算法概述

笔记源文件：[Markdown](), [长图](), [PDF](), [HTML]()

1. 概述

1.1. 算法是什么

1 算法概念：结局问题的方法/过程，是若干指令的又穷序列

2 算法的性质：有输入输出，确定性，有限性

1.2. 算法分析

1 算法复杂性：算法运行所需要的计算机资源，包括时间复杂性$T(n)$与空间复杂性$S(n)$，n是问题规模

2 算法分析：计算算法运行所需资源量

2. 算法时间复杂度分析

2.1. 概述

1 **最坏情况**时间复杂性：$T_{max}(n)=max\{T(I)\mid size(I)=n\}$，使用最多

2 **最好情况**时间复杂性：$T_{min}(n)=min\{T(I)\mid size(I)=n\}$

3 **平均情况**时间复杂性：$T_{avg}(n)=\sum\limits_{size(I)=n}p(I)T(I)$，$I$为规模$n$示例，$P(I)$为$I$出现概率

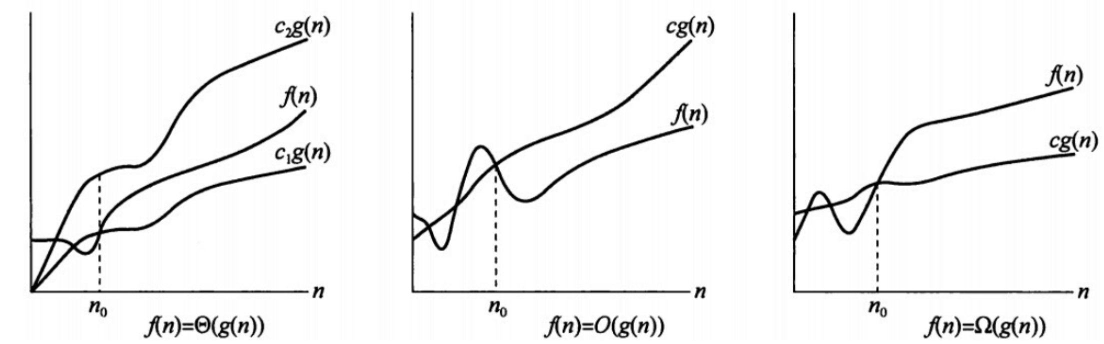
2.2. 渐进复杂度

1 渐进性态：$\lim\limits_{n\to+\infty}T(n)\to\infty$满足时，$T(n)$在$n$很大时的行为

2 渐进表达式：

1. $t(n)$满足$\lim\limits_{n\to+\infty}\frac{T(n)-t(n)}{T(n)}\to 0$时，就称$t(n)$为$T(n)$的渐近复杂度，
2. $t(n)$可近似描述$T(n)$长期行为

2.3. 渐进分析记号



1 渐进上界：

1. $O(g(n))=\{f(n)\mid\exists c>0,n_0$使得$\forall n\geq n_0:0\leq f(n)\leq cg(n)\}$

1. $O(g(n)) = \{f(n) \mid \exists c > 0, n_0$使得$\forall n \geq n_0 : 0 \leq f(n) \leq cg(n)\}$
2. 大致相当于$f(n) = O(g(n)) \leq g(n)$

➕非紧上界

1. $o(g(n)) = \{f(n) \mid \exists c > 0, n_0$使得$\forall n \geq n_0 : 0 \leq f(n) < cg(n)\}$
2. 大致相当于$f(n) = O(g(n)) < g(n)$

2️⃣渐近下界：

1. $\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0$使得$\forall n \geq n_0 : \Omega \leq cg(n) \leq f(n)\}$
2. 大致相当于$f(n) = \Omega(g(n)) \geq g(n)$

➕非紧下界

1. $\omega(g(n)) = \{f(n) \mid \exists c > 0, n_0$使得$\forall n \geq n_0 : \Omega \leq cg(n) < f(n)\}$
2. 大致相当于$f(n) = \Omega(g(n)) > g(n)$

3️⃣ 紧渐进界：

1. $\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0$使得$\forall n \geq n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n)\}$
2. 大致相当于$f(n) = \Theta(g(n)) = g(n)$

2.4. 渐进分析的算数/证明

1️⃣常用公式

1. $O(f(n)) + O(g(n)) = O(\max\{f(n), g(n)\}) = O(f(n) + g(n))$
2. $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$
3. $O(cf(n)) = O(f(n))$
4. $g(n) = O(f(n)) \Rightarrow O(f(n)) + O(g(n)) = O(f(n))$

2️⃣阶乘：$n! = \sqrt{2\pi n}(\frac{n}{e}^n)(1 + \Theta(\frac{1}{n}))$所以$n! = o(n^n)$，$n! = \omega(2^n)$，$\log(n!) = \Theta(n \log n)$

2.5. 分析法则

1️⃣ for/while 循环：循环体内计算时间*循环次数

2️⃣ 嵌套循环：循环体内计算时间*所有循环次数

3️⃣ 顺序语句：各语句计算时间相加

4️⃣ if-else语句：if语句计算时间和else语句计算时间的**较大者**

3. NP完全理论

3.0. 规约

1️⃣含义：归约是一种将一个问题转化为另一个问题的方法

2️⃣性质：每个最优化问题，都可规约为判定问题

3.1. P&NP类问题

1️⃣P类问题(Easy to Find)：

1. 含义：在确定型图灵机上多项式时间内可解决
2. 示例：求1-100总和，规定时间内可以求出

2 NP类问题(Easy to Check)：

1. 含义：在确定型图灵机上多项式时间可验证，在非确定型图灵机上多项式时间可解
2. 示例：求宇宙原子总数，求不出，但是你说原子总数=100那肯定错

3 二者关系：$P \subseteq NP$

3.2. NPC问题(NP完全问题)

1 如果一个问题是NP完全的，它满足两个条件：

1. 给定一个解，则可在多项式时间内验证这个解是否正确
2. 所有NP问题，都可在多项式时间内，归约(reduce)到这个NPC问题

2 公式：$X \in \mathrm{NPC} \iff \begin{cases} 1. \quad X \in \mathrm{NP} \\ 2. \quad \forall Y \in \mathrm{NP},\ Y \leq_p X \end{cases}$

1. $X \in \mathrm{NPC}/X \in \mathrm{NP}$表示$X$是$NPC/NP$问题
2. $Y \leq_p X$表示任意NP问题Y，都可在多项式时间内归约到问题X，$\leq_p$是多项式时间归约

3 示例：TSP问题，给定城市&城市间距离的距离，寻找一条最近路径能结果所有城市

这是一个NPC问题因为

1. 给出特定路径，可在多项式时间内算出是否经过所有城市&路径长
2. 已经证明了许多其他NP问题都可以在多项式时间内归约到TSP问题

3.3. NP难问题

1 含义：那些至少和NP中最难的问题一样难的问题

2 性质：所有的NP问题可以归约到NP-hard问题，但是NP-hard问题不一定是NP问题

3 公式：$X \in \mathrm{NP}_{hard} \iff$ 对于$\forall Y \in NP$, 有$Y \leq\ _pX$