数学

线性代数

高斯消元

```
int a[N][N],n;
int gauss(){    //高斯消元  O(n^3)
    double eps=1e-8;
    for(int i=1;i<=n;i++){
        int r=i;
        for(int j=1;j<=n;j++){
            if(fabs(a[j][j])>eps && j<i)continue;
            if(fabs(a[j][i])>fabs(a[r][i]))r=j;
        }
        if(r!=i)swap(a[i],a[r]);
        if(fabs(a[i][i])<eps)continue;
        for(int j=n+1;j>=i;j--)a[i][j]/=a[i][i];
        for(int j=n+1;j>=i;j--){
            for(int k=1;k<=n;k++){
                if(i==k)continue;
                a[k][j]-=a[k][i]*a[i][j];
            }
        }
    }
    for(int i=1;i<=n;i++){
        // 无解
        if(fabs(a[i][i])<eps && fabs(a[i][n+1])>eps)return -1;
    }
    for(int i=1;i<=n;i++){
        // 无数解
        if(fabs(a[i][i])<eps)return 0;
    }
    return 1;
}
```

行列式

```
int n,mod,a[N][N];
int det(int siz){      // n 阶方阵行列式 O(n^3)
    int s=1;
    for(int i=1;i<=siz;i++){
        for(int j=i+1;j<=siz;j++){
            if(a[j][i]==0)continue;
            // 辗转相减
            while(a[i][i]){
                int d=a[j][i]/a[i][i];
                for(int k=i;k<=siz;k++){
                    a[j][k]=(a[j][k]-a[i][k]*d%mod+mod)%mod;
                }
                swap(a[i],a[j]),s=mod-s;
```

```
            }
            swap(a[i],a[j]),s=mod-s;
        }
    }
    for(int i=1;i<=siz;i++)s=s*a[i][i]%mod;
    return s;
}
```

矩阵树定理

```
// 统计所有带权生成树之和，把边权理解成重边个数
int MT(){       // 无向图，生成树计数
    // 度数矩阵 - 邻接矩阵
    for(int i=1;i<=m;i++){
        int u,v,w;
        cin>>u>>v>>w;
        a[u][u]=(a[u][u]+w)%mod;
        a[v][v]=(a[v][v]+w)%mod;
        a[u][v]=(a[u][v]-w+mod)%mod;
        a[v][u]=(a[v][u]-w+mod)%mod;
    }
    return det(n-1);
}
int MT_out(int rt){       // 有向图，以 rt 为根的外向树计数
    // 入度矩阵 - 邻接矩阵
    for(int i=1;i<=m;i++){
        int u,v,w;
        cin>>u>>v>>w;
        a[v][v]=(a[v][v]+w)%mod;
        a[u][v]=(a[u][v]-w+mod)%mod;
    }
    for(int i=1;i<=n-1;i++){
        for(int j=1;j<=n-1;j++){
            if(i<rt && j>=rt)a[i][j]=a[i][j+1];
            else if(i>=rt && j<rt)a[i][j]=a[i+1][j];
            else if(i>=rt && j>=rt)a[i][j]=a[i+1][j+1];
        }
    }
    return det(n-1);
}
int MT_in(int rt){       // 有向图，以 rt 为根的内向树计数
    // 出度矩阵 - 邻接矩阵
    for(int i=1;i<=m;i++){
        int u,v,w;
        cin>>u>>v>>w;
        a[u][u]=(a[u][u]+w)%mod;
        a[u][v]=(a[u][v]-w+mod)%mod;
    }
    for(int i=1;i<=n-1;i++){
        for(int j=1;j<=n-1;j++){
            if(i<rt && j>=rt)a[i][j]=a[i][j+1];
            else if(i>=rt && j<rt)a[i][j]=a[i+1][j];
            else if(i>=rt && j>=rt)a[i][j]=a[i+1][j+1];
        }
    }
```

```
    return det(n-1);
}
```

BEST 定理

```
// 有向图欧拉回路计数    O(n^3)
const int N=3e2+5,M=1e6+5;
const int mod=1e6+3;
int fac[M];
int a[N][N],vis[N],vis2[N];
int det(int siz){
    int s=1;
    for(int i=1;i<=siz;i++){
        if(!vis2[i])continue;        // 跳过不可达点
        for(int j=i+1;j<=siz;j++){
            if(a[j][i]==0 || !vis2[j])continue;
            while(a[i][i]){
                int d=a[j][i]/a[i][i];
                for(int k=i;k<=siz;k++)
                    a[j][k]=(a[j][k]-a[i][k]*d%mod+mod)%mod;
                swap(a[i],a[j]),s=mod-s;
            }
            swap(a[i],a[j]),s=mod-s;
        }
    }
    for(int i=1;i<=siz;i++)if(vis2[i])s=s*a[i][i]%mod;
    return s;
}
int out[N],in[N];
int MT_in(int rt,int n){       // 有向图，以 rt 为根的内向树计数
    for(int i=1;i<=n-1;i++){
        // 删掉 rt 行列
        if(i>=rt)vis2[i]=vis[i+1];
        else vis2[i]=vis[i];
        for(int j=1;j<=n-1;j++){
            if(i<rt && j>=rt)a[i][j]=a[i][j+1];
            else if(i>=rt && j<rt)a[i][j]=a[i+1][j];
            else if(i>=rt && j>=rt)a[i][j]=a[i+1][j+1];
        }
    }
    return det(n-1);
}
int f[N];
int find(int x){return f[x]==x?x:f[x]=find(f[x]);}
void solve(){        // 从 rt 开始的欧拉回路计数
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)f[i]=i;
    while(m--){
        int u,v;
        cin>>u>>v;
        f[find(u)]=find(v);
        out[u]++,in[v]++;
        a[u][u]++;
        a[u][v]--;
```

```
    }
    int rt=1;
    for(int i=1;i<=n;i++){
        // 不是欧拉回路，不和 rt 联通
        if(in[i]!=out[i] || (in[i] && find(i)!=find(rt))){
            cout<<0<<endl;
            return;
        }
        if(in[i])vis[i]=1;
    }
    int ans=MT_in(1,n)*fac[out[rt]]%mod;
    for(int i=1;i<=n;i++)if(vis[i] && i!=rt)ans=ans*fac[out[i]-1]%mod;
    cout<<ans<<endl;
}
```

矩阵求逆

```
#define int long long
const int N=4e2+5;
const int mod=1e9+7;
int qpow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=ans*a%mod;
        b>>=1;
        a=a*a%mod;
    }
    return ans;
}
int n,a[N][2*N],b[N][N];
bool get_inv(){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            a[i][n+j]=(i==j);
        }
    }
    for(int i=1;i<=n;i++){
        int pos=i;
        for(int j=i+1;j<=n;j++){
            if(a[j][i]!=0)pos=j;
        }
        swap(a[i],a[pos]);
        if(a[i][i]==0){
            // 不可逆
            return false;
        }
        int iv=qpow(a[i][i],mod-2);
        for(int j=2*n;j>=i;j--)a[i][j]=a[i][j]*iv%mod;
        for(int j=2*n;j>=i;j--){
            for(int k=1;k<=n;k++){
                if(i==k)continue;
                a[k][j]=(a[k][j]-a[k][i]*a[i][j]%mod+mod)%mod;
            }
        }
    }
```

```
    for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
        b[i][j]=a[i][j+n];
    return true;
}
```

矩阵乘法

```
struct Matrix{
    int n,m;
    int val[N][N];
    Matrix operator*(const Matrix&a){
        // 保证 this.m=a.n
        Matrix ans;
        ans.n=n;
        ans.m=a.m;
        for(int i=1;i<=ans.n;i++)
        for(int j=1;j<=ans.m;j++)
            ans.val[i][j]=0;
        for(int i=1;i<=ans.n;i++)
        for(int j=1;j<=ans.m;j++)
        for(int k=1;k<=m;k++){
            ans.val[i][j]+=val[i][k]*a.val[k][j];
            ans.val[i][j]%=mod;
        }
        return ans;
    }
};
```

矩阵快速幂

```
const int N=100;
const int mod=1e9+7;
struct Matrix{
    int siz;
    int val[N][N];
    Matrix operator*(const Matrix&a){
        Matrix ans;
        ans.siz=siz;
        for(int i=1;i<=siz;i++)
        for(int j=1;j<=siz;j++)
            ans.val[i][j]=0;
        for(int i=1;i<=siz;i++)
        for(int j=1;j<=siz;j++)
        for(int k=1;k<=siz;k++){
            ans.val[i][j]+=val[i][k]*a.val[k][j];
            ans.val[i][j]%=mod;
        }
        return ans;
    }
};
Matrix mpow(Matrix a,int b){
    Matrix ans;
    ans.siz=a.siz;
```

```
    for(int i=1;i<=a.siz;i++)
    for(int j=1;j<=a.siz;j++){
        if(i==j)ans.val[i][j]=1;
        else ans.val[i][j]=0;
    }
    while(b){
        if(b&1)ans=ans*a;
        a=a*a;
        b>>=1;
    }
    return ans;
}
```

线性基

高斯消元法

```
#define int long long
const int N=3e3+5;
int a[N],row,n,mx;
void gauss(){    //将a[i]构造成线性基
    row=1;
    for(int col=63;col>=0;col--){
        for(int i=row;i<=n;i++){
            if(a[i]&(1ll<<col)){
                swap(a[row],a[i]);
                break;
            }
        }
        if(!(a[row]&(1ll<<col)))continue;
        for(int i=1;i<=n;i++){
            if(i==row)continue;
            if(a[i]&(1ll<<col))a[i]^=a[row];
        }
        row++;
        if(row>n)break;
    }
    row--;
    mx=1;        // 线性基可以表示出的值的个数
    for(int i=1;i<=row;i++)mx*=2;
    if(row==n)mx--;      // 如果至少选一个数，row==n 时要减一
}
int query(int k){        // 查询第k小
    if(k>mx)return -1;
    if(row<n)k--;        // row<n 时，最小值是 0
    int ans=0;
    for(int i=0;i<row;i++){
        if(k&(1ll<<i))ans^=a[row-i];
    }
    return ans;
}
void solve(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    gauss();
```

```
    int ans=0;
    for(int i=1;i<=row;i++){
        ans^=a[i];
    }
    cout<<ans<<endl;
    return;
}
```

贪心法

```
const int N=5e6+5;
int bit[N];
void ins(int x){
    for(int i=62;i>=0;i--){
        if(bit[i]==0&&(x&(1ll<<i))){
            bit[i]=x;
            break;
        }
        else if(x&(1ll<<i))x^=bit[i];
    }
}
int q_max(){
    int ans=0;
    for(int i=62;i>=0;i--)
        if(!(ans&(1ll<<i)))ans^=bit[i];
    return ans;
}
int q_max(int x){
    for(int i=62;i>=0;i--){
        if((x^bit[i])>x)x^=bit[i];
    }
    return x;
}
// 实数线性基
int ins(int x){      // 返回能否成功插入
    for(int i=1;i<=m;i++){      // m 维向量
        if(bit[i]==0 && fabs(a[x][i])>1e-7){
            bit[i]=x;
            for(int j=m;j>=i;j--)a[x][j]/=a[x][i];
            return 1;
        }
        else if(fabs(a[x][i])>1e-7){
            for(int j=m;j>=i;j--){
                a[x][j]/=a[x][i];
                a[x][j]-=a[bit[i]][j];
            }
        }
    }
    return 0;
}
```

可撤销线性基

```
// 线段树分治
const int N=1e3+5,mx=1e3;
bitset<N>bit[N];
int vis[N];
stack<int>st;
int ins(bitset<N> b){
    for(int i=mx;i>=0;i--){
        if(b[i]){
            if(vis[i])b^=bit[i];
            else{
                vis[i]=1;
                bit[i]=b;
                st.push(i);
                return 1;
            }
        }
    }
    return 0;
}
void roll_back(){
    int pos=st.top();
    st.pop();
    vis[pos]=0;
    bit[pos].reset();
}
bitset<N>query(){
    bitset<N>ret;
    ret.reset();
    for(int i=mx;i>=0;i--){
        if(vis[i] && !ret[i])ret^=bit[i];
    }
    return ret;
}
```

数论

筛法

线性筛

```
const int M=1e6+5;      //线性筛质数
int prime[M],cnt;
int vis[M]; //0表示质数
void Prime(){
    for(int i=2;i<M;i++) {
        if(!vis[i])prime[++cnt]=i;
        for(int j=1;j<=cnt && i*prime[j]<M;j++){
            vis[i*prime[j]]=1;
            if(i%prime[j]==0)break;
        }
    }
    vis[1]=1;
}
```

```
const int M=1e6+5;          //线性筛莫比乌斯函数
int mu[M+10],prime[M+10],vis[M+10];
int tot=0;
void get_mu(){
    mu[1]=1;
    for(int i=2;i<=M;i++){
        if(!vis[i]){
             mu[i]=-1;
             prime[++tot]=i;
        }
        for(int j=1;j<=tot&&i*prime[j]<=M;j++){
            vis[i*prime[j]]=1;
            if(i%prime[j]==0){
                mu[i*prime[j]]=0;
                break;
            }
            else{
                mu[i*prime[j]]=-mu[i];
            }
        }
    }
}

const int M=1e6+5;          //线性筛欧拉函数
int phi[M+10],prime[M+10];
int tot=0;
void Get_Eular(){
    phi[1]=1;
    for(int i=2;i<=M;i++){
        if(!phi[i]){             //i为素数
            phi[i]=i-1;
            prime[++tot]=i;
        }
        for(int j=1;j<=tot&&i*prime[j]<=M;j++){
            if(i%prime[j]==0){
                phi[i*prime[j]]=phi[i]*prime[j]; //i*prime[j]的素因子和i是一样的,只相当与上文中的m扩大了
                break;
            }
            else phi[i*prime[j]]=phi[i]*phi[prime[j]];//积性函数的性质，i与prime[j]互质
        }
    }
}
```

杜教筛

```
const int N=5e6+5;
int phi[N],prime[N],mu[N];
int tot=0;
unordered_map<int,int>sumphi,summu;
void get_pre(){      //预处理
    phi[1]=1;
    mu[1]=1;
```

```
    for(int i=2;i<N;i++){
        if(!phi[i]){
            phi[i]=i-1;
            mu[i]=-1;
            prime[++tot]=i;
        }
        for(int j=1;j<=tot && i*prime[j]<N;j++){
            if(i%prime[j]==0){
                mu[i*prime[j]]=0;
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }
            else{
                phi[i*prime[j]]=phi[i]*phi[prime[j]];//积性函数的性质，i与prime[j]互质
                mu[i*prime[j]]=-mu[i];
            }
        }
    }
    sumphi[0]=0,sumphi[1]=1;
    summu[0]=0,summu[1]=1;
    for(int i=2;i<M;i++){
        sumphi[i]=sumphi[i-1]+phi[i];
        summu[i]=summu[i-1]+mu[i];
    }
}
int sum_phi(int n){      // 杜教筛欧拉函数
    if(sumphi.count(n))return sumphi[n];
    int ans=n*(n+1)/2;
    for(int l=2,r;l<=n;l=r+1){
        r=n/(n/l);
        ans-=(r-l+1)*sum_phi(n/l);
    }
    return sumphi[n]=ans;
}
int sum_mu(int n){       // 杜教筛莫比乌斯函数
    if(summu.count(n))return summu[n];
    int ans=1;
    for(int l=2,r;l<=n;l=r+1){
        r=n/(n/l);
        ans-=(r-l+1)*sum_mu(n/l);
    }
    return summu[n]=ans;
}
```

min_25筛

Gcd

```
int gcd(int x,int y){
    while(y^=x^=y^=x%=y);
    return x;
}
int lcm(int x,int y){
    return x/gcd(x,y)*y;
}
```

类欧几里得算法

$f(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor$

$g(a,b,c,n)=\sum_{i=0}^{n}i\lfloor\frac{ai+b}{c}\rfloor$

$h(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor^2$

```
// f(a,b,c,n)   \sum_{i=0}^{n}{ (ai+b)/c }
int cal(int a,int b,int c,int n){
    if(n<0)return 0;
    if(c<0)a=-a,b=-b,c=-c;
    int x=a/c, y=b/c;
    int sum1=n*(n+1)/2;
    if(a<0 || b<0)
        return sum1*(x-1) + (n+1)*(y-1) + cal(a%c+c,b%c+c,c,n);
    else if(a==0)
        return (n+1)*y;
    else if(a>=c || b>=c)
        return sum1*x + (n+1)*y + cal(a%c,b%c,c,n);
    else{
        int m=(a*n+b)/c;
        return n*m-cal(c,c-b-1,a,m-1);
    }
}
// O(log a) 类欧几里得算法
tuple<int,int,int> cal(int a,int b,int c,int n){
    int f=0,g=0,h=0;
    int x=a/c, y=b/c;
    int sum1=n*(n+1)%mod*inv_2%mod, sum2=n*(n+1)%mod*(2*n+1)%mod*inv_6%mod;
    if(a==0){
        f = (n+1)*y%mod;
        g = sum1*y%mod;
        h = (n+1)*y%mod*y%mod;
    }
    else if(a>=c || b>=c){
        auto[ff,gg,hh]=cal(a%c,b%c,c,n);
        f = (sum1*x%mod + (n+1)*y%mod + ff)%mod;
        g = (sum2*x%mod + sum1*y%mod + gg)%mod;
        h = sum2*x%mod*x%mod + (n+1)*y%mod*y%mod + 2*sum1%mod*x%mod*y%mod;
        h = (h + 2*y*ff%mod + 2*x*gg%mod + hh)%mod;
    }
    else{
        int m=(a*n+b)/c;
        auto [ff,gg,hh]=cal(c,c-b-1,a,m-1);
        f = (n*m%mod-ff+mod)%mod;
        g = inv_2*(2*sum1*m%mod - hh - ff + 2*mod)%mod;
```

```
        h = (n*m%mod*(m+1)%mod - 2*gg - 2*ff - f + 5*mod)%mod;
    }
    return {f,g,h};
}
```

同余方程

```
int Congruence(int a,int b,int mod){    // ax = b (% mod) 最小正整数解
    if(b==0)return 0;
    if(a==0)return -1;
    int g=__gcd(a,mod),m=mod/g;
    if(b%g)return -1;
    int x,y;
    exgcd(a/g,m,x,y);
    x=(x%m+m)%m;
    return x*b/g%m;
}
```

Excrt（扩展中国剩余定理）

```
int m[N],a[N],n;    // x=a[i] (mod m[i])
int excrt(){
    for(int i=2;i<=n;i++){
        int x,y;
        int g=exgcd(m[i-1],m[i],x,y),tmp=(a[i]-a[i-1]%m[i]+m[i])%m[i];
        if(tmp%g!=0)return -1;
        a[i]=a[i-1]+x*tmp/g%(m[i]/g)*m[i-1];
        m[i]=m[i]/g*m[i-1];
        a[i]=(a[i]%m[i]+m[i])%m[i];
    }
    return a[n];
}
```

Exgcd（扩展欧几里得）

```
// ax + by = gcd(a,b)
int exgcd(int a,int b,int &x,int &y){       //扩展欧几里得
    if(!b){
        x=1,y=0;
        return a;
    }
    int d=exgcd(b,a%b,x,y);
    int tmp=x;
    x=y;
    y=tmp-(a/b)*y;
    return d;
}

int inv(int x){     //逆元
    int a,b;
    exgcd(x,mod,a,b);
    return (a%mod+mod)%mod;
}
```

二元一次方程

```
// ax+by=c
void cal(int a,int b,int c){
    int g=__gcd(a,b);
    if(c%g!=0){     // 无解
        cout<<-1<<endl;
        return;
    }
    int x0,y0;      // ax0 + by0 = gcd(a,b)
    exgcd(a,b,x0,y0);
    int x1=x0*c/g,y1=y0*c/g;    // ax1 + by1 = c
    int dx=b/g,dy=a/g;    // 方程中 x 和 y 的最小偏差
    if(x1<=0){
        int cnt=-x1/dx+1;
        x1+=cnt*dx,y1-=cnt*dy;
    }
    if(x1>0){
        int cnt=(x1-1)/dx;
        x1-=cnt*dx,y1+=cnt*dy;
    }
    if(y1>0){   // 存在正整数解
        int x_min=x1,y_max=y1;      // 正整数解中 x 的最小值，y 的最大值
        int cnt=(y1-1)/dy;      // 正整数解的个数为 cnt+1
        x1+=cnt*dx,y1-=cnt*dy;
        int x_max=x1,y_min=y1;      // 正整数解中 x 的最大值，y 的最小值
        cout<<cnt+1<<' '<<x_min<<' '<<y_min<<' '<<x_max<<' '<<y_max<<endl;
    }
    else{
        cout<<x1<<' '<<y1<<endl;    // x 为最小正整数的一组解
        int cnt=-y1/dy+1;
        x1-=cnt*dx,y1+=cnt*dy;      // y 为最小正整数的一组解
        cout<<x1<<' '<<y1<<endl;
    }
}
```

二次剩余

```
int mod,I;
struct com{
    int r,i;
    com operator*(const com&a)const{
        com res{};
        res.r=(r*a.r%mod+i*a.i%mod*I%mod)%mod;
        res.i=(r*a.i+i*a.r)%mod;
        return res;
    }
};
com qpow(com a,int b){
    com ans{};
    ans.r=1,ans.i=0;
    while(b){
        if(b&1)ans=ans*a;
        b>>=1;
```

```
        a=a*a;
    }
    return ans;
}
int qpow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=ans*a%mod;
        b>>=1;
        a=a*a%mod;
    }
    return ans;
}
mt19937 Rnd(random_device{}());
int residue(int n){      // n 的二次剩余
    if(n==0)return 0;
    if(qpow(n,(mod-1)/2)!=1)return -1;
    int a=Rnd()%mod;
    while(!a || (qpow((a*a%mod-n+mod)%mod, (mod-1)/2))==1)a=Rnd()%mod;
    I=(a*a%mod-n+mod)%mod;
    com t{};
    t.r=a,t.i=1;
    t=qpow(t,(mod+1)>>1);
    // 两个二次剩余分别为 t.r 和 mod-t.r
    return min(t.r,mod-t.r);
}
```

求原根

```
int hav_g[N];      // 标记有无原根
int cnt,prime[N],vis[N],phi[N];
void init(){
    for(int i=2;i<N;i++){
        if(!vis[i])prime[++cnt]=i,phi[i]=i-1;
        for(int j=1;j<=cnt && i*prime[j]<N;j++){
            vis[i*prime[j]]=1;
            if(i%prime[j]==0){
                phi[i*prime[j]]=phi[i]*prime[j];
                break;
            }
            else phi[i*prime[j]]=phi[i]*phi[prime[j]];
        }
    }
    hav_g[2]=hav_g[4]=1;
    for(int i=2;i<=cnt;i++){         // prime[1]=2 ，不能标记原根
        for(int j=prime[i];j<N;j*=prime[i]){
            hav_g[j]=1;
            if(2*j<N)hav_g[2*j]=1;
        }
    }
}
int qpow(int a,int b,int p){
    int ans=1;
    while(b){
        if(b&1)ans=ans*a%p;
```

```
        b>>=1;
        a=a*a%p;
    }
    return ans;
}
vector<int>fac;     // phi[p] 的质因数
bool check(int x,int p){    // 判断 x 是否为 p 的原根
    if(qpow(x,phi[p],p)!=1)return false;
    for(auto i:fac){
        if(qpow(x,phi[p]/i,p)==1)return false;
    }
    return true;
}
int find_g(int p){     // 返回 p 的一个原根
    if(!hav_g[p])return 0;
    fac.clear();
    int now=phi[p];
    for(int i=2;i*i<=phi[p];i++){
        if(now%i==0){
            fac.push_back(i);
            while(now%i==0)now/=i;
        }
    }
    if(now>1)fac.push_back(now);
    for(int i=1;i<p;i++)
        if(check(i,p))return i;
}
vector<int>all_g(int p){    // p 的所有原根
    vector<int>ans;
    int g=find_g(p);
    if(g==0)return ans;
    int now=1;
    for(int i=1;i<=phi[p];i++){
        now=now*g%p;
        if(__gcd(i,phi[p])==1)ans.push_back(now);
    }
    sort(ans.begin(), ans.end());
    return ans;
}
```

欧拉降幂

```
int get_phi(int x){     //单点欧拉函数
    int res=x,tmp=sqrt(x);
    for(int i=2;i<=tmp;i++){
        if(x%i==0){
            res=res/i*(i-1);
            while(x%i==0)x/=i;
        }
    }
    if(x>1)res=res/x*(x-1);
    return res;
}
bool check(int a,int n,double phi){      //判断a的n层幂塔是否大于phi
    if(n==0)return 1>=phi; //0层幂塔是1
```

```
    if(a>=phi) return 1;
    return check(a,n-1,log(phi)/log(a));
}
int exEular(int a,int n,int m){     //a的n层幂塔对mod取模
    if(m==1)return 0;
    if(n<=1)return qpow(a,n,m);
    int phi=get_phi(m);
    if(__gcd(a,m)==1)return qpow(a,exEular(a,n-1,phi),m);
    if(check(a,n-1,phi))return qpow(a,exEular(a,n-1,phi)+phi,m);
    return qpow(a,exEular(a,n-1,phi),m);
}
```

整除分块

```
int pre(int x){     //前缀和函数
    return x*(x+1)/2;
}
void solve(){
    int n,k;
    cin>>n>>k;
    int ans=0;
    for(int l=1,r;l<=n;l=r+1){      // n 是 i 的取值范围
        if(k/l==0)break;
        r=min(k/(k/l),n);           // k 是整除分块的分子
        ans+=(pre(r)-pre(l-1))*(k/l);
    }
    return;
}

int cal(){          // \sum^n [x/i]*[y/i]
    int ans=0;
    for(int l=1,r;l<=n;l=r+1){
        int r1,r2;
        if(x/l==0)r1=n;         //两项乘积，有2x个端点，每次转移到最近一个
        else r1=min(x/(x/l),n);
        if(y/l==0)r2=n;
        else r2=min(y/(y/l),n);
        r=min(r1,r2);
        ans+=(pre2(r)-pre2(l-1))*(x/l)*(y/l);
    }
    return ans;
}
```

离散对数

BSGS

```
#define int long long
int mod;
int qpow(int a,int b){
    int ans=1;
    while(b>0){
        if(b&1)
            ans=(ans*a)%mod;
```

```
        a=(a*a)%mod;
        b>>=1;
    }
    return ans;
}
int bsgs(int a,int b,int p){    //求 a^x=b(mod p) 的最小非负数x
    mod=p;
    a%=mod,b%=mod;
    if(b==1)return 0;
    int mx=sqrt(p)+1;
    map<int,int>mp;     // 存b*a^B
    int tmp=b;
    for(int i=0;i<=mx;i++){
        mp[tmp]=i;
        tmp=tmp*a%mod;
    }
    tmp=qpow(a,mx);     // 匹配 a^(A*mx)
    int aa=tmp;
    set<int>ans;
    for(int i=1;i<=mx;i++){
        if(mp.find(aa)!=mp.end()){
            ans.insert(i*mx-mp[aa]);
        }
        aa=aa*tmp%mod;
    }
    for(auto i:ans){
        if(i<0)continue;
        return i;
    }
    return -1;
}
```

扩展BSGS

线性求逆元

```
void get_inv(){
    inv[1]=1;
    for(int i=2;i<=n;i++){
        inv[i]=(mod-mod/i)*inv[mod%i]%mod;
    }
}
```

MR判断大质数

```
#define int long long
int qpow(int a,int b,int mod){
    int ans=1;
    a%=mod;
    while(b){
        if(b&1)ans=(__int128)ans*a%mod;
        a=(__int128)a*a%mod;
        b>>=1;
    }
```

```
    return ans;
}
bool MR(int p){
    //srand(time(NULL));
    if(p<2)return 0;
    if(p==2||p==3||p==5||p==7)return 1;
    if(p%2==0)return 0;
    int d=p-1,r=0;
    while(!(d&1)){
        r++;
        d>>=1;
    }
    for(int i=0;i<10;i++){
        int a=rand()%(p-2)+2;
        int x=qpow(a,d,p);
        if(x==1||x==p-1)continue;
        for(int j=0;j<r-1;j++){
            x=(__int128)x*x%p;
            if(x==p-1)break;
        }
        if(x!=p-1)return 0;
    }
    return 1;
}
void solve(){
    srand(time(NULL));
    int x;
    cin>>x;
    if(MR(x))cout<<"YES"<<endl;
    else cout<<"NO"<<endl;
}
```

PR分解大数质因数

```
int gcd(int x,int y){   //最大公约数
    while(y^=x^=y^=x%=y);
    return x;
}
int PR(int x){
    //srand(time(NULL));
    int t=0,s=0;
    int c=(int)rand()%(x-1)+1;
    int goal=1,val=1;
    while(1){
        for(int step=1;step<=goal;step++){
            t=((__int128)t*t+c)%x;
            val=(__int128)val*abs(t-s)%x;
            if(step%127==0){
                int d=gcd(val,x);
                if(d>1)return d;
            }
        }
        int d=gcd(val,x);
        if(d>1)return d;
        s=t;
```

```
        val=1;
        goal*=2;
    }
}
int ans;
vector<int>fac;         //会有重复的质因子
void get_prime(int x){
    if(x<=ans)return;
    if(x<2)return;
    if(MR(x)){
        ans=max(ans,x);
        //fac.push_back(x);     //fac存所有质因数
        return;
    }
    int p=x;
    while(p==x)p=PR(x);
    while(x%p==0)x/=p;
    get_prime(x);
    get_prime(p);
}
void solve(){
    srand(time(NULL));
    int x;cin>>x;
    ans=0;
    if(MR(x)){
        cout<<"Prime"<<endl;
        return;
    }
    //fac.clear();
    get_prime(x);
    cout<<ans<<endl;
}
```

组合数学

组合数

```
int fact[N],inv[N];
void Cinit(){
    fact[0]=fact[1]=1;
    inv[0]=inv[1]=1;
    for(int i=2;i<N;i++)fact[i]=fact[i-1]*i%mod;
    for(int i=2;i<N;i++)inv[i]=(mod-mod/i)*inv[mod%i]%mod;
    for(int i=2;i<N;i++)inv[i]=inv[i]*inv[i-1]%mod;
}
int C(int a,int b){
    if(a<0||b<0||a<b)return 0;
    return fact[a]*inv[a-b]%mod*inv[b]%mod;
}
```

Lucas定理

```
int fact[N],inv[N];
void init(int p){        // p为质数
    fact[0]=fact[1]=1;
    inv[0]=inv[1]=1;
    for(int i=2;i<N;i++)fact[i]=fact[i-1]*i%p;
    for(int i=2;i<N;i++)inv[i]=(p-p/i)*inv[p%i]%p;
    for(int i=2;i<N;i++)inv[i]=inv[i]*inv[i-1]%p;
}
int Lucas(int n,int m,int p){    // C(n,m)%p
    if(n<m)return 0;
    if(n<p) return fact[n]*inv[m]%p*inv[n-m]%p;
    return Lucas(n%p,m%p,p)*Lucas(n/p,m/p,p)%p;
}
```

扩展Lucas定理

```
// C(n,m)%p , p 非质数
int id=0;
int prime[N],power[N],fac[N];
int getfac(int x,int p,int mod){
    if(x<=p)return fac[x];
    return fac[x%mod]*qpow(fac[mod],x/mod%(mod/p*(p-
1)),mod)%mod*getfac(x/p,p,mod)%mod;
}
int getans(int n,int m,int p,int mod){
    fac[0]=1;
    for(int i=1;i<=min(mod,N-1);i++)fac[i]=(i%p==0)?fac[i-1]:fac[i-1]*i%mod;
    int up=getfac(n,p,mod),down=getfac(m,p,mod)*getfac(n-m,p,mod)%mod;
    int L=0,x=n;
    while(x)L+=x/p,x/=p;
    x=m;
    while(x)L-=x/p,x/=p;
    x=n-m;
    while(x)L-=x/p,x/=p;
    for(int i=1;i<=L;i++){
        up=up*p%mod;
        if(!up)return 0;
    }
    return up*qpow(down,mod/p*(p-1)-1,mod)%mod;
}
int exgcd(int a,int b,int &x,int &y){        //扩展欧几里得
    if(!b){
        x=1,y=0;
        return a;
    }
    int d=exgcd(b,a%b,x,y);
    int tmp=x;
    x=y;
    y=tmp-(a/b)*y;
    return d;
}
int crt(int a,int b,int mod1,int mod2){
```

```
    int x,y;
    exgcd(mod1,mod2,x,y);
    return ((x+mod1*mod2)*(b-a+mod1*mod2)%(mod1*mod2)*mod1+a)%(mod1*mod2);
}
int exlucas(int n,int m,int p){
    int tmp=p,mx=sqrt(p)+1;
    for(int i=2;i<=mx;i++){
        if(tmp%i==0){
            prime[++id]=i,power[id]=1;
            while(tmp%i==0)tmp/=i,power[id]*=i;
        }
    }
    if(tmp>1)prime[++id]=tmp,power[id]=tmp;
    int ans=getans(n,m,prime[1],power[1]);
    int mod=power[1];
    for(int i=2;i<=id;i++){
        ans=crt(ans,getans(n,m,prime[i],power[i]),mod,power[i]);
        mod*=power[i];
    }
    return ans;
}
```

卡特兰数

```
// 卡特兰数，从 (0,0) 往上走 u 步，往下走 d 步，走到 (u+d,u-d)，期间位于 x 轴上部分，不越过 x 轴的方案数
int ktl(int u,int d){
    return (C(u+d,d)-C(u+d,d-1)+mod)%mod;
}
```

第二类斯特林数

```
int S[N][N];       // n 个不同的球放在 m 个相同的盒子里，盒子非空
void init(){
    S[1][1]=1;
    for(int i=2;i<N;i++){
        for(int j=1;j<=i;j++){
            S[i][j]=S[i-1][j-1]+S[i-1][j]*j;
        }
    }
}
```

盒子与球模型

```
// n 个球放入 m 个盒子

// 球不同，盒子不同
int cal1(int n,int m){
    return qpow(m,n);
}
// 球不同，盒子不同，盒子至多装一个球
int cal2(int n,int m){
    // A(m,n)
```

```
    if(n>m)return 0;
    return fact[m]*inv[m-n]%mod;
}
// 球不同，盒子不同，盒子至少装一个球
int cal3(int n,int m){
    // S(n,m)*m!
    int ans=0;
    for(int i=0;i<=m;i++){
        int tmp=inv[i]*inv[m-i]%mod*qpow(i,n)%mod;
        if((m-i)%2==0)ans=(ans+tmp)%mod;
        else ans=(ans-tmp)%mod;
    }
    ans=(ans+mod)*fact[m]%mod;
    return ans;
}

// 球不同，盒子相同
int cal4(int n,int m){
    // sum{S(n,i)}  |  i=[1,m]
    VI a(m+1,0),b(m+1,0);
    b[0]=1;
    for(int i=0;i<=m;i++){
        a[i]=qpow(i,n)*inv[i]%mod;
        if(i%2==0)b[i]=inv[i];
        else b[i]=(mod-1)*inv[i]%mod;
    }
    a=a*b;
    int ans=0;
    for(int i=1;i<=m;i++){
        ans=(ans+a[i])%mod;
    }
    return ans;
}
// 球不同，盒子相同，盒子至多装一个球
int cal5(int n,int m){
    if(n>m)return 0;
    return 1;
}
// 球不同，盒子相同，盒子至少装一个球
int cal6(int n,int m){
    // S(n,m)
    int ans=0;
    for(int i=0;i<=m;i++){
        int tmp=inv[i]*inv[m-i]%mod*qpow(i,n)%mod;
        if((m-i)%2==0)ans=(ans+tmp)%mod;
        else ans=(ans-tmp)%mod;
    }
    ans=(ans+mod)%mod;
    return ans;
}

// 球相同，盒子不同
int cal7(int n,int m){
    return C(n+m-1,m-1);
}
```

```
// 球相同，盒子不同，盒子至多装一个球
int cal8(int n,int m){
    return C(m,n);
}
// 球相同，盒子不同，盒子至少装一个球
int cal9(int n,int m){
    return C(n-1,m-1);
}

// 球相同，盒子相同
int cal10(int n,int m){
    // p(n,m)      |     划分数，表示将正整数 n 划分为 m 个自然数的可重集的方案数
    VI f(n+1,0);
    for(int k=1;k<=m;k++){
        for(int i=1;i*k<=n;i++){
            f[i*k]=(f[i*k]+iv[i])%mod;
        }
    }
    f=get_exp(f);
    int ans=f[n];
    return ans;
}
// 球相同，盒子相同，盒子至多装一个球
int cal11(int n,int m){
    if(n>m)return 0;
    return 1;
}
// 球相同，盒子相同，盒子至少装一个球
int cal12(int n,int m){
    // p(n,n-m)
    if(n<m)return 0;
    VI f(n+1,0);
    for(int k=1;k<=n-m;k++){
        for(int i=1;i*k<=n;i++){
            f[i*k]=(f[i*k]+iv[i])%mod;
        }
    }
    f=get_exp(f);
    return f[n-m];
}
```

SG函数打表

```
bool vis[1005];
int sg[1000];
int n=100;
void solve(){
    sg[0]=0,sg[1]=1;
    for(int i=2;i<n;i++){
        memset(vis,0,sizeof(vis));
        for(int j=1;j<=i;j++){
            vis[sg[i-j]]=1;
        }
        for(int j=1;j<i;j++){
            vis[sg[j]^sg[i-j]]=1;
```

```
        }
        int j=0;
        while(vis[j])j++;
        sg[i]=j;
    }
    for(int i=0;i<n;i++){
        cout<<"sg["<<i<<"] : "<<sg[i]<<endl;
    }
    return;
}
```