注意

这里的内容是我在ppt中挑选的，并未涵盖所以考点，所以还是需要看看ppt和书

引言

1. 关注问题是否可以求解的领域称为**可计算理论**
2. 算法运算时需要两种资源：**时间资源**与**空间资源**。需要时间资源的量叫**时间复杂度**，需要空间资源的量叫**空间复杂度。**
3. 对算法来说，时间资源和空间资源相比较而言**时间资源**更为宝贵！
4. 算法是对问题求解过程的**准确描述**，由**有限条指令**组成，这些指令能在**有限时间**内执行完毕并产生**确定性的输出。**
5. 算法复杂度依赖于三个方面：**待求解问题的规模**、**算法的输入**和**算法本身。**
6. 可操作性最强、最有实际价值的是算法**最坏情形下**的时间复杂度。
7. 一个编程实现了的算法的具体运行时间，不仅仅和算法本身相 关，还和很多其他因素密切相关：**机器性能**、**编程语言**、**编译器**、**编程技巧**等等。

算法需满足的4个性质：

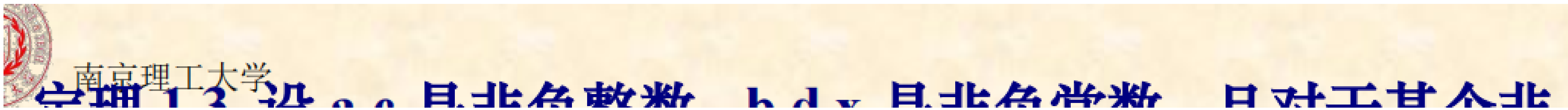
– **输入**：零个或多个外部量作为输入。 – **输出**：至少产生一个量作为输出, 它(们)与输入量之间存在某种特 定的联系。 – **确定性**：组成算法的每条指令都是清晰、无歧义的。 – **有限性**：每条指令的执行次数有限，执行每条指令的时间也有限。

阶

• 进行算法的时间复杂度分析，就是求其T(n)，并用O、 Ω或是Θ以尽可能简单的形式进行表示。 • 理想情况下，希望能够使用**Θ**表示算法的时间复杂性。 退而求其次，可以使用**O或是Ω**。 • 使用O时，希望估计的上界的阶**越小**越好。 • 使用Ω时，希望估计的下界的阶**越大**越好。

•"低阶复杂度的算法比高阶复杂度的算法效率高"这 个结论，只是在**问题的规模充分大**时才成立。 – 复杂度分别为n3与10n2：当n>10时，n3>10n2; 而n<10时，n3<10n2，即复杂度为n3的算法更有效。 • 在问题规模较小时，我们往往并不一味追求低复杂度 的算法，而是更侧重于算法的**简单性。** • 当两个算法的渐近复杂度的阶相同时，必须进一步综 合考察渐近复杂度表达式中的**低阶及常数因子**才能判 别它们的优劣。 – 例如：复杂度为n1ogn/l00 的算法显然比复杂度为l00n1ogn 的算法来得有效。

递推式



定理 1.3 设 a,c 是非负整数，b,d,x 是非负常数，且对于某个非负整数 k 有 $n=c^k$.那么，对于递推式

$$f(n)=\begin{cases} d & \text{if } n=1 \\ af(n/c)+bn^x & \text{if } n\ge 2 \end{cases}$$

的解是

$$f(n)=\begin{cases} \Theta(n^x) & \text{if } a<c^x \\ \Theta(n^x \log n) & \text{if } a=c^x \\ \Theta(n^{\log_c a}) & \text{if } a>c^x \end{cases}$$

特别地，若 x=1，则

$$f(n)=\begin{cases} \Theta(n) & \text{if } a<c \\ \Theta(n\log n) & \text{if } a=c \\ \Theta(n^{\log_c a}) & \text{if } a>c \end{cases}$$

Master Theorem

设a≥1, b >1为常数, f(n)为一给定的函数, T(n)递归定义如下：

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

并且 T(n)有适当的初始值。那么，当 n 充分大时，有：

1) 若有 ε > 0, 使 f(n)=O($n^{(Log_b a)-\varepsilon}$)

(即 f(n)的量级多项式地小于 $n^{Log_b a}$ 的量级)， 则 T(n)= Θ ($n^{Log_b a}$)。

2) 若 f(n)= Θ($n^{Log_b a}$)

(即 f(n)的量级等于 $n^{Log_b a}$ 的量级)， 则 T(n) =Θ($n^{Log_b a} Log\ n$)。

3) 若有 ε >0, 使 f(n)=Ω($n^{(Log_b a)+\varepsilon}$)

(即 f(n)的量级多项式地大于 $n^{Log_b^a}$ 的量级)， 且满足正规性条件：

存在常数 c<1, 使得对所有足够大的 n，有 a*f(n/b)≤c*f(n),

则 T(n)=Θ(f(n))。

正规性条件的直观含义:

对所有足够大的n，a个子问题的分解准备与再组合所需要的时间总和，严格

小于原问题的分解准备和组合所需要的时间。

分治策略

• 把问题实例**划分**为若干个子实例 • 分别递归地**求解**每个子实例 • 然后把这些子实例的解**组合**起来

贪心策略

贪心策略总是做出在**当前**看来最好的选择，并且不从**整体**上考虑最优问题，所做出的每一步选择只是**局部**意义上最优选择(因而效率往往**较高**)，逐步扩大解的规模。

动态规划

1. 与分治法类似，动态规划法也是把问题层层分解为规模较小的同类子问题。 二者之间一个重要的不同点在于， 分治法分解后得到的子问题通常都是**相互独立**的， 而动态规划法分解后得到的子问题很多都是**重复**的。

2. 动态规划的实质是**分治**和**消除冗余**，是一种将问题实例分解为更小的、相似的子问题，并存储**子问题的解**以避免**计算重复**的子问题，来解决**最优化问题**的算法策略

3. 基本步骤： 分析一个最优解应该具备的**结构。** **递归**地定义其最优解。 以**自底向上**的方式计算出最优值。 根据计算最优值时得到的信息，构造最优解。

4. 特点/基本要素 **子问题的高度重复性 最优子结构性质**： 问题的最优解中包含着其每一个子问题的最优解。

随机算法

1. **蒙特卡洛型算法**在一般情况下可以保证：对于问题的所有实例，都以高概率给出正确解，但是通常**无法判断一个解是否正确。** 蒙特卡洛型随机算法的特点是：**总是给出解，但偶尔解是错的，不知道一个解是对是错。** 然而，可以多次运行原算法，设法使得每次运行时的随机选择都相对独立，则可以使非正确解的概率可以减小到任意小。
2. **拉斯维加斯算法**特点是"**或者给出正确解**"、"**或者无解**"。该类型算法不时做出可导致算法陷入僵局的选择，并且算法能够检测是否陷入僵局，如果是，算法承认失败。但是，只要这种行为出现的概率不占多数，当出现失败时，只要在相同的输入实例上再次运行随机算法，就又有成功的可能。