DOI:10.16512/j.cnki.jsjjy.2013.04.026

文章编号：1672-5913(2013)04-0058-04 中图分类号：G642

数据结构、算法和程序之间关系的探讨

沈 华
（湖北工业大学 计算机学院，湖北 武汉 430068）

摘 要：数据结构是计算机科学与技术和信息类相关专业的一门重要综合性专业基础课。针对实际教学过程中学生对数据结构与算法的关系、算法与程序的关系较难理解以及目前多数数据结构教材对 2 种关系阐述不够详细的情况，从算法设计和算法实现 2 个层面深入讨论并阐述数据结构与算法、算法与程序的密切关系。

关键词：数据结构；算法；程序

数据结构与算法是计算机专业的核心课程之一，它是操作系统、软件工程、数据库概论、编译技术、人工智能、计算机图形学等专业课程的必修先行课。很多应用软件都要使用到各种数据结构和算法编写程序，进行科学计算和模拟试验等[1]。数据结构与算法已经成为软件开发工程师必备的基础知识之一。在学校里，它已经成为计算机学科的重要课程，同时也成为许多其他专业的热门选修课。社会上大多数公司在招聘软件开发人员时必然会考查应聘人员对数据结构与算法的掌握程度，并将此作为衡量应聘者水平的重要依据[2]。

实际问题通常很复杂，我们运用计算机求解实际问题的时候，往往是对实际问题的模型进行求解。从本质上讲，模型是从实际系统概念出发的、关于现实世界的一小部分或某几方面的“抽象”的“映像”，系统建模需要进行如下抽象：输入量、输出量、状态变量及它们之间的函数关系[3]。简单地说，模型就是对实际问题的一个简化，是反映问题本质的数据集合以及数据之间关系的集合。模型的求解是对给定的输入找到针对数据的一系列处理步骤（即算法），以得到预期的输出。用计算机求解实际问题还需要把模型映射到存储器并将算法转换为程序。因此，我们有必要深入理解数据结构和算法的关系，弄明白算法和程序之间的区别和联系。

1 认识数据结构

学校的各院、系、部按照隶属关系得到一个层次结构；家庭成员按照父子关系得到一个层次结构；家庭成员按照年龄关系得到一个线性结构；各城市按照它们之间的互通关系得到一个复杂的网状结构。由此可见，结构是关系的一种表现形式。

计算机求解应用问题有 2 个视图：抽象视图和实现视图。抽象视图关注的是数据之间已存在的某种关系，与机器无关，它体现出的结构称为数据的逻辑结构。算法设计者在抽象视图层面上设计抽象算法。实现视图主要解决如何让计算机“看见”数据以及数据之间的逻辑关系，以便计算机能够根据程序设计者设计的程序处理“看见”的数据。计算机的可视区是内存（当然还有高速缓存和寄存器），因此实现视图需要将数据和它们之间的逻辑关系存

基金项目：国家自然科学基金青年基金项目（41204112）。
作者简介：沈华，女，讲师，研究方向为算法、移动计算、服务计算。

储到内存中。计算机在存储数据之间的关系时，数据对应的存储映像之间也会呈现出一种（新）结构，这种结构称为数据的存储结构。例如，线性表是抽象视图中的对象，元素之间的线性关系（逻辑关系）存储在存储器中可能仍然呈现出一种线性结构（顺序表），也可能呈现出一种非线性结构（链表）。程序设计者在实现视图层面上将抽象算法转换成具体的可执行程序。

换句话说，数据的逻辑结构回答了数据之间具有什么样的关系，是分析阶段的产物；数据的存储结构则解决了如何将这个产物装入存储空间的问题，因此才会出现一种逻辑结构可映射成不同存储结构的情况。当然，数据的逻辑结构会被映射为哪一种存储结构要视具体应用决定[4]。

2 认识算法和程序

算法是解决问题的一系列确切步骤，它应满足有效性、确定性、有穷性、0 个或多个输入、1 个或多个输出 5 个特性。算法可以用自然语言描述；为了方便计算机实现，也可用伪代码描述。程序是算法在计算机中的实现，它是程序员选用一种计算机能够识别的语言对被实现算法的重新描述（在这里，程序员在实质上是充当了翻译员的角色）。经过编译器的编译，程序最终被转换为一系列指令。当冯·诺伊曼型计算机中的执行部件执行这一系列指令并得到 1 个或多个输出后，便出现“奇迹”：计算机解决了实际应用问题。

通过上面的讨论，我们可以发现不懂计算机编程的人也完全可以写出非常好的算法，但程序必须由具有一定计算机背景的人来完成。当然，算法的设计和算法的实现（即程序）可能由同一个人完成，也可能由不同的人完成。

此外，程序和算法还有一个显著的不同：程序可以不受有穷性的约束，但算法必须满足此特性。如果算法没有满足有穷性，那么意味着算法没有解决问题，我们也就不能称其为算法。也就是说，算法中一定存在导致求解结束的触发条件（当然可能有多个触发条件，不同的触发条件可能导致不同的输出）。程序是算法的实现，在程序的执行过程中，如果我们一直不给它运行结束的指令，那么它将一直运行下去。

3 数据结构、算法和程序之间的关系

3.1 关系描述

我们首先必须明确的是数据的逻辑结构与算法的设计相关，而数据的存储结构与算法的实现相关。我们应用计算机求解一个实际问题时，首先需要做的是对实际问题进行建模，得到它的数学模型或抽象模型。数据的逻辑结构就是将问题抽象后得到的，换句话说，它是抽象模型的一个组成部分。算法设计者根据问题的抽象模型、可能得到的输入和期望得到的输出设计出求解该问题（实际上是对问题抽象模型的求解）的算法。显然，问题抽象的好坏直接影响在此基础上得到的算法质量。因此，“抽象模型”与“算法”的关系就如同政治经济学中“经济基础”与“上层建筑”的关系；“数据的逻辑结构”与“抽象模型”的关系就如同“生产力水平”与“经济基础”的关系。

得到算法后，我们需要做的是让计算机能够读懂算法并按照算法指定的处理步骤求解目标问题。这需要我们把抽象模型映射到存储器中，包括将算法需要处理的数据以及数据之间的联系映射到存储器中。这种做法在许多科幻大片中都可以找到它的影子。相信大家还记得 2010 年大受欢迎和好评的电影“盗梦空间”吧，这就是一个为了让人们体验虚拟世界中的各种奇妙感受，将社会群体中的人以及人们之间的关系从现实世界映射到虚拟现实世界的例子。抽象模型映射到存储器后，一个直观的结果是计算机可以感知该模型。我们现在需要把算法（对抽象空间中模型的处理）转换为驱动计算机执行的一系列指令（这些指令是对存储空间中模型的处理）。

这个转换过程如何实现？我们可以先回顾一下程序设计语言的发展。计算机能读懂的指令是

一个 0、1 代码串（为什么会这样？这是由于现在的计算机是基于图灵机计算模型和冯 • 诺伊曼体系结构设计得到的缘故），但是这些枯燥的 0、1 代码串不便于程序员使用。为了提升程序员的使用体验，低级程序设计语言（如汇编语言）便出现了，这种语言用符号表示指令（这是对底层指令的一种低级抽象）。程序员用这些符号进行编程比直接用指令感觉好些，但还是感觉不便，于是又出现了高级程序设计语言，如 C、Java、Pathon、C++ 等。有了高级程序设计语言，程序员可以完全投入到编程中，享受编程的乐趣，而不用理会恼人的 0、1 代码串或那些难以记住和理解的符号。

然而，这又产生了计算机与程序员之间语言不互通的矛盾。如何调和矛盾呢？编译程序功不可没。有了编译程序，从虚拟机的角度上来说，我们可以认为此时程序员能够使用高级程序设计语言与计算机进行愉快的交流。如果计算机被比作一个公司的 CEO，那么编译程序就相当于翻译官。

通过前面的分析可知，有了相应的编译程序，计算机就能“ 理解 ”高级程序设计语言，但计算机仍然不能“ 理解 ”自然语言或伪代码。如何让计算机按照设计阶段得到的抽象算法对问题进行求解呢？程序员是既懂自然语言或伪代码，又懂高级程序设计语言的人，因此程序员可以解决这个问题。程序员用某种高级程序设计语言实现抽象算法后得到的产物通常称为程序。计算机“ 理解 ”并执行程序，从而实现按照算法对问题进行求解。

程序员将抽象算法中对数据访问、处理或存取的描述改造为程序段时，需要依据映射到存储器中的模型。也就是说，不同的模型映射方式使得同一抽象算法对应的程序是不相同的。需要注意的是，模型在存储器中的映射方式也是通过程序描述和体现的。

3.2 实例说明

下面我们以 2 个一元多项式相加为例，阐述数据结构、算法和程序之间的密切关系。

1）问题描述。

求 2 个一元多项式 $f_1(x)$、$f_2(x)$ 的和式 $f(x)$，即 $f(x) = f_1(x) + f_2(x)$。

2）对一元多项式进行建模，得到一元多项式的逻辑结构。

我们以 1 个形如 $P(x) = 8x^{14} - 3x^{10} + 10x^4$ 的一元多项式为例。通过观察可以发现，一元多项式中的每一项可以用 1 个序偶 < 系数，指数 > 唯一确定，因此我们可以用这样的序偶集合表示 1 个多项式，如上面的 $P(x)$ 可以表示为 {<8, 14>，<-3, 10>，<10, 4>}。如果约定指数大的序偶排在前面、指数小的序偶排在后面，那么就得到 1 个以序偶为数据元素的线性表 LP:（<8, 14>，<-3, 10>，<10, 4>）。

序偶的类型定义如下：

```
typedef struct
{
float coef;  // 序偶中的系数分量
int exp;  // 序偶中的指数分量
}PElemType;
```

3）在逻辑结构的基础上设计抽象算法。

```
algorithm: polynomial_add
input: 2 个一元多项式 f1 和 f2
output: f1 与 f2 相加的和 f
step1.  初始化 f 为空线性表
step2. i ← 1, j ← 1, k ← 1;
step3. while(f1 和 f2 均未遍历结束 )
step4. {
step5. e1 ← f1 的第 i 个元素 ;
step6. e2 ← f2 的第 j 个元素 ;
step7. if(e1.exp > e2.exp)
step8. {
step9. 将 e1 插入到 f 的第 k 个位置上 ;
step10.          i ← i + 1;
step11.          k ← k+1;
step12. }
step13. else if(e1.exp < e2.exp)
step14.  {
step15. 将 e2 插入到 f 的第 k 个位置上 ;
step16.          j ← j + 1;
```

```
step17.          k←k+1;
step18. }
step19. else  // 合并同类项
step20. {
step21. coef_add ← e1.coef + e2.coef;
step22. if(coef_add 不为 0)
step23. {
step24. e.coef ← coef_add;
step25. e.exp ← e1.exp;
step26. 将 e 插入到 f 的第 k 个位置上 ;
step27. }
step28.          i ←i +1;
step29.          j ←j +1;
step30.          k←k +1;
step31. }
step32. }
step33. while(f1 没有遍历完 )
step34. {
step35. e1 ← f1 的第 i 个元素 ;
step36. 将 e1 插入到 f 的第 k 个位置上 ;
step37.          i ←i +1;
step38.          k ← k+1;
step39. }
step40. while(f2 没有遍历完 )
step41.{
step42. e2 ← f2 的第 j 个元素 ;
step43. 将 e2 插入到 f 的第 k 个位置上 ;
step44.          j ←j +1;
step45.          k ← k+1;
step46. }
step47. return f;
```

4）在存储结构的基础上，根据不同的存储要求使用某种高级程序设计语言，实现上述抽象算法，得到相应程序。

例如，我们根据下面的 4 种存储要求选用某种程序设计语言（如 C 语言）实现算法 polynomial_add，得到 4 个不同的程序。① 序偶对线性表采用顺序存储结构，利用原有的存储空间存放相加结果 f（即不增加新的存储空间）；② 序偶对线性表采用顺序存储结构，相加过程中增加新的存储空间存放相加结果 f，不破坏 f_1 和 f_2 存储空间中的信息；③ 序偶对线性表采用链式存储结构（单链表），利用原有的存储空间存放相加结果 f；④ 序偶对线性表采用链式存储结构（单链表），相加过程中增加新的存储空间存放相加结果 f，不破坏 f_1 和 f_2 存储空间中的信息。

4 结语

计算机解决实际问题必然会涉及信息表示和信息处理的问题。信息表示包括 2 个方面的内容：信息的逻辑表示（包括数据表示和数据逻辑关系表示）和信息的存储表示（包括数据的存储表示和数据逻辑关系的存储表示）。信息处理包括设计处理方案和实现处理 2 个阶段：第 1 个阶段需要在信息的逻辑表示层上设计合理有效的算法；第 2 个阶段需要在信息的存储表示层上运用某种程序设计语言将算法转换成计算机可运行的程序。笔者对计算机解决实际问题中所涉及的 3 个重要内容——数据结构、算法和程序进行探索，力图帮助学生正确理解和掌握这 3 个重要内容以及它们之间的内在联系。

参考文献：
[1] 张铭, 赵海燕, 王腾蛟, 等. 北京大学 “ 数据结构与算法 ” 教学设计[J]. 计算机教育, 2008(20): 5-11.
[2] 周伟明. 多任务下的数据结构与算法[M]. 武汉: 华中科技大学出版社, 2006: 1.
[3] 刘兴堂, 梁炳成, 刘力, 等. 复杂系统建模理论、方法与技术[M]. 北京: 科学出版社, 2008: 6.
[4] 沈华, 杨晓艳, 马驰, 等. 数据结构及应用: C语言描述[M]. 北京: 机械工业出版社, 2011: 1.

（编辑：宋文婷）