第一章　准备知识

1.1　数学归纳法

假设我们对于自然数（正整数）的定义有共识。这其中包括如下的事实：

- 每个非空的正整数集合都有一个最小元素。

所以对于某个关于正整数的命题，我们有：

- 如果关于正整数的一个命题不成立（有反例），则该命题有最小反例。

基于最小反例的存在性，我们有：

- 如果命题P(n)对于某个正整数不成立，则存在n满足

$$P(1) \wedge P(2) \wedge \cdots \wedge P(n-1) \wedge \neg P(n) = true$$

注意到$p \to q = \neg p \vee q = \neg(p \wedge \neg q)$。所以将上述命题变换为其逆否命题，我们有：

- 如果$P(1) \wedge P(2) \wedge \cdots \wedge P(n-1) \to P(n)$对每个正整数n都成立，则命题P(n)对每个正整数都成立。

1.2　实数的定义

对于实数的由来，可以有这样一种理解方式：定义实数的一个动机是使得数域对于求极限运算封闭。如下两个例子很好地说明了有理数集合对于求极限的不封闭：

$$\begin{aligned} \frac{\pi}{4} &= 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots \\ \frac{\pi^2}{6} &= \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots \end{aligned}$$

Theorem 1.2.1
单调且有界的数列一定有极限。

Theorem 1.2.2 (闭区间套定理)
设$I_n = [a_n, b_n]$，n为正整数，且$I_1 \supset I_2 \supset I_3 \supset \cdots \supset I_n \supset I_{n+1} \supset \cdots$。如果这一列区间的长度$|I_n| = b_n - a_n \to 0\ (n \to \infty)$，那么交集$\cap_{n=1}^{\infty} I_n$含有唯一的点。

Theorem 1.2.3 (Bolzano-Weierstrass定理，列紧性定理)
从任何有界的数列中必可选出一个收敛的子列。

Theorem 1.2.4
一个数列收敛的充分必要条件是它是Cauchy列[1]。

Theorem 1.2.5
非空的有上界的集合必有上确界；非空的有下界的集合必有下确界。

Theorem 1.2.6 (Heine-Borel定理，有限覆盖定理)
设$[a, b]$是一个有限闭区间，并且它有一个开覆盖$\{I_\lambda\}$，那么从这个开区间族中必可选出有限个成员（开区间）来，这有限个开区间所成的族仍然是$[a, b]$的开覆盖。

1.3　从算法的角度重新审视数学的概念

很多数学概念，虽然我们已经知晓其定义，但是从算法设计与分析的角度来说，我们还需要进一步将这些数学的概念与具体的算法现象联系起来。下面我们就从这一角度来讨论几组算法设计与分析中常用的数学概念。

1.3.1　取整$\lfloor x \rfloor, \lceil x \rceil$

下取整函数$\lfloor x \rfloor$表示不超过x的最大的整数。上取整函数$\lceil x \rceil$表示不小于x的最小的整数。在算法设计与分析中，我们一般考虑的是x为正实数的情况。取整函数在算法设计与分析中经常被用到，这是因为算法主要处理的是一些离散的对象，例如节点个数、不同排列的个数等。这些离散对象的数学性质一般是由正整数来刻画的。但是当我们做除法、对数等运算的时候，结果常常不是整数。我们需要对结果进行取整，才能严格地刻画离散的对象。

例如在折半查找中，当我们对起点1和终点n作折半的时候，粗略地讲，中点在$\frac{n+1}{2}$的位置。但是精确来说，如果n为偶数，则$\frac{n+1}{2}$不是一个整数，它不是任何一个位置的合法的下标值。此时，我们只需要用取整函数，就可以严格的表达出中点的位置。此时可能的中点有左右两个，它们分别是$\lfloor \frac{n+1}{2} \rfloor$和$\lceil \frac{n+1}{2} \rceil$，在实际算法设计中，我们往往选其中的一个作为折半的中点。类似地，对于任意n个数，我们定义其中位数为第$\lceil \frac{n}{2} \rceil$大的数。

[1] 设$\{a_n\}$是一个实数列。对于任意给定的$\epsilon > 0$，若存在正整数N，使得对任意的正整数$m, n > N$都有$|a_m - a_n| < \epsilon$，则称数列$\{a_n\}$是一个基本列或者Cauchy列。

再例如我们在做分治求解的时候，问题的规模缩小到原来的$\frac{1}{k}$（k为某个大于等于2的自然数）。随着分治递归的进行，经常会出现问题的规模不是k的倍数的情况，此时我们同样可以利用取整函数来精确地描述划分后的子问题的规模。

另外很多算法操作的分析涉及到对数运算（详见下一小节的讨论）。当我们作对数运算时，经常要将取对数的结果再经过取整，来刻画离散的对象。

1.3.2　对数$\log n$

在算法设计与分析中，至少下面三件事情与对数$\log n$ [2] 有比较紧密的关联，我们详细讨论如下：

- 折半：当我们采用分治策略来划分规模为n的问题空间的时候，大约经过$\log n$次划分，问题规模会降低到初始情况（base case）。不同的划分方法（例如两等分，或者三等分）、不同的初始情况大小（初始情况的规模可能是1也可能是稍大的某个常数值），结果会有细节的不同，但是具体的值总是$\log n$乘以某个常系数。另外，由于$\log n$的值并不总是整数，所以我们还需要借助上面讨论的取整函数才能精确描述问题规模降到初始情况的次数。例如规模为n 的问题，至多经过$\lceil \log n \rceil$次折半规模降到1。

- 二叉树：一个n个节点的完全二叉树，它的高度为$\lfloor \log n \rfloor$。更一般地，一个均衡的二叉树的高度大约为$\log n$。

  二叉树的高度和上面我们讨论的“折半至初始情况的次数”之间有紧密的联系。为了简化讨论，我们考虑完全二叉树。一个有n个节点的完全二叉树，有$\frac{n+1}{2}$个叶节点，即大约有一半的节点为叶节点。这样，去掉树的所有叶节点，树的高度减少1；同时从节点数目的角度看，节点数目减少一半。因此完全二叉树的高度与折半至初始情况的次数均大约为$\log n$。

  另外，如果我们按层从上到下，每层从左到右将完全二叉树的节点分别标号$1, 2, 3, 4, \cdots$，如图1.1所示。则二叉树的分层可以看成是对正整数的一个等价类划分，而$\lfloor \log n \rfloor$是划分的依据：

  $$1|2,3|4,5,6,7|8,9,\cdots,15|16\cdots$$

  上述的整数划分，可以帮助我们推导一些与对数、取整函数相关的结论，而这些结论往往在我们分析相关的算法过程中发挥作用。

- 整数n二进制表示的比特数：一个整数n的二进制表示所需的比特数为$\lfloor \log n \rfloor + 1$。这跟前面的折半同样有密切的关联。我们将一个二进制的数去掉一个比特约等于将这个数除以2。这样，整数n二进制表示的比特数，也就约等于将n折半至初始情况的次数，即大约$\log n$。

1.3.3　阶乘$n!$

对于n个不同的元素，其全排列的个数为$n!$，而元素的排列在算法设计与分析中有很多应用。例如，对于待排序的n个不同的元素，它们输入时的初始顺序是所有$n!$种排列中的某一种，而它们排好序后的结果，也是所有排列中的一种。对于一个排序算法，如果我们假设所有可能的输入等概率出现，则每种输入出现的概率为$\frac{1}{n!}$。

[2] 不作特别说明时，我们用$\log n$ 表示$\log_2 n$。

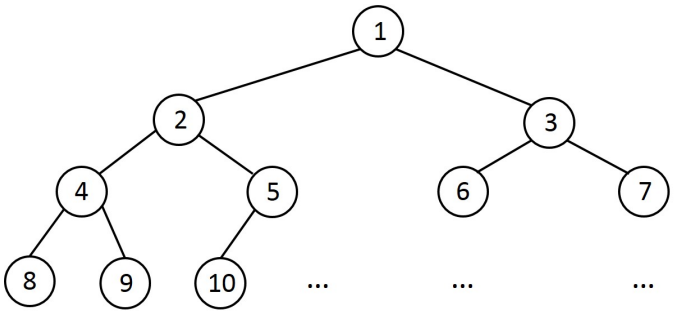

图 1.1: 二叉树节点按层标号

阶乘是一个连乘形式，对它取对数的话则变成求和形式，而连乘和求和形式都不易处理。为此，我们可以利用Stirling公式，将阶乘转换成了更易于处理的形式。具体而言，对于$n \geq 1$我们有：

$$\left(\frac{n}{e}\right)^n \sqrt{2\pi n} < n! < \left(\frac{n}{e}\right)^n \sqrt{2\pi n}\left(1 + \frac{1}{11n}\right)$$

如果借用Θ记号，Stirling公式又可以写成：

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

1.3.4　求和Σ

求和与算法分析的关系非常紧密。算法时间复杂度定义本身就是某种求和：所有关键操作出现次数的和。而平均情况时间复杂度则是结合概率分布的加权求和（即期望值）。再例如当我们结合递归树解分治递归时，我们本质是在做sum of row sums，即对递归树按层求和再对所有层求和。为此，掌握一些常用求和的闭形式对算法分析是有帮助的：

- *多项式级数*（polynomial series）：我们分别给出三组多项式级数，其中一次方的情况是大家熟知的算术级数（arithmetic series）；k次方的情况对于算法设计与分析来说，更有用的是记住其使用Θ记号的结果形式。

$$\begin{aligned}
\sum_{i=1}^{n} i &= \frac{n(n+1)}{2} \\
\sum_{i=1}^{n} i^2 &= \frac{1}{3}n(n+\frac{1}{2})(n+1) \\
\sum_{i=1}^{n} i^k &= \Theta\left(\frac{1}{k+1}n^{k+1}\right)
\end{aligned}$$

- *几何级数*（geometric series）：一般形式为

$$\sum_{i=0}^{k} ar^i = a\left(\frac{r^{k+1}-1}{r-1}\right)$$

它的两个常用的特例是a=1，r=2或者$\frac{1}{2}$的情况：

$$\begin{aligned}\sum_{i=0}^{k}\frac{1}{2^i} &= 2-\frac{1}{2^k}\\ \sum_{i=0}^{k}2^k &= 2^{k+1}-1\end{aligned}$$

- 算术几何级数（arithmetic-geometric series）：

$$\sum_{i=1}^{k}i2^i=(k-1)2^{k+1}+2$$

- 调和级数（harmonic series）：

$$\sum_{i=1}^{k}\frac{1}{i}=\ln k+\gamma+\epsilon_k$$

这里γ是欧拉常数；ϵ_k是一个小量，它近似为$\frac{1}{2k}$，且随着级数项数k的增加而趋于0。

- 斐波那契数列（Fibonacci numbers）：

$$\begin{aligned}F_n &= F_{n-1}+F_{n-2}, n\geq 2\\ F_0 &= 0,\ F_1=1\end{aligned}$$

斐波那契数列的解为：

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

1.3.5　期望$E[X]$

随机变量及其期望值的相关概念主要在算法的平均情况时间复杂度分析中使用。正如我们在小节所定义的，针对不同的输入，算法是否做某个操作可以建模为一个随机变量，它满足某种概率分布，这样算法的平均情况时间复杂度就和随机变量的期望值关联了起来。期望值有如下重要的性质：

Proposition (Linearity of Expectation) 给定任意随机变量$X_1, X_2, \cdots, X_k$ 和它们的线性函数$h(X_1, X_2, \cdots, X_k)$。我们有：

$$E[h(X_1, X_2, \cdots, X_k)]=h(E[X_1], E[X_2], \cdots, E[X_k])$$

注意，这里的任意k个随机变量，无论它们是否独立，上述等式均成立。我们最常用的多个随机变量的线性函数就是它们的求和，即：

$$E\left[\sum_{i=1}^{n}x_i\right]=\sum_{i=1}^{n}E[x_i]$$

有一类特别的随机变量叫指标随机变量（indicator random variable）。指标随机变量与某个随机事件相关联：如果该事件发生，则指标随机变量取值为1；事件未发生则取0。具体定义为：

$$X_i=I\{\text{事件}e\}=\begin{cases}1，\text{事件}e\text{发生}\\ 0，\text{事件}e\text{未发生}\end{cases}$$

我们很容易验证，对于指标随机变量而言，$E[X_i]$就等于事件e发生的概率。

指标随机变量在分析算法的平均情况时间复杂度过程中有广泛的应用。例如，我们可以用指标随机变量描述排序的代价。假设有待排序元素$Z = \{z_1, z_2, \cdots, z_n\}$。我们定义指标随机变量$X_{ij} = I\{z_i$和$z_j$发生了比较$\}$，则排序的代价则可以用指标随机变量表示为：

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

进而排序算法的平均情况时间复杂度可以表示为：

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \qquad \text{（linearity of expection）} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr\{z_i\text{和}z_j\text{发生了比较}\} \end{aligned}$$

基于指标随机变量这一数学工具，我们将排序算法平均情况时间复杂度分析的问题，转换为了一些（相对简单的）事件发生概率的分析，进而再求和的问题。

1.4　算法设计基础

蛮力算法：后续改进的“跳板”。

- 名人问题：对“∀”和“∃”特性的分析
- 常见项问题：基于蛮力算法猜测高效解法的效率 ⇒ 基于猜测的效率反推算法的设计