第10章

如何设计算法

为给定的应用问题设计正确的算法是一项非常重要的创造性行为——你紧抓着一个问题, 想从浩瀚星河中拽出它的解答. 在算法设计中你的选择空间极其广阔, 会给你留下充分的自由去让你绞尽脑汁.

本书的目标是为了让你成为一个更好的算法设计师. 全书分为两卷: 卷I所展示的技术提供了能支撑所有组合算法的基本思想; 卷II中的问题便览将帮助你对应用问题建模, 并会告诉你相关问题的现有研究成果. 然而, 要成为一个优秀的算法设计师光有书本知识可不行, 你得具备一种重要的思维模式——也就是正确的问题求解方法. 光靠书本很难教会你这种思维方式, 但我们将努力让你学会, 因为它对于任何成功的算法设计师来说都是不可或缺的.

算法设计(或任何其他的问题求解任务)的关键是通过向自己提问来引导思维不断深入展开. *我们这样做会怎么样? 我们那样做又会怎么样?* 如果你被某个问题卡住, 最好的办法就是转向下一个问题. 在任何团队的头脑风暴会议中, 屋子里最有用的人就是那个一直在问"*我们为什么不能用这种方法来做*"的人, 而不是随后告诉大家否决原因的人, 因为持续发问的她最终总会灵光突闪而找到一个无法被枪毙的方案.

为了实现这个目标, 我们提供了一系列的问题来引导你合理地寻找待解决任务的正确算法. 要有效地利用它, 你不仅得提出问题, 还要回答问题. 关键在于去仔细地思考这些问题的答案, 而方法则是以日志形式记录思考的全过程. "*我能这样做吗*?"的正确答案永远不会是"不能", 而是"不能, 因为……". 通过阐明为什么有些想法不能奏效的理由, 你可以弄清楚是不是自己不想投入精力认真思考, 而你可能有意无意地在掩饰这点. 有时你找来找去也无法对某些疑问作出令人信服的解释, 最后你会惊奇地发现其原因在于你所给的结论根本就是错的.

*战略和战术*之间的区别在任何设计过程中都是非常重要的, 请务必时刻注意. 战略代表的是如何探求整体图景——能让我们围绕框架逐步构建方案从而达成目标. 战术一般用于赢得那些要完成目标而非打不可的小型战役. 在问题求解中, 不断检查你是否在正确的层级上思考是非常重要的. 如果你对于自己要如何攻破你的问题没有整体战略, 那么担心战术就没意义了. 一个战略问题的例子是: "我能不能把应用问题建模为一个图算法问题?" 而一个战术问题将会是, "能用邻接表或邻接矩阵数据结构表示我的图吗?" 当然, 这种战术决策对于解决方案的最终质量来说也很关键, 但是这些决策只有放在成功的战略下才能被正确地评价.

好多人面对设计问题时都会思路停滞, 这样的人太多了. 在读到或听到问题后, 他们坐下来, 发现自己*不知道下一步该做什么*. 你得避免这种厄运. 请遵循下面所提供的一系列问题的指示, 并紧跟算法问题便览中相关章节所给出的思路. 我们将*告诉*你下一步该做什么!

显而易见, 你在算法设计技术(如动态规划、图算法、难解性和数据结构)上的经验越多, 那么你使用本节的问题列表最后获得成功的可能性就越大. 本书卷I的写作思路就是要强化你的这种技术背景. 然而, 不论你的技艺有多强, 把这些问题全过一遍都会大有裨益. 问题列表中越靠前的条目越重要, 它们着重于让你能对所面临的问题有一个详尽细致的了解, 而这些不需要过多的算法专业技能.

这组问题受到[Wol79]中一段文字的启发, 那本书名叫《真本事》(*The Right Stuff*),[1] 它是一本讲述太空计划的好书. 该书提到了飞机坠毁前试飞员的无线电传输. 人们原以为试飞员会异常惊恐, 并猜想地面控制中心会听到飞行员"啊——"的尖叫, 最后只会以撞到山上的声音而告终. 事实却完全不是那样, 这些飞行员会逐条进行他们能做的各项操作. *我已经试过襟翼了. 我已经检查过引擎了. 两个机翼还在. 我已经复位了*$\cdots\cdots$他们确实有"真本事". 正因为如此, 有时他们试尽了各种方法最终避免了撞山事故.

我希望本书和以下列表能为你提供算法设计师的"真本事". 我也希望它能让你在前行的道路中不会撞到任何山上.

1. 我确实理解了这个问题吗?
   (a) 输入究竟由什么构成?
   (b) 我们到底期望有什么结果或输出?
   (c) 我能创建一个可以手工求解的较小输入算例吗? 当我尝试解决它时会出现什么问题呢?
   (d) 总能找到最佳答案对于我的应用问题来说重要性有多高? 对于和最优解接近的答案, 我会不会满意呢?
   (e) 问题的典型算例有多大规模? 我要处理的是十个元素, 一千个元素, 还是一百万个元素呢?
   (f) 我的应用问题对速度的要求有多高? 这个问题需要在一秒钟, 一分钟, 一小时, 还是一天内解决呢?
   (g) 我能投入多少时间和精力实现算法呢? 我是只有一天时限只够写完简单算法的实现, 还是有充足时间可以实现多种方法并实验对比最后找出哪个是最好的呢?
   (h) 我要解决的是一个数值问题, 图算法问题, 几何问题, 字符串问题, 还是集合问题呢? 哪种建模会最容易实现呢?
2. 我能否为该问题找到一个简单算法或启发式方法?
   (a) 如果用蛮力法在所有子集或所有安排中搜出最好的一个, 这样能*正确地*解决我的问题吗?
      i. 如果可行, 我们为什么会肯定这个算法总能给出正确的答案?
      ii. 一旦解创建完毕, 我该如何衡量解的质量?
      iii. 这个简单又缓慢的解决方案其运行时间是多项式量级还是指数量级? 我的问题规模是不是比较小, 因此用蛮力法求解已足够了呢?
      iv. 问题的定义是否清晰明确, 因此它确实*存在*一个正确解呢?

[1] 译者注: 根据这本书所改编的同名电影也非常精彩, 通常译为《太空先锋》. 这里我们考虑到上下文将书名译作《真本事》.

(b) 如果不断地采用某个简单准则(比如最大元素优先、最小元素优先和随机元素优先)进行处理, 能否解决我的问题?
   i. 如果可行, 这种启发式方法对于哪些类型的输入效果较好? 它们与该应用问题中会出现的数据形态是一致的吗?
   ii. 这种启发式方法对于哪些类型的输入效果较差? 如果无法找到这种算例, 我能证明该方法总会有很好的效果吗?
   iii. 我的启发式方法多久能得出答案? 它有没有简单的程序实现?

3. 我的问题是否在卷II中的算法问题便览中?

(a) 这个问题的研究现状如何? 有没有现成的程序实现可让我使用?
(b) 对于这个问题我找对地方了吗? 我是不是浏览过问题便览中的所有算法示意图了呢? 我是否在书后索引部分将所有可能的关键词都查遍了?
(c) 网上是否有相关的资源? 我在Google网页或Google学术中搜索过吗? 我查过本书配套网页(`http://www.cs.sunysb.edu/~algorith`)吗?

4. 这个问题有没有特例是我能解决的呢?

(a) 我能否在忽略某些输入参数时高效地解决问题?
(b) 当我将某些输入参数设为平凡值(如0或1), 问题会不会变得容易解决?
(c) 可否将问题简化到我能够高效求解的情况?
(d) 为什么这个处理特例的算法不能推广为适应更多类型输入的算法呢?
(e) 我的问题在便览中是不是某个更一般问题的一个特例?

5. 哪个标准算法设计范式与我的问题最相关?

(a) 是否存在一组数据项能通过尺寸或其中某个键就能完成排序? 这种序关系会让寻找答案的过程更容易吗?
(b) 有没有一种方法(也许是二分查找)能把问题分为两个较小的问题? 将所有元素按照大/小或左/右划分又会怎样? 这是否能引出一个分治算法?
(c) 输入对象或所求的解是否自然而然地就带有从左到右的次序, 例如字符串中的字符, 置换中的元素, 或树的叶子? 我能用动态规划深入探讨这种次序吗?
(d) 是否存在某些重复进行的操作, 例如搜索或是寻找最大/最小元素? 我能用某个数据结构来加速这些查询吗? 字典/散列表或堆/优先级队列可行吗?
(e) 我能用随机抽样选择下一个待选对象吗? 建立许多随机排布并选出最好的一个, 这种方法怎么样呢? 我能用像模拟退火之类的方法通过有向随机性(directed randomness)将排布聚到最优解上吗?
(f) 我的问题能用线性规划来建模吗? 整数规划行不行?
(g) 我的问题看起来像是可满足性问题, 旅行商问题或其他NP完全问题吗? 是不是因为该问题是NP完全问题所以没有高效算法呢? 这个问题是否列在Garey和Johnson那本[GJ79]书后的问题中呢?

6. 我还没能解决它?

(a) 我愿意花钱雇专家来告诉我该怎么做吗? 要是愿意的话, 请查一下19.4节中所提到的专业咨询服务.

(b) 为什么不回到列表开头, 再去研究一遍这些问题呢? 在我这次对列表作答的过程中, 是否有答案发生变化呢?

问题求解虽然不是一门科学, 但它却是艺术和技能的交融. 这是一项最值得开发的技艺. 在讨论问题求解的书中, 我最喜欢的依然是Pólya的那本《怎样解题》(*How to Solve It*)[P57]. 它的特色在于对各式问题求解技巧都给出了极其精当的阐述, 读来令人不忍释卷, 我建议你去认真研读此书, 无论是你面临问题之时, 还是你解决问题之后.