Простые числа

A. *Тест Ферма*

Малая теорема Ферма гласит: если число простое, то для любого числа $1 \leqslant a \leqslant n-1$ выполнено сравнение $a^{n-1} \equiv 1 \pmod{n}$. Отсюда следует, что если мы найдём такое a, что это сравнение не выполнено, то число n заведомо не простое. Такая проверка простоты числа называется *тестом Ферма*.

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. 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.

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 хотя бы одно из чисел, взаимно простых с n не пройдёт тест Ферма, то хотя бы половина чисел, взаимно простых с n его не пройдут.

Таким образом, если число n составное, а число случайное из диапазона от 2 до $n-1$, то с вероятностью $\dfrac{1}{2}$ оно 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 $b^{n-1} \equiv 1 \pmod{n}$ для всех целых b, 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. 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

F. *Тест простоты Ферма*

Тест простоты, основанный на тесте Ферма, заключается в следующем. Пусть мы хот 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 A. 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

G. *Свидетели простоты в тесте Миллера-Рабина*

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

Пусть n — некоторое нечётное простое число. Представим число $n-1$ в виде $n-1 = 2^s \cdot u$, где s натуральное, а u — нечётное. Возьмём любое число a от 2 до $n-1$. Тогда из малой теоремы Ферма следует, что $a^{n-1} = a^{2^s u} \equiv 1 \pmod{n}$. Кроме того, по простому модулю n существуют ровно два квадратных корня из единицы: 1 и -1. Поэтому число $a^{2^{s-1}u}$, являясь квадратным корнем из единицы, равно либо 1, либо -1. Если $a^{2^{s-1}u} \equiv 1 \pmod{n}$, то мы снова можем извлечь корень. Мы будем продолжать это действие до тех пор, пока либо не закончится s, либо очередной корень не будет равен -1.

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

Если некоторое число a не является свидетелем простоты, то число n заведомо составное. Оказывается, что для составного числа n не более $\frac{n-1}{4}$ чисел от 1 до $n-1$ являются свидетелями простоты n.

Даны два натуральных числа a и n. Напишите функцию `Miller_Rabin_Test(a, n)`, возвращающую `True`, если число a является свидетелем простоты для n, а в противном случае `False`.

Input	Output
2 5	True
65537 4295098369	False

H. *Количество свидетелей простоты*

Дано число N. 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

I. *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 $\frac{1}{4^m}$ является простым.

Напишите функцию `Miller_Rabin_Primality_Test(n)`, реализующую эту идею. В качестве числа используйте битовую длину числа n (какова вероятность того, что проверив все k-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

J. *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

K. *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:

(a) При помощи решета Эратосфена получаем список простых чисел от 2 до 65537;

(b) Берём случайное число от 2^{k-1} до $2^k - 1$;

(c) Если оно делится на одно из простых от 2 до 65537, то переходим к предыдущему пункту;

(d) Иначе проводим k тестов Миллера-Рабина;

(e) Если число не прошло хотя бы один тест, то переходим к пункту b, иначе принимаем данное случайное число как простое;

(f) 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

L. *Теорема Рабина о свидетелях простоты*

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

M. *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.

N. *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

O. *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 $\dfrac{23 \cdot 22}{2} = 253$. А для 50% вероятности совпадения только номера дня в месяце достаточно всего 7 человек.

Перейдём к описанию алгоритма: рассмотрим последовательность случайных чисел $x_1, x_2, \ldots, x_k$. 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.

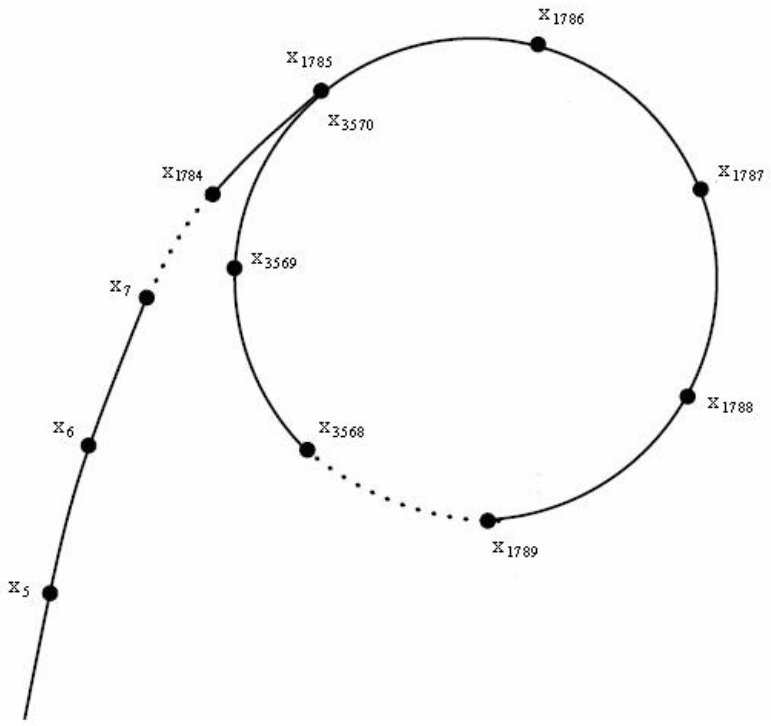
Пусть числа x_i и x_j совпали по модулю p, но не по модулю n. Тогда $(x_i - x_j)$ делится на p и НОД $(xi - xj, n)$ делится на p, но не совпадает с n, то есть мы нашли делитель числа n.

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 $F(F(F(\ldots F(x)\ldots)))$ *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 качекстве 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 $x_{i+t} = x_{s+t} \pmod p$. 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 зацикливании рекомендуется заменить 1 на какое-нибудь другое желательно 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 наименьшее натуральное число, квадрат которого не меньше n. В числе может быть до 100000 десятичных знаков.

В поиске алгоритма поможет последовательность $x_{i+1} = \dfrac{x_i + \dfrac{n}{x_i}}{2}$. 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 $x = \dfrac{a+b}{2}$ и $y = \dfrac{a-b}{2}$ 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.

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]