Цифровые технологии

Ю.В. Нестеренко

Полугодовой спецкурс для студентов 1-2 курсов мехмата МГУ
Весенний семестр 2019-2020 учебного года

Оглавление

Глава 1.

Введение

Основное содержание курса составляют так называемые цифровые технологии, возникшие сравнительно недавно, в связи с образовавшимися потребностями, и переживающие сейчас период становления, они широко обсуждаются, некоторые уже используются, а другие в ближайшее время получат широкое распространение. Деятельность в этой области направлена на создание информации и механизмов её обработки, безопасного хранения и передачи, а в роли инструментов выступают математические алгоритмы, основанные на методах теории чисел, алгебры, дискретной математики. Так что сначала мы освоимся с такими понятиями, как вычислительная сложность алгоритма, познакомимся с конкретными алгебраическими и теоретико-числовыми алгоритмами, имеющими различную сложность, узнаем, что такое хэш-функция и эллиптическая кривая, состоящая из конечного числа точек, научимся строить очень большие простые числа, вычислять дискретные логарифмы и делать многое другое. Всё это составляет основу криптографической науки - важной части цифровых технологий.

Криптография, как область научной деятельности, стала широко известной примерно 40 лет назад, см. [10]. Это было вызвано необходимостью обмена большими массивами конфиденциальной информации (не только государственной и военной, но и банковской, экономической, медицинской, юридической и т.п.), а также возможностью такого обмена, в связи с появлением доступных и эффективных компьютерных средств обработки этой информации. Любая информация может быть закодирована последовательностью чисел. Например, букве "а"можно сопоставить число 1, букве "б" — число 2 и так далее, букве "я"— число 32. Можно сопоставить числа пробелам, точке, другим знакам препинания. После этого процессы зашифрования и расшифрования информации представятся как некоторые алгоритмы, перерабатывающие одни массивы целых чисел в другие.

Стойкость криптографических алгоритмов напрямую зависит от невозможности найти быстрые алгоритмы для решения некоторых задач, другими словами, от того, что некоторые математические задачи сложны в вычислительном отношении.

Сложность алгоритмов теории чисел обычно принято измерять количеством арифметических операций (сложений, вычитаний, умножений и делений с остатком), необходимых для выполнения всех действий, предписанных алгоритмом. Впрочем, это определение не учитывает величины чисел, участвующих в вычислениях. Ясно, что перемножить два стозначных числа значительно сложнее, чем два однозначных, хотя и в том, и в другом случае выполняется лишь одна арифметическая операция. Поэтому иногда учитывают еще и величину чисел, сводя дело к так называемым битовым операциям, т.е. оценивая количество необходимых операций с цифрами 0 и 1 в двоичной записи чисел. Это зависит от рассматриваемой задачи, от целей автора и т.д.

На первый взгляд кажется странным, что операции умножения и деления приравниваются по сложности к операциям сложения и вычитания. Житейский опыт подсказывает, что умножать числа значительно сложнее, чем складывать их. В действительности же, вычисления можно организовать так, что на умножение или деление больших чисел понадобится не намного больше битовых операций, чем на сложение, см. [1], гл.8, теорема 8.5. Примерно таким же количеством битовых операций можно обойтись при выполнении деления с остатком двух двоичных чисел. Говоря далее о сложности алгоритмов, мы будем иметь в виду количество арифметических операций, необходимых для их выполнения. Если же будет необходима битовая сложность алгоритма — это будет специально оговариваться.

Рассмотрим далее три примера, показывающие, как используется сложность алгоритмов в некоторых криптографических применениях. Далее будут появляться как общеизвестные, так и известные только тем, кто их определил — секретные величины. В первом случае эти величины будут обозначаться строчными (маленькими) буквами, например "a", во втором же случае прописными (крупными) буквами, скажем "A". Кроме того, условимся, что обмен информацией между участниками алгоритма или протокола[1] идёт только через Интернет, причём все сообщения пересылаются в зашифрованном виде. Участников обмена информацией, в ближайших примерах их будет двое, принято считать одушевлёнными лицами и присваивать им имена. Обычно легальных действующих

[1] так называется последовательность действий, в которой участвуют двое или более лиц

лиц именуют Алиса и Боб [2], злоумышленник получает имя Ева и так далее. Заметим, что, несмотря на происхождение, действующие лица могут быть и компьютерами, которые иногда для удобства описания наделяются чувствами и желаниями.

Пример 1. Алгоритм шифрования RSA.

В 1977г. был предложен действующий до настоящего времени и наиболее известный алгоритм шифрования RSA. Это название образовано первыми буквами фамилий авторов работы [13], в которой он был описан.

Рассмотрим следующий сюжет. Допустим, что Боб хотел бы получать письма от Алисы, но так, чтобы никто не смог узнать их содержание. Для этого он выбирает два простых числа p, q и вычисляет их произведение $N = pq$. Затем Боб случайным образом выбирает два натуральных числа E, d, меньших $\varphi(N) = (p-1)(q-1)$, [3] и удовлетворяющих сравнению

$$E \cdot d \equiv 1 \pmod{\varphi(N)}. \tag{1.1}$$

Для этого достаточно выбрать число E, $1 < E < \varphi(N)$, взаимно простое с $\varphi(N)$ и обозначить буквой d наименьшее положительное решение сравнения (1.1). Открытым ключом Боба будет пара чисел (N, E). Боб может, например, переслать эту пару чисел Алисе по открытой почте. Секретный ключ Боба — это число d. Числа p и q также должно держать в секрете, но рассматривать всю тройку (p, q, d) в качестве секретного ключа необязательно, потому что числа p, q используются только на стадии генерации чисел N, E, d. Поэтому числа p, q можно уничтожить. Отметим, что любые целые E и d,удовлетворяющие (1.1), взаимно просты с $\varphi(N)$. Итак, для зашифрования сообщений используется открытый ключ, а для расшифрования — другой ключ, который известен только Бобу.

Допустим, что секретное письмо x Алисы удовлетворяет условиям

$$1 < x < N, \qquad (x, N) = 1.$$

Заметим, что последнее условие означает взаимную простоту чисел x и N.

Для того, чтобы зашифровать сообщение x, Алисе достаточно вычислить

$$y \equiv x^E \pmod{N}, \qquad 0 < y < N. \tag{1.2}$$

[2] Их роль подобна латинским буквам a и b в описаниях алгебраических преобразований.

[3] Здесь функция $\varphi(m)$ есть функция Эйлера. Её значение равно количеству целых чисел на отрезке от 1 до m, взаимно простых с m.

Число y условиями (1.2) определяется однозначно. Оно представляет собой письмо Алисы в зашифрованном виде и может быть переслано Бобу через Интернет.

Согласно теореме Эйлера справедливо сравнение

$$y^d \equiv x^{Ed} \equiv x \pmod{N}. \tag{1.3}$$

Последнее сравнение выполняется в силу (1.1).

Боб, получив зашифрованное письмо Алисы y, находит оригинал письма

$$x \equiv y^d \pmod{N}, \qquad 0 < x < N. \tag{1.4}$$

Сравнение (1.3) доказывает корректность алгоритма RSA. Если письмо Алисы по длине больше N, его можно разбить на блоки меньшей длины, чем N, и шифровать каждый блок отдельно. Если же письмо не взаимно просто с N, то к нему можно дописать время отправления или небольшой набор из нулей и единиц, чтобы обеспечить это условие.

Боб может вывесить открытый ключ (N, E) на своём сайте, или поместить его в какой-нибудь справочник вроде старой телефонной книги. Это, как легко видеть, позволит большому числу людей писать ему секретные письма. Если каждый из некоторой группы людей выработает указанным способом свои открытый и секретный ключи и сделает свой открытый ключ доступным всем участникам группы, то члены этой группы смогут обмениваться конфиденциальной информацией через Интернет. Причём понять содержание письма сможет только тот, кому оно адресовано.

Возведение в степень по фиксированному модулю, решение линейных сравнений с одной переменной могут выполняться сравнительно быстро. Об этом мы поговорим немного позже. Сложность расшифрования письма с помощью алгоритма (1.4) зависит от сложности вычисления секретного ключа d. Последний может быть найден очень легко, если известно значение функции Эйлера $\varphi(N) = (p-1)(q-1)$. Но для этого нужно найти числа p, q, т.е. разложить число N на простые сомножители. Эта фундаментальная задача

Разложить заданное число N на нетривиальные множители

очень сложна. Для чисел N размером в 1024 битов при использовании самых мощных в настоящее время компьютеров требуются годы для её решения. А для сохранения очень ценных секретов рекомендуется использовать числа N, записываемые примерно 2048 битами.

Еще Ферма предложил нетривиальный алгоритм разложения чисел на множители. Различные его усовершенствования использовались Эйлером, Гауссом, Лежандром, Чебышевым и другими классиками теории чисел. Современные алгоритмы используют вычисления в полях алгебраических чисел, эллиптические кривые, разнообразные математические конструкции.

Пример 2. Алгоритм шифрования Эль-Гамаля.

Ещё одна очень сложная в вычислительном отношении задача носит название дискретного логарифмирования.

Для заданных простого числа P и целых чисел A, B, не делящихся на P, требуется решить сравнение

$$A^x \equiv B \pmod{P}. \tag{1.5}$$

На большой сложности этой задачи основан алгоритм шифрования Эль-Гамаля.

Как известно, для любого простого числа P ненулевые элементы поля вычетов $\mathbb{F}_P = \mathbb{Z}/P\mathbb{Z}$ образуют циклическую группу $\mathbb{F}_P^*$. Обозначим буквой G образующую этой группы. Порядок G по модулю P равен $P-1$. После выбора открытых параметров P, G Боб, который хотел бы получать сообщения от Алисы, определяет свои открытый и секретный ключи. Секретным ключом может быть любое натуральное число $d, 0 < d < P-1$, Боб выбирает его случайно на интервале от 1 до $P-1$, а открытый ключ определяется условиями

$$E \equiv G^d \pmod{P}, \qquad 1 < E < P-1.$$

Если Алиса хочет отправить Бобу некоторое письмо, она должна разбить его на блоки по величине меньшие, чем P и шифровать каждый блок отдельно. Пусть $m, 1 < m < P$ — целое число, представляющее какой-либо из блоков. Для шифрования Алиса поступает следующим образом:

- выбирает случайно какое-нибудь целое число $k, 1 < k < P$,
- вычисляет $C_1 \equiv G^k \pmod{P}$,
- находит $C_2 \equiv m \cdot E^k \pmod{P}$,

Пара чисел $C = (C_1, C_2)$ есть представление блока m из письма Алисы в зашифрованном виде. Шифртекст C пересылается Бобу в открытом виде по сети Интернет.

При каждом шифровании применяется свой ключ k — кратковременный ключ. Поэтому, шифруя одно сообщение дважды, можно получить разные шифртексты.

Чтобы расшифровать полученное сообщение $C = (C_1, C_2)$, Боб вычисляет $C_2 \cdot C_1^{P-1-d} \pmod{P}$. Это число и будет сообщением Алисы. Действительно, пользуясь определениями чисел C_1 и C_2 находим

$$C_2 \cdot C_1^{P-1-d} \equiv m \cdot G^{kd+k(P-1-d)} \equiv m \left(G^{P-1}\right)^k \equiv m \pmod{P}.$$

Последнее сравнение основано на малой теореме Ферма, согласно которой $G^{P-1} \equiv 1 \pmod{P}$.

Если злоумышленник перехватит шифртекст C, то для восстановления оригинала посланного сообщения ему нужно будет решать задачу дискретного логарифмирования $C_1 \equiv G^k \pmod{P}$, что, как указывалось выше, требует очень много времени. Он не сможет определить число k и затем из сравнения для C_2 найти искомый блок m.

Алгоритм Эль-Гамаля можно организовать так, чтобы вычисления проводить не во всей мультипликативной группе поля $\mathbb{F}_P$, а в её подгруппе простого порядка. Это усложнит задачу дискретного логарифмирования. Пусть P, Q большие простые числа, такие, что $P - 1$ делится на Q. Выберем первообразный корень g по модулю P, и определим число G условиями

$$G \equiv g^{\frac{P-1}{Q}}.$$

Тогда $G \neq 1$ и циклическая подгруппа $(G) \subset \mathbb{F}_P^*$ имеет простой порядок Q. Далее все вычисления открытого ключа E, секретного ключа d, шифртекста (C_1, C_2), а также расшифрование выполняются по тем же формулам, как и ранее.

Заметим, что построение секретного и открытого ключей в системе Эль-Гамаля требует намного меньше времени, чем в RSA. Среди минусов нужно отметить, что длина шифртекста примерно вдвое превосходит длину шифруемого болока.

Глава 2.

Алгоритмы и сложность

Лекция 2.

В этой главе мы обсудим ряд полезных на практике алгоритмов и докажем некоторые оценки их сложности.

2.1. Алгоритм Евклида и теорема Ламе.

Рассмотрим следующую задачу. Пусть a, b — натуральные числа. Требуется найти (a, b) — наибольший общий делитель a, b. Задача эта решается достаточно быстро с помощью хорошо известного алгоритма Евклида без разложения чисел на множители. В основе его лежит равенство $(a, b) = (b, r)$, где r — остаток от деления числа a на b.

Алгоритм 1. Даны: *Натуральные числа a и b; $b < a$.*
Найти: *Наибольший общий делитель (a, b).*

1. Вычислить r — остаток от деления a на b, т.е. найти целое r, удовлетворяющее условиям $a = bq + r,\ 0 \leqslant r < b$.
2. Если $r = 0$, то $(a, b) = b$, СТОП.
3. Если $r \neq 0$, то заменить пару $\{a, b\}$ парой $\{b, r\}$ и перейти в пункт 1 алгоритма.

Пусть r — остаток от деления числа a на b, т.е. $a = bq+r$, $0 \leqslant r < b$. По свойствам делимости каждый общий делитель чисел b и r делит число $bq + r = a$ и, значит, принадлежит множеству общих делителей чисел b и a. Точно так же, каждый общий делитель чисел a и b делит число $a - bq = r$, так что принадлежит множеству общих делителей чисел b и r. Отсюда следует совпадение наибольших общих делителей пар чисел a, b и b, r, т.е. равенство

$$(a, b) = (b, r). \tag{2.1}$$

Это равенство позволяет при нахождении наибольшего общего делителя заменять пару чисел a, b другой парой b, r. Заметим, что $r < b$, т.е. одно из двух чисел, участвующих в алгоритме уменьшилось. Повторяя несколько раз деление с остатком и заменяя каждый раз пару целых чисел новой мы будем каждый раз уменьшать одно из двух чисел (большее), участвующих в работе алгоритма. Ясно, что в какой-то момент одно из чисел станет равным 0 и наибольший общий делитель будет равен второму из чисел.

Рассмотрим алгоритм немного подробнее. Положим $r_0 = a$, $r_1 = b$ и обозначим через $r_2, \ldots, r_n$ — последующие делители в алгоритме Евклида. Тогда получаются следующие равенства

$$\begin{aligned}
a = r_0 &= bq_1 + r_2, & 0 \leqslant r_2 < b,\\
b = r_1 &= r_2q_2 + r_3, & 0 \leqslant r_3 < r_2,\\
r_2 &= r_3q_3 + r_4, & 0 \leqslant r_4 < r_3,\\
&\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\\
r_{n-2} &= r_{n-1}q_{n-1} + r_n, & 0 \leqslant r_n < r_{n-1},\\
r_{n-1} &= r_nq_n.
\end{aligned} \tag{2.2}$$

Алгоритм останавливается, когда деление произойдет без остатка. В приведенном выше тексте последний остаток $r_{n+1} = 0$. В соответствии с равенством (2.1) находим

$$(a, b) = (b, r_2) = (r_2, r_3) = (r_3, r_4) = \cdots = (r_{n-1}, r_n) = (r_n, 0) = r_n.$$

Таким образом, наибольший общий делитель равен последнему делителю (он же последний ненулевой остаток) в алгоритме Евклида.

Следующее утверждение было доказано в 1760г. Л. Эйлером и носит его имя.

Теорема 1 (Теорема Эйлера)**.** *Для каждого целого числа a, взаимно простого с модулем m, выполняется сравнение*

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

Рассмотрим частный случай теоремы Эйлера, в котором $m = p$ есть простое число. Тогда $\varphi(p) = p-1$ и получается утверждение, называемое малой теоремой Ферма.

Теорема 2 (Малая теорема Ферма)**.** *Если целое число a не делится на простое число p, то*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Из этой теоремы следует, что при любом целом a число $a^p - a = a(a^{p-1} - 1)$ делится на p.

Рассмотрим пример вычисления наибольшего общего делителя двух чисел, а именно, вычислим $(89, 55)$. Пользуясь алгоритмом Евклида, находим

$$\begin{aligned} 89 &= 55 \cdot 1 + 34, & 55 &= 34 \cdot 1 + 21, & 34 &= 21 \cdot 1 + 13, \\ 21 &= 13 \cdot 1 + 8, & 13 &= 8 \cdot 1 + 5, & 8 &= 5 \cdot 1 + 3, \\ 5 &= 3 \cdot 1 + 2, & 3 &= 2 \cdot 1 + 1, & 2 &= 1 \cdot 2 + 0. \end{aligned}$$

Итак, $(89, 55) = 1$, и для вычисления наибольшего общего делителя понадобилось 9 делений с остатком. Числа 55 и 89 есть числа Фибоначчи с номерами 10 и 11. Напомним, что последовательность чисел Фибоначчи определяется рекуррентным уравнением $F_{i+1} = F_i + F_{i-1}$ и начальными данными $F_0 = 0$, $F_1 = 1$. Приведённое выше вычисление показывает, и это легко доказать по индукции, что для вычисления наибольшего общего делителя чисел (F_n, F_{n+1}) нужно не менее $n - 1$ делений с остатком.

Доказываемая далее теорема Ламе, даёт верхнюю оценку для количества делений с остатком в алгоритме Евклида вычисления наибольшего общего делителя двух чисел, т.е. даёт верхнюю оценку сложности алгоритма. Заметим, что эта оценка может подходить очень близко к возможной нижней оценке, доставляемой примером с числами Фибоначчи. Таким образом оценка теоремы Ламе очень точна.

Теорема 3 (Ламе, 1845)**.** *Пусть $a > b$ — натуральные числа. При вычислении (a, b) с помощью алгоритма Евклида будет выполнено не более $5m$ операций деления с остатком, где m есть количество цифр в десятичной записи числа b.*

Доказательство. Положим $r_0 = a$ и обозначим $r_1, r_2, \ldots, r_n$ — последовательность делителей в алгоритме Евклида. Тогда $r_1 = b$,

$$r_{i-1} = q_i r_i + r_{i+1}, \quad 0 \leqslant r_{i+1} < r_i, \ q_i \in \mathbb{N}, \quad i = 1, 2, \ldots, n-1,$$

$r_n = (a, b)$.

Докажем справедливость следующих неравенств

$$r_i \geqslant \lambda^{n-i}, \qquad i = 1, 2, \ldots, n, \tag{2.3}$$

где $\lambda = \frac{1+\sqrt{5}}{2} = 1,61...$ есть корень квадратного уравнения $\lambda^2 - \lambda - 1 = 0$. Для этого воспользуемся индукцией по индексу i, двигаясь в обратном направлении от n к 1. При $i = n$ имеем $r_n \geqslant 1 = \lambda^0$. При $i = n-1$ находим $r_{n-1} \geqslant r_n + 1 \geqslant 2 > \lambda$, так что неравенство (2.3) опять справедливо.

Предположим теперь, что $k < n$ и неравенство (2.3) выполняется при всех $i \geqslant k$. Имеем следующую цепочку равенств и неравенств

$$r_{k-1} = q_k r_k + r_{k+1} \geqslant r_k + r_{k+1} \geqslant \lambda^{n-k} + \lambda^{n-k-1} = \lambda^{n-k-1}(\lambda + 1) = \lambda^{n-k+1}.$$

Таким образом, неравенство (2.3) выполняется и при $i = k - 1$. Это доказывает справедливость (2.3) при всех $i = 1, 2, \ldots, n$.

Поскольку десятичная запись b содержит m знаков, то $10^m > b$, и

$$10^m > b = r_1 \geqslant \lambda^{n-1}.$$

Из этих неравенств следует, что

$$m > (n-1)\log_{10}\lambda > (n-1)/5.$$

Последнее неравенство выполняется, поскольку $\lambda > 10^{1/5} = 1,58....$ Таким образом, $n < 5m + 1$, что завершает доказательство теоремы. □

Из теоремы 3 следует, что алгоритм Евклида работает достаточно быстро.

Алгоритмы, в которых необходимое количество арифметических операций оценивается фиксированной степенью длины записи его входных данных, называются *полиномиальными*. Таким является алгоритм Евклида, а также алгоритмы решения других задач (например, линейных уравнений и сравнений), связанные с алгоритмом Евклида.

Конец второй лекции.

Лекция 3.

2.2. Символы Лежандра и Якоби

В этом параграфе мы рассмотрим вопрос о том, как узнать, разрешимо или нет по простому модулю $p > 2$ квадратичное сравнение. Хорошо известно, что сравнение

$$ax^2 + bx + c \equiv 0 \pmod{p}, \qquad a, b, c \in \mathbb{Z},\ p \nmid a,$$

сводится выделением полного квадрата к сравнению

$$x^2 \equiv d \pmod{p}, \qquad d \in \mathbb{Z}. \tag{2.4}$$

Ответ находится тривиально в случае $p \mid d$, решениями являются числа $x \equiv 0 \pmod{p}$.

Если $p \nmid d$, вопрос о разрешимости (2.4) также может быть решен достаточно быстро. Мы изложим ниже соответствующую теорию, отсылая за доказательствами к [?].

Определение 1. Символ Лежандра $\left(\frac{d}{p}\right)$ *есть функция от двух аргументов:* d *есть целое число, а* p *— простое нечетное. Значение символа Лежандра равно 1, если сравнение* (2.4) *разрешимо, оно равно* -1*, если это сравнение не имеет решений и оно равно 0, если* $p \mid d$.

Определенная так функция обладает рядом свойств, позволяющих вычислять её значения и тем самым получать ответ на вопрос о разрешимости сравнения (2.4). При этом иногда требуется проверять простоту чисел, а иногда раскладывать числа d на простые множители. В настоящее время это очень трудоемкая задача. Ниже мы опишем более совершенный способ вычисления символа Лежандра. Соответствующий алгоритм имеет полиномиальную сложность и не использует ни проверки чисел на простоту, ни разложения на простые множители. В основе его лежит возможность продолжить функцию $\left(\frac{d}{p}\right)$ на множество всех пар взаимно простых целых чисел D, P, где P — произвольное нечетное число. Продолженная таким образом функция носит название *символ Якоби.*

Определение 2. *Пусть* $P = p_1 \cdots p_r$*, где* p_j *— нечетные простые числа, и* $D \in \mathbb{Z}$, $(D, P) = 1$. *Символ Якоби* $\left(\frac{D}{P}\right)$ *определяется равенством*

$$\left(\frac{D}{P}\right) = \left(\frac{D}{p_1}\right) \cdots \left(\frac{D}{p_r}\right),$$

где $\left(\frac{D}{p_j}\right)$ *— значения символа Лежандра.*

Если P — простое число, то символы Лежандра и Якоби со знаменателем P совпадают.

Вообще говоря, значение символа Якоби не связано с разрешимостью сравнений.

Пример. Легко проверить, что сравнение $x^2 \equiv 2 \pmod{15}$ не имеет решений. Тем не менее символ Якоби в этом случае равен

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

Следующие свойства подобны соответствующим свойствам символа Лежандра.

Свойства символа Якоби

2. *Если* $a \equiv b \pmod{P}$, *то* $\left(\frac{a}{P}\right) = \left(\frac{b}{P}\right)$.
3. $\left(\frac{1}{P}\right) = 1; \quad \left(\frac{-1}{P}\right) = (-1)^{\frac{P-1}{2}}; \quad \left(\frac{2}{P}\right) = (-1)^{\frac{P^2-1}{8}}$.
4. $\left(\frac{ab}{P}\right) = \left(\frac{a}{P}\right) \cdot \left(\frac{b}{P}\right)$.
5. *Если* P *и* Q *— положительные нечетные взаимно простые числа, то*
$$\left(\frac{P}{Q}\right) \cdot \left(\frac{Q}{P}\right) = (-1)^{\frac{P-1}{2}\frac{Q-1}{2}}.$$

Покажем теперь, как с помощью этих свойств можно вычислить значение символа Якоби. При этом не используется разложение на простые множители, а лишь выделяется максимальная степень числа 2, входящая в разложение числителя. Рассмотрим сначала следующий пример.

Пример. Выяснить, разрешимо ли сравнение

$$x^2 \equiv 753 \pmod{811}$$

Отметим, что число 811 простое, так что для решения задачи достаточно вычислить соответствующий символ Якоби:

$$\left(\frac{753}{811}\right) = \left(\frac{811}{753}\right) = \left(\frac{58}{753}\right) = \left(\frac{29}{753}\right) = \left(\frac{753}{29}\right) = \left(\frac{-1}{29}\right) = 1.$$

При этом вычислении использовались следующие соотношения $753 \equiv 1 \pmod 8$, $811 \equiv 58 \pmod{753}$, $753 \equiv -1 \pmod{29}$, $29 \equiv 1 \pmod 4$. Итак, данное в примере сравнение разрешимо.

Способ вычисления символа Якоби, использовавшийся в рассмотренном примере может быть оформлен в виде следующего алгоритма.

Алгоритм 2. Данные: *Целые взаимно простые числа* Q *и* $P > 2$, P *нечетно.*
Найти: *Значение символа Якоби* $\left(\frac{Q}{P}\right)$.

1. Положить $s = 0,\ u = Q,\ v = P.$
2. Найти r *— наименьший положительный остаток от деления числа* u *на* v, *т.е. целое число, удовлетворяющее условиям*

$$u = vq + r, \qquad 0 < r < v;$$

вычислить целое $k \geqslant 0$ *и нечетное* t, *для которых* $r = 2^k t$; *положить*

$$s \equiv s + k \cdot \frac{v^2 - 1}{8} + \frac{(t-1)(v-1)}{4} \pmod 2.$$

3. Если $t = 1$, *положить*

$$\left(\frac{Q}{P}\right) = (-1)^s.$$

СТОП.
4. Если $t \geqslant 3$, *положить* $u = v,\ v = t$, *перейти в пункт 2.*

Докажем, что приведенный алгоритм действительно вычисляет символ Якоби и получим оценку его сложности.

Теорема 4. *Приведенный алгоритм вычисляет символ Якоби* $\left(\frac{Q}{P}\right)$ *за* $O(m)$ *арифметических операций, где* m *есть количество цифр в десятичной записи меньшего из чисел* P, Q.

Доказательство. В процессе работы алгоритма число v убывает. Это значит, что алгоритм не может работать бесконечно долго и, следовательно, завершит свою работу.

Обозначим буквой n количество проходов алгоритма через пункт 2. Пусть также $s_j,\ u_j,\ v_j$ — значения параметров $s,\ u,\ v$ перед j-м входом в пункт 2. Тогда $s_1 = 0, u_1 = Q, v_1 = P$. Докажем индукцией по j справедливость равенств

$$\left(\frac{Q}{P}\right) = (-1)^{s_j} \cdot \left(\frac{u_j}{v_j}\right), \qquad j = 1, \ldots, n. \tag{2.5}$$

При $j = 1$ это равенство, очевидно, выполняется. Пусть $j < n$ и (2.5) справедливо. В процессе выполнения пункта 2 алгоритма в j-й раз будут найдены числа $q,\ k,\ t$, для которых $u_j = qv_j + 2^k t$. Тогда

$$\left(\frac{u_j}{v_j}\right) = \left(\frac{2^k t}{v_j}\right) = \left(\frac{2}{v_j}\right)^k \cdot \left(\frac{t}{v_j}\right) = (-1)^{k\frac{v_j^2-1}{8} + \frac{t-1}{2}\frac{v_j-1}{2}} \left(\frac{v_j}{t}\right).$$

Это равенство вместе с (2.5) завершает шаг индукции. Итак, равенство (2.5) справедливо для всех j.

При $j = n$, т.е. при последнем выходе из пункта 2) будем иметь $t = 1$ и, следовательно,

$$\left(\frac{Q}{P}\right) = (-1)^{s_n} \cdot \left(\frac{u_n}{v_n}\right) = (-1)^{s_n} \left(\frac{2}{v_n}\right)^k = (-1)^{s_n + k\frac{v_n^2-1}{8}}.$$

Это доказывает, что алгоритм действительно вычисляет значение символа Якоби $\left(\frac{Q}{P}\right)$.

Учитывая равенства $u_j = v_{j-1}, t = v_{j+1}$, находим, что

$$v_{j-1} \geqslant v_j + v_{j+1}, \qquad j \geqslant 2.$$

Получившиеся неравенства, как и в доказательстве теоремы 4 приводят к оценке $v_j \geqslant \lambda^{n-j}$, доказывающей при $j = 1$, что $n = O(\ln P) = O(m)$. Оценка количества арифметических операций в алгоритме имеет, очевидно, тот же порядок. □

2.3. Возведение в степень.

Пусть A — некоторое множество, замкнутое относительно умножения. Это может быть множество целых чисел или многочленов, классов вычетов по какому-нибудь модулю и другие множества. Пусть также d — натуральное число. В этом параграфе мы обсудим быстрый алгоритм вычисления степени a^d. Тривиальный алгоритм, состоящий в последовательном умножении на a результата предшествующего вычисления, требует $d - 1$ умножений. Для некоторых чисел d вычисления могут быть проделаны намного быстрее. Так, например, для $d = 32$ можно обойтись всего лишь 5 умножениями

$$a^2, a^4 = (a^2)^2, a^8 = (a^4)^2, a^{16} = (a^8)^2, a^{32} = (a^{16})^2.$$

А для $d = 23$ достаточно 6 умножений

$$a^2, a^3 = a^2 \cdot a, a^5 = a^3 \cdot a^2, a^{10} = (a^5)^2, a^{13} = a^{10} \cdot a^3, a^{23} = a^{13} \cdot a^{10}.$$

Следующий общий алгоритм вычисляет степень существенно быстрее тривиального.

Алгоритм 3. Данные: *Элемент $a \in A$ и натуральное число d.*
Найти: *Элемент a^d.*

1. Представить d в двоичной системе счисления, т.е. найти такие числа $d_j \in \{0,\ 1\}$, что $d = d_0 2^r + \cdots + d_{r-1}2 + d_r,\ d_0 = 1$.
2. Положить $a_0 = a$ и затем для $i = 1, \ldots, r$ вычислить

$$a_i = a_{i-1}^2 \cdot a^{d_i}. \tag{2.6}$$

3. Положить $a^d = a_r$.

Теорема 5. *Алгоритм действительно вычисляет степень a^d. Он использует для этого не более $2[\log_2 d]$ умножений в кольце A.*

Доказательство. При всех $i = 0, 1, \ldots, r$ справедливы равенства

$$a_i = a^{d_0 2^i + \cdots + d_i}. \tag{2.7}$$

Докажем это индукцией по i. При $i = 0$ утверждение выполняется, т.к. $d_0 = 1$. Подставляя (2.7) в равенство $a_{i+1} = a_i^2 \cdot a^{d_{i+1}}$, находим равенство (2.7) для a_{i+1}. Это доказывает (2.7) для всех i. При $i = r$ получаем $a_r = a^d$, что доказывает корректность алгоритма.

Для оценки сложности обозначим символом $c(d)$ количество умножений, необходимых алгоритму для вычисления a^d. Докажем индукцией по d неравенство

$$c(d) \leqslant 2[\log_2 d]. \tag{2.8}$$

Обозначим для этого $m = d_0 2^{r-1} + \cdots + d_{r-1}$. Тогда $d = 2m + d_r \geqslant 2m$.

При $d = 1,\ 2$ неравенство (**??**), очевидно, выполняется. Пусть теперь $d \geqslant 3$. В силу алгоритма справедливы соотношения

$$c(d) = c(m) + 1 + d_r \leqslant c(m) + 2 \leqslant 2[\log_2 m] + 2 = 2[\log_2 2m] \leqslant 2[\log_2 d],$$

доказывающие нужное неравенство. □

Рассмотрим некоторые примеры использования описанного алгоритма.

Пример 1. Пусть m — натуральное число и $A = \mathbb{Z}/m\mathbb{Z}$ — кольцо вычетов по модулю m. Алгоритм реализует вычисление степеней в кольце вычетов по модулю m. Классическая теорема Эйлера утверждает, что $a^{\varphi(m)} \equiv 1 \pmod m$, где $\varphi(m)$ — функция Эйлера. Поэтому, заменив показатель степени d его остатком от деления на $\varphi(m)$ (на что потребуется одна арифметическая операция), задачу вычисления $a^d \pmod m$ всегда можно свести к случаю $d \leqslant \varphi(m)$. Это показывает, что сложность алгоритма возведения в степень в кольце вычетов $\mathbb{Z}/m\mathbb{Z}$ есть $O(\ln m)$.

В частности, если $m = p$ — простое нечетное число, то в силу одного из свойств символа Лежандра имеем

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p},$$

так что алгоритм 3 дает другой способ вычисления символа Лежандра за $O(\log p)$ арифметических операций.

Пример 2. Пусть R — кольцо и последовательность элементов u_0, u_1, $u_2, \ldots \in R$ задана рекуррентно

$$u_n = a_1 u_{n-1} + \cdots + a_h u_{n-h}, \qquad n \geqslant h, \quad a_j \in R.$$

Например, можно взять $R = \mathbb{Z}/m\mathbb{Z}$. Как вычислить u_d для заданного индекса d? Введем для этого вектора

$$\overline{u}_n = (u_n, u_{n-1}, \ldots, u_{n-h+1}), \qquad n \geqslant h.$$

Тогда справедливо равенство $\overline{u}_n = \overline{u}_{n-1} \cdot M$, где

$$M = \begin{pmatrix} a_1 & 1 & 0 & \cdots & 0 \\ a_2 & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{h-1} & 0 & 0 & \cdots & 1 \\ a_h & 0 & 0 & \cdots & 0 \end{pmatrix},$$

— квадратная матрица из h строк и столбцов с элементами в кольце R. Очевидно равенство $\overline{u}_n = \overline{u}_h \cdot M^{n-h}$, из которого следует, что вычисление u_d сводится к вычислению M^{d-h} в кольце A квадратных матриц размера h с элементами в R. Сложность вычисления u_d таким способом оценивается величиной $O(h \ln d)$, где постоянная в $O(\cdot)$ абсолютна.

Конец третьей лекции.

Лекция 4.

2.4. Быстрое умножение целых чисел.

Обычный метод умножения n - значных чисел "столбиком" требует n^2 перемножений цифр. В настоящее время имеются более эффективные методы, использующие существенно меньше операций умножения. Первый такой метод был предложен в 1960г. А.А. Карацубой.

Пусть a и b — произвольные натуральные числа. Они имеют, соответственно, по $[\log_2 a]+1$ и $[\log_2 b]+1$ цифр в двоичной записи. Пусть n — минимальное целое число, такое что

$$\max\Big([\log_2 a]+1,\ [\log_2 b]+1\Big) \leqslant 2^n.$$

Тогда a, b не более чем 2^n - разрядные целые числа.

Описываемый ниже алгоритм сводит задачу умножения чисел a и b к нескольким умножениям чисел, длина которых в двоичной записи на превосходит $k = 2^{n-1}$, и рекурсивно применяется к этим новым числам. Вычислительный алгоритм называется рекурсивным, если в процессе работы он обращается к самому себе, но с меньшими параметрами. Возникающее при этом множество обращений в конечном итоге сводит задачу к некоторым начальным значениям параметров, при которых выполнение алгоритма становится тривиальным.

Представим числа a и b в виде

$$a = a_1 + 2^k a_2, \qquad b = b_1 + 2^k b_2,$$

где a_1, a_2, b_1, b_2 — числа, длина которых в двоичной записи не превосходит k. Справедливо равенство

$$ab = a_1 b_1 + 2^k((a_1+a_2)(b_1+b_2) - (a_1 b_1 + a_2 b_2)) + 2^{2k} a_2 b_2. \qquad (2.9)$$

Числа $a_1 + a_2$ и $b_1 + b_2$ могут иметь в двоичной записи длину $k+1$, но их можно представить в виде

$$a_1 + a_2 = \alpha + 2a_3, \qquad b_1 + b_2 = \beta + 2b_3,$$

где числа a_3 и b_3 имеют длину не больше, чем k, а числа α и β суть нули или единицы. Стало быть, (2.9) можно переписать в виде

$$ab = a_1 b_1 + 2^k(\alpha\beta + 2\alpha b_3 + 2\beta a_3 + 4a_3 b_3 - (a_1 b_1 + a_2 b_2)) + 2^{2k} a_2 b_2. \quad (2.10)$$

Таким образом, исходная задача при помощи нескольких операций сложения, вычитания и домножения на степень двойки (которое является

просто–напросто приписыванием соответствующего числа нулей) сводится к вычислению трех произведений a_1b_1, a_2b_2 и a_3b_3, каждое из которых является произведением чисел длины не более, чем $k = 2^{n-1}$. К этим трем произведениям рекурсивно применяем описанные рассуждения.

Обозначим через L_n количество битовых операций, необходимое для вычисления данным методом произведения двух произвольных чисел, длина которых в двоичной записи не превосходит 2^n. Тогда найдется такая константа C, что $L_0 \leqslant C$ и

$$L_n \leqslant 3L_{n-1} + 2^n C, \qquad n \geqslant 1.$$

Отсюда по индукции находим, что

$$L_n \leqslant C(3^{n+1} - 2^{n+1}).$$

Действительно, в предположении, что $L_{n-1} \leqslant C(3^n - 2^n)$, получаем:

$$L_n \leqslant 3C(3^n - 2^n) + C2^n = C(3^{n+1} - 2^{n+1}).$$

Учитывая теперь, что $2^{n-1} < [\log_2 a] + 1$, при $a \geqslant b \geqslant 2$ заключаем

$$L_n \leqslant C \cdot 3^{n+1} = 3C \cdot (2^n)^{\log_2 3} < 3C \cdot (2[\log_2 a] + 2)^{\log_2 3} = \\ = 9C \cdot ([\log_2 a] + 1)^{\log_2 3}.$$

Таким образом, если a и b — натуральные числа и $a \geqslant b$, то перемножить их можно не более, чем за $9C \cdot m^{\log_2 3}$ битовых операций, где $m = [\log_2 a] + 1$ - количество цифр в двоичной записи числа a.

В 1971г. Шёнхаге и Штрассен придумали алгоритм умножения больших целых чисел, требующий не более $C_1 m \log_2 m \log_2 \log_2 m$ битовых операций, где m - количество цифр в двоичной записи большего из сомножителей, а C_1 - некоторая абсолютная постоянная, см. [14]. В 2007г. М. Фюрер, см. [11], предложил усовершенствование, позволившее заменить второй логарифм в оценке сложности алгоритма функцией, растущей существенно медленнее. Впоследствии был опубликован ещё ряд работ с усовершенствованиями этого результата. Предполагается, что существует алгоритм умножения с битовой сложностью порядка $m \log_2 m$. Найти такой алгоритм - это известная открытая математическая проблема.

2.5. Вероятностные методы отсеивания составных чисел.

Рассмотренные ранее алгоритмы относятся к разряду детерминированных. Так называют алгоритмы, в которых результат каждого шага

однозначно определяется предшествующими вычислениями. Однако, существуют и другие типы алгоритмов, так называемые вероятностные алгоритмы, весьма удобные на практике.

Пусть N — натуральное число, вообще говоря, большое. Оно может быть либо составным, либо простым. Соответственно, можно рассмотреть две важные в связи с поисками или построением больших простых чисел задачи.

1. N — составное число, и требуется доказать это.

2. N — простое число, и требуется доказать это.

В настоящем параграфе будет рассматриваться первая из задач. Мы покажем, что существуют достаточно быстрые вероятностные алгоритмы, решающие ее и не использующие при этом разложение N на множители.

Выберем целое число $a, 1 < a < N$. Справедливы следующие утверждения:

1) Если наибольший общий делитель $(a, N) > 1$, то N — составное число.

Условие $(a, N) > 1$ легко проверить, вычислив (a, N) с помощью алгоритма Евклида.

2) Если $(a, N) = 1$ и $a^{N-1} \not\equiv 1 \pmod{N}$, то N — составное число.

Это утверждение следует из малой теоремы Ферма, и его также легко проверить с помощью алгоритма возведения в степень.

К сожалению, утверждений 1)-2) не достаточно для доказательства того, что испытываемое число N — составное. Существуют составные числа, для которых выполняется сравнение $a^{N-1} \equiv 1 \pmod{N}$ при любом целом $a, (a, N) = 1$.

Определение 3. *Составное число N называется числом Кармайкла, если при любом $a, (a, N) = 1$, выполняется сравнение*

$$a^{N-1} \equiv 1 \pmod{N}. \tag{2.11}$$

Пример 1. $N = 561 = 3 \cdot 11 \cdot 17$.

Для каждого целого $a, (a, 561) = 1$, имеем $(a, 3) = (a, 11) = (a, 17) = 1$ и по малой теореме Ферма выполняются сравнения

$$a^2 \equiv 1 \pmod{3}, \qquad a^{10} \equiv 1 \pmod{11}, \qquad a^{16} \equiv 1 \pmod{17}.$$

Так как 560 делится на $2, 10$ и 16, то число $a^{560} - 1$ делится на каждую из разностей $a^2 - 1, a^{10} - 1, a^{16} - 1$. Значит, $a^{560} - 1$ делится на 3, 11, 17 и

потому делится на произведение этих чисел, т.е. делится на 561. Число 561 есть число Кармайкла.

В 1994г. было доказано, что множество чисел Кармайкла бесконечно. Существование чисел Кармайкла показывает необходимость уточнения свойств 1)-2) для доказательства непростоты чисел. Следующее утверждение позволяет достаточно эффективно решать рассматриваемую задачу.

Тест 1 (Миллер - Рабин)**.** *Пусть $N > 2$ — нечетное натуральное число. Определим целые числа s, t равенством $N - 1 = 2^s t$, где t нечетно. Выберем целое число $a, a > 1$.*
1) Если $(a, N) > 1$, то N составное число.
2) Если $(a, N) = 1$ и выполнены условия

$$a^t \not\equiv 1 \pmod{N}, \quad a^{2^k t} \not\equiv -1 \pmod{N}, \quad k = 0, 1, \ldots, s-1, \quad (2.12)$$

то N составное число.

Справедливо разложение на множители

$$a^{N-1} - 1 = (a^t - 1)(a^t + 1)(a^{2t} + 1) \cdots (a^{2^{s-1}t} + 1).$$

Если N — простое число, то по малой теореме Ферма обе части последнего равенства должны делиться на N. Поскольку N — простое, то хотя бы один из сомножителей в правой части этого равенства должен делиться на N. Значит, хотя бы одно из условий (2.12) будет нарушено. Это доказывает справедливость второго утверждения теста Миллера - Рабина.

Рассмотрим множество $M(N)$ состоящее из чисел $a \in \mathbb{Z}$, $1 \leqslant a < N$, $(a, N) = 1$, для которых нарушается хотя бы одно из условий (2.12).

Если N — простое число, то $\#M(N) = N - 1$. Это следует из справедливости второго утверждения теста Миллера - Рабина. Для составных N множество $M(N)$ будет достаточно малым. Точнее, выполняется доказанное в 1984г. Рабином утверждение.

Теорема 6. *Пусть N — нечетное составное число, $N \neq 9$. Тогда $\#M(N) \leqslant \frac{1}{4}\varphi(N)$.*

Мы не приводим здесь доказательство этой теоремы. Оно элементарно, но длинно, см. книгу [3].

Пример 2. *Для $N = 9$ имеем $M(9) = \{1,\ 8\}$ и $\#M(9) = 2 = \frac{1}{3}\varphi(9)$.*

Приведённый ниже вероятностный алгоритм, удостоверяет, что заданное число — составное, если оно таковым является. Алгоритм основан на тесте Миллера-Рабина.

Алгоритм 4. Данные: *Нечетное число $N > 9$ повидимому составное.* Доказать: *Число N составное.*

1) Вычислить натуральные числа s, t такие, что $N-1 = 2^s t$ и t нечетно.
2) Выбрать случайным образом число $a, 1 < a < N$ и проверить выполнимость условий 1) и 2) теста Миллера - Рабина.
3) Если хотя бы одно из условий теста выполнено, то число N составное; СТОП.
4) Если оба условия теста нарушаются, то перейти в пункт 2.

Для оценки сложности этого вероятностного алгоритма, что нужно для сравнения эффективности различных алгоритмов, можно использовать среднее время работы алгоритма, оно же иногда называется математическое ожидание времени работы.

Докажем сейчас, что среднее количество арифметических операций, необходимое для выполнения работы этого алгоритма, т.е. его сложность, не превосходит $c_0 \cdot \ln N$.[1]

Согласно определению множество $M(N)$, количество его элементов оценивается в теореме 6, состоит из всех чисел $a \in \mathbb{Z}$, $1 \leqslant a < N$, $(a, N) = 1$, для которых нарушается хотя бы одно из условий (2.12). Таким образом, любое целое число a, $1 \leqslant a < N$, не принадлежащее этому множеству, подтвердит с помощью теста Миллера - Рабина, что число N составное. Значит при случайном выборе a мы с вероятностью

$$\rho = \frac{N - 1 - \#M(N)}{N-1} \geqslant 1 - \frac{1}{4}\frac{\varphi(N)}{N-1} \geqslant \frac{3}{4}$$

попадаем на хорошее a (доказывающее непростоту N). Здесь использовалась оценка сверху $\#M(N) \leqslant \frac{1}{4}\varphi(N)$ количества элементов в множестве $M(N)$, справедливая для всех нечетных составных чисел $N > 9$, см. теорему 6.

Пусть A_k — событие, состоящее в том, что при k испытаниях $k-1$ раз попадались плохие a и в k-й раз попалось хорошее. Тогда $p(A_k) = (1-\rho)^{k-1}\rho$. Так как тест Миллера – Рабина использует только вычисление наибольшего общего делителя двух чисел и возведение в степень по модулю N, то при фиксированном a для проверки условий теста 2

[1] Здесь и далее буквой c с индексами обозначаются различные абсолютные постоянные, т.е. постоянные, не зависящие ни от каких параметром алгоритма, например $3, \pi$ или e^2.

требуется не более $c_1 \ln N$ арифметических операций. Для проверки же принадлежности $a \in A_k$ потребуется не более $c_1 k \ln N$ арифметических операций. Поэтому среднее количество арифметических операций в алгоритме не превосходит

$$c_1 \ln N \cdot \sum_{k=1}^{\infty} k(1-\rho)^{k-1}\rho = \rho \cdot c_1 \ln N \sum_{k=1}^{\infty} k(1-\rho)^{k-1}$$

Дифференцируя тождество $\sum_{k=1}^{\infty} x^k = -1 + \frac{1}{1-x}$, находим

$$\sum_{k=1}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2}.$$

Из этого тождества при $x = 1-\rho$ полуаем, что среднее количество арифметических операций в алгоритме не превосходит величины $\rho^{-1} c_1 \ln N \leqslant \frac{4c_1}{3} \ln N = c_2 \ln N$.

На практике алгоритм 6 является очень важной составляющей общей стратегии доказательства простоты чисел. Если заданное число N, о котором не известно простое оно или составное, прошло достаточно много шагов алгоритма 6, то оно наверное будет простым. Ведь вероятность составному числу N выдержать d испытаний не превосходит 4^{-d}. Например, при $d = 100$ она ничтожно мала 4^{-100}. Таким образом, остается только вопрос, как доказать, что это число простое?

Конец четвёртой лекции.

Лекция 5.

2.6. Доказательство простоты больших чисел

В начале этого параграфа мы обсудим так называемые $(N-1)$ - методы доказательства простоты чисел. Они позволяют, используя некоторую информацию о свойствах простых делителей числа $N-1$, заключить, что N - простое число. Существуют также $(N+1)$ и (N^2+1) - методы, есть и другие обобщения излагаемых ниже идей, но мы их здесь касаться не будем.

Теорема 7 (Лемер, Поклингтон). *Пусть N нечетно, $N-1=F\cdot R$, причем для каждого простого делителя q числа F с некоторым целым b выполнены условия*

$$b^{N-1}\equiv 1 \pmod N, \qquad \left(b^{(N-1)/q}-1,\ N\right)=1. \tag{2.13}$$

Тогда любой простой делитель p числа N удовлетворяет сравнению

$$p\equiv 1 \pmod F$$

В следующем далее доказательстве теоремы 7 используется символ $\nu_q(a)$, обозначающий для любого целого a и простого q кратность, с которой q входит в разложение a на простые сомножители.

Доказательство. Пусть p — простой делитель N, q — простой делитель F и b — число, удовлетворяющее (2.13). Первое из сравнений (2.13) означает, что b не делится на p. Обозначим $a=b^R$. Тогда a не делится на p и по малой теореме Ферма выполняется сравнение $a^{p-1}\equiv 1 \pmod p$. Обозначим буквой d наименьшее натуральное число с условием $a^d\equiv 1 \pmod p$. Разделим $p-1$ на d с остатком: $p-1=ud+v$, где $0\leqslant v<d$. Тогда

$$1\equiv a^{p-1}=(a^d)^u\cdot a^v\equiv a^v \pmod p.$$

Если $v>0$, приходим к противоречию с определением d. Поэтому $v=0$,

$$d|(p-1) \quad \text{и} \quad \nu_q(d)\leqslant\nu_q(p-1). \tag{2.14}$$

Из (2.13) следует, что

$$a^F\equiv 1 \pmod p, \qquad a^{F/q}\not\equiv 1 \pmod p.$$

Отсюда точно так же, как и (2.14) выводятся свойства $d|F$ и $d\nmid F/q$, что возможно лишь в случае, когда

$$\nu_q(F)=\nu_q(d). \tag{2.15}$$

Из (2.14) и (2.15) находим

$$\nu_q(F) = \nu_q(d) \leqslant \nu_q(p-1).$$

Так как последнее неравенство справедливо для любого простого делителя q числа F, то $F|p-1$ или $p \equiv 1 \pmod F$. □

Приведём два следствия теоремы, связанные с доказательством простоты чисел. Если известна достаточно большая часть разложения $N-1$ на простые сомножители, то иногда можно сделать заключение о простоте N.

Следствие 1. *Если в условиях теоремы 7 выполнено неравенство*

$$R \leqslant F+1$$

то N — простое.

Доказательство. Согласно теореме 7 каждое простое $p|N$ удовлетворяет сравнению $p \equiv 1 \pmod F$ и, значит, удовлетворяет неравенству $p \geqslant 1+F$. Предположим, что N — составное. Тогда

$$(F+1)^2 \leqslant N = FR+1 \leqslant F^2+F+1.$$

Получившееся противоречие завершает доказательство. □

Следствие 2. *Пусть F – простое нечётное число, R – чётное число, что удовлетворяющее неравенству $R \leqslant 4F+2$ и $N = FR+1$. Если*

$$b^{N-1} \equiv 1 \pmod N, \qquad \left(b^R-1,\, N\right) = 1. \tag{2.16}$$

с некоторым целым b из промежутка $1 < b < N$, то N – простое число.

Доказательство. Согласно теореме 7 при $q = F$ каждое простое p, делящее N удовлетворяет сравнению $p \equiv 1 \pmod F$. Кроме того, p нечетно, так что $p \equiv 1 \pmod{2F}$ и $p \geqslant 2F+1$. Для составного N имеем неравенства

$$(2F+1)^2 \leqslant N = FR+1 \leqslant 4F^2+2F+1,$$

что неверно. Значит, N — простое число. □

Предположим теперь, что построенное число N действительно является простым, и мы хотим доказать это с помощью Следствия 2. Если при выбранном b условия (2.13) нарушаются, нужно выбрать другое значение b и повторять эти операции до тех пор, пока такое число не будет обнаружено. Зададимся вопросом, сколь долго придётся перебирать

числа b, пока не будет найдено такое, для которого выполнены условия (2.13). Заметим, что для простого числа N первое условие (2.13), согласно малой теореме Ферма, будет выполняться всегда. Те же числа b, для которых нарушается второе условие (2.13), удовлетворяют сравнению $b^R \equiv 1 \pmod N$. Как известно, уравнение $x^R = 1$ в поле вычетов $\mathbb{Z}/N\mathbb{Z}$ имеет не более R решений. Одно из них — $x = 1$. Поэтому на промежутке $1 < b < N$ имеется не более $R - 1$ чисел, для которых не выполняются условия (2.13). Это означает, что, выбирая случайным образом число b на промежутке $1 < b < N$, при простом N можно с вероятностью большей, чем $1 - F^{-1}$, найти число b, для которого будут выполнены условия следствия 2, и тем доказать, что N действительно является простым числом.

Построенное таким способом простое число N будет удовлетворять неравенству $N > F^2$, т.е. будет записываться вдвое большим количеством цифр, чем исходное простое число F. Заменив теперь число F на найденное простое число N и повторив с этим новым F все указанные выше действия, можно построить еще большее простое число. Начав с какого-нибудь простого числа, скажем, записанного 10 десятичными цифрами (простоту его можно проверить, например, делением на маленькие табличные простые числа), и повторив указанную процедуру достаточное число раз, можно построить простые числа нужной величины.

Алгоритм 5. Данные: *Большое положительное целое число M. Алгоритм строит простое число N, превосходящее границу M.*

1) Определить, например, с помощью таблиц простое число F, превосходящее 10^5. Определить счётчик S_1 и положить $S_1 = 0$.
2) Выбрать случайным образом чётное число R из промежутка $F \leqslant R \leqslant 4F + 2$ и положить $N = FR + 1$. Если НОД$[N, 15015] > 1$, повторить пункт 2.
3) Вычислить натуральные числа s, t такие, что $N - 1 = 2^s t$ и t нечетно.
4) Выбрать случайным образом число $a, 1 < a < N$. Если
• $(a, N) > 1$ *или*
• $(a, N) = 1$ *и*

$$a^t \not\equiv 1 \pmod N, \quad a^{2^k t} \not\equiv -1 \pmod N, \quad k = 0, 1, \ldots, s - 1, \quad (2.17)$$

то N составное число. Перейти в пункт 2). В противном случае увеличить счётчик S_1 на 1 и, если $S_1 < 20$, перейти в п. 4. Если же $S_1 = 20$, положить $S_1 = 0$ и $S_2 = 0$, перейти в п. 5).

5) Выбрать случайным образом b из промежутка $1 < b < N$. Если N не удовлетворяет хотя бы одному из условий

$$b^{N-1} \equiv 1 \pmod{N}, \qquad \left(b^R - 1,\, N\right) = 1, \tag{2.18}$$

то
5.1) при $S_2 < 10$ увеличить S_2 на 1 и повторить пункт 5).
5.2) В случае же $S_2 = 10$, перейти в пункт 4.
6) Если N удовлетворяет обоим условиям (2.18) и
6.1) $N < M$, положить $F = N$ и перейти в пункт 2.
6.2) $N \geqslant M$, алгоритм останавливается, нужное простое число построено.

Конец пятой лекции.

Лекция 6.

2.7. Последовательности псевдослучайных чисел.

В двух предшествующих параграфах мы обсуждали вероятностные алгоритмы, которые, впрочем, дают совершенно верные результаты. Лишь оценки их времени работы носят усреднённый характер. Например, если натуральные числа N и a взаимно просты и $a^{N-1} \not\equiv 1 \pmod{N}$, можно с уверенностью утверждать, что N составное число. Нам не известны нетривиальные натуральные сомножители, на которые раскладывается это число, но мы знаем, что они существуют. А сложность алгоритма, отсеивающего составные числа, по-существу, оценивается средним временем работы, необходимой для нахождения подходящего числа a.

В вероятностных алгоритмах используется случайный выбор каких-либо чисел на нужном промежутке. Конечно, у каждого есть своё представление, какая последовательность может считаться случайной, но вместе с тем, понятно, что с помощью детерминированного алгоритма, используемого компьютером, случайную последовательность не построишь. Можно лишь построить последовательность, имитирующую те или иные свойства случайной последовательности. Такие последовательности называются псевдослучайными, а успех вероятностного алгоритма зависит от того, насколько используемая в нём псевдослучайная последовательность близка по своим свойствам к случайной.

2.7.1. Линейный конгруэнтный метод.

В течение длительного времени основным инструментом для построения псевдослучайных последовательностей был так называемый линейный конгруэнтный метод, предложенный в 1948г. Д. Лемером. Нужная последовательность возникала по правилу

$$x_{n+1} \equiv a \cdot x_n + b \pmod{m}, \qquad 0 \leqslant x_{n+1} < m, \quad n \geqslant 0, \tag{2.19}$$

где a, b, m и x_0 - некоторые параметры, определяемые специальным образом. Выбор параметров - непростая задача, как показывают следующие далее примеры.

Пример. Последовательность $x_n, n \geqslant 0$, определённая условиями

$$x_{n+1} \equiv 1986 x_n + 35491 \pmod{2^{16}}, \qquad n \geqslant 0, \qquad x_0 = 1181,$$

имеет вид

$$1181, 21661, 62661, 13469, 46237, 46237, 46237, \ldots .$$

Все её члены, начиная с пятого места, одинаковы, что, конечно, не свойственно случайным последовательностям.

Любая последовательность, построенная по правилам

$$x_{n+1} = f(x_n) \pmod m, \qquad n \geqslant 0, \qquad 0 \leqslant x_{n+1} < m, \tag{2.20}$$

где x_0 - произвольно выбранное начальное значение и $f(x)$ - функция, определённая для всех целых неотрицательных чисел и принимающая целые значения, будет периодической, а длина её периода не превосходит m. Действительно, существует ровно m различных классов вычетов по модулю m. Поэтому множество классов вычетов $f(x_n) \pmod m$, $0 \leqslant n \leqslant m$, состоящее из $m+1$ классов вычетов, должно содержать по крайней мере два одинаковых элемента. Если $f(x_k) \equiv f(x_\ell) \pmod m$, $0 \leqslant \ell < k \leqslant m$, то

$$x_{k+1} \equiv f(x_k) \equiv f(x_\ell) \equiv x_{\ell+1} \pmod m.$$

Учитывая, что $0 \leqslant x_{\ell+1} < m$, $0 \leqslant x_{k+1} < m$ и разность этих чисел делится на m, заключаем, что $x_{k+1} = x_{\ell+1}$ и $f(x_{k+1}) = f(x_{\ell+1})$. Далее точно так же доказывается, что $x_{k+2} = x_{\ell+2}$ и вообще $x_{n+k-\ell} = x_n$ для любого $n \geqslant \ell$. Итак, начиная с номера ℓ последовательность начнёт повторяться с периодом $k - \ell \leqslant m$.

Случайная последовательность не должна быть периодической. Поэтому естественно желание выбирать в качестве псевдослучайных, последовательности, имеющие, по возможности, больший период. Справедливо следующее утверждение. *Длина периода последовательности* (2.19) *будет максимальной (равной m) в том и только том случае, когда:*

1. b и m взаимно просты,

2. $a-1$ кратно всем простым делителям числа m,

3. если m кратно 4, то и $(a-1)$ делится на 4.

Мы не будем здесь доказывать это утверждение, см. .

Примеры. Последовательности

$$x_{n+1} \equiv 19381 \cdot x_n + b \pmod{2^{16}}$$

при любом нечётном b и любом x_0 имеют максимальный возможный период $2^{16} = 65536$. Так будет, например, при $b = x_0 = 1$. Если же мы возьмём последовательность, определённую условиями

$$x_{n+1} \equiv 19381 \cdot x_n + 29772 \pmod{2^{16}})], \qquad x_0 = 1,$$

то она будет иметь вид

$$1, 49153, 32769, 16385, 1, 49153, 32769, 16385, 1, \ldots .$$

Период её равен 4, конечно же она не является псевдослучайной.

Свойство иметь максимальный период ещё не обеспечивает хорошие с точки зрения случайности свойства последовательности. Например, рекуррентное уравнение

$$x_{n+1} \equiv x_n + 1 \pmod{2^{16}}, \qquad x_0 = 1,$$

определяет последовательность максимального периода

$$1, 2, 3, 4, \ldots, 65534, 65535, 0, 1, 2, 3, 4, \ldots,$$

очевидно не случайную.

Рассмотрим ещё один пример. Определим последовательность уравнением

$$x_{n+1} = 16565 \cdot x_n + 4051, \qquad n \geqslant 0, \qquad x_0 = 1. \tag{2.21}$$

Начало этой последовательности имеет вид

$$1, 20616, 65531, 52298, 37, 27132, 65279, 6686, 1801, 18736, 52931,$$
$$65458, 22701, 548, 37703, 61702, 63761, 26840, 12427, 8730$$

Эта последовательность имеет максимальный период и на первый взгляд выглядит как случайная. Тем не менее использовать её, как псевдослучайную не имеет смысла, ведь для всех $n \geqslant 0$ выполняется соотношение

$$x_{n+2} + 7x_n - 2 \equiv 0 \pmod{[)2^(16)]}. \tag{2.22}$$

Предпринимались попытки усложнить функцию $f(x)$, стоящую в (5.4), с тем, чтобы исключить соотношения, подобные (2.22), и другие нежелательные эффекты. С этой целью рассматривались полиномы второй и третьей степени, дробно линейные функции. Параллельно разрабатывались тесты для проверки свойств, характерных для случайных последовательностей. Перспективнее оказались последовательности в рекуррентном определении котоых присутствуют функции от нескольких переменных, например

$$x_{n+q} = f(x_{n+q-1}, x_{n+q-2,\ldots,x_{n+1},x_n}),$$

имеющие существенно большую длину периода.

В криптографических приложениях последовательности, конструируемые с помощью линейного конгруэнтного метода не используются. Для

того, чтобы построить случайное число записываемое в двоичной системе счисления 1024 цифрами 0 и 1, а в настоящее время это нижняя граница величины допустимых простых чисел в задачах дискретного логарифмирования, нужно приложить друг к другу 64 случайных числа, записываемые 16 битами. При этом должна быть обеспечена их независимость. В приведённом выше примере (2.21) уже соседние три члена последовательности связаны соотношением с маленькими коэффициентами. В следующем подразделе мы раскажем о том, как строить псевдослучайные последовательности чисел размером, скажем в 256 битов. Чтобы построить псевдослучайное число размером в 1024 бита, достаточно поместить рядом[2] 4 члена псевдослучайной последовательности чисел, каждое из которых записывается 256 битами. В этой конструкции используются так называемые хеш-функции.

2.7.2. Хеш-функции и псевдослучайные последовательности.

Хеш-функции[3] используются, как составной элемент в различных криптографических приложениях. Эти функции отображают сколь угодно длинные сообщения в целые числа ограниченной величины. Значения хеш-функций называют хеш-значениями. Наиболее важные их применения связаны с подтверждением достоверности информации и со схемами цифровой подписи. При этом хеш-значения сообщений служат аналогами "отпечатков пальцев"человека. Люди имеют индивидуальные отпечатки пальцев. На этом строятся криминологические расследования. Иногда мы будем называть хеш-значения "отпечатками сообщений". Заметим, что по отпечаткам пальцев человека невозможно понять, какой у него цвет глаз, какой рост, каковы его вкусы. Нечто подобное имеет место и для отпечатков сообщений. По хеш-значению сообщения невозможно понять, что в нём содержится.

Перекодировав буквы числами, любое сообщение можно представить числом, вообще говоря большим целым числом. Каждое же целое число можно записать в двоичном виде, другими словами, записать в виде последовательности 0 и 1. Длины таких последовательностей могут быть сколь угодно велики. Множество всевозможных конечных последовательностей из 0 и 1 будем обозначать символом $\{0,1\}^*$. А для сово-

[2]Эта операция называется конкатенацией.

[3]Термин hash function имеет нетривиальный перевод на русский язык "хеш-функция". Английское слово hash — в словарях, не относящихся к компьютерам и информационным технологиям, переводится как перемешивать, резать, рубить, крошить, делать фарш, а если это слово - существительное, им обозначают рубленую смесь готового мяса и овощей, обычно запеченного или обжаренного. Мы будем иногда для словосочетания hash function использовать, как русский перевод, слова - "хеширующая функция"или "перемешивающая функция".

купности таких последовательностей, имеющих фиксированную длину $n, n \geqslant 1$, будем использовать обозначение $\{0,1\}^n$.

Рассмотрим отображение

$$H : \{0,1\}^* \mapsto \{0,1\}^n, n \in \mathbb{N}, \tag{2.23}$$

при каком-либо фиксированном натуральном n. Пример такого отображения - сумма цифр входного сообщения (аргумента H), она же - количество 1 в сообщении, взятая по модулю 2^n.

Отображение H называется *хеш-функцией, функцией хеширования* или *перемешивающей функцией*, если она имеет описываемые далее три свойства.

1. *Однонаправленность.* Значения H должны быть вычислимы с помощью некоторого алгоритма, причём этот алгоритм должен работать достаточно быстро. Конечно, время работы зависит от длины входа, будем считать эту зависимость полиномиальной. При этом вычисление значений обратной функции H^{-1} должно быть очень трудоёмким. Другими словами, по заданному числу y любое решение x уравнения $H(x) = y$ должно быть очень трудно вычислимым. Можно также сказать, что для любого выходящего сообщения функции H должно быть "очень трудно"вычислить какой-нибудь его прообраз (соответствующее входное сообщение).

2. *Отсутствие коллизий. Коллизией* для хеш-функции H называется любая пара конечных последовательностей x и y, состоящих из 0 и 1, для которых выполняется равенство $H(x) = H(y)$. Для любой хеш-функции множество коллизий бесконечно. Это утверждение легко доказать с помощью принципа ящиков Дирихле. Действительно, множество возможных последовательностей, которые могут быть поданы на вход хеш-функции бесконечно, а количество возможных выходов не превосходит 2^n. Количество входов больше количества выходов, поэтому должно быть несколько входов, отображающихся на один какой-нибудь выход. Более того, существует бесконечное количество входов, имеющих один и тот же конкретный выход. Таким образом, любая хеш-функция имеет бесконечное количество коллизий. Упомянутое выше свойство "отсутствие коллизий"у хеш-функции, означает лишь то, что задача нахождения хотя бы одной коллизии очень сложна в вычислительном отношении. Это - отсутствие коллизий до определённой поры.

3. *Сложность нахождения второго прообраза.* Это свойство можно выразить иными словами, сказав, что знание одного решения уравнения (системы уравнений) $H(x) = y$ не упрощает поиск ещё одного решения.

Для некоторых хеш-функций очень просто построить коллизии или какие-нибудь другие нарушения указанных трёх свойств. Например, если $H(x)$ есть сумма цифр в двоичной записи числа x, то $H(9) = H(17) = 2$ - коллизия. Но, вообще говоря, не существует метода, который для заданной хеш-функции доказывал бы, что поиск коллизий у неё требует очень большого времени или же, наоборот, легко находил существующие коллизии. Безопасность хеш-функции это всегда вопрос веры, основанной на многочисленных проверках и большом объёме компьютерных вычислений в попытках скомпрометировать её. Используемые на практике хеш-функции прошли сложные, иногда длящиеся годами, испытания, являются победителями или призёрами специальных конкурсов, и имеют собственные имена.

Как только для какой-нибудь хеш-функции найдены коллизия или какой-нибудь алгоритм, существенно снижающий трудоёмкость решения задач в пунктах 1) или 3), эта функция считается скомпрометированной и заменяется другой. Так было в сравнительно недавнее время с хеш-функциями, называвшимися MD5 и SHA-1.

В РФ используется функция хеширования, носящая имя "Стрибог". Она была введена в 2012 году соответствующим стандартом и заменила другую функцию, действовавшую в период с 1994 по 2011 годы. Длина выходных сообщений выбывшей функции равнялась 256 битов, но для неё в 2008 году был найден алгоритм обращения, см. п. 1), требующий 2^{225} операций. Это количество всё ещё велико для современных суперкомпьютеров, тем не менее эта функция считалась скомпрометированной и была заменена функцией "Стрибог".

В связи с биткойнами используется хеш-функция, называемая SHA-256, [4] все её значения могут быть записаны строками, состоящими из 256 нулей и единиц, т.е. являются целыми числами из промежутка $0 \leqslant x < 2^{256}$. Эта функция была разработана Агенством национальной безопасности США в 2002 году. В настоящее время для неё, как и любой другой работающей в серьёзных приложениях хеширующей функции, коллизии не известны. И не то, чтобы кто-то знал коллизию и скрывал её. Нет, вообще никто не знает коллизию для SHA-256, и за прошедшие годы никто не смог найти и предъявить её.

О том, как устроены и работают хеширующие функции, мы поговорим позже. Некоторое представление об этом дают слова, указанные в комментарии к переводу названия hash function. В дальнейшем мы узнаем больше и о том, какова роль таких функций в приложениях. Сейчас же

[4] SHA — сокращение от Secure Hashing Algorithm.

мы покажем, как используются хеширующие функции при построении псевдослучайных чисел.

Пусть $H(x)$ какая-нибудь принятая хеширующая функция, длину её выходов, как и ранее, будем обозначать буквой n. Следующий алгоритм строит псевдослучайную последовательность натуральных чисел x длины не более $\ell - 1$ битов, т.е. чисел, удовлетворяющих неравенствам $0 \leqslant x < 2^{\ell-1}$.

Алгоритм 6. Данные: ℓ, d - *целые числа,* $\ell > 16$, $d > 0$.

Выход: *Псевдослучайная последовательность длины* d, *состоящая из чисел* x, $0 \leqslant x < 2^{\ell-1}$.

1. Положить $I = \lceil \frac{\ell}{n} \rceil - 1 \geqslant 0$. [5], *а также*

$$a = \ell - 1 - I \cdot n$$

2. Выбрать произвольное натуральное число s.

3. Определить последовательность целых чисел z_k *равенствами*

$$z_0 = H(s), \qquad z_k = z_{k-1} + H(s+k) \cdot 2^{kn}, \quad 1 \leqslant k \leqslant I - 1.$$

4. Положить

$$x = z_{I-1} + (H(s+I) \pmod{2^a}) \cdot 2^{I \cdot n}.$$

При $I = 0$ *в предыдущей формуле присутствует только последнее слагаемое.*

5. Если количество построенных чисел x *меньше* d, *положить* $s = s + I + 2$ *и перейти в пункт 3. В противном случае завершить работу.*

В согласии с определением чисел I, a и свойствами функции $\lceil y \rceil$ имеем

$$\frac{\ell}{n} - 1 \leqslant I < \frac{\ell}{n} \quad \text{и} \quad -1 < a \leqslant n - 1.$$

Ясно что построенное число x неотрицательно. Следующее из пунктов 3 и 4 алгоритма равенство

$$x = \sum_{k=0}^{I-1} H(s+k) \cdot 2^{nk} + (H(s+I) \pmod{2^a}) \cdot 2^{I \cdot n}.$$

есть представление числа x в 2^n-ичной системе счисления, причём старшая "цифра"числа x не превосходит $2^a - 1$. Поэтому

$$x \leqslant (2^n - 1) \sum_{k=0}^{I-1} 2^{nk} + (2^a - 1) \cdot 2^{nI} = 2^{a+nI} = 2^{\ell-1}.$$

[5] Символ $\lceil y \rceil$ обозначает наименьшее целое, превосходящее или равное действительному числу y. При любом $y > 0$ выполняется неравенство $\lceil y \rceil \geqslant 1$.

Можно также сказать, что последовательность цифр 0 и 1 числа x в двоичной системе счисления, расположенная в обратном порядке есть конкатенация[6] векторов $H(s)\|H(s+1)\|\ldots\|H(s+I)$ с обрезанными старшими цифрами у вектора $H(s+I)$.

Для построения псевдослучайной последовательности чисел y на промежутке $2^{\ell-1} \leqslant y < 2^{\ell}$ достаточно сдвинуть полученную выше последовательность чисел x на величину $2^{\ell-1}$, т.е. положить $y = 2^{\ell-1} + x$.

Ещё одна вариация на ту же тему. Предположим нужно построить псевдослучайную последовательность целых чисел t, принадлежащих промежутку $A \leqslant t < B$. Положим для этого $I = \lceil \frac{\log(B-A)}{n} \rceil - 1$, где логарифм вычисляется по основанию 2 и определим x формулой

$$x = A + \left(\sum_{k=0}^{I} H(s+k)2^{nk} \pmod{B-A} \right). \tag{2.24}$$

Тогда $A \leqslant x < A + (B-A) < B$ и максимальное возможное значение суммы в (2.24) равно

$$S = \sum_{k=0}^{I} (2^n - 1) \cdot 2^{nk} = 2^{n(I+1)} - 1.$$

Так как

$$n(I+1) = n\lceil \frac{\log(B-A)}{n} \rceil \geqslant n \cdot \frac{\log(B-A)}{n} = \log(B-A),$$

то $2^{n(I+1)} - 1 \geqslant B - A - 1$.

В следующем параграфе эти методы построения псевдослучайных чисел будут использоваться для построения больших простых чисел.

[6] Конкатенация $a_1\|a_2\|\ldots\|a_m$ векторов $a_1, a_2, \ldots, a_m$ есть вектор, полученный состыковкой этих векторов в единый длинный вектор $a_1 a_2 \ldots a_{m-1} a_m$.