2.6. ИСТОРИЯ И БИБЛИОГРАФИЯ

Линейные списки и прямоугольные массивы информации, содержащиеся в последовательных ячейках памяти, широко использовались с первых дней появления компьютеров с хранимой программой, и в самых ранних работах по программированию приведены базовые алгоритмы для прохождения этих структур. [См., например, J. von Neumann, *Collected Works* **5**, 113–116 (написана в 1946 году); M. V. Wilkes, D. J. Wheeler, S. Gill, *The Preparation of Programs for an Electronic Digital Computer* (Reading, Mass.: Addison-Wesley, 1951), subroutine V-1. Обратите особое внимание на работу Конрада Зусе (Konrad Zuse) *Berichte der Gesellschaft für Mathematik und Datenverarbeitung* **63** (Bonn, 1972), написанную в 1945 году. Зусе первым создал нетривиальные алгоритмы для работы со списками динамически изменяемой длины.] До появления индексных регистров работа с последовательными линейными списками выполнялась при помощи арифметических операций над самими машинными командами, и использование такой арифметики стало одним из самых ранних обоснований создания компьютеров, в которых программа разделяла с обрабатываемыми данными общее пространство памяти.

Технологии, позволяющие линейным спискам переменной длины занимать последовательные адреса памяти таким образом, чтобы при необходимости их можно было сдвигать в ту или иную сторону, как описано в разделе 2.2.2, были, повидимому, гораздо более поздним открытием. Д. Данлэп (J. Dunlap) из Digitek Corporation разработал такие технологии до 1963 года в связи с созданием ряда компилирующих программ. Примерно в то же время данная идея независимо возникла при создании компилятора COBOL в IBM Corporation и набора связанных с ним подпрограмм, называвшегося CITRUS, который впоследствии использовался в различных компьютерах. Технологии оставались неопубликованными до тех пор, пока не были независимо разработаны Яном Гарвиком (Jan Garwick) из Норвегии; см. *BIT* **4** (1964), 137–140.

Идея размещения линейного списка *не* в последовательных ячейках памяти, видимо, появилась в связи с разработкой компьютеров с устройствами хранения информации на магнитных барабанах. После выполнения команды из ячейки n такой компьютер обычно не был готов к работе с командой из следующей ячейки $n + 1$, так как барабан уже прошел эту точку. В зависимости от выполняющейся команды наиболее благоприятным местом расположения следующей команды может оказаться, например, ячейка $n + 7$ или $n + 18$, и при оптимальном размещении команд быстродействие машины может увеличиться в 6–7 раз. [Обсуждение интересных задач, относящихся к оптимальному расположению команд, можно найти в статье автора этой книги в *JACM* **8** (1961), 119–150.] Поэтому в каждой машинной команде появлялось дополнительное адресное поле, служившее связью со следующей командой. Такая идея, названная "адресация 1+1", обсуждалась в работе John Mauchly, *Theory and Techniques for the Design of Electronic Computers* **4** (U. of Pennsylvania, 1946), Lecture 37. В ней в зачаточной форме содержится понятие связанного списка, хотя так часто использовавшиеся нами в этой главе операции динамической вставки и удаления оставались неизвестными. Другое раннее упоминание о связях в программах приведено в меморандуме Г. П. Лана (H. P. Luhn) (1953 г.), в котором для внешнего поиска предлагалось применять "цепочки" (см. раздел 6.4).

Реально технологии использования связанной памяти родились, когда А. Ньювелл (A. Newell), Д. К. Шоу (J. C. Shaw) и Г. А. Саймон (H. A. Simon) начали свои исследования эвристического решения задач с помощью компьютера. Весной 1956 года для написания программ поиска доказательств в математической логике они разработали первый язык обработки списков — IPL-II. (IPL означает Information Processing Language — язык обработки информации.) Это была система, использующая указатели и включающая важные концепции наподобие списка свободного пространства, однако концепция стека тогда еще не была достаточно разработана. Созданный годом позже IPL-III включал команды для помещения в стек и выборки из стека в качестве важных основных операций. [Подробнее о IPL-II речь идет в *IRE Transactions* **IT-2** (September, 1956), 61–70; *Proc. Western Joint Comp. Conf.* **9** (1957), 218–240. Материалы по IPL-III впервые появились в виде конспекта лекций в Университете штата Мичиган летом 1957 года.]

Работа Ньювелла, Шоу и Саймона привлекла многих исследователей к использованию связанной памяти, которую в то время часто называли по первым буквам фамилий авторов NSS-памятью, но, в основном, для решения задач, связанных с моделированием человеческого мышления. Постепенно эти технологии стали базовыми инструментами программирования. Первая статья, описывающая эффективность связанной памяти для выполнения "приземленных" задач, была опубликована Д. В. Карром III (J. W. Carr III) в *CACM* **2**, 2 (February, 1959), 4–6. В ней Карр указал, что связанными списками легко манипулировать на обычных языках программирования без привлечения изощренных подпрограмм или интерпретирующих систем. [См. также G. A. Blaauw, "Indexing and control-word techniques", *IBM J. Res. and Dev.* **3** (1959), 288–301.]

Сначала для связанных таблиц использовались узлы из одного слова, но около 1959 года постепенно различные коллективы убедились в преимуществах узлов из нескольких последовательных слов и "многосвязных" списков. Первой статьей, посвященной сугубо этой идее, была статья D T. Ross, *CACM* **4** (1961), 147–150. В то время для того, что в настоящей главе мы называем узлом, автором указанной статьи использовался термин "плекс" (plex), хотя позднее он употреблял это слово в другом смысле — для описания класса узлов в комбинации с алгоритмами их прохождения.

Обозначения для ссылки на поле в узле, в основном, бывают двух видов: имя поля либо предшествует обозначению указателя, либо следует за ним. Таким образом, то, что в данной главе записывается как "`INFO(P)`", некоторые авторы записывают как "`P.INFO`". Во время создания этой главы обе записи, пожалуй, встречались одинаково часто. Запись, принятая в настоящей книге, имеет то достоинство, что она непосредственно транслируется на языки FORTRAN, COBOL и подобные, если определить массивы `INFO` и `LINK` и использовать в качестве индекса `P`. Кроме того, представляется естественным использовать обозначения математической функции для описания атрибутов узла. Заметьте, что "`INFO(P)`" произносится как "`INFO` от `P`", как принято в математике — $f(x)$ вербально передается как "f от x". Альтернативное обозначение "`P.INFO`" менее естественно, поскольку имеет тенденцию к подчеркиванию `P`, хотя и может быть прочитано как "`P`-е `INFO`". Причина, по которой `INFO(P)` кажется более предпочтительным способом записи, по-видимому, заключается в том, что `P` - переменная, а `INFO` имеет конкрет-

ный смысл при использовании записи. По аналогии можно рассматривать вектор $A = (A[1], A[2], \ldots, A[100])$ как узел, имеющий 100 полей с именами $1, 2, \ldots, 100$. Теперь на второе поле в наших обозначениях можно сослаться как "2(P)", где P указывает на вектор A; но если ссылаться на j-й элемент вектора, то более естественно записать "$A[j]$", помещая переменную "j" второй. Точно так представляется более подходящей запись INFO(P) с переменной "P" на втором месте*.

Возможно, первыми, кто увидели в концепциях стека ("последним вошел — первым вышел"; last-in-first-out) и очереди ("первым вошел — первым вышел"; first-in-first-out) важные объекты изучения, были бухгалтеры, заинтересованные в упрощении процесса начисления подоходных налогов. Обсуждение методов LIFO и FIFO при составлении прайс-листов можно найти в любом учебнике среднего уровня для бухгалтеров [см., например, C. F. and W. J. Schlatter, *Cost Accounting* (New York: Wiley, 1957), Chapter 7]. В середине 40-х годов А. М. Тьюринг (A. M. Turing) разработал механизм стека, названный реверсивной памятью, для связывания подпрограмм, локальных переменных и параметров. Тьюринг использовал для принятых ныне понятий "добавление в стек" и "снятие со стека" (push/pop) понятия "закапывать" (bury) и "откапывать" (disinter/unbury) (см. ссылки в разделе 1.4.5). Несомненно, простые применения стеков, расположенных в последовательных адресах памяти, в программировании с первых дней были обычным делом, поскольку стек представляет собой интуитивно понятную и естественную концепцию. Программирование стеков в связанном виде появилось впервые в IPL, как упоминалось выше; имя "стек" происходит из терминологии IPL (хотя в IPL имелся более официальный термин "pushdown list") и независимо введено Э. В. Дейкстрой (E. W. Dijkstra) [*Numer. Math.* **2** (1960), 312–318]. Термин "дек" (deque) был введен Э. Ю. Швеппе (E. J. Schweppe) в 1966 году.

Происхождение циклических списков и списков с двойными связями не ясно. Вероятно, эти идеи многим показались естественными. Важным фактором в популяризации этих технологий было существование основанных на них систем общего назначения для обработки списков [см. Knotted List Structures, *CACM* **5** (1962), 161–165, и Symmetric List Processor, *CACM* **6** (1963), 524–544, Й. Вейзенбаума (J. Weizenbaum)]. Айвен Сэтерленд (Ivan Sutherland) ввел в использование независимые двусвязные списки в бóльших узлах в своей системе Sketchpad (Ph. D. thesis, Mass. Inst. of Technology, 1963).

С первых "компьютерных" дней многие квалифицированные программисты независимо разработали различные методы адресации и прохождения многомерных массивов информации. Таким образом была рождена еще одна часть неопубликованного компьютерного фольклора. Обзор этой темы можно найти в работе H. Hellerman, *CACM* **5** (1962), 205–207 [см. также J. C. Gower, *Comp. J.* **4** (1962), 280–286].

Структуры деревьев, явно представленные в памяти компьютера, изначально использовались для приложений, работающих с алгебраическими формулами. Ма-

* Аргументация автора безупречна, но... "меняется все в наш век перемен" и место FORTRAN и COBOL заняли языки C++, Pascal и Java (а также Visual Basic и многие другие), в которых полна смысла именно запись P.INFO — "поле INFO структуры P" (или "записи" и т. п. — название варьируется в зависимости от языка программирования). Что же касается аналогии с массивом, то в том же C или C++ на элементы массива a[] можно ссылаться и как на a[j], и как на j[a]. — *Прим. перев.*

шинный язык для ряда ранних компьютеров использовал трехадресный код для представления вычислений арифметических выражений; последний эквивалентен INFO, LLINK и RLINK бинарного представления дерева. В 1952 году Г. Д. Кахриманиан (H. G. Kahrimanian) разработал алгоритмы для дифференцирования алгебраических формул, представленных в расширенном трехадресном коде [см. *Symposium on Automatic Programming* (Washington, D.C.: Office of Naval Research, May, 1954), 6–14].

С тех пор древовидные структуры различных видов независимо изучались многими исследователями в связи с их применением во многих компьютерных приложениях. Однако в публикациях основные технологии для работы с деревьями (не со списками) появлялись лишь в составе описаний отдельных алгоритмов. Первый общий обзор был сделан в связи с изучением структур данных в работе K. E. Iverson and L. R. Johnson [IBM Corp. research reports RC-390, RC-603, 1961]; см. также Iverson, *A Programming Language* (New York: Wiley, 1962), Chapter 3, и G. Salton, *CACM* **5** (1962), 103–114.

Концепция *прошитых* (*threaded*) деревьев принадлежит А. Д. Перлису (A. J. Perlis) и Ч. Торнтону (C. Thornton) [*CACM* **3** (1960), 195–204]. В их статье вводится также важная идея прохода деревьев в различных порядках и приводятся многочисленные примеры алгоритмов алгебраических манипуляций. К сожалению, эта важная статья готовилась наскоро и содержит много опечаток. Прошитые списки Перлиса и Торнтона в нашей терминологии представляют собой правопрошитые деревья. Бинарные деревья, прошитые в *обоих* направлениях, были независимо открыты А. В. Хольтом (A. W. Holt), *A Mathematical and Applied Investigation of Tree Structures* (Thesis, U. of Pennsylvania, 1963). Прямой и непрямой порядки узлов деревьев в работе Z. Pawlak, *Colloquium on the Foundation of Mathematics*, Tihany, 1962 (Budapest: Akadémiai Kiadó, 1965), 227–238, назывались "нормальный прямой порядок" и "дуальный прямой порядок". В приведенной выше работе Айверсон и Джонсон назвали прямой порядок порядком поддеревьев. Графические способы представления связей между древовидными структурами и соответствующими линейными обозначениями были описаны в A. G. Oettinger, *Proc. Harvard Symp. on Digital Computers and their Applications* (April, 1961), 203–224. Представление деревьев в прямом порядке степенями с алгоритмами, связывающими это представление с десятичной записью Дьюи и другими свойствами деревьев, были представлены в работе S. Gorn, *Proc. Symp. Math. Theory of Automata* (Brooklyn: Poly. Inst., 1962), 223–240.

История древовидных структур как математических объектов вместе с библиографией по этой теме приведена в разделе 2.3.4.6.

В то время, когда в 1966 году создавался этот раздел, большинство сведений об информационных структурах было получено благодаря изучению систем обработки списков, игравших очень важную роль в истории. Первой широко используемой системой был язык IPL-V (наследник IPL-III, разработанный в 1959 году). Он представлял собой интерпретирующую систему, в которой программист изучал машиноподобный язык для работы со списками. Примерно в то же время Г. Гелернтером (H. Gelernter) и его сотрудниками был разработан FLPL (набор подпрограмм на языке FORTRAN для работы со списками, который также обязан своему появлению на свет IPL, но в отличие от него вместо интерпретируемого языка использовал вызовы

подпрограмм). Третья система, LISP, была также создана в 1959 году Дж. Мак-Карти (J. McCarthy). LISP существенно отличался от своих предшественников: его программы были (и остаются) математическими функциональными выражениями, скомбинированными с "условными выражениями" (conditional expressions) и конвертированными затем в представление списков. В 60-е годы возникло много систем обработки списков; с исторической точки зрения среди них наиболее известен набор подпрограмм SLIP для реализации списков с двойными связями на языке FORTRAN, созданный Й. Вейзенбаумом (J. Weizenbaum).

В статье Боброу (Bobrow) и Рафаэля (Raphael), *CACM* **7** (1964), 231–240, можно прочесть краткое введение в IPL-V, LISP и SLIP. В ней же дается сравнение этих систем. Превосходное введение в LISP было опубликовано Ф. М. Вудвордом (P. M. Woodward) и Д. П. Дженкинсом (D. P. Jenkins), *Comp. J.* **4** (1961), 47–53. Кроме того, можно ознакомиться с авторским описанием их собственных систем (эти статьи представляют значительную историческую ценность): "An introduction to IPL-V" А. Ньювелла (A. Newell) и Ф. М. Тонге (F. M. Tonge), *CACM* **3** (1960), 205–211; "A FORTRAN-compiled List Processing Language" Г. Гелернтера (H. Gelernter), Д. Р. Хансена (J. R. Hansen) и К. Л. Гербериха (C. L. Gerberich), *JACM* **7** (1960), 87–101; "Recursive functions of symbolic expressions and their computation by machine, I" Джона Мак-Карти (John McCarthy), *CACM* **3** (1960), 184–195; "Symmetric List Processor" Й. Вейзенбаума (J. Weizenbaum), *CACM* **6** (1963), 524–544. Статья Вейзенбаума включает полное описание всех использованных в SLIP алгоритмов. Из всех перечисленных ранних систем только LISP имел все необходимые составляющие для того, чтобы пережить десятилетия дальнейшего прогресса. Мак-Карти описал раннюю историю LISP в *History of Programming Languages* (Academic Press, 1981), 173–197.

В 60-е годы появился ряд систем для *работы со строками*, в которых первостепенное значение приобрели операции со строками переменной длины, содержащими алфавитную информацию — поиск вхождения в строку определенной подстроки, ее замещение другой и т. п. Наиболее важными из них с исторической точки зрения были COMIT [V. H. Yngve, *CACM* **6** (1963), 83–84] и SNOBOL [D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* **11** (1964), 21–30]. Хотя системы работы со строками широко использовались и состоят, в первую очередь, из алгоритмов, подобных рассмотренным в этой главе, они сыграли сравнительно малую роль в истории развития технологий представления информационных структур. Детали внутренней реализации были скрыты от пользователей этих систем. Обзор ранних систем работы со строками можно найти в S. E. Madnick, *CACM* **10** (1967), 420–424.

В системах обработки списков IPL-V и FLPL не использовались ни сборка мусора, ни технология счетчиков ссылок при работе с разделяемыми списками. Вместо этого каждый список "принадлежал" одному списку и "заимствовался" всеми другими списками, которые на него ссылались. Список удалялся, когда его владелец позволял это. Следовательно, программист должен был сам гарантировать, что при удалении список никому не был одолжен. Технология счетчика ссылок для списков введена Д. Э. Коллинзом (G. E. Collins), *CACM* **3** (1960), 655–657, и позже проанализирована в *CACM* **9** (1966), 578–588. Сборка мусора впервые была описана в статье Мак-Карти в 1960 году [см. также примечания Вейзенбаума в *CACM* **7** (1964), 38, и статью Коэна (Cohen) и Триллинга (Trilling), *BIT* **7** (1967), 22–30].

Рост понимания важности работы со связями естественным образом привел к их включению в алгебраические языки программирования, созданные после 1965 года. Новые языки позволяли программистам выбрать подходящую форму представления данных, не прибегая к ассемблеру и избегая накладных расходов универсальных структур списков. Некоторыми из фундаментальных шагов на этом пути были работы N. Wirth and H. Weber, *CACM* **9** (1966), 13–23, 25, 89–99; H. W. Lawson, *CACM* **10** (1967), 358–367; C. A. R. Hoare, *Symbol Manipulation Languages and Techniques*, ed. by D. G. Bobrow (Amsterdam: North-Holland, 1968), 262–284; O.-J. Dahl and K. Nygaard, *CACM* **9** (1966), 671–678; A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster, *Numerische Math.* **14** (1969), 79–218; Dennis M. Ritchie, *History of Programming Languages—II* (ACM Press, 1996), 671–698.

Алгоритмы динамического выделения памяти появились за несколько лет до их описания в печати. Очень интересное обсуждение было подготовлено В. Т. Комфортом (W. T. Comfort) в 1961 году и опубликовано в *CACM* **7** (1964), 357–362. Метод граничного дескриптора, описанный в разделе 2.5, был разработан автором книги в 1962 году для использования в операционной системе компьютера Burroughs B5000. Система двойников впервые использовалась Г. Марковицем (H. Markowitz) в системе программирования SIMSCRIPT в 1963 году, а также была независимо разработана и опубликована К. Ноултоном (K. Knowlton), *CACM* **8** (1965), 623–625 [см. также *CACM* **9** (1966), 616–625]. Дополнительную информацию о динамическом выделении памяти можно найти в статьях Iliffe and Jodeit, *Comp. J.* **5** (1962), 200–209; Bailey, Barnett, and Burleson, *CACM* **7** (1964), 339–346; A. T. Berztiss, *CACM* **8** (1965), 512–513; D. T. Ross, *CACM* **10** (1967), 481–492.

Общее обсуждение информационных структур и их связи с программированием было подготовлено Мэри д'Империо (Mary d'Imperio) — "Data Structures and their Representation in Storage", *Annual Review in Automatic Programming* **5** (Oxford: Pergamon Press, 1969). Ее статья служит серьезным путеводителем по истории предмета, поскольку включает детальный анализ структур, использованных в двенадцати системах обработки списков и работы со строками. Дополнительные исторические детали можно найти в трудах двух симпозиумов, *CACM* **3** (1960), 183–234, и *CACM* **9** (1966), 567–643 (отдельные статьи из этих трудов упоминались ранее).

Великолепно аннотированная библиография ранних работ по манипуляции символами и математическими формулами, имеющая множество связей с материалом этой главы, была составлена Д. Э. Саммет (J. E. Sammet); см. *Computing Reviews* **7** (July–August, 1966), B1–B31.

В настоящей главе очень подробно рассматривались некоторые типы информационных структур, и (чтобы за деревьями суметь рассмотреть лес), вероятно, будет разумно проанализировать, что нами изучено, и подвести краткий итог по теме "информационные структуры" с точки зрения более широкой перспективы. Начав с базовой идеи *узла* (*node*) как элемента данных, можно найти множество примеров, иллюстрирующих удобные пути представления структурных отношений либо неявно (основываясь на относительном порядке, в котором узлы хранятся в памяти компьютера), либо явно (с помощью указывающих на другие узлы связей в узлах). Объем структурной информации, которая должна быть представлена в таблицах компьютерных программ, зависит от операций, которые должны выполняться над узлами.

Из методических соображений мы, в основном, сконцентрировали свое внимание на связях между информационными структурами и их компьютерным представлением, а не рассматривали эти вопросы по отдельности. Однако для углубленного понимания полезно рассмотреть предмет с более абстрактной точки зрения, отделив идеи, которые могут быть изучены самостоятельно. Был разработан ряд достойных внимания подходов такого типа; особо могут быть рекомендованы для ознакомления следующие ранние работы: G. Mealy, "Another look at data", *Proc. AFIPS Fall Joint Computer Conf.* **31** (1967), 525–534; J. Earley, "Toward an understanding of data structures", *CACM* **14** (1971), 617–627; C. A. R. Hoare, "Notes on data structuring", in *Structured Programming* by O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare (Academic Press, 1972), 83–174; Robert W. Engles, "A tutorial on data-base organization", *Annual Review in Automatic Programming* **7** (1972), 3–63.

Материал этой главы не охватывает предмет информационных структур во всей полноте. Как минимум не рассматривались три важных вопроса.

a) Часто необходимо найти узел (или множество узлов, обладающих некоторым значением) в таблице. Такая операция нередко оказывает серьезное влияние на структуру таблицы. Этот вопрос детально описывается в главе 6.

b) В основном, нами рассматривалось внутреннее представление структур в компьютере; однако это всего лишь одна часть темы, поскольку структуры должны быть также представлены как внешние входные или выходные данные. В простых случаях с внешними структурами можно работать, по сути, при помощи тех же технологий, которые были здесь рассмотрены. Однако очень важен процесс преобразования строк символов в более сложные структуры и обратно. Такие процессы анализируются в главах 9 и 10.

c) В основном, в главе обсуждалось представление структур данных в высокоскоростной памяти с произвольным доступом. При использовании медленных устройств, таких как диски и ленты, все структурные проблемы обостряются и все более важными становятся эффективные алгоритмы и эффективные схемы представления данных. Узлы, связанные один с другим, в этих случаях должны располагаться в близких областях памяти. Обычно эти проблемы трудно обсуждать в общем виде, так как они сильно зависят от характеристик конкретных машин. Простейшие примеры из этой главы должны помочь читателю подготовиться к решению сложных проблем, возникающих в связи с использованием менее идеальных устройств хранения информации. Некоторые из этих проблем детально обсуждаются в главах 5 и 6.

В чем же заключается смысл рассматриваемых в этой главе тем? Видимо, наиболее важный вывод, который можно сделать, изучив предложенный материал, состоит в том, что рассмотренные идеи не ограничиваются компьютерным программированием; в целом, они вполне применимы в повседневной жизни. Коллекция узлов, содержащих поля, одни из которых указывают на другие узлы, представляется очень хорошей абстрактной моделью структурных отношений любого вида. Такая модель показывает, каким образом можно построить сложные структуры из простых, а соответствующие алгоритмы для работы со структурами могут быть разработаны естественным образом.

Поэтому представляется разумным продолжить разработку теории о связанных множествах узлов по сравнению с тем, что известно в настоящее время. Возможно,

наиболее очевидный путь создания такой теории состоит в определении нового вида абстрактной машины или "автомата" для работы со связанными структурами. Например, подобное устройство может быть неформально определено следующим образом. Имеются числа k, l, r и s, такие, что автомат обрабатывает узлы, содержащие k полей связи и r информационных полей. Он имеет l регистров связи и s информационных регистров, обеспечивающих управление выполняемыми процессами. Информационные поля и регистры могут содержать любые символы из некоторого заданного набора информационных символов. Каждое из полей связи и регистров связи содержит либо Λ, либо указатель на узел. Машина может (i) создавать новые узлы (помещая связи с узлом в регистр), (ii) проверять равенство символов или значений связей и (iii) переносить информационные символы или значения связей между регистрами и узлами. Непосредственно доступны только узлы, на которые указывают регистры связей. Соответствующие ограничения на поведение машины делают ее эквивалентом ряда других видов автоматов.

Аналогичная модель вычислений была предложена А. Н. Колмогоровым в 1952 году. Его машина, по сути, работала с графом G, имеющим специально созданную стартовую вершину v_0. Действия на каждом шаге зависят только от подграфа G', состоящего из всех вершин на расстоянии $\leq n$ от v_0 в G, и заключаются в замещении G' в G другим графом $G'' = f(G')$, где G'' включает v_0 и вершины на расстоянии, в точности равном n от v_0, и, возможно, другие (вновь созданные) вершины. Остаток графа G является неизменным. Здесь n — фиксированное число, определяющееся отдельно для каждого конкретного алгоритма, которое может быть произвольно большим. Каждой вершине назначается символ из некоторого конечного алфавита с тем ограничением, что у вершины не может быть двух соседних вершин с одинаковыми символами. (См. А. Н. Колмогоров, *Успехи мат. наук* **8**, 4 (1953), 175–176; Колмогоров и Успенский, *Успехи мат. наук* **13**, 4 (1958), 3–28; *Amer. Math. Soc. Translations*, series 2, **29** (1963), 217–245.)

Связывающие автоматы могут легко моделировать машины графов, используя ограниченное сверху количество шагов на один шаг работы графа. Напротив, маловероятно, чтобы машины графов могли моделировать произвольные связывающие автоматы без неограниченного увеличения времени работы, если не перейти от неориентированных графов к ориентированным, чтобы работать с вершинами с ограниченной степенью. И, конечно, связывающая модель гораздо ближе к операциям, доступным программисту на реальной машине, в отличие от модели с использованием графа.

Самыми интересными проблемами, которые предстоит решить для такого рода устройств, являются определение скорости решения поставленных ими задач, т. е. подсчет количества узлов, требуемых для решения той или иной задачи (например, для трансляции какого-либо формального языка). В то время, когда автор приступил к работе над настоящей главой, были получены интересные результаты такого рода (отметим работы Ю. Хартманиса (J. Hartmanis) и Р. Э. Стирнса (R. E. Stearns)), но только для специальных классов машин Тьюринга с множеством лент и головок чтения/записи. Модель машины Тьюринга сравнительно нереальна, а потому полученные результаты имеют мало общего с решением практических задач.

Следует признать, что при стремлении количества созданных связывающим автоматом узлов n к бесконечности неизвестно, как построить такое устройство фи-

зически, поскольку желательно, чтобы операции машины выполнялись за одно и то же время независимо от размера n. Если связывание представлено с использованием адресов в машинной памяти, необходимо определить границу для количества узлов, поскольку поля связей имеют фиксированный размер. Многоленточная машина Тьюринга поэтому представляет собой более реалистичную модель при стремлении n к бесконечности. Представляется также обоснованной уверенность в том, что описанные выше связывающие автоматы приведут к созданию более приемлемой теории сложности алгоритмов, чем машина Тьюринга, даже при рассмотрении асимптотических формул для больших n, так как эта теория больше подходит для практических значений n. Кроме того, когда n становится больше, чем 10^{30} или около того, даже одноленточная машина Тьюринга не является реалистичной: она никогда не будет построена. Принципы важнее реалий.

Со времени первого написания автором большинства из приведенных выше комментариев утекло много воды, и можно порадоваться, что в теории связывающих автоматов (сегодня называемых *машинами указателей* (*pointer machines*)) достигнут определенный прогресс, хотя, конечно же, предстоит еще немало сделать в этой области.

Разработаны общие принципы программирования.
Большинство из них давно используется
на станции Канзас-Сортировочная...

— ДЕРРИК ЛЕМЕР (DERRICK LEHMER) (1949)

Я уверен, вы согласитесь со мной ... что если
со страницей 534 мы встречаемся только во второй главе,
то первая глава должна быть невыносимо длинной.

— ШЕРЛОК ХОЛМС (SHERLOCK HOLMES), *Аллея страха* (*The Valley of Fear*) (1888)