ACM-ICPC 模板

HDU2020　　Team：HDUNchar66

应以周　　舒飘扬　　郑豪杰

目录

1 数论

1.1 埃拉托斯特尼筛

时间复杂度为调和级数 $O(nlnn)$

```
bool vis[maxn+5];
void sieve(){
    for(int i=1;i<=maxn;++i){
        if(!vis[i]){//i是质数
            for(int j=i+i;j<=maxn;j+=i)vis[j]=1;
        }
    }
}
```

1.2 数论函数

设条件表达式 $[x]=\begin{cases}1,x=true\\0,x=false\end{cases}$

单位元函数：$\varepsilon(n)=[n=1]$

一函数：$1(n)=1$

自身函数：$id(n)=n$

欧拉函数：$\varphi(n)=[1,n]$和n互质的整数个数$=\sum_{i=1}^{n}[\gcd(i,n)=1]=n\times\prod_{p_i|n}\frac{p_i-1}{p_i}$，$\varphi(ij)=\frac{\varphi(i)\varphi(j)g}{\varphi(g)}$

莫比乌斯函数：$\mu(n)=\begin{cases}1,n=1\\(-1)^k,n=p_1p_2...p_k\\0,n\text{有平方因子}\end{cases}$

因数个数：$d(n=p_1^{a_1}p_2^{a_2}...p_k^{a_k})=(1+a_1)(1+a_2)...(1+a_k)$

因数和：$\sigma(n=p_1^{a_1}p_2^{a_2}...p_k^{a_k})=\frac{1-p_1^{a_1+1}}{1-p_1}\frac{1-p_2^{a_2+1}}{1-p_2}...\frac{1-p_k^{a_k+1}}{1-p_k}$

因数幂和：$\sigma_k(n)=\sum_{d|n}d^k$

积性函数：满足 $f(xy)=f(x)f(y)\iff\gcd(x,y)=1$，完全积性函数：满足 $f(xy)=f(x)f(y)$

1.3 线性筛

$[1,n]$ 的正整数中质数个数大约有 $\frac{n}{\ln n}$ 个

积性函数可以线性筛，每个数被最小质因子递筛一次

线性筛积性函数 $f(n)$ 时，需要分三类情况递推函数值

1、i 是质数：直接得出 $f(i)$

2、$i\%p_j\neq0,p_j$ 是 $i\times p_j$ 的最小质因数，$f(i\times p_j)=f(i)f(p_j)$

3、$i\%p_j=0,p_j$ 是 $i\times p_j$ 的最小质因数，且存在 p_j 的平方因子，需要根据函数定义计算贡献

有时需要维护最小质因子的幂次 $lpfpow,lpfcnt$

$\varphi(i)=i\prod_{p_k|i}\frac{p_k-1}{p_k}\to\varphi(i\times p_j)=i\times p_j\prod_{p_k|i\times p_j}\frac{p_k-1}{p_k}=\varphi(i)\times p_j$

$\mu(i \times p_j) = 0$，因为有平方因子 p_j^2
$d(p^k) = k+1, \sigma(p^k) = \frac{1-p^k p}{1-p}$
$d(i \times p_j) = d(lpfpow(i \times p_j))d(i \times p_j / lpfpow(i \times p_j))$

```
const int maxn=1e6,N=maxn+4;
int p[N],hp;//p[1~hp]是质数
int lpf[N],lpfpow[N],lpfcnt[N];//最小质因子,lpf的幂,lpf的指数大小
int phi[N],mu[N];//欧拉函数、莫比乌斯函数
int d[N];//约数个数
ll sigma[N];//约数和
void sieve(int n){
    static bool vis[maxn+5];
    phi[1]=mu[1]=d[1]=sigma[1]=1;
    for(int i=2;i<=n;++i){
        if(!vis[i]){//1、i是质数
            p[++hp]=i;mu[i]=-1;phi[i]=i-1;
            lpf[i]=i;lpfpow[i]=i;lpfcnt[i]=1;
            d[i]=2;sigma[i]=i+1;
        }
        for(int j=1,x;j<=hp&&(x=i*p[j])<=n;++j){//maxn<=2e8不会爆int
            vis[x]=1; lpf[x]=p[j];
            if(i%p[j]==0){//2、p[j]是i的最小质因子
                lpfpow[x]=lpfpow[i]*p[j];
                lpfcnt[x]=lpfcnt[i]+1;
                phi[x]=phi[i]*p[j]; mu[x]=0;
                if(lpfpow[x]==x){
                    d[x]=lpfcnt[x]+1;
                    sigma[x]=(1-1ll*x*p[j])/(1-p[j]);
                }
                else{
                    d[x]=d[lpfpow[x]]*d[x/lpfpow[x]];
                    sigma[x]=sigma[lpfpow[x]]*sigma[x/lpfpow[x]];
                }
                break;
            }
            //3、p[j]是i*p[j]的最小质因子
            lpfcnt[x]=1;
            lpfpow[x]=p[j];
            phi[x]=phi[i]*phi[p[j]];
            mu[x]=-mu[i];
            d[x]=d[i]*d[p[j]];
            sigma[x]=sigma[i]*sigma[p[j]];
```

```
        }
    }
}
vector<int> factorize(int x){//对maxn范围内分解质因数O(logn)
    vector<int>v;
    while(x>1){
        int t=lpf[x];
        v.push_back(t);
        while(x%t==0)x/=t;
    }
    return v;
}
vector<int> factorize(ll x){//对maxn^2范围质因数分解O(根号n/logn)
    vector<ll>v;
    for(int i=1;i<=hp&&1ll*p[i]*p[i]<=x){
        if(x%p[i]==0){
            v.push_back(p[i]);
            while(x%p[i]==0)x/=p[i];
        }
    }
    if(x>1)v.push_back(x);//大于根号的质因数最多一个
    return v;
}
```

1.4 扩展欧几里得算法

裴蜀定理：对任何整数 a,b,x,y，$ax+by$ 是 $\gcd(a,b)$ 的倍数，且不定方程 $ax+by=\gcd(a,b)$ 一定有解

若 $a<0$，则把 $ax+by=\gcd(a,b)$ 转化为 $|a|(-x')+by=\gcd(|a|,b)$

用扩展欧几里得算法可以求解不定方程 $ax+by=c$，设 $g=\gcd(a,b)$

若 c 不是 g 的倍数，则无解

若有解，得到一组特解 x_0,y_0，且 $x_0+k\frac{b}{g},y_0-k\frac{a}{g}$ 是通解，x 的最小正整数解为 $x_0+k\frac{b}{g}$ 的最小正整数值 min_x，min_x 对应了 y 的最大正整数解 $max_y=\frac{c-a\cdot min_x}{b}$，若 $max_y\le 0$，则不存在 x,y 均为正整数的解

求 $ax\equiv 1\pmod b$，即求 $ax+by=1$ 的解中 x 的最小正整数值，有解条件是 $\gcd(a,b)=1$

```
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(!b){x=1;y=0;return a;}
    ll g=exgcd(b,a%b,y,x);y-=a/b*x;
    return g;
}
void solve(ll a,ll b,ll c){// 求解不定方程 ax+by = c
    ll x,y; ll g=exgcd(a,b,x,y);
    if(c%g!=0){}//无解
```

```
    a/=g;b/=g;c/=g;x*=c;y*=c;
    ll minx=x%b+(x%b<=0)*b,maxy=(c-a*minx)/b;// x最小正整数解，y最大正整数解
    ll miny=y%a+(y%a<=0)*a,maxx=(c-b*miny)/a;// y最小正整数解，x最大正整数解
    if(maxy<=0){}//没有x,y均为正整数的解
    else{}//正整数解有cnt=(maxx-minx)/b+1个,minx+k*b,maxy-k*a, 0<=k<cnt
}
```

1.5 整除分块

$\lfloor \frac{n}{x} \rfloor$ 的取值有 $2\sqrt{n}$ 种

```
for(ll l=1,r;l<=n;l=r+1){//O(n根号n)
    ll t=n/l;r=n/t;//对n和m同步整除分块时,r=min(n/(n/l),m/(m/l));
    //任意i属于[l,r],n/i==t
}
```

1.6 快速幂、费马小定理

费马小定理：$a^{p-1} \equiv 1 \ (mod \ p)$

```
ll qpow(ll a,ll b){
    ll s=1;
    for(;b;b>>=1){
        if(b&a)s=s*a%mod;
        a=a*a%mod;
    }
    return s;
}
ll inva = qpow(a,mod-2);//a的逆元
```

1.7 线性递推逆元

```
int inv[N];
void invwork(){//递推逆元
    inv[1]=1;
    for(int i=2;i<N;++i){
        inv[i]=(1ll*mod-mod/i)*inv[mod%i]%mod;
    }
}
```

```
ll fac[N],ifac[N],inv[N];
ll C(int n,int m){
```

```
    return fac[n]*ifac[m]%mod*ifac[n-m]%mod;
}
int main(){
    fac[0]=ifac[0]=1;
    for(int i=1;i<N;++i)fac[i]=fac[i-1]*i%mod;
    ifac[N-1]=qpow(fac[N-1],mod-2);
    for(int i=N-2;i;--i)ifac[i]=ifac[i+1]*(i+1)%mod;
    for(int i=1;i<N;++i)inv[i]=ifac[i]*fac[i-1]%mod;
}
```

1.8 欧拉定理/欧拉降幂

欧拉定理：对任意 $a \in \mathbb{Z}, m \in \mathbb{N}^+$，若 $\gcd(a,m)=1$，则 $a^{\varphi(m)} \equiv 1 \pmod m$

欧拉降幂：

$$a^b \equiv \begin{cases} a^{b \mod \varphi(m)}, \gcd(a,m)=1 \\ a^b, \gcd(a,m) \neq 1 \wedge b < \varphi(m) \\ a^{b \mod \varphi(m)+\varphi(m)}, \gcd(a,m) \neq 1 \wedge b \geq \varphi(m) \end{cases} \pmod m$$

注意当 $\gcd(a,m)=1 \wedge b < \varphi(m)$ 时，$a^b \not\equiv a^{b \mod \varphi(m)+\varphi(m)}, (a=2,b=1,m=4)$

求 $a^{a^{a^{\cdots}}} \pmod p$ （无限个 a）

```
ll qpow(ll a,ll b,ll mod){
    ll s=1;
    for(;b;b>>=1){
        if(b&1)s=s*a%mod;
        a=a*a%mod;
    }
    return s;
}
ll f(ll a,int mod){
    if(mod==1)return 0;
    return qpow(a,f(a,phi[mod])+phi[mod],mod);
}
```

1.9 狄利克雷卷积, 莫比乌斯反演

狄利克雷卷积: 若 h 为 f 和 g 的狄利克雷卷积，记作 $h=f*g$, 则 $h(n)=\sum\limits_{d|n} f(d)g(\frac{n}{d})$

莫比乌斯反演：若 $f=g*1 \iff g=f*\mu$

常见狄利克雷卷积公式：$\varepsilon=\mu*1$, $id=\varphi*1, \mu*id=\varphi$

例题，保证 $n \leq m$

1、$\sum_{i=1}^{n}\gcd(i,n)=\sum_{g|n}g\sum_{i=1}^{\frac{n}{g}}[\gcd(i,\frac{n}{g})=1]=\sum_{g|n}g\varphi(\frac{n}{g})$

2、$\sum_{i=1}^{n}\mathrm{lcm}(i,n)=\sum_{i=1}^{n}\frac{in}{\gcd(i,n)}=\sum_{g|n}\sum_{i=1}^{n}\frac{in}{g}[\gcd(i,n)=g]=\sum_{g|n}\sum_{i=1}^{\frac{n}{g}}in[\gcd(i,\frac{n}{g})=1]=n\sum_{g|n}\sum_{i=1}^{g}i[\gcd(i,g)=1]$

其中 $\sum_{i=1}^{n}i[\gcd(i,n)=1]=\sum_{i=1}^{n}i\sum_{d|\gcd(i,n)}\mu(d)=\sum_{d|n}d\mu(d)\sum_{i=1}^{\frac{n}{d}}i=\sum_{d|n}d\mu(d)\frac{1}{2}(\frac{n}{d}+\frac{n^2}{d^2})=\frac{n\varepsilon(n)+n\varphi(n)}{2}$

进而得到 $n\sum_{g|n}\sum_{i=1}^{g}i[\gcd(i,g)=1]=n\sum_{g|n}\frac{\varepsilon(g)+g\varphi(g)}{2}$

3、$\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)=1]=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d|\gcd(i,j)}\mu(d)=\sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor$

4、$\sum_{i=1}^{n}\sum_{j=1}^{m}\gcd(i,j)=\sum_{g=1}^{n}g\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)=g]=\sum_{g=1}^{n}g\sum_{i=1}^{\lfloor\frac{n}{g}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{g}\rfloor}[gcd(i,j)=1]$

$=\sum_{g=1}^{n}g\sum_{i=1}^{\lfloor\frac{n}{g}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{g}\rfloor}\sum_{d|gcd(i,j)}\mu(d)=\sum_{g=1}^{n}g\sum_{d=1}^{\lfloor\frac{n}{g}\rfloor}\mu(d)\lfloor\frac{n}{gd}\rfloor\lfloor\frac{m}{gd}\rfloor$

$=(T=gd)\sum_{T=1}^{n}\lfloor\frac{n}{T}\rfloor\lfloor\frac{m}{T}\rfloor\sum_{g|T}g\mu(\frac{T}{g})=\sum_{T=1}^{n}\lfloor\frac{n}{T}\rfloor\lfloor\frac{m}{T}\rfloor\varphi(T)$

5、$\sum_{i=1}^{n}\sum_{j=1}^{m}\mathrm{lcm}(i,j)=\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{ij}{\gcd(i,j)}=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{g=1}^{min(i,j)}\frac{ij}{g}[\gcd(i,j)=g]$

$=\sum_{g=1}^{n}\sum_{i=1}^{\lfloor\frac{n}{g}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{g}\rfloor}ijg[\gcd(i,j)=1]$　　(原先的 i,j 被替换为枚举 g 的 i,j 倍)

$=\sum_{g=1}^{n}\sum_{i=1}^{\lfloor\frac{n}{g}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{g}\rfloor}ijg\sum_{d|\gcd(i,j)}\mu(d)=\sum_{g=1}^{n}g\sum_{d=1}^{\lfloor\frac{n}{g}\rfloor}\mu(d)d^2\sum_{i=1}^{\lfloor\frac{n}{gd}\rfloor}i\sum_{j=1}^{\lfloor\frac{m}{gd}\rfloor}j$

$=(T=gd,f(n)=\frac{n(n+1)}{2})\sum_{g=1}^{n}g\sum_{d=1}^{\lfloor\frac{n}{g}\rfloor}\mu(d)d^2f(\lfloor\frac{n}{T}\rfloor)f(\lfloor\frac{m}{T}\rfloor)$

$=\sum_{T=1}^{n}f(\lfloor\frac{n}{T}\rfloor)f(\lfloor\frac{m}{T}\rfloor)\sum_{g|T}g\mu(\frac{T}{g})(\frac{T}{g})^2=\sum_{T=1}^{n}f(\lfloor\frac{n}{T}\rfloor)f(\lfloor\frac{m}{T}\rfloor)\sum_{d|T}d\mu(d)T$

6、$\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)是质数]=\sum_{p\in\mathbb{P}}^{n}\sum_{i=1}^{n}\sum_{j=1}^{m}[\gcd(i,j)=p]=\sum_{p\in\mathbb{P}}^{n}\sum_{i=1}^{\lfloor\frac{n}{p}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{p}\rfloor}[\gcd(i,j)=1]$

$=\sum_{p\in\mathbb{P}}^{n}\sum_{i=1}^{\lfloor\frac{n}{p}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{p}\rfloor}\sum_{d|\gcd(i,j)}\mu(d)=\sum_{p\in\mathbb{P}}^{n}\sum_{d=1}^{\lfloor\frac{n}{p}\rfloor}\mu(d)\lfloor\frac{n}{pd}\rfloor\lfloor\frac{m}{pd}\rfloor$

$=(令T=pd)\sum_{p\in\mathbb{P}}^{n}\sum_{d=1}^{\lfloor\frac{n}{p}\rfloor}\mu(d)\lfloor\frac{n}{T}\rfloor\lfloor\frac{m}{T}\rfloor=\sum_{T=1}^{n}\lfloor\frac{n}{T}\rfloor\lfloor\frac{m}{T}\rfloor\sum_{p|T,p\in\mathbb{P}}\mu(\frac{T}{p})$

7、$d(n)=n$ 的约数个数，$d(ij)=\sum_{x|i}\sum_{y|j}[\gcd(i,j)=1]$

$\sum_{i=1}^{n}\sum_{j=1}^{m}d(ij)=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{x|i}\sum_{y|j}[\gcd(i,j)=1]=\sum_{i=1}^{n}\sum_{j=1}^{m}\lfloor\frac{n}{i}\rfloor\lfloor\frac{m}{j}\rfloor[\gcd(i,j)=1]$　(xy 提到前面, 再把 xy 写成 ij)

$= \sum_{i=1}^{n}\sum_{j=1}^{m}\lfloor\frac{n}{i}\rfloor\lfloor\frac{m}{j}\rfloor\sum_{d|\gcd(i,j)}\mu(d) = \sum_{d=1}^{n}\mu(d)\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor}\lfloor\frac{n}{id}\rfloor\sum_{j=1}^{\lfloor\frac{m}{d}\rfloor}\lfloor\frac{m}{jd}\rfloor$ 预处理 $\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$, 对 μ 求前缀和，对 n 和 m 整除分块

1.10 中国剩余定理

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ ... \\ x \equiv a_n \pmod{m_n} \end{cases}$$

保证 m_i, m_j 两两互质，设 $s = \prod_{i=1}^{n} m_i$

解为 $x = (\sum a_i * \frac{s}{m_i} * (\frac{1}{\frac{s}{m_i}} \pmod{m_i}))) \mod s$

```
typedef __int128 ll;//可能会爆ll
ll n,m[15],a[15],s;
ll ans;
void exgcd(ll a,ll b,ll &x,ll &y){
    if(!b)x=1,y=0;
    else exgcd(b,a%b,y,x),y-=a/b*x;
}
int main(){
    cin>>n;s=1;
    for(int i=1;i<=n;++i){
        cin>>m[i]>>a[i];
        s*=m[i];//需要保证m[i]之积在__int128范围之内
    }
    for(int i=1;i<=n;++i){
        ll x,y,t=s/m[i];
        exgcd(t,m[i],x,y);
        x=(x+m[i])%m[i];
        ans+=a[i]*t*x;
    }
    printf("%lld",ans%s);
}
```

扩展中国剩余定理：

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ ... \\ x \equiv a_n \pmod{m_n} \end{cases}$$

不保证 m_i, m_j 两两互质

设 $lcm_i = lcm(m_1, m_2...m_i)$，$ans_i =$ 方程 $1 \sim i$ 的解

方程 $1 \sim i-1$ 的通解为 $ans_{i-1} + t * lcm_{i-1}$

解同余方程 $ans_{i-1} + t * lcm_{i-1} \equiv a_i \pmod{m_i}$

即 $lcm_{i-1} * t + m_i * k = a_i - ans_{i-1}$

若该同余方程无解，则原问题无解，反之用 exgcd 求解出 t

递推式 $ans_i = ans_{i-1} + t \times lcm_{i-1}$

```
int n; ll m[N],a[N];
__int128 exgcd(__int128 a,__int128 b,__int128 &x,__int128 &y){
    if(!b){x=1;y=0;return a;}
    __int128 g=exgcd(b,a%b,y,x);y-=a/b*x;
    return g;
}
ll excrt(){
    __int128 lcm=m[1],ans=a[1],x,y;
    for(int i=2;i<=n;++i){
        __int128 g=exgcd(lcm,m[i],x,y);
        if((a[i]-ans)%g)return -1;
        x*=(a[i]-ans)/g;
        ans+=x*lcm;
        lcm=lcm/g*m[i];
        ans%=lcm;
    }
    return (ans%lcm+lcm)%lcm;
}
int main(){
    scanf("%d",&n);for(int i=1;i<=n;++i)scanf("%lld%lld",&m[i],&a[i]);
    printf("%lld",excrt());
}
```

1.11 BSGS/exBSGS

在 $O(\sqrt{\varphi(p)}\ )$ 的时间复杂度内求 $a^x \equiv b(mod\ \ p)$

```
ll qpow(ll a,ll b,ll p){
    ll s=1;
    for(;b;b>>=1){
        if(b&1)s=s*a%p;
        a=a*a%p;
    }
    return s;
}
```

```
ll bsgs(ll a,ll b,ll p){//p是质数,求a^x=b(mod p)最小自然数x
    unordered_map<ll,ll>mp;
    b%=p;  ll t=sqrt(p)+1;
    for(ll i=1;i<=t;++i){
        b=b*a%p;
        mp[b]=i;
    }
    a=qpow(a,t,p);
    if(!a)return b==0?1:-1;
    ll f=1;
    for(ll i=1;i<=t;++i){
        f=f*a%p;
        int j=mp.find(f)==mp.end()?-1:mp[f];
        if(j>=0&&i*t-j>=0)return i*t-j;
    }
    return -1;//无解
}
//----------exBSGS
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(!b){x=1;y=0;return a;}
    ll g=exgcd(b,a%b,x,y);
    ll t=x;x=y;y=t-a/b*y;
    return g;
}
ll exbsgs(ll a,ll b,ll p) {// a^x=b(mod p)最小自然数x
    if(b==1||p==1)return 0;
    b%=p;ll f=1;
    for (int i = 0; i < 30; i++) {
        if(f==b)return i;
        f=f*a%p;
    }
    ll x,y;
    ll g=exgcd(f,p,x,y);
    if(x<0)x+=p;
    if(b%g)return -1;
    f/=g,b/=g,p/=g;
    b=b*x%p;
    ll ret=bsgs(a,b,p);
    return ~ret?ret+30:-1;//-1表示无解
}
```

1.12 原根

阶：a 模 m 的阶 $\delta_m(a)$ = 最小的正整数 n 使得 $a^n \equiv 1 \pmod m$

$a, a^2, ..., a^{\delta_m(a)}$ 两两不同余 $\pmod m$

若 g 是质数 p 的原根，则 $g, g^2, ..., g^{p-1}$ 两两不同余 $\pmod p$

若 $a^n \equiv 1 \pmod m$，则 $\delta_m(a)|n$

若 $a^p \equiv a^q \pmod m$，则 $p \equiv q \pmod{\delta_m(a)}$

若 $m \in \mathbb{N}^+, a, b \in \mathbb{Z}$，则 $\delta_m(ab) = \delta_m(a)\delta_m(b) \iff \gcd(\delta_m(a), \delta_m(b)) = 1$

原根判定定理：设 $m \geq 3, \gcd(a, m) = 1$，则 a 是模 m 的原根 $\iff$ 任意 $\varphi(m)$ 的质因子 $p, a^{\frac{\varphi(m)}{p}} \not\equiv 1 \pmod m$

原根存在定理：m 有原根 $\iff m = 2, 4, p^k, 2p^k$，其中 p 为奇素数,$k \in \mathbb{N}^+$

原根个数：若 m 有原根，则其原根个数为 $\varphi(\varphi(m))$

最小原根：若 m 有原根，则其最小原根 $\leq m^{0.25}$

```
const int N=1e6+6;
int p[N],hp,lpf[N],phi[N];
bool vis[N],hasrt[N];
void sieve(int n){
    for(int i=2;i<=n;++i){
        if(!vis[i])p[++hp]=i,lpf[i]=i,phi[i]=i-1;
        for(int j=1;j<=hp&&i*p[j]<=n;++j){
            int x=i*p[j];
            vis[x]=1;
            lpf[x]=p[j];
            if(i%p[j]==0){
                phi[x]=phi[i]*p[j];
                break;
            }
            phi[x]=phi[i]*(p[j]-1);
        }
    }
    hasrt[2]=hasrt[4]=1;
    for(int i=2;i<=hp;++i){
        for(ll x=p[i];x<=n;x*=p[i]){
            hasrt[x]=1;
            if(x<<1<=n)hasrt[x<<1]=1;
        }
    }
}
vector<int> factorize(int n){// n很大可以考虑用Pollar-Rho
    vector<int>v;
    while(n>1){
        int t=lpf[n];
```

```
30          while(n%t==0)n/=t;
31          v.push_back(t);
32      }
33      return v;
34  }
35  ll qpow(ll a,ll b,const int mod){
36      ll s=1;
37      for(;b;b>>=1){
38          if(b&1)s=s*a%mod;
39          a=a*a%mod;
40      }
41      return s;
42  }
43  int min_rt(int n) {// 求n最小原根,不存在返回-1  O(pow(n,0.25) logn)
44      if(!hasrt[n]) return -1;
45      auto v=factorize(phi[n]);
46      for(int i=1;i<n;++i){
47          if(__gcd(i,n)!=1)continue;
48          bool f=1;
49          for(auto p:v)if(qpow(i,phi[n]/p,n)==1){f=0;break;}
50          if(f) return i;
51      }
52      return -1;
53  }
54  vector<int> all_rt(int n) { // 求所有原根 nlogn
55      int g=min_rt(n); vector<int>ans;
56      if (g==-1) return {};
57      for (int i=1,t=1;i<=phi[n];++i) {
58          t=1ll*t*g%n;
59          if(__gcd(i,phi[n])==1)ans.push_back(t);
60      }
61  //  sort(ans.begin(),ans.end());//ans无序,考虑是否排序
62      return ans;
63  }
64  vector<int> quick_all_rt(int n){// 快速求所有原根 O(n log(log(log(n))) )
65      int g=min_rt(n);
66      if (g==-1) return {};
67      vector<int>v=factorize(phi[n]),ans;
68      static bool vis[N], ok[N];// 不用清空
69      for(int p:v)for(int i=p;i<=phi[n];i+=p)vis[i]=1;
70      for (int i=1,t=1;i<=phi[n];++i){
```

```
        t=1ll*t*g%n;
        if(!vis[i])ok[t]=1;
        vis[i]=0;
    }
    for(int i=1;i<n;++i)if(ok[i])ans.push_back(i),ok[i]=0;
    return ans;
}
```

1.13 Lucas 定理

$C_n^m \equiv C_{n\%p}^{m\%p} C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} \pmod p$

```
ll C(ll n,ll m){//要预处理1~mod的阶乘
    return fac[n]*ifac[m]%mod*ifac[n-m]%mod;
}
ll lucas(ll n,ll m){//C(n,m)%mod
    if(!m)return 1;
    return C(n%mod,m%mod)*lucas(n/mod,m/mod)%mod;
}
```

1.14 杜教筛

在 $O(n^{3/4})$ 时间复杂度内求积性函数前缀和

$\sum_{i=1}^{n} f(i) = S(n)$

构造一个积性函数 g, 使得 $f*g$ 的前缀和很好求

根据 $\sum_{i=1}^{n}(f*g)(i) = \sum_{i=1}^{n}\sum_{d|i} f(d)g(\frac{i}{d}) = \sum_{d=1}^{n} g(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} f(i) = \sum_{d=1}^{n} g(d) S(\lfloor \frac{n}{d} \rfloor)$

得到：$g(1)S(n) = \sum_{i=1}^{n} g(i)S(\lfloor \frac{n}{i} \rfloor) - \sum_{i=2}^{n} g(i)S(\lfloor \frac{n}{i} \rfloor) = \sum_{i=1}^{n}(f*g)(i) - \sum_{i=2}^{n} g(i)S(\lfloor \frac{n}{i} \rfloor)$

```
ll getS(int n){// 求g(1)S(n)  通常g(1)=1
    if(n<=N)return sum[n];//N以内线性筛
    if(mp.count(n))return mp[n];//N以外unordered_map记忆化
    ll ans=f_g_sum(n);//f*g的前缀和要能O(1)算出
    for(int l=2,r;l<=n;l=r+1){//注意l从2开始
        int t=n/l; r=n/t;
        ans-=(sumg(r)-sumg(l-1))*getS(t);//sum(g)=g的前缀和要能O(1)算出
    }
    return mp[n]=ans; // 记忆化
}
```

求 $\sum_{i=1}^{n}\varphi(i)$，需要构造 $g(n)=1(n)=1,\varphi*1=id$

求 $\sum_{i=1}^{n}\mu(i)$，需要构造 $g(n)=1(n)=1,\mu*1=\varepsilon$

求 $\sum_{i=1}^{n}i^2\varphi(i)$，需要构造 $g(n)=n^2,(f*g)(n)=\sum_{d|n}d^2\varphi(d)(\frac{n}{d})^2=n^2\sum_{d|n}\varphi(d)=n^3$

```
const int N=2e6;//N > pow(maxn,2.0/3)
int p[N+5],hp;
bool vis[N+5];
ll phi[N+5],mu[N+5];
void seive(){
    phi[1]=1;mu[1]=1;
    for(int i=2;i<=N;++i){
        if(!vis[i]){
            p[++hp]=i;
            mu[i]=-1;
            phi[i]=i-1;
        }
        for(int j=1;j<=hp&&i*p[j]<=N;++j){
            vis[i*p[j]]=1;
            if(i%p[j]==0){
                phi[i*p[j]]=phi[i]*p[j];
                break;
            }
            phi[i*p[j]]=phi[i]*(p[j]-1);
            mu[i*p[j]]=-mu[i];
        }
    }
}
unordered_map<int,ll>mp_phi,mp_mu;
ll sumphi(ll n){
    if(n<=N)return phi[n];
    if(mp_phi.count(n))return mp_phi[n];
    ll ans=n*(n+1)/2;
    for(ll l=2,r;l<=n;l=r+1){
        ll t=n/l;r=n/t;
        ans-=(r-l+1)*sumphi(t);
    }
    return mp_phi[n]=ans;
}
ll summu(ll n){
    if(n<=N)return mu[n];
```

```
    if(mp_mu.count(n))return mp_mu[n];
    ll ans=1;
    for(ll l=2,r;l<=n;l=r+1){
        ll t=n/l;r=n/t;
        ans-=(r-l+1)*summu(t);
    }
    return mp_mu[n]=ans;
}
int main(){
    seive();
    for(int i=1;i<=N;++i){
        phi[i]+=phi[i-1];
        mu[i]+=mu[i-1];
    }
    int T;scanf("%d",&T);
    while(T--){
        ll n;scanf("%lld",&n);
        printf("%lld %lld\n",sumphi(n),summu(n));
    }
}
```

1.15 Min25 筛

用于在 $O(\frac{n^{\frac{3}{4}}}{logn})$ 时间复杂度内求 f 的前缀和，要求：

1、f 是积性函数，$f(1)=1$，$\gcd(x,y)=1$ 时，$f(xy)=f(x)f(y)$

2、$f(p^k)$ 可以被表示为项数较小的多项式，如 $f(p^k)=p^k(p^k-1)$，对于有常系数的多项式 $f(p)=k, f(p)=kp^2$，需要化为常系数为 1 的 $f(p)=1, f(p)=p^2$，求和后再乘以常数 k

3、$f(p^k)$ 可以快速求值，如 $\varphi(p^k)=(p-1)p^{k-1}$

4、也可以筛 $\sum\limits_{p\in\mathbb{P}} f(p)$，或者质数贡献方式特殊的函数，如 $f(n)=n$ 的次大质因子

这里令 $\mathbb{P}=[1,n]$ 的质数集合，$\mathbb{P}'=[1,\sqrt{n}]$ 的质数集合，$LPF_i=i$ 的最小质因子，$p_i=$ 第 i 小质数

$$\sum_{i=1}^{n} f(i)=\sum_{p\in\mathbb{P}}^{n} f(p)+\sum_{i\notin\mathbb{P}}^{n} f(i)=\sum_{p\in\mathbb{P}}^{n} f(p)+\sum_{p\in\mathbb{P}}^{\sqrt{n}}\sum_{c=1}^{p^c\le n} f(p^c)\sum_{i=1,LPF_i>p}^{\lfloor\frac{n}{p^c}\rfloor} f(i)$$

一、求质数部分

对某个单项式 $f(p)=p^2$，构造完全积性函数 F，F 和 f 在质数处取值相同，在合数处可能不同

设 $g(n,j)=\sum\limits_{i=2}^{n} F(i)[i\in P\vee LPF_i>p_j]$，则 $g(n,|\mathbb{P}'|)=\sum\limits_{p\in\mathbb{P}}^{n} f(p)$

$g(n,j)=g(n,j-1)-F(p_j)[g(\lfloor\frac{n}{p_j}\rfloor,j-1)-g(p_{j-1},j-1)]$，需要的值一定是 $g(\lfloor\frac{n}{x}\rfloor,j)$ 形式

$\lfloor\frac{n}{x}\rfloor$ 的取值一共 $2\sqrt{n}$ 种，给每个 $\lfloor\frac{n}{x}\rfloor$ 一个下标, 建立映射 $\lfloor\frac{n}{x}\rfloor\to index_{\lfloor\frac{n}{x}\rfloor}$

若 $\lfloor\frac{n}{x}\rfloor\le\sqrt{n}$，则 $idx1[\lfloor\frac{n}{x}\rfloor]=index_{\lfloor\frac{n}{x}\rfloor}$；反之，$idx2[\lfloor\frac{n}{\lfloor\frac{n}{x}\rfloor}\rfloor]=index_{\lfloor\frac{n}{x}\rfloor}$

二、求质数 + 合数

设 $S(n,j)=\sum\limits_{i=2}^{n} f(i)[LPF_i>p_j]$，则 $S(n,0)+f(1)$ 即为所求 $\sum\limits_{i=1}^{n} f(i)$

$S(n,j)$ 的质数部分 $g(n,|\mathbb{P}'|)-\sum_{i=1}^{j}f(p_i)$

$S(n,j)$ 的合数部分 $\sum_{p_k^c\le n\wedge k>j} f(p_k^c)[S(\lfloor\frac{n}{p_k^c}\rfloor,k)+[c\neq 1]]$，递归求解

总结：

线性筛 $\sqrt{n}$ 以内质数，以及质数前缀和 $sump[i]=\sum_{j=1}^{i}f(p_i)$

递推出 $g[index_{\lfloor\frac{n}{x}\rfloor}]=g(\lfloor\frac{n}{x}\rfloor,|\mathbb{P}'|)$，递归求解 $S(n,0)+f(1)$

```
const int N=1e6+5,mod=1e9+7;//N > 2sqrt(n)
ll f(ll pk){return pk*(pk-1)%mod;}//f(pk)=pk(pk-1)=F2(pk)-F1(pk)
ll F1(ll pk){return pk;}
ll F2(ll pk){return pk*pk%mod;}
ll sumF1(ll n){return n*(n+1)/2%mod-1;}//F1(2)+F1(3)+...+F1(n)
ll sumF2(ll n){return n*(n+1)/2*__int128(2*n+1)/3%mod-1;}
ll n,sqrtn,w[N];//w[k]=kth floor(n/x)
int idx1[N],idx2[N],tot;//idx1[n/x]=index(n/x),idx2[n/(n/x)]=index(n/x)
ll g1[N],g2[N];//g1[i]=sigma[isprime(p)&&p<=i] F1(p)
ll sump1[N],sump2[N];//sump1[i]=F1(p[1])+F1(p[2])+...+F1(p[i])
int p[N],hp;bool vis[N];
void sieve(int n){
    for(int i=2;i<=n;++i){
        if(!vis[i])p[++hp]=i;
        for(int j=1;j<=hp&&i*p[j]<=n;++j){
            vis[i*p[j]]=1;if(i%p[j]==0)break;
        }
    }
    for(int i=1;i<=hp;++i){
        sump1[i]=(sump1[i-1]+F1(p[i]))%mod;
        sump2[i]=(sump2[i-1]+F2(p[i]))%mod;
    }
}
ll S(ll x,int y){
    if(p[y]>=x)return 0;
    int k=x<=sqrtn?idx1[x]:idx2[n/x];
    ll s=(g2[k]-sump2[y])-(g1[k]-sump1[y]);
    for(int i=y+1;i<=hp&&1ll*p[i]*p[i]<=x;++i){
        for(ll pc=p[i],c=1;pc<=x;++c,pc*=p[i]){
            s+=f(pc%mod)*(S(x/pc,i)+(c!=1)); s%=mod;
        }
    }
    return s;
}
ll min25(ll n1){//f(1)+f(2)+...+f(n)
```

```
    n=n1; sqrtn=sqrt(n)+1; tot=0;
    for(ll l=1,r;l<=n;l=r+1){
        ll t=n/l,x=t%mod; r=n/t; w[++tot]=t;
        g1[tot]=sumF1(x); g2[tot]=sumF2(x);
        if(w[tot]<=sqrtn)idx1[w[tot]]=tot;
        else idx2[n/w[tot]]=tot;
    }
    for(int i=1;i<=hp;++i){
        for(int j=1;j<=tot&&1ll*p[i]*p[i]<=w[j];++j){
            ll x=w[j]/p[i]; int k=x<=sqrtn?idx1[x]:idx2[n/x];
            g1[j]-=F1(p[i])*(g1[k]-sump1[i-1]); g1[j]%=mod;
            g2[j]-=F2(p[i])*(g2[k]-sump2[i-1]); g2[j]%=mod;
        }
    }
    ll ans=S(n,0)+1;//S(n,0)=f(2)+f(3)+...+f(n)
    ans=(ans%mod+mod)%mod;
    return ans;
}
int main(){
    sieve(1000000);
    ll n;cin>>n;ll ans=min25(n);
}
```

1.16 二次剩余

$p=$ 奇素数，已知 n，求 $x^2 \equiv n \pmod p$，时间复杂度 $O(\log p)$

```
using ll = long long;
struct cp {
    static int mod;
    static ll wp;
    ll x, y; // x:real  y:imaginary
    cp(ll a = 0, ll b = 0) : x(a), y(b) {}
    cp operator *(cp c) {
        ll a = (x * c.x + y * c.y % mod * wp) % mod;
        if (a < 0) a += mod;
        ll b = (x * c.y + y * c.x) % mod;
        if (b < 0) b += mod;
        return cp(a, b);
    }
};
int cp::mod;
```

```
ll cp::wp;
ll cipolla(ll n, int p) { // x*x%p=n, 返回x,另一个解是-x
    cp::mod = p;
    ll &w = cp::wp;
    auto qpowCP = [&](cp a, ll b) {
        cp s(1, 0);
        for (; b; b >>= 1) {
            if (b & 1)s = s * a;
            a = a * a;
        }
        return s.x;
    };
    auto qpow = [&](ll a, ll b) {
        ll s = 1;
        for (; b; b >>= 1) {
            if (b & 1) s = s * a % p;
            a = a * a % p;
        }
        return s;
    };
    n %= p;
    if (qpow(n, p - 1 >> 1) == p - 1) return -1;
    ll a;
    while (1) {
        a = rand() % p;
        w = (a * a - n) % p;
        if (w < 0) w += p;
        if (qpow(w, p - 1 >> 1) == p - 1) break;
    }
    return qpowCP(cp(a, 1), p + 1 >> 1);
}
int main() {
    srand(time(0)); int n, mod;
    scanf("%d%d", &n, &mod);
    if (!n) { puts("0"); return 0; }
    ll ans1 = cipolla(n, mod), ans2 = mod - ans1;
    if (ans1 == -1) puts("no answer!");
    else {
        if (ans1 > ans2) swap(ans1, ans2);
        if (ans1 == ans2) printf("%lld\n", ans1);
        else printf("%lld %lld\n", ans1, ans2);
```

```
    }
}
```

1.17 Pollar-Rho

```
typedef long long ll;
typedef unsigned long long ull;
typedef long double db;
const int prime[]={2,3,5,7,11,13,17,19,23,29,31,37};
ll p[105];int hp;
mt19937_64 rnd(19260817);
ull add(ull x,ull y,ull mod){return x+y>=mod?x+y-mod:x+y;}
ull mul(ull x,ull y,ull mod){//x*y%mod,模数很大时可以溢出，不会出错
    ll s=x*y-ull(db(1)/mod*x*y)*mod;
    return s<0?s+=mod:(s>=mod?s-=mod:s);
}
ll qpow(ll a,ll b,const ll mod){
    ll s=1;
    while(b){
        if(b&1)s=mul(s,a,mod);
        a=mul(a,a,mod);
        b>>=1;
    }
    return s;
}
bool check(ll p,ll n){
    int k=0;ll d=n-1,x,s;
    while(d&1^1)d>>=1,++k;
    x=qpow(p,d,n);
    while(k){
        s=mul(x,x,n);
        if(s==1)return x==1||x==n-1;
        --k;x=s;
    }
    return false;
}
bool miller_rabin(ll n){
    for (int i=0;i<12;++i){
        if (n==prime[i]) return true;
        if (!(n%prime[i])) return false;
        if (!check(prime[i],n)) return false;
```

```
37      }
38      return true;
39  }
40  ll gcd(ll a,ll b){
41      if(!a)return b;if(!b)return a;
42      int t=__builtin_ctzll(a|b);
43      a>>=__builtin_ctzll(a);
44      do{
45          b>>=__builtin_ctzll(b);
46          if(a>b)swap(a,b);
47          b-=a;
48      }while (b);
49      return a<<t;
50  }
51  #define stp(x) (add(mul(x,x,n),r,n))
52  ll find_div(ll n){
53      if(n&1^1) return 2;
54      static const int S=128;
55      ll x=rnd()%n,r=rnd()%(n-1)+1;
56      for(int l=1;;l<<=1){
57          ll y=x,p=1;
58          for(int i=0;i<l;i++) x=stp(x);
59          for(int i=0;i<l;i+=S){
60              ll z=x;
61              for(int j=0;j<S&&j<l-i;++j)x=stp(x),p=mul(p,x+n-y,n);
62              p=gcd(p,n);
63              if(p==1)continue;
64              if(p<n)return p;
65              for (p=1,x=z;p==1;)x=stp(x),p=gcd(x+n-y,n);
66              return p;
67          }
68      }
69  }
70  void pollar_rho(ll n){
71      if(n==1) return;
72      if(miller_rabin(n)) {
73          p[++hp]=n; return;
74      }
75      ll d=n;
76      while(d==n)d=find_div(n);
77      pollar_rho(d);
```

```
    pollar_rho(n/d);
}
int main(){
    ll n;scanf("%lld",&n);
    hp=0;pollar_rho(n);//p[1~hp]为质因数
}
```

2 多项式

2.1 快速傅里叶变换 FFT

单位根：$\omega_n^n = 1, \omega_n = \cos\frac{2\pi}{n} + i\sin\frac{2\pi}{n}, \omega_n^k = \cos\frac{2k\pi}{n} + i\sin\frac{2k\pi}{n}$, $\omega_n^{k+\frac{n}{2}} = \omega_n^{-k}$, $\omega_{an}^{ak} = \omega_n^k$

DFT(离散傅里叶变换)：保证 $n = 2^s$

$A(x) = a_0x^0 + a_1x + a_2x^2 + a_3x^3 + ... + a_{n-1}x^{n-1}$

令 $A_0(x) = a_0 + a_2x + a_4x^2 + ... + a_{n-1}x^{\frac{n}{2}-1}$

令 $A_1(x) = a_1 + a_3x + a_5x^2 + ... + a_nx^{\frac{n}{2}-1}$

得 $A(x) = A_0(x^2) + xA_1(x^2)$

要求出 $A(\omega_n^0), A(\omega_n^1), A(\omega_n^2), ..., A(\omega_n^{n-1})$

$$\begin{bmatrix} A(\omega_n^0) \\ A(\omega_n^1) \\ A(\omega_n^2) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

根据 $A(\omega_n^k) = A_0(\omega_{\frac{n}{2}}^k) + \omega_n^k A_1(\omega_{\frac{n}{2}}^k)$，$A(\omega_n^{k+\frac{n}{2}}) = A_0(\omega_{\frac{n}{2}}^k) - \omega_n^k A_1(\omega_{\frac{n}{2}}^k)$ 用蝴蝶变换递推

$A_0(\omega_n^0)$	$A_0(\omega_n^1)$	$A_0(\omega_n^2)$	$A_0(\omega_n^3)$	$A_1(\omega_n^4)$	$A_1(\omega_n^5)$	$A_1(\omega_n^6)$	$A_1(\omega_n^7)$
$A_0(\omega_n^0)$	$A_0(\omega_n^2)$	$A_1(\omega_n^4)$	$A_1(\omega_n^6)$	$A_0(\omega_n^1)$	$A_0(\omega_n^3)$	$A_1(\omega_n^5)$	$A_1(\omega_n^7)$
$A_0(\omega_n^0)$	$A_1(\omega_n^4)$	$A_0(\omega_n^2)$	$A_1(\omega_n^6)$	$A_0(\omega_n^1)$	$A_1(\omega_n^5)$	$A_0(\omega_n^3)$	$A_1(\omega_n^7)$
a_0	a_4	a_2	a_6	a_1	a_5	a_3	a_7

IDFT(离散傅里叶逆变换)：保证 $n = 2^s$，根据以下等式

$$\begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \dots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{bmatrix} = \begin{bmatrix} n & 0 & 0 & \dots & 0 \\ 0 & n & 0 & \dots & 0 \\ 0 & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & n \end{bmatrix}$$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \dots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} A(\omega_n^0) \\ A(\omega_n^1) \\ A(\omega_n^2) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix}$$

IDFT 过程同 DFT，只需把 ω_n 的次数变为相反数，最后除以 n

以下为基础 FFT，不需要抄，需要抄的是下一段

```
const int L = 1 << 22;
using db = double;
using cp = complex<db>; // complex<double> a; a.real(); a.imag();
using Poly = vector<complex<db>>;
int r[L];
const db pi = acos(-1);
void InitR(int n) { // 蝴蝶变换
    int k = 0;
    while (1 << k < n) ++k;
    for (int i = 0; i < n; ++i) r[i] = r[i >> 1] >> 1 | (i & 1) << k - 1;
}
void FFT(Poly &a, int f) { // f=1是DFT,f=-1是IDFT
    int n = a.size();
    for (int i = 0; i < n; i++) {
        if (i < r[i]) swap(a[i], a[r[i]]);
    }
    for (int k = 1; k < n; k <<= 1) {
        cp wn(cos(pi / k), f * sin(pi / k));
        for (int i = 0; i < n; i += k << 1) {
            cp w(1, 0);
            for (int j = 0; j < k; ++j, w = w * wn) {
                cp t1 = a[i + j], t2 = w * a[i + j + k];
                a[i + j] = t1 + t2;
                a[i + j + k] = t1 - t2;
            }
        }
    }
    if (f == -1) for (auto &x: a) x /= n;
}
Poly Mul(const Poly &A, const Poly &B) {
    int n = 1, n1 = A.size(), n2 = B.size();
    while (n < n1 + n2 - 1) n <<= 1;
    Poly a(n), b(n);
```

```
    for (int i = 0; i < n1; ++i) a[i] = A[i];
    for (int i = 0; i < n2; ++i) b[i] = B[i];
    InitR(n), FFT(a, 1), FFT(b, 1);
    for (int i = 0; i < n; ++i) a[i] = a[i] * b[i];
    FFT(a, -1);
    a.resize(n1 + n2 - 1);
    return a;
}
```

2.1.1 FFT 模板

预处理单位根：$l < 2^k : w[2^k + l] = \omega_{2^k}^l$

DFT 用 DIT（Decimation in Time, 按时域抽取）实现，IDFT 用 DIF（Decimation in Frequency, 按频域抽取）实现，均不需要蝴蝶变换

仅包含 FFT 部分板子，其他多项式操作在 NTT 中

```
using db = double;
const double pi = acos(-1);
struct cp { // complex x + iy
    db x, y;
    cp(db a = 0, db b = 0) : x(a), y(b) {}
    cp operator +(cp b) { return cp(x + b.x, y + b.y); }
    cp operator -(cp b) { return cp(x - b.x, y - b.y); }
    cp operator *(cp b) { return cp(x * b.x - y * b.y, x * b.y + y * b.x); }
};
using Poly = vector<cp>;
const int L = 1 << 22;
cp w[L];
void InitPoly(int n = L) {
    w[1] = 1;
    for (int i = 2; i < n; i <<= 1) {
        auto w0 = w + i / 2, w1 = w + i;
        cp wn(cos(pi / i), sin(pi / i));
        for (int j = 0; j < i; j += 2) {
            w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
        }
    }
}
void DFT(Poly &a) {
    int n = a.size();
    for (int k = n >> 1; k; k >>= 1) {
        for (int i = 0; i < n; i += k << 1) {
```

```
            for (int j = 0; j < k; ++j) {
                cp x = a[i + j], y = a[i + j + k];
                a[i + j + k] = (x - y) * w[k + j], a[i + j] = x + y;
            }
        }
    }
}
void IDFT(Poly &a) {
    int n = a.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                cp x = a[i + j], y = a[i + j + k] * w[k + j];
                a[i + j + k] = x - y, a[i + j] = x + y;
            }
        }
    }
    for (int i = 0; i < n; ++i) a[i].x /= n, a[i].y /= n;
    reverse(a.begin() + 1, a.end());
}
Poly operator *(const Poly &A, const Poly &B) {
    int n1 = A.size(), n2 = B.size(), n = 1 << __lg(n1 + n2 - 1) + 1;
    assert(n1 && n2);
    Poly a(n), b(n);
    for (int i = 0; i < n1; ++i) a[i] = A[i];
    for (int i = 0; i < n2; ++i) b[i] = B[i];
    DFT(a), DFT(b);
    for (int i = 0; i < n; ++i) a[i] = a[i] * b[i];
    IDFT(a);
    a.resize(n1 + n2 - 1);
    return a;
}
```

2.1.2 三次变两次优化

令 $H(x) = F(x) + G(x)i$, $H^2(x) = [F^2(x) - G^2(x)] + 2F(x)G(x)i$

H 的虚部的一半即为所求，这样可以减少一次 DFT 操作，时间复杂度常数乘以 $\frac{2}{3}$，精度差一些

```
Poly Mul(const Poly &A, const Poly &B) {
    int n = 1, n1 = A.size(), n2 = B.size();
    while (n < n1 + n2 - 1) n <<= 1;
    Poly a(n);
```

```
    for (int i = 0; i < n1; ++i) a[i].x = A[i].x;
    for (int i = 0; i < n2; ++i) a[i].y = B[i].x;
    DFT(a);
    for (cp &x: a) x = x * x;
    IDFT(a);
    a.resize(n1 + n2 - 1);
    for (cp &x: a) x = x.y * 0.5;
    return a;
}
```

2.1.3 拆系数 FFT

为了使得整数多项式卷积 $F(x)G(x)$ 获得更高的精度，大约有 10^{14}，比三模 NTT 快很多

将每一项系数变为 $ka+b$，$k=2^{15}(\sqrt{mod})$

$F(x)=kA_1(x)_+B_1(x), G(x)=kA_2(x)+B_2(x)$

$F(x)G(x)=k^2A_1(x)A_2(x)+kA_1(x)B_2(x)+kB_1(x)A_2(x)+B_1(x)B_2(x)$

```
vector<ll> MTT(const vector<int> &A, const vector<int> &B, ll mod) {
    int n1 = A.size(), n2 = B.size(), n = 1 << __lg(n1 + n2 - 1) + 1;
    Poly a(n), b(n), c(n), d(n);
    for (int i = 0; i < n1; ++i) {
        a[i] = A[i] >> 15;
        b[i] = A[i] & 0x7fff;
    }
    for (int i = 0; i < n2; ++i) {
        c[i] = B[i] >> 15;
        d[i] = B[i] & 0x7fff;
    }
    DFT(a), DFT(b), DFT(c), DFT(d);
    for (int i = 0; i < n; ++i) {
        cp ta = a[i], tb = b[i], tc = c[i], td = d[i];
        a[i] = ta * tc;
        b[i] = ta * td + tc * tb;
        c[i] = tb * td;
    }
    IDFT(a), IDFT(b), IDFT(c);
    vector<ll> ret(n1 + n2 - 1);
    for (int i = 0; i < n1 + n2 - 1; ++i) {
        ll x = a[i].x + 0.5;
        ll y = b[i].x + 0.5;
        ll z = c[i].x + 0.5;
        ret[i] = ((x % mod << 30) + (y % mod << 15) + z) % mod;
```

```
    }
    return ret;
}
```

2.2 快速数论变换 NTT

设 g 是质数 p 的原根，$n = 2^i (i > 0)$，$g_n = g^{\frac{p-1}{n}}$ $(p = r \times 2^k + 1, n|p-1)$

$g_n^n \equiv 1 \pmod p$, $g_n^{\frac{n}{2}} \equiv g_n^{\frac{p-1}{2}} \pmod p$, $g_{an}^{ak} \equiv g_n^k \pmod p$, $(g_n^{k+\frac{n}{2}})^2 \equiv g_n^{2k} \pmod p$

g_n 性质和复数单位根 ω_n 类似，$mod = r \times 2^k + 1$ 可以实现 $2^k - 1$ 次多项式 NTT

原根表：g 是 $mod = r \times 2^k + 1$ 的原根

mod	r	k	g
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2281701377	17	27	3
4179340454199820289	29	51	3

以下部分为基础的 NTT，不需要抄，多项式板子在后面的几章

```
const i32 mod = 998244353, g = 3; // 模数，原根
const i32 gi = 332748118; // 原根逆元
int r[L];
void InitR(int n) { // 蝴蝶变换
    int k = 0;
    while (1 << k < n) ++k;
    for (int i = 0; i < n; ++i) r[i] = r[i >> 1] >> 1 | (i & 1) << k - 1;
}
void NTT(Poly &a, bool f) { // f=true是DFT,f=false是IDFT
    int n = a.size();
    for (int i = 0; i < n; i++) {
        if (i < r[i]) swap(a[i], a[r[i]]);
    }
    for (int k = 1; k < n; k <<= 1) {
        i32 wn = qpow(f ? g : gi, (mod - 1) / (k << 1));
        for (int i = 0; i < n; i += (k << 1)) {
            i32 w = 1;
            for (int j = 0; j < k; ++j, w = mul(w, wn)) {
                i32 x = a[i + j], y = mul(w, a[i + j + k]);
                a[i + j] = add(x, y);
```

```
                a[i + j + k] = add(x, mod - y);
            }
        }
    }
    i32 t = qpow(n, mod - 2);
    if (!f) for (i32 &x: a) x = mul(x, t);
}
Poly Mul(const Poly &A, const Poly &B) { // 多项式乘法
    int n = 1, n1 = A.size(), n2 = B.size();
    while (n < n1 + n2 - 1) n <<= 1;
    Poly a(n), b(n);
    for (int i = 0; i < n1; ++i) a[i] = A[i];
    for (int i = 0; i < n2; ++i) b[i] = B[i];
    InitR(n), NTT(a, 1), NTT(b, 1);
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], b[i]);
    NTT(a, 0);
    a.resize(n1 + n2 - 1);
    return a;
}
```

2.2.1 NTT 模板

抄模板注意事项

1、必须保证多项式中所有值永远处于 $[0, mod)$，或在 add 函数中考虑负数后范围变为 $(-mod, mod)$

2、main 函数开头调用 InitPoly();

3、若使用超过 int 范围的模数，可以将开头改为 i32=long long，i64=__int128

4、只抄需要的函数

5、常量 $L = 2^s$ 表示 DFT 和 IDFT 的最大范围是 $[0, L-1]$，必须大于多项式相乘后的最高次数，通常不超过 2^{21}，例如 $n-1$ 次多项式和 $m-1$ 次多项式相乘，须有 $L \geq n+m-1$

最小的满足 $L = 2^s \geq n$ 的 $s = \lfloor \log_2(n-1) \rfloor + 1$

i	2^i	i	2^i	i	2^i
17	131072	20	1048576	23	8388608
18	262144	21	2097152	24	16777216
19	524288	22	4194304	25	33554432

```
using i32 = int;
using i64 = long long;
using Poly = vector<i32>;
const int L = 1 << 21; // 大于多项式相乘后的最大长度
const i32 mod = 998244353, g = 3; // 模数，原根
i32 w[L]; // 单位根
//i32 inv[L + 1]; // 逆元，仅用于积分和多项式平移
```

```
//i32 fac[L + 1], ifac[L + 1]; // 阶乘，仅用于多项式平移
i32 add(i32 x, i32 y) { return (x += y) >= mod ? x - mod : x; }
i32 mul(i64 x, i32 y) { return x * y % mod; } // 注意第一个类型是i64
//using ull=unsigned long long; using ll=long long;
//ull mul(ull x,ull y){//x*y%mod,模数大于int范围可以用这个卡常
//  ll s=x*y-ull((long double)1/mod*x*y)*mod;
//  return s<0?s+mod:(s>=mod?s-mod:s);
//}
i32 qpow(i64 a, i64 b) {
    i64 s = 1;
    while (b) {
        if (b & 1) s = s * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return s;
}
void InitPoly(int n = L) { // main函数开始时调用一次
    w[1] = 1; // 预处理单位根
    for (int i = 2; i < n; i <<= 1) {
        auto w0 = w + i / 2, w1 = w + i;
        i32 wn = qpow(g, (mod - 1) / (i << 1));
        for (int j = 0; j < i; j += 2) {
            w1[j] = w0[j >> 1];
            w1[j + 1] = mul(w1[j], wn);
        }
    }
//  inv[1] = 1; // 预处理逆元，仅用于积分
//  for (int i = 2; i <= n; ++i) inv[i] = mul(mod - mod / i, inv[mod % i]);
//  fac[0] = ifac[0] = 1; // 预处理阶乘，仅用于多项式平移
//  for (int i = 1; i <= n; ++i) fac[i] = mul(fac[i - 1], i);
//  ifac[n] = qpow(fac[n], mod - 2);
//  for (int i = n - 1; i; --i) ifac[i] = mul(ifac[i + 1], i + 1);
}
void DFT(Poly &v) { // 离散傅里叶变换
    int n = v.size();
    for (int k = n >> 1; k; k >>= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                i32 x = v[i + j], y = v[i + j + k];
                v[i + j] = add(x, y);
```

```
                v[i + j + k] = mul(add(x, mod - y), w[k + j]);
            }
        }
    }
}
void IDFT(Poly &v) { // 离散傅里叶逆变换
    int n = v.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                i32 x = v[i + j], y = mul(v[i + j + k], w[k + j]);
                v[i + j] = add(x, y);
                v[i + j + k] = add(x, mod - y);
            }
        }
    }
    i32 t = qpow(n, mod - 2);
    for (i32 &x: v) x = mul(x, t);
    reverse(v.begin() + 1, v.end());
}
Poly operator *(const Poly &a, i32 k) { // 多项式乘以常数,只用于多项式快速幂
    Poly ret(a);
    for (i32 &x: ret) x = mul(x, k);
    return ret;
}
Poly operator +(const Poly &a, const Poly &b) { // 多项式加法, 不用于其他函数
    if (a.size() < b.size()) return b + a;
    Poly c(a);
    for (int i = 0; i < b.size(); ++i) c[i] = add(c[i], b[i]);
    return c;
}
Poly operator *(const Poly &A, const Poly &B) { // 多项式乘法
    int n1 = A.size(), n2 = B.size(), n = 1 << __lg(n1 + n2 - 1) + 1;
    Poly a(n), b(n);
    for (int i = 0; i < n1; ++i) a[i] = A[i];
    for (int i = 0; i < n2; ++i) b[i] = B[i];
    DFT(a), DFT(b);
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], b[i]);
    IDFT(a);
    a.resize(n1 + n2 - 1);
    return a;
```

```
}
Poly Der(const Poly &a) { // 导数
    Poly b(a.size() - 1);
    for (int i = 0; i < b.size(); ++i) b[i] = mul(i + 1, a[i + 1]);
    return b;
}
Poly Int(const Poly &a) { // 积分
    Poly b(a.size() + 1);
    for (int i = b.size() - 1; i; --i) b[i] = mul(a[i - 1], inv[i]);
    b[0] = 0; // 积分加的常数记作0
    return b;
}
```

2.2.2 多项式乘法逆

已知 $n-1$ 次多项式 $F(x)$，求 $G(x)$ 满足 $F(x)G(x) \equiv 1 \pmod{x^n}$
将 $F(x)$ 的补齐为 2 的幂次长度，最高次项为 $2^i - 1$
设 $G_k(x)$ 满足 $F(x)G_k(x) \equiv 1 \pmod{x^k}$
根据 $F(x)G_k(x) \equiv 1 \pmod{x^k}$，$F(x)G_{\frac{k}{2}}(x) \equiv 1 \pmod{x^{\frac{k}{2}}}$
得 $G_k(x) - G_{\frac{k}{2}}(x) \equiv 0 \pmod{x^{\frac{k}{2}}}$
平方，再乘以 F，得 $F(x)[G_k^2(x) - 2G_k(x)G_{\frac{k}{2}}(x) + G_{\frac{k}{2}}^2(x)] \equiv 0 \pmod{x^k}$
移向得递推式：$G_k(x) \equiv 2G_{\frac{k}{2}}(x) - F(x)G_{\frac{k}{2}}^2(x) \pmod{x^k}$
时间复杂度 $O(n \log n)$

```
Poly Inv(const Poly &a) { // 多项式乘法逆
    Poly b(1, qpow(a[0], mod - 2));
    int n = 1 << __lg(a.size() - 1) + 1;
    for (int k = 2; k <= n; k <<= 1) {
        int l = k << 1;
        Poly c(a.begin(), a.begin() + min(k, int(a.size())));
        b.resize(l), c.resize(l);
        DFT(b), DFT(c);
        for (int i = 0; i < l; ++i) {
            b[i] = mul(b[i], add(2, mod - mul(b[i], c[i])));
        }
        IDFT(b);
        b.resize(k);
    }
    b.resize(a.size());
    return b;
}
```

2.2.3 多项式带余除法

已知 $F^{(n)}(x)$ 和 $G^{(m)}(x)$，其中 $F^{(n)}(x)$ 表示 F 的最高次项为 x^n
求 $Q^{(n-m)}(x)$ 和 $R(x)$，使得 $F(x)=G(x)Q(x)+R(x)$
设 $F_R^{(n)}(x)=x^nF^{(n)}(\frac{1}{x})$
要满足 $F(x)=G(x)Q(x)+R(x)$
只要满足 $x^nF(\frac{1}{x})\equiv x^mG(\frac{1}{x})x^{n-m}Q(\frac{1}{x})+x^{n-m+1}x^{m-1}R(\frac{1}{x})$
只要满足 $F_R(x)\equiv G_R(x)Q_R(x)+x^{n-m+1}R_R(x)\pmod{x^{n-m+1}}$
得 $Q_R(x)\equiv F_R(x)G_R^{-1}(x)\pmod{x^{n-m+1}}$，进而得到 Q,R
时间复杂度为 $O(n\log n)$

```
pair<Poly, Poly> Divide(const Poly &a, const Poly &b) { // 多项式带余除法
    // a / b = c 余 d, return pair(c, d)
    int n = a.size(), m = b.size();
    Poly t1(a.rbegin(), a.rbegin() + n - m + 1);
    Poly t2(b.rbegin(), b.rend());
    t2.resize(n - m + 1);
    Poly c = Inv(t2) * t1;
    c.resize(n - m + 1);
    reverse(c.begin(), c.end());
    Poly d = c * b;
    d.resize(m - 1);
    for (int i = 0; i < d.size(); ++i) {
        d[i] = add(a[i], mod - d[i]);
    }
    return {c, d};
}
// 以下两个可以不抄
Poly operator /(const Poly &a, const Poly &b) {
    return Divide(a, b).first;
}
Poly operator %(const Poly &a, const Poly &b) {
    return Divide(a, b).second;
}
```

2.2.4 多项式 Ln

给定 $n-1$ 次多项式 $F(x)$，求 $G(x)\equiv\ln F(x)\pmod{x^n}$，保证 $F[0]=1$
$\frac{d}{dx}\ln F(x)=\frac{F'(x)}{F(x)}$
$\ln F(x)=\int F'(x)\times\frac{1}{F(x)}dx$，常数项为 $\ln F[0]=0$
时间复杂度为 $O(n\log n)$

```
Poly Ln(const Poly &a) { // 多项式ln，要求a[0]=1
```

```
    Poly res = Int(Der(a) * Inv(a));
    res.resize(a.size());
    return res;
}
```

2.2.5 多项式 Exp

给定 $n-1$ 次多项式 $F(x)$，求 $G(x) \equiv e^{F(x)} \pmod{x^n}$，保证 $F[0]=0$
设 $G(x)=e^{F(x)}, H(G(x))=\ln G(x)-F(x), H$ 看作是关于 G 的函数
需要令 $H(G) \equiv 0 \pmod{x^n}$，用牛顿迭代法
最初令 $G=e^{F[0]}=1$，不断迭代 $G=G_0-\frac{H(G_0)}{H'(G_0)}$ 至 $H(G) \equiv 0 \pmod{x^n}$
若 $H(G_0) \equiv 0 \pmod{x^{\frac{n}{2}}}$，则 $H(G) \equiv 0 \pmod{x^n}$，因此迭代 $\log n$ 次，时间复杂度为 $O(n \log n)$

```
Poly Exp(const Poly &a) { // 多项式exp, 要求a[0]=0
    Poly b(1, 1);
    int n = 1;
    while (n < a.size()) n <<= 1;
    for (int k = 2; k <= n; k <<= 1) {
        Poly c(b.begin(), b.end());
        c.resize(k);
        c = Ln(c);
        for (int i = 0; i < min(int(a.size()), k); ++i) {
            c[i] = add(a[i], mod - c[i]);
        }
        c[0] = add(c[0], 1);
        b = b * c;
        b.resize(k);
    }
    b.resize(a.size());
    return b;
}
```

2.2.6 多项式开根

已知 $n-1$ 次多项式 $F(x)$，求一个 $n-1$ 次多项式 $G(x)$ 满足 $G^2(x) \equiv F(x) \pmod{x^n}$
设 $G_k^2(x) \equiv F(x) \pmod{x^k}, G_{\frac{k}{2}}^2(x) \equiv F(x) \pmod{x^{\frac{k}{2}}}$
$G_{\frac{k}{2}}^2(x)-F(x) \equiv 0 \pmod{x^{\frac{k}{2}}}$
$[G_{\frac{k}{2}}^2(x)-F]^2(x) \equiv 0 \pmod{x^k}$
$[G_{\frac{k}{2}}^2(x)+F(x)]^2 \equiv 4G_{\frac{k}{2}}^2(x)F(x) \pmod{x^k}$
$\left[\frac{G_{\frac{k}{2}}^2(x)+F(x)}{2G_{\frac{k}{2}}(x)}\right]^2 \equiv F(x) \pmod{x^k}$
两边同时开根号，得递推式 $\frac{G_{\frac{k}{2}}^2(x)+F(x)}{2G_{\frac{k}{2}}(x)} \equiv G_k(x) \pmod{x^k}$

当 $F[0] \neq 0$ 时，$G[0] = \sqrt{F[0]} \pmod p$（二次剩余）
当 $F[0] = 0$ 时，变形为 $F = x^k H(x)(H[0] \neq 0)$，$k$ 为偶数时才能开根，$\sqrt{F(x)} \equiv x^{\frac{k}{2}} \sqrt{H(x)} \pmod{x^n}$
时间复杂度为 $O(n \log n)$

```
Poly Sqrt(const Poly &a) { // 多项式开根, 保证a[0]=1
    Poly b(1, 1); // 若a[0]!=1, 则b[0]=a[0]的二次剩余(注意选择较大or较小的一个)
    i32 inv2 = mod / 2 + 1;
    int n = 1;
    while (n < a.size()) n <<= 1;
    for (int k = 2; k <= n; k <<= 1) {
        Poly c(b.begin(), b.end()), d(a.begin(), a.begin() +
        min(int(a.size()), k));
        c.resize(k);
        c = Inv(c) * d;
        b.resize(k);
        for (int i = 0; i < k; ++i) b[i] = mul(add(b[i], c[i]), inv2);
    }
    b.resize(a.size());
    return b;
}
```

2.2.7 多项式快速幂

给定 $n-1$ 次多项式 $F(x)$，求 $G(x) \equiv F^k(x) \pmod{x^n}$
$\ln G(x) \equiv k \ln F(x) \pmod{x^n}$
$G(x) \equiv e^{k \ln F(x)} \pmod{x^n}$
由于 $F^p(x) \equiv F(x^p) \pmod p$，且 $n < p$，得 $F^p(x) \equiv F[0] \pmod p$
当 $F[0] = 1$ 时，k 可以对 p 取模，即 $F^k(x) \equiv F^{k \bmod p}(x) \pmod{x^n} \pmod p$
时间复杂度为 $O(n \log n)$

```
Poly Pow(const Poly &a, i32 k) { // 多项式快速幂, 保证a[0]=1, O(nlogn)
    return Exp(Ln(a) * k);
}
```

当 $F[0] \neq 1$ 时，找到最小的系数不为 0 的项，转化为 $F(x) = tx^l H(x)$
$F^k(x) \equiv t^k x^{kl} H^k(x) \equiv t^{k \bmod \varphi(p)} x^{kl} H^{k \bmod p}(x) \pmod{x^n} \pmod p$
需要传入 $k \bmod p, k \bmod \varphi(p)$ 以及 bool 变量 $k \geq n$，范围可以是 $(k \leq 10^{10^5})$

```
Poly Pow(const Poly &a, i32 k_mod_p, i32 k_mod_phip, bool k_geq_n) {
    int n = a.size(), idx = -1;
    for (int i = 0; i < n; ++i) {
        if (a[i]) {
            idx = i;
            break;
```

```
        }
    }
    if (idx == -1 || idx > 0 && k_geq_n || !k_geq_n && i64(idx) * k_mod_p >= n)
        return Poly(n, 0);
    i32 t = a[idx], ti = qpow(t, mod - 2), tk = qpow(t, k_mod_phip);
    Poly b(n, 0);
    for (int i = idx; i < n; ++i) b[i - idx] = mul(a[i], ti);
    Poly c = Pow(b, k_mod_p);
    int offset = idx * k_mod_p;
    for (int i = n - 1; i >= offset; --i) c[i] = mul(c[i - offset], tk);
    for (int i = 0; i < offset; ++i) c[i] = 0;
    return c;
}
```

2.2.8 减法卷积

已知 n 次多项式 $F(x)$ 和 $G(x)$，求 $H(x)=\sum\limits_{k=0}^{n}(\sum\limits_{i=k}^{n}F_{i-k}G_i)x^k$

令 $G^R(x)=x^nG(\frac{1}{x})$ (系数翻转，$G_i=G^R_{n-i}$)

$H_k=\sum\limits_{i=k}^{n}F_{i-k}G_i=\sum\limits_{i=0}^{n-k}F_iG_{i+k}=\sum\limits_{i=0}^{n-k}F_iG^R_{n-k-i}=\sum\limits_{i+j=n-k}F_iG^R_j$

先求出 $H^R(x)=F\times G^R$，再翻转得到 H

```
Poly DMul(const Poly &A, const Poly &B) { // 差卷积，要求A和B长度相同
    int n = 1, l = A.size();
    while (n < 2 * l - 1) n <<= 1;
    Poly a(n, 0), b(n, 0);
    for (int i = 0; i < l; ++i) a[i] = A[i], b[i] = B[i];
    reverse(b.begin(), b.begin() + l);
    DFT(a), DFT(b);
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], b[i]);
    IDFT(a);
    a.resize(l);
    reverse(a.begin(), a.end());
    return a;
}
```

2.2.9 多项式平移

已知 n 次多项式 $F(x)$，求 n 次多项式 $F(x+d)\pmod{x^{n+1}}$

$F(x+d)=\sum\limits_{i=0}^{n}f_i(x+d)^i=\sum\limits_{i=0}^{n}f_i\sum\limits_{k=0}^{i}C_i^kx^kd^{i-k}=\sum\limits_{k=0}^{n}\sum\limits_{i=k}^{n}\frac{i!f_id^{i-k}}{k!(i-k)!}x^k$

$G(x)=\sum\limits_{i=0}^{n}\frac{d^i}{i!}x^i,H(x)=\sum\limits_{i=0}^{n}i!x^i,F(x+d)=\sum\limits_{k=0}^{n}\frac{1}{k!}(\sum\limits_{i=k}^{n}G_{i-k}H_i)x^k$

求出 G 和 H 的减法卷积后，每一项乘以 $\frac{1}{k!}$

```
Poly Offset(const Poly &a, i32 d) { // 多项式平移
    int n = a.size();
    Poly b(n), c(n);
    i32 td = 1;
    for (int i = 0; i < n; ++i) {
        b[i] = mul(td, ifac[i]);
        td = mul(td, d);
    }
    for (int i = 0; i < n; ++i) c[i] = mul(a[i], fac[i]);
    b = DMul(b, c);
    for (int i = 0; i < n; ++i) b[i] = mul(b[i], ifac[i]);
    return b;
}
```

2.2.10 分治 FFT

已知 n 次多项式 $g(x)$，求 n 次多项式 $f(x)$ 满足 $f_i = \sum_{j=1}^{i} f_{i-j} g_j$，其中 $g_0 = 0$(对 f 无影响)，规定 $f_0 = 1$

法一、生成函数 + 多项式求逆

$F(x) = \sum_{i=0}^{\infty} f_i x^i, G(x) = \sum_{i=0}^{\infty} g_i x^i$

$F(x)G(x) = \sum_{i=0}^{\infty} (\sum_{j+k=i} f_j g_k) x^i = \sum_{i=0}^{\infty} (\sum_{j=1}^{i} f_{i-j} g_j) x^i$

$F(x)G(x) \equiv F(x) - f_0 \pmod{x^{n+1}}$

得 $F(x) \equiv \frac{f_0}{1-G(x)} \pmod{x^{n+1}}$

法二、分治 FFT/NTT

先求出 $f[l, mid]$，计算出左半区间对右半区间每个位置的贡献，再递归右半区间

```
void cdq_FFT(Poly &f, const Poly &g, int l, int r) {
    if (l == r) {
        if (l == 0) f[l] = 1;
        return;
    }
    int mid = l + r >> 1;
    cdq_FFT(f, g, l, mid);
    Poly a(f.begin() + l, f.begin() + mid + 1);
    Poly b(g.begin(), g.begin() + r - l + 1);
    a = a * b;
    for (int i = mid + 1; i <= r; ++i) f[i] = add(f[i], a[i - l]);
    cdq_FFT(f, g, mid + 1, r);
}
```

2.2.11 启发式合并卷积

求 m 个多项式的乘积 $\prod_{i=1}^{m} F_i(x)$，若 $\sum_{i=1}^{m} len(F_i) = n$，则时间复杂度为 $O(n \log^2 n)$

```
Poly merge(vector<Poly> &&v) { // 可以传入std::move(v)
    priority_queue<pair<int, Poly>, vector<pair<int, Poly>>, greater<>> q;
    for (auto &&p: v) { q.emplace(p.size(), std::move(p)); }
    while (q.size() > 1) {
        Poly x = q.top().second;
        q.pop();
        Poly y = x * q.top().second;
        q.pop();
        q.emplace(y.size(), std::move(y));
    }
    return q.top().second;
}
```

2.2.12 三模 NTT

精度在 10^{26} 左右，常数巨大

```
using ll = long long;
int Mod; // 最终要模的模数
const int mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, g = 3;
const ll mod12 = 1ll * mod1 * mod2;
ll qpow(ll x, ll n, int p) {
    ll s = 1;
    for (; n; n >>= 1, x = x * x % p)
    if (n & 1) s = s * x % p;
    return s;
}
const int inv1 = qpow(mod1, mod2 - 2, mod2);
const int inv2 = qpow(mod12 % mod3, mod3 - 2, mod3);
int add(int x, int y, int p) { return (x += y) >= p ? x - p : x; }
struct Int {
    int x, y, z;
    Int() { x = y = z = 0; }
    Int(int x, int y, int z) : x(x % mod1), y(y % mod2), z(z % mod3) {}
    Int(int v) : x(v % mod1), y(v % mod2), z(v % mod3) {}
    friend Int operator +(const Int &u, const Int &v) {
        return Int(add(u.x,v.x,mod1), add(u.y,v.y,mod2), add(u.z,v.z,mod3));
    }
    friend Int operator -(const Int &u, const Int &v) {
```

```
        return Int(add(u.x, mod1 - v.x, mod1),
        add(u.y, mod2 - v.y, mod2), add(u.z, mod3 - v.z, mod3));
    }
    friend Int operator *(const Int &u, const Int &v) {
        return Int(1ll*u.x*v.x%mod1,1ll*u.y*v.y%mod2, 1ll*u.z*v.z% mod3);
    }
    int get() const {
        ll t = 1ll * add(y, mod2 - x, mod2) * inv1 % mod2 * mod1 + x;
        return (1ll * add(z, mod3 - t % mod3, mod3) *
        inv2 % mod3 * (mod12 % Mod) % Mod + t) % Mod;
    }
};
using Poly = vector<Int>;
const int L = 1 << 21;
Int w[L];
int inv[L];
void InitPoly(int n = L) {
    inv[1] = 1;
    for (int i = 2; i < L; ++i) inv[i]=(1ll*Mod-Mod/i)*inv[Mod%i]%Mod;
    w[1] = 1;
    for (int i = 2; i < n; i <<= 1) {
        auto w0 = begin(w) + i / 2, w1 = begin(w) + i;
        Int wn = Int(qpow(g, (mod1 - 1) / (i << 1), mod1),
        qpow(g, (mod2 - 1) / (i << 1), mod2),
        qpow(g, (mod3 - 1) / (i << 1), mod3));
        for (int j = 0; j < i; j += 2)
        w1[j] = w0[j >> 1], w1[j + 1] = w1[j] * wn;
    }
}
Poly operator -(const Int &v, const Poly &a) {
    Poly res(a);
    for (int i = 0; i < a.size(); ++i) res[i] = Mod - res[i];
    res[0] = res[0] + v;
    return res;
}
/*
Poly operator -(const Poly &a, const Int &v) {
    Poly b(a);
    b[0] = b[0] + (Mod - v);
    return b;
}
```

```
Poly operator *(const Poly &a, const Int &v) {
    Poly b(a);
    for (int i = 0; i < a.size(); ++i) b[i] = b[i] * v;
    return b;
}*/
void DFT(Poly &a) {
    int n = a.size();
    for (int k = n >> 1; k; k >>= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                Int x = a[i + j], y = a[i + j + k];
                a[i + j + k] = (x - y) * w[k + j], a[i + j] = x + y;
            }
        }
    }
}
void IDFT(Poly &a) {
    int n = a.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                Int x = a[i + j], y = a[i + j + k] * w[k + j];
                a[i + j + k] = x - y, a[i + j] = x + y;
            }
        }
    }
    Int inv = Int(qpow(n, mod1 - 2, mod1),
    qpow(n, mod2 - 2, mod2),
    qpow(n, mod3 - 2, mod3));
    for (int i = 0; i < n; ++i) a[i] = a[i] * inv;
    reverse(a.begin() + 1, a.end());
}
void restore(Poly &a) {
    for (Int &x: a) x = x.get();
}
Poly operator *(const Poly &A, const Poly &B) {
    int n = 1, n1 = A.size(), n2 = B.size();
    while (n < n1 + n2 - 1) n <<= 1;
    Poly a(A), b(B);
    a.resize(n), b.resize(n);
    DFT(a), DFT(b);
```

```
105     for (int i = 0; i < n; ++i) a[i] = a[i] * b[i];
106     IDFT(a);
107     Poly ans(n1 + n2 - 1);
108     for (int i = 0; i < n1 + n2 - 1; ++i) ans[i] = a[i].get();
109     return ans;
110 }
111 Poly Der(const Poly &a) {
112     Poly b(a);
113     for (int i = 0; i < a.size() - 1; ++i) b[i] = (i + 1) * b[i + 1];
114     b.back() = 0;
115     restore(b);
116     return b;
117 }
118 Poly Integral(const Poly &a) {
119     Poly b(a.size() + 1);
120     for (int i = a.size() - 1; i >= 1; --i) b[i] = b[i - 1] * inv[i];
121     b[0] = 0;
122     restore(b);
123     return b;
124 }
125 Poly Inv(const Poly &a) {
126     Poly res(1, Int(qpow(a[0].get(), Mod - 2, Mod)));
127     int n = 1;
128     while (n < a.size()) n <<= 1;
129     for (int k = 2; k <= n; k <<= 1) {
130         Poly t(a);
131         t.resize(k);
132         t = t * res;
133         t.resize(k);
134         res = res * (2 - t);
135         res.resize(k);
136     }
137     res.resize(a.size());
138     restore(res);
139     return res;
140 }
141 Poly Ln(const Poly &a) {
142     Poly b = Integral(Der(a) * Inv(a));
143     b.resize(a.size());
144     restore(b);
145     return b;
```

```
}
Poly Exp(const Poly &a) {
    Poly b(1, 1);
    int n = 1;
    while (n < a.size()) n <<= 1;
    for (int k = 2; k <= n; k <<= 1) {
        Poly c(b.begin(), b.end());
        c.resize(k); c = Ln(c);
        for (int i = 0; i < min(int(a.size()), k); ++i) {
            c[i] = a[i] + (Mod - c[i]);
        }
        c[0] = c[0] + 1;
        restore(c);
        b = b * c;
        b.resize(k);
    }
    b.resize(a.size());
    restore(b);
    return b;
}
```

2.3 上升幂、下降幂

阶乘幂

上升幂：$x^{\overline{n}} = \prod_{i=0}^{n-1}(x+i) = \binom{x+n-1}{n} n!$

下降幂：$x^{\underline{n}} = \prod_{i=0}^{n-1}(x-i) = \binom{x}{n} n!$

上升幂和下降幂的转换：$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}}, x^{\overline{n}} = (-1)^n(-x)^{\underline{n}}$

2.4 卷积字符串匹配

1、求 $t_{0\sim m-1}$ 在 $s_{0\sim n-1}(m \le n)$ 中出现的所有下标

令 $T_i = idx(t_i), S_i = idx(s_i)$，构造一个匹配函数，使得匹配位置取值为 0，匹配不上的位置取值不为 0

$m-1 \le k < n : F(k) = \sum_{i=0}^{m-1}(S_{k-i} - T_{m-1-i})^2 = \sum_{i=0}^{m-1} S_{k-i}^2 + \sum_{i=0}^{m-1} T_i^2 - 2\sum_{i+j=k} S_i T'_j$

其中 T' 表示 T 翻转后的数组，$T'_j = T_{m-1-j}$。可以发现第三项为一个卷积

2、对于 t 和 s 中有通配符的情况，我们可以令通配符的 $idx = 0$，其他字符 $idx = 1 \sim 26$

$F(k) = \sum_{i=0}^{m-1}(S_{k-i} - T_{m-1-i})^2 S_{k-i} T_{m-1-i} = \sum_{i+j=k}(S_i^3 T'_j - 2S_i^2 T'^2_j + S_i T'^3_j)$

6 次 DFT，相乘相加，再 IDFT 得到函数 F，$F(k) = 0$ 的 k 为匹配位置 $s[k-m+1:k]$

```
int idx(char c) {
    if (c == '*') return 0;
    return c - 'a' + 1;
```

```
}
vector<int> match(string t, string s) { // t在s中出现位置的起始下标
    int m = t.length(), n = s.length(), l = 1 << __lg(n + m - 1) + 1;
    reverse(t.begin(), t.end());
    vector<Poly> v(6, Poly(l, 0)); // v[0~2]=s,s^2,s^3 v[3~5]=t,t^2,t^3
    for (int i = 0; i < m; ++i) {
        for (int j = 0, x = idx(t[i]); j < 3; ++j, x *= idx(t[i])) {
            v[j][i] = x;
        }
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0, x = idx(s[i]); j < 3; ++j, x *= idx(s[i])) {
            v[j + 3][i] = x;
        }
    }
    for (auto &p: v) DFT(p);
    for (int i = 0; i < l; ++i) {
        i64 x = (1ll * v[0][i] * v[5][i] - 2ll * v[1][i] * v[4][i] + 1ll *
        v[2][i] * v[3][i]) % mod;
        if (x < 0) x += mod;
        v[0][i] = x;
    }
    IDFT(v[0]);
    vector<int> ret;
    for (int i = m - 1; i < n; ++i) {
        if (v[0][i] == 0) {
            ret.push_back(i - m + 1);
        }
    }
    return ret;
}
```

2.5 快速沃尔什变换 FWT

$c_i = \sum_{i=j\&k} a_j b_k$，保证 a, b, c 长度均为 2^s

```
using ll = long long;
const int mod = 998244353;
using Poly = vector<int>;
int add(int x, int y) { return (x += y) >= mod ? x - mod : x; }
int mul(ll x, int y) { return x * y % mod; }
```

```
void fwt_and(Poly &a, int f) {
    int n = a.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                a[i + j] = add(a[i + j], mul(a[i + j + k], f));
            }
        }
    }
}
Poly conv_and(const Poly &A, const Poly &B) {
    Poly a(A), b(B);
    fwt_and(a, 1), fwt_and(b, 1);
    for (int i = 0; i < A.size(); ++i) a[i] = mul(a[i], b[i]);
    fwt_and(a, mod - 1);
    return a;
}
```

$c_i = \sum_{i=j|k} a_j b_k$，保证 a, b, c 长度均为 2^s

```
void fwt_or(Poly &a, int f) {
    int n = a.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                a[i + j + k] = add(a[i + j + k], mul(a[i + j], f));
            }
        }
    }
}
Poly conv_or(const Poly &A, const Poly &B) {
    Poly a(A), b(B);
    fwt_or(a, 1), fwt_or(b, 1);
    for (int i = 0; i < A.size(); ++i) a[i] = mul(a[i], b[i]);
    fwt_or(a, mod - 1);
    return a;
}
```

$c_i = \sum_{i=j\oplus k} a_j b_k$，保证 a, b, c 长度均为 2^s

```
void fwt_xor(Poly &a, int f) {
    int n = a.size();
```

```
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; ++j) {
                int x = a[i + j], y = a[i + j + k];
                a[i + j] = mul(add(x, y), f);
                a[i + j + k] = mul(add(x, mod - y), f);
            }
        }
    }
}
Poly conv_xor(const Poly &A, const Poly &B) {
    Poly a(A), b(B);
    fwt_xor(a, 1), fwt_xor(b, 1);
    for (int i = 0; i < A.size(); ++i) a[i] = mul(a[i], b[i]);
    fwt_xor(a, mod / 2 + 1);
    return a;
}
```

3 数学

3.1 高斯消元

实数

```
int n,m,r;double a[N][M];
void gauss(double a[N][M],int n,int m){
    r=0;//矩阵的秩
    int lim=min(n,m);
    for(int i=1;i<=lim;++i){//当前到第几列
        bool f=0;
        for(int j=r+1;j<=n;++j){
            if(abs(a[j][i])>1e-8){
                swap(a[r+1],a[j]);
                f=1;break;
            }
        }
        if(!f)continue;
        ++r;double t=a[r][i];
        for(int j=i;j<=m;++j)a[r][j]/=t;
        for(int j=1;j<=n;++j){
            if(j==r)continue;
            t=a[j][i];
```

```
            for(int k=i;k<=m;++k)a[j][k]-=t*a[r][k];
        }
    }
}
```

整数取模

```
ll a[N][M];const int mod=1e9+7;//先写好快速幂qpow
void gauss(ll a[N][M],int n,int m){//注意化简后有负数未取模为正
    r=0;//矩阵的秩
    int lim=min(n,m);
    for(int i=1;i<=lim;++i){
        bool f=0;
        for(int j=r+1;j<=n;++j){
            if(a[j][i]!=0){
                swap(a[r+1],a[j]);
                f=1;break;
            }
        }
        if(!f)continue;
        ++r;ll t=qpow(a[r][i],mod-2);
        for(int j=i;j<=m;++j)a[r][j]=a[r][j]*t%mod;
        for(int j=1;j<=n;++j){
            if(j==r)continue;
            t=a[j][i];
            for(int k=i;k<=m;++k)a[j][k]=(a[j][k]-t*a[r][k])%mod;
        }
    }
}
```

异或 bitset

```
int n,m,r,a[N][M];
bitset<M>g[N];//转为int可以g[i][j]?1:0或int(g[i][j])
void gauss(bitset<M>a[N],int n,int m){
    r=0;int lim=min(n,m);
    for(int i=1;i<=lim;++i){
        bool f=0;
        for(int j=r;j<n;++j){
            if(a[j][i]){
                swap(a[r+1],a[j]);f=1;break;
            }
        }
        if(!f)continue;
```

```
        ++r;
        for(int j=1;j<=n;++j){
            if(j==r)continue;
            if(a[j][i])a[j]^=a[r];
        }
    }
}
```

行列式

```
ll det(ll a[N][N],int n){
    ll s=1;
    for(int i=1;i<=n;++i){
        for(int j=i;j<=n;++j){
            if(a[j][i]){
                if(i!=j){
                    swap(a[j],a[i]);s=-s;
                }
                break;
            }
        }
        if(!a[i][i])return 0;
        s=s*a[i][i]%mod;
        ll inv=qpow(a[i][i],mod-2);
        for(int j=i;j<=n;++j)a[i][j]=a[i][j]*inv%mod;
        for(int j=i+1;j<=n;++j){
            ll t=a[j][i];
            for(int k=i;k<=n;++k)a[j][k]-=t*a[i][k],a[j][k]%=mod;
        }
    }
    return s;
}
```

辗转相减行列式 (模数不是质数时)

```
ll det(ll a[N][N],int n){
    ll ans=1;
    for(int i=1;i<=n;++i){
        for(int j=i+1;j<=n;++j){
            while(a[j][i]){
                ll t=a[i][i]/a[j][i];
                for(int k=i;k<=n;++k){
                    a[i][k]-=t*a[j][k];
                    a[i][k]%=mod;
```

```
                }
                swap(a[i],a[j]);ans=-ans;
            }
        }
        ans=ans*a[i][i]%mod;
    }
    return ans;
}
```

3.2 线性基

```
for(int i=1;i<=n;++i){//n个数的线性基
    ll x;scanf("%lld",&x);
    for(int j=62;j>=0;--j){
        if(x>>j&1){
            if(g[j])x^=g[j];
            else{
                g[j]=x;
                break;
            }
        }
    }
}
ll ans=0;
for(int i=62;i>=0;--i){//选几个数的最大异或
    if(ans>>i&1)continue;
    ans^=g[i];
}
```

3.3 矩阵快速幂

$$\begin{cases} f_1=1, f_2=1 \\ f_n=f_{n-1}+f_{n-2} \end{cases} \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{n-1} \\ f_{n-2} \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix} \Rightarrow \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} f_2 \\ f_1 \end{bmatrix}$$

$$\begin{cases} a_1=7, a_2=12 \\ a_n=2a_{n-1}-3a_{n-2}+n^2, n\geq 2 \end{cases} \Rightarrow \begin{bmatrix} 2 & -3 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ (n-1)^2 \\ n-1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_n \\ a_{n-1} \\ n^2 \\ n \\ 1 \end{bmatrix}$$

```
const int mod=998244353;
const int n1=4;
struct matrix{
    ll a[n1][n1];//此处过大可能导致本地RE
    void clear(){
        for(int i=0;i<n1;++i)memset(a[i],0,n1<<3);
    }
    void I(){
        clear();for(int i=0;i<n1;++i)a[i][i]=1;
    }
    ll* operator [] (int i) {
        return a[i];
    }
    matrix(const initializer_list<ll>&l={}){
        int now=0;
        for(auto x:l){
            a[now/n1][now%n1]=x;
            ++now;
        }
    }
    matrix operator + (const matrix &b){
        static matrix c;
        for(int i=0;i<n1;++i){
            for(int j=0;j<n1;++j){
                c[i][j]=(a[i][j]+b.a[i][j])%mod;
            }
        }
        return c;
    }
    matrix operator * (const matrix &b) {
        static matrix c;c.clear();
        for(int i=0;i<n1;++i){
            for(int j=0;j<n1;++j){
                for(int k=0;k<n1;++k){
                    c[i][j]+=a[i][k]*b.a[k][j];
                    if(k&3)c[i][j]%=mod;
                }
                c[i][j]%=mod;
            }
        }
        return c;
```

```
    }
};
matrix qpow(matrix a,ll b){
    static matrix s;s.I();
    for(;b;b>>=1){
        if(b&1)s=s*a;
        a=a*a;
    }
    return s;
}
matrix powsum(matrix a,ll b){//幂和：A^0+A+A^2+A^3+...+A^b
    matrix I;I.I();
    if(!b)return I;
    if(b&1)return I+a*powsum(a,b-1);
    return I+(I+qpow(a,b/2))*a*powsum(a,b/2-1);
}
int main(){
    matrix A({//需要定义列向量，可以使用矩阵的第一列
        1,1,1,2,
        1,2,1,2,
        3,4,0,3,
        2,4,1,4
    });
}
```

3.4 康托展开

排列 $P=\{p_1,p_2,...,p_n\}$ 是第 $f(P)$ 小的排列 (12345 是第 0 小)

$f(P)=\sum_{i=1}^{n}(n-i)!\times r(i)$　　$r(i)=p_i$ 右边比 p_i 小的个数

逆康托展开：已知排列的排名 $rank$(从 0 开始), 求出该排列

```
for(int i=1;i<=n;++i){
    int t=rank/fac[n-i];
    //p[i]右边比p[i]小的有t个,p[i]应该是剩余数字第t小
    rank%=fac[n-i];int cnt=0;
    for(int j=1;j<=n;++j){
        if(!vis[j])++cnt;
        if(cnt==t+1){
            vis[j]=1;p[i]=j;break;
        }
    }
```

```
}
```

3.5 曼哈顿距离

$|x_1-x_2|+|y_1-y_2| = max(|(x_1+y_1)-(x_2+y_2)|, |(x_1-y_1)-(x_2-y_2)|)$

3.6 高次方求和

用拉格朗日插值法求和，时间复杂度为 $O(k)$

```
ll fac[N],pl[N],pr[N];
ll calc(ll n,int k){//1^k+2^k+3^k+..+n^k  O(k)
    ll y=0,ans=0;
    pl[0]=pr[k+3]=fac[0]=1;
    for(int i=1;i<=k+2;++i){
        fac[i]=fac[i-1]*i%mod;
        pl[i]=(n-i)%mod*pl[i-1]%mod;
    }
    for(int i=k+2;i;--i)pr[i]=(n-i)%mod*pr[i+1]%mod;
    for(int i=1;i<=k+2;++i){
        y+=qpow(i,k); y%=mod;
        ll a=pl[i-1]*pr[i+1]%mod;
        ll b=(k-i&1?-1:1)*fac[i-1]*fac[k+2-i]%mod;
        ans+=y*a%mod*qpow(b,mod-2); ans%=mod;
    }
    return ans;
}
```

高次方求和公式

1 次方求和公式: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$

2 次方求和公式: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$

3 次方求和公式: $\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$

4 次方求和公式: $\sum_{i=1}^{n} i^4 = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$

5 次方求和公式: $\sum_{i=1}^{n} i^5 = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$

6 次方求和公式: $\sum_{i=1}^{n} i^6 = \frac{1}{7}n^7 + \frac{1}{2}n^6 + \frac{1}{2}n^5 - \frac{1}{6}n^3 + \frac{1}{42}n$

7 次方求和公式: $\sum_{i=1}^{n} i^7 = \frac{1}{8}n^8 + \frac{1}{2}n^7 + \frac{7}{12}n^6 - \frac{7}{24}n^4 + \frac{1}{12}n^2$

8 次方求和公式: $\sum_{i=1}^{n} i^8 = \frac{1}{9}n^9 + \frac{1}{2}n^8 + \frac{2}{3}n^7 - \frac{7}{15}n^5 + \frac{2}{9}n^3 - \frac{1}{30}n$

9 次方求和公式: $\sum_{i=1}^{n} i^9 = \frac{1}{10}n^{10} + \frac{1}{2}n^9 + \frac{3}{4}n^8 - \frac{7}{10}n^6 + \frac{1}{2}n^4 - \frac{3}{20}n^2$

10 次方求和公式: $\sum_{i=1}^{n} i^{10} = \frac{1}{11}n^{11} + \frac{1}{2}n^{10} + \frac{5}{6}n^9 - n^7 + n^5 - \frac{1}{2}n^3 + \frac{5}{66}n$

11 次方求和公式: $\sum_{i=1}^{n} i^{11} = \frac{1}{12}n^{12} + \frac{1}{2}n^{11} + \frac{11}{12}n^{10} - \frac{11}{8}n^8 + \frac{11}{6}n^6 - \frac{11}{8}n^4 + \frac{5}{12}n^2$

12 次方求和公式: $\sum_{i=1}^{n} i^{12} = \frac{1}{13}n^{13} + \frac{1}{2}n^{12} + n^{11} - \frac{11}{6}n^9 + \frac{22}{7}n^7 - \frac{33}{10}n^5 + \frac{5}{3}n^3 - \frac{691}{2730}n$

13 次方求和公式: $\sum_{i=1}^{n} i^{13} = \frac{1}{14}n^{14} + \frac{1}{2}n^{13} + \frac{13}{12}n^{12} - \frac{143}{60}n^{10} + \frac{143}{28}n^8 - \frac{143}{20}n^6 + \frac{65}{12}n^4 - \frac{691}{420}n^2$

14 次方求和公式: $\sum_{i=1}^{n} i^{14} = \frac{1}{15}n^{15} + \frac{1}{2}n^{14} + \frac{7}{6}n^{13} - \frac{91}{30}n^{11} + \frac{143}{18}n^9 - \frac{143}{10}n^7 + \frac{91}{6}n^5 - \frac{691}{90}n^3 + \frac{7}{6}n$

15 次方求和公式: $\sum_{i=1}^{n} i^{15} = \frac{1}{16}n^{16} + \frac{1}{2}n^{15} + \frac{5}{4}n^{14} - \frac{91}{24}n^{12} + \frac{143}{12}n^{10} - \frac{429}{16}n^8 + \frac{455}{12}n^6 - \frac{691}{24}n^4 + \frac{35}{4}n^2$

16 次方求和公式: $\sum_{i=1}^{n} i^{16} = \frac{1}{17}n^{17} + \frac{1}{2}n^{16} + \frac{4}{3}n^{15} - \frac{14}{3}n^{13} + \frac{52}{3}n^{11} - \frac{143}{3}n^9 + \frac{260}{3}n^7 - \frac{1382}{15}n^5 + \frac{140}{3}n^3 - \frac{3617}{510}n$

17 次方求和公式: $\sum_{i=1}^{n} i^{17} = \frac{1}{18}n^{18} + \frac{1}{2}n^{17} + \frac{17}{12}n^{16} - \frac{17}{3}n^{14} + \frac{221}{9}n^{12} - \frac{2431}{30}n^{10} + \frac{1105}{6}n^8 - \frac{11747}{45}n^6 + \frac{595}{3}n^4 - \frac{3617}{60}n^2$

18 次方求和公式: $\sum_{i=1}^{n} i^{18} = \frac{1}{19}n^{19} + \frac{1}{2}n^{18} + \frac{3}{2}n^{17} - \frac{34}{5}n^{15} + 34n^{13} - \frac{663}{5}n^{11} + \frac{1105}{3}n^9 - \frac{23494}{35}n^7 + 714n^5 - \frac{3617}{10}n^3 + \frac{43867}{798}n$

19 次方求和公式: $\sum_{i=1}^{n} i^{19} = \frac{1}{20}n^{20} + \frac{1}{2}n^{19} + \frac{19}{12}n^{18} - \frac{323}{40}n^{16} + \frac{323}{7}n^{14} - \frac{4199}{20}n^{12} + \frac{4199}{6}n^{10} - \frac{223193}{140}n^8 + 2261n^6 - \frac{68723}{40}n^4 + \frac{43867}{84}n^2$

20 次方求和公式: $\sum_{i=1}^{n} i^{20} = \frac{1}{21}n^{21} + \frac{1}{2}n^{20} + \frac{5}{3}n^{19} - \frac{19}{2}n^{17} + \frac{1292}{21}n^{15} - 323n^{13} + \frac{41990}{33}n^{11} - \frac{223193}{63}n^9 + 6460n^7 - \frac{68723}{10}n^5 + \frac{219335}{63}n^3 - \frac{174611}{330}n$

21 次方求和公式: $\sum_{i=1}^{n} i^{21} = \frac{1}{22}n^{22} + \frac{1}{2}n^{21} + \frac{7}{4}n^{20} - \frac{133}{12}n^{18} + \frac{323}{4}n^{16} - \frac{969}{2}n^{14} + \frac{146965}{66}n^{12} - \frac{223193}{30}n^{10} + \frac{33915}{2}n^8 - \frac{481061}{20}n^6 + \frac{219335}{12}n^4 - \frac{1222277}{220}n^2$

22 次方求和公式: $\sum_{i=1}^{n} i^{22} = \frac{1}{23}n^{23} + \frac{1}{2}n^{22} + \frac{11}{6}n^{21} - \frac{77}{6}n^{19} + \frac{209}{2}n^{17} - \frac{3553}{5}n^{15} + \frac{11305}{3}n^{13} - \frac{223193}{15}n^{11} + \frac{124355}{3}n^9 - \frac{755953}{10}n^7 + \frac{482537}{6}n^5 - \frac{1222277}{30}n^3 + \frac{854513}{138}n$

23 次方求和公式: $\sum_{i=1}^{n} i^{23} = \frac{1}{24}n^{24} + \frac{1}{2}n^{23} + \frac{23}{12}n^{22} - \frac{1771}{120}n^{20} + \frac{4807}{36}n^{18} - \frac{81719}{80}n^{16} + \frac{37145}{6}n^{14} - \frac{5133439}{180}n^{12} + \frac{572033}{6}n^{10} - \frac{17386919}{80}n^8 + \frac{11098351}{36}n^6 - \frac{28112371}{120}n^4 + \frac{854513}{12}n^2$

24 次方求和公式: $\sum_{i=1}^{n} i^{24} = \frac{1}{25}n^{25} + \frac{1}{2}n^{24} + 2n^{23} - \frac{253}{15}n^{21} + \frac{506}{3}n^{19} - \frac{14421}{10}n^{17} + \frac{29716}{3}n^{15} - \frac{10266878}{195}n^{13} + 208012n^{11} - \frac{17386919}{30}n^9 + \frac{22196702}{21}n^7 - \frac{28112371}{25}n^5 + \frac{1709026}{3}n^3 - \frac{236364091}{2730}n$

25 次方求和公式: $\sum_{i=1}^{n} i^{25} = \frac{1}{26}n^{26} + \frac{1}{2}n^{25} + \frac{25}{12}n^{24} - \frac{115}{6}n^{22} + \frac{1265}{6}n^{20} - \frac{24035}{12}n^{18} + \frac{185725}{12}n^{16} - \frac{25667195}{273}n^{14} + \frac{1300075}{3}n^{12} - \frac{17386919}{12}n^{10} + \frac{277458775}{84}n^8 - \frac{28112371}{6}n^6 + \frac{21362825}{6}n^4 - \frac{1181820455}{1092}n^2$

26 次方求和公式: $\sum_{i=1}^{n} i^{26} = \frac{1}{27}n^{27} + \frac{1}{2}n^{26} + \frac{13}{6}n^{25} - \frac{65}{3}n^{23} + \frac{16445}{63}n^{21} - \frac{16445}{6}n^{19} + \frac{142025}{6}n^{17} - \frac{10266878}{63}n^{15} + \frac{2600150}{3}n^{13} - \frac{20548177}{6}n^{11} + \frac{3606964075}{378}n^9 - \frac{52208689}{3}n^7 + \frac{55543345}{3}n^5 - \frac{1181820455}{126}n^3 + \frac{8553103}{6}n$

27 次方求和公式: $\sum_{i=1}^{n} i^{27} = \frac{1}{28}n^{28} + \frac{1}{2}n^{27} + \frac{9}{4}n^{26} - \frac{195}{8}n^{24} + \frac{4485}{14}n^{22} - \frac{29601}{8}n^{20} + \frac{142025}{4}n^{18} - \frac{15400317}{56}n^{16} + 1671525n^{14} - \frac{61644531}{8}n^{12} + \frac{721392815}{28}n^{10} - \frac{469878201}{8}n^8 + \frac{166630035}{2}n^6 - \frac{3545461365}{56}n^4 + \frac{76977927}{4}n^2$

28 次方求和公式: $\sum_{i=1}^{n} i^{28} = \frac{1}{29}n^{29} + \frac{1}{2}n^{28} + \frac{7}{3}n^{27} - \frac{273}{10}n^{25} + 390n^{23} - \frac{9867}{2}n^{21} + 52325n^{19} - \frac{905901}{2}n^{17} + 3120180n^{15} - \frac{33193209}{2}n^{13} + 65581165n^{11} - \frac{365460823}{2}n^9 + 333260070n^7 - \frac{709092273}{2}n^5 + 179615163n^3 - \frac{23749461029}{870}n$

29 次方求和公式: $\sum_{i=1}^{n} i^{29} = \frac{1}{30}n^{30} + \frac{1}{2}n^{29} + \frac{29}{12}n^{28} - \frac{609}{20}n^{26} + \frac{1885}{4}n^{24} - \frac{26013}{4}n^{22} + \frac{303485}{4}n^{20} - \frac{8757043}{12}n^{18} + \frac{22621305}{4}n^{16} - \frac{137514723}{4}n^{14} + \frac{1901853785}{12}n^{12} - \frac{10598363867}{20}n^{10} + \frac{4832271015}{4}n^8 - \frac{6854558639}{4}n^6 + \frac{5208839727}{4}n^4 - \frac{23749461029}{60}n^2$

30 次方求和公式: $\sum_{i=1}^{n} i^{30} = \frac{1}{31}n^{31} + \frac{1}{2}n^{30} + \frac{5}{2}n^{29} - \frac{203}{6}n^{27} + \frac{1131}{2}n^{25} - \frac{16965}{2}n^{23} + \frac{216775}{2}n^{21} - \frac{2304485}{2}n^{19} + \frac{19959975}{2}n^{17} - \frac{137514723}{2}n^{15} + \frac{731482225}{2}n^{13} - \frac{31795091601}{22}n^{11} + \frac{8053785025}{2}n^{9} - \frac{102818379585}{14}n^{7} + \frac{15626519181}{2}n^{5} - \frac{23749461029}{6}n^{3} + \frac{8615841276005}{14322}n$

3.7 二阶差分

给一段区间等差数列

4 组合数学

4.1 组合数公式

$\binom{n}{m}$ 表示 C_n^m

$A_n^m = \frac{n!}{m!}$,$C_n^m = \frac{n!}{m!(n-m)!}$,$A_n^m = C_n^m A_m^m$,$C_n^m = C_{n-1}^{m-1} + C_{n-1}^m$

$\sum_{i=0}^{m} C_n^i C_m^i = C_{n+m}^m \ \ (n \geq m)$(所有从这 n+m 个里面选 m 个的情况)

$C_n^k + C_{n+1}^k + ... + C_{n+m}^k = [C_{n+1}^{k+1} - C_n^{k+1}] + ... + [C_{n+m+1}^{k+1} - C_{n+m}^{k+1}] = C_{n+m+1}^{k+1} - C_n^{k+1}$ (注意 $C_n^{n+1} = 0$)

$\sum_{i=0}^{n} \binom{k+i}{i} = \binom{n+k+1}{n}, n, k \in N$

斐波那契数列通项公式：$f_n = \frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n]$，生成函数为 $F(x) = \frac{x}{1-x-x^2}$

二项式定理：$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}, n \in N$

$C_n^0 + C_n^1 + C_n^2 + ... + C_n^n = (1+1)^n$

$C_n^0 + C_n^2 + C_n^4 + ... + C_n^n = C_n^1 + C_n^3 + C_n^5 + ... + C_n^{n-1} = 2^{n-1}$ (n 为偶数)

$C_n^0 + C_n^2 + C_n^4 + ... + C_n^{n-1} = C_n^1 + C_n^3 + C_n^5 + ... + C_n^n = 2^{n-1}$ (n 为奇数)

$iC_n^i = nC_{n-1}^{i-1}(n \geq 2, 1 \leq i \leq n)$

$\sum_{i=1}^{n} i\binom{n}{i} = n\sum_{i=1}^{n} \binom{n-1}{i-1} = n2^{n-1}$

广义二项式定理：$(x+y)^a = \sum_{i=0}^{\infty} \binom{a}{i} x^{a-i} y^i$, 其中 $\binom{a}{i} = \frac{a^{\underline{i}}}{i!} = \frac{a(a-1)(a-2)...(a-i+1)}{i!}$

4.2 生成函数

多项式公式

$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ... + \frac{x^k}{k!} + ...$

$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + ... + (-1)^{k+1}\frac{x^k}{k} + ...$

$\frac{1}{1+x} = 1 - x + x^2 - x^3 + ... + (-1)^k x^k + ...$

$\frac{1}{1-x} = 1 + x + x^2 + x^3 + ... + x^k + ...$

$\ln \frac{1}{1-x} = -\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + ... + \frac{x^k}{k} + ...$

五边形定理：$\prod_{i=1}^{\infty}(1-x^i) = \sum_{i=-\infty}^{\infty}(-1)^i x^{i(3i-1)/2} = 1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} + \cdots$

普通生成函数 (OGF):

01 背包：重量为 k 的物品生成函数为 $1 + x^k$

多重背包：m 个重量为 k 的物品生成函数为 $1 + x^k + x^2 k + ... + x^m k = \frac{1-x^{(m+1)k}}{1-x}$

完全背包：重量为 k 的物品生成函数为 $1 + x^k + x^{2k} + ... = \frac{1}{1-x^k}$

背包装了 V 重量的方案数为所有物品生成函数卷积的 x^V 项系数

指数型生成函数 (EGF):

多次排列集计数：

n 个球分 m 种类型，第 i 种有 a_i 个，$\sum_{i=1}^{m} a_i = n$，则这 n 个球的排列方案数为 $\frac{n!}{a_1!a_2!...a_m!}$

$\hat{F}(x) = \sum_{i=0}^{n} \frac{f_i}{i!}x^i, \hat{G}(x) = \sum_{i=0}^{m} \frac{g_i}{i!}x^i, (n \geq m)$

EGF 卷积的组合意义：$\hat{F}(x)\hat{G}(x) = \sum_{i=0}^{n}(\sum_{j=0}^{i} f_{i-j}g_j \frac{1}{j!(i-j)!})x^i = \sum_{i=0}^{n+m} \frac{x^i}{i!} \sum_{j=0}^{i} \binom{i}{j} f_{i-j}g_j$ ($j > m$ 时 $\binom{i}{j} = 0$)

m 种类型的球，第 i 种有 a_i 个，求共取出 n 个进行排列的方案数：

$\hat{F}_i(x) = \sum_{i=0}^{a_i} \frac{1}{i!}x^i$，设 b_i 表示第 i 个球选中的个数

$\prod_{i=1}^{m} \hat{F}_i(x)$ 的 n 次项系数即为所有方案的 $\frac{1}{b_1!b_2!...b_m!}$ 之和，乘以 $n!$ 即为所求多重排列集计数

EGF 的多项式 Exp 的组合意义：

若 $\hat{F}(x)$ 中 $i!f_i$ 表示 i 个有标号的元素放入一个集合的方案数

则 $\hat{G}(x) = e^{\hat{F}(x)}$ 中 $i!g_i$ 表示 i 个有标号元素分成若干个无标号的非空无序集合的方案数

4.3 容斥原理

设有 n 个不同性质，A_i = 具有性质 i 的元素集合，$|A_i| = A_i$ 集合中元素个数

容斥定理用于求 $|A_1 \cup A_2 \cup ... \cup A_n|$ 和 $|\overline{A_1} \cap \overline{A_2} \cap ... \cap \overline{A_n}|$

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{m=1}^{n}(-1)^{m+1} \sum_{p_i<p_{i+1}} \left|\bigcap_{i=1}^{m} A_{p_i}\right|$$

$$\left|\bigcap_{i=1}^{n} \overline{A_i}\right| = |U| - \left|\bigcup_{i=1}^{n} A_i\right|$$

广义容斥原理:

设有 n 种不同的性质

P_k = 至少有 k 个性质的元素个数，Q_k = 恰好有 k 个性质的元素个数

$$Q_k = \sum_{i=0}^{n-k}(-1)^i C_{k+i}^{k} P_{k+i}$$

特别地，不具有所有性质的元素个数 $Q_0 = P_0 - P_1 + P_2 - P_3 + ...$

计数时不一定要计算所有状态，只需要递推至少拥有奇数个/偶数个性质的个数

二项式反演:

$$f_n = \sum_{i=0}^{n} C_n^i g_i \iff g_n = \sum_{i=0}^{n}(-1)^{n-i} C_n^i f_i$$

Min-Max 容斥:

$max(S) = \sum_{T \subseteq S}(-1)^{|T|-1} min(T)$

$min(S) = \sum_{T \subseteq S}(-1)^{|T|-1} max(T)$

第 k 大 $kmax(S)=\sum_{T\subseteq S}(-1)^{|T|-k}C_{|T|-1}^{k-1}min(T)$

第 k 小 $kmin(S)=\sum_{T\subseteq S}(-1)^{|T|-k}C_{|T|-1}^{k-1}max(T)$

4.4 卡特兰数

卡特兰数的组合意义为：n 个数的出栈序列有多少种，n 个结点可构造多少个不同的二叉树

卡特兰数 $H_n=\frac{C_{2n}^n}{n+1}=C_{2n}^n-C_{2n}^{n-1}=\frac{H_{n-1}(4n-2)}{n+1}$，$H_n=\sum_{i=0}^{n-1}H_iH_{n-1-i}$，$H_0=1$

4.5 斯特林数

第一类斯特林数

第一类 Stirling 数 $\begin{bmatrix}n\\k\end{bmatrix}$：将 n 个物体排成 k 个非空圆排列的方案数

递推式：$\begin{bmatrix}n\\k\end{bmatrix}=\begin{bmatrix}n-1\\k-1\end{bmatrix}+(n-1)\begin{bmatrix}n-1\\k\end{bmatrix}$，边界 $\begin{bmatrix}n\\k\end{bmatrix}=[n=0]$

第二类斯特林数

第二类 Stirling 数 $\begin{Bmatrix}n\\k\end{Bmatrix}$：将 n 个物体划分成 k 个非空的无编号集合的方案数

递推公式：$\begin{Bmatrix}n\\k\end{Bmatrix}=\begin{Bmatrix}n-1\\k-1\end{Bmatrix}+k\begin{Bmatrix}n-1\\k\end{Bmatrix}$，边界：$\begin{Bmatrix}n\\0\end{Bmatrix}=[n=0]$

$\begin{Bmatrix}n\\k\end{Bmatrix}=\sum_{i=0}^{k}(-1)^{k-i}\frac{i^n}{i!(k-i)!}$

阶乘幂与斯特林数

$x^{\overline{n}}=\sum_{i=0}^{n}\begin{bmatrix}n\\i\end{bmatrix}x^i$，$x^n=\sum_{i=0}^{n}\begin{Bmatrix}n\\i\end{Bmatrix}(-1)^{n-i}x^{\overline{i}}$，$x^{\underline{n}}=\sum_{i=0}^{n}(-1)^{n-i}\begin{bmatrix}n\\i\end{bmatrix}x^i$

$x^n=\sum_{i=0}^{n}\begin{Bmatrix}n\\i\end{Bmatrix}x^{\underline{i}}$，$x^n=\sum_{i=0}^{n}\begin{Bmatrix}n\\i\end{Bmatrix}x^{\underline{i}}=\sum_{i=0}^{n}\begin{Bmatrix}n\\i\end{Bmatrix}i!\binom{x}{i}$

自然数幂和 $F_k(n)=\sum_{i=0}^{n}i^k$

$x^n=x^{\underline{n}}-\sum_{i=0}^{n-1}\begin{bmatrix}n\\i\end{bmatrix}x^i$

$F_k(n)=\sum_{i=0}^{n}i^k=\sum_{i=0}^{n}[i^{\underline{k}}-\sum_{j=0}^{k-1}(-1)^{k-j}\begin{bmatrix}k\\j\end{bmatrix}i^j]$

$=\sum_{i=0}^{n}k!\binom{i}{k}-\sum_{j=0}^{k-1}(-1)^{k-j}\begin{bmatrix}k\\j\end{bmatrix}\sum_{i=0}^{n}i^j=\frac{(i+1)^{\underline{k+1}}}{k+1}-\sum_{i=0}^{k-1}(-1)^{k-i}\begin{bmatrix}k\\i\end{bmatrix}F_i(n)$

可以 $O(n^2)$ 递推，注意到不需要除法，因为连续 $k+1$ 个数中一定有一个数是 $k+1$ 的倍数

另外还可以做到 $O(k\log k)$，我们考虑利用第二类斯特林数进行化简，可以得到

$$F_k(n)=\sum_{i=0}^{n}i^k=\sum_{i=0}^{n}\sum_{j=0}^{k}\begin{Bmatrix}k\\j\end{Bmatrix}i^{\underline{j}}=\sum_{i=0}^{n}\sum_{j=0}^{k}\begin{Bmatrix}k\\j\end{Bmatrix}\binom{i}{j}j!$$

$$=\sum_{j=0}^{k}\begin{Bmatrix}k\\j\end{Bmatrix}j!\sum_{i=0}^{n}\binom{i}{j}=\sum_{j=0}^{k}\begin{Bmatrix}k\\j\end{Bmatrix}j!\binom{n+1}{j+1}$$

求斯特林数

第一类斯特林数-列

给定 n,k，求所有 $i\in[0,n]$，$\begin{bmatrix}i\\n\end{bmatrix}$，$n,k\le 2\times 10^5$

构造 $k=1$ 的指数型生成函数 $F(x)=\sum_{i=1}^{n}(i-1)!\frac{x^i}{i!}=\ln\frac{1}{1-x}$，答案就是 $\frac{F^k(x)}{k!}$，时间复杂度 $O(n\log n)$

第一类斯特林数-行

给定 n，求所有 $i\in[0,n]$，$\begin{bmatrix}n\\i\end{bmatrix}$，$n\le 2\times 10^5$

考虑 $\begin{bmatrix}n\\k\end{bmatrix}$ 的生成函数 $F(x)=\sum_{i=0}^{n}\begin{bmatrix}n\\i\end{bmatrix}x^i=x^{\overline{n}}$，同时我们有 $x^{\overline{2n}}=x^{\overline{n}}(x+n)^{\overline{n}}$，即 $F(x)=F_{2n}(x)F_n(x+n)$，那么我们直接倍增即可，时间复杂度为 $O(n\log n)$

```
Poly solve(int n) {
    if (!n) return Poly { 1 };
    Poly res = solve(n / 2);
    res = Pol::operator*(res, Pol::Offset(res, n / 2));
    if (n & 1) {
        res.push_back(0);
        for (int i = n; ~i; --i)
        res[i] = ((i ? res[i - 1] : 0) + 1ll * res[i] * (n - 1)) % p;
    }
    return res;
}
```

第二类斯特林数-列

给定 n,k，求所有 $i\in[0,n]$，$\begin{Bmatrix}i\\n\end{Bmatrix}$，$n,k\le 2\times 10^5$

令 $F_k(x)=\sum_{i=0}^{\infty}\begin{Bmatrix}i\\k\end{Bmatrix}x^i$，根据 $\begin{Bmatrix}n\\k\end{Bmatrix}=\begin{Bmatrix}n-1\\k-1\end{Bmatrix}+k\begin{Bmatrix}n-1\\k\end{Bmatrix}$，有 $F_k(x)=xF_{k-1}(x)+kxF_k(x)$，$F_0(x)=1$

即 $F_k(x)=\frac{x^k}{\prod_{i=1}^{k}(1-ix)}$，分治 NTT 求分母后求逆，时间复杂度 $O(n\log^2 n)$

第二类斯特林数-行

给定 n，求所有 $i \in [0,n]$，$\begin{Bmatrix} n \\ i \end{Bmatrix}$，$n \le 2 \times 10^5$

考虑第二类斯特林数通项公式 $\begin{Bmatrix} n \\ k \end{Bmatrix} = \sum_{i=0}^{k} (-1)^{k-i} \frac{i^n}{i!(k-i)!}$，我们令 $A_i = \frac{(-1)^i}{i!}, B_i = \frac{i^n}{i!}$，容易发现就是一个卷积的形式，时间复杂度 $O(n \log n)$

4.6 贝尔数

贝尔数 B_n 表示基数为 n 的集合的划分方法的数目

组合意义: 将 n 个有标号的小球放到任意多个无标号盒子中的方案数

$B_n = \sum_{i=0}^{n} \begin{Bmatrix} n \\ i \end{Bmatrix}$

递推式：$B_n = \sum_{i=0}^{n-1} \binom{n-1}{i} B_i$，边界为 $B_0 = 1$

求贝尔数

$O(n^2)$ 递推，$O(n \log n)$ 多项式 exp

注意 B_n 的递推式 $B_n = \sum_{i=0}^{n-1} \binom{n-1}{i} B_i$，令 $B(x)$ 为 B 的 EGF，那么容易得到 $B(x) = 1 + \int B(x) e^x \mathrm{dx}$，根据 $B(0) = 1$ 能够的得到 $B(x) = e^{e^x - 1}$

```
ll fac[maxn], inv[maxn]; Poly B;
void init(int n) {
    Poly A(n);
    fac[0] = 1; for (int i = 1; i <= n; ++i) fac[i] = fac[i - 1] * i % p;
    inv[n] = pow_mod(fac[n], p - 2); for (int i = n - 1; ~i; --i) inv[i] = inv[i
       + 1] * (i + 1) % p;
    for (int i = 0; i < n; ++i) A[i] = inv[i]; A[0] = 0;//泰勒展开
    B = Pol::Exp(A);
    for (int i = 0; i < n; ++i) B[i] = B[i] * fac[i] % p;
}
```

贝尔数前 15 项:$1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322$

贝尔数性质

$B_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}$

$B_n = \sum_{k=0}^{n} S(n,k)$，即每个贝尔数都是一行第二类斯特林数的和

$B_{p+n} = B_n + B_{n+1} \pmod p, (p \in prime)$

4.7 小球入盒模型

n 个球	m 个盒子	是否允许有空盒子	方案数
不同	不同	是	m^k
不同	不同	否	$m!S_2(n,m)$
不同	相同	是	$\sum_{i=1}^{m} S_2(n,i)$
不同	相同	否	$S_2(n,m)$
相同	不同	是	C_{n+m-1}^{m-1}
相同	不同	否	C_{n-1}^{m-1}
相同	相同	是	$\frac{1}{(1-x)(1-x^2)...(1-x^m)}$ 的 x^n 项系数
相同	相同	否	$\frac{x^m}{(1-x)(1-x^2)...(1-x^m)}$ 的 x^n 项系数

$S_2(n,m)$ 分别表示第二类斯特林数

4.8 欧拉数

对于一个长度为 n 且有 k 个上升 $(p_i < p_{i+1})$ 的排列定义为欧拉数 $\left\langle {n \atop k} \right\rangle$

递推式：$\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle$，边界为 $\left\langle {n \atop 0} \right\rangle = 1$

恒等式：

$x^n = \sum_{k=0}^{n} \left\langle {n \atop k} \right\rangle \binom{x+k}{n}, \left\langle {n \atop k} \right\rangle = \sum_{i=0}^{k} (-1)^i \binom{n+1}{i} (k-i+1)^n$

$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{i=0}^{n} \left\langle {n \atop i} \right\rangle \binom{i}{n-k}, \left\langle {n \atop k} \right\rangle = \sum_{i=0}^{n} (-1)^{n-i-k} i! \left\{ {n \atop i} \right\} \binom{n-i}{k}$

求欧拉数

给定 n，求所有 $i \in [0,n]$，$\left\langle {n \atop i} \right\rangle$，$n \le 2 \times 10^5$

考虑恒等式 $\left\langle {n \atop k} \right\rangle = \sum_{i=0}^{n} (-1)^{n-i-k} i! \left\{ {n \atop i} \right\} \binom{n-i}{k}$，化简可以得到 $k! \left\langle {n \atop k} \right\rangle = \sum_{i=0}^{n} \frac{(-1)^{n-(i+k)}}{(n-(i+k))!} \left\{ {n \atop i} \right\} i!(n-i)!$，我们令 $A_k = \left\{ {n \atop k} \right\} k!(n-k)!, B_k = \frac{(-1)^{n-k}}{(n-k)!}$，容易发现这是一个减法卷积的式子，第二类斯特林数同样可以用卷积求，时间复杂度 $O(n \log n)$

二阶欧拉数

对于有 k 个上升的多重集 $1,1,2,2,\cdots,n,n$ 的排列定义为 $\left\langle \left\langle {n \atop k} \right\rangle \right\rangle$

递推式：$\left\langle \left\langle {n \atop k} \right\rangle \right\rangle = (k+1)\left\langle \left\langle {n-1 \atop k} \right\rangle \right\rangle + (2n-k-1)\left\langle \left\langle {n-1 \atop k-1} \right\rangle \right\rangle$

恒等式：$\begin{Bmatrix} x \\ x-n \end{Bmatrix} = \sum_{i=0}^{n-1} \left\langle\left\langle {n \atop i} \right\rangle\right\rangle \binom{x+n-k-1}{2n}$

4.9 置换群

Burnside 定理：$M = \frac{1}{|G|} \sum_{g \in G} c(g)$

群 G 的等价类数量 = 群 G 中所有置换 g 的轮换数量 $c(g)$ 的平均值

Polya 定理：$M = \frac{1}{|G|} \sum_{g \in G} m^{c(g)}$

4.10 杨表

给定 n 个方格的杨表，把 $[1, n]$ 填入，要求从上到下，从左到右单调递减

$hook(x)$ = 同行右边的方格数 + 同列下面的方格数 $+1$

方案数 $dim\pi_\lambda = \frac{n!}{\prod\limits_{x \in Y(\lambda)} hook(x)}$

5 数据结构

5.1 树状数组

```
ll tr[N]; // tr[i] = [i-lowbit(i)+1,i]区间的信息
void add(int x,ll k){for(;x<=n;x+=x&-x)tr[x]+=k;}//需要保证x!=0
ll qry(int x){ll s=0;for(;x;x-=x&-x)s+=tr[x];return s;}
```

树状数组维护二阶前缀和

要实现区间加区间求和，需要维护差分数组的二阶前缀和

设 d, a, s 分别为差分数组, 原数组, 前缀和数组

$s[n] = \sum\limits_{i=1}^{n} a[i] = \sum\limits_{i=1}^{n} (n - i + 1) d[i]$

接下来分别维护 $d[i]$ 和 $i \times d[i]$ 的前缀和即可

```
ll t1[N],t2[N];
void add(int x,ll k){
    for(int i=x;i<=n;i+=i&-i){
        t1[i]+=k;t2[i]+=x*k;
    }
}
ll qry(int x){
    ll s=0;
    for(int i=x;i;i-=i&-i)s+=(x+1)*t1[i]-t2[i];
    return s;
}
```

树状数组上二分

```
int bs(ll k){//最大的sum<=k的下标,若求最小的sum=k的下标,可以bs(k-1)+1
    int now=0;ll sum=0;
    for(int i=21;i>=0;--i){
        if((now|1<<i)<=n&&sum+t[now|1<<i]<=k){
            now|=1<<i;sum+=t[now];
        }
    }
    return now;
}
```

5.2 线段树

5.2.1 懒惰标记线段树

```
int n,a[N];
struct sgt{
    int l,r;
    ll v,tag;
}t[N<<2];
#define ls (p<<1)
#define rs (p<<1|1)
void pushup(int p){t[p].v=t[ls].v+t[rs].v;}
void up1(int p,ll x){//结点打上标记
    t[p].tag+=x;t[p].v+=x*(t[p].r-t[p].l+1);
}
void pushdown(int p){
    if(t[p].tag){
        up1(ls,t[p].tag);up1(rs,t[p].tag);t[p].tag=0;
    }
}
void build(int l,int r,int p){
    t[p].l=l;t[p].r=r;
    if(l==r){t[p].v=a[l];return;}
    int mid=l+r>>1;
    build(l,mid,ls);build(mid+1,r,rs);
    pushup(p);
}
void upd(int l,int r,int p,ll c){
    if(l<=t[p].l&&t[p].r<=r){
        up1(p,c);return;
```

```
    }
    pushdown(p);
    int mid=t[p].l+t[p].r>>1;
    if(l<=mid)upd(l,r,ls,c);
    if(r>mid)upd(l,r,rs,c);
    pushup(p);
}
ll qry(int l,int r,int p){
    if(l<=t[p].l&&t[p].r<=r)return t[p].v;
    pushdown(p);
    int mid=t[p].l+t[p].r>>1; ll s=0;
    if(l<=mid)s=qry(l,r,ls);
    if(r>mid)s+=qry(l,r,rs);
    return s;
}
```

懒惰标记优先级 1、先乘法后加法：设区间乘法标记为 M_{tag}，加法标记为 A_{tag}，每次更新把区间乘以 M，再增加 A

$\sum a_i = \sum(Ma_i + A) = M\sum a_i + A(r-l+1)$

$M_{tag} = M \times M_{tag}$，$A_{tag} = M \times A_{tag} + A$

```
void up1(int p,int mul,int add){
    t[p].sum=mul*t[p].sum+add;
    t[p].mul*=mul;t[p].add=mul*t[p].add+add;
}
void upd(int p,int l,int r,int mul,int add){
    if(l<=t[p].l&&t[p].r<=r){
        up1(p,mul,add);return;
    }
    pushdown(p);
    int mid=t[p].l+t[p].r>>1;
    if(l<=mid)upd(ls,l,r,mul,add);
    if(r>mid)upd(rs,l,r,mul,add);
    pushup(p);
}
//区间乘c: upd(1,l,r,c,0)    区间加c:upd(1,l,r,1,c)
```

2、维护区间交叉项和

维护区间 $\sum a_i, \sum b_i, \sum a_i^2, \sum b_i^2, \sum a_i b_i$

维护区间乘法矩阵 $M_{tag} = \begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}$，区间加法 $A_{tag} = \begin{bmatrix} t_a \\ t_b \end{bmatrix}$，设加入新的 tag 分别为 M, A

$\sum a_i b_i = \sum(m_{00}a_i + m_{01}b_i + t_a)(m_{10}a_i + m_{11}b_i + t_b)$

$= m_{00}m_{10}\sum a_i^2 + m_{01}m_{11}\sum b_i^2 + (m_{00}m_{11}+m_{01}m_{10})\sum a_i b_i$

$+(t_b m_{00}+t_a m_{10})\sum a_i + (t_a m_{11}+t_b m_{01})\sum b_i + t_a t_b(r-l+1)$

$\sum a_i^2 = \sum(m_{00}a_i + m_{01}b_i + t_a)^2$

$= m_{00}^2\sum a_i^2 + m_{01}^2\sum b_i^2 + 2m_{00}m_{01}\sum a_i b_i + 2t_a m_{00}\sum a_i + 2t_a m_{01}\sum b_i + t_a^2(r-l+1)$

$\sum b_i^2 = \sum(m_{10}a_i + m_{11}b_i + t_b)^2$

$= m_{10}^2\sum a_i^2 + m_{11}^2\sum b_i^2 + 2m_{10}m_{11}\sum a_i b_i + 2t_b m_{10}\sum a_i + 2t_b m_{11}\sum b_i + t_b^2(r-l+1)$

$\sum a_i = \sum(m_{00}a_i + m_{01}b_i + t_a) = m_{00}\sum a_i + m_{01}\sum b_i + t_a(r-l+1)$

$\sum b_i = \sum(m_{10}a_i + m_{11}b_i + t_b) = m_{10}\sum a_i + m_{11}\sum b_i + t_b(r-l+1)$

$M_{tag} = M \times M_{tag}$，$A_{tag} = M \times A_{tag} + A$

3、先清空，再赋值

cl 表示是否清空 tag，add 表示加法 tag

```
void up1(int p,int cl,int add){
    if(cl){
        t[p].sum=add*(t[p].r-t[p].l+1);
        t[p].cl=1;t[p].add=add;
    }
    else{
        t[p].sum+=add*(t[p].r-t[p].l+1);
        t[p].add+=add;
    }
}
```

5.2.2 动态开点线段树

```
const int N=1e5+5,M=N*150;
struct sgt{
    int ls,rs;ll v,tag;//左右儿子,区间和,加法tag
}t[M];
int rt,tot;//多棵树需要rt[N]
int newnode(){
    int p=++tot;t[p].ls=t[p].rs=t[p].v=t[p].tag=0;return p;
}
void up1(int p,ll c,int len){
    if(!p)p=newnode();
    t[p].v+=c*len;t[p].tag+=c;
}
void pushdown(int p,int len){
    if(t[p].tag){
        up1(t[p].ls,t[p].tag,len+1>>1);
```

```
        up1(t[p].rs,t[p].tag,len>>1);
        t[p].tag=0;
    }
}
void upd(int &p,int l,int r,int L,int R,ll c){
    if(!p)p=newnode();
    if(L<=l&&r<=R){up1(p,c,r-l+1);return;}
    pushdown(p,r-l+1);
    int mid=l+r>>1;
    if(L<=mid)upd(t[p].ls,l,mid,L,R,c);
    if(R>mid)upd(t[p].rs,mid+1,r,L,R,c);
    t[p].v=t[t[p].ls].v+t[t[p].rs].v;
}
ll qry(int p,int l,int r,int L,int R){
    if(!p)return 0;
    if(L<=l&&r<=R)return t[p].v;
    pushdown(p,r-l+1);
    int mid=l+r>>1;ll s=0;
    if(L<=mid)s=qry(t[p].ls,l,mid,L,R);
    if(R>mid)s+=qry(t[p].rs,mid+1,r,L,R);
    return s;
}
int main(){
    int n,q;scanf("%d%d",&n,&q);
    for(int x,i=1;i<=n;++i)scanf("%d",&x),upd(rt,1,n,i,i,x);
    while(q--){
        int op,l,r,k;scanf("%d%d%d",&op,&l,&r);
        if(op==1){scanf("%d",&k);upd(rt,1,n,l,r,k);}
        else printf("%lld\n",qry(rt,1,n,l,r));
    }
}
```

5.2.3 复杂信息合并线段树

```
struct sgt{
    int l,r;
    int lmx,rmx;//包含左/右端点的区间最大子段和
    int mx,sum;//区间最大子段和,区间和
}t[N<<2];
#define ls (p<<1)
#define rs (p<<1|1)
```

```
sgt pushup(sgt sl,sgt sr){
    sgt s={sl.l,sr.r};
    s.lmx=max(sl.lmx,sl.sum+sr.lmx);
    s.rmx=max(sr.rmx,sr.sum+sl.rmx);
    s.mx=max({sl.mx,sr.mx,sl.rmx+sr.lmx});
    s.sum=sl.sum+sr.sum;
    return s;
}
void upd(int p,int idx,int c){
    if(t[p].l==t[p].r){
        t[p].lmx=t[p].rmx=t[p].sum=t[p].mx=c;
        return;
    }
    int mid=t[p].l+t[p].r>>1;
    if(idx<=mid)upd(ls,idx,c);
    else upd(rs,idx,c);
    t[p]=pushup(t[ls],t[rs]);
}
sgt qry(int p,int l,int r){
    if(l<=t[p].l&&t[p].r<=r)return t[p];
    int mid=t[p].l+t[p].r>>1;
    if(r<=mid)return qry(ls,l,r);
    if(l>mid)return qry(rs,l,r);
    return pushup(qry(ls,l,r),qry(rs,l,r));
}
```

5.2.4 暴力更新线段树

```
void upd(int l,int r,int p,ll c){//少量暴力更新子树
    if(l<=t[p].l&&t[p].r<=r&&t[p].flag){//该子树可以剪枝
        up1(p,c);//打上标记
        return;
    }
    if(t[p].l==t[p].r){ //叶子节点更新
        return;
    }
    pushdown(p);
    int mid=t[p].l+t[p].r>>1;
    if(l<=mid)upd(l,r,ls,c);
    if(r>mid)upde(l,r,rs,c);
    pushup(p);
```

```
}
```

5.2.5 pushup 带询问

给定 a, b 数组 ($a_i \in [1, 10^9], b_i \in \{0, 1\}$)

询问 $[l, r]$ 区间从 l 开始的最长上升子序列 $a_{p_1}, a_{p_2}, ..., a_{p_k}$(有递增必须选)，求 $\sum_{i=1}^{k-1} b_{p_i} \oplus b_{p_{i+1}}$

单点修改 $a_i := c$，区间修改 $[l, r]$ 的 $b_i := b_i \oplus 1$

```
struct sgt{
    int l,r;
    pair<int,int>mx;//a的最大值,最大值对应的b
    int sum,tag;//区间答案,区间翻转标记
}t[N<<2];
#define ls (p<<1)
#define rs (p<<1|1)
void up1(int p){t[p].tag^=1;t[p].mx.second^=1;}
void pushdown(int p){
    if(t[p].tag){up1(ls);up1(rs);t[p].tag=0;}
}
int calc(int p,pair<int,int>&mx){
    if(t[p].l==t[p].r){
        if(t[p].mx.first>=mx.first){
            int ans=(mx.second!=t[p].mx.second);
            mx=t[p].mx;return ans;
        }
        else return 0;
    }
    pushdown(p);
    int mid=t[p].l+t[p].r>>1;
    if(t[p].mx.first<mx.first)return 0;
    if(t[ls].mx.first<mx.first)return calc(rs,mx);
    else{
        int ans=calc(ls,mx)+t[p].sum-t[ls].sum;
        mx=t[p].mx;return ans;
    }
}
void pushup(int p){
    if(t[ls].mx.first<=t[rs].mx.first)t[p].mx=t[rs].mx;
    else t[p].mx=t[ls].mx;
    pair<int,int>pr=t[ls].mx;
```

```
    t[p].sum=t[ls].sum+calc(rs,pr);
}
void build(int p,int l,int r){
    t[p].l=l;t[p].r=r;
    if(l==r){
        t[p].mx={a[l],b[l]};t[p].sum=1;return;
    }
    int mid=l+r>>1;
    build(ls,l,mid);build(rs,mid+1,r);
    pushup(p);
}
void upd1(int p,int idx,int x){
    if(t[p].l==t[p].r){t[p].mx.first=x;return;}
    pushdown(p);
    int mid=t[p].l+t[p].r>>1;
    if(idx<=mid)upd1(ls,idx,x);
    else upd1(rs,idx,x);
    pushup(p);
}
void upd2(int p,int l,int r){
    if(l<=t[p].l&&t[p].r<=r){up1(p);return;}
    pushdown(p);
    int mid=t[p].l+t[p].r>>1;
    if(l<=mid)upd2(ls,l,r);
    if(r>mid)upd2(rs,l,r);
    pushup(p);
}
int qry(int p,int l,int r,pair<int,int>&mx){
    if(l<=t[p].l&&t[p].r<=r)return calc(p,mx);
    pushdown(p);
    int mid=t[p].l+t[p].r>>1,s=0;
    if(l<=mid)s=qry(ls,l,r,mx);
    if(r>mid)s+=qry(rs,l,r,mx);
    return s;
}
```

5.2.6 线段树上二分

```
int qry(int p,int l,int x){//[l,n]第一个<=x的下标
    if(t[p].l==t[p].r)return t[p].l;
    pushdown(p);
```

```
    int mid=t[p].l+t[p].r>>1;
    if(l<=t[p].l){
        if(t[ls].mn<=x)return qry(ls,l,x);
        if(t[rs].mn>x)return -1;
        return qry(rs,l,x);
    }
    else if(l<=mid){
        int s=qry(ls,l,x);
        if(s!=-1)return s;
        if(t[rs].mn>x)return -1;
        return qry(rs,l,x);
    }
    else{
        if(t[rs].mn>x)return -1;
        return qry(rs,l,x);
    }
}
```

5.2.7 线段树合并

```
void merge(int &p,int q){
    if(!p||!q){p=p|q;return;}
    t[p].v+=t[q].v;
    merge(t[p].ls,t[q].ls);
    merge(t[p].rs,t[q].rs);
}
int merge(int p,int q){//树形DP保留树上所有结点答案时，线段树合并需要新建结点
    if(!p||!q)return p|q;
    int now=++tot;
    t[now].v=t[p].v+t[q].v;
    t[now].ls=merge(t[p].ls,t[q].ls);
    t[now].rs=merge(t[p].rs,t[q].rs);
    return now;
}
```

5.2.8 线段树分裂

```
int n,m,q,rt[N],tot,a[N];
map<int,int>mp;//mp[区间左端点]=区间右端点
int vis[N];//0:升序 1:降序
struct sgt{
```

```
5       int ls,rs;
6       int v,rev,rtag;//tag=1:需要翻转
7   }t[N<<5];
8   int newnode(int op){
9       int p=++tot;
10      t[p].rev=op;
11      return p;
12  }
13  void pushup(int p){
14      t[p].v=t[t[p].ls].v+t[t[p].rs].v;
15  }
16  void upd(int &p,int l,int r,int x){
17      if(!p)p=newnode(0);
18      if(l==r){
19          ++t[p].v;
20          return;
21      }
22      int mid=l+r>>1;
23      if(x<=mid)upd(t[p].ls,l,mid,x);
24      else upd(t[p].rs,mid+1,r,x);
25      pushup(p);
26  }
27  void reverse1(int p){
28      if(!p)return;
29      t[p].rtag^=1;
30      t[p].rev^=1;
31      swap(t[p].ls,t[p].rs);
32  }
33  void pushdown(int p){
34      if(t[p].rtag){
35          reverse1(t[p].ls);
36          reverse1(t[p].rs);
37          t[p].rtag=0;
38      }
39  }
40  void merge(int &p,int q){
41      if(!p||!q){
42          p=p|q;return;
43      }
44      pushdown(p);pushdown(q);
45      merge(t[p].ls,t[q].ls);
```

```
46      merge(t[p].rs,t[q].rs);
47      pushup(p);
48  }
49  int kth(int p,int l,int r,int k){//从左往右第k个
50      if(l==r)return l;
51      pushdown(p);
52      int mid=l+r>>1;
53      if(!t[p].rev){
54          if(t[t[p].ls].v>=k)return kth(t[p].ls,l,mid,k);
55          else return kth(t[p].rs,mid+1,r,k-t[t[p].ls].v);
56      }
57      else{
58          if(t[t[p].ls].v>=k)return kth(t[p].ls,mid+1,r,k);
59          else return kth(t[p].rs,l,mid,k-t[t[p].ls].v);
60      }
61  }
62  void splitL(int p,int &q,ll k){
63      if(!p)return;
64      q=newnode(t[p].rev);//新结点翻转状态一致
65      pushdown(p);
66      ll s=t[t[p].ls].v;
67      if(k>=s){
68          t[q].ls=t[p].ls;t[p].ls=0;
69          if(k>s)splitL(t[p].rs,t[q].rs,k-s);
70      }
71      else splitL(t[p].ls,t[q].ls,k);
72      pushup(p);pushup(q);
73  }
74  void split(int L,int R,int l,int r,int op){//从[L,R]中分裂出[l,r]
75      if(l!=L){
76          rt[l]=rt[L];
77          splitL(rt[l],rt[L],l-L);
78          mp[L]=l-1;mp[l]=R;
79          vis[l]=vis[L];
80      }
81      if(r!=R){
82          rt[r+1]=rt[l];
83          splitL(rt[r+1],rt[l],r-l+1);
84          mp[l]=r;mp[r+1]=R;
85          vis[r+1]=vis[L];
86      }
```

```
87      if(vis[l]!=op){
88          vis[l]=op;
89          reverse1(rt[l]);
90      }
91  }
92  int main(){
93      scanf("%d%d",&n,&m);
94      for(int i=1;i<=n;++i){
95          scanf("%d",&a[i]);
96          upd(rt[i],1,n,a[i]);
97          mp[i]=i;vis[i]=0;
98      }
99      while(m--){
100         int op,l,r;scanf("%d%d%d",&op,&l,&r);
101         auto pr1=*--mp.upper_bound(l);
102         auto pr2=*--mp.upper_bound(r);
103         int a=pr1.first,b=pr1.second;
104         int c=pr2.first,d=pr2.second;
105         if(pr1==pr2){
106             split(a,b,l,r,op);
107         }
108         else{
109             split(a,b,l,b,op); split(c,d,c,r,op);
110             for(auto it=mp.upper_bound(l);it!=mp.end()&&it->second<=r;it=mp.erase
                    (it)){
111                 if (vis[it->first]!=op){
112                     reverse1(rt[it->first]),vis[it->first]=op;
113                 }
114                 merge(rt[l],rt[it->first]);
115             }
116             mp[l]=r;
117         }
118     }
119     scanf("%d",&q);
120     int pos=(--mp.upper_bound(q))->first;
121     int ans=kth(rt[pos],1,n,q-pos+1);
122     printf("%d\n",ans);
123 }
```

5.2.9 扫描线

```
//求n个矩形面积并
struct rectangle{int x1,y1,x2,y2;}a[N];
int n,b[N<<1],h,hb;
struct line{int x1,x2,y,c;}c[N<<1];
bool cmp(line a,line b){return a.y<b.y;}
struct sgt{
    int l,r,v,cnt;
}t[N<<3];//N个矩形有2N个x坐标，要8N线段树结点
#define ls (p<<1)
#define rs (p<<1|1)
void build(int l,int r,int p){
    t[p].l=l;t[p].r=r;
    if(l==r)return;
    int mid=l+r>>1;
    build(l,mid,ls);build(mid+1,r,rs);
}
void pushup(int p){
    if(t[p].cnt)t[p].v=b[t[p].r+1]-b[t[p].l];//整个区间都有
    else{
        if(t[p].l==t[p].r)t[p].v=0;//叶子结点特判避免RE
        else t[p].v=t[ls].v+t[rs].v;//根据儿子更新
    }
}
void upd(int l,int r,int p,int c){//[l,r]被覆盖次数+=c
    if(l<=t[p].l&&t[p].r<=r){
        t[p].cnt+=c; pushup(p);//pushup来更新,比较方便
        return;
    }
    int mid=t[p].l+t[p].r>>1;
    if(l<=mid)upd(l,r,ls,c);
    if(r>mid)upd(l,r,rs,c);
    pushup(p);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i)scanf("%d%d%d%d",&a[i].x1,&a[i].y1,&a[i].x2,&a[i].y2);
    for(int i=1;i<=n;++i){b[++hb]=a[i].x1;b[++hb]=a[i].x2;}
    sort(b+1,b+1+hb);hb=unique(b+1,b+1+hb)-b-1;
    for(int i=1;i<=n;++i){//对x离散化
        a[i].x1=lower_bound(b+1,b+1+hb,a[i].x1)-b;
        a[i].x2=lower_bound(b+1,b+1+hb,a[i].x2)-b;
```

```
    }
    for(int i=1;i<=n;++i){//在y1对x的区间+1,在y2对x的区间-1,大于0就是有的
        c[++h]={a[i].x1,a[i].x2,a[i].y1,1};
        c[++h]={a[i].x1,a[i].x2,a[i].y2,-1};
    }
    sort(c+1,c+1+h,cmp);
    build(1,hb-1,1);//化点为线,x坐标下标:x1--x2--x3--x4--x5--x6--x7
    long long ans=0;//          线段下标:  s1  s2  s3  s4  s5  s6
    for(int i=1;i<h;++i){
        upd(c[i].x1,c[i].x2-1,1,c[i].c);
        ans+=1ll*t[1].v*(c[i+1].y-c[i].y);//线段之和*高度(重合的扫描线高度差为0)
    }
    printf("%lld\n",ans);
}
```

5.2.10 segment tree beats

实现区间加，区间对一个数取 min，区间和

维护区间最大值 mx1 和区间次大值 mx2，当区间对 c 取 min 时，c 小于 mx2 就暴力，否则打标记

```
void pushup(int p){
    t[p].v=t[ls].v+t[rs].v;
    if(t[ls].mx1>t[rs].mx1){
        t[p].mx1=t[ls].mx1;
        t[p].cnt=t[ls].cnt;
        t[p].mx2=max(t[ls].mx2,t[rs].mx1);
    }
    else if(t[ls].mx1==t[rs].mx1){
        t[p].mx1=t[ls].mx1;
        t[p].cnt=t[ls].cnt+t[rs].cnt;
        t[p].mx2=max(t[ls].mx2,t[rs].mx2);
    }
    else{
        t[p].mx1=t[rs].mx1;
        t[p].cnt=t[rs].cnt;
        t[p].mx2=max(t[rs].mx2,t[ls].mx1);
    }
}
void up1(int p,int c){
    if(c<t[p].mx1){
        t[p].tag=c;
```

```
        t[p].v-=1ll*(t[p].mx1-c)*t[p].cnt;
        t[p].mx1=c;
    }
}
void upd(int l,int r,int p,int c){
    if(l<=t[p].l&&t[p].r<=r&&c>=t[p].mx2){
        up1(p,c);return;
    }
    if(t[p].l==t[p].r){
        t[p].cnt=1;
        t[p].mx1=c;
        t[p].mx2=-1;
        t[p].tag=c;
        t[p].v=c;
        return;
    }
    pushdown(p);
    int mid=t[p].l+t[p].r>>1;
    if(l<=mid)upd(l,r,ls,c);
    if(r>mid)upd(l,r,rs,c);
    pushup(p);
}
```

5.3 单调栈

左边最近的小于 a[i] 的下标

```
int a[N],L[N],n;//L[i]=左边最近的小于a[i]的下标
int sta[N],top;//单调递增的栈
int main(){
    for(int i=1;i<=n;++i){
        while(top&&a[sta[top]]>=a[i]){
            //sta[top]左边第一个小于它的是sta[top-1]
            //sta[top]右边第一个小于等于它的是i
            --top;
        }
        L[i]=sta[top];sta[++top]=i;
    }
}
```

最大子矩形

```
//给定一个只有A和B的矩阵,求最大的全是A的矩阵
```

```
int n,m,a[N][N],ans;//a[i][j]=(i,j)最大向上扩展(全A)的长度
for(int i=1;i<=n;++i){//对每一行向上扩展取max
    sta[0]=0;
    for(int j=1;j<=m+1;++j){
        if(s[i][j]=='B'||j==m+1){//该位置是非法位置
            while(top){//清空栈并更新答案
                ans=max(ans,(j-sta[top-1]-1)*a[i][sta[top]]);
                --top;
            }
            sta[0]=j;//更新栈低下标
        }
        else{//该位置是合法位置,需要在单调栈中对最大的矩形取max
            while(top&&a[i][sta[top]]>a[i][j]){
                ans=max(ans,(j-sta[top-1]-1)*a[i][sta[top]]);
                --top;
            }
            sta[++top]=j;
        }
    }
}
```

计算矩形个数

```
struct P{
    int x,y;
    inline bool operator < (const P &b)const{
        return x<b.x||x==b.x&&y<b.y;
    }
};
ll calc(int h[],int l,int r){//[l,r]区间有几个矩形，第i个位置高度为h[i]
    static int sta[N],top;
    //vl[i]=左边最近的小于h[i]的位置,不存在时为l-1
    //vr[i]=右边最近的小于等于h[i]的位置,不存在时为r+1
    //ans=sum(vr[i]-i)*(i-vl[i])*h[i],实际不需要储存vl和vr数组
    ll ans=0;   sta[top=0]=l-1;
    for(int i=l;i<=r;++i){
        while(top&&h[sta[top]]>=h[i]){
            ans+=ll(sta[top]-sta[top-1])*(i-sta[top])*h[sta[top]];
            --top;
        }
        sta[++top]=i;
    }
```

```
    while(top){
        ans+=ll(sta[top]-sta[top-1])*(r+1-sta[top])*h[sta[top]];
        --top;
    }
    return ans;
}
ll cnt_rectangle(vector<P>&v){
    static int h[N][N];//h[i][j]=(i,j)位置往上最多延申几个(包括自身)
    int top=0;
    sort(v.begin(),v.end());
    for(auto &p:v)h[p.x][p.y]=1;//记录询问的点
    ll ans=0;
    for(int i=0;i<v.size();++i){
        int j=i,x=v[i].x;
        while(j+1<v.size()&&v[j+1].x==x)++j;//提取同一行
        for(int k=i;k<=j;++k)h[x][v[k].y]=h[x-1][v[k].y]+1;
        for(int k=i;k<=j;++k){
            int t=k;
            while(t+1<=j&&v[t+1].y==v[t].y+1)++t;//提取相连的
            ans+=calc(h[x],v[k].y,v[t].y);
            k=t;
        }
        i=j;
    }
    for(auto &p:v)h[p.x][p.y]=0;//删除询问的点
    return ans;
}
```

5.4 单调队列

```
//dp[i]=min(dp[i-k],dp[i-k+1],...,dp[i-1])+a[i]
int q[N],fr=0;bc=-1;//单增队列,保证队首为区间最小值
q[++bc]=0;//先加入一个dp[0]=a[0]=0
for(int i=1;i<=n;++i){
    dp[i]=dp[q[fr]]+a[i];
    while(bc>=fr&&dp[q[bc]]>=dp[i])--bc;
    q[++bc]=i;
    while(bc>=fr&&q[fr]<=i-k)++fr;
}
```

5.5 ST 表

```
int st[22][N];
for(int i=2;i<N;++i)Log2[i]=Log2[i>>1]+1;//注意i从2开始
for(int i=1;i<=n;++i)st[0][i]=a[i];
for(int i=1;1<<i<=n;++i){
    for(int j=1;j+(1<<i)-1<=n;++j){
        st[i][j]=max(st[i-1][j],st[i-1][j+(1<<i-1)]);
    }
}
int qry(int l,int r){
    int k=Log2[r-l+1];//也可以用__lg(r-l+1);
    return max(st[k][l],st[k][r-(1<<k)+1]);
}
```

5.6 哈希表

```
const int HA=19260817,maxn=3e6+5;
struct Hashtable{
    int h,hd[HA],nx[maxn];ll key[maxn]; ll val[maxn];
    void add(int x,ll y){key[++h]=y;val[h]=0;nx[h]=hd[x];hd[x]=h;}
    bool find(ll x){for(int i=hd[x%HA];i;i=nx[i])if(key[i]==x)return 1;return 0;}
    ll& operator [] (const ll &x){
        for(int i=hd[x%HA];i;i=nx[i])if(key[i]==x)return val[i];
        add(x%HA,x);return val[h];
    }
}mp1,mp2;
```

5.7 并查集

路径压缩并查集

```
int n,fa[N];
int find(int x){
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
int main(){
    for(int i=1;i<=n;++i)fa[i]=i;//初始化
    fa[find(x)]=find(y);//连接x,y
}
```

划分两个集合，可以令 x 为选择 x，x+n 为不选择 x，若 x 和 y+n 在同一个并查集，代表某一个集合选 x 并且不选 y

按秩合并并查集

```
int fa[N],sz[N];//也可以按dep合并
int find(int x){return x==fa[x]?x:find(fa[x]);}
void merge(int x,int y){
    x=find(x);y=find(y);
    if(sz[x]>sz[y])swap(x,y);
    fa[x]=y;sz[y]+=sz[x];
}
```

可撤销并查集

```
int n,fa[N],sz[N],top;
pair<int,int>sta[N];
int find(int x){
    return x==fa[x]?x:find(fa[x]);
}
void merge(int x,int y){
    int x=find(x),y=find(y);
    if(sz[fx]>sz[fy])swap(x,y);
    fa[x]=y;sz[y]+=sz[x];
    sta[++top]=make_pair(x,y);
}
void del(){//后连接的必须先撤销
    pair<int,int>pr=sta[top--];
    int x=pr.first,y=pr.second;
    fa[x]=x;sz[y]-=sz[x];
}
```

可持久化并查集

```
const int N=1e5+5,Q=2e5+5,LOGN=log2(N)+1;
int n,rt[N<<1],tot,q;
struct sgt{
    int ls,rs;
    int fa,d;
}t[(N+Q)*LOGN];
int build(int l,int r){
    int p=++tot;
    if(l==r){
        t[p].fa=l;t[p].d=1;
```

```
        return p;
    }
    int mid=l+r>>1;
    t[p].ls=build(l,mid);
    t[p].rs=build(mid+1,r);
    return p;
}
int upd_fa(int l,int r,int last,int x,int fa){
    int p=++tot;
    t[p]=t[last];
    if(l==r){t[p].fa=fa;return p;}
    int mid=l+r>>1;
    if(x<=mid)t[p].ls=upd_fa(l,mid,t[last].ls,x,fa);
    else t[p].rs=upd_fa(mid+1,r,t[last].rs,x,fa);
    return p;
}
void upd_dep(int l,int r,int p,int x){
    if(l==r){++t[p].d;return;}
    int mid=l+r>>1;
    if(x<=mid)upd_dep(l,mid,t[p].ls,x);
    else upd_dep(mid+1,r,t[p].rs,x);
}
int qry(int l,int r,int x,int p){
    if(l==r)return p;
    int mid=l+r>>1;
    if(x<=mid)return qry(l,mid,x,t[p].ls);
    else return qry(mid+1,r,x,t[p].rs);
}
int find(int x,int p){
    int id=qry(1,n,x,p);
    int f=t[id].fa;
    return x==f?id:find(f,p);
}
void merge(int x,int y,int &p){
    int fxid=find(x,p);
    int fyid=find(y,p);
    int fx=t[fxid].fa,fy=t[fyid].fa;
    int dx=t[fxid].d,dy=t[fyid].d;
    if(dx>dy)swap(fx,fy);
    p=upd_fa(1,n,p,fx,fy);
    if(dx==dy)upd_dep(1,n,p,fy);
```

```
}
int main(){rt[0]=build(1,n);}
```

5.8 主席树

区间第 k 小

```
int n,q,a[N],b[N],rt[N],tot;
struct sgt{
    int v;
    int ls,rs;
}t[N*33];//nlogC ,离散化后C=n
int insert(int l,int r,int last,int x,int c){
    int p=++tot;
    t[p]=t[last];
    t[p].v+=c;
    if(l!=r){
        int mid=l+r>>1;
        if(x<=mid)t[p].ls=insert(l,mid,t[p].ls,x,c);
        else t[p].rs=insert(mid+1,r,t[p].rs,x,c);
    }
    return p;
}
int qry(int l,int r,int p1,int p2,int k){
    if(l==r)return l;
    int s=t[t[p2].ls].v-t[t[p1].ls].v;
    int mid=l+r>>1;
    if(k<=s)return qry(l,mid,t[p1].ls,t[p2].ls,k);
    else return qry(mid+1,r,t[p1].rs,t[p2].rs,k-s);
}// qry(1,n,rt[l-1],rt[r],k) = [L,R]区间第k小
```

区间 mex

```
int n,q,a[N];
struct sgt{
    int ls,rs,v;//权值区间最后一次出现位置的最小值
}t[N*33];//nlogC
int rt[N],tot;
int insert(int l,int r,int last,int x,int pos){
    int p=++tot;
    t[p]=t[last];
    if(l==r){
        t[p].v=pos;return p;
```

```
    }
    int mid=l+r>>1;
    if(x<=mid)t[p].ls=insert(l,mid,t[last].ls,x,pos);
    else t[p].rs=insert(mid+1,r,t[last].rs,x,pos);
    t[p].v=min(t[t[p].ls].v,t[t[p].rs].v);
    return p;
}
int qry(int l,int r,int lim,int p){//qry(0,n,l,rt[r])=区间[l,r]的mex
    if(l==r)return l;
    int mid=l+r>>1;
    if(t[t[p].ls].v<lim)return qry(l,mid,lim,t[p].ls);
    else return qry(mid+1,r,lim,t[p].rs);
}
```

区间 lcm：

对于一个数，将其分解质因数，若有因子 p^k，那么拆分出 k 个数 $p, p^2, ..., p^k$，权值都为 p，$lcm([l,r])=$ 区间权值之积

区间修改主席树：标记永久化，更新时可能会插入 $2logn$ 个结点，询问时把 tag 递归下传

```
struct sgt{ int ls,rs;ll v,tag;}t[N*33]; int rt[N],tot;
int newnode(){int p=++tot;t[p].ls=t[p].rs=t[p].v=t[p].tag=0;return p;}
ll calc(int p,int len,ll tag){
    if(!p)return 0;
    return t[p].v+(t[p].tag+tag)*len;
}
void pushup(int p,int len){
    t[p].v=calc(t[p].ls,len+1>>1,0)+calc(t[p].rs,len>>1,0);
}
int upd(int last,int l,int r,int L,int R,ll c){
    int p=newnode();
    t[p]=t[last];
    if(L<=l&&r<=R){ t[p].tag+=c;return p;}
    int mid=l+r>>1;
    if(L<=mid)t[p].ls=upd(t[last].ls,l,mid,L,R,c);
    if(R>mid)t[p].rs=upd(t[last].rs,mid+1,r,L,R,c);
    pushup(p,r-l+1);
    return p;
}
ll qry(int p,int l,int r,int L,int R,ll tag){
    if(!p)return 0;
    if(L<=l&&r<=R)return calc(p,r-l+1,tag);
    int mid=l+r>>1;  ll s=0;
```

```
    if(L<=mid)s=qry(t[p].ls,l,mid,L,R,tag+t[p].tag);
    if(R>mid)s+=qry(t[p].rs,mid+1,r,L,R,tag+t[p].tag);
    return s;
}
```

5.9 树套树

树状数组套动态开点线段树/01Trie 树

```
struct sgt{
    int ls,rs;
    int v;
}t[N*200];//nlognlogC
int rt[N],tot;
void upd(int l,int r,int &p,int x,int c){
    if(!p)p=++tot;
    t[p].v+=c;
    if(l==r)return;
    int mid=l+r>>1;
    if(x<=m)upd(l,mid,t[p].ls,x,c);
    else upd(mid+1,r,t[p].rs,x,c);
}
int qry(int L,int R,int l,int r,int p){
    if(!p)return 0;
    if(L<=l&&r<=R)return t[p].v;
    int mid=l+r>>1,s=0;
    if(L<=m)s+=qry(L,R,l,mid,t[p].ls);
    if(R>m)s+=qry(L,R,mid+1,r,t[p].rs);
    return s;
}
void add(int i,int x,int k){//第i棵线段树在x的位置+k
    for(;i<=n;i+=i&-i)upd(1,n,rt[i],x,k);
}
int query(int x,int l,int r){//询问[1,x]区间值域[l,r]的个数
    if(l>r)return 0;
    int s=0;
    for(;x;x-=x&-x)s+=qry(l,r,1,n,rt[x]);
    return s;
}
```

线段树套动态开点线段树

```
//考虑权值套区间，还是区间套权值
```

```
//外层线段树应当只进行单点更新,且不需要pushup
//内存线段树动态开点,且可以进行区间打标记
```

5.10 李超线段树

```
struct Line {
    ll k, b;
} lin[N];
int lcnt;
int add_line(ll k, ll b) {
    lin[++lcnt] = {k, b};
    return lcnt;
}
struct node {
    int ls, rs, u;
} tr[N << 2];
int tot;
ll calc(int u, ll x) {
    return lin[u].k * x + lin[u].b;
}
bool cmp(int u, int v, ll x) {
    return calc(u, x) <= calc(v, x); // 如果要求最大值，只需要修改为大于等于
}
void pushdown(int &p, int l, int r, int v) {
    if (!p) p = ++tot;
    if (l == r) return;
    int mid = (l + r) >> 1;
    int &u = tr[p].u, b = cmp(v, u, mid);
    if (b) swap(u, v);
    int bl = cmp(v, u, l), br = cmp(v, u, r);
    if (bl) pushdown(tr[p].ls, l, mid, v);
    if (br) pushdown(tr[p].rs, mid + 1, r, v);
}
void update(int &p, int l, int r, int L, int R, int v) {
    if (l > R || r < L) return;
    if (!p) p = ++tot;
    int mid = (l + r) >> 1;
    if (l >= L && r <= R) return pushdown(p, l, r, v), void();
    update(tr[p].ls, l, mid, L, R, v);
    update(tr[p].rs, mid + 1, r, L, R, v);
}
```

```
ll query(int p, int l, int r, ll pos) {
    if (!p) return 1e16;
    ll res = calc(tr[p].u, pos);
    int mid = (l + r) >> 1;
    if (l == r) return res;
    if (pos <= mid) {
        res = min(res, query(tr[p].ls, l, mid, pos));
    } else res = min(res, query(tr[p].rs, mid + 1, r, pos));
    return res;
}

int main() {
    lin[0].b = 1e16;
    return 0;
}
```

5.11 Trie 树

```
const int N=1e5+10,M=N*30;
int t[M][26],cnt[M],tot;//M为单词字母个数和
void insert(string &s){
    int p=0;
    for(int i=0;i<s.length();++i){
        int c=s[i]-'a';
        if(!t[p][c])t[p][c]=++tot;
        p=t[p][c];
    }
    ++cnt[p];
}
int qry(string &s){//s有几个
    int p=0;
    for(int i=0;i<s.length();++i){
        int c=s[i]-'a';
        if(!t[p][c])return 0;
        p=t[p][c];
    }
    return cnt[p];
}
```

5.12 可持久化 01Trie

```
//求max[l<=i<=r] a[i]^a[i+1]^...^a[n]^x,即求与(x^s[n])异或起来最大的s[i-1]
int n,q,a[N],s[N],rt[N];
int t[N*30][2],tot,lastvis[N*30];
int insert(int last,int i,int k){
    int p=++tot;
    lastvis[p]=i;
    if(k<0)return p;
    int c=s[i]>>k&1;
    t[p][c^1]=t[last][c^1];
    t[p][c]=insert(t[last][c],i,k-1);
    return p;
}
int qry(int l,int p,int x){//[l,r]上max(s[i]^x)
    for(int i=30;i>=0;--i){
        int c=x>>i&1;
        if(lastvis[t[p][c^1]]>=l)p=t[p][c^1];
        else p=t[p][c];
    }
    return s[lastvis[p]]^x;
}
void debug(int p,int k,string s){
    if(k<0){
        cout<<s<<endl;
        return;
    }
    if(t[p][0])debug(t[p][0],k-1,s+'0');
    if(t[p][1])debug(t[p][1],k-1,s+'1');
}
int main(){
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;++i)scanf("%d",&a[i]);
    for(int i=1;i<=n;++i)s[i]=s[i-1]^a[i];
    lastvis[0]=-1;//0代表不存在的结点,必须赋值为-1
    rt[0]=insert(0,0,30);//建立前缀和数组时s[0]存在,需要插入
    for(int i=1;i<=n;++i)rt[i]=insert(rt[i-1],i,30);
    char op[3];
    for(;q--;){
        scanf("%s",op);
        if(op[0]=='A'){//数组尾部插入一个数字
            int x;scanf("%d",&x);
            a[++n]=x;s[n]=s[n-1]^a[n];
```

```
            rt[n]=insert(rt[n-1],n,30);
        }
        else{
            int l,r,x;scanf("%d%d%d",&l,&r,&x);
            int ans=qry(l-1,rt[r-1],x^s[n]);
            printf("%d\n",ans);
        }
    }
}
```

5.13 莫队

单点修改莫队

```
int n,m,a[N],pos[N];//pos[i]=i所在块
struct QQ{ int l,r,id; }q[N];
inline bool cmp(QQ a,QQ b){//询问按块排序
    return pos[a.l]<pos[b.l]||pos[a.l]==pos[b.l]&&a.r<b.r;
}
ll cnt[M],ans[N],now;//cnt[i]=数字出现次数,now=当前答案
void add(int x){
    now+=cnt[s[x]]; ++cnt[s[x]];
}
void sub(int x){
    now-=cnt[s[x]]-1; --cnt[s[x]];
}
int main(){
    scanf("%d%d",&n,&m);int sz=sqrt(n);
    for(int i=1;i<=n;++i)scanf("%d",&a[i]),pos[i]=i/sz;
    for(int i=1;i<=m;++i){scanf("%d%d",&q[i].l,&q[i].r);q[i].id=i;}
    sort(q+1,q+1+m,cmp);
    int L=1,R=0;  now=0;
    for(int i=1;i<=m;++i){
        while(q[i].l<L)add(--L);
        while(q[i].r>R)add(++R);
        while(q[i].l>L)sub(L++);
        while(q[i].r<R)sub(R--);
        ans[q[i].id]=now;
    }
    for(int i=1;i<=m;++i)printf("%lld\n",ans[i]);
}
```

回滚莫队

```
const int maxn=3e5+25,INF=0x3f3f3f3f;
int block_size;
int ans[maxn],color[maxn],cntl[maxn],cntr[maxn],s[maxn];
int l=1,r=0,sum=0,n,m,mx;
struct query {
    int l,r,id;
    bool operator<(const query& sec)const {
        if(l/block_size!=sec.l/block_size)return l<sec.l;
        return r<sec.r;
    }
} q[maxn];
int tmpl[maxn], tmpr[maxn];
int __l[maxn], __r[maxn];
int main() {
    memset(tmpl,INF,sizeof(tmpl));
    memset(cntl,INF,sizeof(cntl));
    n=read();
    block_size=sqrt(n);
    for (int i = 1; i <= n; i++)s[i]=color[i]=read();
    sort(s+1,s+n+1);
    int sz=unique(s+1,s+n+1)-s-1;
    for(int i = 1; i <= n; i++)color[i]=lower_bound(s+1,s+1+sz,color[i])-s;
    m=read();
    for (int i = 1; i <= m; i++) {
        q[i].l=read();q[i].r=read();q[i].id=i;
    }
    sort(q+1,q+1+m);
    for (int i = 1; i <= m; i++) {
        int curmx = 0;
        if(q[i].l/block_size==q[i].r/block_size) { // l,r在通一块,暴力
            for (int j = q[i].l; j <= q[i].r; j++) {
                if (!__l[color[j]]) __l[color[j]] = j;
                __r[color[j]] = j;
                curmx = max(curmx, __r[color[j]] - __l[color[j]]);
            }
            for (int j = q[i].l; j <= q[i].r; j++) {
                __l[color[j]] = 0;
                __r[color[j]] = 0;
            }
            ans[q[i].id] = curmx;
```

```
            continue;
        }
        if((q[i].l/block_size+1)*block_size!=l) { // l在另一块
            mx=0;
            while(r>=l)cntl[color[r]]=0x3f3f3f3f,cntr[color[r]]=0,r--; // reset
            l=(q[i].l/block_size+1)*block_size;
            r=l-1;
        }
        while(r<q[i].r) {
            ++r;
            cntl[color[r]] = min(r,cntl[color[r]]);
            cntr[color[r]] = r;
            mx=max(mx,cntr[color[r]] - cntl[color[r]]);
        }
        curmx = mx;
        for (int j = q[i].l; j <= l -1; j++) {
            tmpl[color[j]] = min(j,tmpl[color[j]]);
            tmpr[color[j]] = max(j,tmpr[color[j]]);
            curmx=max(curmx,max(tmpr[color[j]], cntr[color[j]]) - min(tmpl[color[
               j]], cntl[color[j]]));
        }
        for (int j = q[i].l; j <= l -1; j++) {
            tmpl[color[j]] = 0x3f3f3f3f;
            tmpr[color[j]] = 0;
        }
        ans[q[i].id]=curmx;
    }
    for (int i = 1; i <= m; i++)printf("%d\n",ans[i]);
}
```

树上莫队

```
struct treenode {
    int fa,hson,sz,depth,top;
    vector<int>nxt;
} t[maxn];
struct Query {
    int l,r,id,version;
    int posl,posr;
    bool operator<(const Query& sec)const {
        if(posl!=sec.posl)return l<sec.l;
        if(posr!=sec.posr)return r<sec.r;
```

```
        return version<sec.version;
    }
} q[maxn];
int block_size,Q;
int color[maxn],cnt[maxn],ver[maxn],save[maxn],v[maxn],w[maxn],c[maxn],vis[maxn];
int in[maxn],out[maxn],id[maxn]; //tree_to_array
int l=1,r=0,n,m,version=0,tot=0,pos,newval,x,y,tmp1,tmp2,trans,stamp=1;
ll sum=0,ans[maxn];
void dfs1(int pos) {
    in[pos]=++stamp;
    id[stamp]=pos;
    t[pos].hson=-1; //heavyson shoule be -1
    t[pos].sz=1;
    for(int k:t[pos].nxt) {
        if(!t[k].depth) {
            t[k].depth=t[pos].depth+1;
            t[k].fa=pos;
            dfs1(k);
            t[pos].sz+=t[k].sz;
            if(t[pos].hson==-1||t[k].sz>t[t[pos].hson].sz) {
                t[pos].hson=k;
            }
        }
    }
    out[pos]=++stamp;
    id[stamp]=pos;
}
void dfs2(int pos,int top) {
    t[pos].top=top;
    if(t[pos].hson==-1)return;
    dfs2(t[pos].hson,top); //to visit heavy_son first
    for(int k:t[pos].nxt) {
        if(k!=t[pos].fa&&k!=t[pos].hson)dfs2(k,k);
    }
}
int query_LCA(int a,int b) {
    while(t[a].top!=t[b].top) {
        if(t[t[a].top].depth<t[t[b].top].depth)swap(a,b);
        a=t[t[a].top].fa;
    }
    return t[a].depth<t[b].depth?a:b;
```

```
}
void add(int cur) { //cur: the number of node, in xor, add and del are equivalent
    if(vis[cur]==0)sum+=1ll*v[color[cur]]*w[++cnt[color[cur]]];
    else sum-=1ll*v[color[cur]]*w[cnt[color[cur]]--];
    vis[cur]^=1;
}
void work(int cur) { //cur: the version
    int num=ver[cur],clr=save[cur]; // clr:color
    if((l<=in[num]&&in[num]<=r)^(l<=out[num]&&out[num]<=r)) {
        sum+=1ll*v[clr]*w[++cnt[clr]];
        sum-=1ll*v[color[num]]*w[cnt[color[num]]--];
    }
    swap(save[cur],color[num]);
}
int main() {
    n=read(),m=read(),Q=read();
    block_size=pow((n<<1),0.667);
    for(int i=1; i<=m; i++)v[i]=read();
    for(int i=1; i<=n; i++)w[i]=read();
    for(int i=0; i<n-1; i++) {
        tmp1=read(),tmp2=read();
        t[tmp1].nxt.push_back(tmp2);
        t[tmp2].nxt.push_back(tmp1);
    }
    t[1].depth=1; dfs1(1); dfs2(1,1);
    for(int i=1; i<=n; i++) color[i]=read();
    for(int i=0; i<Q; i++) {
        trans=read();
        if(trans==1) {
            tmp1=read(),tmp2=read();
            if(in[tmp1]>in[tmp2])swap(tmp1,tmp2);
            if(out[tmp1]<out[tmp2])q[tot].l=out[tmp1];
            else q[tot].l=in[tmp1];
            q[tot].r=in[tmp2];
            q[tot].posl=q[tot].l/block_size;
            q[tot].posr=q[tot].r/block_size;
            q[tot].id=tot;
            q[tot].version=version;
            tot++;
        } else {
            version++;
```

```
            ver[version]=read();
            save[version]=read();
        }
    }
    sort(q,q+tot);
    version=0;
    for(int i=0; i<tot; i++) {
        while(l>q[i].l)add(id[--l]);
        while(r<q[i].r)add(id[++r]);
        while(l<q[i].l)add(id[l++]);
        while(r>q[i].r)add(id[r--]);
        while(version<q[i].version)work(++version);
        while(version>q[i].version)work(version--);
        x=id[l],y=id[r];
        int f=query_LCA(x,y);
        if(f!=x&&f!=y) {
            add(f);
            ans[q[i].id]=sum;
            add(f);
        } else ans[q[i].id]=sum;
    }
    for(int i=0; i<tot; i++)printf("%lld\n",ans[i]);
}
```

5.14 平衡树

Treap

```
struct treap{
    int l,r;
    int v,k;
    int sz,cnt;
}t[N];
int rt,tot;
int New(int x){
    t[++tot].v=x;
    t[tot].k=rand();
    t[tot].sz=t[tot].cnt=1;
    t[tot].l=t[tot].r=0;
    return tot;
}
void upd(int p){
```

```
15      t[p].sz=t[t[p].l].sz+t[t[p].r].sz+t[p].cnt;
16  }
17  void build(){
18      srand(time(0));
19      New(-inf);New(inf);
20      rt=1;t[1].r=2;
21      upd(rt);
22  }
23  void zig(int &p){
24      int q=t[p].l;
25      t[p].l=t[q].r;
26      t[q].r=p;p=q;
27      upd(t[p].r);upd(p);
28  }
29  void zag(int &p){
30      int q=t[p].r;
31      t[p].r=t[q].l;
32      t[q].l=p;p=q;
33      upd(t[p].l);upd(p);
34  }
35  void insert(int &p,int x){
36      if(!p){
37          p=New(x);
38          return;
39      }
40      if(x==t[p].v){
41          ++t[p].cnt;
42          ++t[p].sz;
43          return;
44      }
45      if(x<t[p].v){
46          insert(t[p].l,x);
47          if(t[t[p].l].k>t[p].k)zig(p);
48      }
49      else{
50          insert(t[p].r,x);
51          if(t[t[p].r].k>t[p].k)zag(p);
52      }
53      upd(p);
54  }
55  void del(int &p,int x){
```

```
    if(!p)return;
    if(x==t[p].v){
        if(t[p].cnt>1){
            --t[p].cnt;
            --t[p].sz;
            return;
        }
        if(t[p].l||t[p].r){
            if(!t[p].r||t[t[p].l].k>t[t[p].r].k){
                zig(p);
                del(t[p].r,x);
            }
            else{
                zag(p);
                del(t[p].l,x);
            }
            upd(p);
        }
        else p=0;
        return;
    }
    if(x<t[p].v)del(t[p].l,x);
    else del(t[p].r,x);
    upd(p);
}
int getrk(int p,int x){
    if(!p)return 0;
    if(x==t[p].v)return t[t[p].l].sz+1;
    if(x<t[p].v)return getrk(t[p].l,x);
    return getrk(t[p].r,x)+t[t[p].l].sz+t[p].cnt;
}
int getv(int p,int rk){
    if(!p)return inf;
    if(t[t[p].l].sz>=rk)return getv(t[p].l,rk);
    if(t[t[p].l].sz+t[p].cnt>=rk)return t[p].v;
    return getv(t[p].r,rk-t[t[p].l].sz-t[p].cnt);
}
int getpre(int x){
    int p=rt,s=1;
    while(p){
        if(x==t[p].v){
```

```
97              if(t[p].l){
98                  p=t[p].l;
99                  while(t[p].r)p=t[p].r;
100                 s=p;
101             }
102             break;
103         }
104         if(t[p].v<x&&t[p].v>t[s].v)s=p;
105         if(x<t[p].v)p=t[p].l;
106         else p=t[p].r;
107     }
108     return t[s].v;
109 }
110 int getnext(int x){
111     int p=rt,s=2;
112     while(p){
113         if(x==t[p].v){
114             if(t[p].r){
115                 p=t[p].r;
116                 while(t[p].l)p=t[p].l;
117                 s=p;
118             }
119             break;
120         }
121         if(t[p].v>x&&t[p].v<t[s].v)s=p;
122         if(x<t[p].v)p=t[p].l;
123         else p=t[p].r;
124     }
125     return t[s].v;
126 }
127 int main(){
128     build();memset(t,0,sizeof t);//多测要清空
129     for(int n=rd();n;--n){
130         int op=rd(),x=rd();
131         if(op==1)insert(rt,x);
132         else if(op==2)del(rt,x);
133         else if(op==3)printf("%d\n",getrk(rt,x)-1);
134         //注意平衡树初始有-inf和inf,所以这里要+1/-1
135         else if(op==4)printf("%d\n",getv(rt,x+1));
136         else if(op==5)printf("%d\n",getpre(x));
137         else printf("%d\n",getnext(x));
```

```
    }
}
```

splay

```
const int N=5e5+10,M=4e6+N,inf=1<<30;
int n,a[N],rt,tot;
struct Splay{
    int v,tag;
    int sz;
    int c[2],f;
}t[M];
inline int New(int x){
    t[++tot].v=x;t[tot].sz=1;
    return tot;
}
inline void pushup(int p){
    t[p].sz=t[t[p].c[0]].sz+t[t[p].c[1]].sz+1;
}
inline void pushdown(int p){}
int build(int l,int r,int f){//建树
    if(l>r)return 0;
    int m=l+r>>1;
    int p=New(a[m]);
    //  if(a[m]!=-inf)idx[a[m]]=p;
    t[p].f=f;
    t[p].c[0]=build(l,m-1,p);
    t[p].c[1]=build(m+1,r,p);
    pushup(p);  return p;
}
void rotate(int x){
    int y=t[x].f,z=t[y].f;
    int c=t[y].c[1]==x;
    t[z].c[t[z].c[1]==y]=x;t[x].f=z;
    t[y].c[c]=t[x].c[c^1];t[t[x].c[c^1]].f=y;
    t[x].c[c^1]=y;t[y].f=x;
    pushup(y);
}
void splay(int x,int g){
    while(t[x].f!=g){
        int y=t[x].f,z=t[y].f;
        if(z!=g)rotate((t[y].c[1]==x)^(t[z].c[1]==y)?x:y);
```

```
38          rotate(x);
39      }
40      pushup(x);if(!g)rt=x;
41  }
42  int find(int k){
43      int p=rt;
44      while(1){
45          pushdown(p);
46          if(t[t[p].c[0]].sz>=k)p=t[p].c[0];
47          else if(t[t[p].c[0]].sz+1>=k)return p;
48          else{
49              k-=t[t[p].c[0]].sz+1;
50              p=t[p].c[1];
51          }
52      }
53  }
54  void gettree(int l,int r,int &x,int &y){//伸展[l,r]区间,t[y].c[0]即为区间的树根
55      x=find(l);y=find(r+2);
56      splay(x,0);splay(y,x);
57  }
58  void getpos(int pos,int &x,int &y){//在pos和pos+1之间的位置t[y].c[0]
59      x=find(pos+1);y=find(pos+2);
60      splay(x,0);splay(y,x);
61  }
62  void insert(int x,int y,int p){//已经getpos得到x,y之后,在y的左子树插入结点p(p为结
        点下标)
63      t[y].c[0]=p;t[p].f=y;
64      pushup(y);pushup(x);
65  }
66  int del(int l,int r){//摘下[l,r]区间的子树,返回子树的根结点
67      int x,y;
68      gettree(l,r,x,y);
69      int p=t[y].c[0];
70      t[y].c[0]=0;
71      t[p].f=0;
72      pushup(y);pushup(x);
73      return p;
74  }
75  int main(){
76      scanf("%d%d",&n,&q);
77      tot=0;memset(t,0,(n+5)*sizeof(Splay));
```

```
78      for(int i=1;i<=n;++i)scanf("%d",&a[i]);
79      a[0]=a[n+1]=-inf;
80      rt=build(0,n+1,0);
81  }
```

5.15 动态树

```
1   struct LCT{
2       int c[2],f,rtag;
3       int val,sum;//val=点权,sum=链和
4       int sz,vsz;//sz=子树和,vsz=虚儿子和
5   }t[N];
6   #define notroot(x) (t[t[x].f].c[0]==x||t[t[x].f].c[1]==x)
7   void pushup(int x){
8       t[x].sum=t[t[x].c[0]].sum^t[t[x].c[1]].sum^t[x].val;
9       t[x].sz=t[t[x].c[0]].sz+t[t[x].c[1]].sz+1+t[x].vsz;
10  }
11  void rev1(int x){
12      t[x].rtag^=1; swap(t[x].c[0],t[x].c[1]);
13  }
14  void pushdown(int x){
15      if(t[x].rtag){
16          t[x].rtag=0;
17          if(t[x].c[0])rev1(t[x].c[0]);
18          if(t[x].c[1])rev1(t[x].c[1]);
19      }
20  }
21  void rotate(int x){
22      int y=t[x].f,z=t[y].f;
23      int c=t[y].c[1]==x;
24      if(notroot(y))t[z].c[t[z].c[1]==y]=x;t[x].f=z;
25      t[y].c[c]=t[x].c[c^1];t[t[x].c[c^1]].f=y;
26      t[x].c[c^1]=y;t[y].f=x;
27      pushup(y);pushup(x);
28  }
29  void splay(int x){
30      static int sta[N],top;//注意修改此处的N
31      int y=x; sta[top=1]=y;
32      while(notroot(y))sta[++top]=y=t[y].f;
33      while(top)pushdown(sta[top--]);
34      while(notroot(x)){
```

```
35          int y=t[x].f,z=t[y].f;
36          if(notroot(y))rotate(t[y].c[1]==x^t[z].c[1]==y?x:y);
37          rotate(x);
38      }
39  }
40  void access(int x){
41      for(int f=0;x;f=x,x=t[x].f){
42          splay(x);
43          t[x].vsz+=t[t[x].c[1]].sz-t[f].sz;
44          t[x].c[1]=f; pushup(x);
45      }
46  }
47  void makeroot(int x){
48      access(x);splay(x);rev1(x);
49  }
50  int findroot(int x){
51      access(x);splay(x);
52      while(t[x].c[0])pushdown(x),x=t[x].c[0];
53      splay(x);return x;
54  }
55  void split(int x,int y){
56      makeroot(x);access(y);splay(y);
57  }
58  void link(int x,int y){
59      if(findroot(x)==findroot(y))return;
60      split(x,y); t[x].f=y;
61      t[y].vsz+=t[x].sz;
62      pushup(y);
63  }
64  void cut(int x,int y){
65      if(findroot(x)!=findroot(y))return;
66      split(x,y);
67      if(t[y].c[0]==x&&!t[x].c[1]){
68          t[y].c[0]=t[x].f=0;pushup(y);
69      }
70  }
71  void upd(int x,int c){
72      splay(x);t[x].val=c;pushup(x);
73  }
74  int qrychain(int x,int y){
75      split(x,y);return t[y].sum;
```

```
}
int qrysubtree(int rt,int x){
    split(rt,x);return t[x].vsz+1;
}
```

5.16 二叉堆

改变大/小根堆，需要改变四个注释处的比较符号，当前是小根堆

```
int n,heap[N<<2],tot;
#define ls (p<<1)
#define rs (p<<1|1)
#define fa (p>>1)
void up(){
    int p=tot;
    while(p!=1&&heap[fa]>heap[p]){//1
        swap(heap[fa],heap[p]);
        p=fa;
    }
}
void down(){
    int p=1;int s=ls;
    while(s<=tot){
        if(s<tot&&heap[s+1]<heap[s])++s;//2
        if(heap[s]<heap[p]){//3
            swap(heap[s],heap[p]);
            p=s;s=ls;
        }
        else break;
    }
}
void push(int x){
    heap[++tot]=x;
    int p=tot;
    while(p!=1&&heap[fa]>heap[p]){//4
        swap(heap[fa],heap[p]);
        p=fa;
    }
}
void pop(){
    heap[1]=heap[tot--];
    down();
```

```
}
```

5.17 左偏树 (可并堆)

可并堆

```
struct node{
    int ls,rs,dis;
    ll val,cnt;//权值,个数
}t[M];
int sta[M],top,htot;
int newnode(ll v,ll c){
    int p=top?sta[top--]:++htot;
    t[p]={0,0,1,v,c};
    return p;
}
int merge(int x,int y){
    if(!x||!y)return x+y;
    if(t[x].val>t[y].val)swap(x,y);
    t[x].rs=merge(t[x].rs,y);
    if(t[t[x].ls].dis<t[t[x].rs].dis)swap(t[x].ls,t[x].rs);
    t[x].dis=t[t[x].rs].dis+1;
    return x;
}
int pop(int x){
    sta[++top]=x;
    return merge(t[x].ls,t[x].rs);
}
void push(int &x,ll v,ll c){
    int p=newnode(v,c);
    x=merge(x,p);
}
//用法
push:push(root,v,c);
pop: --t[root].cnt;if(!t[root].cnt)root=pop(root);
top: t[root].val,t[root].cnt
```

带并查集功能

```
struct node{
    int ls,rs,fa;//fa用来维护并查集
    int val,dis;//val是关键字,dis是结点到最近不完整结点(至多一个儿子)的距离
}t[N];
```

```
int merge(int x,int y){//合并以x为根的堆,和以y为根的堆
    if(!x||!y)return x+y;
    if(t[x].val>t[y].val||t[x].val==t[y].val&&x>y)swap(x,y);
    t[x].rs=merge(t[x].rs,y);
    t[t[x].rs].fa=x;
    if(t[t[x].ls].dis<t[t[x].rs].dis)swap(t[x].ls,t[x].rs);
    t[x].dis=t[t[x].rs].dis+1;
    return x;
}
int find(int x){//找到x结点所在堆的堆顶结点下标
    return x==t[x].fa?x:t[x].fa=find(t[x].fa);
}
void pop(int x){//x为根的堆pop堆顶
    t[x].val=-1;//val=-1表示被删除
    t[t[x].ls].fa=t[x].ls;
    t[t[x].rs].fa=t[x].rs;
    t[x].fa=merge(t[x].ls,t[x].rs);
}
int main(){
    int n,q;scanf("%d%d",&n,&q);
    t[0].dis=-1;
    for(int i=1;i<=n;++i)scanf("%d",&t[i].val),t[i].fa=i;
    while(q--){
        int op,x,y;scanf("%d%d",&op,&x);
        if(op==1){//合并x和y结点所在的堆
            scanf("%d",&y);
            if(t[x].val==-1||t[y].val==-1)continue;
            int fx=find(x),fy=find(y);
            if(fx!=fy)t[fx].fa=t[fy].fa=merge(fx,fy);
        }
        else{//找到x结点所在的堆的堆顶并pop
            if(t[x].val==-1)puts("-1");//x结点已被删除
            else{
                int fx=find(x);
                printf("%d\n",t[fx].val);
                pop(fx);
            }
        }
    }
}
```

5.18 笛卡尔树

以下代码为小根堆

```
for(int i=1;i<=n;++i){
    while(top&&a[sta[top]]>a[i])t[i].ls=sta[top--];
    t[sta[top]].rs=i;
    sta[++top]=i;
}//单调栈建二叉树,入度为0是根
int root=sta[1];
```

6 图论

6.1 最小生成树

Kruskal

```
int n,m,fa[N],sum;
int find(int x){
    return x==fa[x]?x:fa[x]=find(fa[x]);
}
struct node{
    int x,y,z;
}b[M];
bool cmp(node a,node b){
    return a.z<b.z;
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;++i)fa[i]=i;
    for(int i=1;i<=m;++i)scanf("%d%d%d",&b[i].x,&b[i].y,&b[i].z);
    sort(b+1,b+1+m,cmp);
    for(int i=1;i<=m;++i){
        int x=b[i].x,y=b[i].y;
        if(find(x)!=find(y)){
            sum+=b[i].z;
            fa[find(x)]=find(y);
        }
    }
}
```

Prim

```
int n,m;
int e[N][N],dis[N];//e[i][j]:边i->j的长度,dis[i]连通图到i的最短边
bool vis[N];
inline void prim(){
    memset(dis,0x3f,n+1<<2);
    memset(vis+1,0,n);
    dis[1]=0;
    for(int i=1;i<n;++i){
        int x=0;
        for(int j=1;j<=n;++j)if(!vis[j]&&dis[j]<dis[x])x=j;
        vis[x]=1;
        for(int j=1;j<=n;++j)if(!vis[j])dis[j]=min(dis[j],e[x][j]);
    }
}
int main(){
    scanf("%d%d",&n,&m);
    memset(e,0x3f,sizeof e);
    for(int i=1;i<=n;++i)e[i][i]=0;
    for(int i=1;i<=m;++i){
        int x,y,z;scanf("%d%d%d",&x,&y,&z);
        e[y][x]=e[x][y]=min(e[x][y],z);
    }
    prim();int sum=0;
    for(int i=1;i<=n;++i)sum+=dis[i];
}
```

6.2 最短路

6.2.1 Floyd

```
int n,m,d[N][N];
int main(){
    scanf("%d%d",&n,&m);
    memset(d,0x3f,sizeof d);
    for(int i=1;i<=n;++i)d[i][i]=0;
    for(int i=1;i<=m;++i){
        int x,y,z;scanf("%d%d%d",&x,&y,&z);
        d[x][y]=min(d[x][y],z);
        d[y][x]=min(d[y][x],z);
    }
    for(int k=1;k<=n;++k)//先枚举中转点
```

```
        for(int i=1;i<=n;++i)
            for(int j=1;j<=n;++j)
                d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
}
```

6.2.2 Floyd 求最小环并计数

```
int n, m;
ll f[N][N], g[N][N], cnt, mi, dis[N][N];

void add(ll &x, ll y) {
    ((x += y) >= mod) && (x -= mod);
}
void upd(ll x, ll y) {
    if (x > mi) return;
    if (x == mi) add(cnt, y);
    else mi = x, cnt = y;
}

void solve() {
    mi = 1e15, cnt = 0;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            f[i][j] = dis[i][j] = 1e15;
            g[i][j] = 0;
        }
    }
    for (int i = 1; i <= m; i++) {
        int u, v;
        ll w;
        cin >> u >> v >> w;
        dis[u][v] = w;
        f[u][v] = w;
        g[u][v] = 1;
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i < k; i++) {
            for (int j = 1; j < i; j++) {
                upd(dis[i][k] + dis[k][j] + f[j][i], g[j][i]);
                upd(dis[k][i] + dis[j][k] + f[i][j], g[i][j]);
```

```
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (f[i][k] + f[k][j] == f[i][j]) {
                    add(g[i][j], g[i][k] * 1ll * g[k][j]);
                } else if (f[i][k] + f[k][j] < f[i][j]) {
                    f[i][j] = f[i][k] + f[k][j];
                    g[i][j] = g[i][k] * 1ll * g[k][j] % mod;
                }
            }
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++) upd(dis[i][j] + dis[j][i], 1);
    }
    if (mi == 1e15) cout << "-1 -1\n";
    else cout << mi << ' ' << cnt << '\n';
}
```

6.2.3 Dijkstra

注意二维费用不能有 0 边，需要先对第一维跑最短路，得到最短路径，用第二维更新时必须在第一维的路径上

```
priority_queue<pair<ll,int> >q;
ll d[N];bool vis[N];
void dij(int s){
    memset(d,0x3f,sizeof d);
    memset(vis,0,sizeof vis);
    d[s]=0;q.push(make_pair(0,s));
    for(;!q.empty();){
        int x=q.top().second;
        q.pop();
        if(vis[x])continue;
        vis[x]=1;
        for(int i=hd[x];i;i=e[i].nx){
            int y=e[i].to;
            if(d[y]>d[x]+e[i].v){
                d[y]=d[x]+e[i].v;
                q.push(make_pair(-d[y],y));
            }
```

```
        }
    }
}
```

6.2.4 SPFA

按点判断负环

```
bool spfa(int s){
    memset(dis,0x3f,sizeof dis);
    //push_time和in_queue要清空
    dis[s]=0;in_queue[s]=1;
    queue<int>q({s});
    while(q.size()){
        int x=q.front();q.pop();
        in_queue[x]=0;
        for(int i=hd[x];i;i=e[i].nx){
            int y=e[i].to;
            if(dis[x]+e[i].v<dis[y]){
                dis[y]=dis[x]+e[i].v;
                if(!in_queue[y]){
                    if(push_time[y]>n)return 1;//有负环
                    q.push(y);in_queue[y]=1;++push_time[y];
                    //if(dis[q.back()]<dis[q.front()])swap(q.back(),q.front());
                }
            }
        }
    }
    return 0;//没有负环
}
```

按边判断负环

```
bool in_queue[N];int edge_cnt[N];
bool spfa(int s){
    memset(dis,0x3f,sizeof dis);
    dis[s]=0;in_queue[s]=1;
    queue<int>q({s});
    while(q.size()){
        int x=q.front();q.pop();
        in_queue[x]=0;
        for(int i=hd[x];i;i=e[i].nx){
            int y=e[i].to;
```

```
            if(dis[x]+e[i].v<dis[y]){
                dis[y]=dis[x]+e[i].v;
                edge_cnt[y]=edge_cnt[x]+1;
                if(edge_cnt[y]>=n+1)return 1;//有负环
                if(!in_queue[y]){
                    q.push(y);
                    in_queue[y]=1;
                }
            }
        }
    }
    return 0;//没有负环
}
```

6.3 二分图最大匹配

```
//二分图两个部分的点编号分别为1到n和1到m,边(i,j)表示左边的i到右边的j
vector<int>e[N];
int n,m,t,match[N],v[N],ans;
bool dfs(int x,int t){
    if(v[x]==t)return 0;
    v[x]=t;
    for(auto y:e[x]){
        if(!match[y]||dfs(match[y],t)){
            match[y]=x;
            return 1;
        }
    }
    return 0;
}
int main(){
    scanf("%d%d%d",&n,&m,&t);
    for(int i=1;i<=t;++i){
        int x,y;scanf("%d%d",&x,&y);
        e[x].push_back(y);
    }
    for(int i=1;i<=n;++i)if(dfs(i,i))++ans;
    printf("%d\n",ans);
}
```

最小路径覆盖 = 最大独立集 = 总节点数 − 二分图最大匹配数

最小点覆盖 = 二分图最大匹配数

最大独立集 + 最小点覆盖集 = V ，最大团 = 补图的最大独立集。

6.4 拓扑排序

```
int n,in[N],rk[N];
vector<int>e[N];
int main(){
    queue<int>q;
    for(int i=1;i<=n;++i)if(!in[i])q.push(i);
    while(q.size()){
        int x=q.front();q.pop();
        for(auto y:e[x]){
            rk[y]=max(rk[y],rk[x]+1);//注意取max
            if(!--in[y])q.push(y);
        }
    }
}
```

6.5 Tarjan

6.5.1 有向图 scc

```
struct edge{
    int to,nx;
}e[M],e1[M];
int hd[N],hd1[N],tot,tot1;
void add(int x,int y){
    e[++tot].to=y;e[tot].nx=hd[x];hd[x]=tot;
}
void add1(int x,int y){
    e1[++tot1].to=y;e1[tot1].nx=hd1[x];hd1[x]=tot1;
}
int n,m,a[N];//a[i]点权
int dfn[N],low[N],sta[N];
bool vis[N];//在栈内
int idx[N],tscc;//i号点所在的scc的下标
int cnt[N];//SCC中点的个数
ll sum[N];//SCC点权和
void tarjan(int x){
    low[x]=dfn[x]=++dfn[0];
```

```
19      vis[x]=1;sta[++sta[0]]=x;
20      for(int i=hd[x];i;i=e[i].nx){
21          int y=e[i].to;
22          if(!dfn[y]){
23              tarjan(y);
24              low[x]=min(low[x],low[y]);
25          }
26          else if(vis[y]){
27              low[x]=min(low[x],dfn[y]);
28          }
29      }
30      if(low[x]==dfn[x]){
31          int y;++tscc;
32          do{
33              y=sta[sta[0]--];vis[y]=0;
34              //接下来添加SCC信息
35              ++cnt[tscc];sum[tscc]+=a[y];
36              idx[y]=tscc;
37          }while(y!=x);
38      }
39  }
40  void init(){
41      memset(vis+1,0,n);
42      memset(hd+1,0,n<<2);
43      memset(hd1+1,0,n<<2);
44      memset(low+1,0,n<<2);
45      memset(dfn+1,0,n<<2);
46      tot=tot1=dfn[0]=sta[0]=tscc=0;
47      //接下来添加初始化
48      memset(cnt+1,0,n<<2);
49      memset(sum+1,0,n<<3);
50  }
51  void getscc(){
52      for(int i=1;i<=n;++i)if(!dfn[i])tarjan(i);
53      for(int x=1;x<=n;++x){
54          for(int i=hd[x];i;i=e[i].nx){
55              int y=e[i].to;
56              if(idx[x]!=idx[y])add1(idx[x],idx[y]);//不属于同一SCC就连边
57          }
58      }
59  }
```

```
int main(){
    n=rd();m=rd();
    init();//初始化要在输入n之后
    for(int i=1;i<=m;++i){
        int x=rd(),y=rd();  add(x,y);
    }
    getscc();
    //缩点后的图用hd1[],e1[]表示
}
```

6.5.2 无向图 edcc(割边)

```
struct edge{
    int to,nx;
}e[M<<1],e1[M<<1];
int n,m,hd[N],tot,hd1[N],tot1;
void add(int x,int y){
    e[++tot].to=y;e[tot].nx=hd[x];hd[x]=tot;
}
void add1(int x,int y){//缩点后的图
    e1[++tot1].to=y;e1[tot1].nx=hd1[x];hd1[x]=tot1;
}
int a[N];//点权
int low[N],dfn[N];
int idx[N],tedcc,cnt[N];//idx[x]=x所属edcc,cnt[i]=第i个edcc里有几个点
ll sum[N];//sum[i]=第i个edcc里的点权之和
bool bridge[M<<1];//割边
void tarjan(int x,int eg){//eg=边下标
    low[x]=dfn[x]=++dfn[0];
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(!dfn[y]){
            tarjan(y,i);
            low[x]=min(low[x],low[y]);
            if(dfn[x]<low[y])bridge[i]=bridge[i^1]=1;//是割边
        }
        else if(i!=(eg^1))low[x]=min(low[x],dfn[y]);//(eg^1)一定要加括号
    }
}
void getedcc(int x){//缩点
    //接下来添加edcc信息
```

```
30      idx[x]=tedcc;sum[tedcc]+=a[x];++cnt[tedcc];
31      for(int i=hd[x];i;i=e[i].nx){
32          int y=e[i].to;
33          if(idx[y]||bridge[i])continue;//走过||是割边的话不走
34          getedcc(y);
35      }
36  }
37  void getgraph(){//求出缩点后的图
38      for(int i=1;i<=n;++i)if(!dfn[i])tarjan(i,0);
39      for(int i=1;i<=n;++i)if(!idx[i])++tedcc,getedcc(i);
40      for(int i=2;i<=tot;i+=2){//枚举所有边
41          int x=e[i].to,y=e[i^1].to;
42          if(idx[x]!=idx[y]){//所在edcc不同
43              add1(idx[x],idx[y]);
44              add1(idx[y],idx[x]);
45          }
46      }
47  }
48  void init(){
49      tot=tot1=1;tedcc=dfn[0]=0;
50      memset(hd+1,0,n<<2);
51      memset(hd1+1,0,n<<2);
52      memset(low+1,0,n<<2);
53      memset(dfn+1,0,n<<2);
54      memset(bridge+1,0,m<<1);
55      memset(idx+1,0,n<<2);
56      //接下来添加初始化
57      memset(cnt+1,0,n<<2);
58      memset(sum+1,0,n<<3);
59  }
60  int main(){
61      n=rd();m=rd();
62      init();//初始化要在输入n之后
63      for(int i=1;i<=n;++i)a[i]=rd();
64      for(int i=1;i<=m;++i){
65          int x=rd(),y=rd();
66          if(x==y)continue;
67          add(x,y);add(y,x);
68      }
69      getgraph();
70      //缩点后的图用hd1[],e1[]表示
```

```
}
```

6.5.3 无向图 vdcc(割点)

```
const int N=1e5+10,N1=N<<1;
struct edge{
    int to,nx;
}e[N<<1],e1[N1<<1];
int n,m,a[N],hd[N],tot,hd1[N<<1],tot1;
void add(int x,int y){
    e[++tot].to=y;e[tot].nx=hd[x];hd[x]=tot;
}
void add1(int x,int y){
    e1[++tot1].to=y;e1[tot1].nx=hd1[x];hd1[x]=tot1;
}
int rt,low[N],dfn[N],sta[N],tvdcc;
vector<int>vdcc[N];
bool cut[N];//是割点
void tarjan(int x){
    low[x]=dfn[x]=++dfn[0];
    sta[++sta[0]]=x;
    int flag=0;
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(!dfn[y]){
            tarjan(y);
            low[x]=min(low[x],low[y]);
            if(dfn[x]<=low[y]){
                ++flag;
                if(x!=rt||flag>1)cut[x]=1;
                vdcc[++tvdcc].push_back(x);
                int z;
                do{
                    z=sta[sta[0]--];
                    vdcc[tvdcc].push_back(z);
                }while(z!=y);
            }
        }
        else low[x]=min(low[x],dfn[y]);
    }
}
```

```
void get_round_square_tree(){
    for(int i=1;i<=n;++i)if(!dfn[i])rt=i,tarjan(i);
    for(int i=1;i<=tvdcc;++i){
        for(auto x:vdcc[i]){
            add1(i+n,x);add1(x,i+n);
            //添加方点信息
        }
    }
}
void init(){
    memset(dfn+1,0,n<<2);
    memset(low+1,0,n<<2);
    memset(hd+1,0,n<<2);
    memset(hd1+1,0,n<<3);
    dfn[0]=sta[0]=tvdcc=0;tot=tot1=1;
    for(int i=1;i<=n;++i)vdcc[i].clear();
    //接下来添加初始化
    memset(cut+1,0,n);
}
int main(){
    n=rd();m=rd();
    init();//初始化要在读入n之后
    for(int i=1;i<=m;++i){
        int x=rd(),y=rd();
        if(x==y)continue;
        add(x,y);add(y,x);
    }
    get_round_square_tree();
    //1~n是圆点下标,n+1 ~ n+tvdcc是方点下标,圆方树存在hd1[],e1[]
    //注意圆方树节点数最多为2N
}
```

6.5.4 仙人掌建圆方树

```
const int N=1e4+5,N1=N<<1,M=2e5+5;
struct edge{
    int to,nx,v;
}e[M<<1],e1[N1<<1];
int hd[N],tot,hd1[N1],tot1;
void add(int x,int y,int z){
    e[++tot].to=y;e[tot].v=z;
```

```
    e[tot].nx=hd[x];hd[x]=tot;
}
void add1(int x,int y,int z){
    e1[++tot1].to=y;e1[tot1].v=z;
    e1[tot1].nx=hd1[x];hd1[x]=tot1;
}
int n,m,q,vercnt;
int low[N],dfn[N],tdfn,fa[N],b[N],sum[N];
void tarjan(int x,int f){
    low[x]=dfn[x]=++tdfn;
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(y==f)continue;
        int z=e[i].v;
        if(!dfn[y]){
            fa[y]=x;b[y]=z;
            tarjan(y,x);
            low[x]=min(low[x],low[y]);
        }
        else
        low[x]=min(low[x],dfn[y]);
        if(dfn[x]>=low[y])continue;
        add1(x,y,z);add1(y,x,z);
    }
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(fa[y]==x||dfn[x]>=dfn[y])continue;
        int s=e[i].v;
        for(int u=y;u!=fa[x];u=fa[u]){
            sum[u]=s;s+=b[u];
        }
        sum[++vercnt]=sum[x];sum[x]=0;
        for(int u=y;u!=fa[x];u=fa[u]){
            int w=min(sum[u],sum[vercnt]-sum[u]);
            add1(vercnt,u,w);add1(u,vercnt,w);
        }
    }
}
int dis[N1],son[N1],d[N1],top[N1],dfn1[N1],sz[N1],ff[N1];
void dfs1(int x,int f){
    sz[x]=1;
```

```
49      for(int i=hd1[x];i;i=e1[i].nx){
50          int y=e1[i].to;
51          if(y==f)continue;
52          ff[y]=x;
53          d[y]=d[x]+1;
54          dis[y]=dis[x]+e1[i].v;
55          dfs1(y,x);
56          sz[x]+=sz[y];
57          if(sz[y]>sz[son[x]])son[x]=y;
58      }
59  }
60  void dfs2(int x,int t){
61      dfn1[x]=++dfn1[0];top[x]=t;
62      if(!son[x])return;
63      dfs2(son[x],t);
64      for(int i=hd1[x];i;i=e1[i].nx){
65          int y=e1[i].to;
66          if(!dfn1[y])dfs2(y,y);
67      }
68  }
69  int lca(int x,int y){
70      while(top[x]!=top[y]){
71          if(d[top[x]]<d[top[y]])swap(x,y);
72          x=ff[top[x]];
73      }
74      return d[x]<d[y]?x:y;
75  }
76  int find(int x,int f){//方点lca下的圆点
77      int s;
78      while(top[x]!=top[f]){
79          s=top[x];x=ff[top[x]];
80      }
81      return x==f?s:son[f];
82  }
83  int main(){
84      scanf("%d%d%d",&n,&m,&q);
85      vercnt=n;
86      for(int i=1;i<=m;++i){
87          int x,y,z;
88          scanf("%d%d%d",&x,&y,&z);
89          add(x,y,z);add(y,x,z);
```

```
    }
    tarjan(1,0);
    dfs1(1,0);dfs2(1,1);
    for(;q--;){
        int x,y;
        scanf("%d%d",&x,&y);
        int f=lca(x,y);
        int ans;
        if(f<=n)ans=dis[x]+dis[y]-2*dis[f];
        else{
            int u=find(x,f),v=find(y,f);
            ans=dis[x]+dis[y]-dis[u]-dis[v];
            if(sum[u]<sum[v])swap(sum[u],sum[v]);
            ans+=min(sum[u]-sum[v],sum[f]-(sum[u]-sum[v]));
        }
        printf("%d\n",ans);
    }
}
```

6.5.5 广义圆方树

```
int stk[N], n, m, top, cnt, low[N], dfn[N], dfc;
bool vis[N];
vector<int> G[N], T[N];

void tarjan(int u) {
    stk[++top] = u;
    low[u] = dfn[u] = ++dfc;
    for (int v : G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
            if (low[v] == dfn[u]) {
                cnt++;
                for (int x = 0; x != v; --top) {
                    x = stk[top];
                    T[cnt].push_back(x);
                    T[x].push_back(cnt);
                    val[cnt]++;
                }
                T[cnt].push_back(u);
```

```
                T[u].push_back(cnt);
                val[cnt]++;
            }
        } else low[u] = min(low[u], dfn[v]);
    }
}
int main() {
    cnt = n;
    for (int i = 1; i <= n; i++) if (!dfn[i]) {
        tarjan(i);
        --top;
    }
}
```

6.6 差分约束

$$\begin{cases} x_{a_1} - x_{b_2} \le dis_1 \\ x_{a_2} - x_{b_2} \le dis_2 \\ x_{a_n} - x_{b_n} \le dis_n \end{cases}$$

$x_i - x_j \le dis$ 相当于一条 $x_j \to x_i$ 距离为 dis 的边

$a \to b$ 的最短路即为 $x_b - x_a$ 的最大值

要求一组解，可以用超级起点连接所有点 (长为 0, 用于处理部分点相互独立的情况)，以超级起点为源点跑 SPFA，$x_i = dis[i]$ 即为满足条件的一组解

若存在负环, 相当于 $x_i - x_i < 0$，即无解

下面的程序用于求出任意一个满足所有不等式的解

```
int n,m,hd[N],tot;
struct edge{
    int to,nx,v;
}e[N<<1];
inline void add(int x,int y,int z){
    e[++tot]={y,hd[x],z};hd[x]=tot;
}
bool inq[N];//在队列内
int dis[N],push_time[N];
bool spfa(int s){
    queue<int>q({s});
    memset(dis,0x3f,sizeof dis);
```

```
    dis[s]=0;inq[s]=1;
    while(q.size()){
        int x=q.front();q.pop();
        inq[x]=0;
        for(int i=hd[x];i;i=e[i].nx){
            int y=e[i].to;
            if(dis[x]+e[i].v<dis[y]){
                dis[y]=dis[x]+e[i].v;
                if(!inq[y]){
                    inq[y]=1;
                    ++push_time[y];
                    if(push_time[y]>n)return 1; //有负环,无解
                    q.push(y);
                }
            }
        }
    }
    return 0;//这里不要漏写
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;++i)add(0,i,0);//超级源点,连通整张图
    for(int i=1;i<=m;++i){
        int x,y,z;scanf("%d%d%d",&x,&y,&z);
        add(y,x,z);
    }
    if(spfa(0))puts("NO");//有负环
    else for(int i=1;i<=n;++i)printf("%d ",dis[i]);
}
```

6.7 2-SAT

有 n 个布尔变量 $x_1, x_2, ..., x_n$，有多组限制条件 $x_i \to x_j$，是否存在满足所有条件的解

每个变量在图上被抽象为 2 个点，i 点表示 $x_i = 1$，$i + n$ 点表示 $x_i = 0$

对于限制条件 $x_i \to x_j$，你需要加一条 $i \to j$ 的边，类似地，$x_i \to \neg x_j$ 需要加一条 $i \to j + n$ 的边

对于限制条件 $x_i \vee x_j$，根据等价式 $\neg p \vee q \iff p \to q$，可以转化为 $i + n \to j$ 和 $j + n \to i$

对于限制条件 $x_i = 1$，相当于 $\neg x_i \to x_i$，需要添加一条 $i + n \to i$ 的边

设 $idx[i] = i$ 所在强连通分量的编号，若 $\forall i : idx[i] \neq idx[i + n]$，则有解，否则无解

因为 $idx[i] = idx[i + n]$ 意义是 $x_i = 1$ 能推出 $x_i = 0$，且 $x_i = 0$ 能推出 $x_i = 1$，即 x_i 无解

Tarjan 判断有解/求一种方案

```
int n,m,dfn[N],low[N],sta[N],idx[N],tscc;
bool vis[N];//在栈内
void tarjan(int x){
    low[x]=dfn[x]=++dfn[0];
    vis[x]=1;sta[++sta[0]]=x;
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(!dfn[y]){
            tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(vis[y]){
            low[x]=min(low[x],dfn[y]);
        }
    }
    if(low[x]==dfn[x]){
        int y;++tscc;
        do{
            y=sta[sta[0]--];vis[y]=0;
            idx[y]=tscc;
        }while(y!=x);
    }
}
int main(){
    scanf("%d%d",&n,&m);
    for(;m--;){//对于约束条件x or y,需要加边!x->y和!y->x
        int x,y,a,b;//x=a or y=b (a,b=0或1)
        scanf("%d%d%d%d",&x,&a,&y,&b);
        if(a&&b)add(x+n,y),add(y+n,x);
        else if(a&&!b)add(y,x),add(x+n,y+n);
        else if(!a&&b)add(x,y),add(y+n,x+n);
        else add(x,y+n),add(y,x+n);
    }//x号结点x=1,x+n表示x=0
    for(int i=1;i<=n*2;++i)if(!dfn[i])tarjan(i);//注意n*2
    for(int i=1;i<=n;++i){//判断不合法
        if(idx[i]==idx[i+n]){
            puts("IMPOSSIBLE");
            return 0;
        }
    }
```

```
    puts("POSSIBLE");
    for(int i=1;i<=n;++i){//输出一种方案
        int x=0;
        if(idx[i]<idx[i+n])x=1;//用Tarjan算法需要选择scc编号小的
        printf("%d ",x);
    }
}
```

求出最小字典序

```
const int N=8005<<1,M=20005<<1;
struct edge{
    int to,nx;
}e[M];
int n,m,hd[N],tot;
void add(int x,int y){
    e[++tot]={y,hd[x]};hd[x]=tot;
}
int sta[N],top,choose[N];//choose=1:选, 2:不选, 0:没访问过
bool dfs(int x){
    if(choose[x]==1)return 1;
    if(choose[x]==2)return 0;
    choose[x]=1;choose[x^1]=2;
    sta[++top]=x;
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(!dfs(y))return 0;
    }
    return 1;
}
bool twosat(){
    memset(choose,0,n<<3);//2n个点清空
    for(int i=0;i<n<<1;++i){
        if(choose[i])continue;
        top=0;
        if(!dfs(i)){
            for(int j=1;j<=top;++j)choose[sta[j]]=choose[sta[j]^1]=0;
            if(!dfs(i^1))return 0;
        }
    }
    return 1;
}
```

```
int main(){
    while(scanf("%d%d",&n,&m)!=EOF){
        memset(hd,0,n<<3);tot=0;
        for(int i=1;i<=m;++i){
            int x,y;scanf("%d%d",&x,&y);
            --x;--y;
            add(x,y^1);add(y,x^1);
        }
        if(!twosat())puts("NIE");
        else{
            for(int i=0;i<n<<1;++i)
            if(choose[i]==1)printf("%d\n",i+1);
        }
    }
}
```

前缀优化建图

1. 当前点选择说明之前的前缀都未选择

之前的前缀选择说明当前点位被选择

2. 之前的前缀选择说明当前前缀被选择

当前前缀未选择说明之前前缀未选择

3. 当前点选择说明当前前缀选择

当前前缀未选择说明当前点未选择

6.8 欧拉回路

```
int n,m,del[N],in[N],out[N],sta[M],top;
vector<int>e[N];
void dfs(int x){
    for(int i=del[x];i<e[x].size();i=del[x]){
        del[x]=i+1;dfs(e[x][i]);
    }
    sta[++top]=x;
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
        int x,y;scanf("%d%d",&x,&y);
        e[x].push_back(y);++out[x];++in[y];
    }
    for(int i=1;i<=n;i++)sort(e[i].begin(),e[i].end());
```

```
    int S=1,c1=0,c2=0;bool flag=1;
    for(int i=1;i<=n;i++){
        if(in[i]!=out[i])flag=0;
        if(out[i]==in[i]+1)++c1,S=i;
        if(in[i]==out[i]+1)++c2;
    }
    if(!flag&&!(c2==c1&&c2==1)){puts("No");}
    else{dfs(S);while(top)printf("%d ",sta[top--]);}
}
```

6.9 最大流

```
const int N=5e4+10,M=1e5+10;
const ll inf=1ll<<60;
struct edge{
    int to,nx;ll v;
}e[M<<1];//注意有反边,边数要乘以2
int n,m,s,t,vercnt,d[N],now[N],hd[N],tot=1;
ll maxflow;  queue<int>q;
void add(int x,int y,ll z){//一次加好正向和反向的边
    e[++tot]={y,hd[x],z};hd[x]=tot;
    e[++tot]={x,hd[y],0};hd[y]=tot;
}
bool bfs(){
    for(;!q.empty();q.pop());
    q.push(s);
    memset(d,0,vercnt+1<<2);
    d[s]=1; now[s]=hd[s];
    while(q.size()){
        int x=q.front();
        q.pop();
        for(int i=hd[x];i;i=e[i].nx){
            int y=e[i].to;
            if(e[i].v&&!d[y]){
                q.push(y);
                now[y]=hd[y];
                d[y]=d[x]+1;
                if(y==t)return 1;
            }
        }
    }
```

```
    return 0;
}
ll dinic(int x,ll flow){
    if(x==t)return flow;
    int i;
    ll k,rest=flow;
    for(i=now[x];i&&rest;i=e[i].nx){
        int y=e[i].to;
        if(e[i].v&&d[y]==d[x]+1){
            k=dinic(y,min(rest,e[i].v));
            if(!k)d[y]=0;
            e[i].v-=k;
            e[i^1].v+=k;
            rest-=k;
        }
    }
    now[x]=i;
    return flow-rest;
}
void init(){
    tot=1;maxflow=0;
    memset(hd,0,vercnt+1<<2);
}
int main(){
    scanf("%d%d",&n,&m);
    s=0;//源点建议为0
    t=n+1;//汇点编号根据实际情况调整
    vercnt=t+1;//点的个数 要求所有点编号在[0,vercnt]范围
    init();//初始化tot=1
    //开始加边

    //加边结束
    ll flow=0;
    while(bfs()){
        while(flow=dinic(s,inf)){
            maxflow+=flow;
        }
    }
    printf("%lld\n",maxflow);
    return 0;
}
```

6.10 最小割

最大流 = 最小割 = 对偶图最短路

平面图：任意两条边的交点都在点上的无向图

平面图的对偶图：点边反转，边划分出的区域变成点，有公共边的区域代表的点直接连边，边权是这条边的流量

6.11 最小费用最大流

```
#include <bits/stdc++.h>
using i64 = long long;

const int N = 5000 + 5;
const int inf = 1e9;
int head[N], cur[N], ecnt, dis[N], s, t, n, m, mincost;
bool vis[N];
struct Edge {
    int nxt, v, flow, cap, w;
}e[100002];
void add_edge(int u, int v, int flow, int cap, int w) {
    e[ecnt] = {head[u], v, flow, cap, w}; head[u] = ecnt++;
    e[ecnt] = {head[v], u, flow, 0, -w}; head[v] = ecnt++;
}
bool spfa(int s, int t) {
    memset(vis, 0, sizeof vis);
    std::fill(dis + 1, dis + t + 1, inf);
    std::queue<int> q;
    q.push(s);
    dis[s] = 0;
    vis[s] = 1;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if (e[i].flow < e[i].cap && dis[u] + e[i].w < dis[v]) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) vis[v] = 1, q.push(v);
            }
        }
```

```
33      }
34      return dis[t] != inf;
35  }
36  int dfs(int u, int a) {
37      if (vis[u]) return 0;
38      if (u == t || !a) return a;
39      vis[u] = 1;
40      int b = 0;
41      int flow = 0, f;
42      for (int& i = cur[u]; i != -1; i = e[i].nxt) {
43          int v = e[i].v;
44          if (dis[u] + e[i].w == dis[v] && (f = dfs(v, std::min(a, e[i].cap - e[i].
                flow))) > 0) {
45              e[i].flow += f;
46              e[i ^ 1].flow -= f;
47              flow += f;
48              mincost += e[i].w * f;
49              a -= f;
50              if (!a) break;
51          }
52      }
53      vis[u] = 0;
54      return flow;
55  }
56
57  signed main() {
58      memset(head, -1, sizeof head);
59      //freopen("in.txt", "r", stdin);
60      scanf("%d %d %d %d", &n, &m, &s, &t);
61      for (int i = 1; i <= m; i++) {
62          int u, v, w, ww;
63          scanf("%d %d %d %d", &u, &v, &w, &ww);
64          add_edge(u, v, 0, w, ww);
65      }
66      int ans = 0;
67      while (spfa(s, t)) {
68          for (int i = 0; i <= n; i++) cur[i] = head[i];
69          ans += dfs(s, inf);
70      }
71      printf("%d %d\n", ans, mincost);
72      return 0;
```

```
}
```

6.12 线段树优化建图

idx = 0 为入树，idx = 1 为出树连完边后入树叶子节点向出树对应节点连边如果是区间-> 区间考虑建出虚点

```
struct Segment_Tree {
    int idx, rt;
    int mp[N];
    struct node {
        int ls, rs;
    } tr[N << 2];
    void build(int &p, int l, int r, int fr) {
        if (p == 0) {
            p = ++tot;
            if (fr != -1) {
                if (idx == 0) G[fr].push_back({p, 0});
                else G[p].push_back({fr, 0});
            }
        }
        if (l == r) {
            mp[l] = p;
            return;
        }
        int mid = (l + r) >> 1;
        build(tr[p].ls, l, mid, p);
        build(tr[p].rs, mid + 1, r, p);
    }
    void update(int p, int l, int r, int L, int R, int to, int val) {
        if (r < L || l > R) return;
        if (l >= L && r <= R) {
            if (idx == 0) G[to].push_back({p, val});
            else G[p].push_back({to, val});
            return;
        }
        int mid = (l + r) >> 1;
        update(tr[p].ls, l, mid, L, R, to, val);
        update(tr[p].rs, mid + 1, r, L, R, to, val);
    }
} tr[2];
```

6.13 Kruskal 重构树

```
int main(){
    vercnt=n;//1-n是原来的点,n+1~2n-1是新建的点,边抽象为点
    sort(b+1,b+1+m,cmp);//边按边权排序
    for(int i=1;i<=m;++i){
        int x=b[i].u,y=b[i].v;
        int x1=find(x),y1=find(y);
        if(x1!=y1){
            ++vercnt;
            h[vercnt]=b[i].h;//边的信息存在新建的点上
            add(vercnt,x1); add(vercnt,y1);
            fa[x1]=vercnt; fa[y1]=vercnt;
            dis[vercnt]=min(dis[x1],dis[y1]);
        }
    }
}
```

6.14 二分图最大权匹配 KM 算法

```
typedef long long ll;
const int N=505;
const ll inf=0x3f3f3f3f3f3f3f3f;
int match[N];//右边的i匹配左边的match[i]
ll e[N][N];//边,e[i][j]=左边的i匹配右边的j的边权
ll dbx[N],dby[N],slack[N];
int pre[N]; bool vis[N];
int n,m;
void bfs(int k) {
    int px,py=0,yy=0;ll d;
    memset(pre,0,sizeof pre);
    memset(slack,0x3f,sizeof slack);
    match[py]=k;
    do {
        px=match[py],d=inf,vis[py]=1;
        for(int i=1; i<=n; i++)
            if(vis[i]==0) {
                if(slack[i]>dbx[px]+dby[i]-e[px][i])
                slack[i]=dbx[px]+dby[i]-e[px][i],pre[i]=py;
                if(slack[i]<d) d=slack[i],yy=i;
            }
```

```
        for(int i=0; i<=n; i++)
            if(vis[i]) dbx[match[i]]-=d,dby[i]+=d;
            else slack[i]-=d;
        py=yy;
    } while(match[py]!=0);
    while(py) match[py]=match[pre[py]],py=pre[py];
}
void km() { //二分图最大权匹配算法
    memset(dbx,0,sizeof dbx);
    memset(dby,0,sizeof dby);
    memset(match,0,sizeof match);
    for(int i=1; i<=n; i++)
    memset(vis,0,sizeof vis),bfs(i);
}
int main() {
    memset(e,0xcf,sizeof e);//初始化为-inf
    scanf("%d%d",&n,&m);
    for(int i=1; i<=m; ++i) {
        int x,y,z; scanf("%d%d%d",&x,&y,&z);
        e[x][y]=z;//左边的x匹配右边的y,价值为z
    }
    km();     ll ans=0;//最大权匹配sum
    for(int i=1; i<=n; ++i)ans+=e[match[i]][i];
    printf("%lld\n",ans);
    for(int i=1; i<=n; ++i)printf("%d ",match[i]);
}
```

6.15 矩阵树定理

一、无向无环图

A 为邻接矩阵，$A[i][j] = i \to j$ 的边数

D 为度数矩阵，$D[i][i] = \sum_{j=1}^{n} A[i][j] = i$ 的度数，其他位置为 0

基尔霍夫矩阵 $K = D - A$, 令 $K' = K$ 的去掉第 k 行第 k 列 (k 任意) 的 $n-1$ 阶主子式

$det(K') =$ 该无向图生成树个数

特别地，完全图生成树个数是 n^{n-2}

二、加权

求所有生成树边权的乘积之和，需要把邻接矩阵中边的条数改为为边权和

度数矩阵改为 $D[i][i] = \sum_{j=1}^{n} A[i][j]$

三、有向图

对于有根外向树，需要把度数矩阵改为入度和，$D[i][i]=\sum_{j=1}^{n}A[j][i]$

对于有根内向树，需要把度数矩阵改为出度和，$D[i][i]=\sum_{j=1}^{n}A[i][j]$

类似地，求所有有向生成树边权的乘积之和，需要把邻接矩阵改为入/出边边权和

四、变形：边权和的和

求所有生成树边权和的和，给原先边权为 w 的边赋值为一次多项式 $wx+1$，多项式乘法对 x^2 取模，$\prod(w_ix+1)$ 的一次项系数即为 w_i 之和

```
struct P {
    ll x,y;//x是一次项系数, y是常数项
    P (ll x=0,ll y=0):x(x),y(y){}
    friend P operator + (const P &u, const P &v) {
        return P(add(u.x, v.x), add(u.y, v.y));
    }
    friend P operator - (const P &u, const P &v) {
        return P(add(u.x, mod - v.x), add(u.y, mod - v.y));
    }
    friend P operator * (const P &u, const P &v) {
        return P(add(mul(u.x, v.y), mul(u.y, v.x)), mul(u.y, v.y));
    }
    friend P operator / (const P &u, const P &v) {
        ll inv=qpow(v.y, mod-2);
        return P(add(mul(u.x, v.y), mod - mul(u.y, v.x)) * inv % mod * inv % mod,
            mul(u.y, inv));
    }
};
P g[N][N];
ll gauss(P g[N][N],int n){
    P res(0,1);
    for(int i=1;i<=n;++i) {
        int pos=-1;
        for(int j=i;j<=n;++j){
            if(g[j][i].y){
                pos=j;break;
            }
        }
        if(pos==-1)return 0;
        swap(g[i],g[pos]);
        if(pos!=i)res=res*P(0,mod-1);
```

```
31          res=res*g[i][i];
32          P inv=P(0,1)/g[i][i];
33          for(int j=i+1;j<=n;++j){
34              P t=g[j][i]*inv;
35              for(int k=n;k>=i;--k)g[j][k]=g[j][k]-t*g[i][k];
36          }
37      }
38      return res.x;
39  }
```

6.16 LGV 引理

G 是一个有限的带权有向无环图。每个顶点的度是有限的，不存在有向环。

起点 $A=\{a_1,a_2,...,a_n\}$，终点 $B=\{b_1,b_2,...,b_n\}$，每条边 e 有边权 w_e

对于一个有向路径 P，定义 $\omega(P)$ 为路径上所有边权的积。

对任意顶点 (a,b)，定义 $e(a,b)=\sum\limits_{P:a\to b}\omega(P)=$ 所有 $a\to b$ 的路径边权乘积之和

$$M=\begin{bmatrix} e(a_1,b_1) & e(a_1,b_2) & \cdots & e(a_1,b_n) \\ e(a_2,b_1) & e(a_2,b_2) & \cdots & e(a_2,b_n) \\ \vdots & \vdots & \ddots & \vdots \\ e(a_n,b_1) & e(a_n,b_2) & \cdots & e(a_n,b_n) \end{bmatrix}$$

一个排列 σ 对应 $a_i\to b_{\sigma(i)}$ 的一种路径方案

$inv(\sigma)=$ 排列 σ 的逆序对，$a_i\to b_{\sigma(i)}$ 的路径交点数至少为 $inv(\sigma)$

设 P_i 表示从 a_i 到 $b_{\sigma(i)}$ 的一条路径，$det(M)=\sum\limits_{P:a\to b}(-1)^{inv(\sigma(P))}\Pi_{i=1}^{n}\omega(P_i)$

即路径交点数至少为 0 个的边权乘积和，减去路径焦点数至少为 1 个的边权乘积和，加上路径交点数至少为 2 个的边权乘积和...

根据广义容斥定理，$det(M)=\sum\limits_{P:a\to b,P_i\cap P_j=\emptyset}\prod\limits_{i=1}^{n}\omega(P_i)$，即所有方案中所有路径边权积的积的和

7 树上问题

7.1 树上倍增

```
1  int d[N];//d[i]=i的深度,d[rt]=1,(不能为0)
2  int f[N][21],mx[N][21];
3  void dfs(int x,int fa){
4      for(int i=hd[x];i;i=e[i].nx){
5          int y=e[i].to;
6          if(y==fa)continue;
7          mx[y][0]=e[i].v;
```

```
8           d[y]=d[x]+1;
9           f[y][0]=x;
10          for(int j=1;j<=20;++j){
11              f[y][j]=f[f[y][j-1]][j-1];
12              mx[y][j]=max(mx[y][j-1],mx[f[y][j-1]][j-1]);
13          }
14          dfs(y,x);
15      }
16  }
17  int lca(int x,int y){
18      if(d[x]<d[y])swap(x,y);
19      for(int i=20;i>=0;--i){
20          if(d[f[x][i]]>=d[y])x=f[x][i];//注意令根的深度=1，否则会跳错
21      }
22      if(x==y)return x;
23      for(int i=20;i>=0;--i){
24          if(f[x][i]!=f[y][i]){
25              x=f[x][i];y=f[y][i];
26          }
27      }
28      return f[x][0];
29  }
30  int chainmax(int x,int y){
31      int s=0;
32      if(d[x]<d[y])swap(x,y);
33      for(int i=20;i>=0;--i){
34          if(d[f[x][i]]>=d[y])x=max(x,mx[x][i]),x=f[x][i];
35      }
36      if(x==y)return x;
37      for(int i=20;i>=0;--i){
38          if(f[x][i]!=f[y][i]){
39              s=max({s,mx[x][i],mx[y][i]});
40              x=f[x][i];y=f[y][i];
41          }
42      }
43      s=max({s,mx[x][0],mx[y][0]});
44      return f[x][0];
45  }
```

7.2 dfs 序

树上问题可以通过 dfs 序转化为序列问题

树的所有 dfs 序方案数为 $\prod_{i=1}^{n} cnt(son_i)!$，（感叹号表示阶乘），$cnt(son_i) = i$ 结点儿子个数

lca(一个点集)=lca(dfs 序最小的点,dfs 序最大的点)

树上从一个点出发，经过一个点集的所有点再回到起点，最短路径是按照 dfs 序走，每条边恰好走两次，set 中按 dfs 序排序点集 $v_1, v_2, ..., v_k$，令 $sum = dis(v_1, v_2) + dis(v_2, v_3) + ... + dis(v_k, v_1)$，树链并即为 $sum/2$

7.3 轻重链剖分

```
//假设线段树已经写好
int dfn[N],sz[N],son[N],fa[N],top[N],d[N],tdfn;
void dfs1(int x,int f){
    sz[x]=1;
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(y==f)continue;
        fa[y]=x;
        d[y]=d[x]+1;
        dfs1(y,x);
        sz[x]+=sz[y];
        if(sz[son[x]]>sz[y])son[x]=y;
    }
}
void dfs2(int x,int t){
    top[x]=t;dfn[x]=++tdfn;
    if(son[x])dfs2(son[x],t);
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(dfn[y])continue;//if(fa[x]==y||y==son[x])continue;
        dfs2(y,y);
    }
}
int lca(int x,int y){
    while(top[x]!=top[y]){
        if(d[top[x]]>d[top[y]])swap(x,y);
        x=fa[top[x]];
    }
    return d[x]<d[y]?x:y;
}
```

```
void updchain(int x,int y,int c){//更新树链,询问也类似
    while(top[x]!=top[y]){
        if(d[top[x]]<d[top[y]])swap(x,y);
        upd(dfn[top[x]],dfn[x],1,c);
        x=fa[top[x]];
    }
    if(d[x]>d[y])swap(x,y);
    upd(dfn[x],dfn[y],1,c);//若维护边的信息,最后需要改为:
                        //if(x!=y)upd(dfn[x]+1,dfn[y],1,c);
}
int main(){
    //输入点/边信息
    dfs1(1,0);dfs2(1,1);
    for(int i=1;i<=n;++i)v[dfn[i]]=a[i];
    build(1,n,1);
    upd(dfn[x],dfn[x]+sz[x]-1,1,c);//子树更新
}
```

带方向的树链询问

```
struct sgt{
    int l=0,r=0,len=0;//注意初始化，或者再qrychain中进行初始化
    int lnum=0,rnum=0;//两边的数字
    int i=0,li=0,ri=0;//LCIS(最长连续严格上升子段),左边开始的LCIS,右边开始的LCIS
    int d=0,ld=0,rd=0;//LCDS(最长连续严格下降子段),左边开始的LCDS,右边开始的LCDS
}t[N<<2];
int qrychain(int x,int y){//x->y有方向性的询问
    sgt sx,sy;//sx=深度递增以x为结尾的链信息，sy=深度递增以y为结尾的链信息
    while(top[x]!=top[y]){
        if(d[top[x]]>d[top[y]]){
            sgt s=qry(dfn[top[x]],dfn[x],1);
            sx=pushup(s,sx);
            x=fa[top[x]];
        }
        else{
            sgt s=qry(dfn[top[y]],dfn[y],1);
            sy=pushup(s,sy);
            y=fa[top[y]];
        }
    }
    if(d[x]>d[y]){
        sgt s=qry(dfn[y],dfn[x],1);
```

```
        sx=pushup(s,sx);
    }
    else{
        sgt s=qry(dfn[x],dfn[y],1);
        sy=pushup(s,sy);
    }
    int ans=max(sx.d,sy.i);
    if(sx.lnum<sy.lnum)ans=max(ans,sx.ld+sy.li);
    return ans;
}
```

7.4 欧拉序

```
const int N=1e6+6;//欧拉序长度 N=2*n+1
int n,q,rt,Log2[N];
int idx[N],euler[N],d[N],h;//euler[1~h]:欧拉序 idx[i]=i在欧拉序中第一次出现的下标
pair<int,int>st[21][N];
vector<int>e[N];
void dfs(int x){//获得欧拉序
    euler[++h]=x;idx[x]=h;
    for(int y:e[x]){
        if(d[y])continue;
        d[y]=d[x]+1;
        dfs(y);
        euler[++h]=x;
    }
}
pair<int,int> qry(int l,int r){
    int k=Log2[r-l+1];
    return min(st[k][l],st[k][r-(1<<k)+1]);
}
int lca(int x,int y){//x,y在欧拉序中第一次出现下标之间深度最小的点
    if(idx[x]>idx[y])swap(x,y);
    return qry(idx[x],idx[y]).second;
}
int main(){
    for(int i=2;i<N;++i)Log2[i]=Log2[i>>1]+1;
    d[rt]=1;dfs(rt);
    for(int i=1;i<=h;++i)st[0][i]={d[euler[i]],euler[i]};
    for(int i=1;1<<i<=h;++i){
        for(int j=1;j+(1<<i)-1<=h;++j){
```

```
            st[i][j]=min(st[i-1][j],st[i-1][j+(1<<i-1)]);
        }
    }  //int f=lca(x,y);
}
```

7.5 点分治

```
int n,rt,hd[N],tot;
struct edge{
    int to,nx,v;
}e[N<<1];
inline void add(int x,int y,int z){
    e[++tot]={y,hd[x],v};hd[x]=tot;
}
int maxson[N],sz[N],sum;
bool vis[N];
void getroot(int x,int fa){
    maxson[x]=0;sz[x]=1;//初始化最大子树大小以及size
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(vis[y]||y==fa)continue;
        getroot(y,x);
        sz[x]+=sz[y];
        maxson[x]=max(maxson[x],sz[y]);
    }
    maxson[x]=max(maxson[x],sum-sz[x]);
    if(!rt||maxson[x]<maxson[rt])rt=x;
}
int b[N],c[N];
void getsubtree(int x,int fa,ll len){//统计子树信息
    c[++c[0]]=len;
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(vis[y]||y==fa)continue;
        getsubtree(y,x,len+e[i].v);
    }
}
void calc(int x){
    b[b[0]=1]=0;//将根加入数组
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
```

```
35          if(vis[y])continue;
36          c[0]=0;//记录子树的数组清空
37          getsubtree(y,x,e[i].v);//枚举所有子树
38          //新的子树和已访问子树连成的所有链
39          for(int j=1;j<=c[0];++j){
40              for(int k=1;k<=b[0];++k){
41
42              }
43          }
44          //将新的子树加入数组
45          for(int j=1;j<=c[0];++j)b[++b[0]]=c[i];
46      }
47      for(int i=1;i<=b[0];++i){//删除子树的贡献
48      }
49  }
50  void divide(int x){
51      vis[x]=1;//分割该点
52      calc(x);//统计该重心的子树
53      for(int i=hd[x];i;i=e[i].nx){//枚举所有子树
54          int y=e[i].to;
55          if(vis[y])continue;
56          sum=sz[y];rt=0;//sum=子树大小,初始化重心
57          getroot(y,0);//找到子树的重心
58          divide(rt);//递归子树
59      }
60  }
61  int main(){
62      n=rd();
63      for(int i=1;i<n;++i){
64          int x=rd(),y=rd(),z=rd();
65          add(x,y,z);add(y,x,z);
66      }
67      sum=n;rt=0;
68      getroot(1,0);
69      divide(rt);
70  }
```

7.6 点分树

需要注意，点分树上的路径与原来的树完全没有关系。

```
1  #include <bits/stdc++.h>
```

```
using namespace std;
using ll = long long;

const int mod = 998244353;
const int N = 2e5 + 5;

int n, q, val[N], rt, siz[N], SUM, mx[N];
int faz[N];
vector<int> T[N];
bool vis[N];

// 假设树剖已经写好

void getrt(int u, int fa) {
    siz[u] = 1;
    mx[u] = 0;
    for (int v : t.G[u]) if (v != fa && !vis[v]) {
        getrt(v, u);
        siz[u] += siz[v];
        mx[u] = max(mx[u], siz[v]);
    }
    mx[u] = max(mx[u], SUM - siz[u]);
    if (mx[u] < mx[rt]) rt = u;
}
void Dfs(int u, int fa) {
    vis[u] = 1;
    for (int v : t.G[u]) if (v != fa && !vis[v]) {
        rt = 0;
        SUM = siz[v];
        getrt(v, rt);
        T[u].push_back(rt);
        faz[rt] = u;
        Dfs(rt, rt);
    }
}

// 假设动态开点权值线段树已经写好

void dfs(int u, const int now) { // 预处理
    add(rt1[now], 0, n, dist(faz[now], u), val[u]);
    add(rt0[now], 0, n, dist(u, now), val[u]);
```

```
    for (int v : T[u]) dfs(v, now);
}
void upd(int u, int d) {
    int x = u;
    while (true) {
        add(rt1[u], 0, n, dist(x, faz[u]), d);
        add(rt0[u], 0, n, dist(x, u), d);
        if (u == root) break;
        u = faz[u];
    }
}
int qry(int u, int k) {
    int ans = query(rt0[u], 0, n, 0, k), x = u;
    while (u != root) {
        if (k - dist(faz[u], x) >= 0) ans += query(rt0[faz[u]], 0, n, 0, k - dist
            (faz[u], x)) - query(rt1[u], 0, n, 0, k - dist(faz[u], x));
        u = faz[u];
    }
    return ans;
}
```

7.7 树上启发式合并

```
int sz[N],son[N],Son;
int cnt[N];//color i 出现的次数
ll ans[N],sum,maxcnt;
void dfs1(int x,int fa){
    int mx=0;sz[x]=1;
    for(auto y:e[x]){
        if(y==fa)continue;
        dfs1(y,x);
        sz[x]+=sz[y];
        if(sz[y]>mx){
            mx=sz[y];
            son[x]=y;
        }
    }
}
void add(int x,int fa){
    ++cnt[a[x]];
    if(cnt[a[x]]>maxcnt){
```

```
        maxcnt=cnt[a[x]];sum=a[x];
    }
    else if(cnt[a[x]]==maxcnt)
    sum+=a[x];
    for(auto y:e[x]){
        if(y==fa||y==Son)continue;
        add(y,x);
    }
}
void del(int x,int fa){
    --cnt[a[x]];
    for(auto y:e[x]){
        if(y==fa)continue;
        del(y,x);
    }
}
void dfs2(int x,int fa,bool f){
    for(auto y:e[x]){
        if(y==fa||y==son[x])continue;
        dfs2(y,x,1);
    }
    if(son[x])dfs2(son[x],x,0);
    Son=son[x];add(x,fa);
    ans[x]=sum;
    if(f){
        del(x,fa);sum=0;maxcnt=0;
    }
}
int main(){dfs1(1,0);dfs2(1,0,0);}
```

7.8 长链剖分

```
int f[N][20],son[N],d[N],maxd[N],top[N],dfn[N],len[N],highbit[N];
vector<int>Up[N],Down[N];
void dfs1(int x,int fa){
    maxd[x]=d[x]=d[fa]+1;f[x][0]=fa;
    for(int i=1;i<20;++i){
        if(!f[x][i-1])break;
        f[x][i]=f[f[x][i-1]][i-1];
    }
    for(int i=hd[x];i;i=e[i].nx){
```

```
        int y=e[i].to;
        if(y==fa)continue;
        dfs1(y,x);
        if(maxd[y]>maxd[son[x]])son[x]=y,maxd[x]=maxd[y];
    }
}
void dfs2(int x,int t){
    top[x]=t;len[x]=maxd[x]-d[top[x]]+1;
    dfn[x]=++dfn[0];
    if(son[x])dfs2(son[x],t);
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(y!=f[x][0]&&y!=son[x])dfs2(y,y);
    }
}
inline int find_kth_fa(int x,int k){
    if(k>d[x])return 0;if(k==0)return x;
    x=f[x][highbit[k]];k^=1<<highbit[k];
    if(!k)return x;
    if(d[x]-d[top[x]]==k)return top[x];
    if(d[x]-d[top[x]]>k)return Down[top[x]][d[x]-d[top[x]]-k-1];
    return Up[top[x]][k-d[x]+d[top[x]]-1];
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<n;++i){
        int x,y;scanf("%d%d",&x,&y);
        add(x,y);add(y,x);
    }
    dfs1(1,0);dfs2(1,1);
    for(int i=1;i<=n;++i){
        if(i==top[i]){
            int l=0,x=i;
            while(l<len[i]&&x)x=f[x][0],++l,Up[i].push_back(x);
            l=0,x=i;
            while(l<len[i])x=son[x],++l,Down[i].push_back(x);
        }
    }
    for(int i=1,mx=1;i<=n;++i){
        if(i>>mx&1)++mx;
        highbit[i]=mx-1;
```

```
    }
    find_kth_fa(x,k);//调用x的k级祖先
}
```

7.9 虚树

```
int h,b[N],root;ll dp[N];
bool vis[N];//该点是询问点
inline bool cmp(int a,int b){
    return dfn[a]<dfn[b];
}
int build_virtual_tree(int b[],int h){
    static int sta[N],top;
    tot1=0;
    sort(b+1,b+1+h,cmp);//按dfs序排序
    sta[top=1]=b[1];
    for(int i=2;i<=h;++i){
        int ff=lca(b[i],sta[top]);
        while(1){
            if(dfn[ff]>=dfn[sta[top-1]]){
                if(ff!=sta[top]){
                    add1(ff,sta[top]);
                    if(ff!=sta[top-1])sta[top]=ff;
                    else --top;
                }
                break;
            }
            else{
                add1(sta[top-1],sta[top]);
                --top;
            }
        }
        sta[++top]=b[i];
    }
    while(--top)add1(sta[top],sta[top+1]);
    return sta[1];//虚树的根节点(所有询问点的lca)
}
void dfs3(int x){//虚树上树形DP
    ll sum=0;
    for(int i=hd1[x];i;i=e1[i].nx){
        int y=e1[i].to;dfs3(y);sum+=dp[y];
```

```
    }
    if(vis[x])dp[x]=mn[x];//询问点和非询问点分别算
    else dp[x]=min(mn[x],sum);
}
void del_tree(int x){//清空虚树,记得在函数外tot1=0
    for(int i=hd1[x];i;i=e1[i].nx)del_tree(e1[i].to);
    hd1[x]=0;
}
int main(){
    //询问b[1~h]这些结点
    for(int i=1;i=h;++i)vis[b[i]]=1;
    root=build_virtial_tree(b,h);
    dfs3(root); ans=dp[root];
    del_tree(root);
    for(int i=1;i<=h;++i)vis[b[i]]=0;
}
```

7.10 树上游走

你在 x 点会等概率选择一条边并经过

$dp_x = x$ 走到 fa_x 需要的期望步数，$d_x = x$ 的度数，$sz_x = x$ 子树中点的个数

$dp_x = 1 + \sum_{y \in son_x} \frac{dp_y + dp_x}{d_x} \Rightarrow d_x dp_x = d_x + (d_x - 1)dp_x + \sum_{y \in son_x} dp_y \Rightarrow dp_x = d_x + \sum_{y \in son_x} dp_y$

子树内每条边被算 2 次，向父亲的边算一次，可得 $dp_x = 2 \times sz_x - 1$

8 动态规划

8.1 树形背包

1、具有树形先后关系的背包 (拿了根才能拿儿子)

```
void dfs(int x,int num){
    for(auto y:e[x]){
        for(int i=0;i<num;++i)dp[y][i]=dp[x][i]+a[y];
        dfs(y,num-1);
        for(int i=1;i<=num;++i)dp[x][i]=max(dp[x][i],dp[y][i-1]);
    }
}
```

2、没有先修关系，dp[x][y]=x 号子树中选了 y 个的价值

```
//假设预处理好子树size
void dfs(int x,int f){
    dp[x][0]=dp[x][1]=0;
    for(auto y:e[x]){
        if(y==f)continue;//枚举到size,时间复杂度为O(nm)
        for(int i=min(sz[x],k);i>=0;--i){//x子树选几个
            for(int j=0;j<=i&&j<=sz[y];++j){//y子树选几个
                dp[x][i]=max(dp[x][i],dp[y][j]+dp[x][i-j]);
            }
        }
    }
}
```

8.2 区间 DP

```
int main(){
    for(int i=1;i<=n;++i)dp[i][i]=0;//设置初始状态
    for(int len=2;len<=n;++len){
        for(int i=1;i+len-1<=n;++i){
            int j=i+len-1; //视情况而定,有两种可能的转移方式
        //1、枚举分段点/最后一个加进去的,有时可优化枚举个数
            for(int k=i;k<=j;++k)dp[i][j]+=dp[i][k]+dp[k+1][j]
        //2、每次只从相邻位置转移
            dp[i][j]=dp[i][j-1]+dp[i+1][j];
        }
    }
}
```

8.3 数位 DP

```
const int N=20;//数字长度
int h,a[N];
//dp第一维是搜索到第几位(从低到高),后面状态可以额外加
//时间复杂度是所有状态数,即dp数组的空间大小
ll dp[N][2][2][2];//dp[pos][f4][f62][pre6]
ll dfs(int pos,bool f4,bool f62,bool pre6,bool lim){//lim:高位均相等
    //lim是必要参数,只有!lim才能剪枝,否则后续状态受到高位限制
    if(pos==0)return !f4&&!f62?1:0;//pos=0整个数字选好了,如果满足就是搜到一个
    if(!lim&&dp[pos][f4][f62][pre6]!=-1)
    return dp[pos][f4][f62][pre6];
```

```
    int mx=lim?a[pos]:9;
    ll s=0;
    for(int i=0;i<=mx;++i)
    s+=dfs(pos-1,f4||i==4,f62||(pre6&&i==2),i==6,lim&&i==mx);
    if(!lim)dp[pos][f4][f62][pre6]=s;
    return s;
}
ll solve(ll x){//0~x的答案
    h=0;memset(dp,-1,sizeof dp);
    while(x){
        a[++h]=x%10;
        x/=10;
    }//a[1]=个位,a[h]=最高位
    return dfs(h,0,0,0,1);
}
int main(){//没有62或4的数字
    ll x,y;
    while(scanf("%lld%lld",&x,&y)!=EOF&&x){
        printf("%lld\n",solve(y)-solve(x-1));
    }
}
```

8.4 最长上升子序列

```
int n,a[N];
int dp1[N],len1;//len1=最长上升子序列长度
int dp2[N],len2;//最少len2个上升子序列可以覆盖整个数组
int main(){
    n=rd();
    for(int i=1;i<=n;++i)a[i]=rd();
    dp1[len1=1]=a[1];
    for(int i=2;i<=n;++i){
        if(a[i]>dp1[len1])dp1[++len1]=a[i];
        else{
            //*lower_bound(dp1+1,dp1+1+len1,a[i])=a[i];
            int l=1,r=len1;
            while(l<=r){
                int m=l+r>>1;
                if(dp1[m]<=a[i])l=m+1;
                else r=m-1;
            }
```

```
            // < < < < < > > > >
            //         r l
            dp1[l]=a[i];
        }
    }
    printf("%d\n",len1);
    dp2[len2=1]=a[1];
    for(int i=2;i<=n;++i){
        if(a[i]<dp2[len2])dp2[++len2]=a[i];
        else{
            int l=1,r=len2;
            while(l<=r){
                int m=l+r>>1;
                if(dp2[m]>=a[i])l=m+1;
                else r=m-1;
            }
            // > > > > > < < < < <
            //         r l
            dp2[l]=a[i];
        }
    }
    printf("%d\n",len2);
}
```

8.5 所有字段和

```
for(int i=1;i<=n;++i){
    dp[i]=dp[i-1]+i*a[i];
    ans+=dp[i];//dp[i]=所有以i结尾的字段和
}
```

8.6 公共子序列

最长公共子序列（若元素不重复，可以重编号后求最长上升子序列）

```
int n,m,a[N],b[N],dp[N][N];
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;++i)scanf("%d",&a[i]);
    for(int i=1;i<=m;++i)scanf("%d",&b[i]);
    for(int i=1;i<=n;++i){
```

```
        for(int j=1;j<=m;++j){
            dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
            if(a[i]==b[j])dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
        }
    }
    printf("%d\n",dp[n][m]);
}
```

所有公共子序列个数

```
int main(){
    for(int i=1;i<=n;++i){
        for(int j=1;j<=m;++j){
            dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1];
            if(a[i]==b[j])dp[i][j]+=dp[i-1][j-1]+1;
        }
    }
}
```

8.7 错位排列数

```
dp[i]=长度为i的排列,所有i满足p[i]!=i的方案数
dp[1]=0;dp[2]=1;dp[i]=(i-1)*(dp[i-1]+dp[i-2])
```

8.8 基环树直径

```
struct edge{
    int to,nx,v;
}e[N<<1];
int hd[N],tot;
inline void add(int x,int y,int z){
    e[++tot]={y,hd[x],z};hd[x]=tot;
}
bool vis[N];
int sta[N],top;
int n,c[N],h;
ll b[N];
ll maxd[N];
ll treeD;
ll presum[N],sufsum[N];
ll preans[N],sufans[N];
```

```
ll premax[N],sufmax[N];
void find_circle(int x,int fa){
    sta[++top]=x;vis[x]=1;
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(y==fa)continue;
        if(h)return;
        b[y]=e[i].v;
        if(!vis[y])find_circle(y,x);
        else{
            while(sta[top]!=y){
                c[++h]=sta[top--];
            }
            c[++h]=y;
            return;
        }
    }
    --top;
}
void getmaxd(int x,int fa){
    for(int i=hd[x];i;i=e[i].nx){
        int y=e[i].to;
        if(y==fa||vis[y])continue;
        getmaxd(y,x);
        treeD=max(treeD,maxd[x]+maxd[y]+e[i].v);
        maxd[x]=max(maxd[x],maxd[y]+e[i].v);
    }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        int x,y,z;scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);add(y,x,z);
    }
    find_circle(1,0);
    memset(vis+1,0,n);
    for(int i=1;i<=h;++i)vis[c[i]]=1;
    for(int i=1;i<=h;++i)getmaxd(c[i],0);
    for(int i=1;i<=h;++i)presum[i]=presum[i-1]+b[c[i-1]];
    for(int i=h-1;i;--i)sufsum[i]=sufsum[i+1]+b[c[i]];
    ll mx=0;
```

```
    for(int i=1;i<=h;++i){
        preans[i]=max(preans[i-1],presum[i]+maxd[c[i]]+mx);
        mx=max(mx,maxd[c[i]]-presum[i]);
    }
    mx=0;
    for(int i=h;i;--i){
        sufans[i]=max(sufans[i+1],sufsum[i]+maxd[c[i]]+mx);
        mx=max(mx,maxd[c[i]]-sufsum[i]);
    }
    for(int i=1;i<=h;++i)premax[i]=max(premax[i-1],presum[i]+maxd[c[i]]);
    for(int i=h;i;--i)sufmax[i]=max(sufmax[i+1],sufsum[i]+maxd[c[i]]);
    ll ans=1ll<<60;
    for(int i=1;i<h;++i){
        ll now=max(preans[i],sufans[i+1]);
        now=max(now,premax[i]+sufmax[i+1]+b[c[h]]);
        ans=min(ans,now);
    }
    ans=max(ans,treeD);
    printf("%.1lf\n",1.0*ans/2);
}
```

8.9 高维前缀和

SOS DP (sum of subset)

```
//a[i][j][k]=sum[1~i][1~j][1~k]
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= n; j++)
        for(int k = 1; k <= n; k++)
            a[i][j][k] += a[i - 1][j][k];
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= n; j++)
        for(int k = 1; k <= n; k++)
            a[i][j][k] += a[i][j - 1][k];
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= n; j++)
        for(int k = 1; k <= n; k++)
            a[i][j][k] += a[i][j][k - 1];
//类似地,四维就再来一遍
```

若每一维只有 2 即 (0,1) 则可优化为 (也即是求子集和)

```
for(int j = 0; j < n; j++)
```

```
    for(int i = 0; i < 1 << n; i++)
        if(i >> j & 1) f[i] += f[i ^ (1 << j)];
```

超集和 (超集: 若 S2 的每一个元素都在 S1 且 S1 可能包含 S2 中没有的元素, 则 S1 为 S2 的一个超集合)(其实就是子集反一下)

```
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(!(i >> j & 1)) f[i] += f[i ^ (1 << j)];
```

9 字符串

9.1 KMP 算法

一个字符串 s 的 border：既是 s 的前缀又是 s 的后缀的真子串

KMP 算法可以建立 $next$ 数组（p 数组）, 使得 $next_i$ = 前缀 i 的最长 border 长度

串长-border 长度 = 周期，周期个数 =border 数量，最小循环节为 $n - next_n$

```
char s1[N],s2[N];
int n,m,p[N];//p[i]=s2[1~i]后缀=前缀的最大长度(不包括本身)
int main(){
    scanf("%s%s",s1+1,s2+1);
    n=strlen(s1+1);m=strlen(s2+1);
    for(int i=2,j=0;i<=m;++i){
        while(j&&s2[i]!=s2[j+1])j=p[j];
        if(s2[i]==s2[j+1])++j;
        p[i]=j;
    }
    for(int i=1,j=0;i<=n;++i){
        while(j&&s1[i]!=s2[j+1])j=p[j];
        if(s1[i]==s2[j+1])++j;
        if(j==m){
            printf("%d\n",i-m+1);//s1中出现s2的下标
            j=p[j];//不能重叠: j=0
        }
    }
}
```

9.2 失配树

建立 KMP 算法的 $next$ 数组（下标从 1 开始），建立 $fail$ 树，$fail[i] = p[i]$

0 到 n 是失配树从根到叶子的拓扑序

求前缀 $prefix(u)$ 和 $prefix(v)$ 的最大公共 $border$

法一、求出 $fail$ 树上 $fa = lca(u, v)$, 若 u 是 v 的祖先, 则答案为 $next[fa]$，否则答案为 fa

```
int n,logn;char s[N];//下标从1开始
int f[21][N],d[N];//f[i][0]=s[1~i]最长border
int lca(int x,int y){
    if(d[x]<d[y])swap(x,y);
    for(int i=logn;i>=0;--i){
        if(d[f[i][x]]>=d[y])x=f[i][x];
    }
    //若x==y,说明xy为祖先关系,最终答案为next[x],即f[0][x]
    for(int i=logn;i>=0;--i){
        if(f[i][x]!=f[i][y]){
            x=f[i][x];y=f[i][y];
        }
    }
    return f[0][x];
}
int main(){
    scanf("%s",s+1);//ababaabab
    n=strlen(s+1);logn=log2(n)+1;
    d[1]=1;//root=0, fa[1]=0, dep[0]=0, dep[1]=1
    for(int i=2,j=0;i<=n;++i){
        while(j&&s[i]!=s[j+1])j=f[0][j];
        if(s[i]==s[j+1])++j;
        f[0][i]=j;d[i]=d[j]+1;
    }
    for(int i=1;i<=logn;++i){
        for(int j=1;j<=n;++j){
            f[i][j]=f[i-1][f[i-1][j]];
        }
    }
    int q;scanf("%d",&q);
    while(q--){
        int x,y;scanf("%d%d",&x,&y);
        printf("%d\n",lca(x,y));
    }
}
```

法二、$pre(u)$ 循环节长度为 $u - next[u]$，根据循环节长度是否大于一半来分类讨论，使 u 或 v 每次减半

```
int n,p[N];char s[N];//下标从1开始
int main(){
    scanf("%s",s+1);n=strlen(s+1);//ababaabab
    for(int i=2,j=0;i<=n;++i){
        while(j&&s[i]!=s[j+1])j=p[j];
        if(s[i]==s[j+1])++j;
        p[i]=j;
    }
    int q;scanf("%d",&q);
    while(q--){
        int u,v;scanf("%d%d",&u,&v);
        u=p[u];v=p[v];//一开始要先跳一次,保证非本身
        while(u!=v){
            if(u<v)swap(u,v);
            int d=u-p[u];//最小循环节
            if(d<u>>1){//小于一半
                if((u-v)%d==0)u=v;//u和v恰好都是循环节的k倍,得到答案
                else u=u%d+d;//循环节只保留一个abcabcabcab->abcab
            }
            else u=p[u];
        }
        printf("%d\n",u);
    }
}
```

9.3 扩展 KMP

```
//z[i]=s[i~n]和s的最长公共前缀
//p[i]=t[i~m]和s的最长公共前缀
int n,m,z[N],p[N];char s[N],t[N];
void init_Z(char s[],int n,int z[]){
    z[1]=n;
    for(int i=2,l=0,r=0;i<=n;++i){
        z[i]=i<=r?min(z[i-l+1],r-i+1):0;
        while(i+z[i]<=n&&s[i+z[i]]==s[z[i]+1])++z[i];
        if(i+z[i]-1>r)l=i,r=i+z[i]-1;
    }
}
void exkmp(char s[],int n,char t[],int m,int z[],int p[]){
    for(int i=1,l=0,r=0;i<=m;++i){
```

```
        if(i<=r)p[i]=min(z[i-l+1],r-i+1);
        while(i+p[i]<=m&&t[i+p[i]]==s[p[i]+1])++p[i];
        if(i+p[i]-1>r)l=i,r=i+p[i]-1;
    }
}
```

9.4 字符串哈希

如果怕自然溢出被卡，可以考虑自然溢出和 1e9+7/998244353 的双模哈希
注意模 1e9+7 时必须用 long long 而非 unsigned long long
哈希判回文串：

```
using ll=long long;using ull=unsigned long long;
const int N=1e5+5;
const ull k=13331;//k看心情随便换
int n;char s[N];
ull hash1[N],hash2[N],kpow[N];
bool check(int l,int r){//s[l,r]是回文串?
    ull hs1=hash1[r]-hash1[l-1]*kpow[r-l+1];//不需要考虑负数
    ull hs2=hash2[l]-hash2[r+1]*kpow[r-l+1];//自然溢出后是模2^64意义下的
    return hs1==hs2;
}
int main(){
    scanf("%d%s",&n,s+1);//n=strlen(s+1);
    kpow[0]=1;hash1[n+1]=0;
    for(int i=1;i<N;++i)kpow[i]=kpow[i-1]*k;
    for(int i=1;i<=n;++i)hash1[i]=hash1[i-1]*k+s[i];
    for(int i=n;i>=1;--i)hash2[i]=hash2[i+1]*k+s[i];
    int q;scanf("%d",&q);
    while(q--){
        int l,r;scanf("%d%d",&l,&r);
        puts(check(l,r)?"YES":"NO");
    }
}
/*  hash("dcbe")='d'*k^3+'c'*k^2+'b'*k+'a'*1
hash1[i]=hash({s[1],s[2],.....,s[i]})
hash2[i]=hash({s[n],s[n-1],...,s[i]})        */
```

双模哈希

```
ull kpow1[N],hs1[N];//% 2^64
ll kpow2[N],hs2[N];//% mod(1e9+7)
```

```
char s[N];int n;
pair<ull,ll>gethash(int l,int r){
    ull a=hs1[r]-hs1[l-1]*kpow1[r-l+1];
    ll b=hs2[r]-hs2[l-1]*kpow2[r-l+1];
    b=(b%mod+mod)%mod;//注意b要用有符号，绝对不要用unsigned long long
    return {a,b};
}
map<pair<ull,ll>,int>mp;//双模哈希为key映射到其他信息
int main(){
    kpow1[0]=kpow2[0]=1;
    for(int i=1;i<N;++i){
        kpow1[i]=kpow1[i-1]*13331;
        kpow2[i]=kpow2[i-1]*13331%mod;
    }
    scanf("%s",s+1);n=strlen(s+1);
    for(int i=1;i<=n;++i){
        hs1[i]=hs1[i-1]*13331+s[i];
        hs2[i]=(hs2[i-1]*13331+s[i])%mod;
    }
}
```

9.5 Manacher

```
const int N=2.2e7+5;//strlen=2倍原字符串长度+1
int n,p[N];char s[N];
bool check(int l,int r){//[l,r]区间是否是回文串
    return p[l+r]>=r-l+1;
}
int main(){
    scanf("%s",s+1);n=strlen(s+1); //将字符串acbc变为 #a#c#b#c#
    for(int i=n;i;--i)s[i<<1|1]='#',s[i<<1]=s[i];
    s[1]='#';s[0]='@';//第0个位置赋为不存在的字符,控制循环中的while的边界
    n=n<<1|1;  int maxr=0,pos=0,ans=0;
    for(int i=1;i<=n;++i){
        if(i<maxr)p[i]=min(p[pos*2-i],maxr-i);
        while(s[i-p[i]-1]==s[i+p[i]+1])++p[i];
        if(i+p[i]>maxr)maxr=i+p[i],pos=i;
    }
    //p[i]=新串以i为中心的回文串单侧最大扩展长度
    //举例 : str = #a#b#b#c(下标从1开始),p[5]=2,p[4]=1
    //max(p[i])=原串最长回文串长度
```

```
    //p[i]在原字符串中代表的回文区间是[(i+1)/2-p[i]/2,(i+1)/2+(p[i]+1)/2]
    //原字符串以第i个字符为中心的回文串长度为p[i*2]
    //原字符串以[i,i+1]两个字符为中心的回文串长度为p[i*2+1]
}
```

9.6 AC 自动机

```
const int N=2e5+10,M=N*30;//注意修改M的大小(文本串长度之和)
int t[M][26],fail[M],cnt[M],idx[M],tot;
string s[N];
vector<int>e[M];
int n;
void insert(const string &s,int id){
    int p=0;
    for(auto ch:s){
        int c=ch-'a';
        if(!t[p][c])t[p][c]=++tot;
        p=t[p][c];
    }
    idx[id]=p;
}
void build_fail(){
    queue<int>q;
    for(int i=0;i<26;++i)if(t[0][i])q.push(t[0][i]);
    for(;!q.empty();){
        int p=q.front();q.pop();
        for(int i=0;i<26;++i){
            if(t[p][i]){
                fail[t[p][i]]=t[fail[p]][i];
                q.push(t[p][i]);
            }
            else t[p][i]=t[fail[p]][i];
        }
    }
}
void query(const string &s){
    int p=0;
    for(auto c:s){
        p=t[p][c-'a'];
        ++cnt[p];
    }
```

```
}
void build_failtree(){
    for(int i=1;i<=tot;++i)e[fail[i]].push_back(i);
}
void dfs(int x){
    for(auto y:e[x]){
        dfs(y);
        cnt[x]+=cnt[y];
    }
}
int main(){
    cin>>n;
    for(int i=1;i<=n;++i)cin>>s[i];
    for(int i=1;i<=n;++i)insert(s[i],i);
    build_fail();   build_failtree();
    string str; cin>>str;
    query(str);  dfs(0);
    for(int i=1;i<=n;++i)cout<<cnt[idx[i]]<<"\n";
}
```

9.7 后缀数组 SA

```
int n,m,s[N];  char str[N];
int sa[N];//sa[i]=排名为i的后缀的下标(从1开始)
int rk[N];//rk[i]=下标为i的后缀的排名
int height[N];//height[i]=lcp(suf(sa[i]),suf(sa[i-1])
//lcp{suf(sa[i]),suf(sa[j])}=min{height[i+1,...,j]}
int st[N][20];//用于求longest commen prefix
int c[N],x[N],y[N];
void get_sa(){//也可以传入int数组
    m=122;//ASCII最大值
    for(int i=1;i<=n;++i)++c[x[i]=s[i]];
    for(int i=2;i<=m;++i)c[i]+=c[i-1];
    for(int i=1;i<=n;++i)sa[c[x[i]]--]=i;
    for(int k=1;k<=n;k<<=1){
        int num=0;memset(c+1,0,m<<2);
        for(int i=n-k+1;i<=n;++i)y[++num]=i;
        for(int i=1;i<=n;++i)if(sa[i]>k)y[++num]=sa[i]-k;
        for(int i=1;i<=n;++i)++c[x[i]];
        for(int i=2;i<=m;++i)c[i]+=c[i-1];
        for(int i=n;i;--i)sa[c[x[y[i]]]--]=y[i],y[i]=0;
```

```
        swap(x,y);
        x[sa[1]]=1;num=1;
        for(int i=2;i<=n;++i)
        x[sa[i]]=y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k]?num:++num;
        if (num==n) break;
        m=num;
    }
}
void get_height(){
    for(int i=1;i<=n;++i)rk[sa[i]]=i;
    int k=0;height[1]=0;
    for(int i=1;i<=n;++i){
        if(!rk[i])continue;
        if(k)--k;
        int j=sa[rk[i]-1];
        while(i+k<=n&&j+k<=n&&s[i+k]==s[j+k])++k;
        height[rk[i]]=k;
    }
}
int Log2[N];
void build_st(){
    for(int i=1;i<=n;++i)st[i][0]=height[i];
    for(int j=1;1<<j<=n;++j){
        for(int i=1;i+(1<<j)-1<=n;++i){
            st[i][j]=min(st[i][j-1],st[i+(1<<j-1)][j-1]);
        }
    }
}
int lcp(int l,int r){
    l=rk[l];r=rk[r];
    if(l>r)swap(l, r);++l;
    int k=Log2[r-l+1];
    return min(st[l][k],st[r-(1<<k)+1][k]);
}
int main(){
    for(int i=2;i<N;++i)Log2[i]=Log2[i>>1]+1;
    scanf("%s",str+1);
    n=strlen(str+1);
    for(int i=1;i<=n;++i)s[i]=str[i];
    get_sa();get_height();build_st();
}
```

9.8 后缀自动机 SAM

```
const int M=1e6+6;//M=size(SAM)=2*len
int ch[M][26];//当字符集较大可改为 map<int,int>ch[M];
int fa[M],sam_cnt,last,len[M],sz[M],sum[M];
void sam_init(){last=sam_cnt=1;}
void extend(int c){
    int p=last,np=++sam_cnt;
    last=np; len[np]=len[p]+1;
    while(p&&!ch[p][c])ch[p][c]=np,p=fa[p];
    if(!p)fa[np]=1;
    else{
        int q=ch[p][c];
        if(len[q]==len[p]+1)fa[np]=q;
        else{
            int nq=++sam_cnt;
            memcpy(ch[nq],ch[q],sizeof ch[nq]);//若ch是map,改为ch[nq]=ch[q]
            len[nq]=len[p]+1; fa[nq]=fa[q]; fa[q]=fa[np]=nq;
            while(p&&ch[p][c]==q)ch[p][c]=nq,p=fa[p];
        }
    }
}
//fa[i]=i点在parent树上的祖先
//len[i]=i点endpos类中的最大字符串长度
//sz[i]=i点endpos类的大小=i点字符串集合出现次数
//sum[i]=从起点开始经过i点的路径数量
void match_suffix(string &str){//匹配后缀
    int p=1,l=0;
    for(int i=0;i<str.size();++i){
        int c=str[i]-'a';
        while(!ch[p][c]){
            p=fa[p];l=len[p];
        }
        if(ch[p][c]){
            p=t[p][c];++l;//l=str[0:i]能匹配上的最长后缀长度
        }
        //--l;if(p!=1&&len[fa[p]]>=l)p=fa[p];//删除当前str最左边的一个字符
    }
}
int temp[M],rk[M];//rk[i]=len第i小的结点(树从根到叶子的拓扑序)
inline void calc(){
    int Type=1;//Type=1:不同位置的相同字符串算不同的,0:相同字符串算一次
```

```
    for(int i=1;i<=sam_cnt;++i)++temp[len[i]];
    for(int i=1;i<=sam_cnt;++i)temp[i]+=temp[i-1];
    for(int i=1;i<=sam_cnt;++i)rk[temp[len[i]]--]=i;
    for(int i=sam_cnt;i;--i)sz[fa[rk[i]]]+=sz[rk[i]];
    for(int i=1;i<=sam_cnt;++i)Type?sum[i]=sz[i]:sum[i]=sz[i]=1;
    sum[1]=sz[1]=0;
    for(int i=sam_cnt;i;--i){
        for(int j=0;j<26;++j){
            sum[rk[i]]+=sum[ch[rk[i]][j]];
        }
    }
}
void findkth(int x,int k){//第k小字典序子串
    if(k<=sz[x])return;
    k-=sz[x];
    for(int i=0;i<26;++i){
        int y=ch[x][i];
        if(sum[y]>=k){//空儿子的sum=0
            putchar('a'+i);
            findkth(y,k);
            return;
        }
        else k-=sum[y];
    }
}
int main(){
    sam_init();
    for(int i=1;i<=n;++i){
        extend(s[i]-'a');
        ++sz[last];
    }
}
```

9.9 广义 SAM

建立一个 SAM，包含了多个串的所有子串

```
const int M=2e6+6;//M=2*sum(strlen)+1
int ch[M][26],len[M],fa[M],sz[M],last,sam_cnt;
void sam_init(){sam_cnt=last=1;}
int extend(int c,int last){//last=extend(c,last)
    if(ch[last][c]){
```

```
        int p=last,q=ch[p][c];
        if(len[p]+1==len[q])return q;
        else{
            int np=++sam_cnt;len[np]=len[p]+1;
            memcpy(ch[np],ch[q],sizeof ch[np]);
            fa[np]=fa[q],fa[q]=np;
            while(p&&ch[p][c]==q)ch[p][c]=np,p=fa[p];
            return np;
        }
    }
    int p=last,np=++sam_cnt;
    last=np; len[np]=len[p]+1;
    while(p&&!ch[p][c])ch[p][c]=np,p=fa[p];
    if(!p)fa[np]=1;
    else{
        int q=ch[p][c];
        if(len[q]==len[p]+1)fa[np]=q;
        else{
            int nq=++sam_cnt;
            memcpy(ch[nq],ch[q],sizeof ch[nq]);
            len[nq]=len[p]+1; fa[nq]=fa[q]; fa[q]=fa[np]=nq;
            while(p&&ch[p][c]==q)ch[p][c]=nq,p=fa[p];
        }
    }
    return np;
}
char s[M];
int main(){
    int n;scanf("%d",&n);sam_init();
    for(int i=1;i<=n;++i){//n个串在线建立广义SAM
        scanf("%s",s);
        int len=strlen(s);last=1;
        for(int j=0;j<len;++j){
            last=extend(s[j]-'a',last);
            ++sz[last];//所有子串都存在
        }
        //++sz[last];//只有完整的串存在
    }
}
```

区间本质不同字串

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+10,M=N*2;
struct sgt{
    int l,r;
    ll v,tag;
}t1[M<<2];
#define ls (p<<1)
#define rs (p<<1|1)
void pushup1(int p){
    t1[p].v=t1[ls].v+t1[rs].v;
}
void build(int l,int r,int p){
    t1[p].l=l;t1[p].r=r;
    if(l==r)return;
    int mid=l+r>>1;
    build(l,mid,ls);
    build(mid+1,r,rs);
}
void up1(int p,ll c){
    t1[p].tag+=c;
    t1[p].v+=c*(t1[p].r-t1[p].l+1);
}
void pushdown1(int p){
    if(t1[p].tag){
        up1(ls,t1[p].tag);
        up1(rs,t1[p].tag);
        t1[p].tag=0;
    }
}
void upd(int l,int r,int p,ll c){
    if(l<=t1[p].l&&t1[p].r<=r){
        up1(p,c);return;
    }
    pushdown1(p);
    int mid=t1[p].l+t1[p].r>>1;
    if(l<=mid)upd(l,r,ls,c);
    if(r>mid)upd(l,r,rs,c);
    pushup1(p);
}
```

```
ll qry(int l,int r,int p){
    if(l<=t1[p].l&&t1[p].r<=r)return t1[p].v;
    if(r<t1[p].l||l>t1[p].r)return 0;
    pushdown1(p);
    return qry(l,r,ls)+qry(l,r,rs);
}

int ch[M][26];
int fa[M],sam_cnt,last,len[M],sz[M],sum[M];
void sam_init(){last=sam_cnt=1;}
void extend(int c){
    int p=last,np=++sam_cnt;
    last=np; len[np]=len[p]+1; sz[np]=1;
    while(p&&!ch[p][c])ch[p][c]=np,p=fa[p];
    if(!p)fa[np]=1;
    else{
        int q=ch[p][c];
        if(len[q]==len[p]+1)fa[np]=q;
        else{
            int nq=++sam_cnt;
            memcpy(ch[nq],ch[q],sizeof ch[nq]);
            len[nq]=len[p]+1; fa[nq]=fa[q]; fa[q]=fa[np]=nq;
            while(p&&ch[p][c]==q)ch[p][c]=nq,p=fa[p];
        }
    }
}

struct LCT{
    int c[2],f;
    int tag,pos;
}t[M];
#define notroot(x) (t[t[x].f].c[0]==x||t[t[x].f].c[1]==x)
void make_tag(int x,int id){
    t[x].tag=t[x].pos=id;
}
void pushdown(int x){
    if(t[x].tag){
        if(t[x].c[0])make_tag(t[x].c[0],t[x].tag);
        if(t[x].c[1])make_tag(t[x].c[1],t[x].tag);
        t[x].tag=0;
    }
```

```
}
void rotate(int x){
    int y=t[x].f,z=t[y].f;
    int c=t[y].c[1]==x;
    if(notroot(y))t[z].c[t[z].c[1]==y]=x;t[x].f=z;
    t[y].c[c]=t[x].c[c^1];t[t[x].c[c^1]].f=y;
    t[x].c[c^1]=y;t[y].f=x;
}
void splay(int x){
    static int sta[M],top;
    int y=x; sta[top=1]=y;
    while(notroot(y))sta[++top]=y=t[y].f;
    while(top)pushdown(sta[top--]);
    while(notroot(x)){
        int y=t[x].f,z=t[y].f;
        if(notroot(y))rotate(t[y].c[1]==x^t[z].c[1]==y?x:y);
        rotate(x);
    }
}
void access(int x,int id){
    int f=0;
    for(;x;f=x,x=t[x].f){
        splay(x);
        t[x].c[1]=f;
        if(t[x].pos){
            upd(t[x].pos-len[x]+1,t[x].pos-len[t[x].f],1,-1);
        }
    }
    upd(1,id,1,1);
    make_tag(f,id);
}

int n,m,idx[N];
char s[N];
vector<pair<int,int>>q[N];
ll ans[M];
int main(){
    sam_init();
    scanf("%s%d",s+1,&m);
    n=strlen(s+1);
    for(int i=1;i<=m;++i){
```

```
        int l,r;scanf("%d%d",&l,&r);
        q[r].push_back({l,i});
    }
    for(int i=1;i<=n;++i){
        extend(s[i]-'a');
        idx[i]=last;
    }
    for(int i=2;i<=sam_cnt;++i){
        t[i].f=fa[i];
    }
    build(1,sam_cnt,1);
    for(int r=1;r<=n;++r){
        access(idx[r],r);
        for(auto pr:q[r]){
            ans[pr.second]=qry(pr.first,r,1);
        }
    }
    for(int i=1;i<=m;++i)printf("%lld\n",ans[i]);
}
```

按 Trie 树构造 (字符在边上)

```
//pos[x]=Trie树上的x在SAM中的编号
//trie.c[x] fa[x] t[x] 分别代表该点字符,父亲和出边
void build(){
    queue<int>q;
    for(int i=0;i<26;++i)if(trie.t[1][i])q.push({i);
    while(q.size()){
        int x=q.front();q.pop();
        pos[x]=extend(trie.c[x],trie.fa[x]);
        for(int i=0;i<26;++i)if(trie.t[x][i])q.push(trie.t[x][i]);
    }
}
```

按树构造 (字符在点上)

```
void dfs(int x,int f,int last){
    int t=extend(c[x],last);
    for(auto y:e[x]){
        if(y==f)continue;
        dfs(y,x,t);
    }
}
```

9.10 子序列自动机

建立一个 0 号起点，和所有位置相连，转移边输入该位置的字符，每个位置记录下一个字符最近的位置 (不包括自身位置)

这样，从 0 开始的所有路径都是原串的子序列

```
int ch[N][26],pos[26];
void build(char s[],int n){
    memset(pos,0,sizeof pos);
    for(int i=n;i;--i){
        memcpy(ch[i],pos,sizeof pos);
        pos[s[i]-'a']=i;
    }
    memcpy(ch[0],pos,sizeof pos);//0号点的出边为pos[]
}
```

值域较大时，可以用 vector/set 保存每个值出现过的位置，每个点右边最近的 x 需要在 vector 上二分

```
int n,q,a[N],b[N],h;
set<int>pos[N];//pos[i]=i出现的位置,vector不带修,set带修
bool match(int b[],int h){//b[1~h]是a[1~n]的子序列?
    int x=0;
    for(int i=1;i<=h;++i){
        auto it=pos[b[i]].upper_bound(x);//二分出下一个位置
        if(it==pos[b[i]].end())return 0;
        x=*it;
    }
    return 1;
}
int main(){
    scanf("%d%d",&n,&q);
    for(int i=1;i<=n;++i){
        scanf("%d",&a[i]);pos[a[i]].insert(i);
    }
    while(q--){
        scanf("%d",&h);
        for(int i=1;i<=h;++i)scanf("%d",&b[i]);
        if(match(b,h))puts("Yes");
        else puts("No");
    }
}
```

9.11 回文自动机 PAM

```
const int N=3e5+10;//PAM结点数<=串长
char s[N];
int ch[N][26],pam_cnt,last,fail[N],len[N],sz[N],pos[N];
//0是偶回文串的根,1是奇回文串的根
//fail[i]=i结点代表的串的最长真回文后缀的结点
//len[i]=i结点代表的串的长度
//sz[i]=i结点包含的回文子串个数
//pos[i]=i结点代表的串的右端点在原串的下标
//cnt[i]=i结点代表的串出现次数，需要子树和
void pam_init(){
    for(int i=0;i<=pam_cnt;++i){
        memset(ch[i],0,sizeof ch[i]);
        len[i]=fail[i]=sz[i]=cnt[i]=0;
    }
    pam_cnt=1; last=0; fail[0]=1; fail[1]=1; len[1]=-1;
}
int getfail(int x,int i){
    while(i-len[x]-1<0||s[i-len[x]-1]!=s[i])x=fail[x];
    return x;
}
void extend(int c,int i) {
    int f=getfail(last,i);
    if(!ch[f][c]){
        int p=++pam_cnt;
        pos[p]=i;
        len[p]=len[f]+2;
        int q=getfail(fail[f],i);
        fail[p]=ch[q][c];
        sz[p]=sz[fail[p]]+1;
        ch[f][c]=p;
    }
    last=ch[f][c];
    ++cnt[last];
}
int main(){
    scanf("%s",s+1);
    int n=strlen(s+1);
    pam_init();
    for(int i=1;i<=n;++i)extend(s[i]-'a',i);//i必须1~n,因为getfail用到下标
    for(int i=2;i<=pam_cnt;++i){//枚举所有本质不同回文串
```

```
        int r=pos[i],l=r-len[i]+1;//s[l,r]

    }
}
```

10 计算几何

10.1 实数运算

```
typedef long double db;//便于修改double和long double
const int N=2e5+5;
const db eps=1e-18,pi=acos(-1);
int sgn(db x){return x<-eps?-1:x>eps;}//负数:-1,零:0,正数:1
int cmp(db a,db b){return sgn(a-b);}//小于:-1,等于:0,大于:1
void wt(db x){printf("%.5Lf",abs(x)<5e-6?0:x);}//避免输出负0(-0.0000)
void wtb(db x){wt(x);putchar(' ');}
void wtl(db x){wt(x);putchar('\n');}
//ceil(x)向上取整,floor(x)向下取整,round(x)四舍五入
//解方程 ax^2+bx+c=0, 解为x1,x2(x1<=x2)
bool solve_eqution(db a,db b,db c,db &x1,db &x2){
    db delta=b*b-4*a*c;
    if(sgn(delta)==-1)return 0;
    db q,t=sqrt(delta);
    if(sgn(b)==-1)q=-0.5*(b-t);
    else q=-0.5*(b+t);
    x1=q/a;x2=c/q;if(cmp(x1,x2)==1)swap(x1,x2);
    return 1;
}
```

10.2 克莱姆法则

$$\text{解方程组}\begin{cases} a_{11}x_1+a_{12}x_2+...+a_{1n}x_n=b_1 \\ a_{21}x_1+a_{22}x_2+...+a_{2n}x_n=b_2 \\ ... \\ a_{n1}x_1+a_{n2}x_2+...+a_{nn}x_n=b_n \end{cases}$$

令 $D=\begin{vmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nn} \end{vmatrix}$，令 $D_i=D$ 的第 i 列变为 $b_1 \sim b_n$，则 $x_i=\frac{D_i}{D}$

```
//a11*x + a12*y = b1,  a21*x + a22*y = b2, 返回pair(x,y)
pair<db,db>solve(db a11,db a12,db b1,db a21,db a22,db b2){
    db z=(a11*a22-a12*a21);
    db x=(b1*a22-a12*b2)/z;
    db y=(b2*a11-b1*a21)/z;
    return {x,y};
}
```

10.3 自适应辛普森积分

```
db simpson(double l,double r){
    return (f(l)+f(r)+4*f((l+r)/2.0))*(r-l)/6.0;
}
db rsimpson(double l,double r){//定积分[l,r] f(x)dx
    double mid=(l+r)/2.0;
    if(abs(simpson(l,r)-simpson(l,mid)-simpson(mid,r))<eps)
        return simpson(l,mid)+simpson(mid,r);
    return rsimpson(l,mid)+rsimpson(mid,r);
}
```

10.4 点/向量类

```
struct P{//点、向量类
    db x,y;
    P(db a=0,db b=0):x(a),y(b){}
    P operator + (P p)const{return P(x+p.x,y+p.y);}
    P operator - (P p)const{return P(x-p.x,y-p.y);}
    P operator * (db a)const{return P(x*a,y*a);}
    P operator / (db a)const{return P(x/a,y/a);}
    db len(){return sqrt(x*x+y*y);}
    db len2(){return x*x+y*y;}
    P rotate90(){return P(-y,x);}
    P rotate270(){return P(y,-x);}
    P rotate(db a){db c=cos(a),s=sin(a);return P(c*x-s*y,s*x+c*y);}//绕原点逆时针旋转a
    P strech(db l){return *this*l/len();}
    void rd(){scanf("%Lf%Lf",&x,&y);}
    void show(){printf("%.2Lf %.2Lf\n",x,y);}
```

```
    friend P operator * (db a,P p){return P(p.x*a,p.y*a);}
    bool operator<(const P&p)const{int c=cmp(x,p.x);return c?c==-1:cmp(y,p.y);}
    bool operator==(const P&p)const{return cmp(x,p.x)==0&&cmp(y,p.y)==0;}
    int up(){return sgn(y)>0||sgn(y)==0&&sgn(x)>0;}//用于极角牌序
};
db dot(P a,P b){return a.x*b.x+a.y*b.y;}//点积
db cross(P a,P b){return a.x*b.y-a.y*b.x;}//叉积
db dis(P a,P b){return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}//距离
db sqdis(P a,P b){return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);}//距离的平方
db get_angle(P a,P b){return abs(atan2(abs(cross(a,b)),dot(a,b)));}//获得向量OA和
   OB的夹角[0,pi/2]
db get_angle1(P a,P b){return atan2(cross(a,b),dot(a,b));}//向量a逆时针转向b的角
   度
P lerp(P a,P b,db t){return a*(1-t)+b*t;}//直线上截取某一点连线
```

10.5 点线关系

```
bool point_on_line(P p,P s,P t){//p在点向式直线s+kt上
    return !sgn(cross(p-s,t));
}
bool point_on_seg(P p,P a,P b){//p在线段a-b上
    return sgn(cross(p-a,b-a))==0&&sgn(dot(p-a,p-b))<=0;
}
P point_proj_line(P p,P s,P t){//p投影到点向式直线s+kt
    return s+t*(dot(p-s,t)/t.len2());
}
```

10.6 线线关系

```
//判断两直线平行,方向向量为t1,t2
bool parallel(P t1,P t2){return !sgn(cross(t1,t2));}
//直线求交点
P line_int_line(P a,P u,P b,P v){//点向式a+ku,b+kv,需要先判断不平行
    db t=cross(b-a,v)/cross(u,v);
    return a+u*t;
}
//判断直线s+kt与线段ab是否相交 0:no 1:yes -1:交于端点
int line_int_seg(P s,P t,P a,P b){
    int d1=sgn(cross(t,a-s));
    int d2=sgn(cross(t,b-s));
```

```
    if((d1^d2)==-2)return 1;// 1 xor -1 = -2
    if(d1==0||d2==0)return -1;
    return 0;
}
//判断线段ab,pq间是否有交点,有交点的话可以用直线交点函数求出交点
int seg_int_seg(P a, P b, P p, P q) {
    if (point_on_seg(p, a, b) || point_on_seg(q, a, b) ||
        point_on_seg(a, p, q) || point_on_seg(b, p, q)) {
        return -1; //重合或交点在端点
    }
    int d1 = sgn(cross(b - a, p - a)), d2 = sgn(cross(b - a, q - a));
    int d3 = sgn(cross(q - p, a - p)), d4 = sgn(cross(q - p, b - p));
    return (d1^d2)==-2&&(d3^d4)==-2; // 1:有交点,且交点不在端点, 0:无交点
}
```

10.7 极角排序

```
//1、可以整数(long long)运算,以原点(0,0)为极点进行极角排序
bool cmp(P a,P b){//角度划分为[0,180)和[180,360),再判断叉积>0
    return a.up()^b.up()?a.up()>b.up():cross(a,b)>0;
    //后半句可以不加,表示角度相同时离原点距离小的在前
    //cross(a,b)>0||cross(a,b)==0&&a.len2()<b.len2();
}
//2、按左下角的点极角排序
void polar_angle_sort(P p[],int n,P O=P(0,0)){//以O为极点,极角排序p[1~n]
    for(int i=1;i<=n;++i){
        if(cmp(p[i].y,O.y)==-1||cmp(p[i].y,O.y)==0&&cmp(p[i].x,O.x)==-1){
            O=p[i];//选取y最小,x最小的点为极点
        }
    }
    sort(p+1,p+1+n,[&](P a,P b){
        db t=cross(a-O,b-O);
        return sgn(t)?t>0:sqdis(a,O)<sqdis(b,O);
    });
}
```

10.8 多边形、凸包

```
bool cmpxy(P a,P b){
    int k=cmp(a.x,b.x);
```

```
3       return k==-1||k==0&&cmp(a.y,b.y)==-1;
4   }
5   int convex_hull(P a[],int n,P p[]){//a[1~n]凸包,返回p[1~top]
6       static int sta[N]; int top=0;
7       sort(a+1,a+1+n,cmpxy);
8       for(int i=1;i<=n;++i){
9           while(top>=2&&sgn(cross(a[i]-a[sta[top-1]],a[sta[top]]-a[sta[top-1]]))
                >=0)--top;
10          sta[++top]=i;
11      }
12      int k=top;
13      for(int i=n-1;i;--i){
14          while(top>k&&sgn(cross(a[i]-a[sta[top-1]],a[sta[top]]-a[sta[top-1]]))>=0)
                --top;
15          sta[++top]=i;
16      }
17      if(n>1)--top;//去除首尾相连点
18      for(int i=1;i<=top;++i)p[i]=a[sta[i]];
19      return top;
20  }
21  //求多边形面积
22  db getS(P a[],int n){
23      db s=0;
24      for(int i=1;i<=n;++i)s+=cross(a[i],a[i+1]);
25      return abs(s)/2;//s>0:点是逆时针顺序,s<0:点是顺时针顺序
26  }
27  //求多边形重心
28  P polycenter(P a[],int n){
29      P ans(0,0); a[0]=a[n];
30      for(int i=0;i<n;i++)ans=ans+(a[i]+a[i+1])*cross(a[i],a[i+1]);
31      return ans/getS(a,n)/6;
32  }
33  //判点在是否在多边形内或边上
34  bool point_in_polygon(P p,P a[],int n){
35      int s=0;a[n+1]=a[1];
36      for(int i=1;i<=n;i++){
37          P u=a[i],v=a[i+1];
38          if(point_on_seg(p,u,v))return 1;
39          if(cmp(u.y,v.y)<=0)swap(u,v);
40          if(cmp(p.y,u.y)>0||cmp(p.y,v.y)<=0)continue;
41          if(sgn(cross(v-p,u-p))==1)++s;
```

```
    }
    return s&1;
}
```

10.9 旋转卡壳

```
db rotating_calipers(P a[],int n,P ans[]){//凸包a[1~n](逆时针)
    //返回凸包直径,ans[0~3]是最小矩形覆盖的四个点(逆时针)
    db d=0,S=1e100; a[0]=a[n];
    if(n==2)return dis(a[0],a[1]);//只有两个点的时候不能构成多边形
    int j=2,l=0,r=0;
    for(int i=0;i<n;++i){
        while(cmp(cross(a[i+1]-a[i],a[j]-a[i]),cross(a[i+1]-a[i],a[j+1]-a[i]))
            ==-1)j=(j+1)%n;
        while(cmp(dot(a[i+1]-a[i],a[r+1]-a[i]),dot(a[i+1]-a[i],a[r]-a[i]))>=0)r=(
            r+1)%n;
        if(i==0)l=r;
        while(cmp(dot(a[i+1]-a[i],a[l+1]-a[i]),dot(a[i+1]-a[i],a[l]-a[i]))<=0)l=(
            l+1)%n;
        d=max({d,dis(a[i],a[j]),dis(a[i+1],a[j])});//只求直径的话其他代码可以删除
        P v=a[i+1]-a[i];
        P A=point_proj_line(a[l],a[i],v);
        P B=point_proj_line(a[r],a[i],v);
        P C=point_proj_line(a[r],a[j],v);
        P D=point_proj_line(a[l],a[j],v);
        db now=dis(A,B)*dis(B,C);
        if(cmp(now,S)==-1)S=now,ans[0]=A,ans[1]=B,ans[2]=C,ans[3]=D;
    }
    return d;//d:凸包直径  S:最小矩形覆盖的矩形面积
}
```

10.10 半平面交

```
typedef long double db;
const db eps=1e-20,inf=1e20;
const int N=1e4+10;
int sgn(db x){return x<-eps?-1:x>eps;}
struct P{
    db x,y;
    P(db a=0,db b=0):x(a),y(b){}
```

```
    P operator + (P b){return P(x+b.x,y+b.y);}
    P operator - (P b){return P(x-b.x,y-b.y);}
    P operator * (db b){return P(x*b,y*b);}
    int up(){return sgn(y)>0||sgn(y)==0&&sgn(x)>0;}//用于极角牌序
}c[N],p[N];
db cross(P a,P b){return a.x*b.y-a.y*b.x;}
db dot(P a,P b){return a.x*b.x+a.y*b.y;}
db angle(P a,P b){return atan2(cross(a,b),dot(a,b));}
P line_int_line(P a,P u,P b,P v){
    db t=cross(b-a,v)/cross(u,v);
    return a+u*t;
}
struct line{
    P a,v;//直线a+kv,半平面是向量v左边的区域
    int id;//有时需要一个id
}a[N];
P line_int_line(line a,line b){
    return line_int_line(a.a,a.v,b.a,b.v);
}
void add(db A,db B,db C){//添加一个半平面Ax+By<=C
    if(sgn(B)!=0)a[++h]={P(0,C/B),P(-B,A)};
    else a[++h]={P(C/A,0),P(0,A)};
}
int n,h;
bool on_right(P p,line l){
    //条件为<0:重合的点代表的边不算;条件为<=0:重合的点代表的边算上
    return sgn(cross(l.v,p-l.a))<0;
}
bool cmp(line l1,line l2){//半平面向量极角排序
    int f1=l1.v.up(),f2=l2.v.up();
    if(f1!=f2)return f1>f2;
    int c=sgn(cross(l1.v,l2.v));
    if(c!=0)return c>0;
    return sgn(cross(l1.v,l2.a-l1.a))>=0;//方向向量相同时，左边的向量排在后
}
int half_plane_intersection(line a[],P p[],int n){
    //输入a[1~n]是半平面,返回h=r-l+1半平面a[1~h],交点p[1~h]
    //线a[i]与a[i+1<=h?i+1:1]的交点是p[i]
    //通常需要考虑添加边界a[++n]=line(P(0,inf),P(-1,0))等
    static line q[N];
    int l=1,r=0;
```

```
    sort(a+1,a+1+n,cmp);
    for(int i=1;i<=n;++i){
        while(i<n&&sgn(angle(a[i].v,a[i+1].v))==0)++i;
        while(r-l>=1&&on_right(p[r-1],a[i]))--r;
        while(r-l>=1&&on_right(p[l],a[i]))++l;
        q[++r]=a[i];
        if(r-l>=1)p[r-1]=line_int_line(q[r-1],q[r]);
    }
    while(r-l>=1&&on_right(p[r-1],q[l]))--r;
    if(r-l<=1)return 0;
    p[r]=line_int_line(q[l],q[r]);
    for(int i=l;i<=r;++i)a[i-l+1]=q[i];
    for(int i=l;i<=r;++i)p[i-l+1]=p[i];
    return r-l+1;
}
```

10.11 圆

```
struct circle{
    P c;db r;
    circle(P p=P(0,0),db x=1):c(p),r(x){}
    P point(db a){return P(c.x+r*cos(a),c.y+r*sin(a));}
};
//过点p做圆的切线
int circletan(P p,circle C,P v[]){
    P u=C.c-p; db d=u.len();
    if(sgn(d-C.r)==-1)return 0;
    if(sgn(d-C.r)==0){v[0]=u.rotate90();return 1;}
    db a=asin(C.r/d);
    v[0]=u.rotate(-a);
    v[1]=u.rotate(a);
    return 2;
}
//两圆共切线,返回切线条数,无数条返回-1, v1[i],v2[i]第i条切线在圆A,B上的切点
int circle_tan(circle A,circle B,P v1[],P v2[]){
    int cnt=0;
    if(cmp(A.r,B.r)==-1)swap(A,B),swap(v1,v2);
    db d2=sqdis(A.c,B.c);
    db rsub=A.r-B.r;
    db rsum=A.r+B.r;
    if(sgn(d2-rsub*rsub)==-1)return 0;//内含
```

```
    db a=atan2(B.c.y-A.c.y,B.c.x-A.c.x);
    if(!sgn(d2)&&!sgn(A.r-B.r))return -1;//重合,无数切线
    if(!sgn(d2-rsub*rsub)){//内切,找到切点
        v1[cnt]=A.point(a);
        v2[cnt]=B.point(a);
        ++cnt;return 1;
    }
    db b=acos((A.r-B.r)/sqrt(d2));
    v1[cnt]=A.point(a+b);v2[cnt++]=B.point(a+b);
    v1[cnt]=A.point(a-b);v2[cnt++]=B.point(a-b);
    if(!sgn(d2-rsum*rsum)){
        v1[cnt]=A.point(a);
        v2[cnt++]=B.point(pi+a);
    }else if(sgn(d2-rsum*rsum)==1){
        b=acos((A.r+B.r)/sqrt(d2));
        v1[cnt]=A.point(a+b);v2[cnt++]=B.point(pi+a+b);
        v1[cnt]=A.point(a-b);v2[cnt++]=B.point(pi+a-b);
    }
    return cnt;
}
//圆与直线交点,lerp(a,b,x1),lerp(a,b,x2)
bool circle_int_line(circle c,P a,P b,db &x1,db &x2){
    P d=b-a;
    db A=dot(d,d),B=dot(d,(a-c.c))*2.0,C=dot(a-c.c,a-c.c)-c.r*c.r;
    return solve_eqution(A,B,C,x1,x2);
}
//圆圆相交的交点
bool circle_int_circle(circle a,circle b,P &p1,P &p2){
    db d=(a.c-b.c).len();
    if(cmp(d,a.r+b.r)==1||cmp(d,abs(a.r-b.r))==-1)return 0;
    db l=((a.c-b.c).len2()+a.r*a.r-b.r*b.r)/(2*d);
    db h=sqrt(a.r*a.r-l*l);
    P vl=(b.c-a.c).strech(l),vh=vl.rotate90().strech(h);
    p1=a.c+vl+vh;p2=a.c+vl-vh;
    return 1;
}
//圆和三角形o+a,o+b,o交的面积，o是圆心
db circle_int_triangle(circle c,P a,P b) {
    if(sgn(cross(a-c.c,b-c.c))==0)return 0;
    P q[5];int cnt=0;
    db t0,t1; q[cnt++]=a;
```

```
65      if( circle_int_line(c,a,b,t0,t1) ) {
66          if(0<=t0&&t0<=1)q[cnt++]=lerp(a,b,t0);
67          if(0<=t1&&t1<=1)q[cnt++]=lerp(a,b,t1);
68      }
69      q[cnt++]=b;db s=0;
70      for(int i=1;i<cnt;++i) {
71          P z=(q[i-1]+q[i])*0.5;
72          if((z-c.c).len2()<=c.r*c.r)
73          s+=abs(cross(q[i-1]-c.c,q[i]-c.c))*0.5;
74          else
75          s+=c.r*c.r*get_angle(q[i-1]-c.c,q[i]-c.c)*0.5;
76      }
77      return s;
78  }
79  //圆与多边形相交的面积
80  db circle_int_polygon(circle C,P p[],int n) {
81      db s = 0;p[n+1]=p[1];
82      for(int i=1;i<=n;++i)
83      s+=circle_int_triangle(C,p[i],p[i+1])*sgn(cross(p[i]-C.c,p[i+1]-C.c));
84      return abs(s);
85  }
86  db circle_cover(vector<P>v,P &o){//最小圆覆盖,不得有重复点,时间复杂度O(N)
87      random_shuffle(v.begin(),v.end());
88      db r2=0;
89      for(int i=0;i<v.size();++i){
90          if((v[i]-o).len2()>r2){
91              o=v[i];r2=0;
92              for(int j=0;j<i;++j){
93                  if((v[j]-o).len2()>r2){
94                      o=(v[i]+v[j])*0.5;  r2=(v[j]-o).len2();
95                      for(int k=0;k<j;++k){
96                          if((v[k]-o).len2()>r2){
97                              o=line_int_line((v[i]+v[j])*0.5,(v[i]-v[j]).rotate90
                                    (),(v[i]+v[k])*0.5,(v[i]-v[k]).rotate90());
98                              r2=(v[k]-o).len2();
99                          }
100                     }
101                 }
102             }
103         }
104     }
```

```
    return sqrt(r2);
}
```

10.12 公式

海伦公式：三角形三边为 a,b,c，令 $p=\frac{a+b+c}{2}, S=\sqrt{p(p-a)(p-b)(p-c)}$

球缺：半径为 r 的球被切除高位 h 的球缺，剩下体积 $V=\frac{h^2(3r-h)\pi}{3}$，若球缺高为为 h，截面直径为 d，则 $V=\frac{h(3d^2+4h^2)\pi}{24}$

平面图欧拉公式：$V-E+F=C+1$，$V=$ 顶点数，$E=$ 边数，$F=$ 区域数，$C=$ 连通分量数

10.13 平面最近点对

乱搞法：随机旋转角度，按 x 排序/按 $x\times y$ 排序，每个点枚举附近 100 个

$O(n\log^2 n)$ 先按 x 排序，再归并排序 y，取出横坐标离 x 中点近的统计答案

```
struct P{db x,y;}a[N],b[N],tmp[N];
int n,h;db ans=1e100;
bool cmpx(P a,P b){return a.x<b.x;}
db dis(P a,P b){return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}
void merge(int l,int r){
    if(l>=r)return;
    int mid=l+r>>1;db midx=a[mid].x;
    merge(l,mid);merge(mid+1,r);
    for(int p=l,q=mid+1,i=l;i<=r;++i){
        if(p>mid||q<=r&&a[q].y<a[p].y)tmp[i]=a[q++];
        else tmp[i]=a[p++];
    }
    for(int i=l;i<=r;++i)a[i]=tmp[i];
    h=0;
    for(int i=l;i<=r;++i)if(abs(a[i].x-midx)<ans)b[++h]=a[i];
    for(int i=1,k=1;i<h;++i){
        while(k<h&&b[k+1].y-b[i].y<ans)++k;
        for(int j=i+1;j<=k;++j){
            ans=min(ans,dis(b[i],b[j]));
        }
    }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i)scanf("%lf%lf",&a[i].x,&a[i].y);
    sort(a+1,a+1+n,cmpx);
```

```
    merge(1,n);
    printf("%.0lf",ans*ans);
}
```

11 博弈

通常可以考虑最终状态，最后一步必胜策略，递归子问题等等

11.1 必败态

必胜态：存在一个后继状态是必败态

必败态：不存在一个后继状态是必败态

复杂状态：当前状态根据后继状态的最坏情况来判断。比如有输、赢、平局，后继最差是输，当前就是赢，后继最差是平局，当前就是平局，后继最差是赢，当前就是输

11.2 SG 函数

必败状态的 $SG = 0$

一个状态的 SG 函数值等于其后继状态 SG 值的 mex, 即 $SG(x) = mex_{x \to y}(SG(y))$

多个组合博弈游戏的和的 SG 值等于其所有子游戏 SG 值的异或和

```
int mex(vector<int>& v) { // O(n) mex
    int n = v.size();
    for (int i = 0; i < n; ++i) {
        while (v[i] >= 0 && v[i] < n && v[i] != v[v[i]]) {
            swap(v[i], v[v[i]]);
        }
    }
    for (int i = 0; i < n; ++i) {
        if (v[i] != i) {
            return i;
        }
    }
    return n;
}
int mex(vector<int>&v){ // O(nlogn) mex
    sort(v.begin(),v.end());
    int ans=0;
    for(int x:v){
        if(x==ans)++ans;
        else if(x>ans)return ans;
```

```
    }
    return ans;
}
```

11.3 反 SG 游戏

最先不能操作的人获胜

必胜：$SG \neq 0$ 且至少有一个子游戏的 $SG > 1$，或 $SG = 0$ 且每一个子游戏的 $SG \leq 1$

否则必败

11.4 巴什博奕

n 个石头，每次可以拿 1 到 m 个，不能拿的失败，$SG = n\%(m+1)$

11.5 Nim 游戏

n 堆石子，第 i 堆又 a_i 个，两人轮流操作，每次选一堆石头并拿走任意个，先不能拿的失败

相当于 n 个子游戏，第 i 堆石子的 SG 值是 a_i，总游戏的 $SG = xor_{i=1}^{n} a_i$

$SG \neq 0$ 时先手拿最多石头的方案：$O(n)$ 枚举 i，a_i 应该变为 $xorsum \oplus a_i$

必败态只拿一个，并让对手也只能只拿一个的方案：$O(logC)$ 枚举所有 $lowbit = lb$，拿一个之后 $xorsum$ 会变为 $xorsum \oplus lb \oplus (lb-1)$，并 $O(n)$ 判断之后状态对手是否最多拿一个

11.6 阶梯 Nim

每次选某一堆，拿走任意个并放到前一堆，第 0 堆不能拿，即 $a[i-1]+=x, a[i]-=x$

后手可以让先手的偶数堆无效：当 A 把 $2k$ 堆拿出 x 个，B 就可以从 $2k-1$ 堆拿出 x 个

相当于只看奇数堆的 nim 游戏，设先手是 A

当 A 的奇数堆 nim 游戏必败时，B 会使 A 的偶数堆无效

当 A 的奇数堆 nim 游戏必胜时，A 会先把奇数堆 nim 游戏变为必败，再使 B 的偶数堆无效

11.7 Nim-k

每次选 k 堆，在这 k 堆拿走分别任意个（可以不同），先不能拿的失败

若二进制每一位的出现次数都是 k+1 的倍数则必败，否则必胜

11.8 威佐夫博弈

有两堆物品，两人轮流操作，每次操作从一堆中取任意个，或从两堆中取相同多个，先不能操作的人失败

必败态为：$(1,2),(3,5),(4,7),(6,10)...(\lfloor \frac{k(1+\sqrt{5})}{2} \rfloor, \lfloor \frac{k(1+\sqrt{5})}{2} \rfloor + k)$

每个 (a_i, b_i) 中 $a_i = mex(\{0, a_1, b_1, a_2, b_2, ...a_{i-1}, b_{i-1}\})$

11.9 树上删边游戏

给定一棵有根树，两人轮流操作，每次操作选择一条边，删掉该边以及与根不连通的所有点和边（整棵子树），不能操作的人时失败，则叶子的 $SG = 0$，非叶子的 $SG(x) = xor_{y \in son(x)}[SG(y) + 1]$

多棵竹子的删边游戏等价于 nim 游戏，等价于这些竹子的根合并为一个点后的树上删边游戏

克朗原理: 一个结点的所有儿子的子树都是竹子，则该节点子树可以等价于一根竹子，长度为各分支长度的异或和

11.10 无向图删边游戏

给定一张无向图，定义一些点在地上，每次选择一条边，删掉该边以及与所有地板上的点不连通的点和边，不能操作的人失败

费森原理：偶环可以直接缩成一个点（类似边双连通分量缩点），奇环缩成一个点并额外挂着一条长为 1 的竹子

12 基础算法

12.1 二分查找

构造单调 *bool* 函数 $check(x)$, 返回值为小的全 1，大的全 0，形如 11...1100...00

可以用二分查找算出最大的 x 使得 $check(x) = 1$，反过来也同理

```
//第k小: <=x的个数>=k的最小的x
//第k大: >=x的个数>=k的最大的x
while(l<=r){
    int mid=l+r>>1;// l+(r-l>>1)
    if(check(mid))ans=mid,l=mid+1;
    else r=mid-1;
}

vector<int>v;//求vector中[l,r]区间有几个数字
int cnt=upper_bound(v.begin(),v.end(),r)-lower_bound(v.begin(),v.end(),l);
```

12.2 去重, 离散化

```
int main(){
    int a[N];//数组离散化
    sort(a+1,a+1+h);//a[1...h]中包括有所有可能的数字
    h=unique(a+1,a+1+h)-a-1;
```

```
    x=lower_bound(a+1,a+1+h,x)-a;//将x离散化后的值
    vector<int>v;//vector离散化
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
    x=lower_bound(v.begin(),v.end(),x)-v.begin()+1;
}
```

12.3 取出第 k 小/中位数

```
int a[N];
nth_element(a+1,a+k,a+1+n);//O(N)
//未排序,但保证a[1]~a[k-1]比a[k]小,a[k+1]~a[n]比a[k]大
```

12.4 bitset

```
int main(){   bitset<100>a;
    a.set();/*全设为1*/ a.reset();/*全设为0*/
    a[2]=1;/*访问下标*/ cout<<a<<endl;//打印bitset,a[0]在最右边
    a.count();/*1的个数*/a.any();/*有1?1:0*/a.flip();//全部取反
}
```

12.5 builtin 函数

```
__builtin_popcount(x);//二进制1的个数
__builtin_popcountll(x);//ll表示参数为long long,后面几个函数也可以加ll
__buitlin_clz(x);//二进制前导零个数
__builtin_ctz(x);//二进制后导零个数
__builtin_parity(x);//二进制1的个数是奇数?1:0
__builtin_ffs(x);//二进制最后一个1在第几位,从1开始,ffs(1)=1,ffs(0)=0
__builtin_sqrt(x);//x位double,开根号,比sqrt(x)快
```

12.6 归并排序求逆序对

```
int a[N],temp[N];ll ans;
void merge(int l,int r){
    if(l>=r)return;
    int mid=l+r>>1;
    merge(l,mid);merge(mid+1,r);
    int p=l,q=mid+1;
```

```
    for(int i=l;i<=r;++i){
        if(p>mid||q<=r&&a[q]<a[p]){//左边用完了or右边没用完and右边小
            ans+=mid-p+1;//逆序对个数
            temp[i]=a[q++];
        }
        else temp[i]=a[p++];
    }
    for(int i=l;i<=r;++i)a[i]=temp[i];
}
```

12.7 计数排序

值域较小时适用，时间复杂度为 $O(N+C)$

```
int n,a[N],cnt[N],C;//值域大小为[1,C]
void count_sort(int a[],int n){//计数排序a[1~n]
    for(int i=1;i<=n;++i)++cnt[a[i]];
    int h=0;
    for(int i=1;i<=C;++i){
        while(cnt[i]--)a[++h]=i;
    }
}
int rk[N];
void count_sort_rank(int a[],int n){//求出rk[i]=a[i]是第几小
    for(int i=1;i<=n;++i)++cnt[a[i]];
    for(int i=1;i<=C;++i)cnt[i]+=cnt[i-1];
    for(int i=1;i<=n;++i)rk[i]=cnt[a[i]]--;
}
```

12.8 基数排序

```
const int N=1e5+5,base=(1<<15)-1;
int n,a[N],cnt[base+1],id[N],t[N];//id[i]=第i小的位置
void radix_sort(int a[],int n){//int数组a[1~n],值域[0,2^30-1]
    for(int i=1;i<=n;++i)++cnt[a[i]&base];
    for(int i=1;i<=base;++i)cnt[i]+=cnt[i-1];
    for(int i=n;i;--i)id[cnt[a[i]&base]--]=i;
    memset(cnt,0,sizeof cnt);
    for(int i=1;i<=n;++i)++cnt[a[i]>>15],t[i]=id[i];
    for(int i=1;i<=base;++i)cnt[i]+=cnt[i-1];
    for(int i=n;i;--i)id[cnt[a[t[i]]>>15]--]=t[i];
```

```
    for(int i=1;i<=n;++i)t[i]=a[id[i]];
    for(int i=1;i<=n;++i)a[i]=t[i];
}
```

12.9 维护区间并集

```
map<int,int>mp;//对于区间[l,r] : mp[l]=r
// [1,2]和[3,4]不能合并版本
void insert(ll l,ll r){
    auto it=mp.upper_bound(l);
    if(it!=mp.begin()){
        --it;
        if(it->second>=l){
            l=it->first;
            r=max(r,it->second);
            mp.erase(it);
        }
    }
    for(it=mp.lower_bound(l);it!=mp.end()&&it->first<=r;it=mp.erase(it))
        r=max(r,it->second);
    mp[l]=r;
}
// [1,2]和[3,4]能合并为[1,4]版本
void insert(ll l,ll r){
    auto it=mp.upper_bound(l);
    if(it!=mp.begin()){
        --it;
        if(it->second>=l-1){
            l=it->first;
            r=max(r,it->second);
            mp.erase(it);
        }
    }
    for(it=mp.lower_bound(l);it!=mp.end()&&it->first<=r+1;it=mp.erase(it))
    r=max(r,it->second);
    mp[l]=r;
}
```

12.10 计算天数

```
int isleap(int x){return x%4==0&&x%100!=0||x%400==0;}
int month[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
int calcday(int y1,int m1,int d1,int y2,int m2,int d2){
    //从y1/m1/d1到y2/m2/d2过了几天
    int sum=0;
    if(y1==y2){
        if(m1==m2)return d2-d1;
        month[2]=isleap(y1)?29:28;
        sum=month[m1]-d1;
        for(int i=m1+1;i<m2;++i)sum+=month[i];
        sum+=d2;
    }
    else{
        month[2]=isleap(y1)?29:28;
        sum=month[m1]-d1;
        for(int i=m1+1;i<=12;++i)sum+=month[i];
        for(int i=y1+1;i<y2;++i)sum+=isleap(i)?366:365;
        month[2]=isleap(y2)?29:28;
        for(int i=1;i<m2;++i)sum+=month[i];
        sum+=d2;
    }
    return sum;
}
```

蔡勒公式:

$c=\lfloor\frac{year}{100}\rfloor=$ 年份前两位,$y=year\%100=$ 年份后两位,$m=$ 月,$d=$ 日

$w=(\lfloor\frac{c}{4}\rfloor-2c+y+\lfloor\frac{y}{4}\rfloor+\lfloor\frac{13(m+1)}{5}\rfloor+d-1)\ mod\ 7$

$year$ 年 m 月 d 日是星期 w，($w=0$ 是星期日)

12.11 表达式求值

```
inline ll fun(ll x,ll y,char c){
    switch(c){
        case '+':return x+y;
        case '-':return x-y;
        case '*':return x*y;
        case '/':{
            if(y==0)throw -1;//div 0
            if(x%y!=0)throw 1;//不能整除
            return x/y;
        }
```

```
        case '%':{
            if(y==0)throw -1;// div 0
            return x%y;
        }
    }
}
inline ll calc(string s){//"1+2++-3*4+-(3+4*1/2+4+5)"
    s="("+s+")";
    vector<ll>num,flag({1});
    vector<char>op;
    for(unsigned i=0;i<s.size();++i){
        if(isdigit(s[i])){
            ll x=s[i]-'0';
            while(i+1<s.length()&&isdigit(s[i+1]))x=x*10+s[i+1]-'0',++i;
            if(flag.back()==-1)x=-x;
            flag.pop_back();
            num.push_back(x);
        }
        else{
            if(s[i]=='+'||s[i]=='-'){
                if(num.size()!=op.size()){
                    while(!op.empty()&&op.back()!='('){
                        ll x=num.back();num.pop_back();
                        char c=op.back();op.pop_back();
                        try{
                            num.back()=fun(num.back(),x,c);
                        }
                        catch(int x){
                            throw x;
                        }
                    }
                    op.push_back(s[i]);
                    flag.push_back(1);
                }
                else if(s[i]=='-')flag.back()=-flag.back();
            }
            else if(s[i]=='*'||s[i]=='/'||s[i]=='%'){
                if(!op.empty()&&op.back()!='('){
                    if(op.back()=='+'||op.back()=='-'){
                        op.push_back(s[i]);
                    }
```

```
52                  else{// * / %
53                      int x=num.back();num.pop_back();
54                      char c=op.back();op.pop_back();
55                      try{
56                          num.back()=fun(num.back(),x,c);
57                      }
58                      catch(int x){
59                          throw x;
60                      }
61                      op.push_back(s[i]);
62                  }
63              }
64              else op.push_back(s[i]);
65              flag.push_back(1);
66          }
67          else if(s[i]=='('){
68              num.push_back(0);op.push_back('(');
69              flag.push_back(1);
70          }
71          else if(s[i]==')'){
72              while(!op.empty()&&op.back()!='('){
73                  ll y=num.back();num.pop_back();
74                  char c=op.back();op.pop_back();
75                  try{
76                      num.back()=fun(num.back(),y,c);
77                  }
78                  catch(int x){
79                      throw x;
80                  }
81              }
82              ll x=num.back();num.pop_back();
83              if(flag.back()==-1)x=-x;
84              flag.pop_back();
85              num.back()=x;op.pop_back();
86          }
87          else{
88              throw 2;//不合法运算符
89          }
90      }
91  }
92  return num.front();
```

```
}
```

13 模拟游戏

13.1 算 24 点

```
int a[5],op[4];//options: 1+ 2- 3* 4/
const int goal=24;
double f(double a,double b,int op){
    switch(op){
        case 0:return a+b;
        case 1:return a-b;
        case 2:return a*b;
        default:
        if(abs(b)<1e-6)throw -1;
        return a/b;
    }
}
int g(int a,int b,int op){
    switch(op){
        case 0:return a+b;
        case 1:return a-b;
        case 2:return a*b;
        default:
        if(b==0||a%b!=0)throw -1;
        return a/b;
    }
}
const char *ch="+-*/";
bool find_double_one(){
    sort(a+1,a+1+4);
do{
    for(int i=0;i<4;++i)
    for(int j=0;j<4;++j)
    for(int k=0;k<4;++k){
        double s;
        try{s=f(f(f(a[1],a[2],i),a[3],j),a[4],k);}
        catch(...){s=-1;}
        if(abs(s-goal)<1e-6){
        printf("((%d%c%d)%c%d)%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
            return 1;
```

```
36          }
37
38          try{s=f(f(a[1],a[2],i),f(a[3],a[4],k),j);}
39          catch(...){s=-1;}
40          if(abs(s-goal)<1e-6){
41          printf("(%d%c%d)%c(%d%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
42              return 1;
43          }
44
45          try{s=f(f(a[1],f(a[2],a[3],j),i),a[4],k);}
46          catch(...){s=-1;}
47          if(abs(s-goal)<1e-6){
48          printf("(%d%c(%d%c%d))%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
49              return 1;
50          }
51
52          try{s=f(a[1],f(a[2],f(a[3],a[4],k),j),i);}
53          catch(...){s=-1;}
54          if(abs(s-goal)<1e-6){
55          printf("%d%c(%d%c(%d%c%d))\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
56              return 1;
57          }
58
59          try{s=f(a[1],f(f(a[2],a[3],j),a[4],k),i);}
60          catch(...){s=-1;}
61          if(abs(s-goal)<1e-6){
62          printf("%d%c((%d%c%d)%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
63              return 1;
64          }
65      }
66  }while(next_permutation(a+1,a+5));
67      return 0;
68  }
69  bool find_int_one(){
70      sort(a+1,a+1+4);
71  do{
72      for(int i=0;i<4;++i)
73      for(int j=0;j<4;++j)
74      for(int k=0;k<4;++k){
75          int s=g(g(g(a[1],a[2],i),a[3],j),a[4],k);
76          if(s==goal){
```

```
77          printf("((%d%c%d)%c%d)%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
78              return 1;
79          }
80          s=g(g(a[1],a[2],i),g(a[3],a[4],k),j);
81          if(s==goal){
82          printf("(%d%c%d)%c(%d%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
83              return 1;
84          }
85          s=g(g(a[1],g(a[2],a[3],j),i),a[4],k);
86          if(s==goal){
87          printf("(%d%c(%d%c%d))%c%d\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
88              return 1;
89          }
90          s=g(a[1],g(a[2],g(a[3],a[4],k),j),i);
91          if(s==goal){
92          printf("%d%c(%d%c(%d%c%d))\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
93              return 1;
94          }
95          s=g(a[1],g(g(a[2],a[3],j),a[4],k),i);
96          if(s==goal){
97          printf("%d%c((%d%c%d)%c%d)\n",a[1],ch[i],a[2],ch[j],a[3],ch[k],a[4]);
98              return 1;
99          }
100     }
101 }while(next_permutation(a+1,a+5));
102     return 0;
103 }
```

13.2 德州扑克

判断牌型

```
1  struct poker{
2      int a,b;//a=牌的数值(必须为2~14),b=牌的花色(1~4),也可以是任意四个不同的数
3  };
4  int hand_type(poker v[]){//判断v[0~4]这五张卡的牌型
5      static int a[5],s;
6      for(int i=0;i<5;++i)a[i]=v[i].a;
7      sort(a,a+5);
8      bool straight=1,flush=1;//是顺子 / 是同花
9      for(int i=1;i<5;++i){
10         if(a[i]-a[0]!=i)straight=0;
```

```
        if(v[i].b!=v[0].b)flush=0;
    }
    if(a[0]==2&&a[1]==3&&a[2]==4&&a[3]==5&&a[4]==14)straight=1;//特判A2345
    if(straight&&flush){
        if(a[4]==14&&a[3]==13)s=10;//royal flush 皇家同花顺(10JQKA)
        else s=9;//straight flush 同花顺
    }
    else if(straight)s=5;//straight 顺子
    else if(flush)s=6;//flush 同花
    else{
        int cnt[15]={0};
        int mx=0;//出现次数最多的牌的
        for(int i=0;i<5;++i)++cnt[a[i]],mx=max(mx,cnt[a[i]]);
        if(mx==4)s=8;//four of a kind 四条(炸弹)
        else if(mx==3){
            int f2=0;
            for(int i=0;i<=4;++i){
                if(cnt[a[i]]==2)f2=1;
            }
            if(f2)s=7;//full house 葫芦(三带二)
            else s=4;//three of a kind 三条(三带一)
        }
        else if(mx==2){
            int c=0;
            for(int i=0;i<=4;++i){
                if(cnt[a[i]]==2)++c;
            }
            if(c==4)s=3;//two pairs 两对
            else s=2;//pair 一对
        }
        else s=1;//high card 高牌(散牌)
    }
    return s;
}
```

相同牌型比大小

```
int cmp_same_type(poker c[],poker d[],int id){//双方牌型均为id,比大小
    static int a[5],b[5];
    for(int i=0;i<5;++i)a[i]=c[i].a,b[i]=d[i].a;
    switch(id){
        case 5://straight 顺子
```

```
6         case 9://straight flush 同花顺
7         {
8             sort(a,a+5);sort(b,b+5);
9             int u=a[4],v=b[4];
10            if(a[0]==2&&a[4]==14)u=5;
11            if(b[0]==2&&b[4]==14)v=5;
12            if(u!=v)return u>v?1:-1;
13            return 0;
14        }
15        case 1://highcard 高牌
16        case 6://flush 同花
17        {
18            sort(a,a+5);sort(b,b+5);
19            for(int i=4;i>=0;--i){
20                if(a[i]!=b[i])return a[i]>b[i]?1:-1;
21            }
22        }
23        case 10://royal flush 皇家同花顺
24            return 0;
25        default:
26        //      case 2://pair 一对
27        //      case 3://two pairs 两对
28        //      case 4://three of a kind 三条
29        //      case 7://full house 葫芦
30        //      case 8://four of a kind 四条
31        {
32            int cnt1[15]={0},cnt2[15]={0};
33            static int t1[5],t2[5];
34            int h1=0,h2=0,p1=4,p2=4,mx=0;
35            for(int i=0;i<5;++i){
36                ++cnt1[a[i]];++cnt2[b[i]];
37                mx=max(mx,cnt1[a[i]]);
38            }
39            for(int i=0;i<5;++i){
40                if(cnt1[a[i]]==mx)t1[p1--]=a[i];
41                else t1[h1++]=a[i];
42                if(cnt2[b[i]]==mx)t2[p2--]=b[i];
43                else t2[h2++]=b[i];
44            }
45            if(id==2||id==3)sort(t1,t1+h1),sort(t2,t2+h2);
46            if(id==3)sort(t1+h1,t1+5),sort(t2+h2,t2+5);
```

```
            for(int i=4;i>=0;--i){
                if(t1[i]!=t2[i])return t1[i]>t2[i]?1:-1;
            }
            return 0;
        }
    }
}
```

全手牌比大小

```
int cmp_hand(poker a[],poker b[]){//五张手牌a[0~4],b[0~4]比大小,返回 -1:<  0:=  1:>
    int u=hand_type(a),v=hand_type(b);
    if(u!=v)return u>v?1:-1;//先比牌型
    return cmp_same_type(a,b,u);//同牌型再比牌大小
}
```

结算筹码

```
void calc_money(int n,int rank[],long long c[],long long ans[]){
    //数组下标 1~n,rank[i]=第i个人的排名是第几大,rank[i]=1是最大的,可能存在并列
    //c[i]=第i个人出的筹码     ans[i]=结算后第i个人获得的筹码
    static int id[N];
    for(int i=1;i<=n;++i)id[i]=i;
    sort(id+1,id+1+n,[&](int i,int j){return c[i]<c[j];});
    map<int,int>cnt;//cnt[i]=排名为i的人数
    for(int i=1;i<=n;++i)++cnt[rank[i]];
    int rk1=cnt.begin()->first;
    long long now=0,sub=0;
    for(int i=1;i<=n;++i){
        int t=id[i];
        long long x=c[t]-sub;
        now+=x*(n+1-i)/cnt.begin()->second;
        ans[t]=rk1==rank[t]?now:0;
        if(!--cnt[rank[t]]){
            cnt.erase(rank[t]);
            if(rk1==rank[t]&&cnt.size()){
                rk1=cnt.begin()->first;
                now=0;
            }
        }
        sub+=x;
    }
```

```
}
```

14 其他技巧

14.1 C++ 编译命令

```
DEV C++使用C++11 工具->编译选项
编译时加入以下命令(√):-std=c++11
debug:工具->编译选项->代码生成/优化->产生调试信息YES
无限栈空间: #pragma comment(linker, "/STACK:102400000,102400000")
或在编译命令中加入 -Wl,--stack=102400000
```

14.2 快读快写

```
ll rd(){
    char c=getchar();ll x=0;bool f=1;
    for(;c<'0'||c>'9';c=getchar())f|=c=='-';
    for(;c>='0'&&c<='9';c=getchar())x=(x<<3)+(x<<1)+(c^48);
    return f?-x:x;
}
template<typename T>bool read(T &s){
    ll x=0;bool f=0;char c=getchar();
    for(;c<'0'||c>'9';c=getchar()){
        if(c==EOF)return 0;f|=c=='-';
    }
    for(;c>='0'&&c<='9';c=getchar())x=(x<<3)+(x<<1)+(c^48);
    if(c!='.')s=f?-x:x;
    else{
        double y=0,k=0.1;
        for(c=getchar();c>='0'&&c<='9';c=getchar())y+=(c^48)*k,k*=0.1;
        s=f?-x-y:x+y;
    }
    return 1;
}
void wt(ll x){
    static char a[44];short h=0;
    if(x<0)x=-x,putchar('-');
    do{ a[++h]='0'+x%10;x/=10;}while(x);
    for(int i=h;i;--i)putchar(a[i]);
}
```

```
27 void wtl(ll x){wt(x);putchar('\n');}
28 void wtb(ll x){wt(x);putchar(' ');}
```

fread/fwrite，若不需要输出优化，也可以删掉所有输出相关函数并使用 printf/cout

```
1  struct FastIO{//不可与scanf,printf,getchar,putchar,gets,puts,cin,cout混用
2      private:
3      static const int BUFSIZE=1e5;//BUFSIZE不需要改
4      char buf[BUFSIZE];int pos,len;//读入buffer(缓冲器)以及读入指针
5      char wbuf[BUFSIZE];int wpos;//输出buffer以及输出指针
6      #define gc() (pos<len||(len=(pos=0)+fread(buf,1,BUFSIZE,stdin))?buf[pos++]:
           EOF)
7      #define pc(c) (wpos<BUFSIZE?wbuf[wpos++]=c:(fwrite(wbuf,1,BUFSIZE,stdout),
           wbuf[(wpos=0)++]=c))
8      public:
9      FastIO():wpos(0),pos(0),len(0){}
10     ~FastIO(){fwrite(wbuf,1,wpos,stdout),wpos=0;}
11     char getc(){return gc();}//读取char
12     void putc(char c){pc(c);}//输出字符
13     long long rd(){//读取long long
14         long long x=0;char c=gc();bool f=0;
15         while(c<'0'||c>'9')f|=c=='-',c=gc();
16         while(c>='0'&&c<='9')x=(x<<3)+(x<<1)+(c^48),c=gc();
17         return f?~x+1:x;
18     }
19     template<typename T>bool read(T &x){//多测读整数while(io.read(n))work();本地
           测试请输入两次ctrl Z
20         x=0;char c=gc();bool f=0;
21         while(c<'0'||c>'9'){if(c==EOF)return 0;f|=c=='-';c=gc();}
22         while(c>='0'&&c<='9')x=(x<<3)+(x<<1)+(c^48),c=gc();
23         if(f)x=-x; return 1;
24     }
25     template<typename T>void wt(T x){//输出整数
26         static char a[22];
27         if(x<0)pc('-'),x=-x;int h=0;
28         do{a[++h]='0'+x%10;x/=10;}while(x);
29         while(h)pc(a[h--]);
30     }
31     template<typename T>void wtl(T x){wt(x);pc('\n');}//write line输出整数并换行
32     template<typename T>void wtb(T x){wt(x);pc(' ');}//write blank输出整数并空格
33     int gets(char *s){int l=0;char c=gc();while(c<=' ')c=gc();while(c>' ')s[l++]=
           c,c=gc();return s[l]=0,l;}//读取字符串
```

```
34      int getline(char *s){int l=0;char c=gc();while(c!=EOF&&c!='\n')s[l++]=c,c=gc
            ();return s[l]=0,l;}//读取一行
35      void puts(const char *s){const char *p=s;while(*p)pc(*p++);}//输出字符串 (不
            带换行)
36      template<typename T>FastIO & operator >> (T &a){return read(a),*this;}//io>>a
            >>b;只能输入整数
37      template<typename T>FastIO & operator << (T a){return wtb(a),*this;}//io<<a<<
            b;输出整数并带有空格
38      #undef gc
39      #undef pc
40  } io;//本地测试出入结束后请输入一次ctrl Z
```

14.3 对拍

```
1   查看是否配置g++编译环境:
2   右键此电脑,属性,高级系统设置,环境变量,下面的系统变量,找到PATH,选中,编辑
3   新建一条:......\Dev-Cpp\MinGW64\bin
4   新建.txt文件,后缀名改为.bat,也可以右键.bat文件->编辑
5   前三句编译命令可以考虑不加,并用DEV进行编译
6   //  g++ -std=c++11 rand.cpp -o rand.exe
7   //  g++ -std=c++11 std.cpp -o std.exe
8   //  g++ -std=c++11 tested.cpp -o tested.exe
9       :loop
10      rand.exe >in.txt
11      std.exe <in.txt >out.txt
12      tested.exe <in.txt >tested.txt
13      fc out.txt tested.txt
14      if not errorlevel 1 goto loop
15      pause
16      goto loop
```

14.4 pbds 库

头文件

```
1   #include <ext/pb_ds/tree_policy.hpp>
2   #include <ext/pb_ds/assoc_container.hpp>
3   using namespace __gnu_pbds;
```

堆

```
1   #include <ext/pb_ds/priority_queue.hpp>
```

```
__gnu_pbds::priority_queue<int,greater<int>> Q;
```

14.5 指令集优化

```
#pragma GCC optimize("Ofast,no-stack-protector,unroll-loops,fast-math")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4.1,sse4.2,avx,avx2,popcnt,tune=native")
#include <immintrin.h>
#include <emmintrin.h>
int main() {
    __m256i a, b, c; int p[8];//a[0~3],a[i]=第i个64位代表的long long
    a=_mm256_set_epi64x(1ll,2ll,3ll,4ll);//a[0]=4,a[3]=1
    a=_mm256_setr_epi64x(1ll,2ll,3ll,4ll);//a[0]=1,a[3]=4
    b=_mm256_set1_epi64x(3);//b[0]=b[1]=b[2]=b[3]=3ll
    c=_mm256_add_epi64(a,b);
    for(int i=0;i<4;++i)printf("%lld ",c[i]);
    a=_mm256_setr_epi32(1,2,3,4,5,6,7,8);
    b=_mm256_set1_epi32(1);//每32位赋值成一个int
    c=_mm256_add_epi32(a,b);
    for(int i=0;i<4;++i)p[i<<1]=c[i],p[i<<1|1]=c[i]>>32;//p[0~7]={2,3,..,9}
}
```