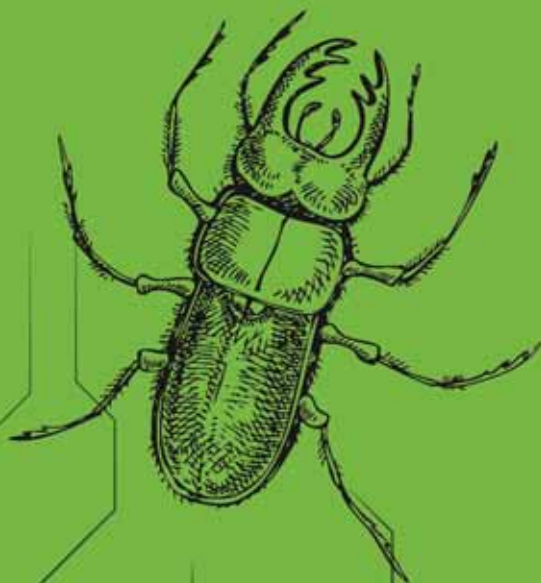


ВЛАДСТОН ФЕРРЕЙРА ФИЛО

ТЕОРЕТИЧЕСКИЙ МИНИМУМ ПО

COMPUTER SCIENCE



ВСЕ, ЧТО НУЖНО ПРОГРАММИСТУ
И РАЗРАБОТЧИКУ



ББК 32.973.23-018
УДК 004.3
Ф54

Феррейра Фило Владстон

Ф54 Теоретический минимум по Computer Science. Все, что нужно программисту и разработчику. — СПб.: Питер, 2018. — 224 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-0587-8

Хватит тратить время на скучные академические фолианты! Изучение Computer Science может быть веселым и увлекательным занятием.

Владстон Феррейра Фило знакомит нас с вычислительным мышлением, позволяющим решать любые сложные задачи. Научиться писать код просто — пара недель на курсах, и вы «программист», но чтобы стать профи, который будет востребован всегда и везде, нужны фундаментальные знания. Здесь вы найдете только самую важную информацию, которая необходима каждому разработчику и программисту каждый день.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.23-018
УДК 004.3

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-0997316001 англ.
ISBN 978-5-4461-0587-8

© Computer Science Distilled, Wladston Ferreira Filho, 2017
© Перевод на русский язык ООО Издательство «Питер», 2018
© Издание на русском языке, оформление ООО Издательство «Питер», 2018
© Серия «Библиотека программиста», 2018

Оглавление

Предисловие	11
Эта книга для меня?.....	12
Но разве computer science не только для ученых?	13
Глава 1. Основы	14
1.1. Идеи.....	15
Блок-схемы	15
Псевдокод	17
Математические модели.....	18
1.2. Логика.....	20
Операторы	21
Булева алгебра	23
Таблицы истинности.....	25
Логика в вычислениях.....	29
1.3. Комбинаторика.....	31
Правило умножения	31
Перестановки	32
Перестановки без повторений.....	34
Комбинации	35
Правило суммирования	36
1.4. Вероятность	38
Подсчет количества возможных вариантов	38
Независимые (совместные) события.....	39
Несовместные события.....	40
Взаимодополняющие события	40
«Заблуждение игрока»	41
Более сложные вероятности.....	42
Подведем итоги	42
Полезные материалы	43

Глава 2. Вычислительная сложность	44
Надейтесь на лучшее, но готовьтесь к худшему	45
2.1. Оценка затрат времени	47
Понимание роста затрат	48
2.2. Нотация «О большое»	50
2.3. Экспоненциальное время	52
2.4. Оценка затрат памяти	54
Подведем итоги	55
Полезные материалы	56
Глава 3. Стратегия	57
3.1. Итерация	58
Вложенные циклы и степенные множества	59
3.2. Рекурсия	62
Рекурсия против итераций	63
3.3. Полный перебор	64
3.4. Поиск (перебор) с возвратом	67
3.5. Эвристические алгоритмы	71
«Жадные» алгоритмы	71
Когда жадность побеждает силу	73
3.6. Разделяй и властвуй	75
Разделить и отсортировать	75
Разделить и заключить сделку	80
Разделить и упаковать	82
3.7. Динамическое программирование	84
Мемоизация Фибоначчи	84
Мемоизация предметов в рюкзаке	85
Лучшая сделка снизу вверх	86
3.8. Ветви и границы	88
Верхние и нижние границы	88
Ветви и границы в задаче о рюкзаке	89
Подведем итоги	92
Полезные материалы	93
Глава 4. Данные	94
Абстракции	95
Тип данных	96

4.1. Абстрактные типы данных	96
Преимущества использования АД	97
4.2. Общие абстракции	98
Примитивные типы данных	98
Стек	99
Очередь	100
Очередь с приоритетом	100
Список	101
Сортированный список	102
Множество	103
4.3. Структуры	104
Массив	104
Связный список	105
Двусвязный список	107
Массивы против связных списков	108
Дерево	109
Двоичное дерево поиска	112
Двоичная куча	115
Граф	117
Хеш-таблица	117
Подведем итоги	118
Полезные материалы	119
Глава 5. Алгоритмы	120
5.1. Сортировка	121
5.2. Поиск	124
5.3. Графы	125
Поиск в графах	126
Раскраска графов	129
Поиск путей в графе	130
PageRank	133
5.4. Исследование операций	133
Задачи линейной оптимизации	134
Задачи о максимальном потоке в Сети	137
Подведем итоги	138
Полезные материалы	139

Глава 6. Базы данных	140
6.1. Реляционная модель	142
Отношения	142
Миграция схемы	145
SQL	146
Индексация	148
Транзакции	151
6.2. Нереляционная модель	152
Документные хранилища	152
Хранилища «ключ — значение»	154
Графовые базы данных	155
Большие данные	156
SQL против NoSQL	157
6.3. Распределенная модель	158
Репликация с одним ведущим	159
Репликация с многочисленными ведущими	159
Фрагментирование	160
Непротиворечивость данных	162
6.4. Географическая модель	163
6.5. Форматы сериализации	165
Подведем итоги	166
Полезные материалы	166
Глава 7. Компьютеры	167
7.1. Архитектура	168
Память	168
Процессор	171
7.2. Компиляторы	177
Операционные системы	181
Оптимизация при компиляции	182
Языки сценариев	183
Дизассемблирование и обратный инженерный анализ	184
Программное обеспечение с открытым исходным кодом	185
7.3. Иерархия памяти	186
Разрыв между памятью и процессором	187
Временная и пространственная локальность	188

Кэш L1.....	189
Кэш L2.....	189
Первичная память против вторичной	191
Внешняя и третичная память	193
Тенденции в технологии памяти.....	194
Подведем итоги	195
Полезные материалы	196
Глава 8. Программирование	197
8.1. Лингвистика	198
Значения.....	198
Выражения.....	198
Инструкции	200
8.2. Переменные	201
Типизация переменных	202
Область видимости переменных.....	202
8.3. Парадигмы	204
Императивное программирование	204
Декларативное программирование.....	207
Логическое программирование.....	213
Подведем итоги	214
Полезные материалы	214
Заключение.....	215
Приложения	217
I. Системы счисления.....	217
II. Метод Гаусса	219
III. Множества	220
IV. Алгоритм Кэдейна	222

Предисловие

Каждый в нашей стране должен научиться программировать, потому что это учит думать.

Стив Джобс

Когда компьютеры начали менять мир, открывая перед людьми беспрецедентные возможности, расцвела новая наука — *computer science*. Она показала, как использовать компьютеры для решения задач. Это позволило нам использовать весь потенциал вычислительных машин. И мы достигли удивительных, просто сумасшедших результатов.

Computer science повсюду, но эта наука по-прежнему преподается как скучная теория. Многие программисты даже не изучали ее! Однако она крайне важна для эффективного программирования. Некоторые мои друзья не могут найти хорошего программиста, чтобы взять его на работу. Вычислительные мощности сегодня в изобилии, а вот людей, способных ими пользоваться, не хватает.

Рис. 1. Компьютерные задачи¹

Эта книга — моя скромная попытка помочь миру, а также подтолкнуть *вас* к эффективному использованию компьютеров. В ней понятия computer science представлены в простой форме. Я свел научные подробности к минимуму. Хочется надеяться, что computer science произведет на вас впечатление, и ваш программный код станет лучше.

Эта книга для меня?

Если вы хотите щелкать задачи как орешки, находя эффективные решения, то эта книга для вас. От вас потребуются только чуть-чуть опыта в написании программного кода. Если вам приходилось этим заниматься и вы различаете элементарные операторы вроде `for` и `while`, то все в порядке. В противном случае вы найдете все необходимое (и даже больше) на каких-нибудь онлайн-курсах программирования². Вы можете пройти такой курс всего за неделю, и притом бесплатно. Для тех же, кто уже знаком с информатикой, эта книга станет превосходным повторением пройденного и поможет укрепить знания.

¹ Рисунок использован с согласия <http://xkcd.com>.

² См., например, <http://code.energy/coding-courses>.

Но разве computer science не только для ученых?

Эта книга — для всех. Она о вычислительном мышлении. Вы научитесь превращать задачи в вычислимые системы. Вы также будете использовать вычислительное мышление в повседневных задачах. Упреждающая выборка и кэширование помогут вам упростить процесс упаковки вещей. Освоив параллелизм, вы станете эффективнее управляться на кухне. Ну и, разумеется, ваш программный код будет просто потрясающим.

Да пребудет с вами сила! 😊

Влад

Глава 1

ОСНОВЫ

Информатика не более наука о компьютерах, чем астрономия — наука о телескопах. Информатика неразрывно связана с математикой.

Эдсгер Дейкстра

Компьютерам нужно, чтобы мы разбивали задачи на посильные для них части. Тут нам понадобится немного математики. Не паникуйте, это не высшая математика — написание хорошего программного кода редко требует знания сложных уравнений. В главе 1 вы найдете набор инструментов для решения разных задач. Вы научитесь:



моделировать *идеи* в блок-схемах и псевдокоде;



отличать правильное от неправильного при помощи *логики*;



выполнять *расчеты*;



уверенно вычислять *вероятности*.

Этого достаточно, чтобы переводить мысли в вычислимые решения.

1.1. Идеи

Оказавшись перед сложной задачей, поднимитесь над ее хитросплетениями и изложите все самое важное на бумаге. Оперативная память человеческого мозга легко переполняется фактами и идеями. Многие подходы к организации работы предполагают изложение мыслей в письменной форме. Есть несколько способов это сделать. Сначала мы посмотрим, как пользоваться блок-схемами для представления процессов. Затем узнаем, как конструировать программируемые процессы на псевдокоде. Мы также попробуем смоделировать простую задачу при помощи математических формул.

Блок-схемы

Когда разработчики «Википедии» обсуждали организацию коллективной работы, они создали блок-схему дискуссии. Договариваться проще, если все инициативы перед глазами и объединены в общую картину (рис. 1.1).

Компьютерный код, как и изображенный выше процесс редактирования вики-страницы, по существу является процессом. Программисты часто пользуются блок-схемами для изображения вычислительных процессов на бумаге. Чтобы другие могли понимать ваши блок-схемы, вы должны соблюдать следующие рекомендации¹:

- записывайте состояния и инструкции внутри прямоугольников;
- записывайте принятие решений, когда процесс может пойти различными путями, внутри ромбов;
- никогда не объединяйте инструкции с принятием решений;

¹ Адаптация схемы с сайта <http://wikipedia.org>.

- соединяйте стрелкой каждый последующий шаг с предыдущим;
- отмечайте начало и конец процесса.

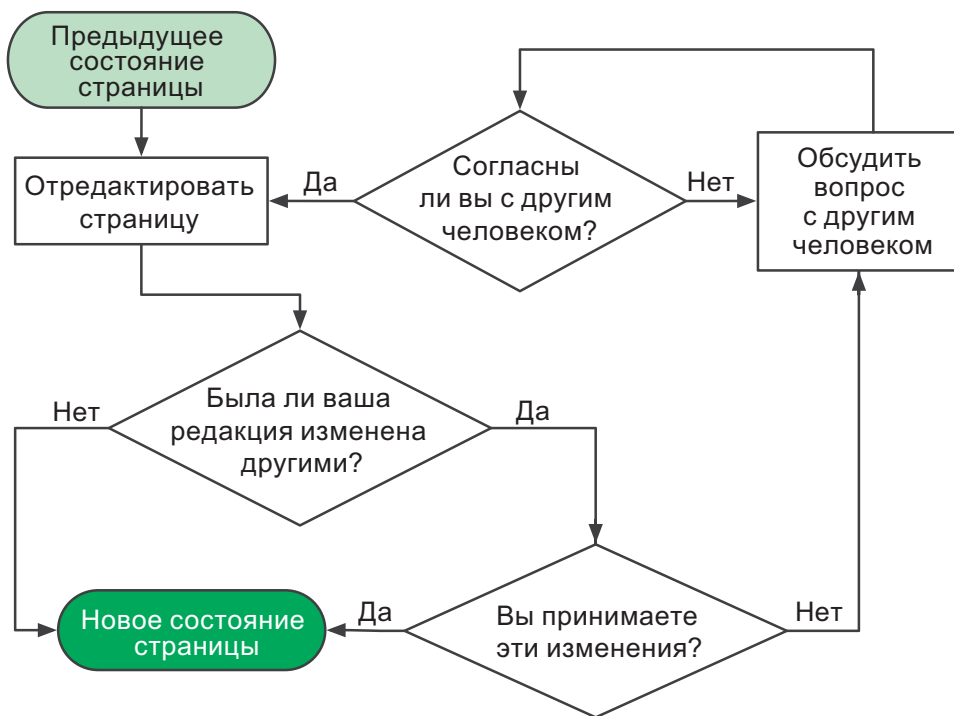


Рис. 1.1. Редакционный процесс в «Википедии»¹

¹ См., например, <http://code.energy/coding-courses>.

Рассмотрим составление блок-схемы на примере задачи поиска наибольшего из трех чисел (рис. 1.2).

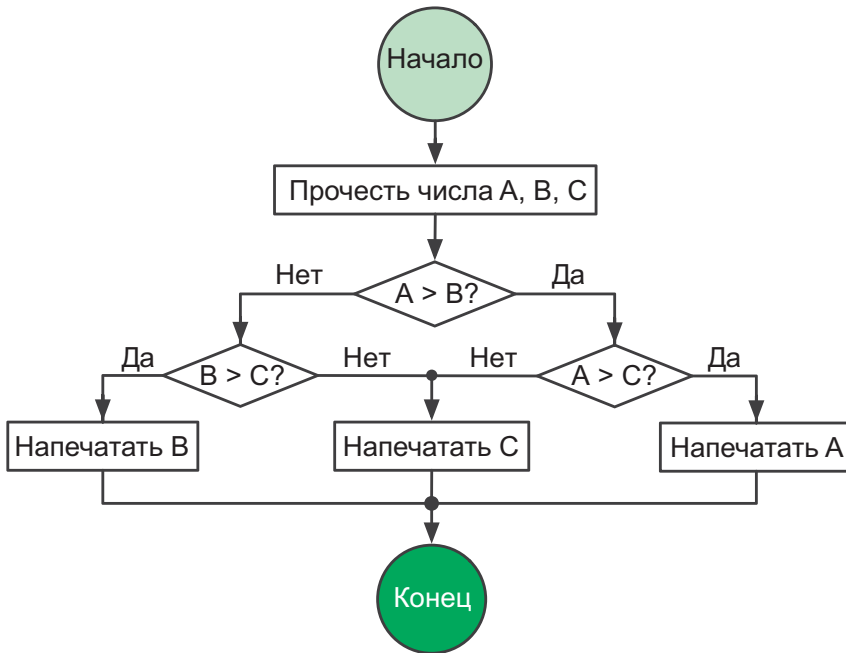


Рис. 1.2. Поиск наибольшего из трех чисел

Псевдокод

Так же, как блок-схемы, *псевдокод* выражает вычислительные процессы. Псевдокод — это код, удобный для нашего восприятия, но непонятный для машины. Следующий пример передает тот же процесс, что был изображен на рис. 1.2. Задержитесь на минуту и проверьте, как он работает с разными значениями A, B и C¹.

¹ Здесь \leftarrow означает оператор присваивания: $x \leftarrow 1$ следует читать как «Присвоить x значение 1».

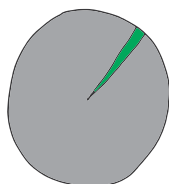
```

function maximum(A, B, C)
  if A > B
    if A > C
      max ← A
    else
      max ← C
  else
    if B > C
      max ← B
    else
      max ← C
  print max

```

Заметили, что этот пример полностью игнорирует синтаксические правила языков программирования? В псевдокод можно вставлять даже разговорные фразы! Когда вы пишете псевдокод, дайте своей творческой мысли течь свободно — как при составлении блок-схем (рис. 1.3 😊).

Применение псевдокода в реальной жизни



- Средство для описания алгоритмов
- Инструмент, которым пользуются программисты-первокурсники для выражения своих глупых мыслей

ctp200.com



Рис. 1.3. Псевдокод в реальной жизни¹

Математические модели

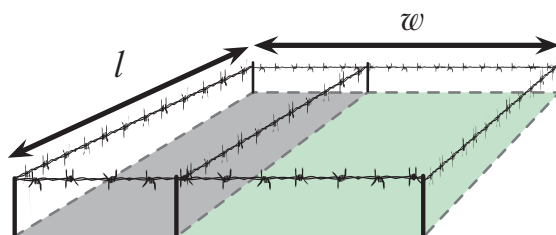
Модель — это набор идей, которые описывают задачу и ее свойства. Модель помогает рассуждать и принимать решения относительно задачи. Создание моделей настолько важно, что их преподают в школе — ведь в математике нужно иметь представление, как последовательно решать уравнения и совершать другие операции с числами и переменными.

¹ Любезно предоставлено <http://ctp200.com>.

Математические модели имеют большое преимущество: их можно приспособить для компьютеров при помощи четко сформулированных математических методов. Если ваша модель основана на графах, используйте теорию графов. Если она задействует уравнения, используйте алгебру. *Встаньте на плечи гигантов*, которые создали эти инструменты, и вы достигнете цели. Давайте посмотрим, как они работают, на примере типичной задачи из средней школы.

Загон для скота 🐄 На ферме содержат два вида домашних животных. У вас есть 100 мотков проволоки для сооружения прямоугольного загона и перегородки внутри него, отделяющей одних животных от других. Как поставить забор, чтобы площадь пастбища была максимальной?

Начнем с того, что именно требуется определить; w и l — это размеры пастбища; $w \times l$ — его площадь. Сделать площадь максимальной означает использовать всю проволоку, потому мы устанавливаем связь между w и l , с одной стороны, и 100 мотками, с другой:



$$A = w \times l$$

$$100 = 2w + 3l$$

Подберем w и l , при которых площадь A будет максимальной.

Подставив l из второго уравнения $\left(l = \frac{100 - 2w}{3}\right)$ в первое, получаем:

$$A = \frac{100}{3}w - \frac{2}{3}w^2.$$

Да это же квадратное уравнение! Его максимум легко найти при помощи *формулы корней квадратного уравнения*, которую проходят в средней школе. Квадратные уравнения так же важны для программиста, как мультиварка — для повара. Они экономят время. Квадратные уравнения помогают быстрее решать множество задач, а это для вас самое главное. Повар знает свои инструменты, вы должны знать свои. Математическое моделирование вам просто необходимо. А еще вам потребуется логика.

1.2. Логика

Программистам приходится иметь дело с логическими задачами так часто, что у них от этого ум за разум заходит. Однако на самом деле многие из них логику не изучали и пользуются ею бессознательно. Освоив формальную логику, мы сможем осознанно использовать ее для решения задач.



Рис. 1.4. Логика программиста¹

¹ Любезно предоставлено <http://programmers.life>.

Для начала мы поэкспериментируем с логическими высказываниями и операторами. Затем научимся решать задачи с таблицами истинности и увидим, как компьютеры опираются на логику.

Операторы

В математике переменные и операторы (+, ×, −, ...) используются для моделирования числовых задач. В математической логике переменные и операторы указывают на достоверность. Они выражают не числа, а истинность (true) или ложность (false). Например, достоверность выражения «Если вода в бассейне теплая, то я буду плавать» основывается на достоверности двух вещей, которые можно преобразовать в логические переменные A и B :

A : Вода в бассейне теплая.

B : Я плаваю.

Они либо истинны (true), либо ложны (false)¹. $A = \text{True}$ обозначает теплую воду в бассейне, $B = \text{False}$ обозначает «Я не плаваю». Переменная B не может быть *наполовину истинной*, потому что я не способен плавать лишь отчасти. Зависимость между переменными обозначается символом \rightarrow , *условным оператором*. $A \rightarrow B$ выражает идею, что $A = \text{True}$ влечет за собой $B = \text{True}$:

$A \rightarrow B$: если вода в бассейне теплая, то я буду плавать.

При помощи других операторов можно выражать другие идеи. Для отрицания идеи используется знак $!$, *оператор отрицания*. $!A$ противоположно A :

$!A$: Вода в бассейне холодная.

$!B$: Я не плаваю.

¹ В нечеткой логике значения могут быть промежуточными, но в этой книге она рассматриваться не будет.

Противопоставление. Если дано $A \rightarrow B$, и я при этом не плаваю, что можно сказать о воде в бассейне? Теплая вода *влечет за собой* плавание, потому, если его нет, вода в бассейне не может быть теплой. Каждое условное выражение имеет *противопоставленный* ему эквивалент:

Для любых двух переменных A и B

$$A \rightarrow B \text{ тождественно } !B \rightarrow !A.$$

Еще пример: если вы не умеете писать хороший код, значит, вы не прочли эту книгу. *Противопоставлением данному суждению является такое:* если вы прочли эту книгу, значит, вы умеете писать хороший код. *Оба предложения сообщают одно и то же, но по-разному*¹.

Двусторонняя условная зависимость. Обратите внимание, что высказывание «Если вода в бассейне теплая, то я буду плавать» не означает: «Я буду плавать только в теплой воде». Данное высказывание ничего не говорит насчет холодных бассейнов. Другими словами, $A \rightarrow B$ не означает $B \rightarrow A$. Чтобы выразить оба условных суждения, используйте *двустороннюю условную зависимость*:

$A \leftrightarrow B$: Я буду плавать, если и только если вода в бассейне теплая.

Здесь теплая вода в бассейне равнозначна тому, что я буду плавать: знание о воде в бассейне означает знание о том, что я буду плавать, *и наоборот*. Опять же, остерегайтесь *обратной ошибки*: никогда не предполагайте, что $B \rightarrow A$ следует из $A \rightarrow B$.

AND, OR и XOR. Эти логические операторы — самые известные, поскольку они часто записываются в исходном коде в явном виде — AND (И), OR (ИЛИ) и XOR (исключающее ИЛИ). AND возвращает True, если все идеи истинны; OR возвращает True, если любая идея истинна; XOR возвращает True, если идеи взаимоисключающие. Представим вечеринку, где подают водку и вино:

¹ И, между прочим 🤪, они оба *фактически* истинные.

A : Вы пили вино. 🍷

B : Вы пили водку. 🍸

$A \text{ OR } B$: Вы пили. 🍹

$A \text{ AND } B$: Вы пили и то и другое. 😞

$A \text{ XOR } B$: Вы пили, не смешивая. 😊

Проверьте, правильно ли вы понимаете, как работают эти операторы. В табл. 1.1 перечислены все возможные комбинации двух переменных. Обратите внимание, что $A \rightarrow B$ тождественно $\neg A \text{ OR } B$, а $A \text{ XOR } B$ тождественно $\neg(A \leftrightarrow B)$.

A	B	$\neg A$	$A \rightarrow B$	$A \leftrightarrow B$	$A \text{ AND } B$	$A \text{ OR } B$	$A \text{ XOR } B$
✓	✓	✗	✓	✓	✓	✓	✗
✓	✗	✗	✗	✗	✗	✓	✓
✗	✓	✓	✓	✗	✗	✓	✓
✗	✗	✓	✓	✓	✗	✗	✗

Таблица 1.1. Логические операции для четырех возможных комбинаций A и B

Булева алгебра

*Булева алгебра*¹ позволяет упрощать логические выражения точно так же, как элементарная алгебра упрощает числовые.

Ассоциативность. Для последовательностей, состоящих только из операций AND или OR, круглые скобки не имеют значения. Так же, как последовательности только из операций сложения или умножения в элементарной алгебре, эти операции могут вычисляться в любом порядке.

¹ Названа так в честь Джорджа Буля (1815–1864). Его публикации положили начало математической логике.

$$A \text{ AND } (B \text{ AND } C) = (A \text{ AND } B) \text{ AND } C;$$

$$A \text{ OR } (B \text{ OR } C) = (A \text{ OR } B) \text{ OR } C.$$

Дистрибутивность. В элементарной алгебре мы раскрываем скобки: $a \times (b + c) = (a \times b) + (a \times c)$. Точно так же и в логике выполнение операции AND после OR эквивалентно выполнению операции OR над результатами операций AND и наоборот:

$$A \text{ AND } (B \text{ OR } C) = (A \text{ AND } B) \text{ OR } (A \text{ AND } C);$$

$$A \text{ OR } (B \text{ AND } C) = (A \text{ OR } B) \text{ AND } (A \text{ OR } C).$$

Правило де Моргана¹. Одновременно лета и зимы не бывает, поэтому у нас *либо не лето, либо не зима*. С другой стороны, оба выражения «не лето» и «не зима» истинны, *если (и только)* у нас не тот случай, когда *либо лето, либо зима*. Согласно этой логике, выполнение операций AND может быть сведено к операциям OR и наоборот:

$$\neg(A \text{ AND } B) = \neg A \text{ OR } \neg B;$$

$$\neg A \text{ AND } \neg B = \neg(A \text{ OR } B).$$

Эти правила позволяют преобразовывать логические модели, раскрывать их свойства и упрощать выражения. Давайте решим задачу.

Перегрев сервера ★ Сервер выходит из строя из-за перегрева, когда кондиционирование воздуха выключено. Он также выходит из строя из-за перегрева, если барахлит кулер. При каких условиях сервер работает?

Моделируя эту задачу в логических переменных, можно в одном выражении сформулировать условия, когда сервер выходит из строя:

¹ Огастес де Морган дружил с Джорджем Булем. Кроме того, он обучал молодую Аду Лавлейс, ставшую первым программистом за век до того, как был создан первый компьютер.

A: Сервер перегревается.

B: Кондиционирование отключено.

C: Не работает кулер.

D: Сервер вышел из строя.

$(A \text{ AND } B) \text{ OR } (A \text{ AND } C) \rightarrow D$.

Используя правило дистрибутивности, выведем за скобки *A*:

$$A \text{ AND } (B \text{ OR } C) \rightarrow D.$$

Сервер работает, когда $\neg D$. Противопоставление записывается так:

$$\neg D \rightarrow \neg(A \text{ AND } (B \text{ OR } C)).$$

Применим правило де Моргана и раскроем скобки:

$$\neg D \rightarrow \neg A \text{ OR } \neg(B \text{ OR } C).$$

Воспользуемся правилом де Моргана еще раз:

$$\neg D \rightarrow \neg A \text{ OR } (\neg B \text{ AND } \neg C).$$

Данное выражение нам говорит, что когда сервер работает, мы имеем либо $\neg A$ (он не перегревается), либо $\neg B \text{ AND } \neg C$ (все в порядке и с кондиционированием воздуха, и с кулером).

Таблицы истинности

Еще один способ анализа логических моделей состоит в сверке данных со всевозможными сочетаниями ее переменных. Каждой переменной в *таблице истинности* соответствует свой столбец. Строки представляют комбинации состояний переменных.

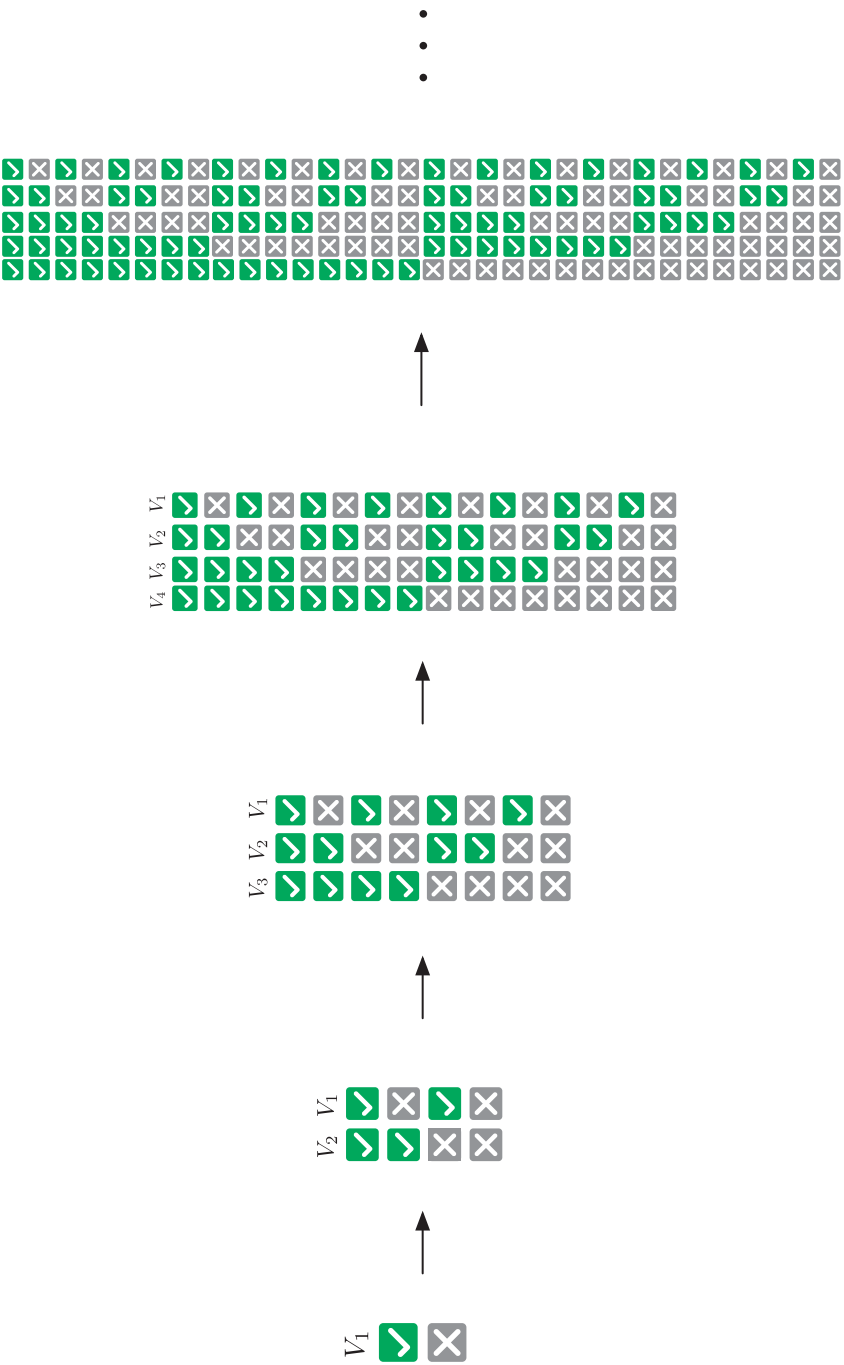



Рис. 1.5. Таблицы со всеми возможными сочетаниями от одной до пяти логических переменных

Одна переменная требует двух строк: в одной она имеет значение `True`, в другой — `False`. Чтобы добавить переменную, нужно удвоить число строк. Новой переменной задается `True` в исходных строках и `False` — в добавленных (рис. 1.5). Размер таблицы истинности увеличивается вдвое с каждым добавлением переменной, поэтому такую таблицу оправданно использовать лишь в случаях, когда переменных немного¹.

Давайте посмотрим, как можно использовать таблицу истинности для анализа задачи.


Хрупкая система  Предположим, что мы должны создать систему управления базами данных с соблюдением следующих технических требований:

- 1) если база данных заблокирована, то мы можем сохранить данные;
- 2) база данных не должна блокироваться при заполненной очереди запросов на запись;
- 3) либо очередь запросов на запись полна, либо полон кэш;
- 4) если кэш полон, то база данных не может быть заблокирована.

Возможно ли это? При каких условиях станет работать такая система?

Сначала преобразуем каждое техническое требование в логическое выражение. Такую систему управления базами данных можно смоделировать при помощи четырех переменных.

<i>A</i> : База данных заблокирована	1: $A \rightarrow B$
<i>B</i> : Есть возможность сохранить данные	2: $\neg(A \text{ AND } C)$.
<i>C</i> : Очередь запросов на запись полна	3: $C \text{ OR } D$.
<i>D</i> : Кэш полон	4: $D \rightarrow \neg A$.

¹ Например, таблица истинности для 30 переменных будет иметь более миллиарда строк .

Далее создадим таблицу истинности со всеми возможными сочетаниями переменных (табл. 1.2). Дополнительные столбцы добавлены для проверки соблюдения технических требований.

Таблица 1.2. Таблица истинности для проверки четырех выражений

Состояние #	A	B	C	D	1	2	3	4	Все четыре
1	✓	✓	✓	✓	✓	✗	✓	✗	✗
2	✓	✓	✓	✗	✓	✗	✓	✓	✗
3	✓	✓	✗	✓	✓	✓	✓	✗	✗
4	✓	✓	✗	✗	✓	✓	✗	✓	✗
5	✓	✗	✓	✓	✗	✗	✓	✗	✗
6	✓	✗	✓	✗	✗	✗	✓	✓	✗
7	✓	✗	✗	✓	✗	✓	✓	✗	✗
8	✓	✗	✗	✗	✗	✓	✗	✓	✗
9	✗	✓	✓	✓	✓	✓	✓	✓	✓
10	✗	✓	✓	✗	✓	✓	✓	✓	✓
11	✗	✓	✗	✓	✓	✓	✓	✓	✓
12	✗	✓	✗	✗	✓	✓	✗	✓	✗
13	✗	✗	✓	✓	✓	✓	✓	✓	✓
14	✗	✗	✓	✗	✓	✓	✓	✓	✓
15	✗	✗	✗	✓	✓	✓	✓	✓	✓
16	✗	✗	✗	✗	✓	✓	✗	✓	✗

Все технические требования удовлетворяются в состояниях с 9-го по 11-е и с 13-го по 15-е. В этих состояниях $A = \text{False}$, а значит, база данных не может быть заблокирована никогда. Обратите внимание, что кэш не заполнен лишь в состояниях 10 и 14.

Чтобы проверить, чему вы научились, попробуйте разгадать загадку «Кто держит зебру?»¹. Это известная логическая задача, ошибочно

¹ См. <http://code.energy/zebra-puzzle>.

приписываемая Альберту Эйнштейну. Говорят, что только 2 % людей могут ее решить, но я сильно сомневаюсь. Используя большую таблицу истинности и правильно упрощая и объединяя логические высказывания, вы ее разгадаете, я уверен в этом.

Всегда, имея дело с ситуациями, допускающими один из двух вариантов, помните: их можно смоделировать с помощью логических переменных. Благодаря этому очень легко получать выражения, упрощать их и делать выводы.

А теперь давайте взглянем на самое впечатляющее применение логики: проектирование электронно-вычислительных машин.

Логика в вычислениях

Группы логических переменных могут представлять числа в двоичной форме¹. Логические операции в случае с двоичными числами могут объединяться для расчетов. *Логические вентили* выполняют логические операции с электрическим током. Они используются в электрических схемах, выполняющих вычисления на сверхвысоких скоростях.

Логический вентиль получает значения через входные контакты, выполняет работу и передает результат через выходной контакт. Существуют логические вентили AND, OR, XOR и т. д. Значения True и False представлены электрическими сигналами с высоким и низким напряжением соответственно. Сложные логические выражения можно вычислять таким образом практически мгновенно. Например, электрическая схема на рис. 1.6 суммирует два числа.

Давайте посмотрим, как работает эта схема. Не поленитесь, проследите за ходом выполнения операций, чтобы понять, как устроена магия (рис. 1.7).

¹ True = 1, False = 0. Если вы не знаете, почему 101 — это 5 в двоичной системе счисления, загляните в приложение I.