligned}
$$

最坏情况

- 插入排序的复杂度为：

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j \\
&\quad + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_9(n-1)
\end{aligned}
$$

- 最坏情况:$t_j = j$带入上式：

$$
\begin{aligned}
T(n) &= \frac{c_5 + c_6 + c_7}{2} n^2 + (c_1 + c_2 + \frac{c_5 - c_6 - c_7}{2} + c_9) n \\
&\quad -(c_2 + c_4 + c_5 + c_8) \\
&= an^2 + bn + c
\end{aligned}
$$

平均情况

- 插入排序的复杂度为：

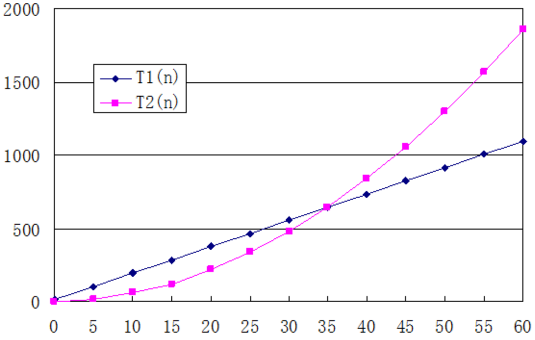
$$
\begin{aligned}
T(n) \quad = \quad & c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j \\
& + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_9(n-1)
\end{aligned}
$$

- 平均情况:$\mathbb{E}[t_j] = \frac{j}{2}$带入上式：

$$
\mathbb{E}[T(n)] \quad = \quad an^2 + bn + c
$$

如何比较复杂度函数

- 设$T_1(n) = 18n + 16$和$T_2(n) = \frac{1}{2}n^2 + n + 1$。
- 如何比较$T_1(n)$和$T_2(n)$?



- 抓住函数中的主要因素(增长最快的部分)。

渐进符号

- 大O

$$O(g(n)) \overset{def}{=} \{f(n)|\exists c, n_0 : \forall n > n_0, 0 \le f(n) \le cg(n)\}$$

- 大Ω

$$\Omega(g(n)) \overset{def}{=} \{f(n)|\exists c, n_0 : \forall n > n_0, 0 \le cg(n) \le f(n)\}$$

- 大Θ

$$\Theta(g(n)) \overset{def}{=} \{f(n)|\exists c_1, c_2, n_0 : \forall n > n_0, c_1 g(n) \le f(n) \le c_2 g(n)\}$$

- 小o

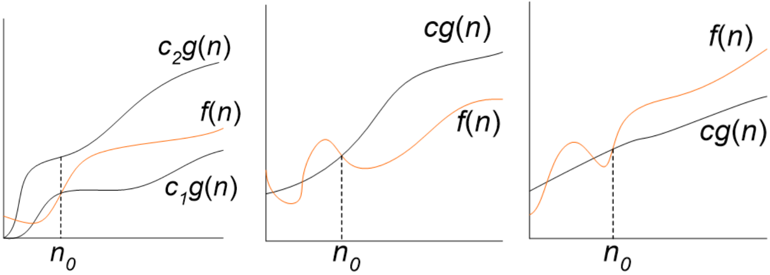
$$o(g(n)) \overset{def}{=} \{f(n)|\forall c, \exists n_0 : \forall n > n_0, 0 \le f(n) \le cg(n)\}$$

- 小ω

$$\omega(g(n)) \overset{def}{=} \{f(n)|\forall c, \exists n_0 : \forall n > n_0, 0 \le cg(n) \le f(n)\}$$

- 渐进符号定义的是函数集合，当我们提及$T(n) = O(n^2)$时，实际是指$T(n) \in O(n^2)$。

渐进符号含义示意图



(a) $f(n) = \Theta(g(n))$ (b) $f(n) = O(g(n))$ (c) $f(n) = \Omega(g(n))$

图: O, Ω, Θ示意图

Theorem 3

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

渐进符号应用例子

- 例1：$an+c=O(n), a>0$
- 根据定义，取$c'=a+|c|, n_0=1$，知$an+c=O(n)$。
- 例2：$an^2+bn+c=\Theta(n^2)$

$$
\begin{aligned}
an^2+bn+c &\leq |a|n^2+|b|n+|c| \\
&\leq (|a|+|b|+|c|)n^2 \\
&= O(n^2) \quad (let\ c'=|a|+|b|+|c|, n_0=1)
\end{aligned}
$$

$$
\begin{aligned}
an^2+bn+c &\geq |a|n^2-|b|n-|c| \\
&= \frac{|a|}{2}n^2+\frac{|a|}{2}n^2-|b|n-|c| \\
&\geq \frac{|a|}{2}n^2 \quad (let\ n_0=\frac{2(|b|+|c|)}{|a|}) \\
&= \Omega(n^2) \quad (let\ c'=\frac{|a|}{2})
\end{aligned}
$$

根据定理3，有

$$an^2+bn+c=\Theta(n^2).$$

复杂度比较的方法

- 设
$$L = \lim_{n\to\infty} \frac{f(n)}{g(n)}.$$
那么
  1. 若$L = a > 0$，则$f(n) = \Theta(g(n))$;
  2. 若$L = 0$，则$f(n) = o(g(n))$;
  3. 若$L = \infty$，则$f(n) = \omega(g(n))$。
- 例，
$$\sum_{i=0}^{d} a_i n^i = \Theta(n^d).$$

渐进符号的演算

- 传递性：
  - $f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$.
  - $f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$.
  - $f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$.
  - $f(n) = o(g(n)) \wedge g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$.
  - $f(n) = \omega(g(n)) \wedge g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$.
- 对称性：$f(n) = \Theta(g(n)) \Longleftrightarrow g(n) = \Theta(f(n))$.
- 自反性：
  - $f(n) = \Theta(f(n)), f(n) = O(f(n)), f(n) = \Omega(f(n))$.
- 反对称性：
  - $f(n) = o(g(n)) \Rightarrow g(n) = \omega(f(n))$.
  - $f(n) = \omega(g(n)) \Rightarrow g(n) = o(f(n))$.

常用函数

- $\sum_{i=0}^{d} a_i n^i = \Theta(n^d)$.
- $\sum_{i=1}^{n} i = \Theta(n^2)$.
- $\sum_{i=1}^{n} i^2 = \Theta(n^3)$.
- $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$.
- $\sum_{i=1}^{n} r^i = \Theta(r^n)$.
- $n! = O(n^n)$.
- $n! = \omega(2^n)$.
- Stirling公式：$n! = \sqrt{2\pi n}(n/e)^n(1+\Theta(1/n))$.
- $\log(n!) = \Theta(n \log n)$.

归并排序的复杂性分析

- **Divide:** 将n个待排的元素划分成两个分别有$n/2$个元素的子序列;
- **Conquer:** 对两个子序列递归地调用merge-sort进行排序;
- **Combine:** 将两个有序的子序列合并成n个元素的有序序列.

- 计算数组中间元素的下标: c;
- 递归调用: $T(\lfloor\frac{n}{2}\rfloor)+T(\lceil\frac{n}{2}\rceil)$;
- Merge过程的复杂度: cn.

$$T(n)=T(\lfloor\frac{n}{2}\rfloor)+T(\lceil\frac{n}{2}\rceil)+cn.$$

$$\Downarrow$$

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

$$\Downarrow$$

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

- 替换法的基本步骤：
  - 猜测解的形式；
  - 利用归纳法证明这个解并确定解中的常数。
- 替换法的关键是对方程的解做出好的猜测
- 替换法既可以用来证明上界也可以用来证明下界

替换法:例1

- 求方程$T(n) = 2T(\lfloor n/2 \rfloor) + n$的解(上界)。
  - 猜测：$T(n) \le cn\log n$。
  - 假设上述解在$T(n/2)$成立，替换上述方程：

$$
\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&= 2c\lfloor n/2 \rfloor \log\lfloor n/2 \rfloor + n \\
&\le 2c \cdot n/2 \log(n/2) + n \\
&= cn\log n - cn + n \\
&\le cn\log n (\text{if we take c} \ge 1).
\end{aligned}
$$

  - 归纳基础(base case):
    $T(1) = 1$，但猜测解在1时的值为0（$\log 1 = 0$)，这是不是意味着归纳的基础不成立？

替换法要注意的问题:例2

- 求方程$T(n) = 2T(\lfloor n/2 \rfloor) + n$的解(上界)。
  - 猜测：$T(n) = O(n)$, 即$T(n) = cn$.
  - 替换：

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &= 2c\lfloor n/2 \rfloor + n \\ &\leq cn + n \\ &= O(n) \end{aligned}$$

- 上述证明逻辑的问题在哪儿?

替换法技巧:例3

- 求方程$T(n)=T(\lfloor n/2\rfloor)+T(\lceil n/2\rceil)+1$的解(上界)。
  - 猜测：$T(n)=O(n)$. 令$T(n)\le cn$.
  - 替换：

$$\begin{aligned}T(n) &= T(\lfloor n/2\rfloor)+T(\lceil n/2\rceil)+1\\ &= c\lfloor n/2\rfloor + c\lceil n/2\rceil + 1\\ &= cn+1\end{aligned}$$

  - 失败了！
- 如果我们对猜测有信心，我们可以将猜测解紧一紧，令$T(n)=cn-b$，再采用替换法：

$$\begin{aligned}T(n) &= T(\lfloor n/2\rfloor)+T(\lceil n/2\rceil)+1\\ &= c\lfloor n/2\rfloor - b + c\lceil n/2\rceil - b + 1\\ &= cn-2b+1\\ &\le cn-b(\text{if we take b} \ge 1)\end{aligned}$$

替换法技巧:例4

- 求方程$T(n)=2T(\sqrt{n})+\log n$的解(上界)。
- 采用变量代换: $m=\log n$，这样

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

$$\Downarrow$$

$$T[2^m]=2T(2^{m/2})+m$$

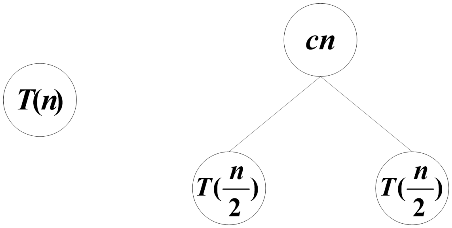
$$\Downarrow$$

$$S(m)=2S(m/2)+m$$

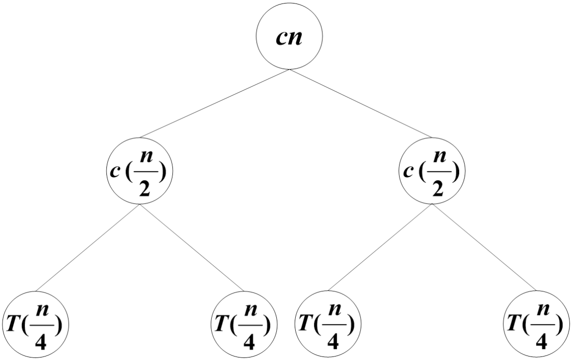
- $S(m)=O(m\log m)\Rightarrow T(n)=(\log n\log\log n)$.

递归树法:例1

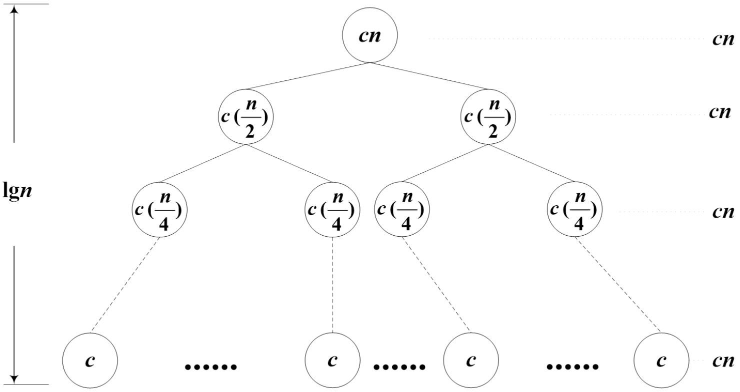
- 求方程$T(n) = 2T(n/2) + \Theta(n)$的解(上界)。



递归树法:例1

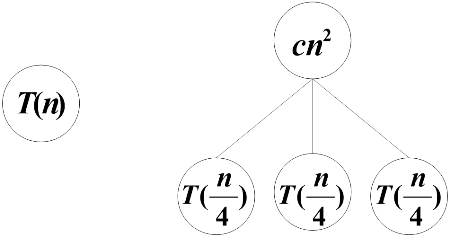


递归树法:例1

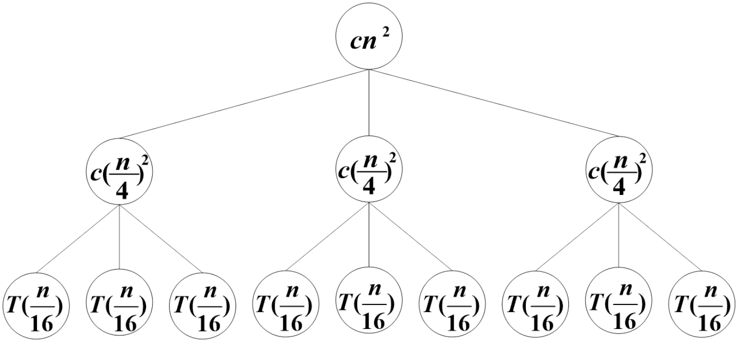


$$T(n) = \Theta(n \log n).$$

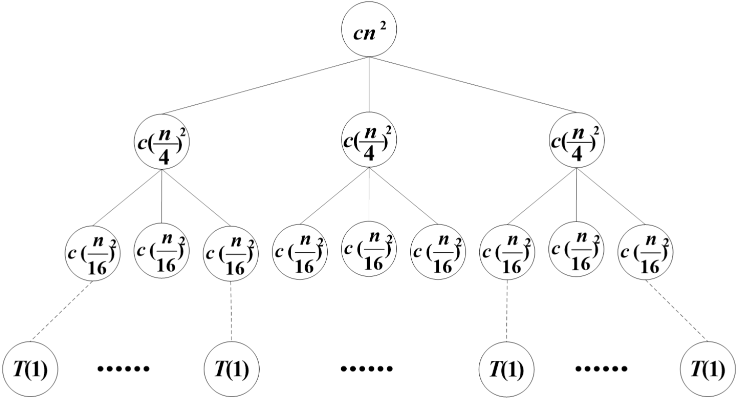
- 求方程$T(n) = 3T(n/4) + \Theta(n^2)$的解。



递归树法:例2


cn^2
$c(\frac{n}{4})^2$
$c(\frac{n}{4})^2$
$c(\frac{n}{4})^2$
$T(\frac{n}{16})$
$T(\frac{n}{16})$
$T(\frac{n}{16})$
$T(\frac{n}{16})$
$T(\frac{n}{16})$
$T(\frac{n}{16})$
$T(\frac{n}{16})$
$T(\frac{n}{16})$
$T(\frac{n}{16})$

递归树法:例2


cn^2
$c(\frac{n}{4})^2$
$c(\frac{n}{4})^2$
$c(\frac{n}{4})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$c(\frac{n}{16})^2$
$T(1)$ …… $T(1)$ …… $T(1)$ …… $T(1)$

递归树法:例2

- 递归树的深度$\log_4 n$，因此共有$\log_4 n+1$层.
- 第零层开销是cn^2，第一层是$3/16cn^2$,
- 第二层是$(3/16)^2cn^2$，依此类推.
- 叶子节点数目：$3^{\log_4 n}=n^{\log_4 3}$. 因此，

$$\begin{aligned}T(n) &= \sum_{i=0}^{\log_4 n-1}(3/16)^i cn^2+\Theta(n^{\log_4 3})\\ &= \frac{(3/16)^{\log_4 n}-1}{3/16-1}cn^2+\Theta(n^{\log_4 3})\\ &= O(n^2)\end{aligned}$$

$$T(n) = aT(\frac{n}{b}) + f(n) \tag{1}$$

- 根据$n^{\log_b a}$与$f(n)$的关系，分成3种cases：

case1 ：如果$f(n) = O(n^{\log_b a - \epsilon})$，其中$\epsilon$为某一常数，则

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

case2 ：如果$f(n) = \Theta(n^{\log_b a} \log^k n), k > 0$为某一常数，则

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

尤其当$k = 0$是，$T(n) = \Theta(n^{\log_b a} \log n)$.

case3 :如果$f(n) = \Omega(n^{\log_b a + \epsilon})$，$\epsilon$为某一常数，且当$n$足够大时满足$af(n/b) \leq cf(n), 0 < c < 1$，则

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