数学问题

最大公约数(GCD)

辗转相除法(欧几里得算法)。

以除数和余数反复做除法运算，当余数为 0 时，取当前算式除数为最大公约数。

```
//时间复杂度O(log max(a,b))
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a%b);
}
```

扩展欧几里得算法

裴蜀定理：若a,b是整数,且gcd(a,b)=d，那么对于任意的整数x,y,ax+by都一定是d的倍数，特别地，一定存在整数x,y，使ax+by=d成立。

> 设整数a,b(a>b)
>
> a=(a/b)*b+c ····①
>
> b=(b/c)*c+d ····②
>
> c=(c/d)*d+0 ····③
>
> 则d为a,b的最大公约数，将d反带入得d=(mn-1)b-an → ax+by=gcd(a,b)
>
> ax+by=1时，当且仅当a,b互质。

算法设计：

> 使用递归定义，假设已经得到bx'+(a%b)y'=gcd(a,b)
>
> 因为：a%b=a-(a/b),得到ay'+b(x'-(a/b)y')=gcd(a,b)
>
> 当b=0时，有，a*1+b*0=a=gcd(a,b) //相当于将③的下一步递归进gcd，判断得到gcd(a,b)

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

int extgcd(int a, int b, int&x, int &y) {
    int d = a;
    if (b != 0) {
        d = extgcd(b, a%b, y, x);
        y -= (a / b)*x;
    }
    else {
        x = 1, y = 0;
    }
    return d;
}

int main() {
    int a, b, x = 0, y = 0;
```

```
    cin >> a >> b;
    extgcd(a, b, x, y);
    cout << x << " " << y;
    return 0;
}
```

最小公倍数

最小公倍数等于两数之积除以其最大公约数。

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

```
class Solution {
    public int lcm(int a, int b) {
        int c = a * b;
        if (a < b) {
            int r = 0;
            r = a;
            a = b;
            b = r;
        }
        while (true) {
            int r = a % b;
            if (r == 0) {
                return c / b;
            } else {
                a = b;
                b = r;
            }
        }
    }
}
```

有关素数的算法

素数判定

n的约束不会大于sqrt(n)

```
//时间复杂度O(sqrt(n))
bool is_prime(int n) {
    for (int i = 2; i * i <= n; i++)
        if (n%i == 0) return false;
    return n != 1;
}
```

> 了解：约数枚举 整数分解

埃氏筛法

在2~n中，最小素数为2，故划去所有2的倍数，然后是3，则划去3所有的倍数……，最终剩下的就是素数。

复杂度O(nloglogn)

```
#define maxsize 100
int prime[maxsize + 1];
bool is_prime[maxsize + 1];
int sieve(int n) {
    for (int i = 0; i <= n; i++)
        is_prime[i] = true;
    is_prime[0] = is_prime[1] = false;
    int p = 0;
    for (int i = 2; i <= n; i++) {
        if (is_prime[i]) {
            prime[p++] = i;
            for (int j = i * 2; j <= n; j += i)
                is_prime[j] = false;
        }
    }
    return p;
}
```

区间筛法

对于区间 [a,b)，对于b只需要检测到sqrt(b),故先筛出 [2,sqrt(b)) 的素数表，然后筛的同时，在区间[a,b)中划去当前被筛的数的倍数。使用max(2LL, (a + i - 1) / i)*i 来算出[a,b)中从2起的倍数，最后0—(b-a)中true的个数为素数个数。

```
#define maxsize 100
typedef long long ll;
bool is_prime[maxsize], is_prime_small[maxsize];
//is_prime[i-a]=true → i为素数
void segment_sieve(ll a, ll b) {
    for (int i = 0; (ll)i * i < b; i++)//根据[a,b)和[a.b]取等号 默认左闭右开
        is_prime_small[i] = true;
    for (int i = 0; i < b - a; i++)
        is_prime[i] = true;
    for (int i = 2; (ll)i*i < b; i++) {
        if (is_prime_small[i]) {
            for (int j = 2 * i; (ll)j*j < b; j += i)
                is_prime_small[j] = false;
            for (ll j = max(2LL, (a + i - 1) / i)*i; j < b; j += i)
                is_prime[j - a] = false;
        }
    }
}
```

模运算

快速幂

x^n=((((x^2)^2)^2)^2)⋯ ,n=2^k1+2^k2+⋯⋯,也可以将n理解为二进制形式

```
typedef long long ll;
//n>0
ll mod_pow(ll x, ll n, ll mod) {
    ll res = 1;
    while (n) {
        if (n & 1) res = res *  x% mod;
        x = x * x % mod;
        n >>= 1;
    }
    return res;
}
```

```
typedef long long ll;
//(a * b)%mod
ll mod_mult(ll a, ll b, ll mod) {
    ll res = 0;
    a = a % mod;
    b = b % mod;
    while (b) {
        if (b & 1)
            res = (res + a) % mod;
        a = (a << 1) % mod;
        b >>= 1;
    }
    return res;
}
//(a^b)%mod
ll mod_pow(ll a, ll b, ll mod) {
    ll res = 1;
    a = a % mod;
    while (b) {
        if (b & 1)
            res = mod_mult(res, a, mod);
        a = mod_mult(a, a, mod);
        b >>= 1;
    }
    return res;
}
```