绪论

基本概念

——计算机科学的核心在于研究计算方法与过程的规律，而不仅仅是作为计算工具的计算机本身。因此， E.Dijkstra 及其追随者更倾向于将这门科学称为计算科学（computing science）

计算 = 信息处理 = 借助某种工具，遵照一定规则，以明确而机械的方式进行。

计算模型 = 计算机 = 信息处理工具

算法

算法，即在特定的计算模型下，旨在解决特定问题的**指令序列**

算法的要素：

- 输入：待处理的信息（问题）
- 输出：经处理的信息（答案）
- 正确性
- 确定性：可以描述为一个有基本操作组成的序列
- 可行性：每一个基本操作都可以实现，且能在常数时间内完成。
- 有穷性：对于任意确定的输入，经过有穷次的基本操作，都可以得到输出。

> 证明算法的有穷性和正确性的一个重要技巧：审视整个计算过程，找到某种不变形和单调性（有效规模通常会不断递减）。

计算模型

统一尺度——如何判断一个算法的正确性质和成本？

图灵机

无限长的，分成无限多个单元格的纸带。

每个单元格上有一个“操作”，往左/往右/停机。

有一个HEAD指向某个单元格，每个“节拍”按照”操作“走一步；

启动到停机的节拍数目就是可以度量的成本。

RAM

Random Access Machine。

有无穷多个、可以 call by rank 访问的寄存器；

每一个基本操作仅需要常数时间。

算法执行的基本运算的次数就是可以度量的成本。

> 然而这些只考虑了时间，而没有考虑空间

算法效率 与 复杂度度量

我们考察解决某个问题的“成本”与问题实例的规模的关系，一般称之为算法效率，或者称之为复杂度。

复杂度分为两类——时间和空间。

记解决规模为 n 的问题所耗费的可度量”成本“为 $T(n)$ 。

渐进复杂度——Big O notation

我们更关心，在问题规模**足够大**的时候，计算成本（也就是复杂度）如何增长。

$\mathcal{O}(f(n))$ 表示上界。

$\Omega(f(n))$ 表示下界。

$\Theta(f(n))$ 表示同阶。

一般我们只用 $\mathcal{O}$ 来表示复杂度，但我们只写成 O 。

层级划分

$O(1) < O(\log^* n) < O(\log n)$

$O(n) < O(n\log^* n) < O(n\log\log n) < O(n\log n)$

$O(n^2) < O(n^3) < O(n^c) < O(2^n) < O(n!)$

复杂度分析

迭代——级数

算术级数：末项平方同阶。$T(n) = \sum i = n^2$

幂方级数：比幂次高出一阶。$\sum k^d = O(n^{d+1})$

几何级数：与末项同阶。$\sum a^k = O(a^n)$

收敛级数：$\sum \frac{1}{n^2} = O(1)$

其他级数:

- $\sum \frac{1}{n} = O(\log n)$
- $\sum \ln k = O(n \log n)$
- $\sum k \log k = O(n^2 \log n)$
- $\sum k \cdot 2^k = O(n \cdot 2^n)$

递归——主定理

Master Theorem 64

❖ **分治策略对应的递推式，通常（尽管不总是）形如：** $T(n) \ = \ a \cdot T(n/b) + \mathcal{O}(f(n))$

（原问题被分为a个规模均为n/b的子任务；任务的划分、解的合并耗时f(n)）

❖ **若** $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ **，则** $T(n) = \Theta(n^{\log_b a})$

- `kd-search`: $T(n) \ = \ 2 \cdot T(n/4) + \mathcal{O}(1) \ = \ \mathcal{O}(\sqrt{n})$

❖ **若** $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$ **，则** $T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$

- `binary search`: $T(n) \ = \ 1 \cdot T(n/2) + \mathcal{O}(1) \ = \ \mathcal{O}(\log n)$
- `mergesort`: $T(n) \ = \ 2 \cdot T(n/2) + \mathcal{O}(n) \ = \ \mathcal{O}(n \cdot \log n)$
- `STL mergesort`: $T(n) \ = \ 2 \cdot T(n/2) + \mathcal{O}(n \cdot \log n) \ = \ \mathcal{O}(n \cdot \log^2 n)$

❖ **若** $f(n) = \Omega(n^{\log_b a + \epsilon})$ **，则** $T(n) = \Theta(f(n))$

- `quickSelect (average case)`: $T(n) \ = \ 1 \cdot T(n/2) + \mathcal{O}(n) \ = \ \mathcal{O}(n)$

分摊分析

连续实时足够多次操作，所需总体成本摊还到单词的操作

更加忠实地刻画了可能出现的操作序列，更加精准地评判数据结构和算法的真实性能。

局限性

缓存

程序对于缓存的利用可能会比复杂度的影响全都要大。

字宽

实际问题中我们需要考虑字宽!

例如，a^n 的二进制展开宽度就是 $\Theta(n)$ 的，打印就需要这么多时间。但我们众所周知，快速幂是 $O(\log n)$ 的。

问题出在 RAM 模型的假设，并非所有操作都是 $O(1)$ 常数时间的。