浅谈信息学竞赛教学中非程序设计题目的应用

李嘉图

清华大学交叉信息研究院

lijt19@mails.tsinghua.edu.cn

摘要：程序设计题目是目前信息学竞赛主要的教学、训练方式。本文从程序设计题目在信息学竞赛教学中的不足出发，通过分析传统的教学模式，探讨了非程序设计题目在信息学竞赛教学中的应用价值。此外，通过引入阶段性模型提出了面向教学目标的非程序设计题目设计方法，总结出一些有效的非程序设计题目形式。本文认为，非程序设计题目作为传统程序设计题目训练的补充，可以在缺乏教师指导的情况下提升整体教学效果，是一种值得推广的教学设计方法。

关键词：信息学竞赛教育，教学方法，教学研究，非程序设计题目

1 引言

1.1 背景

经过三十余年的发展，信息学竞赛（Olympiad in Informatics）已成为受到高校和社会认可 (A.1, A.2) 的、基础教育阶段高水平计算机科学人才培养的重要方式，基于信息学竞赛的计算机科学基础教育培养模式也在世界范围得到了广泛的应用 [8, 18]。

在我国的信息学竞赛社区，程序设计题目是教学、比赛中最常用的题目形式 (A.3)。围绕着程序设计题目，信息学竞赛社区建立起了一套教学、训练和比赛的生态环境，包括教学书籍、命题组织和支撑工具链 (A.4) 。相比之下，非程序设计题目并没有受到国内信息学竞赛社区的重视 (A.5)。

尽管程序设计题目有着许多优势，在教学中完全使用程序设计题目也有其不足之处，其中许多问题并没有被我国信息学竞赛社区所广泛认识。本文的主要观点有二：

1. 程序设计题目有着一些固有的不足，仅仅使用程序设计题目进行训练，存在两个直接问题和一个间接问题。
2. 经过合理的设计，非程序设计题目可以作为“杂题选讲”“算法讲解”等方式的替代，较好地解决两个直接问题和一个间接问题。

基于这些观点和教学实践，本文希望说明程序设计题目的不足之处，并给出在教学中设计和使用非程序设计题目的一些方法。对于教学工作者而言，这些方法可以方便的嵌入的现有的教学体系中，为课程和教学方法设计提供新的基本工具。

1.2 相关工作

非程序设计题目：非程序设计题目即程序设计题目之外的分析、证明、算法设计题目。在国际信息学竞赛官方期刊 Olympiad in Informatics 上，非程序设计题目一直是相当受关注的研究方向。华沙大学的 M. Kubica 教授和 J. Radoszewski 博士指出，非程序设计题目有利于那些不了解编程和算法的学生提升算法思维，作为学生学习计算机科学的有益准备[19]。事实上，波兰信息学竞赛（POI）一直有着重数学分析、轻代码实现的特点，其题目也是我国选手提升思维能力的常见选择 (A.6)。

David Ginat 博士也强调了书面练习（Pencil-Paper Tasks）的重要作用。他认为，好的书面题目应当要运用抽象（Abstraction）、兼具启发性和逻辑推理（Heuristics and Reasoning）、具有创意（Creativity）并能够引出算法（Declarative Perspective）[15]。其他关于非程序设计题目的讨论还包括 [10, 21]。

本文中所讨论的非程序设计题目与以上文献中提到的各类题目有相似的形式，但设计目的有着很大的不同。[15, 19] 中题目的主要设计目的是推广算法教育，提升青少年的算法思维，培养青少年对计算机科学的兴趣；本文中题目的主要设计目的是作为信息学竞赛训练中程序设计题目的补充，在缺乏教师指导的情况下提升整体教学效果。“面向信息学竞赛”和“面向教学”是本文中所讨论的非程序设计题目的两个特点。因此，本文的工作集中在根据新的设计目的，研究非程序设计题目的优势和设计思路。

大学中的算法和数据结构课程：大学计算机科学教育中，和信息学竞赛联系最为紧密的便是算法和数据结构课程。清华大学邓俊辉教授的“数据结构”课程是国内最好的同类课程之一 (A.7)。课程的编程实验（PA）主要由信息学竞赛和大学生程序设计竞赛题目组成，其配套书籍 [6] 中则包含了大量的算法设计、分析书面练习。

麻省理工大学的算法导论课程在国际上享有很高的声誉。从其教材 [11] 中可以看出，课后习题包含了关键定理证明、算法设计、算法分析等多种形式。每一章结尾的思考题则是经过精心设计的综合性题目 (A.8)，其设计了一步一步的求解过程，带领学生思考一个相对困难且有启发性的问题。

大学中的算法和数据结构课程具有相当丰富的教学和训练方式。课堂的算法讲解，各类程序设计、算法分析题目的训练，可以从不同的角度提升学生对算法设计和分析的理解。但由于专业教师的的缺乏，这些教学模式在一段时间内还无法在信息学竞赛教学中得到广泛应用。本文认为，非程序设计题目值得推广的重要原因，便是其可以作为缺乏教师指导时“算法讲解”和“杂题选讲”等教学模式的替代。这些内容的讨论在第 3 节中展开。

2 程序设计题目

2.1 使用程序设计题目训练的“流水线”

程序设计题目的题目描述是能够精确描述待求解任务的一段自然语言，其解答是一段解决这一任务的程序。那么提升学生解决问题的能力，便相当于让学生形成某种思维方式，

使得根据一段题目描述，学生便能用这种思维方式以尽量短的时间、尽量高的概率给出正确的、解决问题的代码。

这种思维方式应当怎样建立呢？常见的方式可以分为两种。其一是学习已有的题目和解答、或是其他能够启发解答的内容，听课、读教材、读他人的解题报告等都属于这种方式；其二是通过独立完成题目进行学习。其中，前者是从已有的证明、解答、程序中学习解决问题的方法，后者则是通过尝试解决问题逐步建立良好的思维方式。由于后者没有标准解答可以参考，想知道一次尝试是否正确，要么需要通过黑盒测试（Black-Box Test）——即检查编写出的程序能否通过一系列已知答案的数据——来检查这次尝试的正确性，要么需要给出程序正确的证明。从实践上来说，黑盒测试对教师和学生来说都是更加方便的方法。

使用程序设计题目进行训练的“流水线”可以这样描述：出题人为题目编写题目描述和标准程序，并利用标准程序生成一系列数据；教师选择题目；学生在阅读题目描述后，尝试给出解答，并通过黑盒测试来确定程序的正确性；教师则根据黑盒测试的结果了解学生的情况，以设计更加有效的训练计划。作为对训练的补充，有经验的教师可能会要求学生通过编写解题报告、讲述思路等方式整理思路，以对学生做更加针对性的指导。

2.2 优势和不足

使用程序设计题目进行训练是一种行之有效的训练方法，其优势是多方面的：其一，程序设计题目综合性强，可以全面训练学生分析问题、解决问题、编写程序的能力；其二，教师在训练过程中不需要对题目做过多的讲解，也不需要亲自判断学生解答的正确性，这一方面使得教师从繁重的体力劳动中解脱出来，另一方面也为教育资源相对落后的地区进行信息学竞赛训练提供了可能。

那么，程序设计题目有那些不足呢？经过教学实践和调研，笔者认为程序设计题目存在两个直接问题和一个间接问题，两个直接问题是过大的思维跨度和较差的灵活性，一个间接问题是无法训练那些不直接和算法设计相关的知识和能力。

先讨论两个直接问题。所谓过大的思维跨度，是说从题目描述到最终编写代码之间的思维跨度往往非常大。学生往往需要先根据题目描述寻找有用的组合性质，综合利用已经掌握的算法和算法设计技术不断尝试，最终设计出正确的算法并完成代码编写 (A.9)。整个过程中任何一步遇到困难，便难以解决整个问题。对于初学者而言，过大的思维跨度带来了相当大的入门难度，可能打击学生的自信心，使学生丧失对程序设计乃至计算机科学的兴趣；对于高水平选手来说，思维过程中任何一步的困难都会导致解题过程的“卡壳”。如果学生迷失在思维的“迷宫”之中，得不到教师的反馈和提示，便难以从题目中学习到算法设计的方法和技巧（如图 1）。

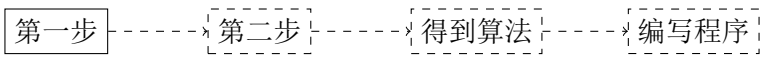

图 1: 如果在第一步遇到问题，之后的部分便丧失了训练效果

另一方面，由于程序设计题目所描述的问题必须是一个程序设计问题，要想利用程序

设计题目进行教学和考察，就要把希望涉及的知识、技巧附着在一个程序设计问题上，而这一点即使对经验丰富的出题人来说也非常困难，在有时甚至是理论上不可能的 (A.10)。这就是所谓程序设计题目较差的*灵活性*。程序设计题目较差的灵活性体现在以下三点。

1. 形式不灵活：程序设计题目仅能具有程序设计问题的形式；
2. 内容不灵活：由于形式不灵活，程序设计题目难以考察算法设计之外的学科能力；
3. 使用不灵活：举例而言，即使仅仅想要介绍解题中的关键一步，学生也需要完成整个题目，从而将训练时间花费在了一些细枝末节上。

程序设计题目的间接问题事实上是较差的灵活性带来的必然结果：程序设计题目较差的灵活性，使得选手无法经由这种方式学习许多计算机科学中非常重要的知识——即使局限在算法和数据结构这一子领域 (A.11)。毫无疑问，信息学竞赛教育应当尽可能使得学生具有全面的学科能力，而非仅仅成为解决一类特定问题的优秀选手。

3 非程序设计题目：分析和应用

3.1 有经验教师的解决方案

在计算机科学中，程序设计往往被认为是一种艺术而非技术。Donald E. Knuth 用“计算机程序设计的艺术”（The Art of Computer Programming）来给他的恢弘著作命名便体现了这一观点。在传统的、使用程序设计题目的训练模式中，教师的任务主要是提供题目和检验成果，对学生“艺术创作”的过程则很难提供帮助；缺少教师指导的学生由于程序设计题目过大的思维跨度，很难领会“艺术”中的门道，因此学习起来困难重重。

有经验的教师往往会使用多种“非程序设计的方法”来解决程序设计题目的不足，其中一种常用的方式称为“*杂题选讲*”的教学模式。教师综合学生的学习情况和遇到的问题，选择一些题目进行讲解。在讲解的过程中，教师并不仅仅是简单地给出解答，而是引导学生主动思考，并在适当的时候加以提示。

如果仔细思考“杂题选讲”模式有效的原因，不难发现其成功之处在于为程序设计题目的两个直接问题和一个间接问题提供了解决方案。由于教师的教学可以穿插于学生思考和解决问题的任何阶段，其*灵活性*相比程序设计题目有了很大的提升，同时教师恰到好处的提示可以为学生搭建“思维的桥梁”，降低困难问题的*思维跨度*。另外，一个具有全面学科能力的教师能够在讲解中融汇计算机科学重要的问题、技术和思想方法，从而提供*全面的学科训练*。

与“杂题选讲”相对，“*算法讲解*”是另一种常用的教学模式。类似数学教学中常用的“定理、证明、注记”的模式，“算法讲解”常常依照“问题、算法、分析”的模式进行讲解 (A.13)。通过课堂讲授介绍这些内容的原因有二：其一，这些问题为算法设计提供了重要的基本工具，承载着典范的算法设计思想，应当通过课堂讲授引起学生足够的重视；其二，由于对这些问题的研究已经相当深入，设计过程中的许多思想已经可以用语言、文字和多媒体工具很好的展示，因而满足通过课堂向学生讲授的基本条件。

那么，这些教学模式的问题存在什么问题呢？其问题不在模式本身，而在于当前专业教师的严重缺乏。以“杂题选讲”模式为例，教师一方面要对所讲解内容的深刻理解和全面认识，另一方面要对学生的思考情况做推测和判断，根据学生的情况“对症下药”。专业教师的数量与教学需求存在着严重的不匹配，因而这些模式在当前的信息学竞赛教学中仍然是相当奢侈的选择 (A.12)。

3.2 阶段性模型

从教学系统设计者的视角来看，要解决专业教师的缺乏问题，一个自然的思路就是在教学效果和对教师的要求之间做权衡，而非程序设计题目正是对教师指导的一种有效近似。为了讨论非程序设计题目的设计和使用，我们首先提出一个关于“程序设计艺术”的模型。

阶段性模型．*在解决程序设计问题的过程中，思维的过程应当具有良好的阶段性，其中第一阶段是找到用“某种中间形式”表示的解决问题的方法，第二阶段是根据“某种中间形式”编写对应的代码。两个阶段之间具有很低的相关性。*

在阶段性模型中，“某种中间形式”是一个相比程序代码、伪代码更加高层次的算法表示方式，这种表示由程序设计语言的基本操作、算法和数据结构以及种类繁多的程序设计技巧组合而成，同时也可能包含了对问题性质的数学推导和对算法正确性启发式的证明。因此，阶段性模型认为在问题到代码过大的思维跨度之间，人们总是经由某个抽象层作为桥梁。这种抽象层提供了一种有效的思考方法，使我们可以在不关心代码实现的前提下有效的寻找解决问题的算法。

尽管还没有直接的实验结论来证明阶段性模型的正确性，但有证据表明，这种模型是合理的。其一，如果在思维过程中没有一个明确的“中间表示”作为问题和代码实现之间的抽象层，那么解决一些长达几百行、上千行的复杂题目所需要的思维复杂性是难以想象的。其二，抽象（Abstraction）是计算机科学中封装细节、降低整体复杂性的重要思想方法 [9, 12, 13]。程序设计语言中的子程序、类和对象，算法与数据结构，乃至操作系统、文件系统、数据库系统等等都是通过抽象为程序员提供了高层次的“中间表示”，以将程序设计和具体的代码实现过程分隔开，降低对程序员的整体思维要求。算法设计是计算机科学的中心问题，在算法设计的过程中存在这样的“中间表示”也是相当合理的。

3.3 从教学目的出发设计题目

阶段性模型为已有的教学方法提供了一个统一的框架（如图 2）。程序设计题目是从问题到代码实现“端到端（End-to-End）”的训练方式，综合性强、思维跨度大；传统的算法讲解关注算法和数据结构的原理、实现及分析，其目的是丰富解决问题的“工具库”，主要关注阶段性模型中的第二阶段；杂题选讲的目的是带领学生体会算法设计的方法和技巧，主要关注阶段性模型中的第一阶段。从总体上说，程序设计题目训练无需教师的参与，算法讲解和杂题选讲则依赖教师的教学艺术与智慧。

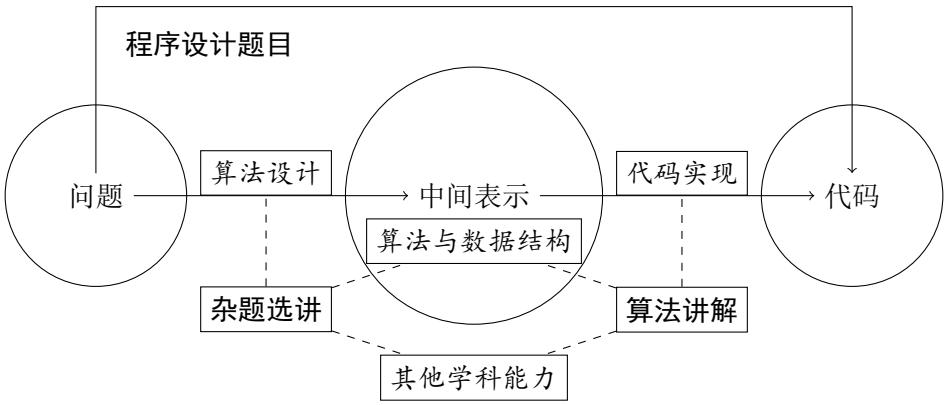

图 2: 阶段性模型与教学设计

根据 3.1 中的讨论，由于专业教师的缺乏，非程序设计题目应当起到的作用就是作为图 2 中所描述的杂题选讲和算法讲解的有效替代。

非程序设计题目的一个重要的特点是其灵活性。非程序设计题目的灵活性首先体现在*形式的灵活性*。由于没有了将知识点附着在程序设计问题上的限制，非程序设计题目可能有着灵活多变的形式，如证明题、计算题、算法设计题、算法分析题等等。题目形式的灵活带来的直接影响便是*内容的灵活性*：题目的内容不再局限于算法设计和代码实现，还可以延伸至算法分析、数学工具的使用等等。形式和内容的灵活性带来了*用途的灵活性*：非程序设计题目可以增进学生对数学工具、算法和数据结构的理解和认识，引导学生整理算法设计的思路，也可以启发学生思考一些难以用语言解释的解决问题的思想方法。

另一方面，非程序设计题目灵活性带来了设计和使用上的困难。与“端到端”的程序设计题目相比，非程序设计题目只有被良好的安排在整体教学之中才能发挥最大的作用，但具体的安排方式所能参考的实践经验和理论依据还相当的有限。对教研人员来说，一种简单的设计方式是“从教学目的出发”设计相应的题目。在阶段性模型中（如图 2），教学目的就可以大致划分为三种类型：培养算法设计能力、训练代码实现能力以及培养计算机科学的其他学科能力。

1. 算法设计类：算法设计类题目的主要作用是带领学生思考，引导学生解决问题，使学生逐步形成算法设计的思维方式。因此，将一些具有代表性的程序设计题目、有用的算法或技巧拆分成有递进关系的若干小问，或从综合性题目中截取关键的逻辑步骤都是有效的形式。为了使学生能够更多的表达思考问题的过程，拆分成的若干小问通常为证明关键结论或设计子问题算法的形式，这种形式可以称为检查点式（参见 3.4）。(B.1) 给出了一个这样的例子。

值得说明的是，教学实验可以为算法设计类题目的设计提供有效的帮助。由于算法设计类题目是作为杂题选讲模式的替代，教研人员可以通过在真实的杂题选讲过程中记录学生思考的过程，作为拆分问题和给出文字提示的参考依据。类似的教学实验也可以帮助问题的选择。学生无从下手、多次提示仍没有进展的题目适合作为讲授题目或例题；学生不断有思路，但需要引导才能连接起来的题目适合拆分成递进小问作为训练题目；大多数学生能够较快解决的问题则适合整体使用。

2. 代码实现类：代码实现类题目的主要作用是让学生熟悉算法和数据结构的具体实现，并排除一些常见的问题。这类题目通常在代码难度较大、需要一些好的代码设计方法时作为编程练习的补充。代码实现类的非程序设计题目应当具有代表性和可推广性，并与教学和程序设计训练相互补充：非程序设计题目关注方法上、思想上的问题，细节的、繁杂的问题则利用编程训练来完成。(B.2) 中的例子通过题目方式，启发学生思考一种常见错误的原因，以在代码实现中避免这种错误。

与算法设计类题目一样，学生在代码实现上的困难也可以通过教学实验进行收集。

3. 其他学科能力类：其他类型的题目则更加灵活多样。作为对信息学竞赛教学内容的补充，这类题目找准寻找教学内容和扩展内容的交集，通过带有提示的题目引导学生思考信息学竞赛之外的知识。这类题目通常供学有余力的学生拓宽视野，可以通过给出资料引导学生自学的方式，锻炼学生的文献检索和阅读能力。一些常见的延伸内容包括线性代数、基础抽象代数、组合数学、微积分等数学知识，以及计算机系统、程序设计语言、计算理论、信息论和通信理论等计算机科学知识。(B.3) 给出了一个证明时间复杂度下界的例子。

在国内的信息学竞赛社区中，许多进入大学的退役选手在学习了更广泛的知识后，将这些知识与信息学竞赛的形式相结合，在信息学竞赛中引入了大量新的知识点。其中，一些知识点已经被选手和命题人广泛接受，另一些则由于难度过大、无法与程序设计题目结合等原因，并没有受到出题人广泛的关注 (A.14)。其中，第二类知识点可以使用非程序设计题目或其他方式向学生讲解，以培养学生全面的学科能力。

3.4 题目形式

非程序设计题目的题目形式决定了引导学生进行思考的方式。因此，应当根据教学目标和具体问题选择适当的形式设计非程序设计题目。常见的题目形式例如检查点式、从特殊到一般的递进式和并列式。

1. 检查点式：检查点式即将解决问题的完整思路拆成若干个小题，每一小题就像一个“检查点”，以小题的形式引导学生逐步解决问题（如图 3）。检查点式的优点在于降低复杂题目的思维跨度，让学生直接从解决问题的例子学习解题方法；其不足在于过多的小题可能会割裂思路的完整性，让学生难以把握解决问题的整体思路。

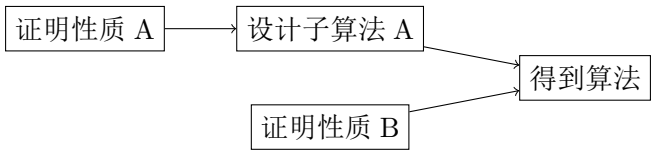

图 3: 检查点式

检查点式的题目大量出现在《算法导论》[11] 的思考题中，如 12.4 不同二叉树的数目、16.5 离线缓存、23.3 瓶颈生成树 以及 35.4 最大匹配。事实上，这种逐层递进的题目在数学教科书中非常常见，例如 [14] 和 [16]。

2. 从特殊到一般的递进式：从特殊到一般是一种常用的解题技巧，从特殊到一般的递进式

题目便是希望引导学生体会这种思考方式。举例而言，为了解决一个有关树的算法设计问题，我们可以将原问题在树的若干特例（如序列、星形图、随机树）上的版本作为子问题，启发学生从特例出发，归纳解决问题的一般方法（如图 4）。

值得说明的是，从特殊到一般的递进式设计也大量应用在传统的程序设计题目中：由于特殊情况下的问题较为简单，特殊情况往往作为整个题目的一个子任务，具有提示解法、增强题目区分度等作用。

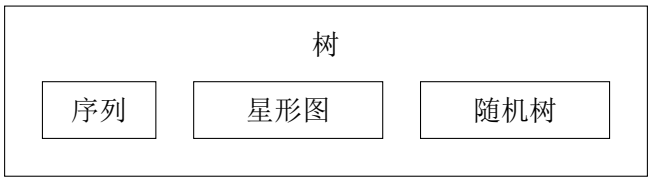

图 4: 从特殊到一般的递进式

3. 并列式：并列多个同类型题目有利于增强学生对一类方法的熟练度，这对一些数学工具和算法的学习来说是相当有效的。并列式可以和前两种形式进行结合，如先并列多个特殊情况，在将这些特殊情况的结果推广到一般。

3.5 非程序设计题目的额外优势

除作为杂题选讲和算法讲解的有益补充外，非程序设计题目还存在一些额外的优势，这些优势使其在多种教学情景下均有应用价值。

1. 学生无需使用计算机：中学阶段学生对计算机的使用并不方便。一方面，许多学校禁止学生在在校期间使用智能电子设备。即使不考虑这一点，拥有笔记本电脑的学生在现阶段也不占多数。另一方面，对教育资源相对落后的地区而言，信息学竞赛往往与学校信息技术教学共用机房，因此上机时间就受到了学校教学的影响。在这种情况下，非程序设计题目可以作为无法使用计算机时的辅助训练方法。

2. 来源多样：非程序设计题目的来源很多。教研工作者可以从大学课程习题或经典题目出发，根据对应的教学目的进行改编，设计新的非程序设计题目。

3. 易于修改：修改非程序设计题目仅需要修改题目描述和解答，相比之下，修改程序设计题目则要修改题面、标准程序和测试数据，整个过程需要较大的人力和时间成本。因此，非程序设计题目可以根据教学实践反复迭代，使其更好的适应于整个教学体系。

4. 无需评测系统支持：大多数中学并没有能力维护自己的评测系统，因此使用程序设计题目进行日常训练就必须依赖在线评测系统，这可能会引起训练安排上的种种不便。非程序设计题目不需要评测，学生可以在完成题目后，根据解答或讲解进行改正和总结。

5. 更加有效的反馈：比起程序设计题目，非程序设计题目的解答更加完整的展示了学生思考过程的细节，可以向教师提供更加有效的反馈。

6. 给所有学生表达的机会：在教学过程中，有经验的教师会鼓励学生向大家讲解自己的解

法、思路。在表达做法的细节的过程中，学生需要深入思考解决问题的思维路径，进而形成良好的思维习惯。然而，并非所有学生都有机会在课堂上进行表达。非程序设计题目可以给那些在课堂上不太活跃的学生思考和表达的机会，从而提升这部分学生的学习效果。

7. 作为书面工作的训练：对许多信息学竞赛选手而言，面对电脑编程的时间往往远远大于进行计算、证明等“书面工作”的时间，很多保送大学的国家集训队员对大学大量的“书面工作”感到不适应。在学生未来的研究、工作中，编写程序仅仅是工作中很小的一部分，因此在基础教学阶段进行必要的书面工作的训练是相当有必要的。

4 结论

本文的主要工作包含两点。其一是说明了程序设计题目在信息学竞赛教学中存在两个直接问题和一个间接问题，在没有充足的优秀教师的前提下，非程序设计题目可以作为程序设计题目训练的有效补充。其二是通过引入阶段性模型，给出了面向教学目标的非程序设计题目设计方法，总结了一些常见的题目形式，并讨论了非程序设计题目的额外优势。

我们希望列举一些关于非程序设计题目未来的研究和实践方向。

阶段性模型的验证：本文中对阶段性模型的有效性仅做了启发式的讨论，但直觉表明，存在一些实验方法验证阶段性模型的有效性。另外，由于对思维过程缺乏认识，阶段性模型事实上并没有对算法设计和代码实现两个过程做更深入的分析。实验方法也许可以增进我们对思维过程的认识，从而更好的指导教学和教研工作。

非程序设计题目的使用：本文主要讨论了非程序设计题目的设计，其在教学中的应用仍然是一个值得讨论的话题。在自学、传统学校教学、在线教学、双师教学等不同的场景下，非程序设计题目的使用应当有不同的原则和方法。

非程序设计题目的整理：在信息学竞赛社区，程序设计题目有着许多整理成册的教材，例如刘汝佳等人的 [2, 3]和秋叶拓哉等人的 [1]。由于缺乏在线题库的帮助，高质量的非程序设计题目的整理和发布是很有意义的实践工作。

在竞赛中引入非程序设计题目的讨论：尽管非程序设计题目一直在 NOI 系列比赛中存在，但由于其难度很低，往往只能起到初步筛选、普及推广等作用 (A.5)。在信息学竞赛中是否应当引入较为困难的非程序设计题目是一个相当有趣的研究话题。

致谢

陈许旻博士关于信息学竞赛教育现状的深入讨论给了作者很大的启发，与刘承奥、范致远等同学的讨论也为本文的研究提供了相当有用的帮助。感谢清华大学曹柳星老师在课程中给予的指导，以及对本文细致的审阅。

本文附录分为两部分：第一部分是文中所需的支撑资料和简要介绍，这一部分主要包括信息学竞赛的社区环境、竞赛制度、社会影响以及对文中一些论点的补充说明；第二部分包括一些非程序设计题目更深入的讨论，供教学、教研工作者参考。

附录所涉及内容在正文中以 (X.n) 的形式引出。

值得说明的是，由于国内信息学竞赛社区缺少高质量教学、教研文章的期刊，第一部分列出的许多资料来自网络，因此仅能作为本文撰写时情况的参考。

A 参考资料

参考 A.1. 例如浙江大学 2019 年自主招生简章 https://zdzsc.zju.edu.cn/2019/0402/c3299a1153281/page.htm 以及清华大学 2019 年自主招生简章 http://www.join-tsinghua.edu.cn/publish/bzw2019/12158/2019/20190516010057260895815/20190516010057260895815_.html 中，信息学竞赛奖项是报名的基本条件之一。

参考 A.2. 读者可以参阅中国计算机大会（CNCC2019）上关于全国信息学奥林匹克竞赛（NOI）的报告 https://www.ccf.org.cn/c/2019-09-04/668693.shtml。

参考 A.3. 一道完整的程序设计题目包含题目描述、样例数据、测试数据和标准程序，选手需要根据题目描述设计算法、编写程序，以通过由出题人设计的测试数据。程序设计题目的优势包括测试自动化、综合程度高、天然具有良好的区分度等。关于这一点在第 2 节中有更详细的讨论。

参考 A.4. 包括竞赛操作系统 NOI Linux，自动化测试工具 Arbiter 以及各种在线评测网站（Online Judge）。其中在线评测网站一些由学生群体维护，例如 Universal Online Judge(http://uoj.ac) 和 LibreOJ(https://loj.ac)；另一些由商业公司维护，例如计蒜客 OJ(https://nanti.jisuanke.com/oi)。值得说明的是，在线评测网站往往是一个信息学竞赛社群的中心，承担着日常训练、比赛和交流的重要作用。

参考 A.5. 在 NOI 系列赛程中，仅有省级初赛和全国决赛使用了书面试题，前者主要是为了对众多参赛选手做初步筛选，而后者主要是为了让选手了解竞赛规程、熟悉竞赛环境。关于竞赛规则的官方文件在 NOI 官网上公布，可以参考 http://www.noi.cn/articles.html?type=6。在教学中的情况尽管没有成文资料，但作者经过走访、调查，发现大多数学校并没有在教学中成体系地使用非程序设计题目。

参考 A.6. 例如 LibreOJ 收录了许多波兰信息竞赛学题目（https://loj.ac/problems/search?keyword=POI），其中包括许多具有相当思维含量的构造、贪心和图论题目。

参考 A.7. 此课程的在线资源可以在学堂在线（https://next.xuetangx.com/）上找到。此外，邓俊辉教授的“数据结构”课程也作为清华大学计算机系的核心专业课。

参考 A.8. 例如习题 12.4 *不同二叉树的数目*，利用四个步骤引导学生使用生成函数方法计算二叉树的数量，并给出一个渐进估计；习题 16.5 *离线缓存* 则分三步让学生证明将来最远

策略（Further in Future）的最优性。中间步骤降低了问题的思维跨度，引导学生掌握算法设计和分析的基本方法。

参考 A.9. 以经典的排序问题为例，第一步要想到使用比较和交换作为基本操作，第二步要想到使用分治的策略，第三步要给出线性时间的划分算法，最终才能够得到快速排序算法——这种长跨度的任务对初学者来说是相当有压力的。另一方面，一些特别复杂的题目对任何人来说都具有相当的挑战。例如 CTSC 2018 题目《字典树》http://uoj.ac/problem/403，其中出题人当时给出的程序被证明有误，直到一年之后才有人给出了正确的程序；相同情况还发生在 NOI 2017 题目《分身术》http://uoj.ac/problem/319:“拯救分身术”在一段时间中曾是信息学竞赛社区的热点话题。

参考 A.10. 仍然用经典的排序问题为例。高效的确定性排序算法（如归并排序算法）是计算机科学关心的问题，但由于选手总能用伪随机数将序列先行打乱，再使用期望时间复杂度正确的快速排序算法。因此，出题人无法用黑盒测试的方法限制选手使用确定性算法。

参考 A.11. 除了算法设计之外，算法的时间复杂度分析、时间复杂度下界的分析、近似算法（Approximation Algorithm）、在线算法（Online Algorithm）等等都是算法和数据结构中关心的问题，而程序设计题目从原理上来讲，就难以考察这些领域的知识和能力。

参考 A.12. 在近几年中（从笔者的经历而言，至少在近五年中）信息学竞赛的主要教学承担者是退役选手，教练则主要关心训练计划的安排和日常训练中的答疑。而从笔者学习信息学竞赛的时候开始，杂题选讲就已经成为了一种广为人知的教学模式，其首创者难以考证。当然，这种模式也可能是同时被许多人独立发现。

由于主要教学承担者是退役选手，退役选手（通常是大学生）课业繁忙，因此高水平授课人往往供不应求。造成退役选手承担主要教学任务的原因是多方面的——缺乏具有相应技能的中学教师、大学教师对竞赛的不了解以及教师行业对高水平选手的吸引力不足都是其社会原因。

参考 A.13. “问题-算法-分析”的模式在 MIT “算法导论”课程中广泛存在。以 [11] 第三十二章“字符串匹配”为例，这一章围绕字符串匹配问题，讲解了 RK 算法和 KMP 算法的引入、设计和分析。

参考 A.14. 第一种知识点的典型例子是网络流算法 [4]、多项式 [7] 和一些高级数据结构 [5]，后一种知识点则例如拟阵、仙人掌相关数据结构和超现实数（Surreal Number）[17] 等等。

B 非程序设计题目的例子

例 B.1 (离线区间 k 大值问题). 给定一个数列 $a_1, a_2, \ldots, a_n$ 和 q 次询问 (l_i, r_i, k_i)，要求 $a_l, a_{l+1}, \ldots, a_r$ 中从小到大第 k 个元素的大小。

1. 定义两个询问区间的编辑距离为通过在一侧加入一个元素、删除一个元素将一个区间变为另一个的最小操作次数。给出一种排列询问的方法，使得任意相邻两个询问区间的编辑距离之和为 $O((n+q)\sqrt{n})$。**提示：**将区间看作平面上的点，将横坐标按照 $\sqrt{n}$ 为大小分块，从左向右依次访问所有块，分别确定横向移动和纵向移动次数的上界。
2. 给出原问题 $O((n+q)\sqrt{n}\log n)$ 的解法。
3. **证明：**设 $q=O(n)$，那么上一问中 $O(n\sqrt{n})$ 的编辑距离之和无法被改进。换言之，存在一组询问使得最优的排列方式也需要 $\Omega(n\sqrt{n})$ 的编辑距离之和。

分析和解. 这一问题是为了向学生解释一种特殊的 $\sqrt{n}$ 优化技巧（在信息学竞赛社区中，这一方法常常称为莫队算法（Mo's Algorithm））（参见 https://codeforces.com/blog/entry/7383）。由于这一问题思维跨度很大，我们将其拆分成三个子问题，并给出了一些提示。

对于第一个子问题，完善提示便可以给出正确的算法：将区间看作平面上的点，将横坐标按照 $\sqrt{n}$ 为大小分块。换言之，将区间 $[l,r]$ 分配到第 $\lfloor l/\sqrt{n}\rfloor$ 组中。不妨设分配到第 i 组的区间集合为 B_i，考虑下面的排列方法：B_i 这一组的排列 $P(B_i)$ 定义为将 B_i 中所有区间 $[l_j,r_j]$ 按照第二维坐标 r_j 从小到大排序得到排列，所有子区间的排列由每一组的排列拼接而成，即 $P=P(B_0)P(B_1)\dots P(B_k)$.

下面分析 P 的编辑距离和。如图 5 所示，所有询问被表示为平面上的点，编辑距离和就是用箭头标出的路径的长度。横向的路径（用红色标出）要么在一组之内且每一条的长度不超过 $\sqrt{n}$，要么在两组之间且长度不超过 n，两者总长度均不超过 $O(n\sqrt{n})$；由于组内按照纵坐标排序，每一组内纵向的路径（用蓝色标出）长度之和不会超过 $2n$，总长度也不会超过 $O(n\sqrt{n})$。因此，总的编辑距离和不会超过 $O(n\sqrt{n})$。

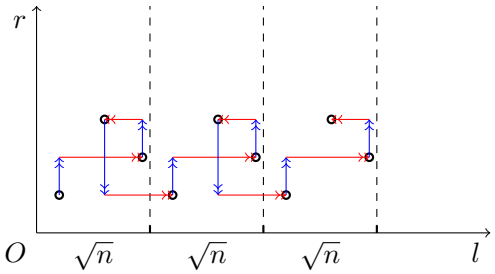

图 5: 莫队算法的分析

很容易发现，上面的构造可以利用排序在 $O(q\log q)$ 的时间复杂度完成，利用这一结论和平衡树便可以解决第二题。由于细节相对繁琐，这里略去其解答。

对于第三问而言，图 (6) 给出了一种简单的构造。由于任意两个结点之间的编辑距离都不小于 $\sqrt{n}$，总的编辑距离和至少是 $\Omega(n\sqrt{n})$。

□

例 B.2 (懒惰的小 Q)**.** 在编写利用 Splay 维护区间翻转、区间加、区间求和问题时，小 Q 常常用这样的方式来维护懒惰标记：u 上的标记表示作用在 u 的子树中，但还没有更新 u

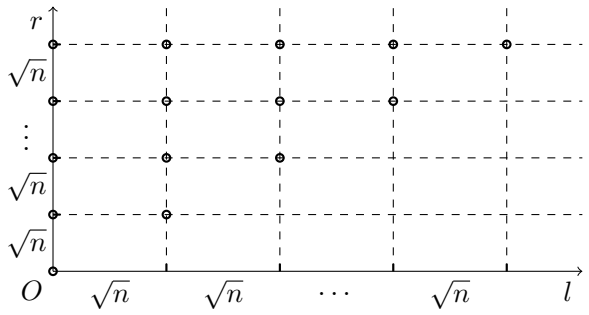

图 6: 一种可能的构造

子树中结点的变换。他常常被标记出错的 bug 所困扰——在一次 Splay 操作之后，尽管翻转标记从没有出错，但子树加标记总是出错。请向小 Q 解释其中的原因。

分析和解. Splay 是一种常用的数据结构，其最初被发表在 [23]，并被用于设计一种被称为 Link-Cut Trees 的动态树维护数据结构 [22]。这一问题主要是为了向学生解释一种和标记下传相关的非常常见的 bug。正确的维护方式应当描述为：u 上的标记表示作用在 u 的子树中，已经更新了结点 u 的信息，但还没有更新 u 子树中其他结点信息的变换。通常来说，维护标记时的 Splay 操作前需要对路径上的结点做标记下传（如图 7 所示）。

```
void pushdown_before_splay(node *nd)
{
    stack<node*> ancestors;
    for (node *p = nd->parent; p != nullptr; p = p->parent) {
        ancestors.push(p);
    }
    while (!ancestors.empty()) {
        node *p = ancestors.top();
        ancestors.pop();
        p->push_down();
    }
}
```

图 7: Splay 操作前的标记下传

然而在 Splay 的过程中被影响的结点并不只有从 `nd` 到根路径上的结点。事实上，这些结点的子结点也可能参与旋转操作，因为他们的父结点可能发生变化。如果标记所代表的含义是“未更新过 u 的信息”，那么在连接 u 和其新父结点的连边前，必须下传 u 的标记并更新 u 的信息，否则其新父结点的便会利用未更新的 u 计算子树信息——这便导致了错误的出现。如果标记所代表的含义是“已更新过 u 的信息”，则不会出现这一问题。

根据上面的分析，翻转标记不会出错的原因正是翻转不会改变子树求和的信息。 □

例 B.3 (最优二叉查找树的下界)**.** 给定 n 个键值 $k_1 < k_2 < \cdots < k_n$ 和它们在询问中出现的次数 $s_1, s_2, \ldots, s_n$，最优二叉查找树是指使得

$$cost(T) = \sum_i s_i \times \text{depth } k_i \tag{1}$$

最小的二叉查找树，其中 depth k_i 是结点在查找树中的深度。证明最优二叉搜索 T_O

的开销下界。

$$cost(T_O) = \Omega\left(\sum_i s_i \log \frac{S}{s_i}\right). \tag{2}$$

提示：考虑用查找树给 s_1 个 k_1，s_2 个 k_2，…，s_n 个 k_n 构成的序列的编码。

分析和解. 问题和解来自于 [20]。由于选手不太熟悉的分析算法下界类型的问题，我们期望学生通过从“证明比较排序比较次数下界”进行迁移，并利用提示想到这一问题的解法。

根据提示，考虑多重集 $U = \{k_1^{s_1}, k_2^{s_2}, \ldots, k_n^{s_n}\}$ 的所有全排列 $\mathcal{P}$。根据组合数学知识

$$|\mathcal{P}| = \frac{S!}{s_1!s_2!\ldots s_n!},$$

其中 $S = s_1 + s_2 + \cdots + s_n$。由于 $k_1 < k_2 < \cdots < k_n$，如果给 $k_1, k_2, \ldots, k_n$ 建立二叉查找树，我们便能够将 k_i 对应到一个长度为 $\text{depth } k_i$ 的二进制串 $\text{encode } k_i$，那么整个排列 $\mathcal{P} = \{p_1, p_2, \ldots, p_S\}$ 便可以被唯一编码到一个三进制串

$$\text{encode}(p_1), 2, \text{encode}(p_2), 2, \ldots, 2, \text{encode}(p_S),$$

其长度为

$$n - 1 + \sum_{1 \le i \le n} s_i \times \text{depth } k_i = n - 1 + cost(T).$$

因为每一个排列都可以唯一对应于一个编码，并且没有任意两个不同的排列具有相同的编码，所以可行编码的总数不少于 $|\mathcal{P}|$，即有

$$\begin{aligned}
3^{n-1+cost(T)} &\ge \frac{S!}{s_1!s_2!\ldots s_n!}, \\
\Rightarrow cost(T) &\ge \frac{1}{2}\log_3 \frac{S!}{s_1!s_2!\ldots s_n!} \\
&= \Omega\left(\log(S!) - \sum_i \log s_1!\right) \\
&= \Omega\left(S\log S - \sum_i s_i \log s_i\right) \\
&= \Omega\left(\sum_i s_i \log S - \sum_i s_i \log s_i\right) \\
&= \Omega\left(\sum_i s_i \log \frac{S}{s_i}\right).
\end{aligned} \tag{3}$$

这便完成了证明。

值得说明的是，教学工作者应该意识到这种方法隐含着信息论的思想，这一点可以在教授排序算法时通过设置题目或讲授的方法向学生解释。对排序问题而言，等概率随机排列的信息熵为 $\Theta(n \log n)$ bit，单次比较操作前后的互信息不会超过 1 bit，因此要想获取序列的全部信息至少要 $\Omega(n \log n)$ 次比较操作。 □

参考文献

[1] 秋叶拓哉. 岩田阳一. 北川宜稔. 挑战程序设计竞赛（第 2 版）. 人民邮电出版社, 2012.

[2] 刘汝佳. 黄亮. 算法艺术与信息学竞赛. 清华大学出版社, 2003.

[3] 刘汝佳. 陈锋. 算法竞赛入门经典——训练指南. 清华大学出版社, 2012.

[4] 胡伯涛. 最小割模型在信息学竞赛中的应用. *2007* 年国家集训队论文集, 2007.

[5] 杨哲. Spoj375 qtree 解法的一些研究. *2007* 年国家集训队作业, 2007.

[6] 邓俊辉. 数据结构习题解析（第 3 版）. 清华大学出版社, 2013.

[7] 金策. 生成函数的运算与组合计数问题. *2015* 年国家集训队论文集, 2015.

[8] Tim Bell. Computer science in k-12 education: The big picture. *Olympiads in Informatics*, 2018.

[9] J. Glenn. Brylow, Dennis. Brookshear. *Computer Science: An overview, 12th Edition*, chapter Introduction. Post & Telecom Press, 2017.

[10] Benjamin A. Burton. Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 2010.

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[12] Edsger W Dijkstra. The humble programmer. *Communications of The ACM*, 15(10):859–866, 1972.

[13] Edsger W. Dijkstra. Structured programming. chapter Chapter I: Notes on Structured Programming, pages 1–82. Academic Press Ltd., London, UK, UK, 1972.

[14] Stephen H. Friedberg, Arnold J. Insel, and Lawrence E. Spence (Author). *Linear Algebra*. Pearson, 2002.

[15] David Ginat. Algorithmic cognition and pencil-paper tasks. *Olympiads in Informatics*, 2018.

[16] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics : a foundation for computer science*. Addison-Wesley, 1994.

[17] Donald E. Knuth. *Surreal numbers : how two ex-students turned on to pure mathematics and found total happiness : a mathematical novelette*. Addison-Wesley, 1974.

[18] Donald E. Knuth. International olympiad in informatics: Roads to algorithmic thinking. *Olympiads in Informatics*, 2017.

[19] Marcin Kubica and Jakub Radoszewski. Algorithms without programming. *Olympiads in Informatics*, 2010.

[20] Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, Dec 1975.

[21] Jakub Radoszewski. More algorithms without programming. *Olympiads in Informatics*, 2014.

[22] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, June 1983.

[23] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985.