计算机能否思考的问题就像潜艇能否游泳的问题一样

—— 艾兹格·迪杰斯特拉

迪杰斯特拉——一个划时代的杰出计算机科学家

陈关荣

（香港城市大学）

你知道 Edsger Wybe Dijkstra 这个名字吗？如果你是计算机专业，熟识编程或者是接触过“最短路径问题”的话，这个名字如雷贯耳。不然，你对这个名字可能会觉得非常陌生。

艾兹格·迪杰斯特拉（Edsger Wybe Dijkstra，1930 年 5 月 11 日—2002 年 8 月 6 日）是荷兰人。他的名字是荷兰语，其中 Edsger 是荷兰弗里西亚（Frisia）地区的名字，具有敏锐果断和领导能力的涵义，Wybe 是战士的意思，Dijkstra 则是一个常见的荷兰姓氏，意指“来自堤坝”，因为历史上建造堤岸预防海潮一直是荷兰的重要战略举措。

迪杰斯特拉的重要学术贡献涵盖了数学图论和计算机科学的编译器、操作系统、分布式系统、程序设计、编程语言、程序验证、软件设计等多个方面。极负盛名的美国计算机科学家唐纳德·克努斯（Donald E. Knuth，1936-）在多篇纪念文章和访谈中，例如在国际计算机协会（Association for Computing Machinery）月刊《ACM 通讯》（Communications of the ACM）发表的悼念文章中，称迪杰斯特拉为“划时代最杰出计算机科学家之一”（one of the greatest computer scientists of all time）。顺便提及，克努斯在 1977 年出访中国前，著名计算机科学家姚期智的夫人储枫给他起了一个中国名字：高德纳。

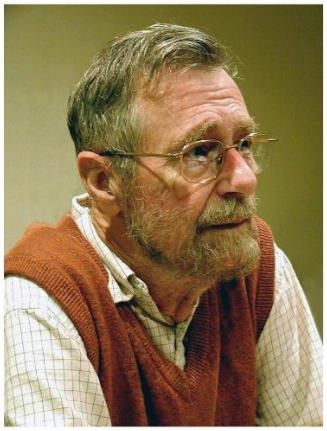


图 1 艾兹格·迪杰斯特拉 （1930-2002）

【一】三页纸论文和二十分钟思考

1959 年，迪杰斯特拉在一个刚创刊的《数值数学》杂志发表了一篇三页纸的短文“关于图论两个问题的注记”（A note on two problems in connexion with graphs，Numerische Mathematik，vol. 1, pp. 269-271, 1959），其中第一个问题是现在称为“Prim 最小生成树算法”，由美国数学家罗伯特·普利姆（Robert C. Prim III，1921-2021）在 1957 年独立提出，而第二个问题是现在称为“Dijkstra 最短路径算法”，是历史上第一个图论最短路径问题的算法。这个算法一直被应用于计算机和通信网络路由、GPS 和高德地图导航、城建交通和物流规划、社会网络传播等许多方面。至今，这三页纸的单篇短文获得近四万次引用。

2001 年，迪杰斯特拉在 EWD1308 号题为“关于开发一个最短路径算法的回忆”的手稿中说，他 26 岁时思考过一个平常人都会遇到的最短路径问题：在荷兰两个城镇鹿特丹（Rotterdam）和格罗宁根（Groningen）之间的最短路经是什么？

> “一天早上我和年轻的未婚妻丽娅（Ria）在阿姆斯特丹（Amsterdam）逛街。我们觉得有点累了，就坐在咖啡厅的露台上喝咖啡。我还在思考是否能够解决上述问题。很快，我便设计出了一个最短路径算法。我说过，那是一个二十分钟的设计。事实上，三年之后的 1959 年它才被正式发布。现在看来，这算法依然很不错。其原因之一是我当时手头上并没有纸和笔，从而不得不极力避免所有可以避免的复杂算法。最终，令我惊讶的是，这个算法成为了我成名的基石之一。”

图 2 迪杰斯特拉（左），朋友 Bram Loops（中），未婚妻 Ria（右）（1954 年）

【二】 最短路径的问题和算法

这里，先介绍一下迪杰斯特拉的“最短路径”算法是有趣的。

迪杰斯特拉算法用于计算无向加权连通图中从任意一个节点到任意另一个节点的最短路径。算法以起点为中心，向外层层扩展，直至扩展到终点为止。

考虑图 3 所示的一个简单例子，其中的加权数字表示节点间的距离。假定从节点 A 出发，寻求到达节点 E 的最短路径 $L(A,E)$。该算法从节点 A 开始，初始值为 $L(A,A)=0$，分别计算到达各个邻居节点的距离，其中计算先后次序可以随意。这里容易看到：$L(A,B)=L(A,A)+3=3$, $L(A,C)=L(A,A)+2=2$, $L(A,D)=L(A,B)+4=7$。其中，$L(A,D)$ 还有两条路经，但因为不是最短的，就不要了。把这三个最短距离记录下来。然后，分别从各个邻居节点出发，向节点 E 方向移动并作类似计算：$L(A,E)=L(A,B)+5=8$, $L(A,E)=L(A,B)+L(B,D)+1=8$, $L(A,E)=L(A,D)+1=10$, $L(A,E)=L(A,C)+L(C,D)+1=9$。选取最短的一个：$L(A,E)=8$。因为已经到达终点 E，本例计算过程结束。

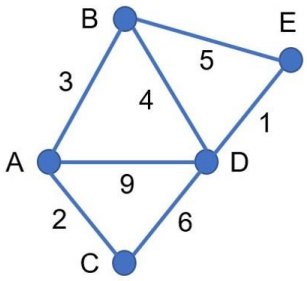


图 3 迪杰斯特拉最短路径问题和算法示例

迪杰斯特拉算法可以得到从起点向外扩展过程中每一层次的最短距离。在本例中，第一层的最短距离是 $L^*=2$, 第二层是 $L^*=8$。当某一层到达了终点，扩展便停止，该层的最短距离 L^* 就是问题的答案。迪杰斯特拉算法事实上得到了从出发节点到图中任何一个节点的最短距离。在本例中，算法给出了：$L(A,B)=3$, $L(A,C)=2$, $L(A,D)=7$, $L(A,E)=8$。显然，这个算法需要计算所有的可能性，因而是一种非多项式时间（NP）的“贪婪算法”（greedy algorithm）。不过，它的优点是容易编程而且它是普适最优（universally optimal）的。

【三】 一个谦卑的程序员

迪杰斯特拉于 1930 年 5 月 11 日出生在荷兰鹿特丹。他的父亲杜韦（Douwe Wybe Dijkstra，1898-1970）是阿姆斯特丹一所中学的化学老师和校长，曾担任过荷兰化学协会主席，母亲名叫布莱希特耶（Brechtje Cornelia Kluijver，1900-1994），学数学出身。迪杰斯特拉在家中四个孩子里排行第三。后来坊间有个传说，迪杰斯特拉小时候问过母亲，数学是不是很难？母亲回应说：不能说很难，但如果你需要写超过五行文字来证明什么的话，你的方向多半是错了。

1948 年，迪杰斯特拉中学毕业。当时他的理想是学习法律，然后去联合国工作。不过，他的数理课程成绩极其优异，进入了莱顿大学（Universiteit Leiden）学习数学和物理学。莱顿大学成立于1575年，由荷兰革命领袖奥兰治亲王威廉（Willem van Oranje，1544-1584）所建，以纪念莱顿市在荷兰从西班牙帝国统治下取得独立战争胜利中的功绩。迪杰斯特拉后来回忆说，他的大学生涯“很穷很累很快乐”。读书期间，父亲建议他报名修读英国剑桥大学刚刚开设的为期三周的世界第一台存储程序式电子计算机 EDSAC 的编程课，希望他学习一些新的计算机技术。该课程由著名计算机科学家莫里斯·威尔克斯爵士（Sir Maurice V. Wilkes，1913-2010）讲授。当时，威尔克斯爵士给曾于 1950 年在剑桥学习的“阿姆斯特丹数学中心”主任阿德里安·范·维恩加登（Adriaan van Wijngaarden，1916–1987）写了封信，想确认一下迪杰斯特拉的基础知识是否足够。回信说迪杰斯特拉的知识肯定是足够的，还说课程结束后会请他到数学中心工作，当一名程序设计员。于是迪杰斯特拉去了剑桥，学习结束后返回荷兰，在阿姆斯特丹数学中心兼职工作同时继续在莱顿大学修读理论物理学。

当年，迪杰斯特拉觉得很困惑：这计算机编程是啥东西呢，它和自己梦想成为受人尊敬的物理学家风马牛不相及。于是他咨询了范·维恩加登，两人作了一次长谈。范·维恩加登对他说，现在的确还没有所谓的编程科学。但是，计算机必将继续存在并且影响整个世界。现在所有这些都是一个充满希望和未知的开始。你怎么知道将来你不会成为编程科学的创始人呢？那次谈话是迪杰斯特拉人生最重要的转折点，让他成为了荷兰的第一个正式职业程序员。1956 年，他从莱顿大学毕业后仍然留在数学中心工作，走上了“码农”的不归路。

不过，当年计算机程序员的确不是一个为人所知的工作职业。迪杰斯特拉后来曾戏说自己是“第一个靠编程赚钱的荷兰人”。1957 年，迪杰斯特拉与玛丽娅（丽娅）·德贝茨（Maria“Ria”Cornelia Debets，1931-2012）结婚。当时，为他们登记结婚的阿姆斯特丹市政府不承认他的职业身份，认为根本就没有“程序员”这种职业。最终，他结婚证明书“职业”一栏里填写的是“理论物理学家”。迪杰斯特拉和丽娅后来养育了三个儿子：马库斯（Marcus）、费姆克（Femke）和罗格斯（Rutger）。

1958 年 11 月，迪杰斯特拉代表阿姆斯特丹数学中心出席了在德国美因茨（Mainz）召开的国际计算机会议。那是一个定义 ALGOL 语言（ALGOrithmic Language）具体内容的预备会议。

当年，迪杰斯特拉依然是一位在职研究生，他一边工作一边在阿姆斯特丹大学修读博士学位，导师是数学家约翰内斯·哈恩捷斯（Johannes Haantjes，1909-1956）。迪杰斯特拉于 1959 年毕业，博士论文题目是“利用自动计算机进行通信”（Communication with an automatic computer），专门为荷兰开发的第一台商用计算机 Electrologica X1 设计汇编语言。

迪杰斯特拉在阿姆斯特丹数学中心长达十年的编程工作中，反复遇到了当年的低级计算机和高级编程技巧的矛盾挑战。在 1960 年代，软件复杂度的激增导致许多科研项目频繁失败。那时候，FORTRAN 是最早获得普及的一种高级程序语言，但其编译系统依然存在不少问题。迪杰斯特拉在战胜失败的过程中通过不懈努力获得了丰富的编程经验，构建了自己一套独特的编程思想，把堆叠（stack）的概念用于编译以及动态存

储分配，首创了优先数编译算法。他与同事亚普·佐内维尔德（Jaap A. Zonneveld，1924-2016）合作开发了第一个结构化编程语言 ALGOL 60。1959 年 12 月，迪杰斯特拉对 ALGOL 60 的诞生感叹地说：“一个奇迹就被这样被简单地创造了。”次年，他在《数值数学》杂志上发表了论文“递归编程”（Recursive programming, Numerische Mathematik, vol. 2, pp. 312-318, 1960），引进了许多新的概念和术语。该文在可编程语言设计和发展演化中影响巨大，为后来计算机程序模块化打下了基础。据说，迪杰斯特拉和佐内维尔德约定在 ALGOL 60 设计全部完成之前谁都不许剃掉胡子。结果，佐内维尔德在项目结束后很快就剃光了胡子，而迪杰斯特拉却终生保留着胡子。

1960 年，阿姆斯特丹数学中心首先在荷兰，然后在英国布莱顿（Brighton）开设了 ALGOL 60 程序设计语言的课程。1962 年 4 月，国际知识产权罗马公约组织同意了其大部分的技术细节。同年 8 月，国际程序设计语言联盟（IFIP）复查并批准了关于 ALGOL 60 的技术报告。1972 年，迪杰斯特荣获 ACM 图灵奖。他在题为“谦卑的程序员”（The humble programmer）的演讲中说：“只有极少几个像 ALGOL 60 报告这样简短的文件能给计算机界带来如此深远的影响。”

1962 年，迪杰斯特拉到了埃因霍温理工大学（Eindhoven University of Technology）担任数学教授。埃因霍温是今天世界著名的光刻机制造商 ASML 的所在地。在埃因霍温理工大学，迪杰斯特拉讲授计算机科学“数值分析”课程并组织了一个计算机研究小组。很快，他们的研究小组就成了有名的“礼拜二下午俱乐部”（The Tuesday Afternoon Club），在那里他们讨论、研究并发展了计算机编程各个方面的思想、技术、理论和应用。特别是，迪杰斯特拉牵头设计了 THE（Technische Hogeschool Eindhoven）计算机操作系统。那是世界首个采用分层架构设计、采用模块化并具有可验证性的系统，在该大学计算中心顺利运行了十年。

然而，在 1960 年代，迪杰斯特拉的学术生涯过得不怎么好，因为社会上特别是数学系的同事们都觉得他的工作并不重要甚至没有价值。

1960 年代，全世界都在使用 IBM 360 系列计算机，其主要程序语言是 FORTRAN，而无条件语句 GOTO 语句则是 FORTRAN 的支柱。但是，该语句导致程序流程混乱、难以理解和维护，被称为是“意大利面条式代码”。1968 年，迪杰斯特拉给《ACM 通讯》写了一封信，题为“反对 GOTO 语句”（A case against the GOTO statement），经审稿后按编辑的意见改名为较为温和的“GOTO 语句可能有害”（Go to statement considered harmful）发表。他开篇就说：“多年来，我注意到一个现象：程序员写程序时使用 GOTO 越多，他的素质就越低。我确信应该从所有‘高级’编程语言中废除 GOTO 语句。”他指出了程序中使用 GOTO 语句的危害，认为它会使程序的控制流程变得复杂、难以理解和维护。他指出 GOTO 语句可以用顺序结构、选择结构以及循环结构来代替。他提倡使用结构化的控制流，如 if-then-else 和 while 循环等，来提高程序的质量和可读性。迪杰斯特拉的公开信引起了广泛的争论，是计算机历史上重大事件之一，他也因之声名鹊起。迪杰斯特拉的思想和建议对程序设计语言的发展产生了深远的影响。后来许多新程序设计语言都极力避免使用 GOTO 语句，或者将其限制在非常有限的范围内。

【四】 哲学家就餐问题

1965 年，迪杰斯特拉研究计算机操作系统和程序进程中的“死锁”（deadlock）问题及解决方法。该问题后来被英国计算机科学家托尼·霍尔爵士（Sir Charles Antony R. Hoare，1934-）重新表述并称之为“哲学家就餐问题”。

假设有 N 台计算机试图访问 N 个共享的磁带驱动器。该问题在 $N = 5$ 时的形象表述如下：假设有五位哲学家围坐在一张圆桌进餐。他们只能做以下两件事情之一：要么吃饭，要么思考。餐桌中间放有一大碗意大利面。假设哲学家必须用两只餐叉才能吃到面条，但两两哲学家之间桌上只放有一只餐叉。哲学家可用左手或右手去取用餐叉。最后假定哲学家们相互绝不交谈，每个人都不知道其他人的想法和动作。这样，很可能每个哲学家都用左手拿着餐叉，永远在等待右手有餐叉可用，或者相反。这对应着计算机的“死锁”状态。另外，也有可能发生“资源耗尽”情形：例如，规定哲学家在等待另一只餐叉超过五分钟的话他就得放下自己手里的餐叉，并且要再等五分钟后才能进行进下一次操作。这个策略消除了死锁，因为系统总会进入到下一个状态，但有可能会发生“活锁”，譬如五位哲学家同时拿起左边的餐叉，同时等待五分钟，然后同时放下手中的餐叉，再同时等待五分钟，然后又同时拿起左边或右边的餐叉，如此不断循环重复，直到所有哲学家都饿死即全部资源被耗尽。

哲学家就餐问题的一个简单解法是引入一个餐厅服务生，让哲学家听从他的调配拿起或放下餐叉。因为服务生知道哪只餐叉正在被谁和怎样使用，所以他能够作出正确判断，避免死锁。作为解释，我们把五位哲学家依次标号为 A 至 E。如果某个时刻只有 A 和 C 正在吃面条，则有四只餐叉在使用中。B 坐在 A 和 C 之间，所以他两边都没有餐叉可用。因为 D 和 E 之间有一只空余餐叉可用，如果 D 或 E 拿起了这只餐叉，他就会引发死锁。但是，如果这时服务生规定所有人都有一定的用餐时间，并让 D 或 E 等待直至有两把餐叉空余出来，这时两者中至少有一位可以得到一对餐叉，从而避免了死锁。当然，这不是唯一的解法，后来还有几种不同条件下的不同解法，可用于应对计算机编程中各种死锁和资源耗尽问题。

【五】 银行家演算法

1965 年，迪杰斯特拉为 THE 计算机操作系统设计了一种避免产生死锁的演算法。该算法以银行借贷系统的分配策略为基础，判断并保证系统的安全运行，后人称之为“银行家演算法”。

在银行业务中，客户申请贷款的数量是有限的，每个客户在第一次申请贷款时要声明完成该项目所需要的最大资金量，并保证满足所有贷款要求，最后及时偿还贷款。银行家在客户申请的贷款数量不超过自己拥有的款项最大值时，都应尽量满足客户的需求。在这样的描述中，银行家就好比操作系统，资金就是资源，客户就相当于要申请资源的运算进程。

银行家演算法的大意是：假定系统中各种资源（如内存、CPU、I/O 设备等）的数量是有限并固定不变的。先列出一个资源分配表，然后判断是否为安全状态。为此，首先要找出它最大需要的资源（need），减去原本已经分配出去的资源，得到剩下待分配

的资源（available）。接下来，找出 need 比 available 小的客户，继续进行分配，直至全部资源用尽。这样的进程被视为处于安全状态。

由于系统无法知道什么时候一个进程将会终止，或者之后它还需要多少资源，因此系统假定：进程中的每一个步骤都试图获取其声明需要的最大资源，并在一定时间内终止。在大多数情况下，这是一个合理的假设，因为只要不死锁，系统不是特别关注各个进程分别运行了多久。基于这一假设，该算法通过尝试寻找一个理想的操作集合，允许每个进程步骤中获得最大资源并满足结束进程的要求即把剩余资源返还给系统，以保证进程状态是安全的。不存在这个集合的状态都是不安全的，因而需要避免。

迪杰斯特拉的银行家算法是理解和避免死锁的重要基础，虽然今天实际系统中较少直接使用，但是其思想对资源管理有深远的启示和影响。

【六】结构化编程奠基人

1968 年，迪杰斯特拉发表了他三年前便设计好的一项“协作顺序进程”策略（Cooperating sequential processes, Chapter in The Origin of Concurrent Programming, Springer, 1968, pp. 65-138），开创了循环编程（recurrent programming）技术。在该文中，他报告了一个关键的同步（synchronization）现象并称之为“进程互斥问题”（mutual exclusion）。事实上，早在另一篇单页短文中（Solution of a problem in concurrent programming control, Communications of the ACM, vol. 8, no. 9, 1965, p. 569），他就已经有了一个解决方案。

1970 年，迪杰斯特拉出版了《结构化编程笔记》（Notes on Structured Programming）一书。这是他最有名的技术文献之一，也是计算机科学史上影响深远的著作。他写于 1969 年的手稿被收录在 1972 年他与霍尔爵士和挪威计算机科学家奥利-约翰·达尔（Ole-Johan Dahl，1931-2002）合编的专著《结构化编程》（Structured Programming）中。这三位合著者后来都分别获得 ACM 图灵奖。迪杰斯特拉率先于 1972 年获得该项大奖。奖词中说：“Notes（即《结构化编程笔记》）提倡的一些设计原则如今在计算机科学界已被广泛接受。大型系统应由较小的组件构建而成，其中每个组件应仅依据其接口而非实现方式来定义，而较小的组件也可遵循类似的分解流程进行设计，从而形成自上而下的设计风格。设计应从列举‘可分离的关注点’开始，并将每组关注点独立于其他组件进行考量。背后的数学逻辑是而且必须是软件设计的基础。”奖词中表彰说：“他对编程这一高智力挑战做出了根本性的贡献。他顽强地坚持并实际上证明了程序应该被正确地编写出来，而不是事后调试到正确。他阐明了对程序设计基础问题的认识。”从此，迪杰斯特拉被尊为结构化编程设计的奠基人。

【七】编程的修炼

1976 年，迪杰斯特拉出版了《编程的修炼》（A Discipline of Programming）一书。这是一本非常独特的编程技术专著，它没有使用任何可以由计算机执行的语言，而是采用自己设计的一种特别的小型语言来进行描述。他开发了“谓词转换器”（predicate transformer），作为定义程序语义的基础和程序推导的工具。他在书中给出了一大批编程实例，并在复杂程序开发完成后给出细致的回顾性分析。该书最重要的关注点是程序的正确性，是关于程序及其意义的严格数学推导和证明。他在前言中

写到，我们的目标应该是“做出高度可信的程序，而不仅仅是 …… 一些胡乱涂抹出来的 …… 随时可能被反例推翻的程序文本”。他认为编程工作的目标应该是得到正确的程序，而不是一些“大致能用”的代码。他还认为程序语言本身是一种严格定义下的形式化描述，用它们写出来的程序都有确切的语义，而用具体计算机运行程序就是实现其语义，但程序的语义并不依赖于运行它的计算机。该书最后没有列出任何参考文献。作者声明：“对于没有参考文献的问题，我不准备解释，也不表示歉意。”为什么呢？因为是首创，根本没有参考文献。尽管如此，该书后来获得大量的引用，被科学引文索引（SCI）评定为“经典引文”（citations classic）。

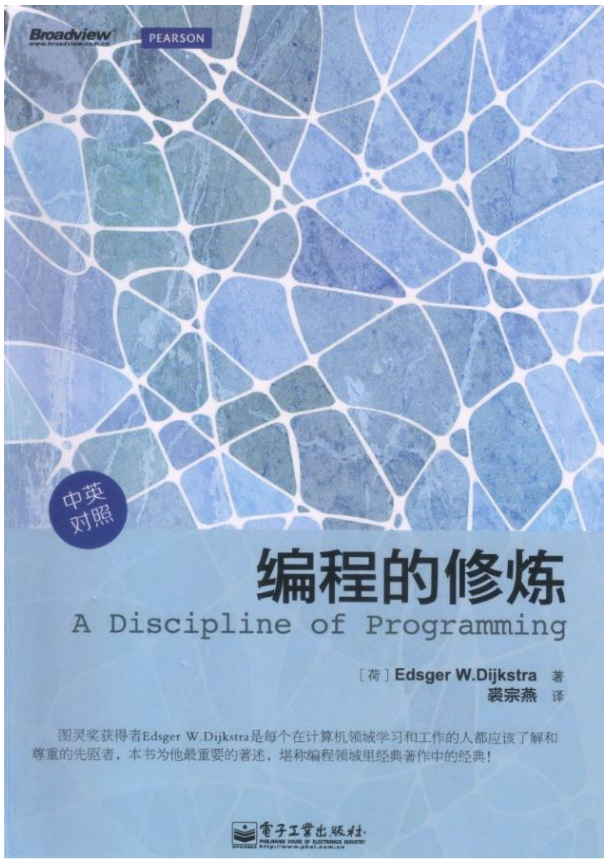


图 4 迪杰斯特拉的专著《编程的修炼》

迪杰斯特拉这一时期的重大学术贡献之一是发展了非确定性（nondeterminacy）理论。这是一个超越传统数学的概念。他首次观察到，非确定性不仅在组件异步交互的计算中至关重要，而且即使在不涉及异步的情况下，它也是推理程序和简化程序设计的有效工具。此外，他还发展了编译器设计和计算器程序等解析数学表达式中的调度场算法（Shunting yard algorithm），利用管理运算符的优先级和结合性，将中缀表达式转换为后缀表达式，实现表达式的重新排列。

在他一系列的论文和著作中，迪杰斯特拉引进了许多有用的概念和短语，极大地丰富了计算语言，包括结构化编程（structured programming）、关注点分离（separation of concerns）、同步（synchronization）、致命拥抱（deadly embrace）、最弱前提条件（weakest precondition）、受保护命令（guarded command）、排除奇迹

（excluded miracle），以及用于控制计算机进程的信号量（semaphore）和被写进《牛津英语词典》在计算环境中使用的向量（vector）和堆叠（stack）。

迪杰斯特拉的上述评论和书籍文风犀利并极具挑战性，旨在激发同行专家们的反思。他的著述因思想深度和影响力而被广泛传播。学界大辩论的结果最终成就了结构化编程的革命，在计算机科学历史上是软件开发从个人技艺走向技术设计的里程碑。

【八】图灵奖之后

迪杰斯特拉在 1972 年获得 ACM 图灵奖后，他所在的埃因霍温理工大学对他的计算机编程研究以及图灵奖并不认可，甚至解散了他的计算机研究小组。1973 年，迪杰斯特拉快快不乐地离开了大学，到了一家美国宝来公司（Burroughs Corporation）在荷兰的计算机分公司当研究员。该公司在 1886 年成立，到 1950 年代成为世界上最大的计算器生产基地。公司为迪杰斯特拉提供了非常自由的科研环境和条件，而且给予他许多出国访问的机会。1974 年，迪杰斯特拉获得了美国信息处理学会联合会（AFIPS）的 Harry Goode 纪念奖。可是到了 1980 年代，宝来公司的业务越来越不景气，迪杰斯特拉的处境渐渐变得大不如前。

1984 年，迪杰斯特拉离开了宝来公司，应聘到了美国得克萨斯大学奥斯汀分校（University of Texas at Austin）任职 Schlumberger Centennial 讲座教授，在那里工作至 1999 年退休。

在得克萨斯大学奥斯汀分校的十五年里，迪杰斯特拉一直是一位多产的研究者。他此前启动了一个长期研究项目“简化数学论证”，1990 年与物理和计算机科学家卡雷尔·舒尔腾（Carel S. Scholten，1925-2009）合著了一本关于谓词演算的专著《谓词演算和程序语义》（Predicate Calculus and Program Semantics），倡导了一种用于数学论证的“计算证明风格”。他将自己的方法应用于许多不同的领域，包括坐标几何、线性代数、图论、顺序和分布式程序设计等。

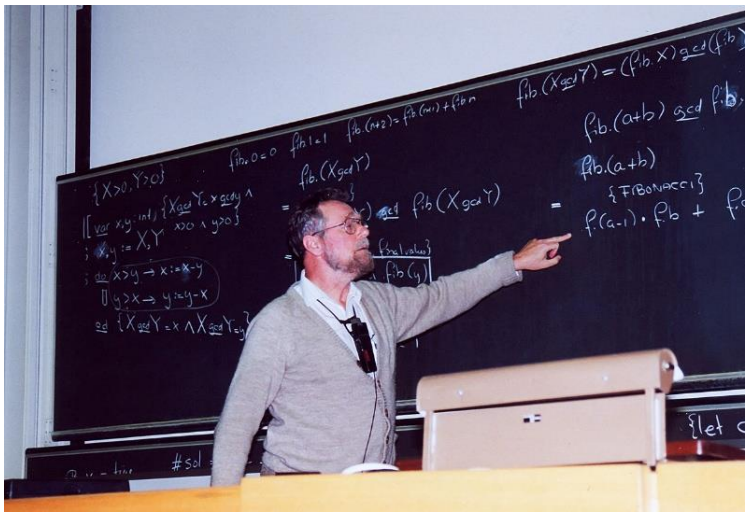


图 5 迪杰斯特拉在授课

在得克萨斯大学奥斯汀分校的教学生涯中，迪杰斯特拉以认真教书而出名。他认为自己最具价值的贡献就是指导了许多很有才华的本科生和研究生。他对年轻人做科学研究的劝诫是：“只去做只有你能做的研究”（Do only what only you can do）。他在一次题为“我对计算机科学的希望”（My hopes to computer science）的演讲中说：“对我而言，计算机科学的第一个挑战是如何把指令维持在有限个内，…… 第二个也是同样重要的挑战是如何传授解决第一个问题的方法：只培养你自己的才智是不够的，它会跟随你进入坟墓，你必须教会其他人如何去发挥他们的才智。你越关注这两个挑战，你越会清楚地看到，它们只不过是同一枚硬币的两个侧面：自学是去发现什么东西是可以被教会的。”他在奥斯汀教学期间一直使用自己 1970 年出版的《结构化编程笔记》作为教科书。该书对开发设计更完善和更系统的程序设计课程产生了很大的影响，其中他特别强调系统性的程序构建以及如何保证其正确性。

1989 年，迪杰斯特拉获得了 ACM Special Interest Group Computer Science Education（特别兴趣小组计算机科学教育）杰出贡献奖。1992 年，他被选为欧洲工程院（European Academy of Engineering）的首批院士。2001 年，他荣获希腊雅典经济商业大学的荣誉博士学位。此前，他已被选为荷兰艺术与科学院院士（1971）和美国艺术与科学院外籍院士（1975），并获得北爱尔兰皇后大学荣誉博士学位（1976）。

2002 年二月，迪杰斯特拉不幸被查出患上癌症。四月，他和夫人丽娅从美国折回荷兰家乡尼厄嫩（Nuenen）市居住。8 月 6 日，迪杰斯特拉不治去世，享年 72 岁。

迪杰斯特拉去世后还荣获 2002 年 ACM Principles of Distributed Computing （分散式计算原理）最具影响力论文奖，表彰他对编程计算自稳定的贡献。为了纪念他，ACM 这个大奖后来更名为“ACM 迪杰斯特拉奖”（Edsger W. Dijkstra Prize in Distributed Computing）。为了纪念他，得克萨斯大学奥斯汀分校也开设了一个名为“迪杰斯特拉纪念讲座”的科学报告会系列。

【九】名人轶事

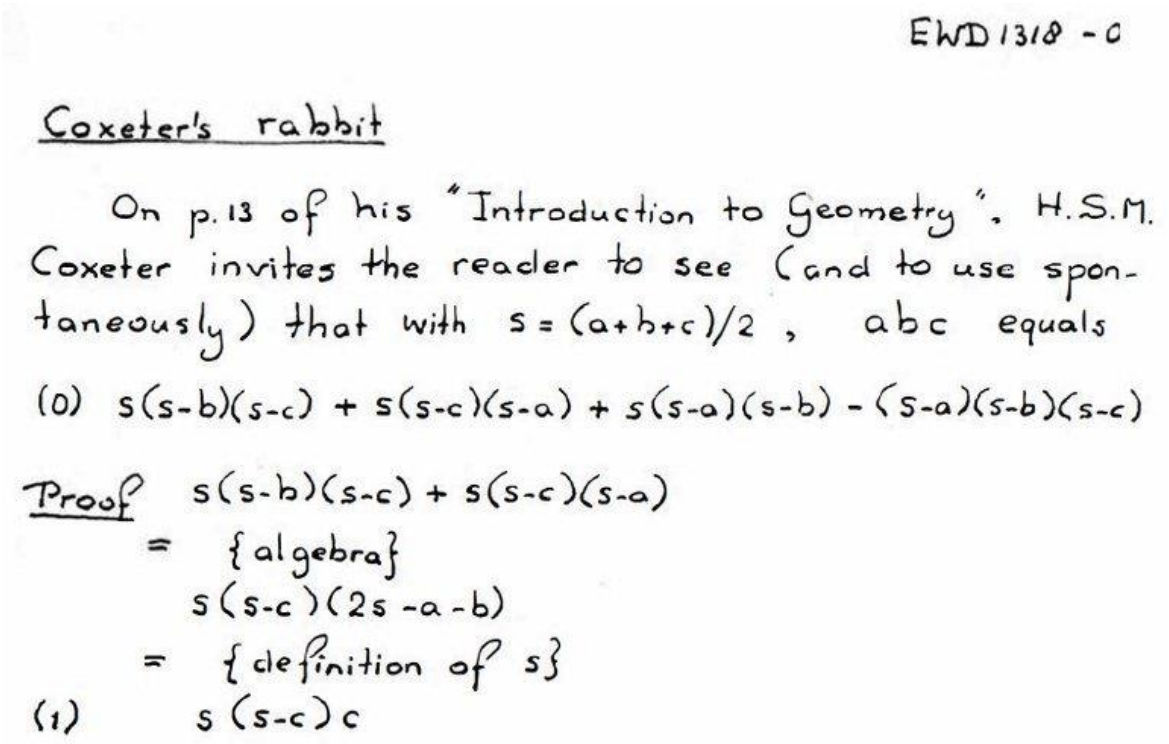
迪杰斯特拉个性极强，经常与同事和朋友争辩学术问题，特别是关于他提出的各种新概念和新技术。他在坊间留下了一些很有趣的轶事。

图 6 个性极强的迪杰斯特拉

迪杰斯特拉从来都不使用秘书，所有和研究相关的工作都自己去完成。他从来都不申请科研基金，只有一次例外：为了资助一个博士生。他家里从来都没有电视机或录影机，但有一部常不离手的《牛津英语词典》。他从来都不用手提电话也不去戏院看电影。不过，他欣赏古典音乐特别是莫扎特，也会弹钢琴。

他的幽默感既简洁又犀利。曾有一位好友问他一生带过多少个博士学生？他回答说："两个"，接着补充道："爱因斯坦一个也没有呢。"不过，如果算上联合培养的学生，他其实有七个博士学生。

他最有趣的轶事是：作为一位计算机科学家，他多年来经常与数百名朋友和同事通信交流，但却不是使用电子邮件而是通过传统的邮寄方式。他几乎从来都不用计算机去打字，而毕生保持着传统习惯——用钢笔书写。他的手稿字迹极其工整，几乎没有涂改的痕迹。他的长子马库斯在写给一个朋友的信中回忆道："由于长时间书写导致右手腕（手臂？）出现问题，父亲他决定也学习用左手书写。几个月后他便学会了，并且坚持经常用左手书写，直到他的右臂恢复正常。"迪杰斯特拉做事更像计算机执行程序一样，极有条理。他用自己名字的缩写 EWD 对所有写下的文稿进行了编号，从 EWD1 开始，到去世前四个月签名留下的最后一篇 EWD1318。这些手稿共约 7700 页，几乎全是他一个人署名的论文和算稿，现在都保存在得克萨斯大学奥斯汀分校的美国历史中心（The Center for American History）档案馆里。



EWD 1318 - 0

Coxeter's rabbit

On p. 13 of his "Introduction to Geometry", H.S.M. Coxeter invites the reader to see (and to use spontaneously) that with $s = (a+b+c)/2$, abc equals

(0) $s(s-b)(s-c) + s(s-c)(s-a) + s(s-a)(s-b) - (s-a)(s-b)(s-c)$

Proof $s(s-b)(s-c) + s(s-c)(s-a)$

$= \{\text{algebra}\}$

$s(s-c)(2s-a-b)$

$= \{\text{definition of } s\}$

(1) $s(s-c)c$

图 7 迪杰斯特拉最后一篇手稿 EWD1318 中的一页