算法分析与设计

Analysis and Design of Algorithm

Lesson 03

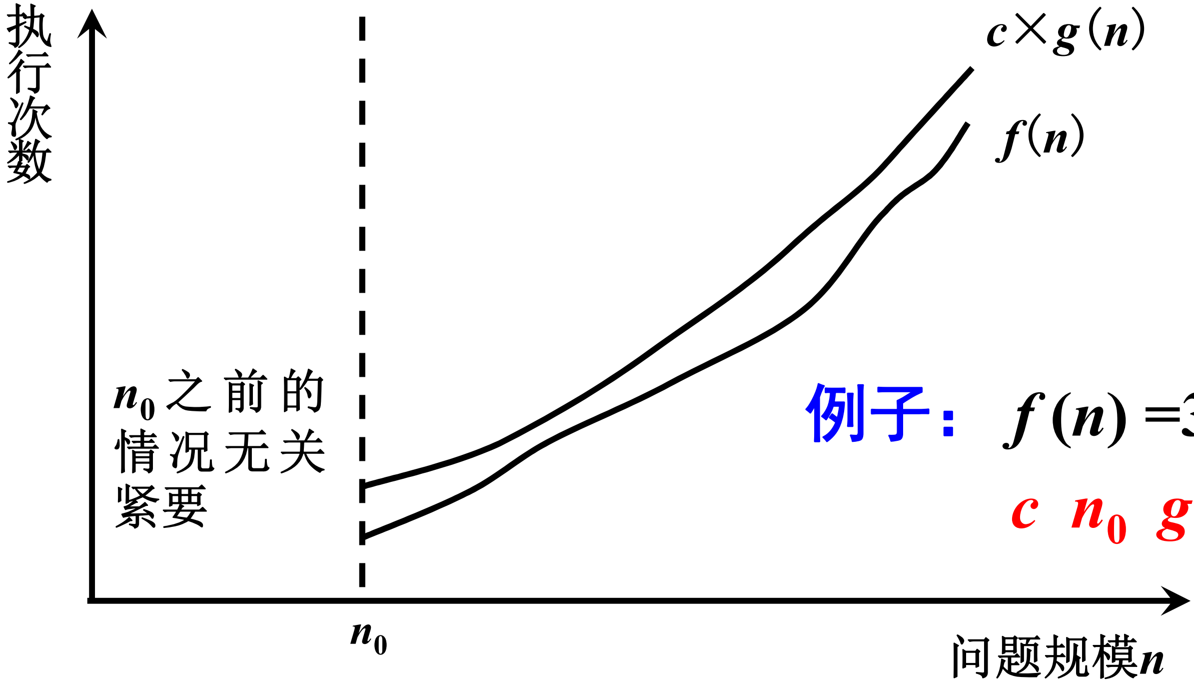
要点回顾

- **算法复杂度的概念**
  - 时间复杂度
    - 最坏情况时间复杂度$W(n)$
    - 平均情况时间复杂度$A(n)$
  - 三大衡量标准
    - 问题规模
    - 基本运算
    - 计量函数
- **复杂度的渐近性态**
  - 略去低阶项所留下的主项
  - 五个渐近分析记号

1. **渐近上界记号** O
2. **渐近下界记号** Ω
3. **紧渐近界记号** Θ
4. **非紧上界记号** o
5. **非紧下界记号** ω

渐近分析的记号

- **渐近上界记号O**
  - 若存在两个正的常数c和n_0，使得对所有$n \geqslant n_0$，都有：$f(n) \leqslant c \times g(n)$，则称$f(n) = O(g(n))$



例子：$f(n) = 32n^2 + 17n + 1$

$c \quad n_0 \quad g(n) = ?$

O的运算性质

- $O(f)+O(g)=O(\max(f,g))$
- $O(f)+O(g)=O(f+g)$
- $O(f)\cdot O(g)=O(f\cdot g)$
- 如果 $g(N)=O(f(N))\Rightarrow O(f)+O(g)=O(f)$
- $O(cf(N))=O(f(N))$ 其中c是一个正的常数
- $f=O(f)$

下面考察性质的证明：

性质：$O(f)+O(g)=O(\max(f,g))$

证明：设$F(N)=O(f)$。根据符号O的定义，存在正常数C_1和自然数N_1，使对所有$N \ge N_1$，都有$F(N) \le C_1 f(N)$。

类似地，设$G(N)=O(g)$，则存在正的常数C_2和自然数N_2，$N \ge N_2$有$G(N) \le C_2 g(N)$。

性质：$O(f)+O(g)=O(\max(f,g))$

令
$$\left.\begin{aligned} C_3 &= \max\{C_1, C_2\} \\ N_3 &= \max\{N_1, N_2\} \\ h(N) &= \max\{f, g\} \end{aligned}\right\} \quad \begin{aligned} &\forall N > N_3 \\ &F(N) \le C_1 f(N) \le C_1 h(N) \le C_3 h(N) \end{aligned}$$

同理 $G(N) \le C_2 g(N) \le C_2 h(N) \le C_3 h(N)$

所以 $O(f)+O(g)=F(N)+G(N)\le C_3 h(N)+C_3 h(N)$

$$= 2C_3 h(N) = O(h) = O(\max(f,g))$$

证毕！

课堂练习

- 证明性质：

$$O(f)+O(g)=O(f+g)$$

$$O(f)\cdot O(g)=O(f\cdot g)$$

NP完全性理论

重要的问题类型

- 查找/检索问题
  - 在给定的集合中寻找一个给定的值
- 排序问题
  - 按升序（降序）重新排列给定列表中的数据项
- 图问题
  - 图的遍历、最短路径以及有向图的拓扑排序
- 组合问题
  - 寻找一个组合对象（排列或子集）满足一些重要特性
- 几何问题
  - 处理类似于点、线、多面体这样的几何对象

问题的复杂度

当为某一问题设计算法时，我们总是追求最好的复杂度。但是，怎样才能知道已达到最佳呢？我们必需考虑问题的复杂度。

- 问题的复杂度就是任一个解决该问题的算法所必需的运算次数。

例如，任何一个采用比较大小的办法将n个数排序的算法需要至少$\lg n!$次比较才行。那么$\lg n!$就是（基于比较的）排序问题的复杂度。因为没有一个算法可用少于$\lg n!$次比较解决排序问题，$\lg n!$就成了算法复杂度的下界，即任一比较排序算法的复杂度必定为$\Omega(\lg n!) = \Omega(n\lg n)$。

- 所以，如果能证明某个问题至少需要$\Omega(g(n))$运算次数，那么$\Omega(g(n))$就是所有解决该问题的算法的复杂度的下界。

问题的复杂度

- 反之，如果某一算法的复杂度是$O(f(n))$，那么它解决的问题的复杂度不会超过$O(f(n))$。
- 因此，任一算法的复杂度也是其所解决的问题的复杂度的上界。
- 通常在已知的某问题的复杂度下界和该问题最好算法的复杂度之间存在距离，算法研究的任务就是努力寻找更好的下界或更优的上界。

结论：找出问题的复杂度，即找出其算法的下界，是一重要的工作，因为它可以告诉我们是否还有改进当前算法的余地而省去很多徒劳无功的努力。

易解问题与难解问题

- 通常将存在$\boxed{\text{多项式时间}}$算法的问题看作是**易解问题**（Easy Problem）；
  - 排序问题、查找问题、欧拉回路
- 而将需要$\boxed{\text{指数时间}}$算法解决的问题看作是**难解问题**（Hard Problem）。
  - TSP问题、Hanio问题、Hamilton回路问题

问题规模

易解问题与难解问题

- 为什么把多项式时间复杂性作为易解问题和难解问题的分界线？
  - 多项式函数与指数函数的增长率有本质的差别

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

不可解问题

- 不能用计算机求解(不论耗费多少时间)的问题称为**不可解问题**(Unsoluble Problem)
  - **<u>图灵</u>**停机问题(Turing Halting Problem)

P类问题和NP类问题

- 判定问题
- 确定性算法与P类问题
- 非确定性算法与NP类问题

判定问题

- 一个判定问题（Decision Problem）是仅仅要求回答“yes”或“no”的问题

- 判定问题的重要特性——证明比求解易

- 判定问题→语言的识别问题→计算模型

确定性算法与P类问题

- **定义1** 设A是求解问题Π的一个算法，如果在算法的整个执行过程中，每一步只有一个确定的选择，则称算法A是**确定性(Determinism)算法**

- **定义2** 如果对于某个判定问题Π，存在一个非负整数k，对于输入规模为n的实例，能够以$O(n^k)$的时间运行一个确定性算法，得到yes或no的答案，则该判定问题Π是一个**P类(Polynomial)问题**

所有易解问题都是P类问题

非确定性算法与NP类问题

- **定义3** 设A是求解问题Π的一个算法，如果算法A以如下猜测并验证的方式工作，就称算法A是**非确定性(Nondeterminism)算法**
  - **猜测阶段：**在这个阶段，对问题的输入实例产生一个任意字符串y，在算法的每一次运行时，串y的值可能不同，因此，猜测以一种非确定的形式工作；
  - **验证阶段：**在这个阶段，用**一个确定性算法**验证：
    - 检查在猜测阶段产生的串y是否是合适的形式，如果不是，则算法停下来并得到no；
    - 如果串y是合适的形式，则验证它是否是问题的解，如果是，则算法停下来并得到yes，否则算法停下来并得到no。

非确定性算法与NP类问题

- **定义4** 如果对于某个判定问题Π，存在一个非负整数k，对于输入规模为n的实例，能够以$O(n^k)$的时间运行一个非确定性算法，得到yes或no的答案，则该问题是一个**NP类(Nondeterministic Polynomial)问题**

关键： 存在一个确定性算法，能够以多项式时间来检查和验证猜测阶段所产生的答案。

例如： NP类问题——Hamilton问题

Hanoi塔问题不是NP类问题

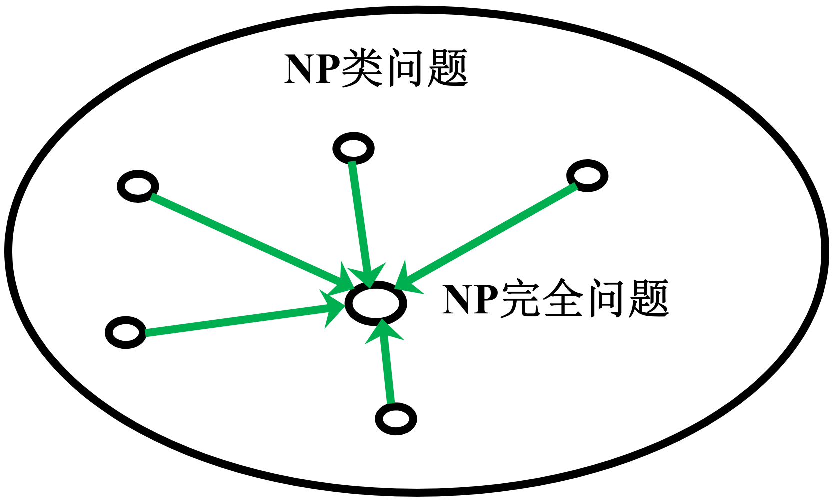
P类和NP类问题的主要差别

- P类问题可以用多项式时间的**确定性算法**来进行判定或求解；
- NP类问题可用多项式时间的**非确定性算法**来进行判定或求解。

$$P \subseteq NP$$

NP完全问题

- **定义5** 令Π是一个判定问题，如果问题Π属于NP类问题，并且对NP类问题中的每一个问题Π'，都有$\Pi' \propto_p \Pi$，则称问题Π是一个**NP完全问题(NP Complete Problem)**，有时把NP完全问题记为**NPC**

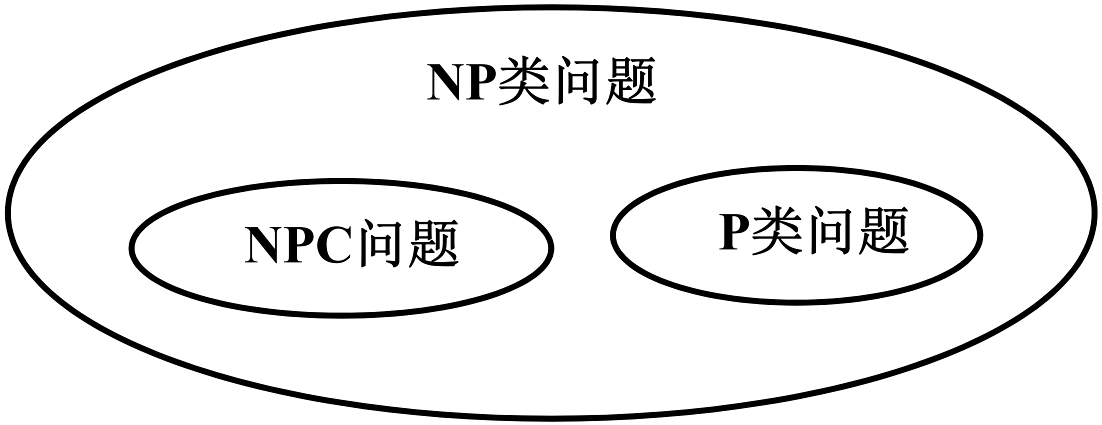


一些基本的NP完全问题

- SAT问题(Boolean Satisfiability Problem)
- 最大团问题(Maximum Clique Problem)
- 图着色问题(Graph Coloring Problem)
- 哈密顿回路问题(Hamiltonian Cycle Problem)
- TSP问题(Traveling Salesman Problem)
- 顶点覆盖问题(Vertex Cover Problem)
- 最长路径问题(Longest Path Problem)
- 子集和问题(Sum of Subset Problem)

三者关系

- **目前人们猜测P≠NP，则P类问题、NP类问题、NP完全问题之间的关系如下：**

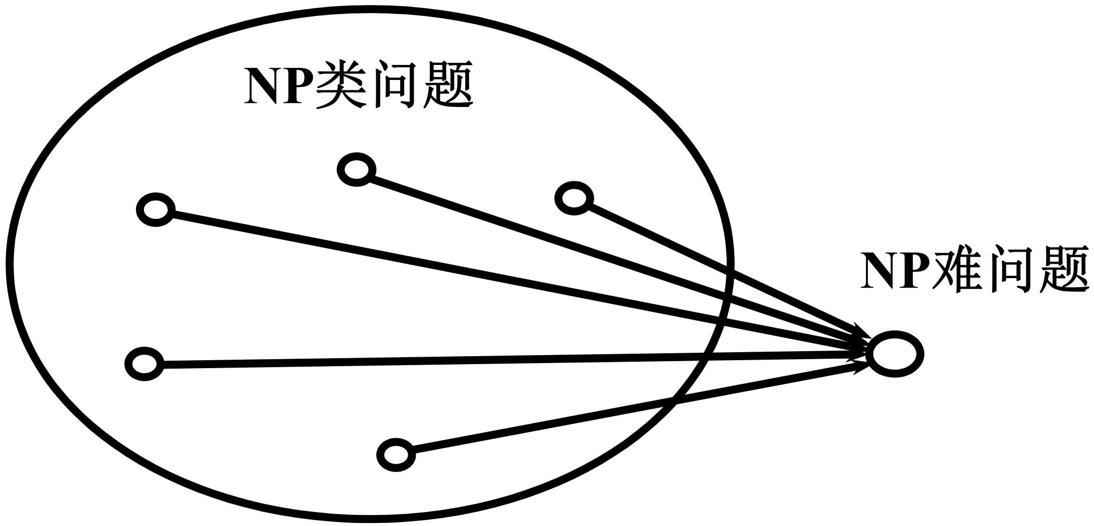


千僖难题

1. P类问题≠NP类问题猜想
2. 霍奇(Hodge)猜想
3. 庞加莱(Poincare)猜想
4. 黎曼(Riemann)假设
5. 杨-米尔斯(Yang-Mills)存在性和质量缺口
6. 纳维叶-斯托克斯(Navier-Stokes)方程的存在性与光滑性
7. 贝赫(Birch)和斯维讷通-戴尔(Swinnerton-Dyer)猜想

NP难问题

- **定义6**　令Π是一个判定问题，如果对于NP类问题中的每一个问题Π'，都有$\Pi' \propto_p \Pi$，则称判定问题Π是一个**NP难问题**。



NPC和NP难问题的区别

- 如果，Π是NPC问题，Π'是NP难问题，那么，他们之间的差别在于Π必定是NP类问题，而Π'不一定在NP类问题中。

一般而言，若判定问题属于NP完全问题，则相应的最优化问题属于NP难问题。

例如，判定图G = (V, E) 中是否存在哈密顿回路是NP完全问题，而求哈密顿回路中最短路径的TSP问题则是NP难问题

课后作业

- 上传群里
  - LaTeX版本：hw01.tex
- 提交要求：
  - 一份答案电子版（分析题+实现题算法描述）
    - 命名：学号-hw01.tex
  - 一份实现题的可执行源代码（C++）
  - 打包成一个文件：学号-姓名-次数.zip
  - 电子邮件地址：xlinguo@seu.edu.cn
- 截止时间：
  - 一周内完成（如：周二课堂布置，下周一前提交；周四课堂布置，下周三前提交）

LaTex教程

- TEX是斯坦福大学的教授Donald E. Knuth开发的一个功能强大的幕后排版系统。当时在撰写名为《The Art of Computer Programming》的书，由于出版商把书中的数学式子排版得很难看，决定推迟出版，自行研发一套排版系统进行排版。这个系统就是TEX系统。
- LaTeX：
  - TEX是很低阶的排版语言，对于绝大多数人来说，学起来会很吃力，而且排版工作也会变得相当繁复，难以被更多人使用，效率也不是很高。所以，一些经常用到的功能，如果我们事先定义好，到要用的时候只引用一小段代码就可以实现一个相对复杂的功能，那不仅提高了排版效率，而且版面也会清晰很多。这种事先定义好的功能，叫做宏集(macro)。
  - LaTeX是TEX众多宏集之一，是由Leslie Lamport编写的。

LaTex教程

- **CTeX：**
  - **CTeX是利用TEX排版系统中文套装的简称**
  - **CTeX中文套装在MiKTeX的基础上增加了对中文的完整支持，集成了编辑器WinEdt和PostScript处理软件 Ghostscript 和 GSview 等主要工具**
  - **支持CCT和CJK两种中文TeX处理方式**
  - **http://www.ctex.org**

LaTex教程

- **Texmaker**
  - 是一款跨平台的开源LaTeX编辑器，界面干净并集成有PDF阅读器