# 数学基础

## 1. 素数（质数）

定义：除了1和它本身以外不再有其它因素。

### 1.1 判断素数

- 枚举2~n,如果都除不尽，证明这个数是素数。时间复杂度O(n)。

```
#include <iostream>
using namespace std;

bool isprime(int n){
    if(n<=3)
        return true;
    for(int i=2;i<n;i++){
        if(n%i==0)
            return false;
    }
    return true;
}

int main(){

    for(int i=1;i<=100;i++){
        printf("%3d is prime: %d \n",i,isprime(i));
    }

    return 0;
}
```

- 对于一个小于n的整数X，如果n不能整除X，则n必定不能整除n/X，所以只要从2枚举到$\sqrt{n}$ 即可。时间复杂度$O(\sqrt{n})$

```
#include <iostream>
#include <cmath>
using namespace std;

bool isprime(int n){
    if(n<=3)
        return true;
    int rt = sqrt(n);
    for(int i=2;i<=rt;i++){
        if(n%i==0)
            return false;
    }
    return true;
}

int main(){

    for(int i=1;i<=100;i++){
        printf("%3d is prime: %d \n",i,isprime(i));
    }

    return 0;
}
```

- 根据对于大于等于1的自然x的来说，质数总等于6x-1或6x+1。

```
#include <iostream>
#include <cmath>
using namespace std;

bool isprime(int n){
    if(n<=3)
        return true;
    if(n%6!=1 && n%6!=5)
        return false;

    int rt = sqrt(n);
    for(int i=5;i<=rt;i+=6){
        if(n%i==0 || n%(i+2)==0)
            return false;
    }
    return true;
}

int main(){
```

```
    for(int i=1;i<=100;i++){
        printf("%3d is prime: %d \n",i,isprime(i));
    }

    return 0;
}
```

证明一下：

1. 首先6x肯定不是质数，因为它能被6整除；
2. 其次6x+2肯定也不是质数，因为它还能被2整除；
3. 依此类推，6x+3肯定能被3整除；6x+4肯定能被2整除；
4. 那么，就只有6x+1和6x+5(即等同于6x-1)可能是质数了。

## 1.2 求1~n的所有素数

- 一般线性筛选法

```
#include <iostream>
using namespace std;

const int maxn = 100000005;
int used[maxn],number[maxn];

int prime(int n){
    int cnt=0;
    for(int i=2;i<=n;i++){
        if(!used[i]){
            cnt++;
            number[cnt]=i;
        }
        for(int j=i*i;j<=n;j+=i){
            used[j]=true;
        }
    }
    return cnt;
}

int main(){

    int cnt = prime(100);

    for(int i=1;i<=cnt;i++)
        cout << number[i] << endl;

    return 0;
}
```

used

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

number

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 |

这种方法比较好理解，初始时，假设全部都是素数，当找到一个素数时，显然这个素数乘上另外一个数之后都是合数。但仔细分析能发现，这种方法会造成重复筛除合数，影响效率。比如30，在i=2的时候，k=2*15筛了一次；在i=5，k=5*6的时候又筛了一次。所以，也就有了快速线性筛选法。

- 快速线性筛选法

```
#include <iostream>
using namespace std;

const int maxn = 100000005;
int used[maxn],number[maxn];

int prime(int n){
    int cnt=0;
    for(int i=2;i<=n;i++){
        if(!used[i]){
            cnt++;
            number[cnt]=i;
        }
        for(int j=1;(j<=cnt)&&(i*number[j]<=n);j++){
            used[i*number[j]]=true;
            if(i%number[j]==0)  //如i=6,则number[]={2,3,5},那么会用2*6消除12
```

```
                break;          //但不会继续用3*6消除18，因为18会被i=9时消除
        }
    }
    return cnt;
}

int main(){

    int cnt = prime(100);

    for(int i=1;i<=cnt;i++)
        cout << number[i] << endl;

    return 0;
}
```

used

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 2 | | 3 | | 4 | 3 | 5 | | 6 | | 7 | 5 | 8 | | 9 | | 10 | 7 | 11 | | 12 | 5 | 13 |

number

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 | 31 |

# 2. GCD与LCM

## 2.1 GCD：最大公约数

1. 辗转相除法（欧几里德法）

```
int gcd1(int a,int b){
    if(a%b==0)
        return b;
    return gcd1(b,a%b);
}
```

```
int gcd2(int a,int b){
    int r=a%b;
    while(r){
        a=b;
        b=r;
        r=a%b;
    }
    return b;
}
```

2. 穷举法

```
int gcd3(int a, int b){
    int temp=(a>b)?b:a;
    while(temp>0){
        if(a%temp==0&&b%temp==0)
            break;
        temp--;
    }
    return temp;
}
```

3. 更相减损法

```
int gcd4(int m,int n){
    int i=0,temp,x=1;
    while(m%2==0&&n%2==0){
        m/=2;
        n/=2;
        i+=1;
    }
    if(m<n){
        temp = m;
        m=n;
        n = temp;
    }
    while(x){
        x=m-n;
        m=(n>x)?n:x;
        n=(n<x)?n:x;
        if(n==(m-n)) break;
    }
```

```
    if(i==0)
        return n;
    else
        return (int) pow(2,i)*n;
}
```

4. Stein

```
int gcd5(unsigned int x, unsigned int y){
    int factor = 0,temp;
    if(x < y){
        temp = x;
        x = y;
        y = temp;
    }
    if ( 0 == y )
        return 0;
    while (x!=y){
        if (x & 0x1 ){          /* when x is odd */
            if( y & 0x1 ){      /* when x and y are both odd */
                y = (x - y) >> 1;
                x -= y;
            }else{              /* when x is odd and y is even */
                y >>= 1;
            }
        }else{                  /* when x is even */
            if ( y & 0x1 ){     /* when x is even and y is odd */
                x >>= 1;
                if ( x < y ){
                    temp = x;
                    x = y;
                    y = temp;
                }
            }else{              /* when x and y are both even */
                x >>= 1;
                y >>= 1;
                ++factor;
            }
        }
    }
    return x << factor;
}
```

### 2.2 LCM：最小公倍数

```
int lcm1(int a,int b){
    int i;
    if(a<b)
        swap(a,b);
    for(i=a;;i++){
        if(i%a==0 &&i%b==0){
            break;
        }
    }
    return i;
}
```

### 2.3 GCD与LCM的关系

1. $gcd(a,b) \times lcm(a,b) = a \times b$
2. $lcm(a,b) = \frac{a \times b}{gcd(a,b)} = \frac{a}{gcd(a,b)} \times b$

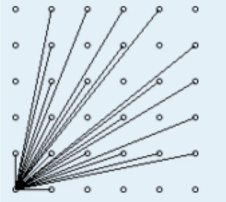
```
int lcm2(int a,int b){
    return a / gcd(a,b) * b;
}
```

### 2.4 GCD与LCM满足分配率

1. gcd(a,lcm(b,c)) = lcm(gcd(a,b),gcd(a,c))
2. lcm(a,gcd(b,c)) = gcd(lcm(a,b),lcm(a,c))

### 2.5 例一：格点问题

在第一象限的格点(x，y), (x,y>=0)除原点外，能够被原点看到，当且仅当(x,y)与原点的连线不经过其他的格点。



现给定正整数N，请计算纵坐标等于N的一行中有多少个(x,y)能够被原点看到。其中（0<=x<=N）

### Sample Input

```
5
10
```

### Sample Output

```
4
4
```

### Sample Code

对于纵坐标N=10我们发现可以被看见的点是(1,10)、(3,10)、(7,10)、(9,10)其他点看不到的原因是，比如(4,10)点一定被(2,5)这一点遮挡。所以我们可以看得出我们需要求的就是gcd(i,N)==1的数。

**注意：** 在座标里，将点(0, 0)和(a, b)连起来，通过整数座标的点的数目（除了(0, 0)一点之外）就是gcd(a, b)。

```
#include <iostream>
using namespace std;

int gcd(int a,int b){
    int r = a%b;
    while(r){
        a = b;
        b = r;
        r = a % b;
    }
    return b;
}

int n,ans=0;

int main(){
    cin >> n;

    for(int i=1;i<=n;i++){
        if(gcd(i,n)==1){
            ans ++;
        }
    }

    cout << ans << endl;

    return 0;
}
```

## 2.6 例二：[NOIP2001普及组]最大公约数和最小公倍数问题

输入2个正整数x,y(2≤x≤100000，2≤y≤1000000)，求出满足下列条件的P、Q的个数。 条件:

1. P、Q是正整数
2. 要求P、Q以x为最大公约数，以y为最小公倍数。

试求，满足条件的所有可能的两个正整数的个数。

### 输入格式

2个正整数x,y

### 输出格式

1个数，表示求出满足条件的P,Q的个数

### 输入样例

```
3 60
```

### 输出样例

```
4
```

1. 3,60
2. 15,12
3. 12,15
4. 60,3

### 样例代码

我们知道P*Q/x=y,那么P*Q=x*y； 我们假设枚举P，那么Q=x*y/P,然后验证P、Q是否满足条件的要求，满足就是计数

```
#include <iostream>
using namespace std;

int x,y,p,q,ans=0;

int gcd(int a,int b){
    int r=a%b;
    while(r){
        a = b;
        b = r;
        r = a%b;
    }
    return b;
}

int lcm(int a,int b){
    return a / gcd(a,b) * b;
}

int main(){

    cin >> x >> y;

    for(p=1;p<=y;p++){
        q = y / p * x;
        if(gcd(p,q)==x && lcm(p,q)==y){
            ans ++;
        }
    }

    cout << ans << endl;

    return 0;
}
```