**Сириус. Теория чисел**

1. **Введение в теорию чисел**

# **Теоретический блок 1: арифметика остатков[1]**

Деление целого  **a** на целое  **b** (**b>0**) с остатком — это представление **a** в виде **b⋅q+r**, где **r** и **q** — целые числа, **0≤r<b**. Число **q** называют неполным частным при делении **a** на **b**, а **r** — остатком деления.

|  |  |  |
| --- | --- | --- |
| A | b | a=b∗q+r |
| 37 | 5 | 37=5∗7+2 |
| −12 | 5 | −12=5∗(−3)+3 |

В языках программирования остаток деления **a** на **b** часто обозначают **%**. Операция нахождения неполного частного деления **a** на **b** — может обозначаться по разному для различных языков. Используя эти две операции, мы можем выполнять деление с остатком при написании программ.

Важно отметить, что если для целого числа **a≥0** операции деления с остатком на целое число **b>0** в языках программирования выполняются в точности с данным выше определением, то при  **a<0**, ситуация сложней. В языке Python эти операции выполняются в соответствии с нашим определением, а в языке C++ нет. В Python выражение **−12%5=3**, а в C++ это же выражение будет равно **−2**. Чтобы остаток от деления всегда соответствовал данному выше определению, можно отдельно разбирать случай при **a<0** или писать выражение для нахождения остатка в виде **(a%b+b)%b**. А чтобы найти неполное частное, можно вычитать найденный остаток из числа **a** и после этого результат делить на **b**.

При этом, в языках программирования разрешено выполнять операции деления с остатком и для **b<0**. Но *мы не рекомендуем производить деление с остатком на отрицательное число*. Всегда можно преобразовать формулы в задаче так, чтобы делить с остатком только на положительное целое число.

Пусть дано целое  **m>1**. Говорят, «**a** сравнимо с **b** по модулю **m»** (пишут **a≡b(mod m))**, если **(a−b) ⋮ m** (**a−b** делится нацело на **m**). Например: **37≡2(mod 5); 37≡7(mod 5); 37≡−3(mod 5)**

Для сравнений верны следующие свойства:

* **a≡a(mod m)** (рефлексивность); **a≡b(mod m)⇒b≡a(mod m)** (симметричность)
* **a≡b(mod m)**, **b≡c(mod m)⇒a≡c(mod m)** (транзитивность)

Пусть **a1≡b1(mod m)** и **a2≡b2(mod m)**, тогда верны следующие сравнения:

* **a1±a2≡b1±b2(mod m); a1a2≡b1b2(mod m); a1n≡b1n(mod m)**, где **n** — натуральное число

Рассмотрим некоторое целое число **m>1**. Любое целое число даёт какой-то остаток при делении на **m**. В рамках данного подхода считаем числа, дающие одинаковый остаток при делении на **m**, — одинаковыми. Тогда получается, что есть лишь конечное число различных чисел. Например, в качестве такого конечного набора остатков можно выбрать целые числа от **0** до **m−1**. Множество остатков по модулю **m** в математике принято обозначать через **Zm**. Так как остатков по модулю **m** конечное число, можно постороить таблицу сложения и умножения для всех возможных пар чисел из **Zm**.

|  |  |
| --- | --- |
| Например, таблица сложения для **Z4**: | Или таблица умножения для **Z5**: |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | + | 0 | 1 | 2 | 3 | | 0 | 0 | 1 | 2 | 3 | | 1 | 1 | 2 | 3 | 0 | | 2 | 2 | 3 | 0 | 1 | | 3 | 3 | 0 | 1 | 2 | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | × | 0 | 1 | 2 | 3 | 4 | | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 1 | 2 | 3 | 4 | | 2 | 0 | 2 | 4 | 1 | 3 | | 3 | 0 | 3 | 1 | 4 | 2 | | 4 | 0 | 4 | 3 | 2 | 1 | |

Из данных таблиц можно сделать вывод о существовании в **Zm** некоторых особых чисел:

* **0** — элемент, при прибавлении к которому любого числа результат равен этому числу **0+x≡x(mod m)**
* **1** — такой элемент, что при умножении его на любое число результат равен этому числу **1⋅x≡x(mod m)**

Обратное число в **Zm** для числа **a** — такое число **a-1**, что **a⋅a-1≡1(mod m)**.

Например: **1−1≡1(mod 5); 2−1≡3(mod 5); 3−1≡2(mod 5); 4−1≡4(mod 5)**

Если **m** — простое число, то у любого числа в **Zm**, кроме числа **0**, существует обратное.

**Задача 1. Вася строит дом**

Вася живёт на координатной прямой, где в точке A располагается школа, а в точке B — любимый Васин компьютерный клуб. Также в точках …,−d,0,d,…,k, где k — целое число, а d — чётное натуральное число, расположены киоски с мороженым. Вася хочет построить дом в некоторой точке с целой координатой. При этом, ему хочется, чтобы расстояние от дома до школы и от дома до компьютерного клуба было одинаковым. Если это вдруг невозможно, то он хочет, чтобы сумма этих расстояний была как можно меньше, а также чтобы расстояния отличались как можно меньше друг от друга. Если подходит несколько вариантов точек, то он выберет ту, расстояние от которой до ближайшего киоска минимально.

Помогите Васе выбрать точку, где строить дом, а также выведите расстояние до ближайшего киоска с мороженым. Вася может строить дом в точке, где уже есть другие строения.

**Входные данные:** В единственной строке заданы три числа — A, B и d. Гарантируется, что A и B — целые числа, по модулю не превышающие 2⋅109, A≠B. d — чётное натуральное число, 2≤d≤2⋅109.

**Выходные данные:** В единственной строке выведите два целых числа — координату точки, где Васе необходимо построить дом, и расстояние до ближайшего киоска с мороженым.

A,B,d=map(int,input().split())

t1=(A+B)//2

if t1\*2!=(A+B):

d1=min(abs(t1)%d,d-abs(t1)%d)

d2=min(abs(t1+1)%d,d-abs(t1+1)%d)

if d1<d2: print(t1,d1)

else: print(t1+1,d2)

else: print(t1,min(d-abs(t1)%d,abs(t1)%d))

# **Теоретический блок 2: Обратное число[2]**

**Малая теорема Ферма**

Пусть дано простое число p и целое число a, не делящееся на p, тогда ap−1≡1(mod p).

Например, при p=5: 14≡1(mod 5); 24≡1(mod 5); 34≡1(mod 5); 44≡1(mod 5)

Докажем, что из теоремы следует утверждение a−1≡ap−2(mod p).

Для этого умножим обе части выражения на a: a−1⋅a≡ap−2⋅a(mod p),что равносильно 1≡ap−1(mod p), что верно по малой теореме Ферма. Заметим, что теперь мы можем легко найти обратное число к a в Zp для простого модуля p. Для этого достаточно возвести a в степень p−2 по модулю p.

Так как число p может быть большим, то для ускорения поиска рекомендуется алгоритм быстрого возведения в степень, что позволит искать обратный элемент по простому модулю за O(log p) операций.

# **Теоретический блок 3: НОД, НОК[3]**

**Основная теорема арифметики**

Пусть дано натуральное число **n>1**. Тогда существует и единственно представление **n** в виде **n=p1α1p2α2...psαs**, где **p1<p2<⋯<ps** — простые числа, а **α1,α2,…,αs** — натуральные числа.

Воспользуемся этой теоремой, чтобы выписать формулы для НОД и НОК через разложения чисел на простые. Пусть нам даны два натуральных числа **a** и **b**. Выпишем их разложения на простые числа по основной теореме арифметики. Наборы простых в разложениях чисел **a** и **b** могут отличаться, поэтому объединим эти наборы в одно множество, в итоге получим:

**a=p1α1⋅p2α2⋯psαs, αi≥0; b=p1β1⋅p2β2⋯psβs, βi≥0**

Тогда для НОД и НОК чисел aa и bb верны следующие формулы:

**gcd(a,b)=pmin(α1,β1)⋅pmin(α2,β2)⋯pmin(αs,βs)**

**lcm(a,b)=pmax(α1,β1)⋅pmax(α2,β2)⋯pmax(αs,βs)**

**Задача 2. Сравнимость по модулю**

Нужно ответить на 1≤t≤105 запросов. Каждый состоит из двух целых чисел 2≤p≤109 и 0<a<p, число p - простое. На каждый запрос вывести в отдельной строке целое число 0<b<p такое, что (a⋅b−1) ⋮ p.

**Входные данные:** В первой строке дано целое число t — количество запросов.

В следующих t строках даны по два числа pi и ai, i=1,…,t.

**Выходные данные:** Выведите t целых чисел (каждое - в отдельной строке) — ответы на запросы.

**Примеры**

|  |  |
| --- | --- |
| Ввод  4  5 1  5 2  5 3  5 4 | Вывод  1  3  2  4 |

**Ограничения:** Время выполнения: 5 секунд

Принцип решения: Быстрое вычисление остатка деления для больших чисел со степенью

def f1(a,n,n0):

if n==0: return 1

elif n%2==0:

t=f1(a,n//2,n0)%n0

return (t\*t)%n0

else: return ((a%n0)\*(f1(a,n-1,n0)%n0))%n0

s=int(input())

for i in range(s):

n,a=map(int,input().split())

print(f1(a%n,n-2,n))

### Задача 3. Отрезок

На клетчатой бумаге нарисовали отрезок, соединяющий точки с координатами (a,b) и (c,d). Через сколько клеток проходит этот отрезок (считается, что отрезок проходит через клетку, если он проходит через её внутренность, а не через вершину или по границе)?

**Входные данные:** Четыре целых числа в одной строке: a,b,c,d. Все по модулю не превосходят 106.

**Выходные данные:** Выведите ответ на задачу.

def f(a,b):

if b==0: return a

return f(b,a%b)

a,b,c,d=map(int,input().split())

if a==c or b==d: print(0)

else:

a=abs(c-a); b=abs(d-b)

t=f(a,b)

print(a+b-t)

# **Теоретический блок 4: Диофантово уравнение[4]**

Рассмотрим следующее уравнение: ax+by=c, где a, b и c — целые числа, a≥0, b≥0 (a или b≠0). Необходимо найти все пары целых чисел x и y, которые являются решением уравнения.

Пусть d=gcd(a,b). Если c не делится на d, то уравнение не имеет решений.

Рассмотрим уравнение:ax0+by0=d. Пусть мы нашли решение для уравнения bx1+(a%b)y1=d

Данное уравнение можно переписать в виде:bx1+(a−⌊ab⌋b)y1=d. Отсюда ay1+b(x1−⌊ab⌋y1)=d

Следовательно x0=y1, y0=x1−⌊ab⌋y1

Таким образом, получаем соотношения для рекурсивного алгоритма поиска решения уравнения. Такой алгоритм называется расширенным алгоритмом Евклида.

Рекурсия здесь будет прекращаться тогда, когда b=0, как и в обычном алгоритме Евклида.

Заметим, что алгоритм найдёт только одно решение уравнения — x0 и y0. Все остальные подходящие пары x и y связаны следующим свойством: x=x0+bdt, y=y0−adt,где t — любое целое число

Вернёмся к изначальному уравнению ax+by=c. Пусть x0 и y0 — решение уравнения ax0+by0=d, тогда x=cdx0+bdt, y=cdy0−adt, где t — любое целое число

**Реализация расширенного алгоритма Евклида**

**def** gcd\_ext(a, b):

**if** b == 0:

**return** a, 1, 0

d, x, y = gcd\_ext(b, a % b)

x, y = y, x - (a // b) \* y

**return** d, x, y

**Задача 4. Диофантово уравнение**

Даны натуральные числа a, b, c. Если уравнение ax+by=c имеет решения в целых числах, то выберите то решение, в котором число x имеет наименьшее неотрицательное значение, и выведите это решение (два числа x и y через один пробел). Если решения не существует, то выведите −1.

**Входные данные:** Натуральные числа a, b и c. Числа заданы одной строкой через пробел и ≤ 109.

**Выходные данные:** Выведите ответ на задачу.

**Примеры. 1)** Ввод: 1 2 3; Вывод: 1 1; **2)** Ввод: 2 2 2; Вывод: 0 1

### Принцип решения:1)Ищем частное решение для X 2)Приближаем к нулю с положительной стороны

### def diofant(a,b):

### if b==0: return a,1,0

### d,x,y=diofant(b,a%b)

### x,y=y,x-(a//b)\*y

### return d,x,y

### a,b,c=map(int,input().split())

### d,x,y=diofant(a,b)

### if abs(c)%d!=0: print(-1)

### else:

### x=x\*c//d; y=y\*c//d

### i=-(x\*d//b)

### if x<0:

### if b>0:

### x+=i\*b//d;y-=i\*a//d

### if x<0:x+=b//d;y-=a//d

### print(x,y)

### else:

### x-=i\*b//d;y+=i\*a//d

### if x<0:x-=b//d;y+=a//d

### print(x,y)

### else:

### if b>0:

### x+=i\*b//d;y-=i\*a//d

### if x-b//d>0:x-=b//d;y+=a//d

### print(x,y)

### else:

### x+=i\*b//d;y-=i\*a//d

### if x+b//d>0:x+=b//d;y-=a//d

### print(x,y)

### Задача 5. Факториал

Для заданного натурального N найти последнюю ненулевую цифру числа N!.

**Входные данные:** Задано число N(0 ≤ N ≤ 1 000 000).

**Выходные данные:** Выведите одну число — последнюю ненулевую цифру факториала числа N.

Принцип решения:1)вычисляя факториал, удаляем последние нули и сокращаем величину числа

n=int(input())

f=1

for i in range(1,n+1):

f=f\*i

while (f%10==0): f//=10

f=f%1000000 #максимальный множитель не превышает 10^6

print(f%10)

**Задача 6.Степень**

Для того чтобы проверить, как ученики умеют считать, Мария Ивановна каждый год задаёт им на дом одну и ту же задачу — для заданного натурального A найти минимальное натуральное N такое, что N в степени N кратно A. От года к году и от ученика к ученику меняется только число A.

Вы решили помочь будущим поколениям, написав программу, решающую эту задачу.

**Входные данные:** Единственное число A (1≤A≤109).

**Выходные данные:** Выведите число N.

**Примеры. 1)** Ввод: 8; Вывод: 4; **2)** Ввод:1; Вывод: 1

Принцип решения: 1) быстрая факторизация A на простые, вычисление их произведения t.

2)Поиск такого наименьшего 1≤k, что N=k\*t кратно A.

def R(n):

i = 2; t=1

while i\*i<=n:

k=0

while n%i==0:

k+=1; n=n//i

if k>0:t\*=i

i+=1

if n>1: t\*=n

return t

A=int(input())

if A==1:print(1)

else:

t=R(A)#произведение всех простых делителей числа

if t>29: print(t) #даже наименьшее целое в разложении - 2 в степени 30 >2^9

else:

tt=t

for k in range(1,30):

tt=k\*t; if (tt\*\*tt)%A==0:break

print(tt)

**II. Теория чисел: Базовые алгоритмы**

# **Теоретический блок 5: Простые числа – функции делителей[5]**

Натуральное число **p** называют простым, если у него **2** различных натуральных делителя: **1** и оно само.

Рассмотрим алгоритм проверки. Пусть дано натуральное **n**. Проверим, является ли оно простым.

**Утверждение**. Если **n** — составное, то у него найдётся натуральный делитель **d>1** такой, что

Рассмотрим некоторые функции, связанные с делителями:

* **σ0(n)** — количество делителей числа **n; σ1(n)** — сумма делителей числа **n**.

Выпишем некоторые соотношения, связанные c **σ0(n)** и **σ1(n)**.

По основной теореме арифметики любое натуральное **n>1** можно едиственным образом представить произведением **n=p1α1p2α2...psαs**, где **p1<p2<⋯<ps** — простые, а **α1,α2,…,αs** — натуральные числа.

Докажем, что **σ0(n)=(α1+1)(α2+1)...(αs+1)**.

Любой делитель **d** числа **n** представим в виде **d=p1β1p2β2...psβs**, где **0≤βi≤αi** для **i** отрезка **[0;s]**. То есть каждое **βi** выбирается **αi+1** способом, тогда из комбинаторики имеем: **σ0(n)=(α1+1)(α2+1)...(αs+1)**.

Выпишем формулу для **σ1(n): σ1(n)=(1+p1+p12+...+p1α1)(1+p2+...+p2α2)...(1+ps+...+psαs)**

Для доказательства достаточно раскрыть скобки и понять, что получается сумма из слагаемых **p1β1p2β2...psβs**, каждое из которых - делитель числа **n**, при этом в сумме будут встречаться все делители.

**Определение**. Функция **f** ***мультипликативна***, если для любых натуральных чисел **a** и **b**, таких что **gcd(a,b)=1**, выполнено **f(ab)=f(a)f(b)**. Заметим, что функции **σ0(n)** и **σ1(n)** мультипликативны.

**Задача 7.** **Проверка числа на простоту**

Дано натуральное  x>1. Проверьте его на «простоту». Вывести YES, если число простое, иначе - NO.

**Входные данные:** Вводится натуральное число, не превосходящее 231.

**Выходные данные:** Выведите ответ на задачу.

def prost(x):

if x==2 or x==3: return 'YES'

else:

i=2

while i\*i<=x:

if x%i==0:return 'NO'

i+=1

return 'YES'

print(prost(int(input())))

**Задача 8.** **Минимальный простой делитель**

Дано целое число, не меньшее 2. Выведите его наименьший простой делитель.

**Входные данные:** Вводится целое положительное число N≤2⋅109.

**Выходные данные:** Выведите ответ на задачу.

def prost(x):

if x==2 or x==3:

return 1

else:

i=2;

while i\*i<=x:

if x%i==0:return 0

i+=1

return 1

def minprost(x):

if x<4: return x

else:

i=2

while i\*i<=x:

if (x%i==0) and (prost(i)==1): return i

i+=1

return x

print(minprost(int(input())))

**Задача 9.** **Числовые функции**

Количество всех натуральных делителей натурального числа n обозначается σ0(n). Сумма всех натуральных делителей числа n обозначается σ1(n).

**Входные данные:** Дано натуральное n≤109.

**Выходные данные:** Выведите σ0(n) и σ1(n).

**Примечание:** Данную задачу рекомендуется решать путём перебора всех делителей числа до .

def R0(n):

i = 2; t=1

while i\*i<=n:

k=0

while n%i==0:

k+=1; n=n//i

if k>0:

t\*=(k+1)

i+=1

if n>1: t\*=2

return t

def R1(n):

i = 1; t=0

while i\*i<=n:

if n%i==0:

t+=i

if i\*i!=n: t+=(n//i)

i+=1

return t

x=int(input())

print(R0(x),R1(x))

# **Теоретический блок 6: Решето Эратосфена[6]**

Пусть нужно проверить на простоту числа от **1** до **n**. Можно проверять каждое за  O(), но есть более быстрый метод — решето Эратосфена. Идея: На отрезке **2…n**  вычеркнем все числа, *кратные* **2**, кроме  **2,** затем кратные **3**, кроме**3** и т.д. Невычеркнутыми останутся простые от **2** до **n**. Алгоритм можно усовершенствовать — вычёркивать все числа, делящиеся на простое **i** начиная от **i2**, поскольку числа от **2i** до **(i−1)i** имеют меньший простой делитель и уже вычеркнуты. Сложность алгоритма — O(n log(log n)).

**Реализация**

prime = [True] \* (n + 1)

prime[0] = prime[1] = False

**for** i **in** range(2, n + 1):

**if** **not** prime[i]:

**continue**

**for** j **in** range(i \* i, n + 1, i):

prime[j] = False

**Задача 10.** **Простое число**

По введённому натуральному числу K, не превосходящему 100000, выдать K-е по счёту простое число.

**Входные данные:** Дано натуральное число K.

**Выходные данные:** Выведите K-е простое число.

n=int(input()); x=[True]\*10000001

x[0]=x[1]=False; c=0; i=0

for i in range(2,10000001):

if x[i]:

c+=1

if c==n:

print(i); break

for j in range(i\*i,10000001,i): x[j]=False

# **Теоретический блок 7: Факторизация числа[7]**

Будем искать разложение  **n** на простые множители, то есть представление  **n** в виде **n=p1α1p2α2...psαs**, где **p1<p2<⋯<ps** — простые, а **α1,α2,…,αs** — натуральные числа (оно всегда есть и единственно).

Будем перебирать в переменной **d** все числа от **2** до  и делить **n** на **d** до тех пор, пока возможно. Заметим, что тогда **n** всегда будет делиться только на простые числа, так как если в переменной **d** записано составное число, то это значит, что до этого мы уже разделили **n** на все простые числа, которые являются делителями **d**. Таким образом, мы нашли все простые делители числа **n**, не превосходящие . Возможно, у нас остался один простой делитель, больший  (если бы **n** содержало два таких делителя, то их произведение было бы больше **n**) – он хранится в  **n** после перебора всех **d**. Сложность: O().

Также полезно *асимптотически оценить количество различных простых* *делителей* **n: p1**,**p2**,...**ps**. Заметим, что деление на каждый делитель уменьшит число **n** минимум в два раза, а значит количество **s** различных простых делителей есть **O(log n)**. Если учитывать каждый простой делитель столько раз, сколько он встречается в разложении, то есть рассмотреть сумму **α1+α2+…+αs**, то и её оценка **O(log n)**.

**Реализация**

**def** factorization(n):

p = []; d = 2

**while** d \* d <= n:

**while** n % d == 0:

p.append(d); n //= d

d += 1

**if** n > 1: p.append(n)

**return** p

**Задача 11.** **Разложение на простые**

Требуется разложить целое число N на простые множители с учётом их степени и вывести результат в порядке возрастания множителей.

**Входные данные:** Программе дано число N(2≤N≤109).

**Выходные данные:** Вывести разложение N на простые множители. Знак возведения в степень: **^**.

**Примеры: 1)**Ввод: 2. Вывод: 2; **2)**Ввод: 1008. Вывод: 2^4\*3^2\*7

def R0(n):

i = 2;c=0

while i\*i<=n:

k=0

while n%i==0: k+=1; n=n//i#считаем количество делителей «2»

if k>0:

if k==1:

if c==0: print(i,sep='',end='')

else: print('\*',i,sep='',end='')

else:

if c==0: print(i,'^',k,sep='',end='')

else: print('\*',i,'^',k,sep='',end='')

c+=1

i+=1

if n>1:

if c==0: print(n)

else: print('\*',n,sep='')

R0(int(input()))

# **Теоретический блок 8: Функция Эйлера[8]**

**Функция Эйлера** от **n** (обозначается **φ(n)**) — это количество чисел от **1** до **n**, взаимно простых с **n**.

Например, **φ(1)=1**. Если **n=p1α1p2α2…psαs**, то **φ(n)=(p1α1−p1α1−1)(p2α2−p2α2−1)…(psαs−psαs−1)**.

Преобразуем формулу:

**φ(n)**=(p1α1−p1α1−1)(p2α2−p2α2−1)…(psαs−psαs−1)=p1α1 p2α2…psαs(1−1/p1)…(1−1/ps)=**n(1−1/p1)(1−1/p2)…(1−1/ps)**

Из первой формулы для функции Эйлера видно, что она мультипликативна. А вторая формула удобна для реализации алгоритма вычисления функции Эйлера.

**Теорема Эйлера:** Для взаимно простых натуральных чисел **m>1** и **a** выполнено **aφ(m)≡1(mod m)**.

Эта теорема является обощением малой теоремы Ферма,что легко проверить, вычиcлив значение функции Эйлера от простого числа **p**: **φ(p)=p−1**.

Теорема Эйлера помогает при поиске обратного для числа aa по составному модулю **m** в **Zm**, если **a** и **m** взаимно простые. Верно соотношение: **a−1≡aφ(m)−1(mod m)**. Если **gcd(m,a)≠1**, то для  **a** нет обратного.

Докажем это. Пусть **d=gcd(a,m)>1**. Предположим, что **a−1** существует, тогда **a−1a≡1(mod m)**. Это значит, что **(a−1a−1) ⋮ m**, откуда и **(a−1a−1) ⋮ d**, однако **a−1a** тогда также кратно **d**, а **1** - нет. Имеем противоречие.

**Задача 12.** **Функция Эйлера *(проверяется теоретически или программно)***

Для пар (a,b) таких, что a=239,1≤b<239, верно ли, что φ(ab)=φ(a)φ(b), где φ — функция Эйлера?

Варианты: 1)Верно для всех пар 2)Неверно для всех пар 3)Верно для некоторых (не всех) пар

**Задача 13.** **Функция Эйлера *(проверяется теоретически или программно)***

Для пар (a,b) таких что a=91,1≤b<91, верно ли, что φ(ab)=φ(a)φ(b), где φ — функция Эйлера?

Варианты: 1)Верно для всех пар 2)Неверно для всех пар 3)Верно для некоторых (не всех) пар

**Задача 14.** **Функция Эйлера *(проверяется теоретически или программно)***

Для пар (a,b) таких, что a≥123, b=a2+1, верно ли, что φ(ab)=φ(a)φ(b), где φ — функция Эйлера?

Варианты: 1)Верно для всех пар 2)Неверно для всех пар 3)Верно для некоторых (не всех) пар

**Задача 15.** **Функция Эйлера**

Дано натуральное n. Определите количество натуральных чисел, меньших n и взаимно простых с ним.

**Входные данные:** Дано натуральное число n≤109. **Выходные данные:** Выведите φ(n).

def Eu(n):

i = 2;R=n

while i\*i<=n:

if n%i==0: R-=R//i

while n%i==0: n//=i

i+=1

if n>1: R-=R//n

return R

print(Eu(int(input())))

**Задача 16.** **Обратное число (*функция Эйлера, НОД, вычисление числа в степени по модулю*)**

Даны два целых числа m и a. Если не существует обратного числа к a по модулю m, то выведите  −1, а если существует, то выведите это число (ответ должен лежать в границах от 0 до m−1).

**Входные данные:** В единственной строке входных данных даны два целых числа 1<m≤109 и 0<a<m.

**Выходные данные:** Выведите ответ на задачу. **Пример.** Ввод: 179 57; Вывод: 22

def gcd\_ext(a, b): #расширенный поиск НОД – нужно только gcd\_ext[0] – для проверки на простоту

if b == 0: return a, 1, 0

d, x, y = gcd\_ext(b, a % b)

x, y = y, x - (a // b) \* y

return d, x, y

def Eu(n):#функция Эйлера

i = 2;R=n

while i\*i<=n:

if n%i==0: R-=R//i

while n%i==0: n//=i

i+=1

if n>1: R-=R//n

return R

def f1(a,n,m): #(a^n)%m (быстрый алгоритм)

if n==0: return 1

elif n%2==0:

t=f1(a,n//2,m)%m

return (t\*t)%m

else: return ((a%m)\*(f1(a,n-1,m)%m))%m

def Eu2(m,a): #головная функция

d=gcd\_ext(m,a)[0]

if d>1: return -1

else: return f1(a,Eu(m)-1,m)

m,a=map(int,input().split())#main

print(Eu2(m,a))

**Задача 17.** **Делители - *неприятная, но полезная задача (по TL)***

Дано натуральное число n. Подсчитайте количество таких пар чисел (a;b), что:

* a и b — делители n; a<b; a и b — взаимно простые; a\*b≤n.

**Входные данные:** Натуральное число n≤108. **Выходные данные:** Количество подходящих пар.

def fact (N) :#быстрый поиск всех делителей N

D = [1]; DD=[N]; d = 2

while (d \* d <= N) :

if N % d == 0 :

D.append(d)

d\_new = N // d

if d\_new != d : DD.insert(0,d\_new)

d += 1

return D+DD

def gcd\_ext(a, b):# НОД

if b == 0: return a, 1, 0

d, x, y = gcd\_ext(b, a % b)

x, y = y, x - (a // b) \* y

return d, x, y

p=fact(int(input())); s=0

for i in range(len(p)-1):#считаем количество пар без самопересечений

for j in range(i+1,len(p)):

if gcd\_ext(p[i],p[j])[0]==1: s+=1

print(s)

**Задача 18.** **Разложение на чётнопростые - *полезная задача (по TL и частным случаям)***

В этой задаче рассматриваются только чётные целые числа. Чётное натуральное число n будем называть чётнопростым числом, если его нельзя представить в виде произведения двух чётных чисел. Например, числа 2 и 6 — чётнопростые. Очевидно, что каждое число либо является чётнопростым, либо разлагается в произведение чётнопростых. Но такое разложение на чётнопростые не всегда единственно.

**Входные данные:** Дано чётное натуральное n≤109.

**Выходные данные: 1.**Если число n чётнопростое, выведите слово **prime**.

**2.**Если это число единственным образом разлагается в произведение *двух и более* чётнопростых, то выведите слово **single**, а в следующей строке - разложение этого числа на чётнопростые множители.

**3.**Если число допускает несколько различных разложений на чётнопростые, то выведите слово **many**, а в следующих двух строках - два каких-нибудь различных разложения числа на чётнопростые множители.

**Примеры. 1)**Ввод: 6; Вывод: prime; **2)**Ввод: 4; Вывод: single На следующей строке: 2 2

def factoriz(n,nn): # число двоек в разложении и общий массив делителей

p = []; d = 2; k=0

while d \* d <= nn:

while n % d == 0:

if d==2: k+=1

p.append(d); n //= d

d += 1

if n > 1:

p.append(n)

if n==2:k+=1

return k,p

def ff(x):

if x%4!=0: print('prime')

else:

x//=4; k,p=factoriz(x,x)

if len(p)==0: print('single'); print(2,2)

elif len(p)-k==0: print('single'); print(2,2,end=' '); for i in range(1,k+1): print(2,end=' ')

elif len(p)-k==1:

print('single'); for i in range(1,k+1): print(2,end=' ')

print(2,2\*p[len(p)-1])

elif len(p)-k>1:

print('many'); for i in range(1,k+1): print(2,end=' ')

t=1; for i in range(k,len(p)-1): t\*=p[i]

print(2\*t,2\*p[len(p)-1]); for i in range(1,k+1): print(2,end=' ')

t=1; for i in range(k,len(p)): t\*=p[i]

print(2,2\*t)

ff(int(input()))

**Задача 19.** **Делители факториала (*простая задача на изученные ранее функции*)**

Для заданного натурального числу N найти количество делителей его факториала N! Например, при N=4, N!=4⋅3⋅2⋅1=24. Его делители: 1,2,3,4,6,8,12,24, то есть их количество составляет 8.

Напишите программу, которая по натуральному Nнаходит количество делителей его факториала.

**Формат входных данных:** Одно целое число N(1≤N≤45).

**Формат выходных данных:** Одно целое число — количество делителей числа N!

import math

def R0(n):

i = 2; t=1

while i\*i<=n:

k=0

while n%i==0: k+=1; n=n//i

if k>0: t\*=(k+1)

i+=1

if n>1: t\*=2

return t

print(R0(math.factorial(int(input()))))

**Задача 20.** **Суперчисла (*идея решения видна по коду, использующего решето Эратосфена*)**

Суперчислом называется число, являющееся суммой двух простых чисел из диапазона [2…B]. Требуется найти все суперчисла из заданного диапазона [A…B].

**Входные данные:** Два числа в одной строке A и B (2≤A≤B≤10000).

**Выходные данные:**Все найденные суперчисла заданного диапазона в возрастающем порядке.

def Erat(n):

prime = [True] \* (n + 1); prime[0] = prime[1] = False

for i in range(2, n + 1):

if not prime[i]:continue

for j in range(i \* i, n + 1, i):prime[j] = False

return prime

A,B=map(int,input().split())

s=[]; q=[]; p=Erat(B)

for i in range(2,B+1):

if p[i]:s.append(i)

for i in range(len(s)):

for j in range(i,len(s)):

if (A<=s[i]+s[j]<=B):q.append(s[i]+s[j])

q.sort(); print(q[0])

for i in range(1,len(q)):

if q[i]!=q[i-1]:print(q[i])

**Задача 21.** **Забавная игра (*массив простых делителей до+поиск максимальной цепочки + учет, что после простой делитель совпадает с числом, т.е. считаем цепочки одинаковых*)**

Вы с друзьями играете в следующую игру. Друзья пишут на доске подряд N натуральных чисел. Задача — найти максимум подряд идущих чисел, которые бы делились на одно и то же число, большее 1. Так как вручную искать ответ сложно, вы решили написать программу, которая сделает работу за вас.

**Входные данные:** В первой строке число N(1 ≤ N ≤ 100000). Во второй строке через пробел N целых чисел A1...AN(1 ≤ Ai ≤ 1000, 1 ≤ i ≤ N), в том же порядке, как они расположены на доске.

**Выходные данные:** Максимум длины цепочки чисел кратных одному и тому же числу >1.

def Erat(n):

prime = [True] \* (n + 1); prime[0] = prime[1] = False

for i in range(2, n + 1):

if not prime[i]:continue

for j in range(i \* i, n + 1, i): prime[j] = False

return prime

n=int(input())

x=[]; p=[]; t=map(int,input().split()); q=Erat(round(1000\*\*0.5)+1)

for i in t: x.append(i)

for i in range(len(q)): if q[i]:p.append(i)

mx=0

for i in range(len(p)):

k=0:

for j in range(len(x)):

if x[j]%p[i]==0: k+=1

else:

if k>mx:mx=k

k=0

if k>mx:mx=k

k=0

for j in range(1,len(x)):

if x[j]==x[j-1]: k+=1

else:

if k>mx:mx=k

k=0

if k>mx: mx=k

print(mx)

# **Приложения: Ссылки на видеолекции (всего около 1,5 часа)**

* 1. Арифметика остатков. К.ф-м.н Мамай И.Б., СУНЦ МГУ,17мин. <https://youtu.be/qm66IGK7UkA>
  2. Обратный элемент по простому модулю. Мамай И.Б., 6мин. <https://youtu.be/L51Aq1MQfs0>
  3. НОД и НОК. Мамай И.Б., 14мин.<https://youtu.be/bxwUlIgCzD8>
  4. Расширенный алгоритм Евклида. Мамай И.Б., 11мин. <https://youtu.be/AModLv5FcjI>
  5. Простые числа, функция делителей. Мамай И.Б., 11мин. <https://youtu.be/zaMhhKIciV0>
  6. Решето Эратосфена. Мамай И.Б., 7мин. <https://youtu.be/BOuqLCFiWkM>
  7. Алгоритмы факторизации числа. Мамай И.Б., 7мин. <https://youtu.be/9Gyk99De4qY>
  8. Функция Эйлера. Мамай И.Б., 11мин. <https://youtu.be/lz8C3Rgacp4>