Факторизация. Проверка на простоту

Пусть задано натуральное число n. Требуется определить, простое ли оно.

Тривиальный алгоритм, вытекающий из определения простого числа, состоит в переборе чисел от 2 до n-1 и поиске среди них делителей числа n. Если делитель найден, то число n составное. Если делителя среди всех рассмотренных чисел нет, то n простое. Процедура использует не более n-2 пробных делений.

Этот метод можно улучшить. В качестве пробных делителей достаточно рассматривать числа, не превышающие корня из n, сократив количество действий.

```
def Factor(n):
    Ans = []
    d = 2
    while d * d <= n:
        if n % d == 0:
            Ans.append(d)
            n //= d
        else:
            d += 1
    if n > 1:
        Ans.append(n)
    return Ans
```

Рассмотрим задачу построения списка простых чисел. Пусть требуется найти k первых простых числа. Переберем все числа в порядке возрастания и проверим каждое из них на простоту. Для проверки будем действовать, как и в прошлом алгоритме, но теперь в качестве делителей будем рассматривать только найденные на данный момент простые числа, не превышающие корня из последнего проверенного числа. Повторяем, пока не найдено k простых чисел.

Такой алгоритм называется решетом Эратосфена. Заметим, что найдя новое простое число, можно вычеркнуть все кратные ему числа. Таким образом, алгоритм состоит из следующих шагов:

1. Выписать подряд все целые числа от двух до n (2, 3, 4, …, n).
2. Пусть переменная pr изначально равна двум — первому простому числу.
3. Вычеркнуть из списка все числа от 2*pr до n, делящиеся на pr (то есть, числа 2pr, 3pr, 4pr, …)
4. Найти первое не вычеркнутое число, большее чем pr, и присвоить значению переменной это число.
5. Повторять шаги 3 и 4 до тех пор, пока pr не станет больше, чем n
6. Все не вычеркнутые числа в списке — простые числа.

Однако в случае очень больших чисел нам не хватит памяти для перебора всех возможных простых чисел – давайте ограничимся задачей поиска ближайшего простого. Для этого вовсе не нужно использовать решето Эратосфена – достаточно пройти линейным поиском от заданной стартовой точки, делая проверку на простоту.

```
def IsPrime(n):
   d = 2
   while d * d <= n and n % d != 0:
       d += 1
   return d * d > n
def Next(n):
   while not IsPrime(n):
       n += 1
   return n
print(Next (10 ** 10))
```

НОД

Говорят, что целое число d является делителем целого числа u , если для некоторого целого q справедливо u = qd . В этом случае также говорят, что u кратно d.

Если некоторое целое число d является делителем одновременно каждого из двух заданных чисел u и v , то d называется их общим делителем.

Среди всех общих делителей двух заданных целых чисел u и v выберем максимальный. Он называется наибольшим общим делителем этих чисел и обозначается НОД(u, v) или gcd(u, v) (сокращение от greatest common divisor). НОД определен, если, по крайней мере, одно из чисел u и v не равно нулю. Обычно полагают НОД(0, 0) = 0.

Алгоритм Евклида

Пусть u – целое, а v – натуральное число. Тогда существуют, причем единственные, целые числа q и r , такие, что u = qv + r , q = u // v , r = u % v .

Алгоритм Евклида основан на последовательном использовании следующего соотношения:

НОД(u, v) = НОД(v, u % v).

Если u и v имеют общий делитель d , то он также является делителем остатка r = u % v . Верно и обратное, что если v и r = u % v делятся на некоторое число d , то на него делится u .

Заметим, что второй аргумент в выражении НОД(v, u % v) всегда меньше первого; поэтому последним шагом алгоритма будет НОД(u, 0) = u.

Для a>=b

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

Быстрое возведение в степень

Бинарное возведение в степень

Заметим, что для любого числа a и чётного числа n выполнимо очевидное тождество (следующее из ассоциативности операции умножения):

$$a^n = \left(a^{\frac{n}{2}}\right)^2 = a^{\frac{n}{2}} \cdot a^{\frac{n}{2}}$$

Оно и является основным в методе бинарного возведения в степень. Действительно, для чётного n за одну операцию умножения можно свести задачу к вдвое меньшей степени.

Если степень n нечётна, перейдём к степени n-1, которая будет уже чётной:

$$a^n = a^{n-1} \cdot a$$

Мы нашли рекуррентную формулу: если степень n чётна, к n/2, а иначе — к n-1.

```
def power(a, n):
    if n == 0:
        return 1
    elif n % 2 == 1:
        return power(a, n - 1) * a
    else:
        return power(a, n // 2) ** 2
```