Number Theory

قابلية القسمة

- قابلية القسمة على 3:
يقبل عدد القسمة على 3 إذا كان مجموع أرقام هذا العدد يقبل القسمة على 3
مثال:

45612 --> 4+5+6+1+2=18
و18 يقبل القسمة على 3.. فيكون 45612 يقبل القسمة على 3

- قابلية القسمة على 9:

يقبل عدد القسمة على 9 إذا كان مجموع أرقامه يقبل القسمة على 9

- قابلية القسمة على 4:

يقبل عدد القسمة على 4 إذا كانت أول خانتين من العدد (الأحاد والعشرات) تكونان عدد يقبل القسمة على 4
4654153168749846572
الآحاد والعشرات هي 72 = 4 * 12

- قابلية القسمة على 11:

يقبل عدد القسمة على 11 إذا كان حاصل طرح مجموع الخانات الزوجية من مجموع الخانات الفردية يقبل القسمة على 11.
مثال:
العدد 918082
مجموع الخانات الفردية = 2+0+1=3
مجموع الخانات الزوجية = 8+8+9= 25
حاصل طرح المجموعين = 25-3 = 22
2*11=22
إذن يقبل عددنا القسمة على 11.

القاسم المشترك الأكبر: GCD

تابع لإيجاده يعتمد على خوارزمية إقليدس

```
int gcd(int a,int b)
{
if(b==0) return a;
return gcd(b,a%b);
}
```

المضاعف المشترك الأصغر : LCM

نعلم أن :

a*b=gcd(a,b)*lcm(a,b)

فنجد أبسط طريقة لحساب المضاعف المشترك الأصغر

Lcm(a,b)=a*b/gcd(a,b)

مسألة إيجاد قواسم عدد

نريد حساب قواسم العدد n وإضافتها إلى vector ما أو مجموعة ما set

يمكن ذلك ببساطة من خلال المرور على كل الأعداد الأصغر من n

```
set<int>divisors;
for(int i=1;i<=n;i++)
{
       If(n%i==0 ) divisors.insert(i);
}
```

ونحتاج هنا إلى n عملية ..
ولكن قد يكون هذا مكلفاً جداً عندما يزداد حجم n

لنلاحظ التالي:
إذا كان

a*b=n

لنفترض أن a أكبر من جذر العدد n
عندها يكون حكما b أصغر من جذر العدد n
وبملاحظة أن كل قاسم ل n أصغر من جذره يقابله قاسم له أكبر من جذره
نجد كفاية المرور على الأعداد الأصغر من جذر n لمعرفة قواسمه كما يلي:

```
set<int> divisors;
for(int i=1;i<=sqrt(n);i++)
{
     If(n%i==0)
      {
              divisors.insert(i);
             divisors.insert(n/i);
      }
```

}

ويوجد مسائل مشابهة من إيجاد العوامل الأولية لعدد أو عددها أو...
ستصادفك في مسائل الماراثون.

مسألة إيجاد الأعداد الأولية بين 1 و n

n<=10^6

لنتبع الطريقة التالية:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

N=27
في البداية مقابل كل عدد 0

نبدأ من العدد 2
نجد مقابله 0 (أي 2 أولي) ثم نقوم بوضع واحد مقابل كل مضاعف للعدد 2:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

يأتي بعد العدد 2 العدد 3 ونجد مقابله 0 فهو أولي
ونقوم بوضع 1 مقابل كل مضاعف للعدد 3

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1

نصل للعدد 4 نجد مقابله 1 فهو إذاً غير أولي (ولا داع لوضع 1 مقابل مضاعفاته لأنها حكما مقابلها 1)

نصل للعدد 5 ومقابله 0 فهو عدد أولي ثم نضع مقابل مضاعفاته 1 (كم تكلفة هذا ؟ n/5)

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0	1	1	1	0	1	1	1	1

ثم نتابع هكذا حتى n... كلما صادفنا عدد مقابله 0 نقوم بوضع 1 مقابل كل مضاعفاته (ما عدا نفسه طبعاً)

ولكن كم تكلفة ذلك؟!

عندما وضعنا 1 مقابل مضاعفات العدد 2 كلفنا ذلك n/2 عملية
عندما وضعنا 1 مقابل مضاعفات العدد 5 كلفنا ذلك n/5 عملية
عندما وضعنا 1 مقابل مضاعفات العدد 11 كلفنا ذلك n/11 عملية

أي لأجل كل عدد أولي x نجده يساهم بحوالي n/x عملية

أي عدد العمليات الكلي يحسب تقريبيا هكذا:

P1حتى Pall يقصد بها الأعداد الأولية فقط بين 1 وn

$$\sum_{k=P1}^{Pall} \frac{n}{k} \le \sum_{i=1}^{n} \frac{n}{i} = n * \sum_{i=1}^{n} \frac{1}{i}$$

و $\sum_{i=1}^{n} \frac{1}{i}$ مقدار صغير جدا مهما ازداد n

حيث نجد عندما n=10^6

يكون $\sum_{i=1}^{n} \frac{1}{i}$ <=15

أي تكلفة معرفة الأعداد الأولية من 1حتى n عندما n<=1000000
هي قرابة 15*n عملية ..

خوارزمية إيجاد الأعداد الأولية:

```
#define ll long long int
bool prime[100100];
void sieve(ll n)
{
    for(int i=0;i<=n;i++) //make all elements 0
        prime[i]=0;
    prime[0]=1; prime[1]=1; //0 and 1 are not prime
    for(ll i=2;i<=n;i++)
        if(prime[i]==0) //i is prime
            for(ll j=2*i;j<=n;j+=i)
                prime[j]=1;
}
```

إيجاد القواسم الأولية لعدد ما n

يتم ذلك بطريقة مشابهة لما سبق وبملاحظة أن أي عدد يكتب كجداء أعداد أولية مرفوعة إلى أسس كما يلي:

$$n = {p_1}^{k_1} \times {p_2}^{k_2} \times {p_3}^{k_3} \times \dots$$

نقوم بالمرور على الأعداد من 1 حتى جذر n، وعندما نصادف عدداً يقسم n سيكون هذه العدد أولياً، نقسم n على هذا العدد حتى يصبح n لا يقبل القسمة عليه ونتابع، بعد انتهاء العملية يمكن أن يكون n>1 وعندها يكون العدد المتبقي n أولياً نضيفه إلى مصفوفة القواسم الأولية:

```
vector<ll>find_divisors(ll x)
{
    vector<ll>divs;
    for(ll i=2;i*i<=x;i++)
    {
        if(x%i==0)
        {
            while(x%i==0)
            {
                x/=i;
            }
            divs.push_back(i);
        }
    }
    if(x>1)
        divs.push_back(x);
    return divs;
}
```

باقي القسمة

يطلب في المسائل عندما يكون الناتج كبيرا لا يمكن طباعته كعدد long long int

خواصه:

- (a + b) % c = ((a % c) + (b % c)) % c
- (a * b) % c = ((a % c) * (b % c)) % c
- (a – b) % c = ((a % c) – (b % c) +c) % c

رفع العدد إلى قوة FAST POWER

مثلا عندما نريد حساب 831^{456}

فإن هذه العبارة تعني ناتج جداء العدد 831 بنفسه456 مرة
أي يمكن حلها بعدد عمليات موافق للأس
ولكن ماذا لو كان الأس عدد كبير جدا؟؟

كيف يمكن تقليل عدد العمليات؟

لنلاحظ:

$$3^{16} = (3^8)^2 = ((3^4)^2)^2 = (((3^2)^2)^2)^2$$

نلاحظ أنه يمكن حساب 3 مرفوع للأس 16 ب 16 عملية ضرب بالعدد 3
ولكن من خلال آخر عبارة نلاحظ أننا نحتاج فقط إلى 4 عمليات ...

مثال آخر:

$$5^9 = 5 \times 5^8 = 5 \times (5^2)^4 = 5 \times ((5^2)^2)^2$$

بدلا من 9 عمليات جداء لدينا 4 عمليات فقط
أي يمكن تحسين تعقيد الحساب من رتبة خطية إلى رتبة لوغاريتمية

```
#define ll long long
ll mod=1000000007;
ll power(ll a,ll n,ll mod) // to compute a^n
{
   if(n==0) return 1;
   if(n%2==0)
   {
       ll ans=power(a,n/2,mod);
       ans%=mod;
       ans*=ans;
       ans%=mod;
       return ans;
   }
  else
   {
       ll ans=power(a,n/2,mod);
       ans%=mod;
       ans*=ans;
       ans%=mod;
       ans*=a;
       ans%=mod;
       return ans;
   }
}
```

Modular Inverse

عندما نريد حساب باقي قسمة عددين A/B حيث A و B عددان كبيران جداً، وB يقسم A، ولكن نعلم باقي قسمة كل من A و B على MOD، يعطى باقي القسمة هذا بالعلاقة:

$$\left(\frac{A}{B}\right)\%MOD = A \times B^{MOD-2}$$

```
ll inverseMod(ll A,ll B,ll mod)
{
    ll ans=power(B,mod-2,mod);
    ans%=mod;
    ans*=A;
    ans%=mod;
    return ans;
}
```

التابع فاي $\text{ø}(n)$

يعطي عدد الأعداد الأولية مع n وأصغر منه.

مثال: عدد الأعداد الأولية مع 5 وأصغر منه: 4 وهي 1،2،3،4 $\quad ø(\mathbf{5}) = 4$

عدد الأعداد الأولية مع 6 وأصغر منه :2 وهي 1،5 $\quad ø(\mathbf{6}) = 2$

يعطى التابع بالعلاقة:

$$ø(\mathbf{n}) = \mathrm{n} \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \ldots \times \left(1 - \frac{1}{p_k}\right)$$

حيث P_i هي القواسم الأولية للعدد n.

أي يكفي إيجاد القواسم الأولية للعدد n وبالتالي يكون تعقيد الخوارزمية هو $\sqrt{n}$ كما رأينا سابقاً.

```
int phi(int n)
{
    int res = n;
    for (int i = 2; i * i <= n; i++)
    {
        if(n % i == 0)
        {
            while(n % i == 0)
                n /= i;
            res -= res / i;
        }
    }
    if(n > 1)
        res -= res / n;
    return res;
}
```

كما يمكن حساب قيمة phi لكل الأعداد من 1 إلى n كما يلي بنفس الطريقة التي اتبعناها في إيجاد الأعداد الأولية:

```
#define ll long long int
ll phi[200];
void computePhi(int n)
{

    for (int i=1; i<=n; i++)
        phi[i] = i;
    for (int p=2; p<=n; p++)
    {
        if (phi[p] == p) // p is prime
        {
            phi[p] = p-1;
            for (int i = 2*p; i<=n; i += p)
            {
               phi[i] = (phi[i]/p) * (p-1); //(1-1/p)
            }
        }
    }
}
```

حيث نلاحظ أن تعقيد الخوارزمية السابقة هو نفس تعقيد تابع إيجاد الأعداد الأولية.

ملاحظة: لاحظ أنه إذا كان n عدد أولي فإن $\text{ø}(\mathbf{n}) = \mathrm{n} - 1$

مبرهنة اويلر:

ليكن x و m أوليين فيما بينهما فيكون :

$$x^{\Phi(m)} \bmod m = 1$$

و إذا كان m أولي أي $\Phi(m)=m-1$ نحصل على مبرهنة فيرما
: Fermat’s theorem

$$X^{m-1} \bmod m = 1$$

مثلا $m=5$ و $x=2$:

$$2^4 \bmod 5 = 1$$

خوارزمية اقليدس المعممة extended GCD:

تفيد في ايجاد قيم x و y التي تحقق:

ax + by = gcd (a, b)

مثال:

a = 30, b = 20
gcd(a,b) = 10

فيكون:

x = 1, y = -1
(30*1 + 20*(-1) = 10)

كما في خوارزمية حساب القاسم المشترك الاكبر الاساسية، يتم حساب gcd(a,b) بالاعتماد على القيم المحسوبة عوديا بعد استدعاء gcd(a,b%a)، لتكن x1 و y1 القيمتان المحسوبتان عوديا، فتكون:

x = y1 - ⌊b/a⌋ * x1
y = x1

الكود:

```cpp
#include <bits/stdc++.h>
using namespace std;
int gcdExtended(int a, int b, int *x, int *y)
{
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }

    int x1, y1;
    int gcd = gcdExtended(b%a, a, &x1, &y1);

    *x = y1 - (b/a) * x1;
    *y = x1;

    return gcd;
}

int main()
{
    int x, y, a = 35, b = 15;
    int g = gcdExtended(a, b, &x, &y);
    cout << "GCD(" << a << ", " << b
         << ") = " << g << endl;
}
```

تابع لحساب عدد القواسم لعدد d(n)

إن أي عدد يكتب على شكل جداء عوامل أولية مرفوعة لأسس هكذا:

$$n = {p_1}^{k_1} \times {p_2}^{k_2} \times {p_3}^{k_3} \times \ldots$$

عدد قواسم هذا العدد يعطى بالعلاقة:

$$d(n) = (k1 + 1) * (k2 + 1) * (k3 + 1) * \ldots$$

مثال:

قواسم العدد 12 هي {1,2,3,4,6,12} وعددها هو 6 قواسم

العدد 12 يكتب بالشكل:

$$12 = 2^2 \times 3^1$$

وبالتالي عدد قواسم العدد 12 حسب العلاقة هو

6 = (1+1)*(1+2)

تابع لحساب عدد قواسم عدد D(n)

إن أي عدد يكتب على شكل جداء عوامل أولية مرفوعة لأسس هكذا:

$$n = {p_1}^{k_1} \times {p_2}^{k_2} \times {p_3}^{k_3} \times \ldots$$

عدد قواسم هذا العدد يعطى بالعلاقة:

$$D(n) = \prod_{i=1}^{m} (k_i + 1) * (k_i + 2)/2$$

حيث m هي عدد العوامل الأولية التي يتكون منها العدد.
مثال العدد 12 أيضا:

d(12) = 6 : {1,2,3,4,6,12}
d(6) = 4 : {1,2,3,6}
d(4) = 3 : {1,2,4}
d(3) = 2 : {1,3}
d(2) = 2 : {1,2}
d(1) = 1 : {1}

D(12) = d(12)+d(6)+d(4)+d(3)+d(2)+d(1) = 18

لنستخدم الآن علاقتنا الجميلة:

$12 = 2^2 \times 3^1$

D(12) = ((2+1)*(2+2)/2)*((1+1)*(1+2)/2)
= 6*3 = 18!!