# 数据结构与算法笔记

X

2017 年 11 月 24 日

本文是作者关于数据结构与算法的读书笔记，侧重于记录和总结算法相关的数学方法，主要参考了Mark Allen Weiss的*数据结构与算法分析(C语言描述)*。本文的章节顺序，数学符号等都尽量与该书保持一致，同时也参考了网络资源或者其他书籍，均在对应章节或者习题序号下列出。由于水平所限，文中谬误在所难免，欢迎指正。

# 目录

# 1 初等数论基础

[夜深人静写算法：初等数论，http://www.cppblog.com/menjitianya/archive/2015/12/02/212395.html]

## 1.1 基本概念

### 1.1.1 整除性

若$a,b$为整数，$a$整除$b$是指$b$是$a$的倍数，$a$是$b$的约数，记做$a|b$。关于整除的性质有

1. 任意性：若$a|b$，则对于任意非零整数$m$，都有$am|bm$。
2. 传递性：若$a|b$，$b|c$，则$a|c$。
3. 可消性：若$a|bc$且$a,c$互素，则$a|b$。
4. 组合性：若$c|a$且$c|b$，则对于任意整数$m,n$，都有$c|ma+nb$。

**Exercise 1.1.** 假设$x,y,z$均为整数，若$11|(7x+2y-5z)$，求证$11|(3x-7y+12z)$。

**Solution 1.1.**

*Proof.* 令$3x-7y+12z=m(7x+2y-5z)+11(ax+by+cz)$，其中$m,a,b,c$均为整数。

如果等式要成立，则两边$x,y,z$的系数均要相等，得到

$$\begin{cases} 7m+11a=3 \\ 2m+11b=-7 \\ -5m+11c=12 \end{cases} \tag{1}$$

可知其中的一个解为$m=2,a=-1,b=-1,c=2$。

故可以得到$3x-7y+12z=2(7x+2y-5z)+11(-1x-1y+2z)$。即$(3x-7y+12z)$可以分解为$11$与$(7x+2y-5z)$的加权之和。

又因为$11|(7x+2y-5z)$，以及$11|11$，故根据整除性的组合性质，$11|(3x-7y+12z)$。

□

# 2 算法分析

[主定理的证明，http://blog.csdn.net/u014627430/article/details/53510696]
[Mark Allen Weiss, 数据结构与算法分析，第十章]

## 2.1 算法复杂度的数学定义

**Definition 2.1.** 关于算法的复杂度本文使用如下定义

1. 如果对于所有足够大的$n$, $T(N)$的上界由$f(N)$的常数倍决定，也就是说，如果存在正常数$c$和$n_0$，使得当$N \geq n_0$时，都有$T(N) \leq cf(N)$，则记为$T(N) = \mathcal{O}(f(N))$。
2. 对于所有足够大的$n$，$T(N)$的下界由$g(N)$的常数倍决定，也就是说，如果存在正常数$c$和$n_0$，使得当$N \geq n_0$时，都有$T(N) \geq cg(N)$，则记为$T(N) = \Omega(g(N))$。
3. 如果对于所有足够大的$n$，$T(N)$的上界和下界由$h(N)$的常数倍决定，也就是说，如果存在正常数$c_1, c_2$和$n_0$，使得当$N \geq n_0$时，都有$c_1 g(N) \leq T(N) \leq c_2 g(N)$，则记为$T(N) = \Theta(g(N))$。
4. 如果$T(N) = \mathcal{O}(p(N))$ 且$T(N) \neq \Theta(p(N))$，则$T(N) = o(p(N))$。

## 2.2 算法复杂度的性质

**Theorem 2.1.** 如果$T_1(N) = \mathcal{O}(f(N))$且$T_2(N) = \mathcal{O}(g(N))$，那么

- $T_1(N) + T_2(N) = \mathcal{O}\left(\max\{f(N), g(N)\}\right)$ 。
- $T_1(N)T_2(N) = \mathcal{O}\left(f(N)g(N)\right)$。

*Proof.* 因为$T_1(N) = \mathcal{O}(f(N))$，则存在正常数$c_1$和$n_1$，使得当$N \geq n_1$时，都有$T_1(N) \leq c_1 f(N)$；又因为$T_2(N) = \mathcal{O}(g(N))$，同理存在正常数$c_2$和$n_2$，使得当$N \geq n_2$时，都有$T_2(N) \leq c_2 g(N)$。

令$n = \max\{n_1, n_2\}$，故当$N \geq n$时，都有

$$\begin{aligned} T_1(N) + T_2(N) &\leq c_1 f(N) + c_2 g(N) \\ &\leq \max\{c_1, c_2\}\left(f(N) + g(N)\right) \\ &\leq 2\max\{c_1, c_2\}\max\{f(N), g(N)\} \end{aligned}$$

令$c = 2\max\{c_1, c_2\}$，则说明当$N \geq n$，都有$T_1(N) + T_2(N) \leq c\max\{f(N), g(N)\}$，即$T_1(N) + T_2(N) = \mathcal{O}\left(\max\{f(N), g(N)\}\right)$ 。

对于第二条性质，令$d = c_1 c_2$，则当$N \geq n$，都有$T_1(N)T_2(N) \leq df(N)g(N)$，即$T_1(N)T_2(N) = \mathcal{O}\left(f(N)g(N)\right)$。

□

**Theorem 2.2.**

对于任意整数$k$, $\log^k N = \mathcal{O}(N)$。该条定理说明对数增长非常缓慢。

*Proof.* 当$k < 0$时，$\log^k N < 1 = \mathcal{O}(N)$。

下面用数学归纳法证明$k \geq 0$ 的情况。

- $k=0$时，$\log^k N = 1 = \mathcal{O}(N)$
- 假设$k=i$时都有$\log^k N = \mathcal{O}(N)$，当$k=i+1$时

$$\begin{aligned}
\lim_{N\to\infty}\frac{\log^{i+1} N}{N} &= \lim_{N\to\infty}(i+1)\frac{\log^i N}{N} \quad (\text{根据洛必达法则}) \\
&= (i+1)\lim_{N\to\infty}\frac{\log^i N}{N} \\
&= (i+1)*0 \quad \left(\text{根据}\log^k N = \mathcal{O}(N)\text{可知}\lim_{N\to\infty}\frac{\log^i N}{N}=0\right) \\
&= 0
\end{aligned}$$

所以对于任意整数$k$，都有$\log^k N = \mathcal{O}(N)$。 □

## 2.3 复杂度方程的解法

**Lemma 2.1.** 假设定义在非负整数上的函数$f(n), g(n)$满足关系$f(n) = af(\frac{n}{b}) + g(n)$，其中实数$a \geq 1$，整数$b > 1, n > 0, k > 0$且$n = b^k$，则有

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

*Proof.* 由$f(n), g(n)$的关系联立得到$k$个等式

$$\begin{aligned}
f(n) &= af(\frac{n}{b}) + g(n) \\
f(\frac{n}{b}) &= af(\frac{n}{b^2}) + g(\frac{n}{b}) \\
&\cdots \\
f(b) &= af(1) + g(b)
\end{aligned}$$

对这$k$个等式依次两边分别乘以$1, a, a^1, ...a^{k-1}$，再求和，消去等式两边相等的项，得到

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

□

**Theorem 2.3** (Master Theorem)**.**

$T(N)$是定义在非负整数的函数，满足

$$T(N) = aT(\frac{N}{b}) + cN^k$$

其中$a, k$为实数，$b$为整数，且$a \geq 1, b > 1$，另外满足$N = b^m, m$为整数，则

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

*Proof.* 根据引理(2.1)可知

$$\begin{aligned} T(N) &= a^m T(1) + c \sum_{j=0}^{m-1} a^j \left(\frac{N}{b^j}\right)^k \\ &= a^{\log_b N} T(1) + cN^k \sum_{j=0}^{\log_b N - 1} \left(\frac{a}{b^k}\right)^j \end{aligned} \tag{2}$$

当$a = b^k$, 公式(2)等价于

$$\begin{aligned} & b^{k \log_b N} T(1) + cN^k \sum_{j=0}^{\log_b N - 1} \left(\frac{b^k}{b^k}\right)^j \\ = \ & \left(b^{\log_b N}\right)^k T(1) + cN^k \log_b N \\ = \ & N^k T(1) + + cN^k \log_b N \end{aligned}$$

根据定理(2.1)算法的复杂度由具有较大增长次数的部分决定，那么

$$T(N) = \mathcal{O}(N^k \log_b N) = \mathcal{O}(N^k \log N)$$

当$a \neq b^k$时,

$$\begin{aligned} T(N) &= a^{\log_b N} T(1) + c \sum_{j=0}^{\log_b N - 1} \left(\frac{a}{b^k}\right)^j N^k \\ &= a^{\log_b N} T(1) + cN^k \frac{\frac{a}{b^k}\left[1 - \left(\frac{a}{b^k}\right)^{\log_b N - 1}\right]}{1 - \frac{a}{b^k}} \\ &= a^{\log_b N} T(1) + \frac{cN^k \left[a - \frac{a^{\log_b N}}{b^{k(\log_b N - 1)}}\right]}{b^k - a} \\ &= a^{\log_b N} T(1) + \frac{cN^k}{b^k - a}\left(a - \frac{a^{\log_b N} b^k}{b^{k \log_b N}}\right) \\ &= a^{\log_b N} T(1) + \frac{acN^k}{b^k - a} - \frac{cN^k a^{\log_b N} b^k}{(b^k - a)\, b^{k \log_b N}} \end{aligned} \tag{3}$$

根据换底公式可知$a^{\log_b N} = N^{\log_b a}$，故公式(3)第一项等价于$N^{\log_b a} T(1)$。
进一步化简公式(3)第三项可得

$$\frac{cN^k a^{\log_b N} b^k}{(b^k - a)\, b^{k \log_b N}} = \frac{cN^k N^{\log_b a} b^k}{(b^k - a)\, N^k} = \frac{cb^k N^{\log_b a}}{b^k - a}$$

所以公式(3)可化简为

$$\begin{aligned} T(N) &= N^{\log_b a}T(1) + \frac{acN^k}{b^k - a} - \frac{cb^k N^{\log_b a}}{b^k - a} \\ &= N^{\log_b a}\left(T(1) - \frac{cb^k}{b^k - a}\right) + N^k\left(\frac{ac}{b^k - a}\right) \end{aligned}$$

当$a < b^k$，$\log_b a < \log_b b^k = k$

$$T(N) = \mathcal{O}\left(N^k\left(\frac{ac}{b^k - a}\right)\right) = \mathcal{O}(N^k)$$

当$a > b^k$，$\log_b a > \log_b b^k = k$

$$T(N) = \mathcal{O}\left(N^{\log_b a}\left(T(1) - \frac{cb^k}{b^k - a}\right)\right) = \mathcal{O}(N^{\log_b a})$$

□

# 3 树

[Mark Allen Weiss, 数据结构与算法分析，第四、七章]

## 3.1 二叉树

**Lemma 3.1.** 深度为$d$的二叉树最多有$2^d$片树叶。

*Proof.* 用数学归纳法证明。

1. 当深度$d = 0$时，二叉树只有一个根，最多存在一个树叶（就是根自己），所以基准情况为真。
2. 假设当深度为$d-1$时($d$是大于0的整数)，二叉树最有$2^{d-1}$片树叶。

   那么对于深度为$d$的树$T$，其左右子树$T_1, T_2$，每一个的深度最多是$d-1$。根据假设，其左右子树最多有$2^{d-1}$个树叶。

   $T$的树叶等于左右子树的树叶之和，其最多等于$2^{d-1} + 2^{d-1} = 2^d$。

□

**Lemma 3.2.** 具有$L$片树叶的二叉树深度至少是$\lceil \log L \rceil$

*Proof.* 由引理(3.1)可立即得出。 □

## 3.2 二叉查找树

二叉查找树(Binary Search Tree)是指一颗空树或者具有如下性质的二叉树：

1. 若任意节点的左子树不空，则左子树上所有节点的值均小于它的根节点的值;
2. 若任意节点的右子树不空，则右子树上所有节点的值均大于它的根节点的值;
3. 任意节点的左、右子树也分别为二叉查找树;
4. 没有键值相等的节点。

二叉查找树相对于其他数据结构的优势在于查找、插入的时间复杂度较低，为$\mathcal{O}(\log N)$。

# 4 树

[Mark Allen Weiss, 数据结构与算法分析，第七章]

## 4.1 排序算法的一般下界

任何只使用比较方法来排序的算法都可以用决策树来表示：决策树的每一个节点表示状态空间，包含在比较之前可能的排序, 每条边表示比较的结果。

1. 排序算法的终止条件是节点(实际上就是树叶)只存在一种顺序状态。
2. 排序算法所使用的比较次数等于树的深度。

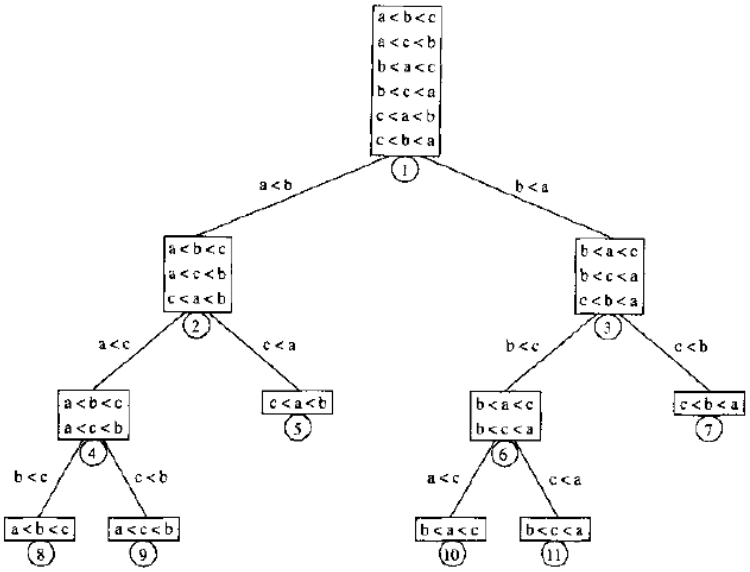


图 7-17 三元素排序的决策树

**Theorem 4.1.**