Числовые алгоритмы

Наумов Д.А., каф. ИТГД

Алгоритмы и структуры данных, 2021

Рассмотрим две задачи:

1. **разложение на множители**: представить данное число N как произведение простых
    - разложение на множители является очень сложной задачей;
    - самые быстрые алгоритмы разложения числа N на простые множители по-прежнему требуют экспоненциального времени.
2. **проверка на простоту**: выяснить, является ли данное число N простым.
    - проверить простоту числа N можно довольно быстро;
    - на этом разрыве между двумя родственными задачами основаны современные технологии безопасного обмена информацией.

Сложение

Сумма любых трёх цифр будет однозначным или двузначным числом.

перенос:	1			1	1	1		
		1	1	0	1	0	1	(53)
		1	0	0	0	1	1	(35)
	1	0	1	1	0	0	0	(88)

- Как число операций зависит от размера входных данных (количества битов в записи x и y)?
- Существует ли более быстрый алгоритм?

Умножение

«Школьный» метод умножения двух чисел x и y «в столбик».

$$
\begin{array}{cccccccccl}
 & & & \times & 1 & 1 & 0 & 1 & & \\
 & & & & 1 & 0 & 1 & 1 & & \\
\hline
 & & & & 1 & 1 & 0 & 1 & & \text{(1101 умножить на 1)} \\
+ & & & 1 & 1 & 0 & 1 & & & \text{(1101 умножить на 1, сдвинутое на 1)} \\
 & & 0 & 0 & 0 & 0 & & & & \text{(1101 умножить на 0, сдвинутое на 2)} \\
 & 1 & 1 & 0 & 1 & & & & & \text{(1101 умножить на 1, сдвинутое на 3)} \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & & \text{(143 в двоичной системе счисления)}
\end{array}
$$

Для последовательного сложения этих n чисел нужно:

$$\underbrace{O(n) + O(n) + \ldots + O(n)}_{n-1 \text{ раз}} = O(n^2)$$

Время работы квадратично относительно размера входов и по-прежнему полиномиально ($((n^2)$), хотя и гораздо больше, чем для сложения).

Умножение. Алгоритм Аль-Хорезми

1. Запишем множители x и y рядом.
2. Теперь будем повторять следующую операцию: поделим первое число пополам, отбросив дробную часть $1/2$ (если число было нечётным), а второе число удвоим.
3. Будем делать так до тех пор, пока первое число не станет единицей.
4. После этого вычеркнем все строки, в которых первое число чётно, и сложим оставшиеся числа из второй колонки.

11	13	
5	26	
2	52	(вычёркиваем)
1	104	
	143	(ответ)

Умножение. Рекурсивный алгоритм Multiply

Алгоритм реализует следующее правило:

$$x \cdot y = \begin{cases} 2\left(x \cdot \left\lfloor \frac{y}{2} \right\rfloor\right), & \text{если } y \text{ чётно,} \\ x + 2\left(x \cdot \left\lfloor \frac{y}{2} \right\rfloor\right), & \text{если } y \text{ нечётно.} \end{cases}$$

Корректен ли данный алгоритм? Каково время работы алгоритма?

```
функция MULTIPLY(x, y)
{Вход: множители x и y, где y ⩾ 0.}
{Выход: x y.}
если y = 0:  вернуть 0
z    MULTIPLY(x, ⌊y/2⌋)
если y чётно:
   вернуть 2z
иначе:
   вернуть x + 2z
```

Деление

Поделить целое число x на положительное целое число y означает найти такие числа q (частное) и r (остаток), что:

$$x = y \cdot q + r \text{ и } 0 \leq r < y$$

функция $\text{DIVIDE}(x, y)$
{Вход: n-битовые x и y, причём $y \geqslant 1$.}
{Выход: частное и остаток от деления x на y.}
если $x = 0$: вернуть $(q, r) = (0, 0)$
$(q, r) \leftarrow \text{DIVIDE}(\lfloor x/2 \rfloor, y)$
$q \leftarrow 2 \cdot q$, $r \leftarrow 2 \cdot r$
если x нечётно: $r \leftarrow r + 1$
если $r \geqslant y$: $r \leftarrow r - y$, $q \leftarrow q + 1$
вернуть (q, r)

Арифметика сравнений

- При сложении и особенно умножении размеры чисел могут сильно возрасти.
- Иногда полезно «сбрасывать» ставшие слишком большими числа.
- В интересующих нас приложениях используются большие числа, но и они имеют ограниченную длину, и при сложении и умножении используется арифметика остатков, или сравнений.

Фиксируем некоторое целое положительное N.

Определим $x \; mod \; N$ (x по модулю N) как остаток от деления x на 6N:

$$\text{если } x = qN + r, \text{ где } 0 \leq r < N, \text{то } x \; mod \; N = r$$

Это позволяет ввести отношение эквивалентности на числах:

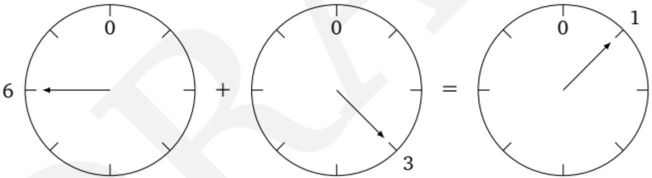
- x сравнимо с y по модулю N,
- если x и y дают при делении на N одинаковый остаток,
- то есть если их разность делится на N:

$$x \equiv y \pmod{N} \quad \Leftrightarrow \quad (x - y) \text{ делится на } N.$$

Арифметика сравнений

Работая с числами по модулю N, мы ограничиваемся отрезком $\{0, 1, \dots, N-1\}$, и, достигнув конца отрезка, мы просто попадаем в его начало.

Сложение по модулю 8.



Сложение и умножение по модулю N по-прежнему обладают свойствами коммутативности, ассоциативности и дистрибутивности:

$$x + (y + z) \equiv (x + y) + z \pmod N \qquad \text{ассоциативность}$$
$$xy \equiv yx \pmod N \qquad \text{коммутативность}$$
$$x(y + z) \equiv xy + yz \pmod N \qquad \text{дистрибутивность}$$

Арифметика сравнений

Можно также представлять себе, что мы работаем со всеми целыми числами, но они поделены на N классов эквивалентности, каждый из которых представляет собой арифметическую прогрессию с разностью N.

$$\begin{array}{ccccccccc} \ldots & -9 & -6 & -3 & 0 & 3 & 6 & 9 & \ldots \\ \ldots & -8 & -5 & -2 & 1 & 4 & 7 & 10 & \ldots \\ \ldots & -7 & -4 & -1 & 2 & 5 & 8 & 11 & \ldots \end{array}$$

При выполнении арифметических операций по модулю N любые промежуточные результаты можно заменять на их остатки по модулю N:

$$2^{345} \equiv (2^5)^{69} \equiv 32^{69} \equiv 1^{69} \equiv 1 \pmod{31}$$

Сложение по модулю

Чтобы сложить два числа x и y по модулю N, мы должны их сложить обычным образом и затем взять остаток при делении на N.

- Поскольку x и y лежат между 0 и $N-1$, их сумма лежит в интервале от 0 до $2(N-1)$.
- Если сумма превосходит $N-1$, нам остаётся лишь вычесть N.
- Таким образом, всё вычисление состоит из сложения, сравнения и (возможно) вычитания целых чисел, не превосходящих 2N.

Время работы будет линейно, то есть $O(n)$, где $n = \lceil log(N) \rceil$ есть размер двоичной записи числа N.

Умножение по модулю

Умножение чисел x и y по модулю N делается точно так же: сперва мы перемножаем числа, после чего берём остаток по модулю N.

- Произведение не превосходит $(N-1)^2$ и занимает не более $2n$ битов, поскольку $log(N-1)^2 = 2 \cdot log(N-1) \leq 2n$.
- Чтобы найти остаток по модулю N, мы используем алгоритм деления.

И умножение, и деление требуют квадратичного времени, и мы получаем квадратичный алгоритм.

Деление по модулю

- Деление по модулю N, то есть решение уравнения $ax \equiv b(mod \ N)$ является более сложной операцией.
- В отличие от обычной арифметики, где проблемы возникают только при делении на ноль, в арифметике сравнений есть и другие сложные случаи.
- Мы увидим, что деление (если оно вообще возможно) может быть выполнено за кубическое время, то есть $O(n)^3$.

Возведение в степень по модулю

В криптосистеме, которую мы рассмотрим чуть позже, нам придётся вычислять $x^y \ mod \ N$ для x, y и N, состоящих из нескольких сотен битов. Как это сделать быстро?

- $x^y \ mod \ N$ - число в несколько сот битов;
- x^y - может иметь несколько миллонов битов.

$$(2^{19})^{(2^{19})} = 2^{19 \cdot 524288}$$

- вычислять остаток от деления на N нельзя в самом конце вычислений;
- если вычислять степень последовательно?

$$x \ mod \ N \to x^2 \ mod \ N \to x^3 \ mod \ N \to \ldots x^y \ mod \ N$$

- проблема: если y содержит 500 битов, то потребуется $y - 1 \approx 2^{500}$;
- время работы будет пропорционально y, то есть расти экспоненциально.

Быстрое возведение в степень по модулю

К счастью, можно возводить в степень быстрее: начав с x, будем последовательно возводить в квадрат по модулю N:

$$x \ mod \ N \to x^2 \ mod \ N \to x^4 \ mod \ N \to \ldots x^{2^{\lfloor log(y) \rfloor}} mod \ N$$

- Каждое умножение требует времени $O(logN)$, всего же умножений будет $log(y)$.
- Чтобы вычислить $x^y \ mod \ N$, мы просто перемножаем те степени x, которые соответствуют ненулевым позициям в двоичной записи y.

Например:

$$x^{25} = x^{11001_2} = x^{10000_2} \cdot x^{1000_2} \cdot x^{1_2} = x^{16} \cdot x^8 \cdot x^1$$

Данный алгоритм - полиномиальный (от размера входа) по времени.

Алгоритм **ModExp** реализует ту же идею, что и описанное ранее возвежение в квадрат:

$$x^y = \begin{cases} (x^{\lfloor y/2 \rfloor})^2 \text{если x четно,} \\ (x \cdot x^{\lfloor y/2 \rfloor})^2, \text{если x нечетно} \end{cases}$$

функция $\textsc{ModExp}(x, y, N)$
{Вход: n-битовые числа x, $y \geqslant 0$, $N > 0$.}
{Выход: $x^y \bmod N$.}
если $y = 0$: вернуть 1
$z \leftarrow \textsc{ModExp}(x, \lfloor y/2 \rfloor, N)$
если y чётно:
  вернуть $z^2 \bmod N$
иначе:
  вернуть $x \cdot z^2 \bmod N$

- Обозначим через n максимальную длину чисел x, y, N в двоичной записи.
- Алгоритм *ModExp* делает не более n рекурсивных вызовов, на каждом из которых умножаются числа длины не более n (все операции производятся по модулю N и требуют времени $O(n^2)$.
- Значит, общее время работы есть $O(n^3)$.

Алгоритм Эвклида

Алгоритм Эвклида находит НОД – наибольший общий делитель двух чисел.

Правило Эвклида

для любых двух чисел $x \geq 0, y \geq 0$ выполняется равенство:

$$\text{НОД(x,y)} = \text{НОД(x mod y, y)}$$

Правило Евклида позволяет записать элегантный рекурсивный алгоритм:

функция $\textsc{Euclid}(a, b)$
{Вход: целые числа $a \geqslant b \geqslant 0$.}
{Выход: НОД(a, b).}
если $b = 0$: вернуть a
вернуть $\textsc{Euclid}(b, a \bmod b)$

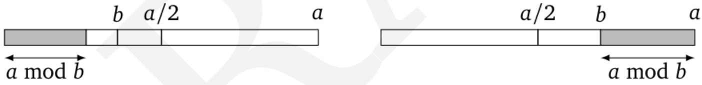
Для оценки времени работы нам нужно понять, как быстро уменьшаются значения аргументов:

- от пары (a, b) алгоритм переходит к паре (b, a mod b),
- то есть большее из чисел (a) заменяется на (a mod b).

Лемма

если $a \geq b$, то $a \ mod \ b < a/2$

Доказательство:



- после двух последовательных рекурсивных вызовов оба числа a и b уменьшатся хотя бы вдвое
- будет произведено не более $2n$ рекурсивных вызовов.
- на каждом из них производится деление, требующее квадратичного времени, общее время работы будет $O(n^3)$.

Расширенный алгоритм Эвклида

Как проверить, что некоторое число d является НОД(a,b)?

Лемма

Если d делит оба числа a и b, а также $d = ax + by$ для некоторых целых чисел x и y, то $d = (a, b)$

Как найти коэффициенты x, y?

функция $\textsc{ExtendedEuclid}(a, b)$
{Вход: целые числа $a \geqslant b \geqslant 0$.}
{Выход: целые числа x, y, d, для которых $d = \text{НОД}(a, b)$ и $ax + by = d$.}
если $b = 0$: вернуть $(1, 0, a)$
$(x', y', d) \leftarrow \textsc{ExtendedEuclid}(b, a \bmod b)$
вернуть $(y', x' - \lfloor a/b \rfloor y', d)$

Пример. Вычисление НОД(25,11) по алгоритму Эвклида:

$$\underline{25} = 2 \cdot \underline{11} + 3 \tag{1}$$
$$\underline{11} = 3 \cdot \underline{3} + 2 \tag{2}$$
$$\underline{3} = 1 \cdot \underline{2} + 1 \tag{3}$$
$$\underline{2} = 2 \cdot \underline{1} + 0 \tag{4}$$

Таким образом:

НОД(25, 11) = НОД(11, 3) = НОД(3, 2) = НОД(2, 1) = НОД(1, 0) = 1.

Для нахождения x и y, для которых $25x + 11y = 1$, мы должны сначала представить 1 как линейную комбинацию чисел последней пары (1, 0).

$$1 = \underline{1} - \underline{0}$$

После этого мы возвращаемся и представляем его как линейную комбинацию чисел в парах (2, 1), (3, 2), (11, 3) и (25, 11).

Чтобы выразить 1 через числа пары (2, 1), мы заменяем 0 на его выражение из предыдущего шага $0 = 2 - 2 \cdot 1$, получается

$$1 = \underline{1} - (\underline{2} - 2 \cdot \underline{1}) = -1 \cdot \underline{2} + 3 \cdot \underline{1}$$

Далее вспоминаем, что $1 = 3 - 1 \cdot 2$, и продолжаем:

$$1 = -1 \cdot \underline{2} + 3 \cdot \underline{1} = -1 \cdot \underline{2} + 3 \cdot (\underline{3} - 1 \cdot \underline{2}) = 3 \cdot \underline{3} - 4 \cdot \underline{2}$$

Поскольку $2 = \underline{11} - 3 \cdot \underline{3}$ и затем $3 = \underline{25} - 2 \cdot \underline{11}$, то

$$1 = 3 \cdot \underline{3} - 4 \cdot (\underline{11} - 3 \cdot \underline{3}) = -4 \cdot \underline{11} + 15 \cdot \underline{3} = -4 \cdot \underline{11} + 15 \cdot (\underline{25} - 2 \cdot \underline{11}) = 15 \cdot \underline{25} - 34 \cdot \underline{11}.$$

Деление по модулю

Мультипликативно обратное число

Число x называется **мультипликативно обратным** к a по модулю N, если $ax \equiv 1 (mod\ \ N)$.

- если мультипликативно обратное число существует, то оно единственно;
- если a и N имеют общий делитель, то есть $d = (a, N) > 1$, то a не имеет обратного по модулю N.

Расширенный алгоритм Евклида найдёт такие x и y что $ax + Ny = 1$, из чего следует, что $ax \equiv 1 (modN)$.

Число x, таким образом, будет обратным к a по модулю N.

Пример нахождения мультипликатвно обратного числа

Пример: найти $11^{-1} mod\ 25$. Используя расширенный алгоритм Евклида, находим, что

$$15 \cdot 25 - 34 \cdot 11 = 1$$

Переходя к модулю 25, получаем:

$$-34 \cdot 11 \equiv 1 (mod\ 25)$$

Поэтому $-34 = 16\ mod\ 25$ является обратным к 11 по модулю 25.

Обращение по модулю N

- У числа a есть обратное по модулю N тогда и только тогда, когда a и N взаимно просты.
- Когда обратное существует, оно может быть найдено за время $O(n^3)$ (где n, как обычно, обозначает размер входных данных) расширенным алгоритмом Евклида.

Проверка чисел на простоту

Числа 7, 17, 19, 71, 79 являются простыми, но как насчёт числа 717197179?

Сократить перебор кандидатов на роль делителей:

- перебираем только простые делители;
- перебираем делители от 2 до $\sqrt{N}$

Существует ли эффективный алгоритм проверки числа на простоту? Нет - быстро разлагать числа на множители пока никто не умеет.

Современная криптография основана на следующей важной **идее**: раскладывать число на множители трудно, в то время как проверить число на простоту легко.
Но проверка эта не даёт делителей, когда оно оказывается составным!

Малая теорема Ферма

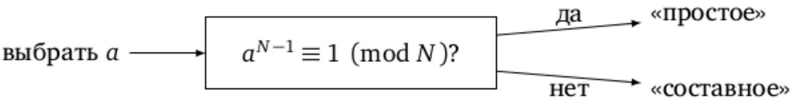
Для любого простого числа p и любого $1 \leq a \leq p-1$ выполнено сравнение

$$a^{p-1} \equiv 1 (mod\ \ p)$$

.

Эта теорема позволяет установить, что число p составное, не находя его делителей: достаточно убедиться, что $a^{p-1} \not\equiv 1(mod\ \ p)$ при некотором $a = 1, 2, \dots, p-1$.

Может быть, можно проверять простоту чисел с её помощью?



Тест Ферма

Условие в теореме Ферма не является необходимым (хотя является достаточным): теорема ничего не говорит про случай, когда N составное.

Пример: $341 = 31 \cdot 11$ составное, но в то же время $2^{340} \equiv 1 (mod\ 341)$.

Можно всё же надеяться, что для составных N таких значений a не очень много.

```
функция PRIMALITY(N)
{Вход: положительное целое число N.}
{Выход: да/нет.}
взять случайное целое число a от 1 до N − 1
если a^(N−1) ≡ 1 (mod N):
  вернуть «да»
иначе:
  вернуть «нет»
```

- Существуют составные числа N, которые проходят тест Ферма для всех a (взаимно простых с N).
- Эти числа наш алгоритм сочтёт простыми. Их называют числами Кармайкла (Carmichael numbers).
- Числа Кармайкла встречаются довольно редко.
- На остальных числах (кроме чисел Кармайкла) наш алгоритм работает довольно хорошо.
- Составное число, не являющееся числом Кармайкла, не проходит тест хотя бы при одном значении a. Такие числа не проходят тест **для половины** всех возможных значений a.

```
функция PRIMALITY2(N)
{Вход: положительное целое число N.}
{Выход: да/нет.}
взять k случайных целых положительных чисел a_1, a_2, ..., a_k < N
если a_i^(N-1) ≡ 1 (mod N) для всех i = 1, 2, ..., k:
  вернуть «да»
иначе:
  вернуть «нет»
```

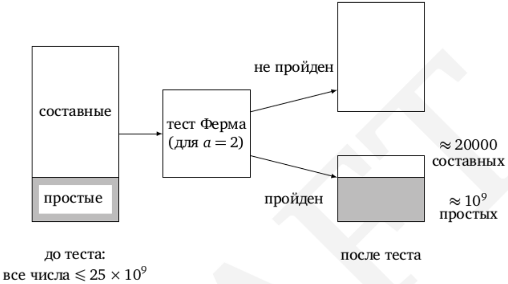
Генерация случайных простых чисел

Закон распределения простых чисел

Обозначим через $\pi(x)$ количество простых чисел, не превосходящих x.

Тогда $\pi(x) \approx x/ln(x)$ или, более формально

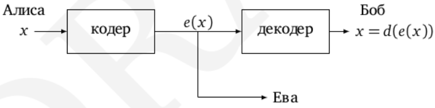
$$\lim_{x\to\infty} \frac{\pi(x)}{x/ln(x)} = 1$$



Криптография

Алиса (A) и Боб (B), которые хотят переговорить без свидетелей, а любопытная Ева (E), которая их подслушивает.

- Пусть, скажем, Алиса хочет послать Бобу секретное сообщение – строку битов x.
- Она шифрует x с помощью функции $e(\cdot)$ и посылает Бобу зашифрованное сообщение $e(x)$.
- Боб использует функцию $d(\cdot)$, чтобы восстановить (дешифровать) исходное сообщение: $d(e(x)) = x$.



Алиса и Боб опасаются, как бы Ева не подслушала $e(x)$. Они надеются, что Ева, не знающая алгоритма расшифровки d, не сможет извлечь никакой информации об x из $e(x)$.

Криптография с закрытым ключом

- Алиса и Боб встречаются заранее и выбирают ключ – битовую строку r той же длины, что и будущее сообщение.
- Шифрование сообщения x состоит в побитовом сложении x с ключом r.

Каждый бит строки e(x) равен сумме по модулю два соответствующих битов x и r: $e_r(x) = x \otimes r$.

Пример: если выбран ключ $r = 01110010$, то сообщение $x = 11110000$ кодируется как

$$e_r(11110000) = 11110000 \otimes 01110010 = 10000010.$$

Функция e_r обратна к самой себе:

$$e_r(e_r(x)) = (x \otimes r) \otimes r = x \otimes (r \otimes r) = x \otimes \overline{0} = x$$

здесь $\overline{0}$ –строка из одних нулей.

Дешифрование просто повторяет шифрование:

$$d_r(y) = y \otimes r$$

Криптография с закрытым ключом

Как надо выбирать r, чтобы эта схема была надёжной?

Простой способ: подбросить монетку столько раз, сколько битов в r, и записать результаты бросаний, при этом все варианты (все элементы множества $\{0,1\}^n$) равновероятны, и прибавление к ним любого сообщения x перемешивает их, оставляя равновероятными, так что Ева (не знающая r) ничего не узнает.

Конечно, эту схему можно использовать только однажды (отсюда и название).

Если послать таким способом два сообщения x и z, то Ева сможет восстановить $x \otimes z$, поскольку

$$(x \otimes r) \otimes (z \otimes r) = (x \otimes z) \otimes (r \otimes r) = x \otimes z \otimes \overline{0} = x \otimes z$$

Криптография открытым ключом

Схема шифрования RSA является протоколом **с открытым ключом**.

- Если Боб хочет получать конфиденциальные сообщения, он публикует открытый ключ, с помощью которого любой желающий может шифровать посылаемые ему сообщения.
- А кроме этого, у Боба есть закрытый ключ, который нужен для расшифровки и который известен только ему.
- Ева его не знает и потому расшифровать ничего не может (если только она не научилась быстро раскладывать числа на множители).

Сообщения будем считать числами по модулю N (сообщения большей длины можно разбить на части). Шифрование будет перестановкой множества $\{0, 1, \ldots, N-1\}$, а дешифрование – обратной перестановкой.

Криптография открытым ключом

Свойство

Пусть N – произведение двух простых чисел p и q, а e взаимно просто с $(p-1)(q-1)$.

Тогда:

1. Отображение $x \to x^e mod\ N$ является перестановкой остатков по модулю N (множества $\{0, 1, \ldots, N-1\}$).
2. Обратной перестановкой будет возведение в степень d, где d – обратное к e число по модулю $(p-1)(q-1)$:

$$(x^e)^d \equiv x(mod\ N)$$

при всех $x = 0, 1, \ldots, N-1$.

Криптография открытым ключом

- Первое свойство гарантирует, что при шифровании $x \to x^e mod\ N$ информация не теряется. Опубликовав пару (N, e) как открытый ключ, Боб позволяет всем желающим выполнять операцию шифрования.
- Второе свойство позволяет быстро расшифровать сообщение, зная закрытый ключ d.
- Зная только открытый ключ e, можно расшифровать сообщение перебором всех вариантов, но при большом N это долго.

Криптография открытым ключом

Пример:

- Пусть $N = 55 = 5 \cdot 11$.
- Возьмём $e = 3$; это значение e допустимо, так как НОД$(e, (p-1)(q-1)) = text(3, 40) = 1$.
- В качестве d надо взять $3^{-1} mod\ 40 = 27$.
- Теперь кодом любого сообщения x будет $y = x^3 mod\ 55$, а декодирование выполняется так: $x = y^{27} mod\ 55$.
- Например, если $x = 13$, то $y = 13^3 mod\ 55 = 52$, а $13 = 52^{27} mod\ 55$.

Протокол RSA

Боб выбирает открытый и закрытый ключи

- Сначала он выбирает два случайных простых числа p и q длиной n бит каждое.
- Открытый ключ Боба — это пара (N, e), где $N = pq$, а e — взаимно простое с $(p-1)(q-1)$ число. Часто в качестве e выбирают число 3, чтобы упростить кодирование.
- Закрытый ключ Боба — число d, обратное к e по модулю $(p-1)(q-1)$. Боб знает p и q и может найти d с помощью расширенного алгоритма Евклида.

Шифрование сообщения x

- Используя открытый ключ Боба (N, e), Алиса посылает Бобу число $y = x^e \bmod N$ (алгоритм возведения в степень по модулю мы разбирали).

Дешифрование

- Получив y, Боб вычисляет $x = y^d \bmod N$.

Протокол RSA

Надёжность протокола RSA основана на таком предположении:

- Зная N, e и $y = x^e modN$, нереально найти x.
- Данное предположение выглядит правдоподобно (хотя не доказано).
- В самом деле, Ева могла бы перебирать все возможные x (и проверять равенство $x^e \equiv y(mod\ \ N)$), но это экспоненциально долго.
- Другой вариант: Ева могла бы разложить N на множители, после чего вычислить d (как это делает Боб), но разложение числа на множители кажется вычислительно трудной задачей.

Парадоксальное обстоятельство: обычно для практики полезен быстрый алгоритм решения какой-то задачи; здесь же нам важно отсутствие быстрого алгоритма разложения на множители.