

Простые числа

A. Тест Ферма

Малая теорема Ферма гласит: если число n простое, то для любого числа $1 \leq a \leq n-1$ выполнено сравнение

$$a^{n-1} \equiv 1 \pmod{n}$$

Отсюда следует, что если мы найдём такое a , для которого это сравнение неверно, то число n заведомо не простое.

Такая проверка простоты числа называется *тестом Ферма*. Будем говорить, что число a проходит тест Ферма для числа n , если указанное сравнение для него верно и не проходит тест Ферма в противном случае.

Даны два натуральных числа: a и n . Напишите функцию `Fermat_Test(a, n)`, возвращающую `True`, если число a проходит тест Ферма для n , иначе вывести `False`.

Не забудьте, что операцию возведения в степень следует выполнять по модулю n .

Input	Output
2 5	True
65537 4295098369	False

B. Тест Ферма - 2

Дано число n , $1 < n < 10^5$. Выведите в первой строке слово `Prime`, если число n простое, и `Composite` иначе. Выведите во второй строке количество чисел от 1 до $n-1$, проходящих тест Ферма для числа n .

Input	Output
5	Prime 4
5555	Composite 8

C. Качество теста Ферма

Если число n — простое, то любое число пройдёт тест Ферма.

- Докажите, что если число n — составное, то все числа не взаимно простые с n тест не пройдут.
- Кроме того докажите, что если найдётся одно число, взаимно простое с n , которое не проходит тест Ферма, то хотя бы половина чисел, взаимно простых с n его не пройдут.

Таким образом, если число n составное, то случайное число из диапазона от 2 до $n-1$ с вероятностью $\frac{1}{2}$ не пройдёт тест Ферма.

Однако это утверждение не совсем верно, см. задачу D.

D. Числа Кармайкла

Числом Кармайкла называется всякое составное число n , которое удовлетворяет сравнению $a^{n-1} \equiv 1 \pmod{n}$ для всех целых a , взаимно простых с n .

В 1994 году доказали, что чисел Кармайкла бесконечно много, более того, при $n \gg 0$ среди чисел от 1 до n по крайней мере $n^{2/7}$ кармайкловых чисел. Из-за этого тест Ферма работает не идеально, хотя вероятность встретить число Кармайкла с ростом n стремится к нулю.

Даны числа a, b ($1 \leq a \leq b \leq 10^5$). Выведите все числа Кармайкла на отрезке $[a, b]$ или пустую строчку, если их нет.

Input	Output
1000 2000	1105 1729

E. Числа Кармайкла и невероятные совпадения

Напишите программу, которая вычисляет два числа:

- наименьшее число, представимое в виде суммы двух квадратов четырьмя способами
- наименьшее число, представимое в виде суммы двух кубов двумя способами

Input	Output
	12345 67890

Ф. Тест простоты Ферма

Тест простоты, основанный на тесте Ферма, заключается в следующем. Пусть мы хотим проверить простоту числа n . Зафиксируем натуральное число m , выберем m случайных чисел от 2 до $n - 1$, и для каждого проведём тест Ферма из задачи А. Если хотя бы одно число не прошло тест, то число n заведомо составное. Если же все тесты пройдены, то либо число n — число Кармайкла, либо с вероятностью по крайней мере $\frac{1}{2^m}$ является простым.

Среди чисел от 1 до 10^9 существует 50847534 простых числа и всего 646 чисел Кармайкла. То есть вероятность наткнуться на число Кармайкла всего $\frac{646}{50847534} \approx 1.27 \cdot 10^{-5} \approx \frac{1}{100000}$. При выборе $m = 20$ вероятность того, что составное число, не являющееся числом Кармайкла, пройдёт тест, будет равна $\frac{1}{2^{20}} \approx \frac{1}{1000000}$.

Напишите функцию `Fermat_Primality_Test(n)`, проверяющую простоту числа при помощи 30 раундов теста Ферма. Функция должна возвращать `False`, если число составное, и `True`, если число скорее всего простое. Число может содержать до 400 десятичных знаков.

Для того, чтобы выбрать случайные числа из диапазона, используйте функцию `randint(a, b)` из модуля `random`, возвращающую случайное число из отрезка $[a, b]$.

Input	Output
170141183460469231731687303715884105721	False
170141183460469231731687303715884105727	True

Г. Свидетели простоты в тесте Миллера-Рабина

Тест Миллера-Рабина является усовершенствованием теста Ферма. Числа Кармайкла не проходят тест Миллера-Рабина, поэтому он является универсальным.

Возьмём целое нечётное $n > 1$ и выделим максимальную степень двойки из числа $n - 1$, получим: $n - 1 = 2^e k$ где $e \geq 1$ и k нечётное. В дальнейшем мы будем использовать e и k именно в таком смысле.

Будем раскладывать многочлен $x^{n-1} - 1 = x^{2^e k} - 1$ на множители до тех пор, пока показатель степени x это чётное число:

$$\begin{aligned}
 x^{2^e k} - 1 &= (x^{2^{e-1} k})^2 - 1 \\
 &= (x^{2^{e-1} k} - 1)(x^{2^{e-1} k} + 1) \\
 &= (x^{2^{e-2} k} - 1)(x^{2^{e-2} k} + 1)(x^{2^{e-1} k} + 1) \\
 &\vdots \\
 &= (x^k - 1)(x^k + 1)(x^{2k} + 1)(x^{4k} + 1) \dots (x^{2^{e-1} k} + 1).
 \end{aligned}$$

Если n простое и $1 \leq a \leq n - 1$, то $a^{n-1} - 1 \equiv 0 \pmod n$ по малой теореме Ферма, поэтому из полученного разложения на множители следует

$$(a^k - 1)(a^k + 1)(a^{2k} + 1)(a^{4k} + 1) \dots (a^{2^{e-1} k} + 1) \equiv 0 \pmod n.$$

Число a называется *свидетелем простоты* числа n , если либо $a^k \equiv 1 \pmod n$, либо в последовательности $a^k, a^{2k}, \dots, a^{2^{s-1} k}$ встречается -1 по модулю n .

Иными словами, нам надо проверить, что ряд значений $a^k, a^{2k}, a^{4k}, \dots, a^{2^{e-1} k}$ по модулю n выглядит вот так:

$$(1, \dots)$$

или вот так:

$$(\dots, -1, \dots)$$

Даны два натуральных числа a и n ($1 \leq a < n \leq 10^{400}$).

Напишите функцию `Miller_Rabin_Test(a, n)`, возвращающую `True`, если число a является свидетелем простоты для n , в противном случае возвращающую `False`.

Input	Output
2 5	True
65537 4295098369	False

Н. *Количество свидетелей простоты*

Дано число N ($3 \leq N \leq 10^5$). Выведите в первой строке слово **Prime**, если число N простое, и **Composite** иначе. Выведите во второй строке количество чисел от 2 до $N - 2$, являющихся свидетелями простоты числа N .

Input	Output
65537	Prime 65534
65533	Composite 4

И. *Тест простоты Миллера-Рабина*

Пусть мы хотим проверить простоту нечётного числа n . Зафиксируем натуральное число m , выберем m случайных чисел от 2 до $n - 1$. Если хотя бы одно из чисел не является свидетелем простоты, то число n заведомо составное. Иначе число с вероятностью по крайней мере $1 - \frac{1}{4^m}$ является простым.

Напишите функцию `Miller_Rabin_Primality_Test(n)`, реализующую эту идею. В качестве числа m используйте битовую длину числа n (какова вероятность того, что проверив все m -битовые числа, мы хоть раз ошибёмся?). Функция должна возвращать **False**, если число n составное, и **True**, если число n скорее всего простое. Число может содержать до 400 десятичных знаков.

Input	Output
170141183460469231731687303715884105721	False
170141183460469231731687303715884105727	True

Ж. *Ближайшее простое число*

Дано число n . Найдите минимальное простое число, большее либо равное n . Число n может содержать до 400 десятичных знаков.

Input	Output
1000000000000000000	1000000000000000003

К. *Случайные простые числа*

Даны числа k и n . Выведите n случайных простых чисел битовой длины k .

Для генерации больших случайных простых чисел часто используют следующий алгоритм:

- При помощи решета Эратосфена получаем список простых чисел от 2 до 65537;
- Берём случайное число от 2^{k-1} до $2^k - 1$;
- Если оно делится на одно из простых от 2 до 65537, то переходим к предыдущему пункту;
- Иначе проводим k тестов Миллера-Рабина;
- Если число не прошло хотя бы один тест, то переходим к пункту *b*, иначе принимаем данное случайное число как простое;
- Повторяем пункты *b* – *e* до тех пор, пока не накопится необходимое количество простых чисел.

Для ускорения рекомендуется выполнить некоторые оптимизации, чтобы исключить дублирование. Даже арифметические операции с большими числами занимают много времени.

Input	Output
7 5	113 67 107 103 107
50 5	574328689204999 1093469055310991 934759628998883 763430397828173 985863319402817

Л. *Теорема Рабина о свидетелях простоты*

Докажите, у составного числа n не более $\frac{n-1}{4}$ свидетелей простоты.

М. *Числа Мерсенна*

Числа Мерсенна — это числа вида $M_n = 2^n - 1$, где n — натуральное число. Докажите, что если число Мерсенна M_n является простым, то число n также является простым.

Н. Тест Люка-Лемера (Lucas-Lehmer primality test)

Оказывается, что для проверки простоты чисел Мерсенна существуют крайне эффективный алгоритм. Он называется тестом простоты Люка-Лемера, благодаря которому простые числа Мерсенна давно удерживают лидерство, как самые большие известные простые числа. На январь 2015 года самым большим известным человечеству простым числом является число $M_{74207281}$, в котором 22338618 десятичных знаков. Это простое число было обнаружено 7 января 2016 года. Самостоятельно найдите описание алгоритма, разберитесь и реализуйте его.

Напишите программу, которая по числу k вычисляет k -е простое число Мерсенна. Гарантируется, что это число не превосходит M_{3000} .

Input	Output
1	3
10	618970019642690137449562111

О. ρ -алгоритм Полларда для поиска делителей

Разложение числа на множители называется факторизацией. Мы уже убедились в том, что очень легко проверять число на простоту и находить большие простые числа. Задача факторизации наоборот, оказывается крайне сложной.

Обычно любое разложение на множители начинается с поиска нетривиального делителя. В этой задаче мы изучим красивый и очень простой в реализации алгоритм поиска делителя составного числа.

Пусть n — нечётное составное число, а p — его наименьший простой делитель.

Наивный алгоритм потребует порядка p операций для его нахождения. Пусть, например, число n содержит 30 десятичных цифр и является произведением двух больших простых чисел (примерно по 15 знаков в каждом). Наивному алгоритму потребуется порядка 10^{15} операций. Если выполнять по 10^9 операций в секунду, то на поиск делителя уйдёт порядка 11 дней.

Оказывается можно найти делитель за время порядка \sqrt{p} (правда, не обязательно наименьший). Если в нашем примере 10^9 операций в секунду заменить на более реальные 10^6 операций в секунду для Python, то потребуется порядка 30 секунд.

Основная идея алгоритма прячется за идеей парадокса дней рождения: если в группе хотя бы 23 человека, то с вероятностью хотя бы 50% у двух из них совпадают дни рождения. Никакого парадокса в этом утверждении нет, однако утверждение может показаться неочевидным, так как вероятность совпадения дней рождения двух человек в любой день года ($1/365 = 0,27\%$), помноженная на число человек в группе из 23, даёт лишь $23/365 = 6,3\%$. Весь секрет в том, что для совпадения дней рождения необходимо совпадения хотя бы в одной из пар чисел, а их не 23, а $\frac{23 \cdot 22}{2} = 253$. А для 50% вероятности совпадения только номера дня в месяце достаточно всего 7 человек.

Перейдём к описанию алгоритма: рассмотрим последовательность случайных чисел x_1, x_2, \dots, x_k . Будем смотреть на неё как на последовательность чисел по модулю n , и как на последовательность чисел по модулю p (пока нам неизвестному). Понятно, что вероятность того, что два числа совпадут по модулю p больше, чем вероятность совпадения по модулю n .

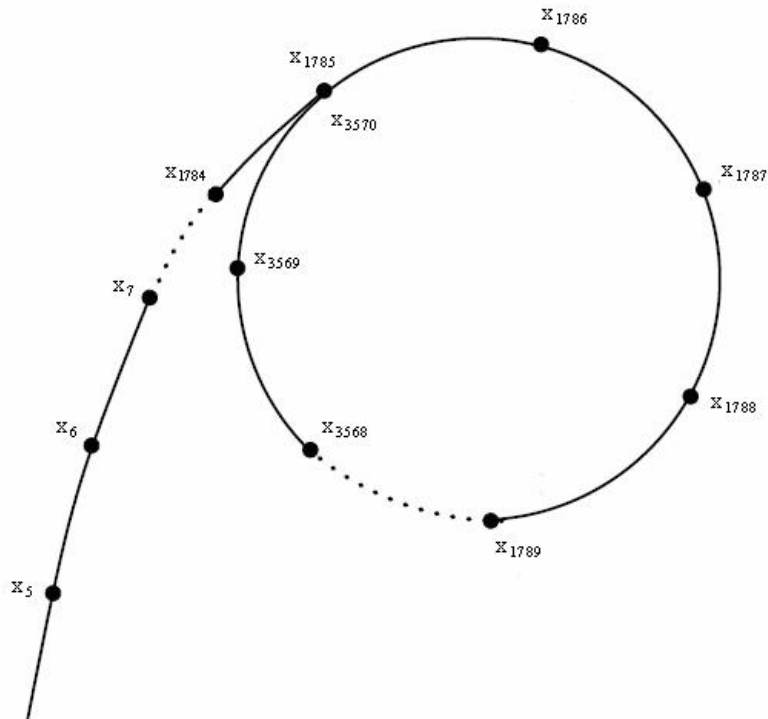
Пусть числа x_i и x_j совпали по модулю p , но не по модулю n . Тогда $(x_i - x_j)$ делится на p и НОД $(x_i - x_j, n)$ делится на p , но не совпадает с n , то есть мы нашли делитель числа n .

Однако пока эта идея в реальности работает плохо, так как нужно слишком часто считать НОД (порядка квадрата количества случайных чисел). Для дальнейшей оптимизации требуется идея, от которой алгоритм и получил имя ρ -алгоритм.

Рассмотрим какую-нибудь функцию $F(x)$ такую, что её, во-первых, можно легко вычислять, во-вторых, для любого числа p если $x \equiv y \pmod{p}$, то $F(x) \equiv F(y) \pmod{p}$, в-третьих последовательность чисел $F(F(F(\dots F(x) \dots)))$ выглядит достаточно случайной, и наконец $F(x)$ не взаимно однозначна. Джон Поллард использовал $F(x) = (x^2 - 1) \pmod{n}$, сейчас в реализациях часто используют также $F(x) = (x^2 + 1) \pmod{n}$.

Выберем в качестве x_1 какое-нибудь случайное число, и будем далее строить последовательность по правилу $x_k = F(x_{k-1})$. Пусть на каком-то очередном шаге после добавления числа x_i возникло совпадение остатка по модулю p с более ранним числом x_s . Тогда для любого натурального t имеем $x_{i+t} \equiv x_{s+t} \pmod{p}$. Если нарисовать картинку, то как раз получится греческая буква ρ .

Теперь воспользуемся идеей Роберта Флойда для экономного поиска цикла из буквы ρ . Для этого заметим, что найдется такое число w , не превосходящее i , что $x_w \equiv x_{2w} \pmod{p}$ (покажите



это, явно указав такое w). Поэтому мы можем взять две равных переменных, к первой последовательно применять $F(x)$, а ко второй — $F(F(x))$, до тех пор, пока не возникнет совпадения по модулю какого-нибудь делителя (которое мы обнаружим НОД'ом), либо по модулю самого числа n , если не повезло. Отсюда уже следует наш алгоритм:

- В качестве x берём случайное число, затем $y = x$;
- Заменяем x на $F(x)$, а y на $F(F(y))$ (таким образом если $x = x_i$, то $y = x_{2i}$).
- Вычисляем $d = \gcd(x - y, n)$
- Если $d = 1$, то переходим к шагу 2, если $d = n$, то переходим к шагу 1, иначе возвращаем d .

Надо сказать, что в приведённом алгоритме выше есть неточность. В некоторых случаях функция $F(x) = (x^2 + 1) \pmod{n}$ не позволяет найти делитель ни при каком начальном числе. Найдите несколько таких n , что в них общего? При первом заиклиивании рекомендуется заменить 1 на какое-нибудь другое желательное небольшое по модулю число, но не 0, и не 2.

Пример работы алгоритма при факторизации числа 11021:

x	y	НОД(x-y, n)
2	2	1
5	26	1
26	6469	1
677	1770	1
6469	7548	1
1225	4445	1
1770	1984	107

Делитель 107 числа 11021 найден всего за 7 шагов. Вот пример с факторизацией числа 112313779:

x	y	НОД(x-y, n)
2	2	1
5	26	1
26	458330	1
677	47580192	1
458330	97053593	1
39622171	54070390	1
47580192	43379730	1
83156460	62097056	1
97053593	91301546	11909

Реализуйте ρ -алгоритм Полларда для поиска делителей числа в виде функции `PollardsDivisor(n)`, принимающей на вход составное число и возвращающей любой его нетривиальный делитель.

Input	Output
6	3
10403	101
288152667470641644773611607061622753	627076271

P. *ρ -алгоритм Полларда факторизации чисел*

Используя ρ -алгоритм Полларда для поиска делителей и тест простоты Миллера-Рабина, напишите функцию `factor(n)`, возвращающую список простых делителей числа (не забудьте, что бывают чётные числа). Выведите этот список при помощи функции `print`, предварительно его отсортировав.

Input	Output
132	[2, 2, 3, 11]
597131754515728882877009356793407519	[753702137, 873511201, 877672009, 1033402943]

Q. *Целочисленный корень*

Напишите функцию `IntSqrt(n)`, возвращающую целочисленный корень из n , то есть наибольшее натуральное число, квадрат которого не больше n . В числе может быть до 100000 десятичных знаков.

$$x_i + \frac{n}{x_i}$$

В поиске алгоритма поможет последовательность $x_{i+1} = \frac{x_i + \frac{n}{x_i}}{2}$. Нужно разобраться, почему она работает, и как ей эффективно воспользоваться для извлечения целочисленного квадратного корня.

Input	Output
16	4
35	5
98269816438745095196487194932272150644479	313480169131549843236

R. *Метод Ферма поиска делителей*

Метод Ферма позволяет быстро найти делитель нечётного числа n , наиболее близкий к \sqrt{n} . Если таких делителей нет (например, в случае произведения маленького простого и очень большого простого числа), то алгоритм работает ещё хуже, чем наивный перебор. Однако в криптографических алгоритмах часто используются составные числа вида $p \cdot q$, и если p и q окажутся близки друг к другу, то метод Ферма позволит найти делитель быстро.

Идея алгоритма крайне проста: если $n = ab$, то для $x = \frac{a+b}{2}$ и $y = \frac{a-b}{2}$ выполнено равенство $n = x^2 - y^2 = (x-y)(x+y)$. Таким образом, если мы представим n в виде разности квадратов, то найдём нетривиальный делитель.

Дальнейшие подробности алгоритма придумайте сами, задача про целочисленный квадратный корень была неспроста.

Реализуйте метод Ферма для поиска делителей числа в виде функции `FermatDivisor(n)`, принимающей на вход составное нечётное число и возвращающей любой его нетривиальный делитель.

Input	Output
9	3
25	5
1152996944542614893	1073761099

S. *Метод Ферма факторизации чисел*

Используя метод Ферма для поиска делителей и тест простоты Миллера-Рабина, напишите функцию `factor(n)`, возвращающую список простых делителей числа. Выведите этот список при помощи функции `print`, предварительно его отсортировав.

Input	Output
132	[2, 2, 3, 11]
1152996944542614893	[1073761099, 1073792807]