КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
Институт вычислительной математики и информационных технологий
Кафедра системного анализа и информационных технологий

Ш.Т. Ишмухаметов, Р.Х. Латыпов,
Р.Г. Рубцова, Е.Л. Столов

Введение в теорию чисел и теорию кодирования

Учебно-методическое пособие
для студентов Института вычислительной
математики и информационных Технологий

Казань 2014

УДК 511, 519.6

Печатается по решению редакционно-издательского совета ФГАОУВПО
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ
УНИВЕРСИТЕТ»,
методической комиссии факультета вычислительной
математики и кибернетики, протокол № 6 от 11 декабря 2014 г.

Авторы-составители —
докт. физ. мат. наук, проф. каф. САИТ КФУ Ш.Т. Ишмухаметов
докт. техн наук, проф. каф. САИТ КФУ Р.Х. Латыпов
докт. техн наук, проф. каф. САИТ КФУ Е.Л.Столов
ст.препод. каф. САИТ Р.Г.Рубцова

Введение в теорию чисел и теорию кодирования: учебное пособие / Ш.Т. Ишмухаметов, Р.Х. Латыпов, Е.Л.Столов,Р.Г.Рубцова.– Казань: Казан. ун. 2014.– 65 с.

Предназначено для студентов старших курсов факультета вычислительной математики и кибернетики.

Содержание

1 Вспомогательные сведения из теории алгоритмов

Теория сложности вычислений является разделом теории вычислений, изучающим сложность(стоимость) работы, требуемой для решения вычислительной проблемы.

Сложность обычно измеряется абстрактными понятиями времени и пространства, называемыми вычислительными ресурсами.

Время определяется количеством тривиальных шагов, необходимых для решения проблемы, тогда как пространство определяется объёмом памяти или места на носителе данных или количеством процессоров для обработки данных.

1.1 Основные понятия теории сложности

Одним из наиболее важных понятий математики является понятие алгоритма.

Определение. Алгоритм – это конечный набор инструкций для решения класса однотипных задач.

Отметим наиболее важные характеристики алгоритма:

- *Конечность.* Алгоритм состоит из конечного числа инструкций.
- *Наглядность.* Каждая инструкция представляет собой простое действие.
- *Массовость.* Алгоритм применим не к одной задаче, а к классу задач, давая для каждой задачи из области определения свое решение.
- *Детерминированность.* Действие алгоритма однозначно по отношению к одним и тем же входным данным независимо от условий выполнения алгоритма, текущего времени или других условий.

Впрочем, есть класс недетерминированных алгоритмов, в которых последнее условие не выполняется. В таких алгоритмах выполнение вычисления можно выполняться одновременно по нескольким альтернативным путям, приводящим к разным ответам.

1.2 Задачи теории сложности

Основными задачами теории сложности алгоритмов являются следующие:

- Обеспечить методику количественной оценки сложности проблемы в абсолютных терминах.
- Дать метод сравнения сложности двух различных проблем.
- Дать строгое определение эффективного алгоритма.

Размер проблемы

Под размером проблемы понимается длина входных данных в том или ином представлении. Обычно длина входных данных измеряется в битах. Например, длина ключа RSA равна 1024 или 2048 битам. Представление данных в разных системах может иметь разную длину, однако разные представления отличаются обычно в константное число раз. Например, представление ключей RSA в десятичном представлении имеет длину, меньшую битового, в $log_2 10$ раз.

Замечание. Некоторые представления задачи могут иметь существенно меньшую длину. В алгоритмической теории информации определяется колмогоровская сложность объекта как мера вычислительных ресурсов, необходимых для точного определения этого объекта. Некоторые ресурсы по этой мере могут иметь существенно меньшую длину представления. Следует отметить, однако, представляя данные в сокращенной форме, можно получить дополнительные издержки при выполнении алгоритма.

Оценка скорости работы алгоритма.

Существуют *временная* и *пространственная* сложность алгоритма.

Временная сложность оценивает время или количество тактов работы алгоритма относительно длины входных данных. Например, время работы алгоритма сортировки n-мерного числового массива методом пузырька оценивается сверху величиной $C \cdot n^2$ для некоторой константы $C > 0$. Длина входных данных здесь равна $q\,n$, где q – длина представления элемента массива.

Поскольку заданную длину могут иметь много входных задач, то временная оценка строится для всех задач из класса задач с входными

данными одинаковой длины. Поэтому могут существовать наилучшая, средняя и наихудшая оценка времени работы алгоритма.

- *Наилучший случай.* Оценивает время работы самой простой задачи из класса задач заданной длины. Дает мало информации о параметрах сложности.
- *Средний случай.* Оценивает вероятностное распределение параметров сложности.
- *Наихудший случай.* Облегчает анализ проблемы.

Пространственная сложность оценивает количество памяти, требуемой для выполнения вычисления. Эти две меры тесно связаны. Обычно, алгоритмы, требующие много времени вычисления, потребляют много памяти, но бывают и исключения. Некоторые алгоритмы - требовательны к памяти, и при нехватке памяти работают значительно медленнее.

1.3 Временная сложность (асимптотика)

Для сравнения алгоритмов используются специальные обозначения.

Обозначение	Интуитивное объяснение	Определение
$f(n) \in O(g(n))$	f ограничена сверху функцией g с точностью до постоянного множителя асимптотически	$(\exists C)(\exists n_0)(\forall n \geq n_0)$ $f(n) \leq C \cdot g(n)$
$f(n) \in \Omega(g(n))$	f ограничена снизу функцией g с точностью до постоянного множителя асимптотически	$(\exists C)(\exists n_0)(\forall n \geq n_0)$ $f(n) \geq C \cdot g(n)$
$f(n) \in \Theta(g(n))$	f ограничена сверху и снизу функцией g с точностью до постоянного множителя асимптотически	$(\exists C_1)(\exists C_2)(\exists n_0)(\forall n \geq n_0)$ $C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$
$f(n) \in o(g(n))$	g доминирует над f асимптотически	$\lim_{n\to\infty} f(n)/g(n) = 0$
$f(n) \in \omega(g(n))$	f доминирует над g асимптотически	$\lim_{n\to\infty} f(n)/g(n) = \infty$
$f(n) \sim (g(n))$	f эквивалентна g асимптотически	$\lim_{n\to\infty} f(n)/g(n) = 1$

Критерии оценки сложности (1)

- Если создаваемая программа будет использована только несколько раз, тогда стоимость написания и отладки программы будет доминировать

в общей стоимости программы. В этом случае следует предпочесть алгоритм, являющийся наиболее простым для реализации.

- Если программа будет работать только с "малыми" входными данными, то степень роста времени выполнения будет иметь меньшее значение, чем константа, присутствующая в формуле времени выполнения. Вместе с тем и понятие «малости» входных данных зависит от точного времени выполнения конкурирующих алгоритмов. Существуют алгоритмы, такие как алгоритм целочисленного умножения, асимптотически самые эффективные, но которые никогда не используют на практике даже для больших задач, так как их константы пропорциональности значительно превосходят подобные константы других, более простых и менее "эффективных" алгоритмов.

- Эффективные, но сложные алгоритмы могут быть нежелательными, если готовые программы будут поддерживать лица, не участвующие в написании этих программ.

- Известно несколько примеров, когда эффективные алгоритмы требуют таких больших объемов машинной памяти (без возможности использования более медленных внешних средств хранения), что этот фактор сводит на нет преимущество «эффективности» алгоритма.

- В численных алгоритмах точность и устойчивость алгоритмов не менее важны, чем их временная эффективность.

1.4 Некоторые важные классы алгоритмов

Определение. Функция $f(n) = n^k$, где $k > 0$, называется полиномиальной. Функция $f(n) = a^n$, где $a > 1$, называется экспоненциальной функцией или простой экспонентой.

Полиномиальные алгоритмы.

Определение. Алгоритм, временная сложность которого ограничена полиномиальной функцией, называется *полиномиальным алгоритмом.*

Пример: Нахождение кратчайшего пути в графе с неотрицательным весом. *Сложность:* $O(n^2)$.

Другие примеры: сортировка, поиск во множестве, выяснение связности графов.

Экспоненциальные алгоритмы

Определение. Алгоритм, временная сложность которого ограничена экспоненциальной функцией, называется *экспоненциальным алгоритмом.*

Пример: Выбор n-разрядного числа перебором. Сложность: $O(10^n)$.

Другие примеры: задача коммивояжера, задача выполнимости булевых формул.

Субэкспоненциальные алгоритмы.

Определение. Алгоритм, временная сложность которого меньше экспоненциальной, но не является полиномиальной.

Пример: Разложение n-разрядного числа на простые множители (факторизация n) методом решета числового поля

$$\text{Сложность: } O(e^{C(\ln\ln n)^{1/3}(\ln n)^{2/3}})$$

Другие примеры: Вычисление дискретного логарифма в конечном поле.

1.5 Модели вычислений

Для формального описания алгоритма в 30-е годы XX столетия было предложено несколько моделей вычислений. Самая известная из них – машина Тьюринга.

Машина Тьюринга.

Машина Тьюринга – абстрактная вычислительная машина, предложенная английским математиком Аланом Тьюрингом в 1936 году для формализации понятия алгоритма.

В состав машины Тьюринга (МТ) входит бесконечная в одну сторону лента (в некоторых описаниях - бесконечная в обе стороны лента), разделённая на ячейки, и читающее - записывающее устройство (ЧЗУ). Каждая ячейка

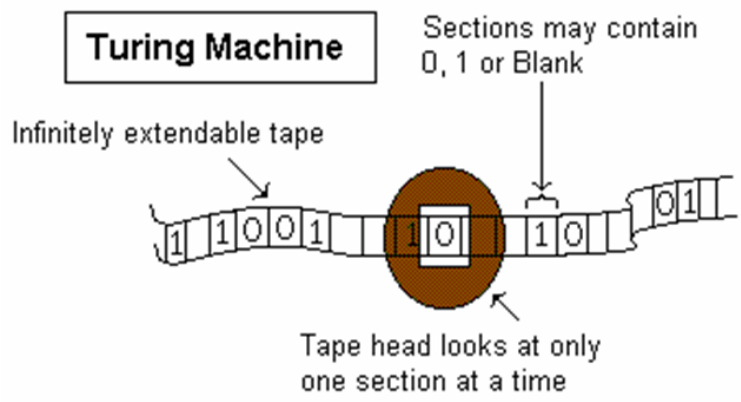


Рис. 1: Машина Тьюринга

на ленте может содержать либо пустой символ λ, либо один из символов конечного алфавита $A = \{a_0, a_1, \ \ldots, a_n\}$ (обычно алфавит A состоит из 0 и 1).

МТ работает по тактам, начиная с такта 0. Каждый такт МТ находится в одном из внутренних состояний множества $Q = \{q_0, q_1, \ \ldots, q_k\}$, причем состояние q_1 является начальным, а состояние q_0 - конечным (финальным).

ЧЗУ каждый момент времени обозревает одну ячейку ленты, может перемещаться на одну клетку влево и вправо или остаться на месте, читать и записывать в ячейки ленты символы алфавита A.

В начальный момент работы МТ на ленте записано входное слово символами алфавита A, начиная с клетки ленты с номером 0, ЧЗУ читает содержимое клетки 0, и МТ находится в состоянии q_1.

Пустой символ заполняет все клетки ленты, кроме конечного числа, на которых записаны входные данные. Управляющее устройство работает согласно правилам перехода, которые представляют алгоритм, реализуемый данной машиной Тьюринга. Каждое правило перехода предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей клетке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево, вправо или остаться на месте.

Вид команды машины Тьюринга: $< \mathbf{q_i}, \mathbf{a_i} \ \rightarrow \mathbf{q_j}, \ \mathbf{a_j}, \ \mathbf{s_j}, >, \quad s_j \in \{L, R, S\}.$

Эта команда выполнится в момент времени t, если в этот момент

машина находится во внутреннем состоянии q_i, и читающее устройство обозревает ячейку, в которой записан символ a_i.

При выполнении команды, машина перейдет в новое состояние q_j, заменит символ a_i на a_j и передвинет читающее устройство на одну клетку влево, вправо или оставит его в той же позиции в зависимости от значения s_j.

Машина Тьюринга работает до тех пор, пока не перейдет в финальное состояние q_0. Выходным значением (результатом работы) машины станет число y, равное общему количеству единиц на ленте в момент остановки машины T.

Машина Тьюринга называется детерминированной, если каждой комбинации состояния и ленточного символа в таблице соответствует не более одного правила. Если существует несколько команд с одинаковой левой частью – входной парой «состояние – ленточный символ», то такая машина Тьюринга называется недетерминированной. Если в какой-то момент времени появляется две или более команды с одинаковой левой частью, соответствующей текущей конфигурации машины Тьюринга, вычисление разделяется на несколько ветвей, соответствующих разным командам с одинаковой левой частью. Каждая ветвь продолжает свое выполнение в параллельном режиме. Результатом вычисления недетерминированной машины Тьюринга полагается результат, полученный на той ветви, которая закончит свое вычисление раньше остальных.

Поскольку, число параллельный ветвей никак не ограничено, то, очевидно, что никакой реальный компьютер не может служить реализацией недетерминированной МТ. Однако, можно моделировать вычисление, выделяя каждому процессу некоторые ресурсы последовательно. В этом случае, работы машины Тьюринга будет выполнена, хотя вычисление продлится несколько дольше.

Отметим, что на некоторых входных аргументах машины Тьюринга могут выдавать бесконечное вычисление, не переходя никогда в финальное состояние. В таких случаях говорят, что машина Тьюринга *зацикливается* на аргументе x. Функция, вычислимая на таких машинах, является частично определенной. Возможность реализации частичных функций является важной характеристикой машин Тьюринга.

Класс функций, вычислимых на м.Тьюринга. Рекурсивные функции.

Хотя машины Тьюринга представляют собой весьма примитивные вычислительные модели, в ходе обширных исследований было выяснено, что *любой* известный человечеству алгоритм можно реализовать на какой-нибудь машине Тьюринга, т.е. машина Тьюринга является универсальным вычислительным устройством, позволяющим реализовывать сколь угодно сложные алгоритмы.

Немецкий математик Курт Гедель в 1931 году определил и исследовал другую модель алгоритмов – алгоритмов, реализующих *рекурсивные функции.* Эти функции строятся как комбинации трех базовых операторов суперпозиции, минимизации и примитивной рекурсии, примененных к простых базовым функциям (функции, тождественно равной 0, функции прибавления 1 к аргументу, и набора функций выборки, выбирающих из n-и входных аргументов какой-нибудь один). Алгоритм вычисления каждой такой функции заложен в самом описании этой функции, поэтому каждая рекурсивная функция является *алгоритмически вычислимой.*

Дадим точное определение рекурсивной функции:

Базовыми рекурсивными функциями являются:
1. 0-местная функция (константа) O – ноль,
2.Функция следования $s(n) = n + 1$,
3.Функции выборки I_n^k : $I_n^k(x_1, x_2, \ldots, x_n) = x_k$, $0 < k \leq n$.

Оператором суперпозиции называется оператор подстановки одних функции в качестве аргументов других: пусть даны функции $f(x_1, x_2, \ldots, x_n)$, $g_1(y_1, \ldots, y_k)$, $\ldots$, $g_n(y_1, \ldots, y_k)$. Результатом суперпозиции будет функция $z = f(g_1, g_2, \ldots, g_n)[y_1, \ldots, y_k]$

Оператором примитивной рекурсии называется оператор, позволяющий из n-местной функции g и $(n + 2)$-местной функции h строить $n + 1$-местную функцию f по следующим правилам:

$$f(x_1, \ldots x_n, 0) = g(x_1, \ldots x_n)$$
$$f(x_1, \ldots x_n, k + 1) = h(x_1, \ldots x_n, k, f(x_1, \ldots x_n, k))$$

(для вычисления $f(x_1, \ldots x_n, k + 1)$ в h было подставлено значение f в предыдущей точке k).

Определение. Любая функция, которую можно получить из базовых

функций применением конечного числа операторов суперпозиции и примитивной рекурсии, называется *примитивно-рекурсивной.*

Оператором минимизации называется оператор, строящий по $(n+1)$-местной функции g n-местную функцию f по следующему правилу:

$$(\forall x_1, \ldots, x_n)\ f(x_1, \ldots, x_n) = \min t\ [g(x_1, \ldots, x_n, t) = 0].$$

Примеры примитивно-рекурсивных функций:

1. $f(x, y) = x + y$
2. $f(x, y) =$ Н.О.Д. (x, y),
3. $f(x, y, z) = x^{y^z}$.

Определение. Любая функция, которую можно получить из базовых функций применением конечного числа операторов суперпозиции примитивной рекурсии и минимизации, называется *рекурсивной.*

Таким образом, все примитивно-рекурсивные функции являются рекурсивными. Обратное не верно. Важное отличие рекурсивных функций от примитивно-рекурсивных состоит в том, что первые являются всюду определенными в то время, как вторые могут быть и не всюду определенными.

1.6 Тезис Черча

Дальнейшие исследования показали, что класс функций, вычислимых на машинах Тьюринга, и класс рекурсивных функций совпадают.Более того, и другие подобные модели (нормальные алгорифмы Маркова, машины Поста, RAM-машины и др.) давали тот же класс функции, который покрывал все известные арифметические функции. Таким образом, можно было предположить, что все известные человечеству алгоритмы можно реализовать в рамках класса рекурсивных функций. Данное предположение (гипотеза, т.к. точно доказать ее невозможно) получила название *тезиса Черча* по имени английского математика Аллоиза Черча.

Тезис Черча: Любая *алгоритмически вычислимая* функция вычислима на какой-нибудь машине Тьюринга.

Формулировка тезиса Черча явилось одним из величайших достижений теории алгоритма, поскольку позволило дать формализацию (точное математическое описание) понятию *алгоритмически разрешимой* функции, как

рекурсивной функции. Иначе говоря, если математическая задача не могла быть разрешена с помощью машины Тьюринга, то она не имела никакого решения вообще, т.е. являлась *алгоритмически неразрешимой.*

Конечные автоматы

Полезным инструментом для реализации математических алгоритмов явились *конечные автоматы (КА)* Также как и машина Тьюринга КА имеет конечное множество внутренних состояния и бесконечную ленту.

Главное отличие конечного автомата от машины Тьюринга – это линейность работы. Автомат работает по тактам, считывая конечное слово на входной ленте по одному символу за такт работы. Читающее устройство движется только в одном направлении, и работа закончится, когда будет пройдено входное слово. Поэтому время работы автомата всегда равно длине входного слова.

Каждый момент времени автомат находится в одном из конечного множества внутренних состояний. Множество внутренних состояний автомата Q делится на два непересекающихся подмножества Q_1 и Q_2. Если при завершении работы состояние автомата принадлежит Q_1, то говорят, что входное слово принимается данным автоматом. Иначе, входное слово *отвергается* автоматом.

Таким образом, конечный автомат выполняется разбиение множества входных слов на два непересекающихся подмножества: подмножество слов, принимаемых автоматом, и подмножество слов, отвергаемых автоматом.

1.7 Решающие проблемы и классы сложности

Решением алгоритмической проблемы (задачи) является, как правило, числовой ответ. Однако, есть класс проблем, решением которых является лишь константное значение **ДА** или **НЕТ**.

Пример 1. *Совместность системы уравнений*: для заданной системы уравнений относительно n неизвестных, определить, существует ли решение?

$$\begin{cases} x_1 + x_2^2 = 0 \\ x_1^3 - x_2 = 0 \\ x_1 + x_2 = 0 \end{cases}$$

Пример 2. *Проверка простоты натурального числа.* Для заданного натурального числа определить, является ли оно простым или составным.

Сводимости.

Сводимость – это процедура, сводящая решение задачи из одного класса к решению из другого класса. Сводимость ранжирует классы алгоритмов по их сложности: тот класс, к которому сводятся другие классы, является более трудновычислимым. Дадим формальное определение сводимости.

Определение. Проблема P_1 сводится к проблеме P_2 если существует эффективная функция (алгоритм), позволяющая для каждого экземпляра Z_1 проблемы P строить экземпляр Z_2 проблемы P_2, так что задача Z_1 имеет решение "Да" тогда и только тогда, когда такое же решение имеет задача Z_2.

Обозначение: $P_1 \leq P_2$.

Полиномиальная сводимость

Проблема P_1 полиномиально сводится к проблеме P_2 если проблема P_1 сводится к проблеме P_2, и сводящая функция (алгоритм) полиномиально вычислима.

Обозначение: $P_1 \leq_p P_2$.

Пример. "Задача коммивояжера" полиномиально сводится к задаче "Выполнимость булевой функции".

Отметим, что если выполнено $P_1 \leq_p P_2$, то из существования полиномиального по времени алгоритма для P_2 следует существование полиномиального временного алгоритма для P_1.

Классы сложности

Определение. Класс P содержит все решающие проблемы, для которых существует алгоритм для машины Тьюринга, который приводит к правильному «да/нет» ответу за число шагов, ограниченному полиномом от размера задачи.

Определение. Класс NP содержит все решающие проблемы, для которых, имея некоторый ответ, существует полиномиальная проверка (сертификат) С того, что данный ответ дает правильное «да/нет» решение проблемы.

Определение. Класс $co - P$ содержит все решающие проблемы, для которых существует полиномиальный алгоритм, который проверяет, какие из «да/нет» ответов неверны.

Определение. Класс $co - NP$ содержит все решающие проблемы, так что существует полиномиальная проверка (сертификат) C , которая определяет, верно ли, что проблема не имеет правильных «да/нет» ответов.

Важнейшие гипотезы

Хотя определения классов алгоритмов P, $co-P$, NP и $co-NP$ интуитивно понятны, важнейшие утверждения относительно непустоты классов разностей $co - NP - P$, $NP - P$ и $NP - co - P$ не доказаны.

Впервые вопрос о равенстве классов P и NP был поставлен Стивеном Куком (S.A.Cook) в 1971 году. Этот вопрос является одной из центральных открытых проблем уже более 40 лет. В теории алгоритмов вопрос о равенстве классов сложности P и NP является одной из центральных открытых проблем. Положительный ответ будет означать, что существует (по-крайней мере, теоретическая возможность) решать многие сложные задачи существенно быстрее, чем сейчас. Однако, вероятность такого исхода практически равна нулю. Для отрицательного решения необходимо построить пример алгоритмической задачи, решаемой на какой-то недерминированной м.Тьюринга, которую невозможно решить ни на какой детерминированной м.Тьюринга.

Проблема $\mathbf{P} = \mathbf{NP}$ является одной из семи задач тысячелетия, за решение которой Математический институт Клэя назначил премию в миллион долларов США. Основными гипотезами, которых придерживается большинство математиков, являются следующие:

- $P = co - P$
- $NP \neq co - NP$
- $P \neq NP$

NP-полные проблемы

Определение. (NP-полнота) *NP-полная решающая проблема* Z – это такая задача из класса NP, к которой можно свести любую другую решающую проблему Z' из класса NP, то есть, $Z' \leq_p Z$.

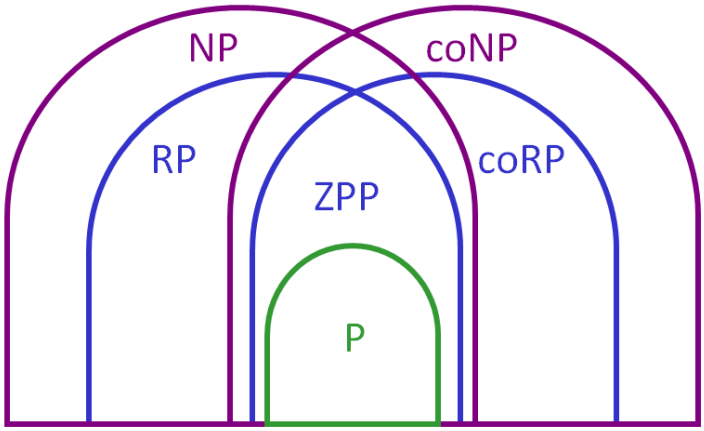


Рис. 2: Соотношения классов сложности

Таким образом, NP-полные задачи образуют в некотором смысле подмножество «самых сложных» задач в классе NP; и если для какой-то из них будет найден «быстрый» алгоритм решения, то и любая другая задача из класса NP может быть решена так же «быстро». В настоящее время доказано, что NP-полными являются многими известные математические задачи:

1. Задача SAT проверки выполнимости произвольно булевой формулы (достаточно ограничиться формулами от трех переменных).
2. Задача коммивояжера.
3. Проблема раскраски графа.
4. Задача о вершинном покрытии.

Как избежать NP-полноты?

- Попытаться доказать $P = NP$ (приз - 1 млн.долларов).
- Использовать рандомизированные алгоритмы.
- Использовать приближенные алгоритмы.
- Использовать эвристические алгоритмы.

- Использовать квантовые алгоритмы.

2 Основы теории кодирования

КОДИРОВАНИЕ И ДЕКОДИРОВАНИЕ - процессы представления информации в определенной стандартной форме и обратный процесс восстановления информации по ее такому представлению. В математической литературе кодированием называется отображение произвольного множества в множество конечных последовательностей (слов) в некотором алфавите , а декодированием - обратное отображение.

Примерами кодирования являются: представление натуральных чисел в r-й системе счисления, при котором каждому числу $N = 1, 2, ...l$ ставится в соответствие слово $b_1,\ b_2, ...b_l$ в алфавите , $0 \leq r \leq r-1$, такое, что $b_1 \neq 0$ и $b1 \cdot r^{l-1} + ... + b_{l-1} \cdot r + b_l = N$.

Примерами такого кодирования могут служить преобразования текстов на русском языке с помощью телеграфного кода в последовательности, составленные из посылок тока и пауз различной длительности; отображение, применяемое при написании цифр почтового индекса. В последнем случае каждой десятичной цифре соответствует слово в алфавите В $B_2 = \{0, 1\}$ длины 9, в котором символами 1 отмечены номера использованных линий (например, цифре 5 соответствует слово 110010011). Исследование различных свойств кодирований/декодирований и построение эффективных в определенном смысле кодирований, обладающих требуемыми свойствами, составляет проблематику теории кодирования. Обычно критерий эффективности кодирования так или иначе связан с минимизацией длин кодовых слов (образов элементов множества А), а требуемые свойства кодирования связаны с обеспечением заданного уровня помехоустойчивости, понимаемой в том или ином смысле. В частности, под помехоустойчивостью понимается возможность однозначного декодирования при отсутствии или допустимом уровне искажений в кодовых словах. Помимо помехоустойчивости, к кодированию может предъявляться ряд дополнительных требований. Например, при выборе кодирования для цифр почтового индекса необходимо согласование с обычным способом написания цифр. В качестве дополнительных требований часто используются ограничения, связанные с допустимой сложностью схем, осуществляющих К. и д. Проблематика теории кодирования в основном создавалась под влиянием разработанной К. Шенноном (C. Shannon, [87]) теории передачи информации.

Основной литературой по данному разделу являются книги [45], [65], [69], [73] и [87].

2.1 Кодирование информации и шумы. Глоссарий.

Канал - среда передачи информации, например, телефонная линия или атмосфера

Кодирование источника – устранение «лишней», сжатие информации.

Кодирование канала – добавление избыточности для обнаружения и/или исправления ошибок (в результате шума) – защита от случайных воздействий.

Шум – фактор, влияющий на передачу информации. Шум может возникнуть из-за магнитной бури, молнии, метеоритного дождя, случайного искажения звука в радиопередаче, плохой печати изображения или текста, плохой слышимости.

В результате шума сообщение может исказиться.

2.2 Структура кодера и декодера

Преобразование дискретного сообщения в сигнал обычно осуществляется в виде двух операций - кодирования и модуляции. Кодирование представляет собой преобразование сообщения в последовательность кодовых символов.

Простейшим примером дискретного сообщения является текст. Любой текст состоит из конечного числа элементов: букв, цифр, знаков препинания. Их совокупность называется алфавитом источника сообщения. Так как число элементов в алфавите конечно, то их можно пронумеровать и тем самым свести передачу сообщения к передаче последовательности чисел.

Так, для передачи букв русского алфавита (их 32) необходимо передать числа от 1 до 32. Для передачи любого числа, записанного в десятичной форме, требуется передача одной из десяти цифр от 0 до 9 для каждого десятичного разряда. На практике при кодировании дискретных сообщений широко применяется двоичная система счисления.

При кодировании происходит процесс преобразования элементов сообщения в соответствующие им числа (кодовые символы). Каждому элементу сообщения присваивается определенная совокупность кодовых символов,

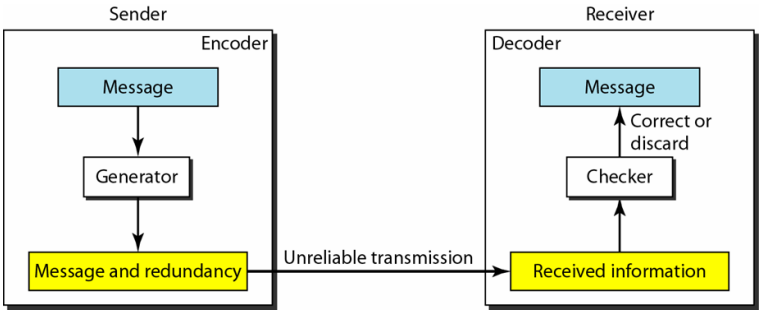


Рис. 3: Структура кодера и декодера

которая называется кодовой комбинацией. Совокупность кодовых комбинаций, обозначающих дискретные сообщения, образует код.

Правило кодирования может быть выражено кодовой таблицей, в которой приводятся алфавит кодируемых сообщений и соответствующие им кодовые комбинации. Множество возможных кодовых символов называется кодовым алфавитом, а их количество m - основанием кода.

В общем случае при основании кода m правила кодирования N элементов сообщения сводятся к правилам записи N различных чисел в m-й системе счисления. Число разрядов n, образующих кодовую комбинацию, называется значностью кода, или длиной кодовой комбинации. В зависимости от системы счисления, используемой при кодировании, различают двоичные и m-е (недвоичные) коды.

Коды, у которых все комбинации имеют одинаковую длину, называют равномерными. Для равномерного кода число возможных комбинаций равно $m \cdot n$. Примером такого кода является пятизначный код Бодо, содержащий пять двоичных элементов ($m = 2$, $n = 5$). Число возможных кодовых комбинаций равно $2^5 = 32$, что достаточно для кодирования всех букв алфавита. Применение равномерных кодов не требует передачи разделительных символов между кодовыми комбинациями.

Неравномерные коды характерны тем, что у них кодовые комбинации отличаются друг от друга не только взаимным расположением символов, но и их количеством. Это приводит к тому, что различные комбинации имеют различную длительность. Типичным примером неравномерных кодов является код Морзе, в котором символы 0 и 1 используются только в двух сочетаниях - как одиночные (1 и 0) или как тройные (111 и

000). Сигнал, соответствующий одной единице, называется точкой, трем единицам - тире. Символ 0 используется как знак, отделяющий точку от тире, точку от точки и тире от тире. Совокупность 000 используется как разделительный знак между кодовыми комбинациями.

По помехоустойчивости коды делят на простые (примитивные) и корректирующие. Коды, у которых все возможные кодовые комбинации используются для передачи информации, называются простыми, или кодами без избыточности. В простых равномерных кодах превращение одного символа комбинации в другой, например 1 в 0 или 0 в 1, приводит к появлению новой комбинации, т. е. к ошибке.

2.3 Корректирующие коды

Одним из наиболее значимых методов борьбы с ошибками является *избыточное кодирование.* Целью избыточного кодирования является обнаружение и (или) исправление ошибок путем добавления дополнительной (избыточной) информации в код. С этой целью корректирующие коды строятся так, что для передачи сообщения используются не все кодовые комбинации m^n, а лишь некоторая часть их (так называемые разрешенные кодовые комбинации).

Декодирование состоит в восстановлении сообщения по принимаемым кодовым символам. Устройства, осуществляющие кодирование и декодирование, называют соответственно кодером и декодером. Как правило, кодер и декодер выполняются физически в одном устройстве, называемым кодеком.

Методы борьбы со случайными ошибками.

При определении помехоустойчивости кодирования формализуется понятие ошибки и вводится в рассмотрение некоторая модель образования ошибок.

Ошибкой типа замещения (или просто ошибкой) называется преобразование слова, состоящее в замещении одного из его символов другим символом алфавита В рис.2 произошло замещение символа 0 символом 1. Возможность обнаружения и исправления ошибок основана на том, что для кодирования f, обладающего ненулевой избыточностью, декодирование f^{-1} может быть произвольным образом доопределено на r подсловах, не являющихся кодовыми. В частности, если множество разбито на m непересекающихся подмножеств $D_0, \ldots, D_{m-1}$ таких, что декодирование f^{-1} доопределено так, что $f^{-1}(D_i) = i$, то при декодировании будут исправлены все ошибки,

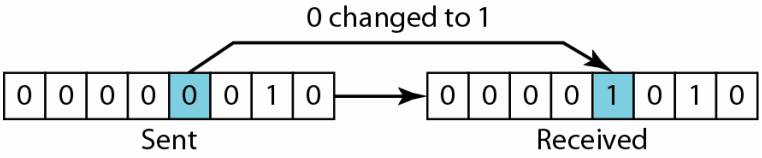


Рис. 4: Ошибка в одном разряде

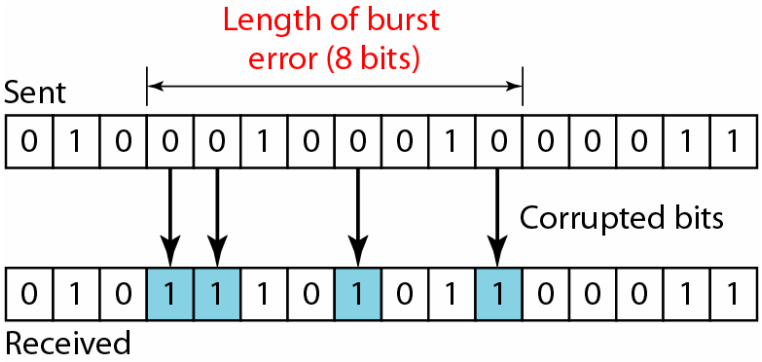


Рис. 5: Пакет ошибок длины 8

преобразующие кодовое слово $f(i)$ в D_i, $i = 0, ..., m-1$. Аналогичная возможность имеется и в случае ошибок других типов таких, как стирание символа (замещение символом другого алфавита), изменение числового значения кодового слова на $b \in \{1, \ldots, r-1\}$ (арифметическая ошибка), выпадение или вставка символа и т. п.

Другие модели ошибок

Стирающий канал. Стирающие коды являются самыми «молодыми» среди всех известных ныне помехоустойчивых кодов, и их изученность на сегодня не является полной.

Согласно определению, *стирание* – это ошибка, позиция которой известна, а значение не определено.

Примером таких ошибок могут служить потери пакетов, имеющие место в сетях IP. Поскольку в сетях IP используется сквозная нумерация

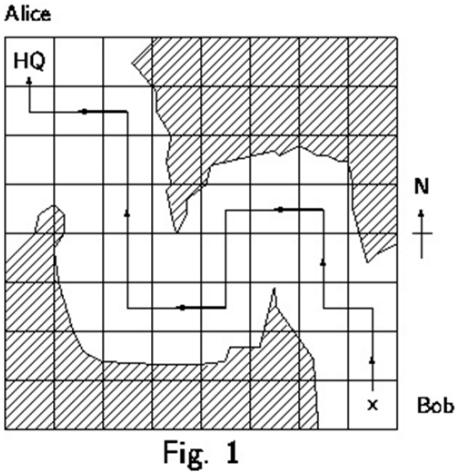


Рис. 6: Путь от Боба до Алисы

пакетов, то существует возможность определить позицию потерянного пакета. Значение стертого пакета неизвестно получателю (декодеру).

Канал со вставками. Метод использует дополнительные биты для вставки контрольной информации.

2.4 Пример избыточного кодирования.

Предположим, что требуется закодировать путь от Боб до Алисы (рис.4).

Обозначим один шаг в направлении "Север", "Запад", "Юг", "Восток" кодами "N", "W", "S", "E" соответственно. Тогда весь путь изобразится последовательностью

$$S_{A,\,B} = \{N,\ N,\ W,\ N,\ N,\ W,\ W,\ S,\ S,\ W,\ W,\ N,\ N,\ N\ \}$$

1. **Неизбыточное кодирование.** Будем кодировать направления движения двухбитовыми последовательностями:

$$\{00, 01, 10, 11\}.$$

Очевидно, что любая ошибка в одном бите приводит к потере информации.

2. **Избыточное кодирование.** Будем кодировать направления движения трехбитовыми последовательностями:

$$\{000, 011, 101, 110\}.$$

Теперь изменение одного бита позволяет определить *наличие ошибки.*

3. **Избыточное кодирование 2.** Будем кодировать направления движения пятибитовыми последовательностями:

$$\{00000, 01101, 10110, 11011\}.$$

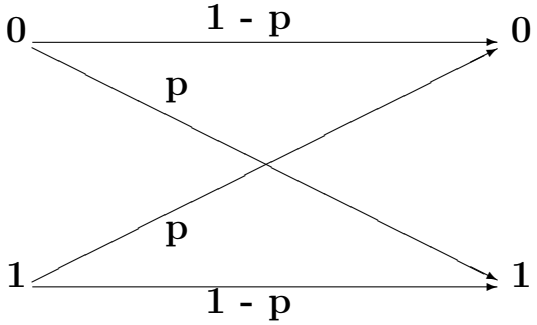
Теперь ошибка в одном бите не только будет обнаружена, но и ее можно будет исправить.

Цели передачи по каналу с шумом

1. Быстрое кодирование информации.
2. Простой способ передачи закодированного сообщения.
3. Быстрое декодирование полученной информации.
4. Надежная очистка от шума.
5. Передача максимального объема информации в единицу времени.

ДСК – двоичный симметричный канал.

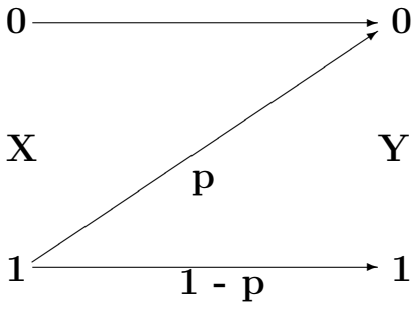
Двоичность: Алфавит ={0,1}
Симметричность: $p(0 \to 1) = p(1 \to 0)$



Другие модели каналов

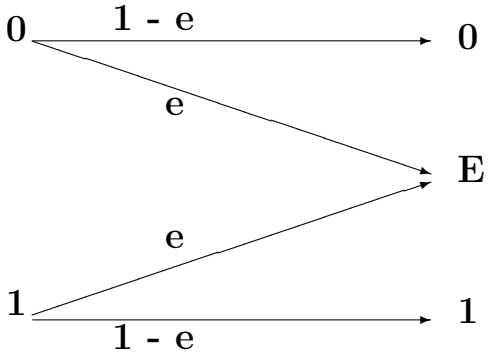
Z-канал (оптический)

0-light on, **1**-light off



MAC-канал (стирающий)

$p(X = 0) = p_0$.



BER – bit error rate – это средняя вероятность ошибки одного бита передаваемой информации.

1. Мобильные каналы: $BER \approx 10^{-2}$.
2. Проводные каналы: $BER \approx 10^{-5}$.
3. Оптоволоконные каналы: $BER \approx 10^{-12}$.

При помехоустойчивом кодировании возможны следующие две стратегии:

- Исправление ошибки за счет избыточности (FEC – forward error correction).
- Обнаружение ошибок с последующим запросом на повторную передачу ошибочно принятой информации (ARR – automatic repeat request).

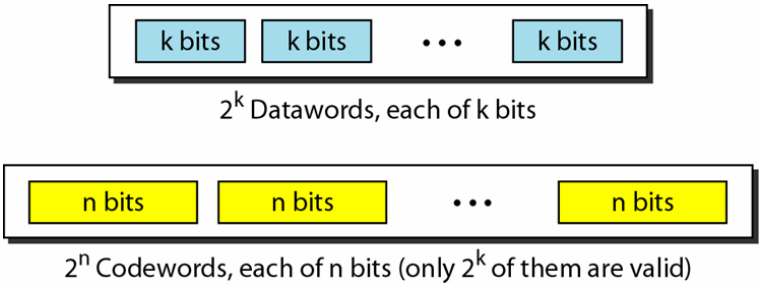


Рис. 7: Блоковые коды

Каждая из этих стратегий имеет свои достоинства и недостатки. Первая стратегия требует значительного увеличения избыточности, однако при сокращает расходы на исправление ошибок. Вторая уменьшает общую избыточность, однако увеличивает накладные расходы при повторной пересылке пакета. Поэтому, вторая стратегия выгоднее в тех случаях, когда величина BER является небольшой.

Области применения помехоустойчивого кодирования

1. Хранение информации с высокой плотностью записи – CD-ROM, DVD.
2. Передача данных при ограниченной мощности сигнала –спутниковая и мобильная связь.
3. Передача информации по сильно зашумленным каналам – высокоскоростные проводные линии связи, мобильная связь.
4. Передача данных по каналам связи с повышенными требованиями к надежности информации – вычислительные сети, линии

передачи со сжатием.

Пример кодирования.

Информационное слово	Кодовое слово
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

В общем случае, кодирование – это отображение из множества информационных слов I в множество кодовых слов C:

$$f: \ I \subseteq B^k \ \longrightarrow C \subseteq B^n, \quad B = \{b_1, \, b_2 \, \ldots, b_n\}.$$

В качестве алфавита B обычно берется множество $B = \{0, 1\}$.

Расстояние Хэмминга.

На множестве кодовых слов вводится некоторая метрика, называемая *расстоянием Хэмминга.* Расстояние Хэмминга между двумя словами есть число разрядов, в которых эти слова различаются.

Пример. Расстояние Хэмминга $d(000, 011) = 2$.

Декодирование – исправление ошибки, если она произошла. Декодирование неправильного слова выполняется в сторону правильного слова, до которого расстояние *наименьшее.*

Пример. Множество кодовых слов 00000, 01101, 10110, 11011. Если полученное слово 10000, то декодируем в «ближайшее»

слово 00000. Если же полученное слово равно 11000, то можно установить только факт ошибки, т.к. существует два варианта восстановления: 11000 может перейти в 00000 или в 11011.

Выводы: Если в процессе передачи по зашумленному каналу кодовое слово отобразится в другое кодовое слово, не совпадающее с переданным, то происходит *необнаруживаемая* ошибка –ошибка декодирования.

Хорошие коды должны иметь такую структуру, чтобы была возможность не только обнаруживать, но и исправлять ошибки.

3 Сведения из теории чисел

Пусть **Z** обозначает множество целых чисел. Все рассматриваемые в нашем пособии числа, если не указано особо, принадлежат **Z**.

Определение 3.1 *Говорят, что два целых числа* a *и* b сравнимы *по модулю* p*, записывается,*

$$a \equiv b \,(\bmod p),$$

если $p|(a-b)$ *(разность* $a-b$ *делится на* p *без остатка).*

Отношение сравнения по модулю натурального числа обладает следующими свойствами:

1. Рефлексивность: $a \equiv a \,(\bmod p)$.
2. Симметричность: $a \equiv b \,(\bmod p) \to b \equiv a \,(\bmod p)$.
3. Транзитивность: $a \equiv b \,(\bmod p) \,\&\, b \equiv c \,(\bmod p) \to a \equiv c \,(\bmod p)$.

Значит отношение сравнения по модулю является отношением эквивалентности на множестве целых чисел. Классы эквивалентности,

образованные целыми числами по этому отношению, называются *вычетами*. Вычет, содержащий число k, обозначается $\overline{k}$. Множество классов вычетов по модулю числа натурального $n > 0$ содержит ровно n элементов, записываемых как $\mathbf{Z}_n = \{\overline{0}, \overline{1}, \ldots \overline{n-1}\}$.

Над вычетами можно выполнять арифметические операции сложения, вычитания, умножения и возведения в степень, а если число n простое или является некоторой степенью простого числа, то и деление. Будем обозначать множество вычетов по модулю n через $\mathbf{Z}_n$.

Отметим, что для любых a и b выполняется формула $\overline{a} + \overline{b} = \overline{a+b}$ (то же для других перечисленных выше операции), поэтому операции над вычетами выполняются как над обычными числами, приводя результат к значению, принадлежащему интервалу $[0,\, n-1]$ путем выполнения операции вычисления остатка от деления результата на число n (т.е. операции, обозначаемой $\bmod n$). Например, в множестве $\mathbf{Z}_7$ $\overline{2} \cdot \overline{5} = \overline{10} = \overline{3}\,(\bmod 7)$. Чаще пишут просто: $2 \cdot 5 = 3\,(\bmod 7)$.

Множество классов вычетов по модулю n образует структуру, являющуюся *кольцом*. Кольцом K называется непустое множество элементов, на котором определены две арифметические операции *сложения* $+$ и *умножения* $\cdot$, относительно которых выполняются следующие формулы:

1. Ассоциативность по сложению: $(\forall a,\, b,\, c \in K)\ a + (b + c) = (a + b) + c$,
2. Существование нулевого элемента: $(\exists \mathbf{0} \in K)(\forall a \in K)\ a + \mathbf{0} = \mathbf{0} + a = a$,
3. Существование обратного элемента: $(\forall a \in K)(\exists b \in K)\ a + b = b + a = \mathbf{0}$
4. Ассоциативность по умножению: $(\forall a,\, b,\, c \in K)\, a \cdot (b \cdot c) = (a \cdot b) \cdot c$,
5. Дистрибутивность: $(\forall a,\, b,\, c \in K)\ a \cdot (b + c) = a \cdot b + a \cdot c$, $(b + c) \cdot a = b \cdot a + c \cdot a$.

Обратный по сложению к a элемент обозначается через $(-a)$. Множество элементов, удовлетворяющих только первым трем свойствам, называется *группой*. Если в группе $< G, + >$ выполняется свойство коммутативности $a + b = b + a$, то группа называется *коммутативной* или *абелевой*. Очевидно, что группа по сложению кольца $\mathbf{Z}_n$ является абелевой группой.

Если модуль n является простым числом, то множество ненулевых элементов кольца $\mathbf{Z}_n$ (обозначаемое через $\mathbf{Z}_n^*$) образует коммутативную группу по умножению, т.е. существует нейтральный элемент $\mathbf{1}$ $a \cdot \mathbf{1} = \mathbf{1} \cdot a$, и для каждого элемента a имеется обратный по умножению a^{-1} со свойством $a \cdot a^{-1} = \mathbf{1}$.

Алгебраические структуры, содержащие абелеву группы по сложению и группу по умножению, связанные законами дистрибутивности, называются *полями*. Конечные поля называют также полями Галуа по имени гениального французского математика Эвариста Галуа (1811 – 1832), исследовавшего эти поля, и обозначают $GF(q)$. Более подробные сведения о конечных полях читатель может получить из монографии Р.Лидла и Г.Нидеррайтера «Конечные поля» [68].

Пусть G–произвольная группа по умножению.

Определение 3.2 *Порядком элемента a группы G (обозначается через $ord_G(a)$) называется* наименьшее *число k такое, что $a^k = 1$. Порядком группы называется число ее элементов.*

Следующее свойство, связывающее порядки элементов с порядком группы, широко используется в различных алгоритмах, описанных ниже. Эта теорема была доказана знаменитым французским математиком Жозефом Луи Ланранжем (1736–1813).

Теорема 3.1 (Лагранж). *Порядок любого элемента конечной группы является делителем порядка группы.*

Доказательство. Пусть элемент a конечной группы $< G, \cdot >$ имеет порядок $k > 1$. Тогда элементы $a,\ a^2,\ \ldots\ ,\ a^{k-1},\ a^k = 1$ различны и сами образуют группу A, содержащую k элементов и являющуюся подгруппой G. Различные смежные классы $b \cdot A$ для $b \in G$ имеют также мощность k, а объединение их дает в совокупности группу G. Значит, число элементов G равно $k \cdot m$, где m–число смежных классов, откуда вытекает утверждение теоремы.

Пример. Рассмотрим кольцо $\mathbf{Z}_p$ при $p = 29$. Ненулевые элементы этого кольца образуют группу по умножению, порядок которой равен $p - 1 = 28$. По теореме Лагранжа порядок любого элемента a этой группы является делителем 28, т.е. может принимать одно из следующих значений: 1, 2, 4, 7, 14 и 28.

Элемент $a \in G$ называется *примитивным* элементом или *генератором* группы, если его порядок $ord_G(a)$ равен порядку группы. Не любая группа имеет генератор. Группа, в которой есть генератор, порождается одним элементом и называется *циклической.*

Малая теорема Ферма

Знаменитый французский математик Пьер Ферма (1601–1665) доказал теорему, которая известна как *малая теорема Ферма.*

Теорема 3.2 (Малая теорема Ферма) *Если число p–простое, то для любого натурального числа, не сравнимого с p выполняется сравнение*

$$a^{p-1} \equiv 1 \,(\bmod\ p) \qquad (3.1)$$

Эта теорема является частным случаем теоремы Лагранжа (теор.3.1). Действительно, при простом p множество ненулевых элементов кольца Z_p образует группу по умножению, имеющую $p-1$ элемент. Будем обозначать это множество через Z_p^*. По теореме Лагранжа порядок любого элемента $a \in Z_p^*$ является делителем порядка $p-1$, откуда $a^{p-1} \equiv 1 \,(\bmod\, p)$.

Из теоремы Ферма сразу следует, что если для некоторого $a < p$ выполнено условие $a^{p-1} \not\equiv 1 \,(\bmod\, p)$, тогда число p является составным. Однако обращение малой теоремы Ферма не верно – существуют составные числа p, для которых выполняется условие Ферма для каждого a, не сравнимого с p. Такие числа называются числами Кармайкла .

3.1 Функция Эйлера $\varphi(n)$

Знаменитый математик Леонард Эйлер (1707–1783), проживший в России большую часть своей жизни и написавший огромное количество математических трудов в разных областях математики, ввел в обиход функцию φ (Euler' totient function), определенную на целых положительных числах, значением которой на аргументе n является количество положительных чисел, меньших n и взаимно-простых с n.

Очевидно, что для всех $n > 1$ $\varphi(n) < n$, и для простого числа p значение $\varphi(n)$ равно $p-1$. Также выполнены и другие формулы, полезные для вычисления функции $\varphi(n)$:

$$\begin{aligned} &\varphi(p) = p-1 \text{ для всех простых } p, \\ &\varphi(p^k) = p^k - p^{k-1} \text{ для простых } p \text{ и натуральных } k\ , \\ &\varphi(n_1 \cdot n_2) = \varphi(n_1) \cdot \varphi(n_2) \end{aligned} \qquad (3.2)$$

Алгоритм RSA использует несколько вспомогательный алгоритмов, такие как алгоритм генерации простых чисел, алгоритм вычисления

обратного элемента по модулю и алгоритм быстрого возведения в степень, которые мы опишем в следующих параграфах.

3.2 Расширенный алгоритм Евклида

Расширенный алгоритм Евклида (РАЕ) используется во многих криптографических и теоретико–числовых алгоритмах. Он состоит из двух частей. В первой части алгоритма для заданных целых чисел A и B вычисляется их наибольший общий делитель (greatest common divisor d) d. Вычисление Н.О.Д. натуральных чисел A и B выполняется по рекуррентной формуле:

$$\text{Н.О.Д.}(A,\, B) = \text{Н.О.Д.}(B,\, A \bmod B), \tag{3.3}$$

где $A \bmod B$ означает операцию вычисления остатка при целочисленном делении A на B. Производится последовательное использование этой формулы, пока остаток от деления первого операнда на второй не станет равным 0. Последнее ненулевое значение второго операнда и есть искомый общий делитель:

```
int Euclid(int A,  B)
{
while (A mod B !=0) {
   int C=A mod B;
    A=B; B=C ; }
return B;
}
```

Для решения уравнений вида $Ax + By = d$, где A, B – заданные числа, а d – их наибольший общий делитель, используется *расширенный* алгоритм Евклида. Первая часть РАЕ в результате

которой мы находим Н.О.Д. d, выполняется также, как описано выше. Значения A, B, а также целую часть и остаток от деления A на B сохраняются в таблице, содержащей 4 столбца. Третий столбец содержит остаток от деления A на B, а четвертый столбец - целую часть от деления A на B.

Во второй части работы алгоритма к таблице добавляются два новых столбца, озаглавленных x и y. Поместим в последнюю строчку столбцов x и y значения 0 и 1. Затем, считая значения x_{i+1} y_{i+1} известными, последовательно вычисляем значения x_i и y_i, $i \geq 0$, по формулам:

$$x_i = y_{i+1}, \qquad y_i = x_{i+1} - y_{i+1} \cdot (A \, div \, B)_i$$

Пример. Решить уравнение $72x + 25y = 1$. Помещаем в первую строчку значения $A = 72$, $B = 25$. Вычисляем $A \, mod \, B$ – остаток от деления A на B, и $[A \, / B]$ – целую часть от деления A на B. Потом переносим значения B и $A \bmod B$ на строчку вниз и на одну клетку влево. Повторяем вычисления во второй строке. Продолжаем вычисления, пока значение в столбце $A \, mod \, B$ не станет равным 0. Тогда заносим в последнюю строчку столбцов x и y значения 0 и 1, и ведем вычисление снизу вверх по формулам, описанным выше.

A	B	A mod B	[A/B]	x	y
72	25	22	2	8	-25
25	22	3	1	-7	8
22	3	1	7	1	-7
3	1	0	3	0	1

Ответ: Н.О.Д.(72, 25) $= 1$ – последнее значение в столбце B. Пара $(x, y) = (8, -25)$, дающая решение уравнению $72x + 25y = 1$, берется из первой строки таблицы.

Пример 2. Найти обратный элемент для $e = 7$ по составному модулю $\varphi(n) = 40$ из примера параграфа **??**.

Решение. Запустим расширенный алгоритм Евклида, взяв $A = \varphi(n) = 40$ и $B = e = 7$. Получим:

A	B	A mod B	[A/B]	x	y
40	7	5	5	3	-17
7	5	2	1	-2	3
5	2	1	2	1	-2
2	1	0	2	0	1

Значение $y = -17$, находящееся в верхней строке, и есть искомое значение обратного элемента:

$$d = y \mod \varphi(n) = -17 \mod 40 = 23$$

Оценка сложности алгоритма Евклида

Расширенный алгоритм Евклида используется во многих криптографических методах, поэтому оценка его производительности играет важную роль в расчетах эффективности криптографических алгоритмов.

Основным фактором в оценке РАЕ является число итераций в главном цикле вычисления новой пары $(A;\ B)$, или, другими словами, число строчек в таблице вычисления вычислений. Чтобы оценить это число, заметим, что на шаге k произвольной итерации возможны два случая:

Случай 1. $B < A/2$. На шаге $k + 1$ новое значение A, равное предыдущему B, будет меньше, чем $A/2$.

Случай 2. $B \geq A/2$. Остаток $r = A \mod B = A - B$ будет меньше $A/2$, и на шаге $k + 2$ новое значение A станет равным остатку $r < A/2$.

В любом случае, после каждой пары итераций первый аргумент A уменьшается более, чем в 2 раза, значит, общее число итераций не может быть больше, чем $2\log_2 A$. Число операций на каждой итерации постоянно (как при прямом ходе, так и при подъеме при вычислении коэффициентов уравнения $Ax + By = d$), поэтому, оценка PAE равна $O(L)$, где $L = \lceil \log_2 A \rceil$–длина двоичного представления меньшего из чисел.

Эта оценка не является завышенной, т.к. существует последовательность *чисел Фибоначчи*, $\{F_n\}$, на парах соседних элементов которых и достигается эта верхняя оценка. Последовательность или ряд Фибоначчи определяется следующими формулами:

$F_0 = 1, \;\; F_1 = 1, \;\; F_{n+2} = F_n + F_{n+1}, \; n \geq 2.$

Выпишем начальный интервал этого ряда:

$S = \{1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ 233,\ 377,\ 610,\ 987,\ 1597,\ 2584,\ 418$

3.3 Алгоритм быстрого возведения в степень по модулю

Большинство операций в кольце вычетов $\mathbf{Z}_n$ можно выполнять, выполнив сначала действия с числами, а затем находя остаток от деления результата на модуль n. Однако с операциями возведения в степень и вычисления дискретного (модулярного) логарифма, такой порядок является очень неэффективным. Например, если мы захотим вычислять $2^{199} \pmod{1003}$, используя калькулятор, входящий в состав операционной системы Windows, то результат окажется неверным. В то же время эту же операции несложно выполнить с помощью алгоритма *быстрого возведения в степень по модулю* заданного натурального числа, который мы сейчас опишем.

Предположим, что требуется вычислить $z = a^b \bmod n$. Рассмотрим следующий алгоритм:

1. Представим b в двоичный системе исчисления: $b = (b_0\, b_1 \ldots b_k)_2$, $b_i \in \{0,\, 1\}$. Например, $199 = 11000111_2$,

2. Заполним следующую таблицу

b	b_0	b_1	...	b_k
a	a_0	a_1	...	a_k

где $a_0 = a$, $a_{i+1} = \begin{cases} a_i^2 \bmod n, & \text{если } b_{i+1} = 0, \\ a_i^2 \cdot a \bmod n, & \text{если } b_{i+1} = 1 \end{cases}$ для $i \geq 0$.

Результат появится в последней ячейке второй строки.

Пример. Вычислить $2^{199} \mod 1003$:

b	1	1	0	0	0	1	1	1
c	2	8	64	84	35	444	93	247

Ответ: $2^{199} \mod 1003 = 247$.

Приведем здесь еще один вариант алгоритма быстрого возведения в степень, не требующий предварительного перевода степени в двоичное представление:

```
long powm(long a, long b, long n) {
   long c = 1;
   while (b) {
     if (b%2 == 0)
       { b/ = 2; a = (a * a)% n;}
     else
       { b − −; c = (c * a)% n;}
     }
   return c;
}
```

3.4 Генерация простых чисел. Решето Эратосфена.

Очевидно, что любое простое число, не равное 2, является нечетным. Существуют признаки делимости целых чисел на различные простые числа, например, чтобы число в десятичном виде делилось на 3 и 9 достаточно, чтобы сумма его цифр делилась на 3 и 9 соответственно. Чтобы число делилось на 5, достаточно, что его последняя цифра была 0 или 5.

Такие частные признаки делимости можно использовать, если нужно уменьшить множество кандидатов проверки на простоту или отсечь заведомо составные числа. Альтернативным способом получения простых чисел является *решето Эратосфена*, приписываемое древнегреческому ученому Эратосфену Киренскому, жившему примерно в 276 - 194 г. до н.э.

Для нахождения множества простых до заранее выбранной верхней границы B мы сначала выписываем последовательность всех нечетных чисел от 3 до B. Затем выбираем первое число в списке, т.е. тройку, и оставляя его в списке, вычеркиваем все кратные 3, начиная с 6. Потом переходим ко второму числу списка (пятерке) и вычеркиваем его кратные, оставив саму пятерку и т.д., пока не дойдем до конца списка. В оставшемся списке будут только простые числа.

Следующая теорема дает критерий проверки простоты числа p и одновременно примитивности корня a:

Теорема 3.3 *(Критерий примитивности и простоты). Если для некоторых a и p выполнены условия:*

1. $a^{p-1} \equiv 1(\bmod p)$,
2. $a^{(p-1)/q} \not\equiv 1(\bmod p)$ *для* $\forall q|(p-1)$,

тогда число p–простое, и a является примитивным корнем поля GF_p (т.е. генератором группы по умножению поля GF_p).

Пример. $n = 1\ 022\,333\,835\,329\,657, \quad n-1 = 2 \cdot 2957 \cdot 146\,063 \cdot 292\,877.$

$3^{n-1} \equiv 1(\bmod n)$,
$3^{(n-1)/2} \equiv -1\ (\bmod n)$,
$3^{(n-1)/2597} \equiv 324224767363906\ (\bmod n)$,
$3^{(n-1)/146\,063} \equiv 697302646321792\ (\bmod n)$,
$3^{(n-1)/292\,877} \equiv 736785752408036\ (\bmod n)$.

Поэтому число n в нашем примере является простым, а 3 является примитивным корнем поля Галуа GF_n.

Отметим, что разбиение $n-1$ в произведение простых сомножителей само является очень сложной задачей, поэтому для длинных чисел этот критерий простоты неприменим.

3.5 Метод пробных делений

Метод пробных делений (the trial division) является наиболее простым методом проверки простоты входного составного числа n или нахождения его делителей. Будем использовать обозначение $\lfloor x \rfloor$ для функции floor(x), равной наибольшему целому числу, не превышающему x (округление вниз). Аналогично, $\lceil x \rceil$ используется для обозначения функции ceil(x), равной наименьшему целому числу, большему или равному x (округление вверх).

Для этого в цикле выполняется пробное деление n на все целые числа от 2 до $\sqrt{n}$:

```
int Tr_div(int n)
{
for(int i = 2; i < ⌊√n⌋; i++)
if (n%i == 0) return i;
return 0}
```

Каждое деление имеет асимптотическую сложность $O(log^2 n)$, поэтому общая сложность метода может быть оценена как $O(n^{1/2} log^2 n)$. Обозначим через L длину двоичного представления числа n, $L = \lceil \log_2 n \rceil$. Тогда можно записать последнюю оценку в более стандартном для теории вычислимости виде:

$$T(n) = O(L^2 \cdot e^{L/2}). \tag{3.4}$$

Значит, алгоритм пробных делений имеет экспоненциальную оценку относительно длины входного числа, поэтому этот метод не может быть использован для тестирования больших чисел.

3.6 Решето Аткина

Решето Аткина — быстрый современный алгоритм нахождения всех простых чисел до заданного целого числа.Это оптимизированная версия старинного решета Эратосфена: решето Аткина проделывает некоторую предварительную работу, а затем вычеркивает числа, кратные квадрату простых. Алгоритм был создан А. Аткиным (A. Atkin) и Д. Бернштейном (D. Bernstein) [2].

Ниже представлена упрощенная версия кода, иллюстрирующая основную идею алгоритма – использование квадратичных форм.

```
int limit = 1000;
int sqr_lim;  bool  is_prime[1001];  int x2,  y2;  int i, j; int n;
// Инициализация решета
sqr_lim =  (int) sqrt((long double) limit);
for (i = 0; i <= limit; i++) is_prime[i] = false;
is_prime[2] = true; is_prime[3] = true;
// Предположительно простые - это целые с нечетным числом
// представлений в данных квадратных формах.
// x2 и y2 - это квадраты i и j (оптимизация).
x2 = 0;
for (i = 1;  i <= sqr_lim; i++) {
    x2 + = 2 * i - 1;
```

```
    y2 = 0;
    for (j = 1;  j <= sqr_lim;  j++) {
        y2 += 2 * j - 1;
        n = 4 * x2 + y2;
        if ((n <= limit) && (n % 12 == 1 || n % 12 == 5))
            is_prime[n] = ! is_prime[n];
        // n = 3 * x2 + y2;
        n -= x2; // Оптимизация
        if ((n <= limit) && (n % 12 == 7))
            is_prime[n] = ! is_prime[n];
        // n = 3 * x2 - y2;
        n -= 2 * y2; // Оптимизация
        if ((i > j) && (n <= limit) && (n % 12 == 11))
            is_prime[n] = ! is_prime[n];
        }
    }
// Отсеиваем квадраты простых чисел в интервале [5,  √limit].
// (основной этап не может их отсеять)
for (i = 5;  i <= sqr_lim;  i++) {
        if (is_prime[i]) {
            n = i * i;
        for (j = n;  j <= limit;  j += n) {
            is_prime[j] = false;
        }
    }
}
// Вывод списка простых чисел в консоль.
printf("2, 3, 5");
for (i = 6; i <= limit; i++) {
// добавлена проверка делимости на 3 и 5. В оригинальной
// версии алгоритма потребности в ней нет.
        if (is_prime[i]) && (i % 3 <> 0) && (i % 5 <> 0){
            printf(" % d " , i);  }
}
```

Обоснование алгоритма. Алгоритм основан на следующей теореме Аткина:

Теорема. Пусть n–натуральное число, свободное от квадратов (т.е. не делящееся ни на какой квадрат простого числа) и удовлетворяющее

условию $n \equiv 1 \mod 4$. Тогда, n – просто тогда и только тогда, когда

$$\#S = |\{(x, y) : \ x > 0, y > 0, 4x^2 + y^2 = n\}| \text{ – нечетно}$$

Алгоритм полностью игнорирует любые числа, которые делятся на три, пять и семь. Все числа, четные по модулю 60, делятся на два и заведомо не простые. Все числа, равные (по модулю 60) 3, 9, 15, 21, 27, 33, 39, 45, 51 или 57, делятся на три и тоже не являются простыми. Все числа, равные (по модулю 60) 5, 25, 35 или 55, делятся на пять и также не простые. Все эти остатки (по модулю 60) игнорируются.

Все числа, равные (по модулю 60) 1, 13, 17, 29, 37, 41, 49 или 53, имеют остаток от деления на 4 равный 1. Эти числа являются простыми тогда и только тогда, когда количество решений уравнения $4x^2 + y^2 = n$ нечётно и само число не является квадратом (squarefree).

Числа, равные (по модулю 60) 7, 19, 31 или 43, имеют остаток от деления на 6 равный 1. Эти числа являются простыми тогда и только тогда, когда количество решений уравнения $3x^2 + y^2 = n$ нечётно и само число не является квадратом.

Числа, равные (по модулю 60) 11, 23, 47 или 59, имеют остаток от деления на 12 равный 11. Эти числа являются простыми тогда и только тогда, когда количество решений уравнения $3x^2 - y^2 = n$ нечётно и само число не является квадратом.

Ни одно из рассматриваемых чисел не делится на 2, 3 или 5, значит они не могут делиться и на их квадраты. Поэтому проверка того, что число не является квадратом, не включает чисел 22, 32 и 52.

Оценка сложности решета Аткина

По оценке авторов алгоритм имеет асимптотическую сложность

$$O\left(\frac{n}{\ln \ln n}\right)$$

и требует $O(n^{1/2+o(1)})$ бит памяти. Ранее были известны столь же асимптотически быстрые алгоритмы, но они требовали существенно больше памяти.

3.7 Тест Поклингтона

Если у числа $n-1$ найдено один или несколько простых делителей, то это позволяет ограничить область значений простых делителей числа n или даже показать, что n является простым. Следующая теорема подтверждает это наблюдение:

Теорема 3.4 *(H.C. Pocklington). Пусть $n-1 = F \cdot R$, и полное разложение множителя F на простые множители известно. Тогда, если для некоторого $a < n$ выполняются условия:*

1. $a^{n-1} \equiv 1 \pmod{n}$,
2. *Н.О.Д.*$(a^{(n-1)/q}, n) \neq 1$ *для любого* $q|F$,

тогда любой делитель числа n сравним с 1 по модулю p.

Доказательство. Пусть p–простой делитель числа n. Из п.1 предположений теоремы следует, что порядок k элемента a^R в мультипликативной группе поля GF_p является делителем $(n-1)/F = R$. Из второго пункта предположений следует, что

k не может быть собственным делителем, т.е. $k = F$. Отсюда $F|(p-1)$, т.е. $p = 1 + m \cdot F$ для некоторого целого m.

Следствие. Если $F > \sqrt{n}$, тогда число n–простое.

Действительно, в этом случае, любой нетривиальный делитель p числа n должен быть больше $\sqrt{n}$, что невозможно.

Пример. Пусть $n = 618\,970\,019\,642\,690\,137\,449\,462\,111$. Число $n-1$ имеет полное разложение вида

$$n - 1 = 2 \cdot 3 \cdot 5 \cdot 17 \cdot 23 \cdot 89 \cdot 353 \cdot 397 \cdot 683 \cdot 2113 \cdot 2\,931\,542\,417.$$

Отметим, что наибольший делитель $n-1$, равный $2\,931\,542\,417$, меньше $\lfloor\sqrt{n}\rfloor = 24\,879\,108\,095\,803$.

Базис $a = 2$ не подходит по условию теоремы, т.к. $2^{(n-1)/q} \equiv 1 \,(\bmod\, n)$ для всех делителей q. Выполним тест с базой $a = 3$, $p = 2\,931\,542\,417$:

$$m = 3^{(n-1)/p} - 1 \equiv 180\ 591\ 065\ 836\ 317\ 083\ 554\ 066\ 745 \neq \pm 1 \,(\bmod\, n),$$

и, Н.О.Д.$(n,\, m) = 1$. Значит, возможные делители числа n имеют вид $1 + k \cdot p < \sqrt{n}$, откуда, $0 < k < 8486$. Простым перебором всех k можно убедиться, что n не имеет простых делителей, и, значит, является простым.

Можно было также вместо выполнения делений применить теорему для F, равного произведению трех наибольших делителей $n-1$ (для них годится та же база $a = 3$), тогда $F > \lfloor\sqrt{n}\rfloor$, откуда сразу следует, что n–простое.

3.8 Генерация простых чисел

Рассмотрим один способ генерации больших простых чисел, основанный на тесте Поклингтона (стр.42) Пусть задано

простое число p:

1. Выберем случайным образом чётное число R на промежутке $p \leq R \leq 4p + 2$ и определим $n = pR + 1$.

2. Проверим число n на отсутствие малых простых делителей, разделив его на малые простые числа.

3. Выполним для числа n тест Миллера-Рабина (см.ниже на с.46) с использованием нескольких различных баз $a < p$. Если при одном из тестов выяснится, что n–составное число, то выберем новое значение R и повторим вычисления.

Оценка эффективности этого метода зависит от плотности распределения простых чисел и расстояния между соседними простыми числами. Вопрос этот является вовсе не простым и зависит от справедливости обобщенной гипотезы Римана (ОГР). Если допустить справедливость ОГР, то этот алгоритм является полиномиальным.

3.9 Символ Лежандра

Определение 3.3 *Пусть $n > 1$–целое число. Число a, принадлежащее интервалу* $[0,\, n-1]$ *называется* квадратичным вычетом *по модулю n, если найдется целое число x такое, что* $x^2 \equiv a \pmod{n}$.

Если такого x не существует, то a называется *квадратичным невычетом*. Отметим, что ровно половина элементов из интервала $[0,\, n-1]$ является квадратичными вычетами.

Условия того, является ли a квадратичным вычетом по простому модулю p, проверяется с помощью, так называемого, символа

Лежандра:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, \text{ если } (\exists\, x)\, x^2 \equiv a \bmod p, \\ -1, \text{ если не}(\exists\, x)\, x^2 \equiv a \bmod p, \\ 0, \text{ если } p \mid a. \end{cases} \tag{3.5}$$

Вычисление символа Лежандра может быть выполнено по следующей формуле, полученной Леонардом Эйлером:

$$\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \mod n. \tag{3.6}$$

Однако использование этой формулы на практике сопряжено с вычислениями больших степеней, поэтому предпочтительнее пользоваться законом квадратичной взаимности, доказанный Карлом Гауссом в возрасте 17 лет.

Закон квадратичной взаимности: для любых нечетных простых чисел p и q выполняется формула

$$\left(\frac{q}{p}\right) = \left(\frac{p}{q}\right)(-1)^{(p-1)(q-1)/4}.$$

Иначе говоря,

$$\left(\frac{q}{p}\right) = -\left(\frac{p}{q}\right), \text{ если } p \equiv q \equiv 3 \bmod 4, \quad \text{и} \quad \left(\frac{q}{p}\right) = \left(\frac{p}{q}\right), \text{ иначе.}$$

Гаусс в течение своей жизни неоднократно возвращался к этому закону и получил несколько его доказательств, основанных на совершенно различных идеях.

Для быстрого вычисления символа Лежандра полезными являются также следующие формулы:

$$\left(\frac{q}{p}\right) = \left(\frac{q \bmod p}{p}\right), \qquad \left(\frac{q \cdot r}{p}\right) = \left(\frac{q}{p}\right) \cdot \left(\frac{r}{p}\right), \qquad \left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} \mod n.$$

Пример. Вычислить (15/17):

$$\left(\frac{15}{17}\right) = \left(\frac{3}{17}\right)\cdot\left(\frac{5}{17}\right) = \left(\frac{2}{3}\right)\cdot\left(\frac{2}{5}\right) = (-1)\cdot(-1)^3 = 1$$

Для составных чисел n используется символ Якоби, который является обобщением символа Лежандра на произвольные целые числа и обладает следующим свойством:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{r_1} \cdot \left(\frac{a}{p_1}\right)^{r_2} \cdot \ldots \left(\frac{a}{p_k}\right)^{r_k}, \tag{3.7}$$

где $n = p_1^{r_1} \cdot p_2^{r_2} \cdot \ \ldots\ p_k^{r_k}$–разложение n в произведение степеней простых чисел.

3.10 Тест простоты Миллера–Рабина

В качестве критерия проверки, является ли заданное число n простым или составным, может служить следующая теорема:

Теорема 3.5 *(Критерий непростоты) Нечетное число* $n \geq 3$ *является* составным *тогда и только тогда, когда* n *является либо полным квадратом, либо найдутся два натуральных числа* x *и* y *такие, что*

$$x \not\equiv \pm y \,(\bmod n),\ \textit{и}\ x^2 \equiv y^2 \,(\bmod n). \tag{3.8}$$

Доказательство. Если выполняются условия (3.8) , то Н.О.Д.$(n, x^2 - y^2) \neq 1,\ \neq n$. Обратно, если $n = p \cdot q$, где $p > q$, то определим $x = (p+q)/2$, $y = (p-q)/2$. Очевидно, x и y удовлетворяют (3.8).

На критерии непростоты основан известный вероятностный тест Миллера–Рабина, который старается найти пару x и $y = 1$, удовлетворяющие критерию непростоты.

Пусть n–число, которое необходимо проверить на простоту. Представим $n-1$ в виде $n-1=2^s\cdot d$, где d–нечетно. Назовем произвольное число $a\in\mathbf{Z}_n^*$ *свидетелем простоты* n, если выполняет одно из следующих условий:

$$\begin{aligned}&1.\ x=a^d\equiv\pm1\,(\mathrm{mod}\,n),\ \text{или}\\&2.\ (\exists k,\,0<k<s)\ x^{2^k}\equiv-1\,(\mathrm{mod}\,n).\end{aligned}\tag{3.9}$$

В противном случае, назовем a *свидетелем непростоты* n.

Докажем сначала, что если для числа n найдется хотя–бы один свидетель его непростоты, то n–составное.

Действительно, пусть для некоторого $a\in\mathbf{Z}_n^*$ не выполнен ни один из п.1–2 условий (3.9), тогда последовательность

$$x_0=a^d(\ \mathrm{mod}\ n),\ x_1=x_0^2(\ \mathrm{mod}\ n),\ \ldots,\ x_{s-1}=x_{s-2}^2(\ \mathrm{mod}\ n)$$

не содержит -1. Вычислим x_s, равное $x_{s-1}^2(\mathrm{mod}\,n)$. Оно не равно 1. Но если n–простое, то $x_s=a^{d\cdot2^s}=a^{n-1}$ должно равняться по малой теореме Ферма единице. Значит, число n – составное.

Для оценки эффективности теста Миллера–Рабина определим понятие функции Эйлера.

Рабин доказал теорему о том, что если нечетное число $n>2$–составное, то множество свидетелей его простоты имеет мощность не более $\varphi(n)/4<n/4$. Отсюда следует, что если при проверке k произвольно выбранных чисел $a<n$ все они окажутся свидетелями простоты n, то n–простое с вероятностью ошибки, не превышающей 4^{-k}. На этом наблюдении строится следующий тест Миллера–Рабина.

Тест Миллера–Рабина

Пусть число $n > 2$–нечетно и $n-1 = 2^s \cdot d$, где d–нечетно. Для каждого числа a от 2 до $r+1$, где r – число проверок в тесте, выполним следующие действия:

1. Вычислим $x_0 = a^d (\mathrm{mod}\, n)$.

2. Проверим условие $x_0 \in \{1,\ n-1\}$. Если оно выполнится, тогда a–свидетель простоты. Перейдем к следующему a.

3. Иначе проверим, содержится ли число $n-1$ в последовательность $\{x_1,\ x_2,\ ...,\ x_{s-1}\}$, где каждый последующий x вычисляется по формуле $x_{i+1} = x_i^2 \,(\mathrm{mod}\, n)$.

Если ответ положительный, то a–свидетель простоты. Перейдем к следующему $a \le r+1$.

Иначе, найден свидетель непростоты n. Завершаем тест с сообщением «число n–составное».

Если после r проверок окажется r свидетелей простоты, то заканчиваем тест с сообщением «n–вероятно простое».

Пример. Пусть $n = 1729$. Разложим $n-1 = 2^6 \cdot 3^3$. Выполним тест Миллера–Рабина для $a = 2$:

$x_0 = 2^{27} \bmod 1729 = 645 \neq 1, \neq n-1,$

$x_1 = x_0^2 \bmod 1729 = 645^2 \bmod 1729 = 1065.$

$x_2 = x_1^2 \bmod 1729 = 1065^2 \bmod 1729 = 1.$

Последующие элементы $\{x_i\}$ для $i = 3, 4, 5$ равны 1, и последовательность $\{x_1,\ x_2,\ ...,\ x_{s-1}\}$, не содержит $n-1$. Значит, 2 является свидетелем непростоты n, и $n = 1729$ – составное число.

Оценка эффективности теста Миллера–Рабина

Следующая лемма была доказана Рабином в предположении справедливости обобщенной гипотезы Римана о распределении простых чисел:

Лемма 3.1 *Пусть число n–нечетное число, и $n-1=2^s\cdot d$, где d–нечетно. Если для всех x, $0<x<2\cdot(\,log_2\,n)^2$ выполняется $x^d\equiv 1\,(\mathrm{mod}\,n)$, или $x^{2^k\cdot d}\equiv -1\,(\mathrm{mod}\,n)$ для некоторого $0\leq k<s$. Тогда число n является простым.*

Оценка, приведенная в этой лемме, является полиномиальной, однако, с теоретической точки зрения она не может быть использована, пока не доказана обобщенная гипотеза Римана, которая на сегодняшний день является самой известной из нерешенных «проблем тысячелетия». Кроме того, для практических расчетов эта оценка является сильно завышенной. Вместо нее обычно используется граница порядка $\mathrm{O}(log_2 n)$.

3.11 Вероятностный тест простоты Соловея–Штрассена

Тест Соловея — Штрассена опирается на малую теорему Ферма (разд. 3.2) и свойства символа Якоби (разд. 3.9):

Теорема 3.6 *Если n — нечетное составное число, то количество целых чисел a, взаимно простых с n и меньших n, удовлетворяющих сравнению*

$$a^{(n-1)/2}\equiv\left(\frac{a}{n}\right)\ (\,\mathrm{mod}\,n),\qquad(3.10)$$

не превосходит $n/2$.

Алгоритм Соловея — Штрассена

Сначала для алгоритм Соловея — Штрассена выбирается целое число $k \geq 1$. Тест проверки простоты числа n состоит из k отдельных раундов. В каждом раунде выполняются следующие действия:

1. Случайным образом выбирается число $a < n$, и вычисляется $d =$Н.О.Д.(a, n).

2. Если $d > 1$, то выносится решение о том, что n составное. Иначе проверяется сравнение (3.10). Если оно не выполнено, то n - составное. Иначе, a является свидетелем простоты числа n.

Если после завершения k раундов найдено k свидетелей простоты, то делаем заключение «n–вероятно простое число».

Вычислительная сложность и эффективность теста

В каждом раунде вероятность отсеять составное число больше $1/2$, поэтому через k раундов тест Соловея–Штрассена определяет простое число с вероятностью ошибки, меньшей 2^{-k}. Поэтому этот тест сравним по эффективности с тестом Ферма, но имеет преимущество перед тестом Ферма в том, что он отсеивает все числа Кармайкла (числами Кармайкла называются нечетные составные натуральные числа n такие, что для каждого натурального a, $1 \leq a < n$, выполнено условие $a^{n-1} \bmod n = 1$).

С другой стороны, он проигрывает тесту Миллера–Рабина, который за k раундов имеет ошибку, меньшую 4^{-k}.

Общая вычислительная сложность алгоритма оценивается как $O(k \log_2 n)$.

3.12 Полиномиальный критерий простоты AKS

Одной из важных проблем, долгое время стоявших перед исследователями, была проблема построения детерминированного алгоритма проверки простоты натуральных чисел, имеющего полиномиальную оценку времени работы. Алгоритм Миллера–Рабина, упомянутый в предыдущем разделе, имеет полиномиальную оценку, но не является детерминированным. Другие тесты, например тест Поклингтона (разд.3.7), являются детерминированными, но не имеют полиномиальной оценки.

В 2004 г. тремя молодыми индийскими математиками Агравелой, Каялом и Саксеной ([1]) был разработан детерминированный полиномиальный безусловный тест AKS проверки простоты заданного натурального числа. Тест AKS основывается на следующей теореме:

Теорема 3.7 *(Agrawel, Kayal, Saxena [2004].) Пусть n–нечетное натуральное число, r–простое число и выполнены условия:*

1. *Число n не делится ни на одно из чисел, меньших или равных r,*
2. *Порядок n в мультипликативной группе $\mathbf{Z}_p^*$ поля GF_p не меньше $(\log_2(n))^2$,*
3. *Для всех a, $0 \le a \le r$, выполнена формула*

$$(X+a)^n \equiv X^n+a \text{ в кольце многочленов } \mathbf{Z}_n[X]/\frac{X^r-1}{X-1}\,. \tag{3.11}$$

Тогда число n является простым.

В этой теореме используются вычисления в кольце многочленов $\mathbf{Z}_n[X]$ с коэффициентами, ограниченными сверху числом n,

факторизованных по модулю многочлена деления круга

$$\Phi_r(X) = \frac{X^r - 1}{X - 1} = X^{r-1} + X^{r-2} + ... + X + 1.$$

Конечно, если n–просто, то эквивалентность $(X + a)^n \equiv X^n + a \bmod n$ в силу малой теоремы Ферма выполняется и в кольце $\mathbf{Z}_n[X]$, однако эти вычисления слишком громоздки, чтобы их можно было реально выполнить. Суть замечательной идеи Агравелы, Каялы и Саксены состояла в том, чтобы заменить кольцо $\mathbf{Z}_n[X]$ на гораздо меньшее кольцо $\mathbf{Z}_n[X]/\Phi_r(X)$.

На этой теореме основан следующий тест проверки простоты числа n:

1. Проверим, что n не является полным квадратом,
2. Используя числа $r = 2, 3, 5, ...$, найдем наименьшее простое число r такое, что r не является делителем n, и не является делителем $n^i - 1$ для всех $i \in \{0, 1, 2, ..., (\log_2 n)^2)\}$.
3. Проверим, что выполнены условия пункта 3 теоремы.

Если эти условия выполнены, то n–простое, иначе n–составное.

Замечание. Несмотря на то, что тест AKS явился решением крупной и долгостоявшей научной проблемы, он является не слишком удобным с практической точки зрения. Проверка условий пункта 3 является настолько громоздкой, что общая оценка времени работы алгоритма достигает $O(\log^{18} n)$ (см. Д. Вентури. Лекции по алгоритмической теории чисел [38]). Поэтому этот тест следует применять лишь в тех случаях, когда надо получить *гарантированное* доказательство того, что число n является простым.

В следующем главе мы выясним, как распределяются простые числа и дадим формулировку знаменитой *проблемы Римана.*

3.13 Извлечение квадратного корня в конечных полях

В реализациях методов квадратичного решета и решета числового поля, описываемых в 4-й и 5-й главах, будет использован алгоритм извлечения квадратного корня в конечных полях, разработанный Шенксом и Тоннелли. Опишем данный алгоритм в этом параграфе.

Рассмотрим конечное поле $GF_p,\ p > 2$, и элемент a, являющийся квадратичным вычетом по модулю p. Требуется найти x такое, что $a \equiv x^2 \pmod p$.

Представим число $p-1$ в виде $p-1 = 2^r \cdot s$, где s–нечетно. Заметим, что поскольку $p-1$–четно, $r \geq 1$. Пусть z–некоторый квадратичный невычет по модулю p (его можно найти просто перебором по элементам F_p, пока символ Лежандра (z/p) не окажется равным -1).

Рассмотрим 2 случая:
1. $p \equiv 3 \pmod 4$. В этом случае можно сразу найти решение

$$x = a^{\frac{p+1}{4}} \pmod p$$

2. $p \equiv 1 \pmod 4$.

Вычислим $y = z^s \mod p$. Поскольку порядок любого элемента является делителем числа $2^r \cdot s$, то порядок y является делителем 2^r, откуда $y^{2^r} \equiv 1 \pmod p$. Можно также показать, что $y^{2^r-1} \equiv -1 \pmod p$, т.е. порядок элемента y равен в точности 2^r. Вычислим далее элементы

$$\lambda_0 = a^s \pmod p, \quad w_0 = a^{(s+1)/2} \pmod p. \tag{3.12}$$

Заметим, что

$$w_0^2 \equiv a \cdot \lambda_0 \pmod p \ \text{ и } \ x^2 \equiv a \pmod p \rightarrow x^{2s} \equiv a^s = \lambda_0 \pmod p. \tag{3.13}$$

Поскольку порядок элемента x^s является делителем 2^r, то порядок λ_0 является делителем 2^{r-1}. Идея метода Шенкса–Тоннелли состоит в построении последовательности пар чисел $(\lambda_i,\, w_i)$, удовлетворяющих условию

$$w_i^2 \equiv a{\cdot}\lambda_i \ (\mod p), \ \ i = 0,\, 1,\, 2,\, \ldots\,, \tag{3.14}$$

причем порядок λ_{i+1} является собственным делителем порядка λ_i, до тех пор, пока порядок очередного λ_i не окажется равным 0. Тогда для найденного i выполнятся условия $\lambda_i = 1$ и

$$w_i^2 \equiv a \ (\mathrm{mod}\, p),$$

откуда $x = w_i$ является искомым корнем.

Поскольку исходные значения $(\lambda_0,\, w_0)$, удовлетворяющие (3.14), согласно (3.13), уже определены, то осталось просто описать формулы для вычисления значений $(\lambda_{i+1},\, w_{i+1})$:

$$\lambda_{i+1} = \lambda_i{\cdot}y^{2^{r-m}}, \quad w_{i+1} = w_i{\cdot}y^{2^{r-m-1}}, \tag{3.15}$$

где 2^m–порядок элемента λ_i.

Пример. Рассмотрим пример вычисления квадратного корня из $a = 2$ в простом поле GF_p при $p = 41$:

1. Имеем, $p - 1 = 40 = 2^3 \cdot 5$, откуда, $s = 5$, $r = 3$.

2. Вычислим исходные значения $(\lambda_0,\, w_0)$ по формулам (3.12):

$$\lambda_0 = a^s \ (\mod p) = 2^5 \ (\mod 41) = 32,$$

$$w_0 = a^{(s+1)/2} \ (\mod p) = 2^3 \ (\mod 41) = 8.$$

3. Найдем порядок элемента λ_0:

$$\lambda_0^2 \bmod p = 32^2 \mod 41 = 40 \equiv -1 \ (\mod 41), \ \lambda_0^4 \equiv 1 \mod p.$$

Отсюда, $ord(\lambda_0) = 2^m = 4$, $m = 2$.

4. Будем искать квадратичный невычет. Вычислим символ Лежандра для $z = 3$:

$$\left(\frac{z}{p}\right) = \left(\frac{3}{41}\right) = \left(\frac{41 \bmod 3}{3}\right)(-1)^{(41-1)(3-1)/2} = \left(\frac{2}{3}\right) = -1,$$

значит, $z = 3$ является квадратичным невычетом и может быть использован для вычисления пар (λ_{i+1}, w_{i+1}).

5. Найдем $y = z^s (\bmod\ p) = 3^5 (\bmod\ 41) = 38$.

6. Вычислим степень, в которую надо возводить y:
$d = 2^{r-m} = 2^{3-2} = 2, \quad y^d = 3^2 = 9.$

7. Вычислим $\lambda_1 = \lambda_0 \cdot y^d (\bmod\ p) = 32 \cdot 9 (\bmod\ 41) = 1,\ w_1 = w_0 \cdot y^{d-1} (\bmod\ p) = 8 \cdot 3 (\bmod\ 41) = 24$. Поскольку очередное λ_i оказалось равным 1, то процедура закончена. Корень $x = w_1 = 24$. Выполним проверку:

$$x^2 \bmod\ p = 24^2 \ \bmod\ 41 = 2 = a.$$

4 Список литературы

Список литературы

[1] Agrawal M. *PRIMES is in P* / M.Agrawal, N.Kayal, N.Saxena.– Annals of Mathematics.– 2004, v.160, p. 781–793.

[2] Atkin A. *Prime sieves using binary quadratic forms*/ A. Atkin, D. Bernstein.– http://cr.yp.to/papers/primesieves–19990826.pdf

[3] Berstein D. *ECM using Edwards curves*/ D. Berstein, P. Birkner, T.Lange, C. Peters.–2008, p.1–40 http://eecm.cr.yp.to/eecm-20100616.pdf

[4] Berstein D. *Faster addition and doubling on elliptic curves.*/ D. Berstein, T.Lange. in AsiaCrypt'2007, p.29–50

[5] Berstein D. *Explicit-formulas Database.*/ D. Berstein, T.Lange. 2007 http:// hyperelliptic.org/EFD

[6] Berstein D. *Starfish on Strike*/ D. Berstein, P. Birkner, T.Lange, C. Peters.–LATINCRYPT 2010, edited by Michel Abdalla and Paulo S. L. M. Barreto. Lecture Notes in Computer Science 6212. Springer, 2010, p.61–80

[7] Boldyreva A. *Efficient Thresholf Signature, Multisignature and Blind Signature Schemes based on Diffie–Hellman–Group Signature Scheme.* Crypto'2003, Lect.Not.Comp.Sci., p.31–46

[8] Boneh D.,Franklin M. *Identity based encryption from the Weil pairing.* In J.Killan, editor, Proceeding of Crypto'2001, volume 2139, Lect.Notes in Comp.Sci., 2001, p.213–229

[9] Brent R.P. *Some integer factorization algorithms using elliptic curves/* R.P. Brent.– Austral.Comput.Sci.Comm, 1986, v.8, p. 149–163.

[10] Buhler J.P. *Factoring integers with the number field sieve /* J. P. Buhler, H. W. Lenstra, C. Pomerance.– in The Developement of the Number Field Sieve, Springer–Verlag, Berlin, Germany, 1993, p. 50–94.

[11] Chaum D. *Zero-knowledge undeniable signatures.* In I.Damgard, editor, Advances in Cryptology–Crypto'90, Lect.Not.Comp.Sci., v.740, 1992, p.89-105

[12] Cocks C. *An identity based encryption scheme based on quadratic residues.* Cryptography and Coding, 2001.

[13] Cohen H. *A course in computational algebraic number theory* / H. Cohen.– Springer–Verlag, Berlin, 1993, 545 p.

[14] Crandall R. *The prime numbers: a computational perspertive* / R. Crandall, C. Pomerance.– sec.ed. Springer–Verlag, Berlin, 2005, 604 p.

[15] Dunham W. *Euler : The Master of Us All.* Mathematical Association of America, 1999, 185 p.

[16] Edwards H.M. *A normal form for elliptic curves./* H.M. Edwards.–Bull. Amer. Math. Soc. 44 (2007), p. 393-422

[17] Elkenbracht-Huising M. *An implementation of the Number Field Sieve* / M. Elkenbracht-Huising.– Experimental Mathematics, 1996, v.5, p. 231—-253.

[18] Gardner M. *A new kind of cipher that would take millions years to break* / M. Gardner.– Sci. Amer. 1977, p. 120–124.

[19] Golomb S.W *Shift register sequences* San Francisco, Holden-Day — 1967 — 115 p.

[20] Granville A.*Smooth numbers: Computational number theory and beyond*/ A. Granville.– Proc. of MSRI workshop, 2004, 268–363

[21] Hackmann P. *Elementary Number Theory* / P. Hackmann.– HHH Publ, 2007, 411 p.

[22] Jaeschke G. *On Strong Pseudoprimes to Several Bases* / G/ Jaeschke. - Mathematics of Computation, v.61, 1993, p.915-926

[23] Joux A. *A one round protocol for tripartie Diffie-Hellman.* / A. Joux.– Algorithmic Number Theory: 4-th International Symposium, ANT–IV, Lecture Notes in Computer Science, v.1838(2000), Springer–Verlag, p. 385–393.

[24] Joux A. *The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems.* Proceedings of the 5th International Symposium on Algorithmic Number Theory, Springer-Verlag London, 2002, p.20–32

[25] Lenstra H.W. *Factoring integers with elliptic curves* / H.W. Lenstra.– Ann.Math. v.126 (1987), p. 649–674.

[26] Lenstra A. *The Development of the Number Field Sieve* / A. Lenstra and H. Lenstra (eds.).– Lect.Not.in Math.**1554**, Springer–Verlag, Berlin, 1993, 139 p.

[27] Longa P.*Fast Point Arithmetic for Elliptic Curve Cryptography*/ P. Longa.– Presentation at CliCC, University of Ottawa, Ottawa, Canada, 2006.

[28] Longa P. *ECC Point Arithmetic Formulae (EPAF): Jacobian coordinates*/ P. Longa, C. Gebotus. In Proc. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2010), 2010.

[29] Menezes A. *Reducing Elliptic Curve Logarithms to a Finite Field* / A. Menezes, T. Okamoto, S. Vanstone.– IEEE Trans. Info. Theory, v.39, 1993, p. 1639–1646.

[30] Menezes A. *Elliptic Curve Public Key Cryptosystems* / A. Menezes.– 1993, 144 p.

[31] Montgomery P.L. *Speeding the Pollard and Elliptic Curve Methods of Factorization.*/P.L. Montgomery.– Mathematics of Computation, v.48, iss.177, 1987, p.234–264.

[32] Montgomery P.L. *An FFT-extension of the Elliptic Curve Method of Factirization* / P.L. Montgomery.– Doctoral Dissertation, 1992, Univ.Calif. USA, 118 p.

[33] Pollard J.M. *Theorems on factorization and primality testing* / J.M. Pollard. – Proc.Cambridge Phil.Society. 1974, v.76, p. 521-578.

[34] Pomerance C. *Smooth Numbers and the Quadratic Sieve* / C. Pomerance. – MSRI publications, **v.44** – 2008, p. 69–82.

[35] Shoup V. *A Computational Introduction to Number Theory and Algebra*/ V. Shoup. – Cambridge University Press, Sec.Edition, 2005, 600 p. http://shoup.net/ntb/

[36] SciLab *Free open source software for numerical computation* /http://www.scilab.org

[37] Sheng Lin, Yong-Bin Kim, Lombardi F. CNTFET-Based Design of Ternary Logic Gates and Arithmetic Circuits //IEEE Transactions on Nanotechnology, March. 2011, vol.10, №2, С. 217-225

[38] Venturi D. *Lecture Notes on Algorithmic Number Theory.*/ D. Venturi. – Springer-Verlag, New-York, Berlin, 2009, 217 p.

[39] Washington L. *Elliptic Curves Number Theory and Cryptography* /L. Washington. – Series Discrete Mathematics and Its Applications, Chapman & Hall/CRC,second ed. 2008, 524 p.

[40] Аграновский А.В. *Практическая криптография: алгоритмы и их программирование* / А.В. Аграновский, Р.А. Хади.– М.: Солон-Пресс, 2009, 256 с.

[41] Айерленд К. *Классическое введение в современную теорию чисел.* / К. Айерленд, М. Роузен. – М.: Мир, 1987, 428 с.

[42] Акритас А. *Основы компьютерной алгебры и приложениями.* / А. Акритас. – М.: Мир, 1994, 544 с.

[43] *Алгебраическая теория автоматов, языков и полугрупп*/ Под редакцией М.А. Арбиба — М.: Статистика — 1975 — 334 с.

[44] Беллман Р.*Введение в теорию матриц*/Р.Беллман – М.: Наука, – 1969. – 367 с.

[45] Берленкэмп Э.*Алгебраическая теория кодирования.* /Э.Берленкэмп. – М.: Мир, – 1971. – 478 с.

[46] Богопольский О.В. *Алгоритмическая теория чисел и элементы криптографии.* / О.В. Богопольский.– Спецкурс для студентов НГУ, Новосибирск, 2005, 35 с. / http://math.nsc.ru/ bogopolski/Articles/SpezkNumber.pdf

[47] Болотов А.А. *Элементарное введение в эллиптическую криптографию: протоколы криптографии на эллиптических кривых.* / А.А. Болотов, С.Б. Гашков, А.Б. Фролов. – М.:КомКнига, 2004, 280 с.

[48] Болотов А.А. *Алгоритмические основы эллиптической криптографии.* / А.А. Болотов, С.Б. Гашков, А.Б. Фролов, Часовских А.А.. – М.:РГСУ, 2004, 499 с.

[49] Боревич З.И. *Теория чисел.* / З.И. Боревич, И.Р. Шафаревич. – 3-е издание, М.: Наука, 1985, 504 с.

[50] Ван дер Варден Б.Л. *Алгебра.*/ Б.Л.ван дер Варден. – изд.2, М.: Наука, 1979, 623 с.

[51] Василенко О.Н. *Теоретико-числовые алгоритмы в криптографии*/ О.Н. Василенко. – МЦНМО, 2003, 326 с.

[52] Вельщенбах М. *Криптография на C и C++ в действии: учебное пособие* / М. Вельщенбах. – М.: Триумф, 2008, 464 с.

[53] Гатмахер Ф.Р. *Теория матриц.*/ Ф.Р. Гатмахер – М.: Наука. – 1967. – 575 с.

[54] Гилл А *Линейные последовательностные машины*/А. Гилл — М.:Наука — 1974 — 287 с.

[55] Ермаков С.М. *Курс статистического моделирования*/С.М. Ермаков, Г.А. Михайлов — М.:Наука — 1976 — 318 с.

[56] Захаров В.М. *Вычисления в конечных полях: уч.–метод. пособие* / В.М. Захаров, Б.Ф. Эминов. – Казань: КГТУ им. А.Н.Туполева, 2010, 132 с.

[57] Ишмухаметов Ш.Т. *Методы факторизации натуральных чисел*/ Ш.Т.Ишмухаметов. – Казань, 2012, 189 с.

[58] Кнут Д. *Искусство программирования для ЭВМ. т.2 Получисленные алгоритмы*/Д. Кнут — М.:Мир — 1977 — 724 с.

[59] Коблиц Н. *Курс теории чисел и криптографии* / Н. Коблиц. – М.: ТВП, 2001, 260 с.

[60] Корешков Н.А. *Теория чисел.*/Н.А. Корешков. – Уч.-мет. пособие, Казань, КФУ, 2010, 35 с.

[61] Кормен Т. *Алгоритмы: построение и анализ* /Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 1999.

[62] Крамер Г. *Математические методы статистики* /Г. Крамер — М.:Мир — 1975 — 648 с.

[63] Кузнецов В.М. *Марковская модель цифрового стохастического генератора*/В.М. Кузнецов, В.М. Песошин, Е.Л. Столов — Автоматика и Телемеханика — 2008 – № 9, с. 62-68

[64] Кузнецов В.М. *Марковская модель цифрового стохастического генератора*/В.М. Кузнецов, В.М. Песошин, Е.Л. Столов *Стабильные состояния*

асинхронного генератора. – Ученые записки казанского государственного университета, –2010 – Том. 152, Книга 1. Серия - Физико-математические науки, с.174-180

[65] Кунегин С.В. *Системы передачи информации.* С.В. Кунегин.- Курс лекций. М., в/ч 33965, 1997, 317 с. http://kunegin.com/ref/sod_lec.htm

[66] Лазарева С.В. *Математические основы криптологии: тесты простоты и факторизация.* / С.В. Лазарева, А.А. Овчинников. Учебное пособие, Санкт-Петербург, СПбГУАП, 2006, 65 с.

[67] Латыпов Р.Х.*Периодические последовательности, порождаемые регистр сдвига с нелинейными обратными связями*/ Р.Х. Латыпов. Изв. Вузов. Математика, №5, 1989, С.5-13

[68] Лидл Р. *Конечные поля*/Р. Лидл,Г. Нидеррайтер.– Т. 1, 2. М.: Мир, 1988, 428 с.

[69] Мак-Вильямс Ф.Дж., Слоэн Н.Дж.А. *Теория кодов, исправляющих ошибки..*– Ф.Дж. Мак-Вильямс, Н.Дж.А. Слоэн.– М. - Связь, 1979, 372 с.

[70] Максимов В.М.*О сходимости неоднородных дважды стохастических цепей Маркова*/В.М. Максимов —ТВП, т.15, вып. 4 — 1970 — с.622-636

[71] Маркус М., Минк Х.*Обзор по теории матриц и матричных неравенств*/М. Маркус , Х. Минк – М.: Наука, – 1972. – 232 с.

[72] Молдовян Н.А. *Криптография. От примитивов к синтезу алгоритмов* / Н.А.Молдовян,

А.А. Молдовян,М.А. Еремеев. – БХВ-Петербург, 2004, 446 с.

[73] Морелос-Сарагоса Р. *Искусство помехоустойчивого кодирования - методы, алгоритмы, применение./* Р. Морелос-Сарагоса.– Москва, Техносфера, 2006, 312 с.

[74] Нестеренко Ю.В. *Теория чисел/* Ю.В. Нестеренко. – Москва, Изд.Центр Академия, 2008, 273 с.

[75] Отнес Р.*Прикладной анализ временных рядов* /Р. Отнес, Л. Эноксон — М.:Мир — 1982 — 428 с.

[76] Песошин В.А. *Цифровые генераторы случайных сигналов для защиты информационных средств телекоммуникации* /В.А Песошин, В.М. Кузнецов, Н.Н. Сергеев. – Вопросы радиоэлектроники. Серия ЭВТ. Вып.4 —1993 — с.95-113.

[77] Песошин В.А. *Генераторы псевдослучайных и сдучайных чисел на регистрах сдвига* /В.А. Песошин, В.М. Кузнецов — Казань: КГТУ им. А.Н.Туполева — 2007 – 296 с.

[78] А.Г. Ростовцев, Е.Б. Маховенко. Теоретическая криптография, Профессионал, Санкт-Петербург, 2005, 479 с.

[79] Сизый С.В. *Лекции по теории чисел: учебное пособие для математических специальностей* / С.В. Сизый.– Екатеринбург, УрГУ, 1999, 136 с.

[80] Столов Е.Л. *Методы компактного тестирования цифровых устройств* / Е.Л. Столов — Казань: Казанский государственный университет — 1993—115 с.

[81] Столов Е.Л. *Генератор случайных чисел составленный из нелинейных асинхронных элементов*/ Е.Л. Столов. Исследования по прикладной математике, вып. 26, Институт информатики КГУ — Казань, — 2006 – с.101-106.

[82] Столов Е.Л. *Математическая модель генератора случайных чисел на основе трехзначной логики*/Е.Л. Столов – Прикладная дискретная математика – 2012 – № 2, с 43-49

[83] Хинчин А.Я. *Работы по математической теории массового обслуживания.*/А.Я. Хинчин – М.: Фзматгиз. – 1963. - 235 с.

[84] Чандрасекхаран К. *Введение в аналитическую теорию чисел*/ К. Чандрасекхаран.–М.– Мир, 1974, 187 с.

[85] Черемушкин А.В. *Лекции по арифметическим функциям в криптографии* / А.В. Черемушкин.– М.: МЦНМО, 2002, 103 с.

[86]) Шаньгин Ф.Ф. *Защита компьютерной информации: эффективные методы и средства* /Ф.Ф. Шаньгин.– М.:ДМК, 2008, 542 с.

[87]) Шеннон К. *Работы по теории информации и кибернетике.*/К. Шеннон.– М.: Изд-во иностранной литературы, 1963. — 830 с.