0.1 符号

$$
\begin{aligned}
a\&(b \mid c) &= (a\&b) \mid (a\&c) \\
a \oplus (b \mid c) &= (a \oplus b) \mid (a \oplus c) \\
a \mid (a\&b) &= a \\
a\&(a \mid b) &= a \\
(a+b) &= (a \mid b) + (a\&b)
\end{aligned}
$$

0.2 快速幂

```
ll ksm(ll b,ll p)
{
    ll r=1;
    while(p>0)
    {
        if(p&1) r=(r%mod*(b%mod))%mod;
        p>>=1;b=(b%mod*(b%mod))%mod;
    }
    return r;
}
```

0.3 快速乘

```
ll ksc(ll a,ll b)
{
        ll re=0;
        while(b)
        {
                if(b&1) re=(re+a)%mod;
                b>>=1;a=(a+a)%mod;
        }
        return re;
}
```

0.4 组合数学

$$C_{n+1}^{m}=C_{n}^{m}+C_{n}^{m-1}$$
$$C_n^0+C_n^1+C_n^2+...+C_n^n=2^n$$
$$C_n^0+C_n^2+C_n^4+...=C_n^1+C_n^3+C_n^5+...=2^{n-1}$$

不相邻的排列

$1\sim n$这n个自然数中选k个，这k个数中任何两个数都不相邻的组合有$\binom{n-k+1}{k}$种。

错位排列

n封不同的信，编号分别是$1,2,3,4,5$,现在要把这五封信放在编号$1,2,3,4,5$的信封中，要求信封的编号与信的编号不一样。问有多少种不同的放置方法？

$f(n)$为有n封信的错排方式，$f(n)=(n-1)(f(n-1)+f(n-2))$。

错位排列数列的前几项为$0,1,2,9,44,265$。

圆排列

n个人围成一个圆，圆的所有组成排列数记为Q_n^n，其中$Q_n^n\times n=A_n^n$。

由此可知部分圆排列的公式：

$$Q_n^r=\frac{A_n^r}{r}=\frac{n!}{r\times(n-r)!}$$

0.5 Lucas定理

```
ll C(ll n,ll m)
{
    if(n<m) return 0;
    if(m>n-m) m=n-m;
    ll a=1,b=1;
    for(int i=0;i<m;i++)
    {
        a=(a*(n-i))%mod;b=(b*(i+1))%mod;
    }
    return a*ksm(b,mod-2)%mod;
}
ll lucas(ll n,ll m)
```

```
{
    if(m==0)  return 1;
    return lucas(n/mod,m/mod)*C(n%mod,m%mod)%mod;
}
```

0.6 预处理版本组合数

```
void binom_init(int x) {
    fac[0] = fac[1] = 1;
    inv[1] = 1;
    finv[0] = finv[1] = 1;
    for(int i=2; i<x; i++){
        fac[i] = fac[i-1]*i%mod;
        inv[i] = mod-mod/i*inv[mod%i]%mod;
        finv[i] = finv[i-1]*inv[i]%mod;
    }
}
ll binom(ll n, ll r){
    if(n<r || n<0 || r<0) return 0;
    return fac[n]*finv[r]%mod*finv[n-r]%mod;
}
```

0.7 线性筛

```
int primes[N],cnt; //primes[]存储所有素数
bool st[N]; //st[x]存储x是否被筛掉
void get_primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!st[i]) primes[cnt++]=i;
        for (int j=0;primes[j]<=n/i;j++)
        {
            st[primes[j]*i]=true;
            if(i%primes[j]==0) break;
        }
    }
}
```

0.8 Miller-Rabin素性测试

```
bool millerRabin(int n)
{
    if(n<3||n%2==0) return n==2;
    int a=n-1,b=0;
    while(a%2==0) a/=2,++b;
    // test_time 为测试次数,建议设为不小于8
    // 的整数以保证正确率,但也不宜过大,否则会影响效率
    for(int i=1,j;i<=test_time;++i)
    {
        int x=rand()%(n - 2)+2,v=quickPow(x,a,n);
        if(v==1) continue;
        for(j=0;j<b;++j)
        {
            if(v==n-1) break;
            v=(long long)v*v%n;
        }
        if(j>=b) return 0;
    }
    return 1;
}
```

0.9 欧拉函数

欧拉函数，即$\varphi(n)$，表示的是小于等于n和n互质的数的个数。比如说$\varphi(1)=1$。当n是质数的时候，显然有$\varphi(n)=n-1$。

```
void ora(int n)
{
    phi[1]=1;
    for(int i=2;i<=n;i++)
    {
        if(!st[i])  //i是质数
        {
            primes[cnt++] = i;
            phi[i] = i-1;
        }
        for(int j=0;primes[j]*i<=n;j++)
        {
            st[i*primes[j]]=true;
```

```
            //情况1 primes[j]是i的最小质因子
            if(i%primes[j]==0)
            {
                phi[i*primes[j]] = phi[i]*primes[j];
                break;
            }
            //情况2
            phi[i*primes[j]] = phi[i]*(primes[j]-1);
        }
    }
}
```

0.10 线性代数

0.10.1 矩阵

矩阵乘法

设A为$P \times M$的矩阵，B为$M \times Q$的矩阵，设矩阵C为矩阵A与B的乘积，其中矩阵C中的第i行第j列元素可以表示为：

$$C_{i,j} = \sum_{k=1}^{M} A_{i,k}B_{k,j}$$

口诀为**前行乘后列**。

矩阵乘法满足结合律，不满足一般的交换律。利用结合律，矩阵乘法可以利用**快速幂**的思想来优化。

使用二维数组模拟矩阵

```
struct mat
{
    ll a[sz][sz];
    inline mat(){memset(a,0,sizeof(a));}
    inline mat operator-(const mat& T) const
    {
        mat res;
        for(int i=0;i<sz;i++)
            for(int j=0;j<sz;j++)
                res.a[i][j]=(a[i][j]-T.a[i][j])%mod;
        return res;
```

```
    }
    inline mat operator+(const mat& T) const
    {
        mat res;
        for(int i=0;i<sz;i++)
            for(int j=0;j<sz;j++)
                res.a[i][j]=(a[i][j]+T.a[i][j])%mod;
        return res;
    }
    inline mat operator*(const mat& T) const
    {
        mat res;
        int r;
        for(int i=0;i<sz;i++)
            for(int k=0;k<sz;k++)
            {
                r=a[i][k];
                for(int j=0;j<sz;j++)
                    res.a[i][j]+=T.a[k][j]*r,res.a[i][j]%=
                        mod;
            }
        return res;
    }
    inline mat operator^(ll x) const
    {
        mat res,bas;
        for(int i=0;i<sz;i++) res.a[i][i]=1;
        for(int i=0;i<sz;i++)
            for(int j=0;j<sz;j++) bas.a[i][j]=a[i][j]%mod;
        while(x)
        {
            if(x&1) res=res*bas;
            bas=bas*bas;x>>=1;
        }
        return res;
    }
};
```

矩阵加速递推

矩阵加速的核心是加速矩阵的构造。以斐波那契数列为例，其递推式：

$$F(n) = F(n-1) + F(n-2)$$

也就是对于$F(n)$来说，对其推导有用的只有$F(n-1)$和$F(n-2)$。我们将他们放在同一个矩阵里。(顺序无所谓)

$$\begin{bmatrix} F(n-2) \\ F(n-1) \end{bmatrix}$$

考虑我们对这个矩阵不断左乘一个加速矩阵有什么效果。

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} F(n-2) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} F(n-1) \\ F(n) \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} F(n-1) \\ F(n) \end{bmatrix} = \begin{bmatrix} F(n) \\ F(n+1) \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} F(n) \\ F(n+1) \end{bmatrix} = \begin{bmatrix} F(n+1) \\ F(n+2) \end{bmatrix}$$

所以：

$$\begin{bmatrix} F(n-1) \\ F(n) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-2} \times \begin{bmatrix} F(1) \\ F(2) \end{bmatrix}$$

矩阵快速幂即可快速求解。

其他数列的递推方法

假设我们目标构造$F(n) = F(n-1) + F(n-2) + 1$。首先，将对$F(n)$有用的信息放在一个列向量里：

$$\begin{bmatrix} F(n-2) \\ F(n-1) \\ 1 \end{bmatrix}$$

为其构造$n * n$大小的加速矩阵，该矩阵应为：

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

floyed矩阵优化DP

我们定义floyed矩阵乘法如下：

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} f[n] \\ f[n-1] \end{bmatrix} = \begin{bmatrix} max\{f[n]+1, f[n-1]+1\} \\ max\{f[n]+1, f[n-1]+0\} \end{bmatrix}$$

0.10.2 线性基

```
void add(int x)
{
    for(int i=62;i>=0;i--)
    {
        if(x>>i&1)
        {
            if(!p[i])
            {
                p[i]=x;break;
            }
            else x^=p[i];
        }
    }
}
```

性质

- 原序列里面的任意一个数都可以由线性基里面的一些数异或得到。
- 线性基里面的任意一些数异或起来都不能得到0。
- 线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的。

证明性质1

若数字x**不能插入线性基**，显然就是它在尝试插入时异或若干个数之后变成了0，即$x \oplus p[a] \oplus p[b] \cdots = 0$，则有$p[a] \oplus p[b] \cdots = x$，所以，如果$x$不能成功插入线性基，一定是因为当前线性基里面的一些数异或起来可以等于x。

若数字x**能插入线性基**，我们假设x插入到了线性基的第i个位置，显然，它在插入前可能异或若干个数，那么就有：$p[a] \oplus p[b] \cdots = p[i]$。同理这样的$x$也可以在线性基上找到。

证明性质2

若性质2可以满足，则有$p[a] \oplus p[b] \oplus \cdots \oplus = p[c]$，那么$p[c]$就不可能会被插入线性基中。结果矛盾，性质无法成立。

序列异或最大值

```
ll query_max()
{
    ll anss=0;
    for(int i=60;i>=0;i--)//记得从线性基的最高位开始
        if((anss^p[i])>anss)anss^=p[i];
    return anss;
}
```

序列异或最小值

```
ll query_min()
{
    for(int i=0;i<=60;i++)
        if(p[i]) return p[i];
    return 0;
}
```

序列异或第k小

```
void work()//处理线性基
{
    for(int i=1;i<=60;i++)
    for(int j=1;j<=i;j++)
    if(d[i]&(1ll<<(j-1)))d[i]^=d[j-1];
}
ll k_th(ll k)
{
    //假如k=1，且原来的序列可以异或出0，返回0。tot表示线性基中的元素个数，
        n表示序列长度
    if(k==1&&tot<n) return 0;
    //类似上面，去掉0的情况，因为线性基中只能异或出不为0的解
    if(tot<n) k--;
    work();
    ll ans=0;
```

```
    for(int i=0;i<=60;i++)
        if(d[i]!=0)
        {
            if(k%2==1) ans^=d[i];
            k/=2;
        }
}
```

0.11 概率论

概率公式

加法公式：$P(A\cup B)=P(A)+P(B)-P(A\cap B)$

减法公式：$P(A-B)=P(A)-P(AB)$

分配律：$P((A\cup B)\cap C)=P(AC\cup BC), P((AB)\cup C)=P((A\cup C)\cap(B\cup C))$

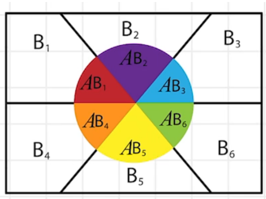
对偶律：$P(\overline{A\cup B})=P(\overline{A}\cap\overline{B}), P(\overline{A\cap B})=P(\overline{A}\cup\overline{B})$，长道变短道，开口换方向。

条件概率：记$P(B|A)$表示在A事件发生的前提下，B事件发生的概率，则$P(B|A)=\dfrac{P(AB)}{P(A)}$

乘法公式：$P(AB)=P(B|A)\cdot P(A)=P(A|B)\cdot P(B)$

全概率公式：当所求的事件A可以分成几种情况时，A发生的概率就是这些情况的概率和。$P(A)=\sum\limits_{i=1}^{n}P(AB_i)=\sum\limits_{i=1}^{n}P(A|B_i)P(B_i)$

贝叶斯公式：$P(B_i|A)=\dfrac{P(B_iA)}{P(A)}=\dfrac{P(A|B_i)P((B_i))}{\sum\limits_{j=1}^{n}P(A|B_j)P(B_j)}$



独立性

若事件AB独立，则$P(AB)=P(A)P(B), P(B)=P(B|A)=P(B|\overline{A})$。

A和B互相独立，则A和$\overline{B}$，$\overline{A}$和B，$\overline{A}$和$\overline{B}$也互相独立。

随机变量

对于随机变量X，称函数$F(X)=P(X\le x)$为X的**分布函数**，分布函数单调递增，$F(-\infty)=0,F(\infty)=1$。

离散型随机变量

分布函数:$F(X)=\sum\limits_{x_k\le x}p_k$

连续型随机变量

分布函数:$F(x)=\int_{-\infty}^{x}f(t)\mathrm{d}t$，同样的$\int_{-\infty}^{\infty}f(t)\mathrm{d}t=1$

概率密度:$f(x)=F'(x)$

二项分布与泊松分布

二项分布：$P\{X=k\}=C_n^kp^kq^{n-k}(q=1-p)$

泊松分布：$P\{X=k\}=\frac{\lambda^ke^{-\lambda}}{k!}$

正态分布

$X\sim N(\mu,\sigma^2)$其中μ是期望，σ^2是方差。

方差:$D(X)=E(X^2)-[E(x)]^2$。标准差$\sigma=\sqrt{D(x)}$

期望

离散型随机变量的期望:$E(X)=\sum\limits_{i=1}^{\infty}x_ip_i$

连续型随机变量的期望:$E(X)=\int_{-\infty}^{\infty}xf(x)\mathrm{d}x$

$E(CX)=CE(X),E(X+Y)=E(X)+E(Y)$

设X,Y互相独立，则$E(XY)=E(X)E(Y)$

0.12 高精度算法

```
const int maxn = 1000;//越大速度越慢maxn
struct bign{
    int d[maxn], len;
    void clean() { while(len > 1 && !d[len-1]) len--; }
    bign() { memset(d, 0, sizeof(d)); len = 1; }
    bign(int num) { *this = num; }
    bign(char* num) { *this = num; }
    bign operator = (const char* num){
```

```
        memset(d, 0, sizeof(d)); len = strlen(num);
        for(int i = 0; i < len; i++) d[i] = num[len-1-i] - '0';
        clean();
        return *this;
    }
    bign operator = (int num){
        char s[20]; sprintf(s, "%d", num);
        *this = s;
        return *this;
    }

    bign operator + (const bign& b){
        bign c = *this; int i;
        for (i = 0; i < b.len; i++){
            c.d[i] += b.d[i];
            if (c.d[i] > 9) c.d[i]%=10, c.d[i+1]++;
        }
        while (c.d[i] > 9) c.d[i++]%=10, c.d[i]++;
        c.len = max(len, b.len);
        if (c.d[i] && c.len <= i) c.len = i+1;
        return c;
    }
    bign operator - (const bign& b){
        bign c = *this; int i;
        for (i = 0; i < b.len; i++){
            c.d[i] -= b.d[i];
            if (c.d[i] < 0) c.d[i]+=10, c.d[i+1]--;
        }
        while (c.d[i] < 0) c.d[i++]+=10, c.d[i]--;
        c.clean();
        return c;
    }
    bign operator * (const bign& b)const{
        int i, j; bign c; c.len = len + b.len;
        for(j = 0; j < b.len; j++) for(i = 0; i < len; i++)
            c.d[i+j] += d[i] * b.d[j];
        for(i = 0; i < c.len-1; i++)
            c.d[i+1] += c.d[i]/10, c.d[i] %= 10;
        c.clean();
        return c;
    }
```

```
    bign operator / (const bign& b){
        int i, j;
        bign c = *this, a = 0;
        for (i = len - 1; i >= 0; i--)
        {
            a = a*10 + d[i];
            for (j = 0; j < 10; j++) if (a < b*(j+1)) break;
            c.d[i] = j;
            a = a - b*j;
        }
        c.clean();
        return c;
    }
    bign operator % (const bign& b){
        int i, j;
        bign a = 0;
        for (i = len - 1; i >= 0; i--)
        {
            a = a*10 + d[i];
            for (j = 0; j < 10; j++) if (a < b*(j+1)) break;
            a = a - b*j;
        }
        return a;
    }
    bign operator += (const bign& b){
        *this = *this + b;
        return *this;
    }

    bool operator <(const bign& b) const{
        if(len != b.len) return len < b.len;
        for(int i = len-1; i >= 0; i--)
            if(d[i] != b.d[i]) return d[i] < b.d[i];
        return false;
    }
    bool operator >(const bign& b) const{return b < *this;}
    bool operator<=(const bign& b) const{return !(b < *this)
       ;}
    bool operator>=(const bign& b) const{return !(*this < b)
       ;}
    bool operator!=(const bign& b) const{return b < *this ||
        *this < b;}
```

```
    bool operator==(const bign& b) const{return !(b < *this)
         && !(b > *this);}

    string str() const{
        char s[maxn]={};
        for(int i = 0; i < len; i++) s[len-1-i] = d[i]+'0';
        return s;
    }
};
istream& operator >> (istream& in, bign& x)
{
    string s;
    in >> s;
    x = s.c_str();
    return in;
}
ostream& operator << (ostream& out, const bign& x)
{
    out << x.str();
    return out;
}
int main()
{
    bign a, b;
    while (cin >> a >> b) cout << a+b << endl;
    return 0;
}
```