1.4 绪论-算法分析

备注	课后练习!
学习日	@2020/03/07
视频起始	https://www.bilibili.com/video/av75509584?p=11
讲义	01.Introduction.D.Algorithm_analysis.pdf

▼ 01-D-1 算法分析

- 两个主要任务：正确性（不变性/单调性）+复杂性
- 确定后者就是要把算法转化为一些基本操作指令。（但不需要描述为RAM的基本指令，进而统计操作次数）
- 利用现有C++高级语言的基本指令，钧等效于RAM的基本指令

1. 分支转向：goto
2. 迭代循环：for(). while() -》 1)**级数求和**
3. 调用+递归（自我调用）-》2)**递归跟踪+递归方程**

3) 猜测+验证，三种方法进行分析

▼ 01-D-2 级数

▼ 算数级数：复杂度于末项平方同阶

$$T(n) = 1 + 2 + ... + n = n(n+1)/2 = \mathcal{O}(n^2)$$

▼ 幂方级数：比幂次高出一阶

有以下通项公式，中间第一步有一个近似过程

$$\sum_{k=0}^{n} k^d \approx \int_0^n x^{d+1} dx = \frac{1}{d+1} x^{d+1}|_0^n = \frac{1}{d+1} n^{d+1} = \mathcal{O}(n^{d+1})$$

则对应时间复杂度：

$$T_2(n) = 1^2 + 2^2 + 3^2 + ... + n^2 = n(n+1)(2n+1)/6 = \mathcal{O}(n^3)$$

▼ 几何级数（a>1）：时间复杂度等于末项同阶

$$T_a(n) = a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1) = \mathcal{O}(a^n)$$

如底数a=2，则复杂度如下所示：

$$1 + 2 + 4 + \ldots + 2^n = 2^{n+1} - 1 = \mathcal{O}(2^{n+1}) = \mathcal{O}(2^n)$$

▼ 收敛级数

❖ **收敛级数**

$$1/1/2 + 1/2/3 + 1/3/4 + \ldots + 1/(n-1)/n = \underline{1 - 1/n} = \mathcal{O}(1)$$

$$1 + 1/2^2 + \ldots + 1/n^2 < 1 + 1/2^2 + \ldots = \underline{\pi^2/6} = \mathcal{O}(1)$$

$$1/3 + 1/7 + 1/8 + 1/15 + 1/24 + 1/26 + 1/31 + 1/35 + \ldots = \underline{1} = \mathcal{O}(1)$$

- 有必要讨论这类分数级数吗?

例如投硬币过程，正面为lambda，那么第一次投出反面的概率即为以下：

❖ **有必要讨论这类级数吗？**

难道，基本操作次数、存储单元数可能是分数？某种意义上！

$1-\lambda$

$$(1-\lambda)\cdot[1 + 2\lambda + 3\lambda^2 + 4\lambda^3 + \ldots] = 1/(1-\lambda) = \mathcal{O}(1),\ 0 < \lambda < 1 \quad \text{//几何分布}$$

- 还有些不收敛，但是长度有限的级数

$$h(n) = 1 + 1/2 + 1/3 + \ldots + 1/n = \mathcal{O}(\log n)$$

$$\log 1 + \log 2 + \log 3 + \ldots + \log n = \log(n!) = \mathcal{O}(n \log n)$$

▼ 01-D-3 循环和级数

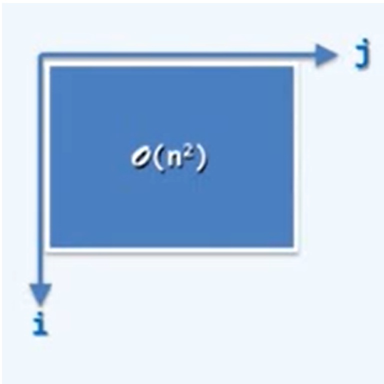
- 二重循环

```
for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
    O1Operation(i, j);
```

算术级数：

$$\sum_{i=0}^{n-1} n = n + n + \ ... + n = n * n = O(n^2)$$

矩阵面积即为计算所需次数



- 有限制的二重循环

```
for (int i = 0; i < n; i++)
for (int j = 0; j < i; j++)
    O1Operation(i, j);
```

算术级数：

$$\sum_{i=0}^{n-1} i = 0 + 1 + \ ... + (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

但是与上述算法在渐进意义上相同，运算次数相等于一个三角形

- 有限制的二重循环并带有变大的步长

  常系数的缩小：

```
❖ for (int i = 0; i < n; i++)
    for (int j = 0; j < i; j += 2013)
      O1Operation(i, j);
  算术级数： ...
```

- 进一步变化：i << = 1 相当于每次操作左移移位，等效于乘以2

```
❖ for (int i = 1; i < n; i <<= 1)
    for (int j = 0; j < i; j++)
      O1Operation(i, j);
```

几何级数：

$$1 + 2 + 4 + \ldots + 2^{\lfloor \log_2(n-1) \rfloor}$$

$$= \sum_{k=0}^{\lfloor \log_2(n-1) \rfloor} 2^k \qquad (\text{let } k = \log_2 i)$$

$$= 2^{\lceil \log_2 n \rceil} - 1 = \mathcal{O}(n)$$

即i的增大过程为1，2，4，8，一直到小于n的2的倍数，则次数最后可以整理为

$$2^{[\log_2]n} - 1 = \mathcal{O}(n)$$

- 习题分析：

```
❖ for (int i = 0; i <= n; i++)
  for (int j = 1; j < i; j += j)
      O1Operation(i, j);
```

几何级数：$\sum_{k=0}^{n}\lceil \log_2 i \rceil = \mathcal{O}(n\log n)$

(i = 0, 1, 2, 3~4, 5~8, 9~16, ...)

= 0 + 0 + 1 + 2*2 + 3*4 + 4*8 + ..

= $\Sigma_{k=0..\log n}(k * 2^{k-1})$

= $\mathcal{O}(\log n * 2^{\log n})$ (CM page#33)

若是，j每次翻倍增长，则可以发现，

当i = 0-1时，运行0次；
当i = 2时，运行1次；
当i = 3-4时，运行2次；
当i = 5-8时，运行3次；

则可以推断总运行次数最大可能为log2N次，则可以导出几何级数；

根据原理，几何级数算法复杂度即为最大想对应的：

$$\mathcal{O}(\log n 2^{\log n})$$

▼ 01-D-4 取非极端元素、起泡排序

接下来进行算法分析：

▼ 取非极端元素

❖ 问题：给定整数子集S，$|S| = n \geq 3$

找出元素 $a \in S$，$a \neq \max(S)$ 且 $a \neq \min(S)$

❖ 算法：从S中任取三个元素{x, y, z}

//若S以数组形式给出，不妨取前三个

//由于S是集合，这三个元素必互异

确定并排除其中的最小、最大者

//不妨设 $x = \max\{x, y, z\}$, $y = \min\{x, y, z\}$

输出剩下的元素z

❖ 无论输入规模n多大，上述算法需要的执行时间都不变

$T(n) = \text{常数} = \mathcal{O}(1) = \Omega(1) = \Theta(1)$

这个问题说明，只要对于一个集合取其中的任意三个互斥元素，找到这三个元素的非极端元素，即为这个集合的非极端元素

-》可以导出，存在一些算法，他可能并不随着问题的规模增大复杂度变大，所需的**复杂度则是保持为常数**

▼ 冒泡排序

给定n个整数，按照非降序排序

在无序的数列中，肯定出现有一对紧邻的数与所需方向相反，因此将会进行**扫描交换**！未再发现，则会继续终止！

开始，sorted置为true，如果遇到乱序，则sorted=!sorted，并且交换相邻元素，然后每次进行循环操作，直至结束

进行第K次交换后，则前K个数必定排序正确！

▼ 01-D-5 起泡排序的分析

- 不变性：**进行k轮交换后，则前K个数必定排序正确！**
- 单调性：**经过k轮交换后，问题会缩小至n-k → 必然会单调减小**
- 正确性：**经过至多n轮扫描后，算法必定终止，且正确解答**

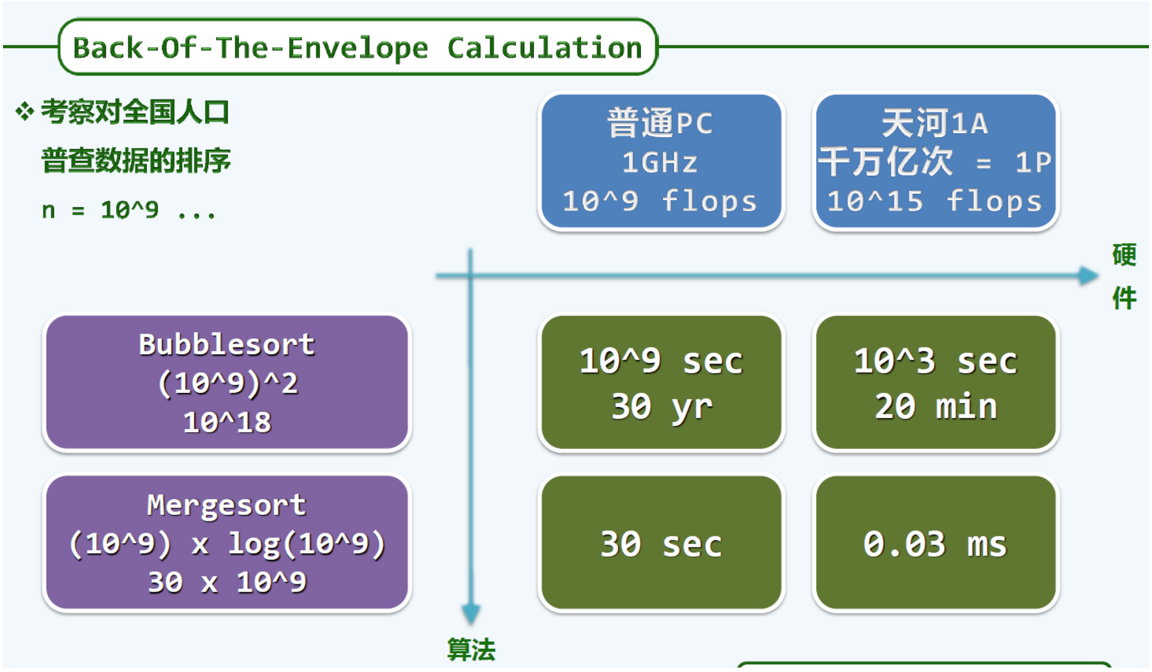
▼ 01-D-6 封底分析

Back-of-the-envelop Calculation

以小测大的定量估算方法

▼ 01-D-7 封底分析实例

算法与硬件的改进同样会促进计算的提高！



▼ **课后练习 !!!**

- 试按照“不变性＋单调性”的模式
- 归纳证明本章各算法的正确性
- 试举例说明，01Operation()对循环体的复杂度也可能有实质影响
- 学习不同开发环境提供的 Profiler工具，并藉此优化你的程序性能
- 习题[1-32]