МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М.В.ЛОМОНОСОВА

Отчет по мероприятию

Организация учебных выездных лекций и семинаров для школьников и абитуриентов в образовательных учреждениях общего среднего образования города Москвы

НОМ 2

Научно-популярные материалы для школьников о современной математике

Москва 2011

Математика внутри компьютера

Для того, чтобы вывести из ничтожества все,
достаточно единицы.
Готфрид Вильгельм Лейбниц

Компьютеры помогают решать математические задачи- это общеизвестно. Многие думают, что математика без компьютеров практически уже не обходится, а некоторые даже полагают, что со временем компьютеры вообще заменят математиков (и не только их), когда появится искусственный разум. На самом деле в этих мнениях значение компьютеров для математики сильно преувеличено (хотя, возможно, есть в них и рациональное зерно). На самом деле многие математики используют компьютер только как печатную машинку (для того, чтобы писать тексты с формулами), изредка пользуются готовыми программами (Mathematica, Maple, Mathcad, Mathlab и др.) для вспомогательных вычислений, и редко сами пишут программы для каких-то своих целей (если их не устраивают доступные программы, написанные другими).

Но мы не будем далее обсуждать затронуты темы, а рассмотрим совсем другую - как применяется математика в самом компьютере, т.е. не в написании программ для него, а непосредственно внутри него, а также при проектировании и производстве компьютеров (и другой электронной цифровой техники). С этой темой также связаны различные заблуждения. Часто полагают, что указанные вопросы относятся скорее к технике, или на крайний случай к физике (и немного химии), но не к математике. Мы постараемся на простых примерах показать, что внутри компьютеров (и вообще в любой цифровой технике) живет и работает очень интересная математика, которая активно развивается вместе с развитием электронной техники

Алгебра логики и компьютерная арифметика

Увы! То, что некогда возвышалось
как монумент монотеизму,
очутилось в чреве компьютера
Т.Данциг

В последнее время почти вся техника, связанная с передачей и обработкой информации, стала цифровой. Цифровыми стали аудио и видеомагнитофоны, превратившись в DVD-плейеры и айподы, телевизоры, фотоаппараты, а все виды электронно-вычислительной техники и мобильные телефоны были цифровыми изначально. Это означает, что информация, циркулирующая в этих устройствах, представляется (кодируется) в цифровом виде, т.е. как правило, в виде строк (последовательностей), состоящих из нулей и единиц. Этим строкам можно сопоставить по некоторым правилам целые числа, для чего обычно используется двоичная позиционная система их записи.

Таким образом, с определенной точки зрения, все цифровые устройства генерируют потоки целых чисел, по некоторым правилам их преобразуют, обрабатывают, кодируют и декодируют и т.д. и т.п. и передают другим цифровым устройствам.

Существенную роль в этом играют алгоритмические процедуры, выполняющие арифметические и логические операции с различными типами числовых данных. Проектированием подобных алгоритмов и устройств, их реализующих, занимается компьютерная арифметика. Ее математической основой является теория сложности функций алгебры логики.

В большей своей части компьютерная арифметика является двоичной арифметикой. Этому есть две причины. Во-первых, алгоритмы арифметических операций двоичной арифметики (арифметики, использующей двоичную позиционную систему) очень просты и являются в определенном смысле простейшими среди подобных алгоритмов для всех позиционных числовых систем. Во-вторых, дискретные электронные схемы, как самые современные, так использованные много лет назад, имеют в определенном смысле двоичную природу и легко описываются на языке алгебры логики. Алгебра логики применяется как для моделирования функционирования этих схем, так и для их проектирования (синтеза).

Булевы функции и логические функциональные элементы

Джон Буль установил истинную связь
алгебры и логики.
Август де Морган.

Современные электронные схемы, состоят из элементов, называемых также ячейками, которые имеют микроскопические размеры и представляют из себя особые участки кремниевого кристалла. Физика и химия протекающих в них процессов весьма сложна, и не будет здесь обсуждаться. Также сложна и технология их производства. Но логика работы этих элементов довольно проста, каждый из них реализует определенную логическую операцию.

Математической моделью простейших (однотактных) электронных схем являются логические функциональные элементы.

Функциональный элемент - это абстрактное устройство с одним или несколькими входами, на которые подается информация, и одним или несколькими выходами, информация с которых подается на входы других элементов или на выход всего большого устройства, содержащего этот элемент. Упомянутая информация содержится в состояниях входов или выходов элемента. Этих состояний у каждого полюса элемента (это совокупное название его входов и выходов) два, и они обозначаются нулем или единицей. Если состояния входов элемента определены (т.е. известны сопоставленные им значения 0 или 1), то значения на выходах элемента тоже однозначно определены, т.е. являются функциями значений входов.

Эти функции имеют аргументы, которые могут быть равными только 0 или 1, и сами эти функции принимают только значения 0 или 1. Такие функции называются функциями алгебры логики, или булевыми функциями. Последнее название дано в честь одного из первопроходцев математической логики, английского математика Джорджа Буля[1].

Говорят, что элемент реализует (на своих выходах) упомянутые булевы функции, однозначно определенные самим этим элементом.

Простейшим примером логического функционального элемента является элемент с одним входом и одним выходом, значение которого всегда противоположно значению входа. Это значит, что он реализует булеву функцию f(x), определяемую равенствами f(0)=1, f(1)=0. Такую функцию называют логическим отрицанием, обычно обозначают символом ¬x или чертой над символом переменной, а реализующий ее элемент называют иногда инвертором.

Примерами элементов с двумя входами и одним выходом являются элементы, называемые иногда конъюнктором и дизъюнктором. Первый из них реализует функцию, называемую в алгебре логики конъюнкцией. Она принимает единичное значение только тогда, когда оба ее аргумента равны единице (в остальных случаях она равна нулю). По существу она совпадает с обычной операцией умножения целых чисел, если ее применять только к числам, равным нулю или единице. Поэтому ее иногда называют логическим умножением и обозначают точкой, которую часто в записи формул опускают. Однако во многих случаях полезно подчеркнуть логическую природу этой функции и ее тесную связь с логической связкой, также называемой конъюнкцией, и тогда для ее обозначения используют символ &, который пишут между символами ее аргументов, например как x&y, а не &(x,y), как следовало бы делать, придерживаясь функционального способа записи. В исчислении высказываний (разделе

1 Джордж Буль(1815 - 1864) – профессор университета в Корке (Ирландия). Отец шести дочерей, одна из которых стала математиком, одна ---первой женщиной-профессором химии в Англии, а одна ---писательницей, известной под именем Этель Лилиан Войнич, автором романа <<Овод>>.

математической логики) символ конъюнкции, поставленный между двумя высказываниями, интерпретируется как союз <<и>>, поэтому в англоязычных странах инженеры обозначают конъюнктор символом AND.

Другим примером элемента с двумя входами является дизъюнктор, реализующий булеву функцию, называемую дизъюнкцией. Эта функция является двойственной к конъюнкции, и определяется двойственным образом, т.е. подобно конъюнкции, но с заменой 0 на 1 и наоборот. Более явно определение дизъюнкции можно дать следующим образом: дизъюнкция равна нулю тогда и только тогда, когда оба ее аргумента нули. Для обозначение дизъюнкции используется символ $\vee$ (ее иногда называют логическим сложением, но нужно помнить, что в отличие от обычного сложения $1 \vee 1$ равно 1, а не 2), а в англоязычных странах дизъюнктор обозначают символом OR, так как функция дизъюнкция связана с логической связкой, также называемой дизъюнкцией, которая интерпретируется как союз <<или>>. В русском, как и в английском, союз <<или>> обычно понимается в разделительном смысле, который отличен, от того, как понимается этот союз в логике.

Двоичная система и логические операции

Девушкам надо лишь уметь отличать
единицу от нуля […]
Нил Стивенсон <<Система мира>>, 2011

Произвольное натуральное число можно единственным способом представить в виде суммы степеней двойки, например 23 = 16+4+2+1. Обозначая входящие в эту сумму степени единицами, а не входящие в ее степени нулями, можно кратко обозначить эту сумму булевым набором (в другой терминологии - вектором) $(10111)_2$. Индекс 2 напоминает о том, что число записано в двоичной системе. Единица, стоящая в младшем (самом левом) разряде, означает слагаемое 1, единица во втором слева разряде означает слагаемое 2, единица в третьем разряде означает 4, а нуль в четвертом разряде означает отсутствие слагаемого 8, единица в четвертом (старшем) разряде означает присутствие слагаемого 16 (в большинстве случаев разумно рассматривать только такие записи чисел в двоичной системе, в которых в старшем разряде стоит единица).

Главное достоинство двоичной системы - исключительная простота алгоритмов арифметических операций в ней. Таблица умножения в ней совсем не требует запоминания: любое число, умноженное на нуль дает нуль, а умноженное на единицу равно самому себе. Правило деления сводится к двум равенствам 0/1 = 0, 1/1 =1, благодаря чему деление столбиком в двоичной системе делается проще, чем в десятичной, и по существу сводится к многократному вычитанию. Таблица сложения в двоичной системе чуть сложнее таблицы умножения (в отличие от десятичной системы), так как $1+1 = (10)_2$ и возникает перенос в следующий разряд.

В общем виде операцию сложения однобитовых чисел можно записать в следующем виде $x+y = 2w+v$, где w,v--- биты результата. Внимательно посмотрев на таблицу сложения, можно заметить, что бит переноса w --- это просто произведение xy, потому что он равен единице лишь когда x и y равны единице. Ясно также, что бит v равен x+y, за исключением случая x=y=1, когда он равен не 2, а 0. Операцию, с помощью которой по битам x,y вычисляют бит v, называют по-разному. Мы будем использовать для нее название сложение по модулю 2 и символ $\oplus$. Таким образом, сложение битов выполняется фактически не одной, а двумя операциями.

Логические тождества

Указанные операции связаны между собой множеством тождеств, из которых мы приведем здесь только следующие:

$$x \vee y = \neg(\neg x \neg y),\ xy = \neg(\neg x \vee \neg y),\ x \oplus y = x\neg y \vee y\neg x,\ x \vee y = x \oplus y \oplus xy,$$
$$(x \vee y)z = xz \vee yz,\ (x \oplus y)z = xy \oplus xz,\ xy \vee z = (x \vee z)(y \vee z),\ \neg x = x \oplus 1$$

Задача. Проверьте эти тождества.

Иногда удобнее записывать отрицание длинной формулы в виде черты сверху, например вместо $\neg(xyz)$ писать $\overline{xyz}$. Можно выразить указанные операции, называемые логическими, через обычные арифметические операции. Конъюнкция, например, просто совпадает с обычным умножением, поэтому ее при записи часто обозначают точкой, или вообще пропускают. Справедливы также равенства

$$x \oplus y = x+y-2xy,\ x \vee y = x+y-xy.$$

Можно выразить конъюнкцию и дизъюнкцию также через операции взятия максимума или минимума:

$x \vee y = \max(x,y) = 1-\min(1-x,1-y)$, $x\, y = \min(x,y) = 1-\max(1-x,1-y)$.

Логические операции и исчисление высказываний

Почему введенные выше операции называют логическими? Потому, что если сопоставить каждому высказыванию A его <<истиностное>> значение t(A)=0, если высказывание A ложно, и положить t(A)=1, если высказывание A истинно, то истинностное значение составного высказывания A и B можно вычислить через истинностные значения высказываний A,B по формуле t(A и B) = t(A) & t(B) и, аналогично, t(A или B) = t(A)∨t(B).

В последней формуле мы предполагали, что высказывание A или B будет истинным и тогда, когда оба высказывания A и B истинны. Иногда союз или понимают в несколько другом, разделительном, смысле: составное высказывание A или B считается истинным только в случае, если ровно одно из высказываний A,B истинно, но не оба сразу. Но логическая связка, соответствующая разделительной дизъюнкции, в логике тоже есть, и соответствующая ей булева функция принимает единичное значение тогда и только тогда, когда один из ее аргументов единица, а второй нуль. В алгебре логики ее часто обозначают символом ⊕ и называют суммой по модулю два, так как она действительно совпадает с этой алгебраической операцией. В англоязычных странах реализующий ее элемент обозначают символом XOR (eXclusive OR).

Эту функцию можно выразить через остальные три функции, например следующим образом: $x \oplus y = (x \& \neg y) \vee (\neg x \& y)$.

В алгебре логики часто приходится выражать одни булевы функции через другие, причем во многих случаях это можно сделать множеством способов, которые порождают множество логических тождеств. Например, справедливо тождество

$$x \oplus y = (x \vee y)\ \neg(x \& y).$$

Краткая история алгебры логики и вычислительной техники

–А я смогу? Неужели это девушке под силу?
–Ты слыхала про леди Аду Байрон? – рассмеялся
Мик. – Дочь премьер-министра и королева машин!
Ада Байрон, ученица самого Беббиджа!
Лорда Беббиджа, отца разностной машины и
Ньютона современности!
Уильям Гибсон и Брюс Стерлинг,
<<Машина различий>>, 2002

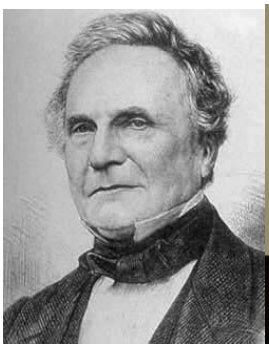
Идею о возможности построить <<логическую машину>> впервые в достаточно четкой форме высказывал знаменитый немецкий математик и философ Готфрид Вильгельм Лейбниц.[2]

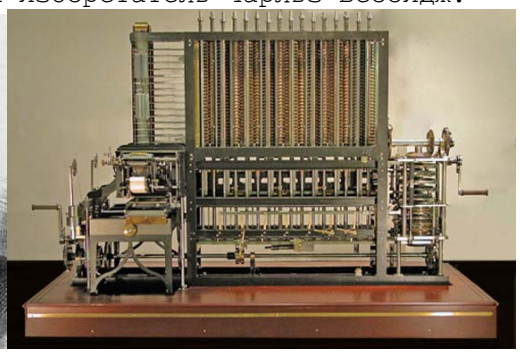
Он же предлагал использовать в ней двоичную систему. Однако до реализации этой идеи у него руки не дошли по понятным причинам. Алгебры логики в то время конечно еще не было, но Лейбниц выдвигал идеи и в этом направлении. Например, он

[2] Лейбниц является одним из героев серии фантастических романов Нила Стивенсона <<<Барочный цикл >>, недавно опубликованной в русском переводе. Герой этих романов Дэниел Уотерхауз, общий знакомый Лейбница и Ньютона (который тоже неоднократно появляется в этих романах), пытается построить логическую машину Лейбница (участвуя попутно в различных авантюрах).

писал, что когда-нибудь люди вместо того, чтобы спорить о том, правильно рассуждение или нет, просто сядут за вычисления, которые и решат спор.

Попытку реализовать идеи Лейбница предпринял в первой половине 19-го века английский математик и изобретатель Чарльз Беббидж. [3]





Он эти идеи ясно оформил и развил, добавив к ним много нового, и даже построил первый вариант этой машины.

Однако в то время физика электрических явлений была в недостаточно развитом состоянии (хотя современник и соотечественник Беббиджа Майкл Фарадей уже сделал свои замечательный открытия в области электромагнетизма) и Беббидж решил строить свою машину из механических деталей. Понятно, что ее быстродействие не могло быть высоким. Но проблема была еще в том, что эти детали надо было изготовлять вручную с высокой точностью, работе машины мешало трение и прочие неприятные механические эффекты. Преодолеть эти трудности Беббидж не смог, что совершенно не удивительно. Его проект явно опережал свое время на сотню лет, и был реализован (на другой технической базе) в сороковые годы 20-го века в США, где был построен первый электро-механический компьютер, а потом и компьютер на электронных лампах. Любопытно, что два человека, имена которых скорее всего вспоминают в связи с историей развития компьютеров – Буль и Беббидж- жили в одной стране и в одно время, знали о существовании друг друга, но совершенно не предполагали, что сферы их деятельности будут пересекаться в будущем и их достижения будут упоминаться рядом, тем более что Беббидж, хотя и предполагал использовать в своей машине двоичную систему, совершенно не подозревал о возможности использования алгебры логики для описания ее работы, так как в гораздо большей степени для этого была нужна точная механика.

После Буля к исследованиям в области алгебры логики присоединились и другие ученые (увы, Беббидж не заинтересовался логикой). Среди них можно назвать Августа де Моргана и Чарльза Доджсона, более известного под псевдонимом Льюис Кэрролл.[4]

Россия также имеет давние традиции исследований области алгебры логики. Существенный вклад в ее развитие внес казанский астроном Платон Сергеевич Порецкий (1846-1907).

---

[3] Интересно, что Беббидж, а также его светская знакомая Ада Августа Лавлейс, написавшая статью, популяризировавшую работы Беббиджа, в которой она объясняла, как можно использовать его машину, благодаря чему и вошла в историю как <<первая программистка>>, и ее отец, знаменитый поэт лорд Байрон, стали героями романа <<Машина различий>>, написанного У. Гибсоном и Б. Стерлингом в модном жанре альтернативной фантастики (и одновременно киберпанка). В этом романе описан мир, в котором научно-техническая революция началась в Англии в в двадцатые годы 19-го века. Лорд Байрон, вместо того, чтобы поддерживать разрушителей машин – луддитов, а потом уехать воевать за освобождение Греции, стал премьер-министром, а мозговым центром этой революции стал, естественно, лорд Беббидж (в настоящей реальности он лордом не был). Не отставали и французы – в романе упоминается " мощнейший вычислитель французской академии << Император Наполеон>>". Впрочем, основная интрига закручивается вокруг вымышленных персонажей, и машины Беббиджа играют в ней незначительную роль, хотя и дали название роману, по-английски звучащему как <<The Difference Engine>> - разностная машина (так и назывался в реальности первый вариант машины Беббиджа). Трудно сказать, то ли переводчик не знал об этом, и неправильно перевел название романа, то ли выбрал такой перевод умышленно, как содержащий тонкую игру смыслов.

[4] Кстати, Кэрролл кроме всемирно известных сказок написал и несколько замечательных научно-популярных книг, например <<Историю с узелками>>, которую неоднократно издавали в русском переводе. В этой книге есть большой раздел, посвященный логическим задачам и головоломкам, в котором Кэрролл развил свой собственный аппарат, ориентированный на решение подобных задач.

В его работах, например, фактически впервые появились дизъюнктивные и конъюнктивные нормальные формы булевых функций.[5] Он их использовал для решения булевых уравнений.

Идея о применении алгебры логики для моделирования работы электрических или электро-механических схем впервые была высказана в начале прошлого века известным голландским физиком Паулем Эренфестом[6]. Развивать эту идею дальше он не стал, и ее забыли до конца тридцатых годов, когда она была одновременно и независимо опубликована в СССР научным сотрудником физфака МГУ В. И. Шестаковым[7], в США инженером и математиком К. Шенноном и в Японии инженером-электриком А. Накасимой. По-видимому первым был Шестаков, но его диссертация была опубликована спустя несколько лет после ее написания и защиты, и Шеннон опубликовал свою статью раньше.

Шеннон рассматривал задачу моделирования работы электромеханических устройств, построенных из реле, и предложил в качестве модели понятие контактной схемы.

Если реле пропускало ток, то соответствующий ему контакт являлся замкнутым, и его состояние описывалось символом 1, а если нет, то контакт становился разомкнутым, и его состояние описывалось символом 0. С тех времен элементная база электронно-вычислительной техники полностью сменилась несколько раз, пройдя путь от электронных ламп довольно больших размеров до миниатюрных транзисторов, представляющих из себя зоны кремниевого кристалла, размеры которых измеряются нанометрами, и которых на кристалле может располагаться несколько миллионов. Такие электронные схемы в начале их появления назывались интегральными схемами, потом большими интегральными схемами (английская аббревиатура LSI), и наконец сверхбольшими интегральными схемами (аббревиатура VLSI).

Физика и химия происходящих в электронных схемах явлений также менялась вместе с изменением принципов их работы, но по прежнему для их математического моделирования используется аппарат алгебры логики, только вместо контактных схем (которые в определенных ситуациях также используются) применяются логические схемы из функциональных элементов. Разумеется, для изучения тонких аспектов поведения реальных электронных схем (например, для вычисления их различных числовых характеристик, скажем силы протекающих в них токов, потребляемой мощности, температуры различных зон) приходится использовать сложные физические модели, использующие, в частности, дифференциальные уравнения, но для моделирования процессов обработки информации, происходящих в схеме, применяется алгебра логики. Конечно, с определенной точки зрения алгебро-логическая модель функционирования электронной схемы является грубым приближением (низкий

[5] Американец Блэйк, занимавшийся дизъюнктивными нормальными формами в тридцатые годы двадцатого века, в своей диссертации ссылался на работы Порецкого.

[6] Кстати, долгое время жившим в России. Его женой была математик Татьяна Афанасьева. У них была дочь, тоже Татьяна и тоже математик.

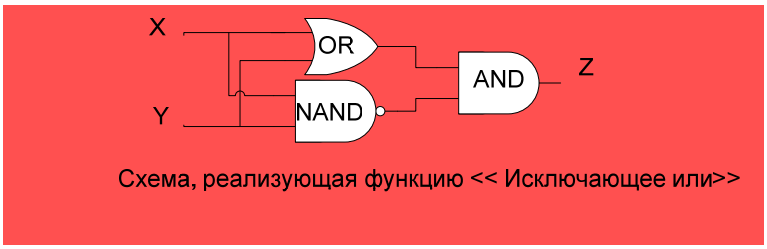
[7] Виктор Иванович Шестаков (1907-1987) – советский логик и теоретик-электротехник. К сожалению, его фотографии, также как и фотографий некоторых других упоминаемых здесь ученых, нам обнаружить не удалось.

потенциал обозначают нулем, высокий - единицей), но это приближение адекватно описывает работу схемы и без его применения ни анализ, ни синтез сколь либо сложных схем был бы невозможен.

Логические схемы компьютерной арифметики

В главе, посвященной Пуницланскому институту, я описал логическую схему И- НЕ, которая используется в пуницланских компьютерах, но не рассказал о том, как она может применяться в качестве составной части более сложных схем.
Александр Дьюдни, <<Планиверсум>>, 2010.

Если у нас нет в распоряжении элемента XOR, но есть элементы OR, AND и NAND (так часто обозначают элемент, реализующий отрицание конъюнкции, используется также аналогичное обозначение NOR), то можно построить логическую схему из функциональных элементов, реализующую функцию XOR. Эта схема состоит из трех элементов, соединенных друг с другом, как показано на следующем рисунке.



Схема, реализующая функцию << Исключающее или>>

Символы X и Y в этой схеме обозначают ее входы, теми же символами удобно обозначать булевы переменные (т.е. переменные, принимающие значения 0 или 1), значения которых определяют состояния этих входов в рассматриваемый момент. Символ Z обозначает ее выход. Если состояния входов схемы описываются значениями X и Y (иногда говорят, на входы поданы значения X и Y), то на выходе элемента OR появляется значение X(Y, на выходе элемента NAND – значение $\neg(XY)$,а на выходе элемента AND (он же является выходом всей схемы) – значение $(X \vee Y)(\neg XY)$. Согласно указанному выше тождеству это значение совпадает с $X \oplus Y$. Поэтому данная схема реализует функцию $f(x,y) = x \oplus y$.

Неформально говоря, схема из функциональных элементов есть произвольный способ соединения выходов некоторых элементов с входами других элементов, в котором не появляются замкнутые циклы. Наличие таких циклов в реальной микросхеме приведет к ее неустойчивой работе в некоторых ситуациях, а наличие циклов в логической схеме сделает невозможным корректное определение ее функционирования и в частности определение булевых функций, реализуемых выходами схемы.

Схемы из функциональных элементов имеют такое длинное название, чтобы их отличать от других видов схем, также предназначенных для реализации булевых функций, например, контактных схем. Так как другие виды схем далее не рассматриваются, то для краткости будем называть схемы из функциональных элементов просто схемами (в теоретическом программировании известно равносильное понятие, называемое неветвящейся программой).

Что такое сложность схемы

Размером (или сложностью) схемы называется число составляющих ее элементов. В рассмотренном примере сложность равна 3.

В практических приложениях реальные схемы размещены на кремниевом кристалле, и под размером схемы естественно понимать ее площадь. Площадь схемы определяется суммарной площадью составляющих ее элементов и площадью, занимаемой проводами, соединяющими элементы. Задача минимизации площади схемы является важной прикладной задачей (в последнее время ее актуальность снизилась, так как современные технологии позволяют укладывать на кристалл схемы из огромного числа элементов).

Оценить долю площади, занимаемой проводами довольно сложно (она сильно зависит от используемого алгоритма укладки элементов схемы, называемого в

англоязычной литературе placement, и от алгоритма разводки или трассировки проводов, по-английски называемого routing), и хотя у сложных схем она может быть довольно велика, грубо ее можно оценить как сумму площадей составляющих схему элементов. Поэтому иногда используется следующее обобщение понятия сложности схемы : сложность схемы – это сумма размеров (или весов) составляющих ее элементов (предполагается, что каждому элементу сопоставлено число, называемое весом или размером элемента).

Как правило, чем больше сложность, тем больше площадь.

Формулы как частный вид схем

В рассмотренном примере выход каждого элемента присоединяется к входу только одного другого элемента (говорят, что элементы схемы не имеют ветвлений) и схема имеет только один выход, и тем самым, реализует только одну функцию. Такие схемы часто называют формулами. Входы схемы в рассмотренном примере имеют ветвления (каждый из них присоединяется к входам двух элементов). Формулы, у которых и входы не имеют ветвлений, называют бесповторными формулами. Возможности таких формул для реализации булевых функций весьма ограничены.

Базисы и технологическая библиотека

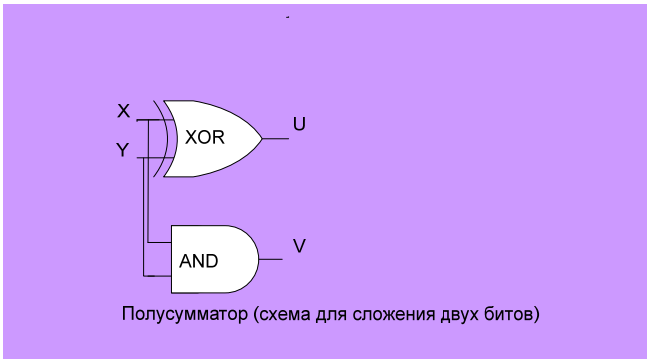
Рассмотренная схема построена из элементов OR, AND, NAND. Кружок при элементе NAND изображает инвертор (обозначаемый в англоязычной литературе NOT), но на самом деле технология такова, что элемент NAND составляется из отдельных частей, называемых в англоязычной литературе gate даже более просто, чем элемент AND, и имеет меньшую площадь и задержку при сравнимых электрических характеристиках.

Набор различных элементов, использованных при построении схемы, называется ее базисом. В теоретических исследованиях обычно используется базисы {OR,AND,NOT}, {OR,XOR,AND,NOT} или базис из всех двухвходовых элементов. На практике базис, который можно использовать при построении схем, обычно определен заранее и известен разработчику. В инженерной терминологии он называется технологической библиотекой.

Современные библиотеки содержат сотни элементов (в англоязычной литературе вместо термина элемент используют термин cell, что в переводится как <<ячейка>>), элементы могут иметь три, четыре, а иногда и более входов, иногда два или несколько выходов, логически одинаковые элементы могут иметь разные модификации, имеющие разные значения площади, задержки и разные электрически характеристики.

Полусумматор -схема для сложения двух битов

Для выполнения сложения двух однобитовых чисел делают обычно даже специальный логический элемент с двумя входами x,y и двумя выходами u,v, как бы составленный из элемента конъюнкции и элемента сложения по модулю два. Этот элемент часто называют полусумматором (по-английски half adder) и обозначают HA (или F2). Реализующая его схема выглядит следующим образом.



Полусумматор (схема для сложения двух битов)

Задача. Постройте в базисе {AND,OR, NOT} схему для полусумматора сложности 4.
Указание: воспользуйтесь формулой $x \oplus y = (x \vee y) \& \neg(x \& y)$.

Глубина схемы и ее задержка

Другой важной характеристикой схемы является ее глубина. Глубиной схемы называется максимальное число ее элементов, образующих цепь, соединяющую какой-либо вход схемы с одним из ее выходов. Например, у первой из рассмотренных выше схем глубина равна единице, а у второй --- двум.

Задержкой схемы называется время, прошедшее от момента появления сигнала на входах схемы (как правило, значения входов схемы стабилизируются в разные момент времени, и тогда отсчет начинается с того, чье значение стабилизировалось последним) до момента появления сигнала на ее выходе.

Глубина схемы – не менее важная характеристика схемы, чем ее сложность. Сложность логической схемы в значительной степени определяет площадь соответствующей реальной схемы, расположенной на кремниевом кристалле. Глубина же логической схемы в значительной мере определяет задержку реальной схемы. Как правило, чем больше глубина, тем больше задержка.

Сложность схемы часто не имеет существенного значения, так как современные технологии позволяют разместить на кристалле очень большие схемы. А минимизация задержки схемы очень важна, так как задержка комбинационной части многотактной схемы определяет ее тактовую частоту –чем меньше задержка, тем выше частота.

Теоретически вычислить задержку реальной схемы очень сложно. Цепей элементов схемы, соединяющих ее входы с выходами (эти цепи также называют путями), обычно довольно много и задержка схемы определяется задержкой по самому плохому в определенном смысле пути, который называется критическим. Задержка по данному пути определяется не только суммой задержек всех элементов, лежащих на этом пути. Следует учитывать также задержку соединяющих эти элементы проводов. Задержка элемента зависит от того, между каким его входом и каким его выходом она измеряется, а также от электрических характеристик самого элемента и элементов непосредственно с ним связанных в рассматриваемой схеме, она зависит от температуры схемы и даже от того, какие логические значения подаются в рассматриваемый момент на входы этого элемента и изменяется ли (и в какую сторону) значение на его выходе.

Тем не менее, хотя и не очень точно, задержку пути можно оценить как сумму задержек его элементов. Если задержки всех элементов равны, то эта величина, как правило, определяется глубиной схемы. [8] Разумеется, понятие глубины схемы можно расширить, допустив, что элементы базиса могут иметь произвольные неотрицательные задержки, а глубину цепи элементов определить как сумму их задержек.

Схема для сложения трех битов

Ниже приводится более сложный пример схемы, в которой есть и ветвления элементов и два выхода, которые не присоединяются ни к каким входам других элементов (но выход элемента, который присоединяется к входу другого элемента при желании тоже может быть объявлен выходом схемы).

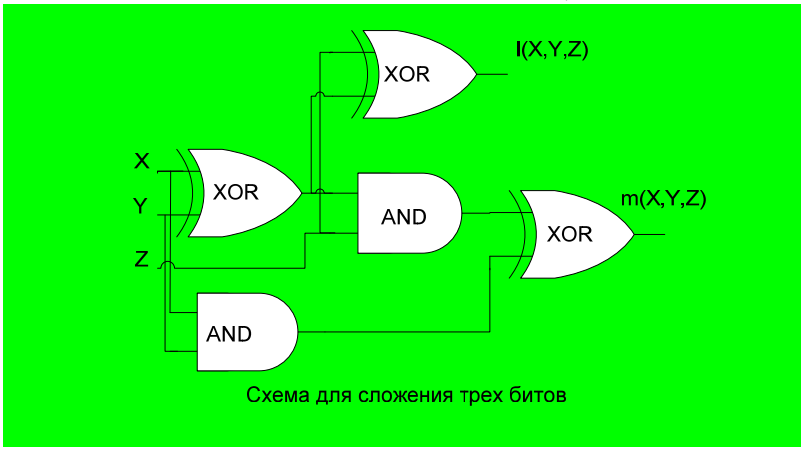


Схема для сложения трех битов

[8] Имеется замеченное В. М. Храпченко тонкое различие между глубиной и задержкой даже в предположении, что задержки всех элементов равны единице, а задержкой проводов и прочими эффектами пренебрегаем. Но здесь мы не можем касаться этого вопроса.

В этой схеме входы обозначены соответствующими им символами булевых переменных X,Y,Z, а на двух ее выходах реализуются функции $l(x,y,z)= x\oplus y\oplus z$ – сумма трех переменных по модулю два (называемая также линейной функцией трех переменных или счетчиком нечетности с тремя переменными), и функция $m(x,y,z) = xy\vee xz\vee yz$, называемая функцией голосования комитета из трех человек, мажоритарной функцией или короче – медианой.

Первая из этих функций может быть определена следующим словесным описанием. Ее значение по аргументам x,y,z вычисляется как остаток от деления обычной суммы x+y+z на 2. Равносильным образом можно определить эту функцию формулой $(x\oplus y)\oplus z$.

Медиану можно определить следующим словесным описанием. Она равна единице тогда и только тогда, когда не менее двух из ее аргументов единицы. Равносильным образом ее можно определить формулой $(xy\vee xz)\vee yz$ или чуть более короткой формулой $x(y\vee z)\vee yz$. Не так очевидно, что ту же функцию можно определить формулами $xy\oplus(xz\oplus yz) = x(y\oplus z)\oplus yz$.

Рассмотренная схема (ее в англоязычной литературе называют Full Adder и обозначают FA3) называется схемой сложения трех битов потому, что она действительно выполняет сложении трех одноразрядных чисел в двоичной системе по следующим формулам

$x+y+z = 2u+v$, $u = m(x,y,z) = (x\oplus y)z\oplus xy$, $v = l(x,y,z) = (x\oplus y)\oplus z$

Для непосредственной проверки этих формул нужно перебрать все 8 значений булевого трехмерного вектора (x,y,z) от (0,0,0) до (1,1,1). Но если учесть, что функции $l(x,y,z)$ и $m(x,y,z)$ симметрические, т.е. не зависят от порядка переменных, то перебор сокращается до четырех следующих вариантов: среди чисел x,y,z нет единиц, есть ровно одна единица, ровно две единицы, ровно три единицы. При этом следует заметить, что результат сложения трех битов в двоичной системе в рассматриваемых случаях находится следующим образом:

$0+0+0 = 0 = (00)_2$, $1+0+0 =1 = (01)_2$, $1+1+0 = 2 = (10)_2$, $1+1+1=3 = (11)_2$.

Задача. Построить в базисе {AND,OR, NOT} схему сумматора трех битов сложности 10.

Указание: воспользуйтесь формулами

$x\oplus y=(x\vee y)\neg(xy)$, $l(x,y,z) = (x\oplus y)\oplus z = ((x\oplus y)\vee z)\neg((x\oplus y)z)$,

$m(x,y,z)= (x\vee y)z\vee xy$.

Двоичные сумматоры -схемы для сложения двоичных чисел

Сложение двух n-разрядных чисел $(x_n,...,x_1)_2$ и $(y_n,...,y_1)_2$, как и в десятичной системе, приводит к появлению переносов в следующий разряд, которые необходимо учитывать в вычислении. Эти переносы также равны нулю или единице (если перенос равен нулю, то в ручном вычислении он фактически не выполняется, но логическая схема обязана правильно работать и в этом случае, ведь она не знает, какой перенос пришел из предыдущего разряда). Обозначим перенос из (i-1)-го разряда в следующий i-й разряд через w_i ($w_1=0$, потому что предыдущего разряда в этом случае просто нет). Тогда для вычисления z_i (i-го бита результата) нужно сложить биты x_i и y_i и бит переноса w_i. Это сложение выполняем по формулам

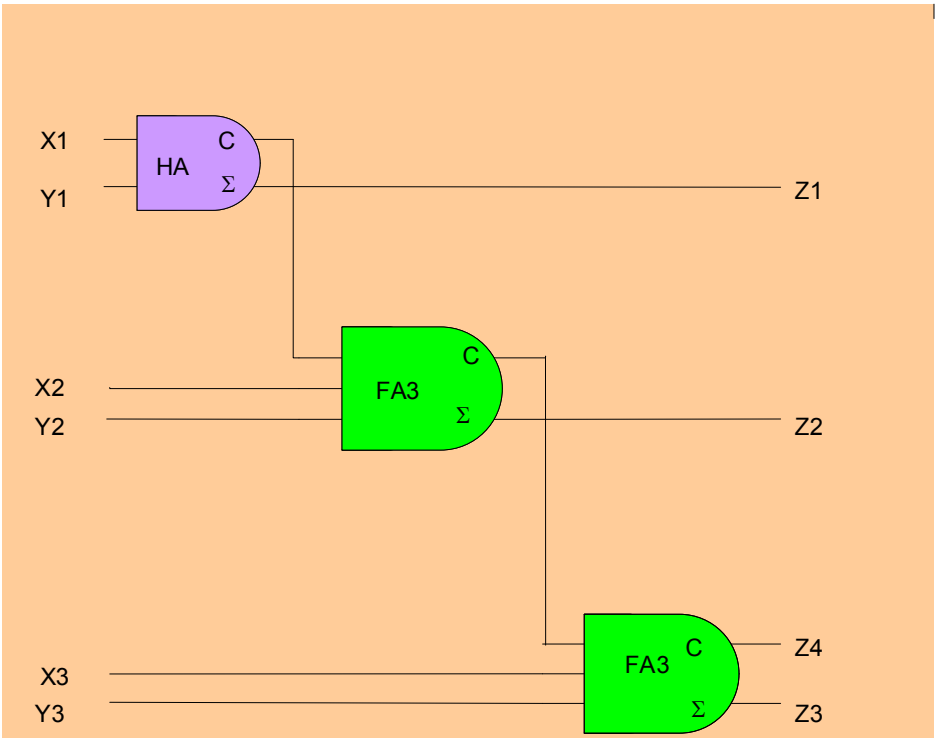
$x_i + y_i + w_i = 2v_i + u_i$, $v_i=m(x_i, y_i, w_i)$, $u_i=l(x_i, y_i, w_i)$

с помощью схемы FA3. Тогда $z_i=u_i=l(x_i, y_i, w_i)$, а следующий бит переноса $w_{i+1} = v_i=m(x_i, y_i, w_i)$.

При сложении n-разрядных чисел получается вообще говоря n+1-разрядное число. Его старший бит $z_{n+1}= w_{n+1}$ равен последнему переносу.

Задача. Докажите, что переносы действительно вычисляются по указанным формулам.

Схема сложения трехразрядных чисел приведена на следующем рисунке. Аналогичным образом выглядит и схема сложения n-разрядных чисел.



Последовательный сумматор трехразрядных чисел

Задача Докажите, что сложность указанного n-разрядного сумматора в базисе {AND,OR,NOT,XOR} равна 5n-3, а в базисе {AND,OR,NOT} – равна 10n-6.

Н.П.Редькин [9] доказал, что сумматоров меньшей сложности в базисе {AND,OR,XOR,NOT} не существует. Построенный сумматор является поэтому минимальной схемой. Но у этой схемы есть существенный недостаток – она имеет большую глубину. Ее глубина в базисе {AND,OR, XOR}равна 2n-1 при подходящем выборе схем, реализующих модули FA3.

Задача Проверьте, что это действительно так.

Другие схемы для сумматоров

Известно огромное число различных схем для сумматоров. Мы рассмотрим на примере трехразрядного сумматора схему, имеющую существенно меньшую глубину, чем схема из предыдущего раздела. Прежде чем ее построить, выведем явную формулу для вычисления переносов при сложении n-разрядных чисел. В предыдущем разделе было показано, что при сложении n-разрядных чисел $(x_n,\dots,x_1)_2$ и $(y_n,\dots,y_1)_2$ переносы вычисляются последовательно по формулам $w_{i+1} = v_i = m(x_i, y_i, w_i)$, где $w_0=0$. Вспоминая формулу $m(x,y,z) = (x \vee y)z \vee xy$, получаем, что

$$w_1 = x_1y_1,\quad w_{i+1}=(x_i \vee y_i)w_i \vee x_iy_i.$$

Для удобства обозначим $x_i \vee y_i$ через v_i, а x_iy_i – через u_i. Тогда предыдущие формулы перепишется в виде $w_{i+1}=v_iw_i \vee u_i$, $w_1=0$. Из них последовательно получаем, что

$$w_2 = v_1\&0 \vee u_1 = u_1,\quad w_3 = v_2w_1 \vee u_2 = v_2u_1 \vee u_2,$$

$$w_4 = v_3w_3 \vee u_3 = v_3(v_2u_1 \vee u_2) \vee u_3 = v_3v_2u_1 \vee v_3u_2 \vee u_3.$$

Задача. Докажите, что

$$w_{n+1} = v_n \& v_{n-1} \& \dots \& v_2 \& u_1 \vee v_n \& v_{n-1} \& \dots \& v_3 \& u_2 \vee \dots \vee v_n \& v_{n-1} \& u_{n-2} \vee v_n \& u_{n-1} \vee u_n.$$

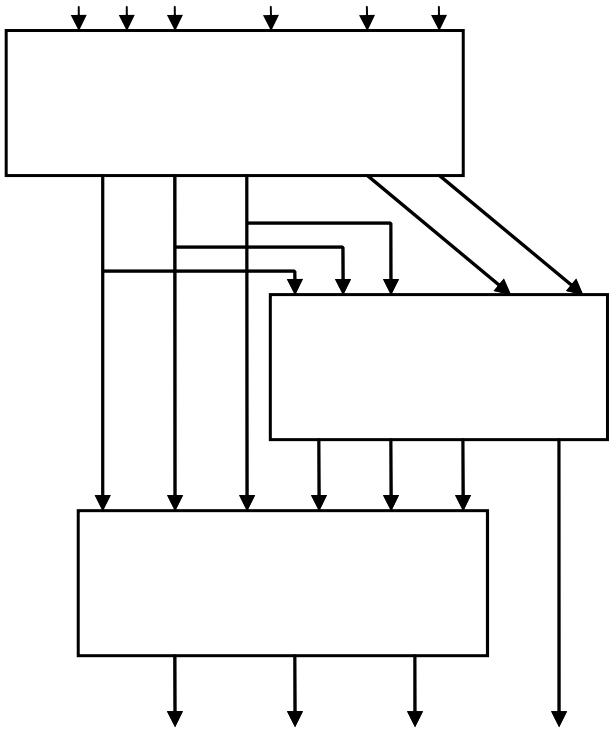
(в формуле использованы знаки конъюнкции, а не пробелы, как ранее, чтобы напомнить, что пробел у нас был всего лишь сокращенным обозначением конъюнкции; скобки в формуле опущены, так как предполагается, что операция конъюнкции связывает сильнее, чем дизъюнкция, подобно тому, как в школьной алгебре считается, что умножение связывает сильнее, чем сложение и соответствующие скобки опускаются).

[9] Николай Петрович Редькин – профессор кафедры дискретной математики мехмата МГУ, известный специалист в области синтеза и тестирования логических схем.

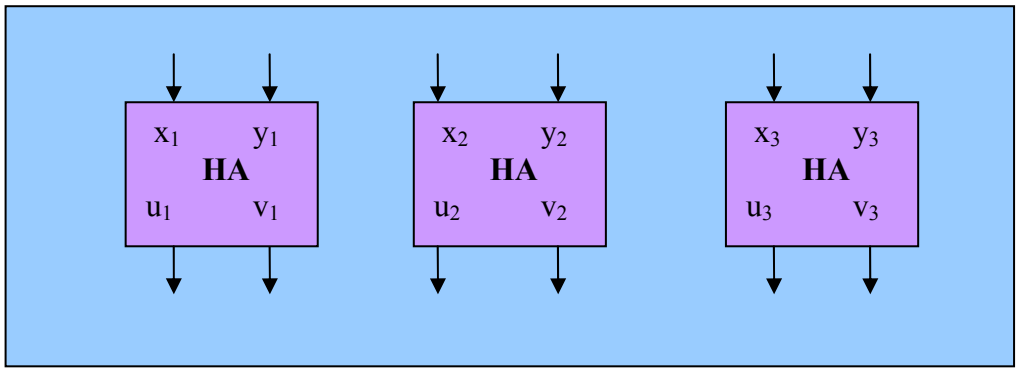
Задача Если определить v_i как $x_i \oplus y_i$, то справедливо равенство

$w_{n+1} = v_n \& v_{n-1} \& \ldots \& v_2 \& u_1 \oplus v_n \& v_{n-1} \& \ldots \& v_3 \& u_2 \oplus \quad \oplus v_n \& v_{n-1} \& u_{n-2} \oplus v_n \& u_{n-1} \oplus u_{nv}$

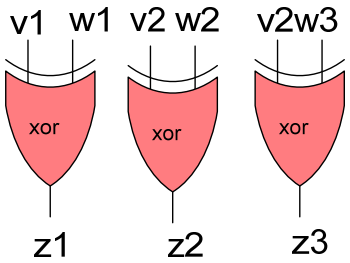
Вспоминая формулы $z_i = x_i \oplus y_i \oplus w_i$, видим, что схему для сложения можно составить из трех блоков (модулей), изображенных на следующем рисунке.



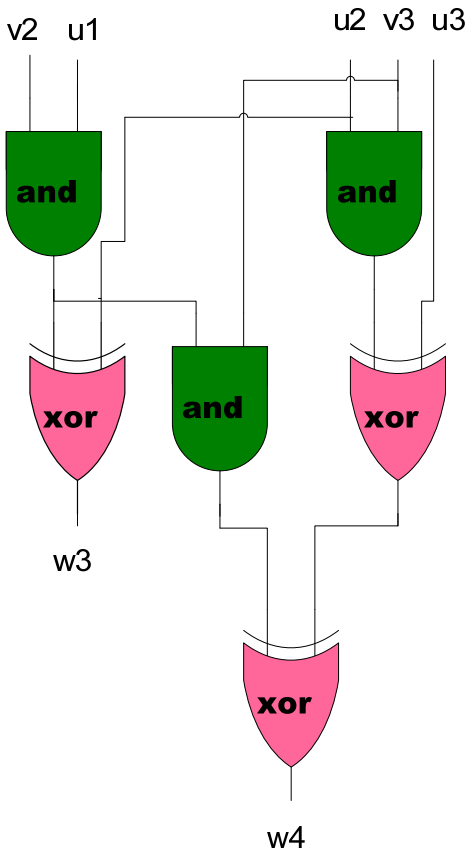
Модуль UV реализует функции $u_i = x_i y_i$, $v_i = x_i \oplus y_i$ и состоит из параллельно расположенных полусумматоров, как показано на следующем рисунке



Модуль Z реализует функции $z_i = x_i \oplus y_i \oplus w_i = v_i \oplus w_i$ и состоит из параллельно расположенных элементов XOR, как показано на следующем рисунке



Модуль W реализует переносы w_i и изображен на следующем рисунке



Задача. Проверьте, что в базисе {AND,OR,XOR} сложность построенной схемы равна 15, а глубина равна 5. В базисе {AND,OR,NOT} сложность этой схемы равна 42, (так как сложность схемы, реализующей XOR, равна 4), а глубина в базисе {AND,NAND, OR} равна 9.

Можно построить лучшую схему, если в модулях UV и W заменить все элементы XOR на элементы OR. Но тогда придется в модуле Z заменить все элементы XOR на модули, реализующие функции l(x,y,z) = x⊕y⊕z. Очевидно, каждый такой модуль реализуется схемой из двух элементов XOR.

Задача. Проверьте, что новый вариант схемы в базисе {AND,OR,NOT} имеет сложность равна 33 (так как элементы, реализующие $x_i \vee y_i$ можно использовать при реализации функций $x_i \oplus y_i$ в модуле Z), а глубина в базисе {AND,NAND, OR} равна 6.

Параллельные сумматоры с почти минимальной глубиной

Глубина указанной выше схемы последовательного n-разрядного сумматора равна 2n-1. Это очень много и построенная таким образом реальная схема будет иметь большую задержку. На практике используются схемы, имеющие одновременно малую сложность, не превосходящую Cn (где C – небольшая константа) и малую глубину, приблизительно равную $2\log_2 n$. В.М. Храпченко[10] в 1970 г. построил схему малой сложности и глубины, асимптотически равной $\log_2 n$ (т.е. равную $(1+o((n))\log_2 n$, где o(n) – некоторая величина, которая стремится к нулю, т.е. может стать сколь угодно малой при достаточно большом n)[11]. Он же недавно доказал, что глубина сумматора не может быть меньше

$$\log_2 n + \log_2(\log_2(\log_2 n)).$$

Поэтому построенная им схема имеет асимптотически минимальную глубину. Однако его схема имеет меньшую глубину по сравнению с обычными схемами только при n порядка тысячи. Тем не менее существует некоторая модификация его схемы с глубиной приблизительно равной $\log_\varphi n$, где $\varphi = (5^{1/2}+1)/2$, и эта схема имеет

[10] Валерий Михайлович Храпченко, научный сотрудник ИПМ РАН, известный специалист в области теории сложности булевых функций.

[11] Двоичный логарифм $\log_2 x$ обладает следующим свойством: при $2^n \le x \le 2^{n+1}$ справедливо неравенство $n \le x \le n+1$. Для понимания дальнейшего больше ничего о нем знать не нужно.

глубину меньшую, чем стандартные схемы, уже начиная с n = 8. В 2008 г. М.И.Гринчук[12] построил схему глубины не большей

$$\log_2 n + \log_2(\log_2 n)+6,$$

которая уже при малых n имеет меньшую глубину, чем все известные схемы.

Схемы для умножения - мультиплеры

Задача построения оптимальных схем для умножения n-разрядных чисел оказалась еще труднее, чем задача о построении оптимальных сумматоров. Легко построить схему для умножения n-разрядных чисел в базисе {OR,AND,XOR,NOT} сложности приблизительно равной $6n^2$.

Задача. Попробуйте построить такую схему.

Для этого можно использовать указанную выше схему для сумматора. Однако ее глубина будет велика. В начале 60-х годов несколько исследователей (в СССР Столяров и Офман, в США Авиценис и Уоллес) независимо построили схему для умножения сложности порядка n^2 и глубины порядка $\log_2 n$. В смысле глубины эти схемы по порядку оптимальны, но до сих пор остается нерешенной задача построения схемы для умножения асимптотически минимальной глубины. В смысле сложности эти схемы оказались далеки оптимальных.

Мало кто знает, что относительно недавно были открыты гораздо более быстрые алгоритмы умножения и деления многозначных чисел и многочленов. Первый такой алгоритм придумал в 1962 г., отвечая на вопрос академика А. Н. Колмогорова, аспирант мехмата МГУ А. А. Карацуба[13]. Алгоритм Карацубы умножает n-разрядные двоичные числа за $Cn^{1,6}$ операций (точнее, за $Cn^{\log_2 3}$ операций, где C – некоторая константа, не зависящая от n, а двоичный логарифм трех равен 1,58496…). Используя его, можно построить схему для умножения n разрядных чисел сложности не больше $Cn^{\log_2 3}$.

Идею метода Карацубы можно пояснить на следующем примере. Пусть перемножаются восьмизначные числа в обычной десятичной системе

$$U = (u_1u_2...u_8)_{10} \quad \text{и} \quad V = (v_1v_2...v_8)_{10}.$$

Представим их как двузначные числа в системе счисления по основанию 10^4=10000:

$$U = (U_1U_2)_{10000},\ V = (V_1V_2)_{10000}.$$

Тогда их произведение можно представить в следующем виде:

$$UV = U_1V_1\ 10^8 + ((U_1 - U_2)(V_2 - V_1) + U_1\ V_1 + U_2V_2)10^4 + U_2\ V_2.$$

Эта формула сводит умножение 8-значных чисел к трем операциям умножения и шести операциям сложения и вычитания 4-значных чисел (с учетом переносов в следующие разряды). Обычный способ требует четырех умножений и трех сложений-вычитаний, но так как три раза сложить 4-значные числа можно быстрее, чем один раз перемножить, то метод Карацубы уже 8-значные числа перемножает быстрее, чем обычный школьный метод.

---

[12] Михаил Иванович Гринчук, доцент каф. дискретной математики мехмата МГУ, впоследствии сотрудник фирмы LSI Logic, известный специалист в области теории сложности булевых функций. Окончил ФМШ (ныне СУНЦ) МГУ. Золотой медалист международной математической олимпиады 1981 г.

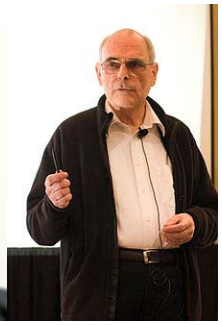
[13] Выдающийся специалист по теории чисел профессор Анатолий Алексеевич Карацуба скончался в конце 2008 г.

Впоследствии, в 1963 г. студентом мехмата МГУ А.Л. Томом[14], а в 1970 г. немецкими математиками Фолкером Штрассеном и Арнольдом Шенхаге были построены еще более быстрые алгоритмы для умножения чисел. Алгоритм Тоома имеет оценку сложности $n^{1+o(1)}$.

Существенно уменьшить в этой оценке показатель степени уже нельзя, так как очевидно, что сложность умножения n-разрядных чисел не может быть меньше n. Схема для умножения, построенная методом Шенхаге-Штрассена, имеет сложность не больше

$$Cn \log (\log n),$$

где C- константа, не зависящая от n.



В 2007 г. американский математик Мартин Фюрер предложил чуть более быстрый алгоритм умножения, чем алгоритм Шенхаге-Штрассена.

В оценке сложности его алгоритма вместо повторного логарифма стоит функция, которая с ростом n растет медленнее, чем k-кратный повторный логарифм при любом k. Однако алгоритмы Фюрера и Шенхаге-Штрассена имеют меньшую сложность, чем алгоритм Карацубы, только при огромных значениях n, и поэтому на практике пока не применяются.

Схемы для деления

Американский математик Стивен Кук [15] в своей диссертации построил схему для деления 2n-разрядного числа на n-разрядное, у которой сложность по порядку не превосходит сложность умножения n-разрядных чисел.

[14] Андрей Леонович Тоом , известный специалист в области теории сложности булевых функций и эргодической теории, ныне работает в Америке. Победитель международной математической олимпиады 1959 г.

[15] Известный американский математик, лауреат премии имени Тьюринга.

Известно также, что нижняя оценка сложности схемы для деления по порядку не меньше нижней оценки сложности умножения. Поэтому в смысле оценок сложности деление не представляет ничего нового в сравнении с умножением.

Однако долгое время наилучшей верхней оценкой глубины схем для деления по порядку была $C(\log_2 n)^2$. В последствии были найдены схемы для деления с глубиной не большей $C\log_2 n$, но их сложность оказалась очень велика. Американцы Рейф и Тейт построили схемы для деления глубины не большей $C\log_2 n \log_2(\log_2 n)$ и сложности не большей $Cn \log_2 n \log_2(\log_2 n)$, однако и эти схемы, как и схемы Шенхаге-Штрассена и Фюрера пока не нашли практических применений, так как начинают превосходить используемые на практике схемы при огромных значениях n.

В рассмотренных выше примерах схем для арифметических операций предполагалась что операнды (числа, над которыми выполняются операции) положительны. Можно конечно построить схемы для выполнения операций и с числами, которые могут быть как положительными, так и отрицательными.

Как представляются отрицательные числа в компьютере

Когда было введено понятие отрицательных чисел,
меньших нуля, математику, как науку, в которой
самым важным является ясность и очевидность,
накрыло непроницаемым облаком и ввергло в
лабиринт парадоксов, один загадочней другого.
Лазарь Карно

Вычитание в подлинном смысле слова
происходит при вычислении разности
двух чисел одинакового знака.
Б. Парлетт, 1980

Обычный школьный способ записи отрицательных чисел состоит в постановке знака минус перед записью модуля этого числа. Этот способ в компьютерной арифметике называется прямым кодом. В случае использования в компьютере, у него есть очевидный недостаток, а именно неоднозначность записи нуля в этой системе (<<проблема минус нуля>>). Для компьютера, как правило, удобнее дополнительный код, в котором, например, число $-(123456789)_{10}$ записывается как

$$-10^9+1+(987654321)_{10} = (1987654322)_{10}.$$

Единица в старшем разряде является символом знака, и означает, что данное число отрицательное. Запись $(a_0a_1...a_n)_{10}$ n-разрядного числа в дополнительном коде во всех случаях означает число $-a_0 10^n+(a_1... a_n)_{10}$, где $(a_1... a_n)_{10}$ - обычная запись положительного n-разрядного числа. Если символ знака $a_0=0$, то очевидно $(0a_1...a_n)_{10}= (a_1...a_n)_{10}$. Дополнительный код удобен еще и тем, что при вычитании из меньшего числа большего с помощью обычного алгоритма вычитания как раз и получается разность, записанная в дополнительном коде. Например, если вычесть из меньшего числа $(089)_{10}$ большее число $(098)_{10}$, то получится запись

$$\begin{array}{r} 089 \\ - \\ 098 \\ \hline 191 \end{array}$$

так как в старший разряд мы должны поместить <<займ>>, равный единице. Однако такой результат выглядит странно, и при использовании прямого кода в этом случае

числа переставляют, опять выполняют вычитание, получают уже положительное число, и перед ним ставят знак минус. Но при использовании дополнительного кода эти трудности исчезают, так как полученное число $(191)_{10}$ со знаковым разрядом 1 в дополнительном коде означает -100+91 =-9, и это верный результат.

На самом деле в компьютере дополнительный код, конечно, используется для двоичной системы, а не для десятичной. Применение дополнительного кода удобно еще и тем, что позволяет выполнять сложение чисел любых знаков тем же алгоритмом, который осуществляет сложение положительных чисел. Например, если компьютер был бы 8-разрядный, то сложение положительного и отрицательного числа он бы выполнял следующим образом

$$\begin{array}{r} 00111010 \\ + \\ \hline 10011111 \\ \hline 11011001 \end{array}$$

Первое слагаемое здесь равно

$$(00111010)_2 = (0111010)_2 = 32+16+8+2 =58,$$

второе слагаемое равно

$$(10011111)_2 = -128+(0011111)_2 = -128+16+8+4+2+1 = -97,$$

сумма равна

$$(11011001)_2 = -128 + (1011001)_2 = -128+64+16+8+1=-39,$$

что очевидно правильно, так как 58+(-97) = -39.

Сложение отрицательных чисел выполнялось бы, например, так:

$$\begin{array}{r} 10111010 \\ + \\ \hline 11011111 \\ \hline 10011001 \end{array}$$

Получившийся в результате переноса девятый бит просто отбрасывается, и результат оказывается равным

$$(10011001)_2 = -128+(0011001)_2 = -128+16+8+1 = -103.$$

Впрочем, если получившийся девятый бит воспринимать как знаковый, то результат будет таким же :

$$(110011001)_2 = -256+(10011001)_2 = -256+128+(0011001)_2 = -128+(0011001)_2 = -103.$$

Он, очевидно, правильный, так как

$$(10111010)_2 = -128+(0111010)_2 = -128+32+16+8+2=-70,$$

$$(11011111)_2 = -128 +(1011111)_2 = -128+64+16+8+4+2+1 =-33,$$

и -70+(-33) = -103. Однако, если абсолютная величина суммы слишком велика (в случае 8-разрядных чисел больше 127), то результат сложения будет неправильным (будет иметь даже неверный знак), как в следующем примере

$$\begin{array}{r} 10111010 \\ + \\ \hline 10011111 \\ \hline 01011001 \end{array}$$

В результате сложения отрицательных чисел $(10111010)_2 = -70$ и $(10011111)_2 = -97$ у нас получилось не -167, а $(01011001)_2 = 64+16+8+1=89$ (правда, возник перенос в девятый разряд, но мы условились их игнорировать).

Конечно, это неприятное явление является недостатком дополнительного кода. Но обратим внимание на два обстоятельства. Во-первых, число -167 (и вообще любое число, большее по абсолютной величине 128) невозможно изобразить в нашей системе 8-разрядных знаковых чисел в дополнительном коде, так как в этой системе записываются только числа из отрезка [-128,127], поэтому появление числа, большего 128 вызывает переполнение. Во-вторых, почти правильный ответ мы все же получили, так как он отличается от правильного ответа на 256, другими словами, наш алгоритм выполняет *сложение по модулю* $2^8 = 256$. Причина этого явления, конечно в том, что мы отбросили появившийся девятый разряд. В реальных компьютерах все происходит примерно также, только длина машинного слова у них сейчас 64 разряда (а когда-то было и восемь).

Но у дополнительного кода есть и недостатки. Складывать и вычитать в нем действительно очень удобно. А вот умножать и делить --- не очень. Для умножения

удобнее упоминавшийся выше (и хорошо всем известный) прямой код. И вычислять абсолютную величину числа в дополнительном коде не совсем тривиально. Тем не менее, в цифровой технике почти повсеместно используется именно дополнительный код.

Задача. Докажите, что абсолютная величина числа $(a_8...a_1)_2$, заданного в дополнительном коде, равна $(a_7 \oplus a_8, a_6 \oplus a_8, \ldots, a_1 \oplus a_8) 2 + a_8$.

Кроме дополнительного кода изредка используют также обратный код, в котором, например число $-(123456789)_{10}$ записывается как $(876543210)_{10}$ (каждая цифра дополняется до 9.) В этом коде сложение и вычитание производятся фактически по модулю $10^9 - 1$ (с точностью до слагаемого 10^9-1).

Немного о теории сложности булевых функций

Усложнять легко – упрощать сложно
Древняя китайская мудрость

Ее основателем является американский инженер и математик ирландского происхождения Клод Шеннон, который в 1949 г. опубликовал статью, где получил совпадающие по порядку оценки сложности реализации класса всех булевых функций контактными схемами.

Шеннон заметил, что сложность различных функций от одного и того же числа переменных может сильно различаться между собой, и изучил в этой статье асимптотическое поведение функции $L_k(n)$, которую определил как максимальное значение $L_k(f)$ – сложности реализации произвольной булевой функции $f = f(x_1,\ldots,x_n)$ от n переменных. Он показал, что $L_k(n)$ не превосходит $(4+o(n))2^n/n$, где $o(n)$ стремится к нулю при n, стремящемся к бесконечности.

Оказалось, что эта оценка по порядку величины, вообще говоря, неулучшаема, так как он в той же работе показал, что почти все функции n переменных имеют сложность не меньше $(1-o(n))2^n/n$. Словосочетание <<почти все>> имеет здесь строгий математический смысл. Оно означает, что доля функций от n переменных у которых сложность меньше, чем $(1-o(n))2^n/n$, стремится к нулю при n, стремящемся к бесконечности.

Несколько лет спустя американский математик Д. Маллер перенес результаты Шеннона на случай реализации булевых функций схемами из функциональных элементов в базисе{OR,AND,NOT}. Легко показать, что при переходе к любому другому конечному полному базису сложность любой функции по порядку не изменяется.

В 1958 г. О.Б.Лупанов[16] улучшил результаты Малера и Шеннона, и показал что для обоих видов схем сложность реализации ими почти всех булевых функций n переменных асимптотически равна сложности самой сложной функции n переменных, и эта сложность равна $(1+o(n))2^n/n$. В терминах функций $L_k(n)$ и $L(n)$ (которые по предложению Лупанова называются функциями Шеннона) его результаты можно записать так: $L_k(n) = (1+o(n))2^n/n$, $L(n) = (1+o(n))2^n/n$.

Несмотря на то, что почти все булевы функции n переменных имеют экспоненциально высокую схемную сложность, не известно ни одного примера булевой функции n переменных (а точнее говоря, последовательности функций $\{f_n = f_n(x_1,\ldots,x_n)\}$), для которой, например в случае базиса {OR,AND,NOT}, получена хотя бы нелинейно растущая нижняя оценка сложности. Эффективное (в разумном смысле) построение такой функции вместе с доказательством того, что ее сложность растет быстрее любого полинома от n, решило бы в отрицательном смысле известную проблему P =? NP.

[16] Олег Борисович Лупанов (1932-2006) – выдающийся ученый в области математической кибернетики, декан мехмата МГУ (1981-2006). Его учениками были упоминавшиеся выше Редькин, Храпченко, Гринчук, Тоом.

В случае реализации булевых функций некоторыми другими типами схем, или схемами с наложенными на них ограничениями, эффективные нелинейные нижние оценки сложности булевых функций получены. Например, для схем в неполном базисе {OR,AND} А. А. Разборов [17] получил нижнюю оценку вида $n^{c \log n}$ для так называемого логического перманента, а А. Е. Андреев[18] получил для специально (но эффективно) построенной монотонной булевой функции n переменных оценку $2^{h(n)}$, где $h(n) = n^{1/3-o(n)}$, o(n) стремится к нулю при n, стремящемся к бесконечности.

Арифметические коды

В арифметике и начинающие математики и профессора
могут допустить ошибку и выполнить неправильные вычисления.
Томас Гоббс, <<Левиафан>>, 1651.

Можно установить точно положение ошибки и даже исправить ее, в предположении, что она только одна. Для этого надо применить так называемые арифметические коды. Приведем их простейший пример.

Допустим, что при умножении десятичных чисел получилось пятнадцатизначное число с ошибкой в одном разряде. Для нахождения величины ошибки применим проверку по модулю 9 и найдем, что она по модулю 9 равна a. Если ошибка произошла в i-ом разряде, то величина ошибки в произведении равна $a10^i$ или $(a-9)10^i$,а если a = 0, то или ошибки не было, или она равна $\pm 9 \cdot 10^i$.

Далее применим проверку по модулю 31. После нее станет известно значение ошибки $b10^i$ по модулю 31. Если остаток по модулю 31 равен нулю, то ошибки не было. Пусть он не равен нулю. Выпишем остатки от деления чисел $10, 10^2, \ldots, 10^{15}$ на 31. Они равны 10, 7, 8, 18, 25, 2, 20, 14, 16, 5, 19, 4, 9, 28, 1. Заметим, что невозможно равенство по модулю 31 чисел $a10^i$ и $(a-9)10^j$, так как тогда при некоторых a = 1,2,3,4 и i =0,1,…,14 были бы равны по модулю 31 числа $a10^i$ и a-9, а невозможность этого проверяется непосредственно с помощью вычисленной выше таблицы остатков. Точно также проверяется невозможность совпадения по модулю 31 чисел $9 \cdot 10^i$ и $(-9) \cdot 10^j$. Вычисляя остатки по модулю 31 у всех чисел $a10^i$ и $(a-9)10^i$, при i = 1,…,15 и сравнивая их с найденным ранее остатком, находим значение i и точную величину ошибки a или a-9. Аналогично поступаем в случае ошибки $\pm 9 \cdot 10^i$.

Приведенный пример иллюстрирует принципиальную возможность построения арифметического кода, кажущуюся на первый взгляд парадоксальной. Прикладного значения при ручных вычислениях он не имеет хотя бы потому, что в данном случае проще еще раз перемножить эти числа, чем выполнять указанные выше операции.

Но такие алгоритмы можно применить для контроля правильности работы арифметических схем и этот контроль может быть осуществлен специальным блок в схеме. В рассматриваемом случае сложность схемы со встроенным арифметическим кодом рассмотренного вида увеличивается не более чем в два раза. Если рассмотреть подобные коды с большой длиной (p-1)/2 и проверочным множителем 9p, где p -- простое число вида 280k + 31, такое, что $10^{(p-1)/2}$ при делении на p дает остаток 1, а 10^k, где $k<(p-1)/2$, при делении на p дает остатки, отличные от 1, то сложность исправления ошибки будет мала при больших p в сравнении со сложностью умножения (p-1)/4-разрядных чисел.

На самом деле для повышения надежности арифметических устройств нужно использовать не десятичные, а двоичные коды, но они устроены подобным же образом.

Задача. Примените указанный арифметический код для расшифровки ребуса

[17] Александр Александрович Разборов, известный специалист по теории сложности, выпускник кафедры математической логики мехмата МГУ, ныне работает в Чикаго. Золотой медалист международной математической олимпиады 1979 г. Окончил физ-мат школу N 2 г. Москвы.

[18] Александр Егорович Андреев – известный специалист по теории сложности, профессор кафедры МАТИС мехмата МГУ, впоследствии сотрудник фирмы LSI Logic, сейчас занимается автоматизацией проектирования сверхбольших интегральных схем.

9

×

******

×

31

×

*****

425021067*

Можно с помощью этого кода расшифровать и ребус, котором ровно в одном разряде результата имеется ошибка, но неизвестно в каком. Тот же код можно использовать для демонстрации арифметического фокуса: Вы предлагаете Вашему другу задумать два не слишком больших числа, перемножить их и результат умножить на якобы случайное число 279, а потом сообщить Вам ответ с одной ошибкой в каком-нибудь разряде. Используя указанный код (если перед эти предварительно подготовить все нужные таблицы и немного потренироваться), вы быстро укажете эту ошибку. Заметьте, что полный перебор вариантов требует в случае 15-разрядного ответа 150-кратного деления предполагаемого ответа на 279.

Коды, исправляющие ошибки

Золотая карта теперь была в круглых дырочках, словно швейцарский сыр. Все они располагались в узлах координатной сетки мсье Декарта, однако не все узлы были пробиты. Результат являл собой странное смешение упорядоченного и случайного; таким наверное, предстает четко отпечатанный, но зашифрованный текст.
Нил Стивенсон <<Система мира>>, 2011.

Ошибки появляются не только из-за неправильных вычислений. Они возникают при передаче информации как по проводам, так и с помощью радиосвязи, а также из-за ненадежной работы устройств, хранящих и считывающих эту информацию (дисководов для чтения DVD-дисков, жестких дисков компьютера, различных внешних запоминающих устройств и т.п.) Эти ошибки можно найти и исправить с помощью кодов, исправляющих ошибки (их называют также самокорректирующимися кодами или кодами, контролирующими ошибки). Теория построения таких кодов (теория кодирования) --- один из важнейших разделов современной информатики, тесно связанный со многими разделами современной математики (алгеброй, теорией чисел, комбинаторикой и др.). Здесь у нас есть возможность рассказать только о простейшем классе кодов, исправляющих ошибки - кодах Хемминга, исправляющих одну ошибку. Известно много различных семейств кодов, исправляющих много ошибок, но, к сожалению, их построение слишком сложно, что бы излагать его школьникам.

Коды Хемминга были впервые опубликованы американским математиком Р. Хеммингом[19] около 50 г. двадцатого века.

Независимо от Хемминга эти коды были открыты также американским математиком М. Голеем. Частный случай этих кодов был указан[20] еще в конце 40-х годов в знаменитой статье К. Шеннона, заложившей основы теории информации. Этот частный случай мы рассмотрим далее, но прежде чем приступить к нему, разберем одну интересную задачу, опубликованную Эдуардом Люка во второй половине 19-го века в его книге по занимательной математике. В этой задаче Люка предвосхитил

[19] Ричард Уэсли Хэмминг (1915-1998) – известный американский ученый в области теории информации и вычислительной математики. Лауреат премии Тьюринга и др. почетных наград.
[20] Со ссылкой на Хэмминга.

основную идею построения кодов Хемминга. Фактически задача Люка - это арифметический фокус, который он назвал

<<Волшебный веер>>

Для выполнения фокуса исполнитель заранее заготавливает пять полосок бумаги, на которых написаны числа от 1 до 31 (на каждой полоске по 16 чисел, на разных полосках написаны разные наборы чисел). Зрителю предлагается задумать число от 1 до 31. Фокусник предлагает ему сказать, в каких полосках он видит задуманное число, а в каких - нет. После этого фокусник мгновенно называет это число.

Конечно, суть фокуса - в наборах чисел, написанных на полосках (они и составляют <<волшебный веер>>).На первой полоске написаны числа 1,3,…, 31 (все нечетные числа). На второй полоске- числа 2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31 (можно заметить, что каждая следующая четверка чисел получается из предыдущей прибавлением 8 к каждому числу четверки). На третьей полоске - числа 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31, на четвертой полоске- числа 8, 9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28,29, 30, 31, и на последней - все числа от 16 до 31 (все эти полоски изображены ниже в виде таблицы).

0	0	0	0	1	1	0	0	0	1	0	2	0	0	1	0	0	4	0	1	0	0	0	8	1	0	0	0	0	16
0	0	0	1	1	3	0	0	0	1	1	3	0	0	1	0	1	5	0	1	0	0	1	9	1	0	0	0	1	17
0	0	1	0	1	5	0	0	1	1	0	6	0	0	1	1	0	6	0	1	0	1	0	10	1	0	0	1	0	18
0	0	1	1	1	7	0	0	1	1	1	7	0	0	1	1	1	7	0	1	0	1	1	11	1	0	0	1	1	19
0	1	0	0	1	9	0	1	0	1	0	10	0	1	1	0	0	12	0	1	1	0	0	12	1	0	1	0	0	20
0	1	0	1	1	11	0	1	0	1	1	11	0	1	1	0	1	13	0	1	1	0	1	13	1	0	1	0	1	21
0	1	1	0	1	13	0	1	1	1	0	14	0	1	1	1	0	14	0	1	1	1	0	14	1	0	1	1	0	22
0	1	1	1	1	15	0	1	1	1	1	15	0	1	1	1	1	15	0	1	1	1	1	14	1	0	1	1	1	23
1	0	0	0	1	17	1	0	0	1	0	18	1	0	1	0	0	20	1	1	0	0	0	24	1	1	0	0	0	24
1	0	0	1	1	19	1	0	0	1	1	19	1	0	1	0	1	21	1	1	0	0	1	25	1	1	0	0	1	25
1	0	1	0	1	21	1	0	1	1	0	22	1	0	1	1	0	22	1	1	0	1	0	26	1	1	0	1	0	26
1	0	1	1	1	23	1	0	1	1	1	23	1	0	1	1	1	23	1	1	0	1	1	27	1	1	0	1	1	27
1	1	0	0	1	25	1	1	0	1	0	26	1	1	1	0	0	28	1	1	1	0	0	28	1	1	1	0	0	28
1	1	0	1	1	27	1	1	0	1	1	27	1	1	1	0	1	29	1	1	1	0	1	29	1	1	1	0	1	29
1	1	1	0	1	29	1	1	1	1	0	30	1	1	1	1	0	30	1	1	1	1	0	30	1	1	1	1	0	30
1	1	1	1	1	31	1	1	1	1	1	31	1	1	1	1	1	31	1	1	1	1	1	31	1	1	1	1	0	31

Услышав ответы, фокусник мысленно составляет набор из пяти двоичных чисел (a_5,a_4,a_3,a_2,a_1) следующим образом: число $a_i=1$, если задуманное число записано на i-й полоске, и $a_i=0$ в противном случае.

После этого он находит число с двоичной записью $(a_5,a_4,a_3,a_2,a_1)_2$, для чего суммирует степени двойки

$$a_1+2a_2+4a_3+8a_4+16a_5$$

(вместо двоичных цифр a_i можно, конечно, запоминать соответствующие степени двойки a_1, $2a_2$,…, $16a_5$ и их складывать в уме, пока зритель говорит, есть число в очередной полоске, или нет).

Разгадка этого поразительного на первый взгляд фокуса очень проста: в i-й полоске записаны все числа, в двоичных записях которых на i-й позиции (считая справа налево) стоит единица. В этом можно наглядно убедится, разглядывая следующую таблицу, в которой выписаны числа из всех полосок рядом с их двоичными записями (хотя можно все понять и не глядя в нее, но проведя несложные рассуждения).Поэтому, какое бы число не загадывал зритель, двоичный набор (a_5,a_4,a_3,a_2,a_1), который возникает в уме у фокусника, является в точности двоичной записью задуманного числа.

Фокус Люка легко преобразовать в способ определения фальшивой монеты среди данных 31. Предполагается, что монеты по виду неразличимы, но фальшивая имеет другой вес (неизвестно, легче она или тяжелее), и вес настоящих монет нам известен. Также предполагается, что у нас есть весы, на которых мы можем точно определять вес положенных на них нескольких монет сразу. За какое наименьшее число взвешиваний можно найти фальшивую монету?

Очевидно, что результат каждого взвешивания дает нам информацию, содержится ли среди взвешенных монет фальшивая ли нет. Если взвешиваний было n,

то различных результатов всей совокупности взвешиваний будет 2^n. Если $2^n < 31$, то по этим результатам найти фальшивую монету, вообще говоря, нельзя, потому что фальшивой может быть любая из 31 монеты, а по результатам взвешиваний она должна определяться однозначно. Поэтому $n > 4$.

Здесь стоит оговориться, что план взвешиваний мы должны составить заранее и он не должен меняться в зависимости от результата очередного взвешивания. В противном случае речь шла бы об адаптивном, или условном, алгоритме взвешивания. Для таких алгоритмов указанное доказательство нижней оценки для числа взвешиваний некорректно.

Для того чтобы найти монету за 5 взвешиваний (неадаптивным алгоритмом!), достаточно в i взвешивании класть на весы все монеты, номера которых написаны на i полоске. Результат взвешивания определяет двоичную цифру a_i точно также, как и в фокусе Люка. А число с двоичной записью $(a_5,a_4,a_3,a_2,a_1)_2$ указывает номер фальшивой монеты. Конечно, на весы можно класть не монеты с номерами из i-й полоски, а монеты с номерами, не записанными на i-й полоске. Алгоритм поиска фальшивой монеты от этого не изменится. Сами взвешивания будут чуть более удобными, так как число взвешиваемых монет уменьшится на единицу в каждом взвешивании.

Разумеется, рассмотренная задача поиска фальшивой монеты имеет и другие интерпретации, некоторые из которых представляют и реальный практический интерес. Например, надо выяснить какая из 31 взятых проб содержит опасный микроб. Можно конечно анализировать все пробы последовательно, но если не повезет, это будет долго и дорого. А можно (сливая содержимое нескольких проб вместе) гарантировано определить это за 5 анализов.

Здесь уместно сказать, что задача поиска фальшивой монеты имеет много различных вариантов, среди которых есть и легкие и трудные, и большинство этих вариантов требует для своего решения соображений, не связанных с рассмотренными выше. Например, есть варианты, в которых разрешается использовать только коромысловые весы, не определяющие вес точно, а лишь сравнивающие веса грузов, положенных на чашки весов. Может быть известно заранее, что фальшивая монета легче или тяжелее, а может быть это и неизвестно. Число фальшивых монет может быть больше единицы, или даже вообще неизвестным, алгоритм взвешиваний может быть условным (адаптивным) и т.д.

Некоторые из этих задач очень легкие. Например, в случае разрешения использовать адаптивные алгоритмы, фальшивую монету можно найти с помощью простого двоичного поиска.

Задача. Как это сделать?

Еще легче следующая задача. Есть 100 мешков с монетами, в одном из которых все монеты фальшивые. Вес настоящей монеты известен точно. Вес фальшивой тоже. За сколько взвешиваний можно найти мешок с фальшивыми монетами?

Ответ: если в каждом мешке не менее 100 монет, то за одно.

Рассмотренная задача о фальшивой монете кажется несколько искусственной, так как для ее решения более естественно использовать именно адаптивные алгоритмы, а тогда она становится довольно простой.

Сейчас мы рассмотрим фактически вариант той же задачи, в котором использование адаптивных алгоритмов невозможно. Допустим, что мы хотим передать по ненадежному каналу связи двоичное число из 31 бита (двоичной цифры). Нам известно, что в среднем на 31 переданный бит приходится одна ошибка. Несколько упрощая ситуацию, будем предполагать, что в полученном слове $x_1,\ldots, x_{31}$ максимум один бит ошибочный. Как его найти? Вспомним про 5 полосок Люка. Для каждой из них вычислим сумму битов, номера которых написаны в этой полоске. На самом деле нам точная сумма будет не нужна, а нужен только остаток от ее деления на два, т.е. фактически информация о том, четная сумма или нечетная. Если есть некто, кто знает, какой из битов неправильный, то он мог бы просто нам сообщить, входит ошибочный бит в рассматриваемую сумму, или нет. После этого мы описанным выше способом нашли бы номер этого бита. Но такого человека нет. Как же быть?

Хемминг придумал такой выход из положения. Он предложил передавать по каналу связи такие 31-битные числа, у которых все указанные выше суммы, соответствующие полоскам Люка, четны (равны нулю по модулю два, как принято говорить в современной алгебре). Тогда в случае, если один из битов данной полоски оказался ошибочным, т.е. вместо нуля он стал единицей или наоборот, то сумма по модулю два всех битов этой полоски будет равна не нулю, а единице (т.е станет нечетной из-за ошибочного бита). Поэтому, просто вычисляя все указанные

суммы, мы получим ту информацию, которую нам мог сообщить всезнающий некто. Значит, мы найдем ошибочный бит указанным выше способом.

Но как же добиться того, чтобы все указанные суммы были равны нулю по модулю два? Ведь если 31 бит выбран произвольно, то этого сделать нельзя. В этом случае мы будем выбирать произвольно не все 31 битов, а только 26, например биты с номерами, отличным от 1,2,4,8,16 (эти биты содержат передаваемую информацию и поэтому называются *информационными*). А биты с номерами 1,2,4,8,16 подберем так, чтобы сумма битов каждой полоски была четна (равна нулю по модулю два). Эти биты назовем *проверочными*. Это можно сделать, так как эти биты записаны в разных полосках (по одной в каждой). Действительно, достаточно для каждой полоски вычислить сумму всех ее битов, кроме проверочного бита (он в полоске один). И выбрать проверочный бит так, чтобы его четность совпадала с четностью указанной суммы (т.е. одновременно они должны быть или четными или нечетными). Тогда сумма всех битов полоски, включая проверочный, будет четна (равна нулю по модулю два). И у такого слова мы можем найти ошибочный бит, или установить, что такого бита нет - это будет тогда, когда все суммы для каждой полоски равны нулю (по модулю два).

Указанная выше схема кодирования, как видим, позволяет передавать 26 битов и исправлять возможную единственную ошибку. Правда, при этом надо передавать еще 5 проверочных битов, не несущих полезной информации. В теории кодирования в этом случае говорят, что пропускная способность (или скорость) кода равна 26/31. Хемминг показал, как подобную схему кодирования применить для передачи 2^n-n-1 информационного бита с помощью n проверочных. Скорость кода Хемминга равна $1-(n+1)/(2^n-1)$ и быстро стремится к нулю с ростом n.

Следует заметить, что указанное решение задачи кодирования с исправлением одной ошибки является корректным и с вероятностной точки зрения, хотя конечно, с некоторой малой вероятностью в полученном сообщении будет несколько ошибок и код их исправить не сможет.

А вот такое очевидное решение задачи кодирования с исправлением одной ошибки не является корректным: повторим каждый передаваемый бит три раза, тогда одну ошибку можно легко исправить, заменив бит, отличающийся от двух равных соседних, на значение этих соседних битов (две ошибки этот код тоже, вообще говоря, не исправляет). Дело даже не в том, что у кода очень мала скорость (равна одной трети), а в том, что при передаче слова утроенной длины в нем в среднем произойдет три ошибки, и код их исправить с высокой вероятностью не сможет.

Простейший пример кода, исправляющего одну ошибку

Клод Шеннон[21] первым выдвинул идею *помехоустойчивого кодирования* (кодирования с исправлением ошибок) и привел в своей статье, заложившей основы теории передачи информации, следующий пример кода, исправляющего одну ошибку (частный случай кода Хемминга).

Пусть нужно передать двоичное слово (x_1,x_2,x_3,x_4) Добавим к нему проверочные символы $x_5 = x_1 \oplus x_3 \oplus x_4, x_6 = x_1 \oplus x_2 \oplus x_4, x_7 = x_1 \oplus x_2 \oplus x_3$ (знак $\oplus$ здесь обозначает сложение по модулю два; символы x_1,x_2,x_3,x_4 являются *информационными*).

Процедура вычисления по информационным символам проверочных и составления из них Код, в котором вначале идут информационные символы, а в конце проверочные символы, называется систематическим. Рассматриваемый пример является таким кодом.

Передаем закодированное сообщение $c =(x_1,\dots x_7)$ и получаем зашумленное сообщение $r = c \oplus e$, где $e = (e_1,\dots,e_7)$ - вектор ошибок. В нашем примере он имеет вес 1, т.е. в нем только одна единица, так как по предположению ошибка может произойти (если произойдет) только в одной позиции.

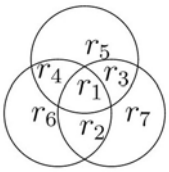
Например, возможно $e = e_3 = (0,0,1,0,0,0,0)$. Тогда

$r = c \oplus e = (c_1,c_2,c_3 \oplus 1,c_4,c_5,c_6,c_7) = (c_1,c_2,c_3,c_4,c_5,c_6,c_7) \oplus (0,0,1,0,0,0,0)$.

.Число 3 будет в рассматриваемом случае *позицией ошибки*. Для определения позиции ошибки (а значит, и обнаружения самой ошибки) можно вычислить *проверочные суммы* $r_5 \oplus r_1 \oplus r_3 \oplus r_4$, $r_6 \oplus r_1 \oplus r_2 \oplus r_4$, $r_7 \oplus r_1 \oplus r_2 \oplus r_3$

[21] Клод Эльвуд Шеннон (1916-2001)– выдающийся американский математик и инженер, один из основателей теории информации, теории кодирования, теории сложности булевых функций и теоретической криптологии. Во время войны занимался расшифровкой японских радиограмм.

Наглядно все эти суммы изображены на следующем рисунке.



Каждая сумма содержится в своем круге. К сожалению, подобная иллюстрация для кодов Хемминга больших размерностей невозможна.

Литература

1. Гашков С.Б. Системы счисления и их применение. МЦНМО 2004 (готовится новое издание в 2012 г)
2. Гашков С.Б. Занимательная компьютерная арифметика. Книга готовится к изданию в изд. URSS, выход ожидается в 2012 г.
3. Ч. Петцольд. Код. Русская редакция Микрософт пресс. 2001.
4. http://www.lomonosov-fund.ru/enc/ru/encyclopedia:0135751
5. http://www.lomonosov-fund.ru/enc/ru/encyclopedia:0135750
6. http://www.lomonosov-fund.ru/enc/ru/encyclopedia:0135754