Будущим поколениям

Оглавление

Предисловие

Рекурсия – одно из самых фундаментальных понятий в информатике и ключевая методика программирования, позволяющая, подобно итерации, многократно повторять вычисления. Это достигается за счёт использования методов, вызывающих самих себя, когда решение исходной задачи сводится к решению нескольких экземпляров той же самой задачи, но меньшего размера. Крайне важно то, что рекурсия – мощный подход к решению задач, позволяющий программистам разрабатывать лаконичные, интуитивно понятные и изящные алгоритмы.

Несмотря на значимость рекурсии для создания алгоритмов, большинство книг по программированию не уделяет внимания её деталям. Обычно они посвящают ей лишь одну-единственную главу или коротенький раздел, которых зачастую недостаточно для освоения понятий, необходимых для овладения предметом. Исключениями являются книги [11], [14], [15], посвящённые исключительно рекурсии. Данная книга также рассматривает рекурсию со всех сторон, но несколько отличается от предыдущих.

Многие преподаватели программирования и исследователи в области обучения информатике признают, что рекурсия трудна для студента-новичка. С учётом этого книга содержит несколько элементов, усиливающих ее педагогический эффект. Во-первых, перед погружением в более сложный материал она предлагает множество простых задач для закрепления основных понятий. Кроме того, одно из основных достоинств книги – использование пошаговой методики и специально созданных схем, сопровождающих и иллюстрирующих процесс разработки рекурсивных алгоритмов. Книга также содержит специальные главы по комбинаторным задачам и взаимной рекурсии. Эти темы позволяют расширить понимание рекурсии студентами, побуждая их применять освоенные понятия несколько иначе или более изощрённо. Наконец, вводные курсы программирования обычно сосредоточиваются на императивной парадигме программирования, когда студенты изучают прежде всего итерацию, усваивая то и овладевая тем, *как* работают программы. Рекурсия же подразумевает совсем иной способ мышления, когда упор делается на то, *что* вычисляют программы. В связи с этим некоторые исследователи призывают при объяснении рекурсии не раскрывать или откладывать на потом механизмы работы рекурсивных

программ (поток управления, деревья рекурсии, программный стек или связь между итерацией и хвостовой рекурсией), так как усвоенные идеи и приобретённые навыки итеративного программирования могут существенно помешать овладению рекурсией и декларативным программированием. По этой причине темы, связанные с итерацией и исполнением программы, раскрываются в конце книги, когда читатель уже овладел разработкой рекурсивных алгоритмов с чисто декларативных позиций.

В книге также есть большая глава по теоретическому анализу стоимости вычислений рекурсивных программ. С одной стороны, она содержит обширный материал о рекуррентных соотношениях – основном математическом инструменте анализа рекурсивных алгоритмов и времени их выполнения. С другой стороны, она содержит раздел с предварительными математическими сведениями, в которых даётся обзор понятий и свойств, необходимых не только для решения рекуррентных соотношений, но и для понимания условий и решений вычислительных задач этой книги. В связи с этим раздел предоставляет ещё и возможность попутно изучить или вспомнить немного элементарной математики. Желательно, чтобы читатель изучил этот материал, так как он важен во многих областях информатики.

Примеры кода написаны на языке Python 3, который сегодня является, видимо, самым популярным языком для вводных курсов программирования в ведущих университетах. Все они были проверены в основном в SPYDER (Scientific PYthon Development EnviRonment – научная среда разработки на языке Python). Читатель должен понимать, что цель книги не в изучении языка Python, а в овладении при решении задач навыками рекурсивного мышления. Поэтому такие аспекты, как простота и удобочитаемость кода, были важнее его эффективности. По этой причине код не содержит расширенных возможностей языка Python. Так что студенты, знакомые с такими языками программирования, как C++ или Java, должны без усилий понять этот код. Конечно, методы из данной книги могут быть реализованы различными способами, и читатели вольны писать более эффективные версии с включением более сложных конструкций языка Python или разрабатывать альтернативные алгоритмы. Наконец, в книге приводятся рекурсивные варианты итерационных алгоритмов, которые обычно сопутствуют другим хорошо известным рекурсивным задачам. Например, в книге приводятся рекурсивные версии метода разделения Хоара, используемого в алгоритме быстрой сортировки, или метода слияния в алгоритме сортировки слиянием.

В конце каждой главы предлагаются многочисленные упражнения, полные решения которых включены в руководство преподавателя, доступное на официальном веб-сайте книги (см. www.crcpress.com). Многие из них связаны с задачами из основного текста книги, что делает их подходящими кандидатами для экзаменов и заданий.

Код из данного текста также будет доступен для загрузки на веб-сайте книги. Кроме того, я буду поддерживать дополнительный веб-сайт книги *https://sites.google.com/view/recursiveprogrammingintro/*. Буду более чем признателен читателям за присланные комментарии, предложения по усовершенствованию, альтернативный (более ясный или эффективный) код, версии на других языках программирования или обнаруженные опечатки. Письма присылайте, пожалуйста, по адресу: *recursion.book@gmail.com*.

Целевая аудитория

Основная цель книги – на множестве примеров разнообразных вычислительных задач научить студентов думать и программировать рекурсивно. Она предназначена главным образом для студентов, обучающихся информатике или связанным с ней техническим дисциплинам, которые охватывают программирование и алгоритмы (например, биоинформатика, инжиниринг, математика, физика и т. д.). Книга может быть также полезна для программистов-любителей, студентов массовых открытых сетевых курсов или более опытных профессионалов, которые хотели бы освежить знакомый им материал или взглянуть на него иначе, проще.

Чтобы понять код в книге, студенты должны владеть некоторыми базовыми навыками программирования. Читатель должен быть знаком с такими понятиями базового курса программирования, как выражения, переменные, условные и циклические конструкции, методы, параметры и элементарные структуры данных (массивы или списки). Эти понятия не объясняются в книге. Кроме того, код в книге следует процедурной парадигме программирования и не использует объектно-ориентированные средства. Что касается языка Python, то знание его основ может быть полезно, но совсем не обязательно. Наконец, студент должен владеть математикой в объёме средней школы.

Книга также может оказаться полезной и для преподавателей информатики, причём не только как справочник с большой коллекцией разнообразных задач, но и, принимая во внимание описанные методику и схемы, как пособие по разработке рекурсивных решений. Более

того, преподаватели могут использовать её структуру для организации своих занятий. Книга могла бы использоваться как приемлемый учебник по вводному (CS1/2) курсу программирования, в углублённых курсах – по разработке и анализу алгоритмов (она, например, охватывает такие темы, как «разделяй и властвуй» или перебор с возвратами). Кроме того, поскольку книга закладывает прочный фундамент рекурсии, она может использоваться в качестве дополнительного материала в курсах, посвящённых структурам данных или функциональному программированию. Однако читатель должен иметь в виду, что книга не касается ни структур данных, ни понятий функционального программирования.

Содержание и структура книги

Глава 1 предполагает, что читатель не имеет никакого представления о рекурсии, и вводит основные её понятия, систему обозначений и даёт первые примеры рекурсивного кода.

Глава 2 описывает методику разработки рекурсивных алгоритмов, а также схем, способствующих рекурсивному мышлению, которые иллюстрируют исходную задачу и её декомпозицию (разложение) на меньшие экземпляры той же самой задачи. Это одна из самых важных глав, так как методика и рекурсивные схемы будут использоваться во всей остальной части книги. Желательно, чтобы читатели ознакомились с этой главой независимо от их предыдущих знаний о рекурсии.

Глава 3 даёт обзор основных математических принципов и обозначений. Кроме того, она описывает методы решения рекуррентных соотношений, которые являются основным математическим инструментом для теоретического анализа стоимости вычислений рекурсивных алгоритмов. Глава может быть опущена во вводных курсах по рекурсии. Однако она помещена в начало книги, чтобы заложить почву для оценки и сравнения эффективности различных алгоритмов, что важно в расширенных курсах по разработке и анализу алгоритмов.

Глава 4 посвящена «линейной рекурсии». Этот тип рекурсии приводит к простейшим рекурсивным алгоритмам, когда решение исходной вычислительной задачи вытекает из решения единственного её меньшего экземпляра (подзадачи). Хотя предлагаемые задачи можно легко решить посредством итерации, они идеально подходят для введения основных понятий рекурсии, а также в качестве примеров применения рекурсивной методики и рекурсивных схем.

Глава 5 посвящена особому типу линейной рекурсии, называемой «хвостовой рекурсией», когда рекурсивный вызов методом самого себя является последним действием этого метода. Особенность хвостовой рекурсии заключается в том, что она тесно связана с итерацией. Однако объяснение этой связи будет отложено до главы 11. А эта глава, напротив, посвящена решениям, основанным на чисто декларативном подходе с опорой исключительно на понятие рекурсии.

Преимущества рекурсии над итерацией проявляются главным образом в случае «множественной рекурсии», когда методы могут вызывать себя несколько раз, а алгоритмы основаны на объединении нескольких решений меньших экземпляров той же самой задачи. *Глава 6* вводит множественную рекурсию методами, основанными на известной парадигме разработки алгоритмов «разделяй и властвуй». Хотя часть примеров этой главы может использоваться во вводном курсе программирования, глава в целом уместна скорее для углублённых курсов и более сложных классов алгоритмов.

Глава 7 содержит дополнительные развивающие задачи, относящиеся к пазлам и фракталам. Они также решаются с помощью множественной рекурсии, но не предполагают применения подхода «разделяй и властвуй».

Рекурсия широко используется в комбинаторике – разделе математики, относящемся к подсчёту и применяемом в углублённом анализе алгоритмов. *Глава 8* предлагает использовать рекурсию для решения комбинаторных задач подсчёта, которые обычно не ограничиваются программированием текстов. Эта уникальная глава заставит читателя применять приобретённые навыки рекурсивного мышления к разнообразным семействам задач. Наконец, хотя некоторые примеры относятся к развивающим, многие из их решений появляются в ранних главах. Поэтому такие примеры могут использоваться и во вводном курсе программирования.

Глава 9 вводит понятие «взаимной рекурсии», когда несколько методов вызывает себя косвенно. Их решения гораздо сложнее, так как необходимо думать о нескольких задачах сразу. Тем не менее этот тип рекурсии опирается на те же основные понятия из предыдущих глав.

Глава 10 посвящена низкоуровневым аспектам рекурсивных программ. Она включает такие их аспекты, как трассировка и отладка, стек программы или деревья рекурсии. Кроме этого, она содержит краткое введение в мемоизацию (memorization) и динамическое программирование, которые являются ещё одной важной парадигмой разработки алгоритмов.

К итерации можно привести не только алгоритмы с хвостовой рекурсией; некоторые из них также разрабатываются итеративно. *Глава 11* подробно исследует связь между итерацией и хвостовой рекурсией. Кроме того, она содержит краткое введение во «вложенную рекурсию» и включает стратегию разработки простых функций с хвостовой рекурсией, которые обычно определяются итеративно, но с исключительно декларативным подходом. Эти две последние темы можно отнести к курьёзам рекурсии и опустить во вводных курсах.

Последняя *глава 12* посвящена перебору с возвратами (backtracking), который является ещё одним важным методом проектирования алгоритмов, применяющимся для поиска решения вычислительных задач в больших дискретных пространствах состояний. Такая стратегия обычно используется для решения задач соблюдения (удовлетворения) заданных ограничений (constraint satisfaction) и дискретной оптимизации. Например, в главе рассматриваются такие классические задачи, как расстановка на шахматной доске *N* ферзей, поиск пути в лабиринте, решение судоку или укладка рюкзака 0–1.

Примерные учебные курсы

Можно охватить лишь несколько глав. Для вводных курсов программирования достаточно глав 1, 2, 4, 5. Преподаватель должен решить, включать ли примеры глав 6–9 и освещать ли первый раздел главы 11.

Если студенты уже владеют навыками разработки линейно-рекурсивных методов, то расширенный курс по анализу и разработке алгоритмов мог бы включать главы 2, 3, 5, 6, 7, 9, 11 и 12. Тогда как главы 1 и 4 можно рекомендовать лишь для чтения, чтобы освежить знакомый материал.

В обеих из этих учебных программ глава 8 является необязательной. Наконец, вслед за пройденными главами важно ознакомиться и с главой 11.

Благодарности

Содержание этой книги использовалось в курсах обучения программированию в Университете Рея Хуана Карлоса в Мадриде (Испания). Я благодарен студентам за их отклики и предложения. Также я хотел бы поблагодарить Анхеля Веласкеса и членов исследовательской группы LITE (Лаборатория информационных технологий в образовании) за

полезные замечания по содержанию книги. Ещё хотел бы выразить благодарность Луису Фернандесу (Luís Fernández), профессору информатики Политехнического университета Мадрида, за его советы и опыт по обучению рекурсии. Особая благодарность – Джерту Лэнкриту и сотрудникам Лаборатории компьютерного слуха в Университете Калифорнии, Сан-Диего.

Мануэль Рубио-Санчес,
июль 2017

Глава 1

Основные понятия рекурсивного программирования

Итерация – от человека, рекурсия – от Бога.

– Лоуренс Питер Дейч

Рекурсия – широкое понятие, которое используется в таких разных дисциплинах, как математика, биоинформатика или лингвистика, и присутствует даже в искусстве и в природе. В программировании рекурсия понимается как мощная стратегия, позволяющая разрабатывать простые, компактные и изящные алгоритмы решения вычислительных задач. В данной главе вводятся терминология, обозначения и фундаментальные понятия рекурсии, которые в дальнейшем будут раскрыты в этой книге.

1.1. Распознание рекурсии

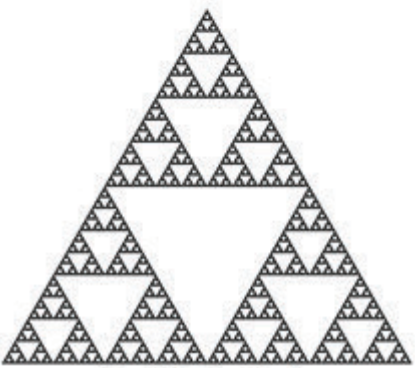
Говорят, что объект или понятие рекурсивны, когда в его состав входят более простые или меньшие подобные ему элементы. Природа даёт множество примеров этого свойства (см. рис. 1.1). Например, ветку дерева можно считать основой для меньших веток, которые отходят от неё и, в свою очередь, состоят из других, меньших, веток, и так далее до почки, листа или цветка. Строение кровеносных сосудов или рек тоже подобно структуре ветвления деревьев, когда бóльшая структура содержит подобные ей элементы, но в меньших масштабах. Другой родственный рекурсивный пример – капуста романеско (Romanesco broccoli), где отдельные маленькие цветки явно напоминают всё растение. Другие примеры включают горные цепи, облака или рисунок кожи животных.

Ветви дерева

Реки с притоками

Брокколи Romanesco

Спиральный эффект Дросте



Треугольник Серпиньского

Кукла-матрёшка

Рис. 1.1. Примеры рекурсивных объектов

Рекурсия также появляется в искусстве. Известный пример – эффект Дросте (Droste-effect), который представляет собой картинку, повторяю-

щуюся внутри себя. Теоретически такой процесс бесконечен, но на практике он, конечно же, заканчивается, когда наименьшая картинка становится настолько малой, что занимает всего один пиксель в цифровом изображении. Создаваемый компьютером фрактал – ещё один тип рекурсивного изображения. Например, треугольник Серпиньского состоит из трёх меньших идентичных треугольников, которые затем делятся ещё на три меньших. Бесконечно повторяя этот процесс, мы обнаружим, что каждый маленький треугольник имеет ту же структуру, что и оригинал. И последний, классический пример иллюстрации рекурсии – набор кукол-матрёшек. В этом ремесле каждая кукла имеет такой размер, чтобы уместиться внутри большей куклы. Отметим, что рекурсивный объект – это не одна полая кукла, а вся вложенная коллекция кукол. Таким образом, рекурсивное мышление предполагает, что вся коллекция кукол может быть описана как одна-единственная (наибольшая) кукла, которая содержит внутри себя меньшую коллекцию кукол.

Хотя в приведённых примерах рекурсивные объекты – явно материальные, рекурсия встречается и в огромном числе абстрактных понятий. В этом отношении рекурсию можно понимать как процесс определения понятий через их собственные определения. Таким способом можно выразить многие математические формулы и определения. Самые очевидные примеры – последовательности, n-й член которых задаётся некоторой формулой или процедурой, использующей предыдущие члены. Рассмотрим следующее рекурсивное определение:

$$s_n = s_{n-1} + s_{n-2}. \tag{1.1}$$

Формула говорит о том, что член последовательности s_n – это просто сумма двух предыдущих членов s_{n-1} и s_{n-2}. Сразу видно, что формула рекурсивна, так как определяемый ею объект s появляется в обеих частях уравнения. Таким образом, элементы последовательности явно определяются через самих себя. Кроме того, заметьте, что рекурсивная формула (1.1) описывает не конкретную последовательность, а целое семейство последовательностей, где каждый её член есть сумма двух предыдущих членов. Чтобы задать конкретную последовательность, мы должны предоставить дополнительную информацию. В данном случае достаточно задать любые два члена последовательности. Как правило, чтобы определить такую последовательность, достаточно задать два первых её члена. Например, если $s_1 = s_2 = 1$, то мы получим последовательность

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots,$$

которая является известной последовательностью чисел Фибоначчи.

Последовательности могут начинаться и с члена s_0.

Последовательность *s* можно считать функцией, которая в качестве аргумента получает положительное целое число *n*, а возвращает *n*-й член этой последовательности. В этом случае функция Фибоначчи, обозначенная просто как *F*, может быть определена как:

$$F(n) = \begin{cases} 1, & \text{если } n = 1, \\ 1, & \text{если } n = 2, \\ F(n-1) + F(n-2), & \text{если } n > 2. \end{cases} \quad (1.2)$$

Всюду в этой книге мы будем использовать для описания функции такие обозначения, когда определения включают два типа выражений или случаев. *Начальные условия* (base cases) соответствуют случаю, когда результат функции может быть получен тривиально без привлечения значений функции от дополнительных параметров. По определению, начальные условия для чисел Фибоначчи – это $F(1) = 1$ и $F(2) = 1$. *Рекурсивные условия* (recursive cases) представляют собой более сложные рекурсивные выражения, включающие обычно определённую функцию от предыдущих значений входных параметров. Функция Фибоначчи имеет одно рекурсивное условие: $F(n) = F(n - 1) + F(n - 2)$, если $n > 2$. Начальные условия необходимы для получения из рекурсивных условий конкретных значений членов последовательности. В заключение следует сказать, что рекурсивное определение может содержать несколько начальных и рекурсивных условий.

Ещё одна функция, которая может быть выражена рекурсивно, – это факториал некоторого неотрицательного целого числа *n*:

$$n! = 1 \times 2 \times \cdots \times (n - 1) \times n.$$

В этом случае рекурсивность функции не столь очевидна из-за явного отсутствия в правой части определения факториала. Но поскольку $(n - 1)! = 1 \times 2 \times \cdots \times (n - 1)$, можно переписать формулу рекурсивно: $n! = (n - 1)! \times n$. Наконец, по определению $0! = 1$, что следует из рекурсивной формулы для $n = 1$. Таким образом, функция факториала может быть определена рекурсивно как

$$n! = \begin{cases} 1, & \text{если } n = 0, \\ (n-1)! \times n, & \text{если } n > 0. \end{cases} \quad (1.3)$$

Рассмотрим также задачу вычисления суммы первых *n* положительных целых чисел. Соответствующая функция $S(n)$ может быть, очевидно, определена как:

$$S(n) = 1 + 2 + \cdots + (n-1) + n. \tag{1.4}$$

И снова мы пока не видим в правой части определения слагаемого, содержащего функцию S. Однако первые $n - 1$ членов можно сгруппировать так, чтобы получить $S(n - 1) = 1 + 2 + \ldots + (n - 1)$, что приводит к следующему рекурсивному определению:

$$S(n) = \begin{cases} 1, & \text{если } n = 1, \\ S(n-1) + n, & \text{если } n > 1. \end{cases} \tag{1.5}$$

Отметим, что $S(n - 1)$ – *подзадача*, подобная $S(n)$, но *более простая,* так как для получения её результата требуется меньшее количество операций. Таким образом, говорят, что подзадача имеет меньший *размер* (*размерность*). Кроме того, мы говорим, что выполнена *декомпозиция* (*разложение*) исходной задачи ($S(n)$) на *меньшие* для формирования рекурсивного определения. Таким образом, $S(n - 1)$ – меньший *экземпляр* исходной задачи.

Другой математический объект, рекурсивность которого не вполне очевидна, – неотрицательные целые числа. Эти числа можно разложить и определить рекурсивно через меньшие числа разными способами. Например, неотрицательное целое число n можно представить как предшествующее ему число плюс один:

$$n = \begin{cases} 0, & \text{если } n = 0, \\ predesessor(n) + 1, & \text{если } n > 0. \end{cases}$$

Заметим, что в рекурсивном условии n находится по обе стороны от знака равенства. Кроме того, если мы считаем, что функция *predecessor* обязательно возвращает неотрицательное целое число, то она неприменима к 0. Поэтому определение становится законченным только вместе с тривиальным начальным условием для $n = 0$.

Ещё один способ определения (неотрицательных) целых чисел – считать их упорядоченной последовательностью цифр. Например, число 5342 может быть конкатенацией следующих пар меньших чисел:

(5, 342), (53, 42), (534, 2).

На практике простейший способ декомпозиции целого числа – это представление его в виде соединения наименьшей его значащей цифры с остальной его частью. Поэтому целое число можно определить следующим образом: