比赛相关

编译选项

```
g++ code.cpp -o code -g -std=gnu++20 -Wall -Wextra -Wshadow -pedantic -Wformat=2
-fno-ms-extensions '-Wl,--stack=214749647' -fsanitize=undefined -fsanitize-
undefined-trap-on-error -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
```

读入、输出优化

```
#include<iostream>
#include<cstdio>
using namespace std;
class Fast_char
{
private:
    static const int SIZE=1<<21;
    char fr[SIZE],*pr1=fr,*pr2=fr;
    char fw[SIZE];
    int pw;
public:
    char getc()
    {
        return (pr1==pr2&&(pr2=(pr1=fr)+fread(fr,1,SIZE,stdin),pr1==pr2)?
EOF:*pr1++);
    }
    void putc(char ch)
    {
        (pw<SIZE?fw[pw++]=(ch):(fwrite(fw,1,SIZE,stdout),fw[(pw=0)++]=(ch)));
        return;
    }
    void flush()
    {
        fwrite(fw,1,pw,stdout);
        fflush(stdout);
        pw=0;
        return;
    }
    ~Fast_char()
    {
        flush();
    }
}fast_char;
#define getchar fast_char.getc
#define putchar fast_char.putc
template<typename T>
inline T read(T &x)
{
    char ch;
    bool flag=false;
```

```
    x=0;
    for(ch=getchar();ch<'0'||ch>'9';ch=getchar())
        if(ch=='-') flag=true;
    while(ch>='0'&&ch<='9')
        x=x*10+ch-'0',ch=getchar();
    if(flag) x=-x;
    return x;
}
template<typename T>
void write(T x)
{
    if(x<0) putchar('-'),x=-x;
    if(x>9) write(x/10);
    putchar(x%10+'0');
    return;
}
template<typename T>
void writeln(T x)
{
    write(x),putchar('\n');
    return;
}
char* getstring(char *s)
{
    char ch;
    char *t=s;
    for(ch=getchar();ch!='\n'&&ch!=EOF;ch=getchar()) *t=ch,t++;
    *t='\0';
    return s;
}
int putstring(char *s)
{
    int len=0;
    for(char *t=s;*t!='\0';t++)
        putchar(*t),len++;
    putchar('\n');
    return len;
}
#define gets getstring
#define puts putstring
```

数学

二进制集合操作

枚举子集

```
// 升序遍历 S 的子集
for(int T=0;;T=((T^S)-1)&S^S)
{
    ...
    if(T==S) break;
}

// 降序遍历 S 的子集
for(int T=S;;T=(T-1)&S)
{
    ...
    if(T==0) break;
}
```

枚举超集

```
// 升序遍历 S 的超集
for(int T=S;;T=(T+1)|S)
{
    ...
    if(T==U) break;
}

// 降序遍历 S 的超集
for(int T=U;;T=((T^S)-1)&(U^S)^S)
{
    ...
    if(T==S) break;
}
```

快速幂

```
int qpow(int a,int b)
{
    int res=1;
    while(b)
    {
        if(b&1) res=(long long)res*a%MOD;
        a=(long long)a*a%MOD,b>>=1;
    }
    return res;
}
```

光速幂

```
template<int MOD>
struct Fastest_Pow
```

```
{
    static constexpr int S=sqrt(MOD)+1;
    int x,pw[S+5],pws[MOD/S+5];
    void init(int x)
    {
        pw[0]=1;
        for(int i=1;i<S;i++)
            pw[i]=(long long)pw[i-1]*x%MOD;
        int base=(long long)pw[S-1]*x%MOD;
        pws[0]=1;
        for(int i=1;i<=MOD/S;i++)
            pws[i]=(long long)pws[i-1]*base%MOD;
        return;
    }
    int pow(long long b)
    {
        return (long long)pw[b%S]*pws[b/S]%MOD;
    }
};
```

素数

Miller Rabin

```
bool miller_rabin(long long n)
{
    if(n==1) return false;
    if(n%2==0) return n==2;
    static const int prime[]={2,325,9375,28178,450775,9780504,1795265022};
    long long u=n-1;
    int t=0;
    while(u%2==0) u/=2,t++;
    for(int a:prime)
    {
        if(a%n==0) continue;
        long long x=qpow(a,u,n);
        if(x==1||x==n-1) continue;
        for(int s=0;s<t;s++)
        {
            x=(__int128)x*x%n;
            if(x==n-1) break;
        }
        if(x!=n-1) return false;
    }
    return true;
}
```

分解质因数

Pollard Rho 算法

```
long long find_factor(long long n)
{
    static mt19937_64
rnd(chrono::steady_clock::now().time_since_epoch().count());
    long long s=0,t=0;
    long long c=rnd()%(n-1)+1;
    for(int goal=1;;goal*=2)
    {
        long long val=1;
        for(int step=1;step<=goal;step++)
        {
            t=((__int128)t*t+c)%n;
            val=(__int128)val*abs(t-s)%n;
            if(step%127==0)
            {
                long long d=gcd(val,n);
                if(d>1) return d;
            }
        }
        long long d=gcd(val,n);
        if(d>1) return d;
        s=t;
    }
}
vector<pair<long long,int>>pollard_rho(long long n)
{
    if(n==1) return {};
    if(miller_rabin(n)) return {make_pair(n,1)};
    long long p=n;
    while(p>=n) p=find_factor(n);
    int cnt=0;
    while(n%p==0) n/=p,cnt++;
    vector<pair<long long,int>>res1=pollard_rho(n),res2=pollard_rho(p);
    for(auto &[x,i]:res2)
        i*=cnt;
    vector<pair<long long,int>>res;
    int p1=0,p2=0;
    while(p1<(int)res1.size()&&p2<(int)res2.size())
    {
        if(res1[p1].first<res2[p2].first) res.emplace_back(res1[p1]),p1++;
        else if(res1[p1].first>res2[p2].first) res.emplace_back(res2[p2]),p2++;
        else if(res1[p1].first==res2[p2].first)
res.emplace_back(make_pair(res1[p1].first,res1[p1].second+res2[p2].second)),p1++,
p2++;
    }
    while(p1<(int)res1.size())
        res.emplace_back(res1[p1]),p1++;
    while(p2<(int)res2.size())
        res.emplace_back(res2[p2]),p2++;
    return res;
}
```

数论

拓展欧几里得

```
long long exgcd(long long a,long long b,long long &x,long long &y)
{
    if(b==0)
    {
        x=1,y=0;
        return a;
    }
    long long d=exgcd(b,a%b,x,y);
    long long tmp=x;
    x=y,y=tmp-a/b*y;
    return d;
}
```

线性同余方程

```
// ax=c (mod b)
long long linear_equation(long long a,long long b,long long c)
{
    long long x,y;
    long long d=exgcd(a,b,x,y);
    if(c%d!=0) return -1;
    long long s=b/d;
    x=((__int128)x*c/d%s+s)%s;
    return x;
}
```

中国剩余定理

```
long long excrt(const vector<long long> &m,const vector<long long> &a)
{
    assert(m.size()==a.size());
    long long M=m[0],ans=a[0];
    for(int i=1;i<(int)m.size();i++)
    {
        long long c=((a[i]%m[i])-(ans%m[i])+m[i])%m[i];
        long long k=linear_equation(M,m[i],c);
        ans+=k*M;
        M=M/gcd(M,m[i])*m[i];
    }
    return ans;
}
```

卢卡斯定理

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

二次剩余

Cipolla

```
struct Complex
{
    static int w;
    static int MOD;
    int real,imag;
    Complex(){}
    Complex(int _real,int _imag):real(_real),imag(_imag){}
    Complex operator*(const Complex &b)const
    {
        Complex res;
        res.real=((long long)real*b.real+(long long)w*imag%MOD*b.imag)%MOD;
        res.imag=((long long)real*b.imag+(long long)imag*b.real)%MOD;
        return res;
    }
    friend Complex qpow(Complex a,int b)
    {
        Complex res=Complex(1,0);
        while(b)
        {
            if(b&1) res=res*a;
            a=a*a,b>>=1;
        }
        return res;
    }
};
int Complex::w,Complex::MOD;
int cipolla(int n,int p)
{
    static mt19937
myrand(chrono::system_clock::now().time_since_epoch().count());
    if(n==0) return 0;
    auto check=[&](int x){return qpow(x,(p-1)/2,p)==1;};
    if(!check(n)) return -1;
    int a=myrand()%(p-1)+1;
    while(check(((long long)a*a-n+p)%p)) a=myrand()%(p-1)+1;
    Complex::w=((long long)a*a-n+p)%p;
    Complex::MOD=p;
    Complex res=qpow(Complex(a,1),(p+1)/2);
    return res.real;
```

```
}
```

原根

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<numeric>
using namespace std;
long long get_phi(long long x)
{
    long long res=x;
    for(int i=2;i*i<=x;i++)
        if(x%i==0)
        {
            res=res/i*(i-1);
            while(x%i==0) x/=i;
        }
    if(x>1) res=res/x*(x-1);
    return res;
}
long long qpow(long long a,long long b,long long p)
{
    long long res=1;
    while(b)
    {
        if(b&1) res=(__int128)res*a%p;
        a=(__int128)a*a%p,b>>=1;
    }
    return res;
}
bool is_prime(long long n)
{
    if(n<2) return false;
    for(long long i=2;i*i<=n;i++)
        if(n%i==0) return false;
    return true;
}
long long get_primitive_root(long long m)
{
    if(m==1) return -1;
    if(m==2) return 1;
    if(m==4) return 3;
    long long mm=m;
    if(mm%2==0) mm/=2;
    if(mm%2==0) return -1;
    for(long long i=2;i*i<=mm;i++)
        if(mm%i==0)
        {
            while(mm%i==0) mm/=i;
            if(mm!=1) return -1;
        }
    long long p=get_phi(m);
    long long tmp=p;
```

```
    vector<long long>factors;
    for(long long i=2;i*i<=tmp;i++)
        if(tmp%i==0)
        {
            factors.push_back(i);
            while(tmp%i==0) tmp/=i;
        }
    if(tmp>1) factors.push_back(tmp);
    for(long long g=2;g<m;g++)
    {
        bool flag=true;
        if(gcd(g,m)!=1) continue;
        for(long long u:factors)
            if(qpow(g,p/u,m)==1)
            {
                flag=false;
                break;
            }
        if(flag) return g;
    }
    return -1;
}
```

离散对数

BSGS

```
long long BSGS(long long y,long long z,long long p)
{
    y%=p,z%=p;
    if(z==1) return 0;
    if(y==0) return z==0?1:-1;
    int m=ceil(sqrt(p));
    unordered_map<long long,long long>book;
    long long sum=z;
    for(int i=0;i<m;i++)
    {
        book[sum]=i;
        sum=sum*y%p;
    }
    long long t=qpow(y,m,p);
    sum=t;
    for(int i=1;i<=ceil((double)p/m);i++)
    {
        if(book.count(sum)) return i*m-book[sum];
        sum=sum*t%p;
    }
    return -1;
}
```

扩展 BSGS

```
int ex_BSGS(int y,int z,int p)
{
    y%=p,z%=p;
    if(z==1) return 0;
    if(y==0)
    {
        if(z==0)
        {
            if(p==1) return 0;
            else return 1;
        }
        else return -1;
    }
    int k=0;
    int a=1;
    while(1)
    {
        int d=gcd(y,p);
        if(d==1) break;
        if(z%d!=0) return -1;
        z/=d,p/=d;
        k++;
        a=(long long)a*y/d%p;
        if(z==a) return k;
    }
    unordered_map<long long,long long>book;
    int m=ceil(sqrt(p));
    int sum=z;
    for(int i=0;i<m;i++)
    {
        book[sum]=i;
        sum=(long long)sum*y%p;
    }
    int t=qpow(y,m,p);
    sum=(long long)t*a%p;
    for(int i=1;i<=(p+m-1)/m;i++)
    {
        if(book.count(sum)) return i*m-book[sum]+k;
        sum=(long long)sum*t%p;
    }
    return -1;
}
```

多项式

NTT

初始化

```
constexpr int MOD=998244353;
int add(int a,int b)
{
    a+=b;
    return a>=MOD?a-MOD:a;
}
int sub(int a,int b)
{
    return a>=b?a-b:a+MOD-b;
}
int qpow(int a,int b)
{
    int res=1;
    while(b)
    {
        if(b&1) res=(long long)res*a%MOD;
        a=(long long)a*a%MOD,b>>=1;
    }
    return res;
}
int getinv(int x)
{
    return qpow(x,MOD-2);
}
typedef vector<int> Poly;
int gw[(1<<21)+1];
vector<int>inv;
void init(int _n)
{
    static int n=1;
    if(_n<n) return;
    int t=1;
    while((1<<t)<_n) t++;
    t=min(t-1,21);
    gw[0]=1,gw[1<<t]=qpow(31,1<<(21-t));
    for(int i=t;i>0;i--)
        gw[1<<(i-1)]=(long long)gw[1<<i]*gw[1<<i]%MOD;
    for(int i=1;i<(1<<t);i++)
        gw[i]=(long long)gw[i&(i-1)]*gw[i&-i]%MOD;
    inv.resize(_n+1);
    inv[1]=1;
    for(int i=2;i<=_n;i++)
        inv[i]=(long long)(MOD-MOD/i)*inv[MOD%i]%MOD;
    n=_n;
    return;
}
```

多项式乘法

```
constexpr int BIT=10;
void dft(Poly &F,int _n)
{
    int n=1;
    while(n<_n) n<<=1;
    init(n);
    F.resize(n);
    vector<unsigned long long>f(n);
    for(int i=0;i<n;i++)
        f[i]=F[i];
    for(int len=n;len>1;len>>=1)
    {
        if((len>>BIT)&1)
        {
            for(int i=0;i<n;i++)
                f[i]%=MOD;
        }
        for(int i=0,*w=gw;i<n;i+=len,w++)
            for(int k=i;k<i+(len>>1);k++)
            {
                unsigned long long l=f[k];
                int r=(*w)*f[k+(len>>1)]%MOD;
                f[k]=l+r;
                f[k+(len>>1)]=l+MOD-r;
            }
    }
    for(int i=0;i<n;i++)
        F[i]=f[i]%MOD;
    return;
}
void idft(Poly &F,int _n)
{
    int n=1;
    while(n<_n) n<<=1;
    init(n);
    F.resize(n);
    vector<unsigned long long>f(n);
    for(int i=0;i<n;i++)
        f[i]=F[i];
    for(int len=2;len<=n;len<<=1)
    {
        if((len>>BIT)&1)
        {
            for(int i=0;i<n;i++)
                f[i]%=MOD;
        }
        for(int i=0,*w=gw;i<n;i+=len,w++)
            for(int k=i;k<i+(len>>1);k++)
            {
                unsigned long long l=f[k];
                int r=f[k+(len>>1)]%MOD;
                f[k]=l+r;
                f[k+(len>>1)]=(l+MOD-r)*(*w)%MOD;
```

```
            }
    }
    int invn=getinv(n);
    for(int i=0;i<n;i++)
        F[i]=f[i]%MOD*invn%MOD;
    reverse(F.begin()+1,F.end());
    return;
}
Poly ntt(const Poly &F,const Poly &G,const function<int(int,int)> &mul)
{
    Poly f=F,g=G;
    int n=f.size()-1,m=g.size()-1;
    m+=n,n=1;
    while(n<=m) n<<=1;
    f.resize(n);
    g.resize(n);
    dft(f,n);
    dft(g,n);
    for(int i=0;i<n;i++)
        f[i]=mul(f[i],g[i]);
    idft(f,n);
    f.resize(m+1);
    return f;
}
```

多项式求逆

```
Poly getinv(const Poly &F)
{
    Poly f=F;
    int m=f.size()-1;
    int n=1;
    while(n<=m) n<<=1;
    f.resize(n);
    Poly g={getinv(f[0])};
    for(int m=2;m<=n;m<<=1)
    {
        Poly t(f.begin(),f.begin()+m);
        g=ntt(t,g,[&](const int &x,const int &y){return (2*y-(long
long)y*y%MOD*x%MOD+MOD)%MOD;});
        g.resize(m);
    }
    g.resize(m+1);
    return g;
}
```

多项式开根

```
struct Complex
{
    static int w;
    int real,imag;
    Complex(){}
    Complex(int _real,int _imag):real(_real),imag(_imag){}
    Complex operator *(const Complex &b)const
    {
        Complex res;
        res.real=((long long)real*b.real+(long long)w*imag%MOD*b.imag)%MOD;
        res.imag=((long long)real*b.imag+(long long)imag*b.real)%MOD;
        return res;
    }
    friend Complex qpow(Complex a,int b)
    {
        Complex res=Complex(1,0);
        while(b)
        {
            if(b&1) res=res*a;
            a=a*a,b>>=1;
        }
        return res;
    }
};
int Complex::w;
int cipolla(int n)
{
    static mt19937
myrand(chrono::system_clock::now().time_since_epoch().count());
    if(n==0) return 0;
    function<bool(int)>check=[&](int x){return qpow(x,(MOD-1)/2)==1;};
    if(!check(n)) return -1;
    int a=myrand()%(MOD-1)+1;
    while(check(((long long)a*a-n+MOD)%MOD)) a=myrand()%(MOD-1)+1;
    Complex::w=((long long)a*a-n+MOD)%MOD;
    Complex res=qpow(Complex(a,1),(MOD+1)/2);
    return res.real;
}
Poly sqrt(const Poly &F)
{
    Poly f=F;
    int m=f.size()-1;
    int n=1;
    while(n<=m) n<<=1;
    f.resize(n);
    int g0=cipolla(f[0]);
    Poly g={min(g0,MOD-g0)};
    int inv2=getinv(2);
    for(int m=2;m<=n;m<<=1)
    {
        Poly t(f.begin(),f.begin()+m);
        g.resize(m);
```

```
        g=g*inv2+ntt(t,getinv(g),[&](const int &x,const int &y){return (long
long)inv2*x%MOD*y%MOD;});
        g.resize(m);
    }
    g.resize(m+1);
    return g;
}
```

多项式除法 & 取模

```
Poly operator/(const Poly &F,const Poly &G)
{
    Poly f=F,g=G;
    int n=f.size()-1,m=g.size()-1;
    if(n<m) return Poly(n-m+1,0);
    reverse(f.begin(),f.end());
    reverse(g.begin(),g.end());
    f.resize(n-m+1);
    g.resize(n-m+1);
    Poly q=f*getinv(g);
    q.resize(n-m+1);
    reverse(q.begin(),q.end());
    return q;
}
Poly operator%(const Poly &F,const Poly &G)
{
    Poly f=F,g=G,q=f/g;
    int m=g.size()-1;
    g.resize(m);
    q.resize(m);
    Poly c=g*q;
    c.resize(m);
    f.resize(m);
    return f-c;
}
```

多项式求导 & 积分

```
Poly poly_derivatives(const Poly &F)
{
    Poly f=F;
    int n=f.size()-1;
    Poly g(n);
    for(int i=1;i<=n;i++)
        g[i-1]=(long long)f[i]*i%MOD;
    return g;
}
Poly poly_integration(const Poly &G)
{
    Poly g=G;
    int n=g.size()-1;
    init(n+2);
```

```
    Poly f(n+2);
    for(int i=1;i<=n+1;i++)
        f[i]=(long long)g[i-1]*inv[i]%MOD;
    return f;
}
```

多项式对数函数

```
Poly ln(const Poly &F)
{
    Poly f=F;
    int n=f.size()-1;
    Poly g=poly_derivatives(f)*getinv(f);
    g.resize(n+1);
    g=poly_integration(g);
    g.resize(n+1);
    return g;
}
```

多项式指数函数

```
Poly exp(const Poly &F)
{
    Poly f=F;
    int m=f.size()-1;
    int n=1;
    while(n<=m) n<<=1;
    f.resize(n);
    Poly g={1};
    for(int m=2;m<=n;m<<=1)
    {
        Poly t(f.begin(),f.begin()+m);
        Poly s=g;
        g.resize(m);
        g=s*(t-ln(g)+Poly(1,1));
        g.resize(m);
    }
    g.resize(m+1);
    return g;
}
```

多项式幂函数

```
Poly qpow(const Poly &F,const int &k)
{
    Poly f=F;
    int n=f.size()-1;
```

```
    Poly g(n+1);
    int pos=-1;
    for(int i=0;i<=n;i++)
        if(f[i]>0)
        {
            g[i]=f[i];
            pos=i;
            break;
        }
    if(pos==-1) return g;
    int mu=f[pos],invm=getinv(mu);
    for(int i=0;i<=n-pos;i++)
        f[i]=(long long)f[i+pos]*invm%MOD;
    for(int i=n-pos+1;i<=n;i++)
        f[i]=0;
    g[pos]=0;
    if((long long)pos*k<=n||pos==0)
    {
        g=exp(k*ln(f));
        int v=qpow(mu,k);
        for(int i=n;i>=pos*k;i--)
            g[i]=(long long)g[i-pos*k]*v%MOD;
        for(int i=pos*k-1;i>=0;i--)
            g[i]=0;
    }
    return g;
}
Poly qpow(const Poly &F,const string &k)
{
    bool isbig=false;
    int k1=0,k2=0;
    for(int i=0;i<(int)k.size();i++)
    {
        if(10ll*k1+k[i]-'0'>=MOD-1) isbig=true;
        k1=(10ll*k1+k[i]-'0')%(MOD-1);
        k2=(10ll*k2+k[i]-'0')%MOD;
    }
    Poly f=F;
    int n=f.size()-1;
    Poly g(n+1);
    int pos=-1;
    for(int i=0;i<=n;i++)
        if(f[i]>0)
        {
            g[i]=f[i];
            pos=i;
            break;
        }
    if(pos==-1) return g;
    int mu=f[pos],invm=getinv(mu);
    for(int i=0;i<=n-pos;i++)
        f[i]=(long long)f[i+pos]*invm%MOD;
    for(int i=n-pos+1;i<=n;i++)
        f[i]=0;
    g[pos]=0;
    if((!isbig&&(long long)pos*k1<=n)||pos==0)
```

```
    {
        g=exp(k2*ln(f));
        int v=qpow(mu,k1);
        for(int i=n;i>=pos*k1;i--)
            g[i]=(long long)g[i-pos*k1]*v%MOD;
        for(int i=pos*k1-1;i>=0;i--)
            g[i]=0;
    }
    return g;
}
```

多项式多点求值

```
Poly poly_multiple_point_evaluation(const Poly &F,const Poly &A)
{
    Poly f=F,a=A;
    int m=max(f.size()-1,a.size());
    f.resize(m+1),a.resize(m);
    vector<Poly>g(m<<2);
    function<void(int,int,int)> poly_multiple_point_evaluation_init=[&](int i,int
l,int r)->void
    {
        if(l==r)
        {
            g[i].resize(2);
            g[i][0]=1,g[i][1]=a[l]==0?0:MOD-a[l];
            return;
        }
        int mid=(l+r)/2;
        poly_multiple_point_evaluation_init(i*2,l,mid);
        poly_multiple_point_evaluation_init(i*2+1,mid+1,r);
        g[i]=g[i*2]*g[i*2+1];
        return;
    };
    poly_multiple_point_evaluation_init(1,0,m-1);
    Poly v(A.size());
    function<void(int,int,int,const Poly&)>poly_multiple_point_evaluation_solve=
[&](int i,int l,int r,const Poly &h)->void
    {
        if(l>=(int)A.size()) return;
        if(l==r)
        {
            v[l]=h[0];
            return;
        }
        int mid=(l+r)/2;
        Poly hl=mul_t(g[i*2+1],h),hr=mul_t(g[i*2],h);
        hl.resize(mid-l+1),hr.resize(r-mid);
        poly_multiple_point_evaluation_solve(i*2,l,mid,hl);
        poly_multiple_point_evaluation_solve(i*2+1,mid+1,r,hr);
        return;
    };
    Poly h=mul_t(getinv(g[1]),f);
```

```
    h.resize(m);
    poly_multiple_point_evaluation_solve(1,0,m-1,h);
    return v;
}
```

多项式快速插值

```
int poly_calc(const Poly &F,const int &x)
{
    Poly f=F;
    int n=f.size()-1;
    int fc=1,res=0;
    for(int i=0;i<=n;i++)
        res=(res+(long long)f[i]*fc)%MOD,fc=(long long)fc*x%MOD;
    return res;
}
Poly poly_interpolation(const vector<pair<int,int>> &p)
{
    int n=p.size()-1;
    vector<Poly>g(n<<2);
    function<void(int,int,int)> init_poly_eval=[&](int i,int l,int r)
    {
        if(l==r)
        {
            g[i].resize(2);
            g[i][0]=MOD-p[l].first,g[i][1]=1;
            return;
        }
        int mid=(l+r)/2;
        init_poly_eval(i*2,l,mid);
        init_poly_eval(i*2+1,mid+1,r);
        g[i]=g[i*2]*g[i*2+1];
        return;
    };
    init_poly_eval(1,0,n);
    Poly x(n+1);
    for(int i=0;i<=n;i++)
        x[i]=p[i].first;
    Poly F=poly_multiple_point_evaluation(poly_derivatives(g[1]),x);
    vector<int> a(n+1);
    for(int i=0;i<=n;i++)
        a[i]=(long long)p[i].second*getinv(F[i])%MOD;
    vector<Poly>res(n<<2);
    function<void(int,int,int)> solve_poly_inte=[&](int i,int l,int r)
    {
        if(l==r)
        {
            res[i]={a[l]};
            return;
        }
        int mid=(l+r)/2;
        solve_poly_inte(i*2,l,mid);
        solve_poly_inte(i*2+1,mid+1,r);
```

```
        res[i]=res[i*2]*g[i*2+1]+res[i*2+1]*g[i*2];
        return;
    };
    solve_poly_inte(1,0,n);
    return res[1];
}
```

MTT

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstdlib>
#include<ctime>
#include<vector>
#include<random>
#include<functional>
#include<algorithm>
using namespace std;
const double PI=acos(-1);
const int N=400005;
int MOD;
struct Complex
{
    double real,imag;
    Complex():real(0),imag(0){}
    Complex(double xx,double yy):real(xx),imag(yy){}
    Complex operator=(const int &b)
    {
        real=b,imag=0;
        return *this;
    }
    Complex operator+(const Complex &b)const
    {
        return Complex(real+b.real,imag+b.imag);
    }
    Complex operator-(const Complex &b)const
    {
        return Complex(real-b.real,imag-b.imag);
    }
    Complex operator*(const Complex &b)const
    {
        return Complex(real*b.real-imag*b.imag,real*b.imag+imag*b.real);
    }
    friend Complex getinv(const Complex &rhs)
    {
        return Complex(rhs.real,-rhs.imag);
    }
};
Complex W[2][N<<1];
void init_omega(int n=400000)
{
    for(int len=1;len<=n;len<<=1)
        for(int k=0;k<len;k++)
```

```
            w[0][len+k]=Complex(cos(2*PI*k/len),sin(2*PI*k/len)),w[1]
[len+k]=Complex(cos(2*PI*k/len),-sin(2*PI*k/len));
    return;
}
typedef vector<int> Poly;
Poly mtt(const Poly &F,const Poly &G,const function<Complex(const Complex &,const
Complex &)> &mul)
{
    int n=F.size()-1,m=G.size()-1;
    vector<Complex> a(n+1),b(n+1),c(m+1),d(m+1);
    for(int i=0;i<=n;i++)
        a[i]=F[i]>>15,b[i]=F[i]&0x7fff;
    for(int i=0;i<=m;i++)
        c[i]=G[i]>>15,d[i]=G[i]&0x7fff;
    m+=n,n=1;
    while(n<=m) n<<=1;
    a.resize(n),b.resize(n),c.resize(n),d.resize(n);
    vector<int>rev(n);
    for(int i=0;i<n;i++)
    {
        rev[i]=rev[i/2]>>1;
        if(i&1) rev[i]|=n/2;
    }
    function<void(vector<Complex> &)> dft=[=](vector<Complex> &F)
    {
        int n=F.size();
        vector<Complex> f(n);
        for(int i=0;i<n;i++)
            f[i]=F[rev[i]];
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+len/2;k++)
                {
                    Complex l=f[k];
                    Complex r=f[k+len/2]*w[0][len+k-i];
                    f[k]=l+r;
                    f[k+len/2]=l-r;
                }
        for(int i=0;i<n;i++)
            F[i]=f[i];
        return;
    };
    dft(a),dft(b),dft(c),dft(d);
    vector<Complex> ac(n),ad(n),bc(n),bd(n);
    for(int i=0;i<n;i++)

 ac[i]=mul(a[i],c[i]),ad[i]=mul(a[i],d[i]),bc[i]=mul(b[i],c[i]),bd[i]=mul(b[i],d[
i]);
    function<void(vector<Complex> &)> idft=[=](vector<Complex> &F)
    {
        int n=F.size();
        vector<Complex> f(n);
        for(int i=0;i<n;i++)
            f[i]=F[rev[i]];
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
```

```
                for(int k=i;k<i+len/2;k++)
                {
                    Complex l=f[k];
                    Complex r=f[k+len/2]*w[1][len+k-i];
                    f[k]=l+r;
                    f[k+len/2]=l-r;
                }
        for(int i=0;i<n;i++)
            F[i]=f[i];
        return;
    };
    idft(ac),idft(ad),idft(bc),idft(bd);
    Poly AC(m+1),AD(m+1),BC(m+1),BD(m+1);
    for(int i=0;i<=m;i++)
        AC[i]=(long long)round(ac[i].real/n)%MOD,AD[i]=(long
long)round(ad[i].real/n)%MOD,BC[i]=(long long)round(bc[i].real/n)%MOD,BD[i]=(long
long)round(bd[i].real/n)%MOD;
    Poly res(m+1);
    for(int i=0;i<=m;i++)
        res[i]=(((long long)(AC[i])<<30)+((long long)(AD[i]+BC[i])
<<15)+BD[i])%MOD;
    return res;
}
Poly operator*(const Poly &F,const Poly &G)
{
    return mtt(F,G,[=](const Complex &x,const Complex &y){return x*y;});
}
int main()
{
    init_omega();
    int n,m;
    scanf("%d%d%d",&n,&m,&MOD);
    Poly f,g;
    f.resize(n+1),g.resize(m+1);
    for(int i=0;i<=n;i++)
        scanf("%d",&f[i]);
    for(int i=0;i<=m;i++)
        scanf("%d",&g[i]);
    Poly res=f*g;
    for(int u:res)
        printf("%d ",u);
    return 0;
}
```

分治 FFT

> 给定序列 $g_{1\dots n-1}$，求序列 $f_{0\dots n-1}$。
>
> 其中 $f_i=\sum_{j=1}^{i} f_{i-j}g_j$，边界为 $f_0=1$。

```
void cdq_ntt(Poly &F,const Poly &G,int l,int r)
{
    if(l==r) return;
    int mid=(l+r)/2;
```

```
    cdq_ntt(F,G,l,mid);
    Poly f(r-l),g(r-l);
    for(int i=0;i<=r-l-1;i++)
        f[i]=i+l<=mid?F[i+l]:0,g[i]=G[i+1];
    Poly res=f*g;
    for(int i=mid+1;i<=r;i++)
        F[i]=(F[i]+res[i-l-1])%MOD;
    cdq_ntt(F,G,mid+1,r);
    return;
}
```

FWT

快速 OR 卷积

```
Poly or_convolution(const Poly &F,const Poly &G)
{
    Poly f=F,g=G;
    int m=max(f.size()-1,g.size()-1),n=1;
    while(n<=m) n<<=1;
    f.resize(n),g.resize(n);
    auto fwt=[&](Poly &F)
    {
        int n=F.size();
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+(len>>1);k++)
                    F[k+(len>>1)]=add(F[k+(len>>1)],F[k]);
        return;
    };
    fwt(f);
    fwt(g);
    for(int i=0;i<n;i++)
        f[i]=(long long)f[i]*g[i]%MOD;
    auto ifwt=[&](Poly &F)
    {
        int n=F.size();
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+(len>>1);k++)
                    F[k+(len>>1)]=sub(F[k+(len>>1)],F[k]);
        return;
    };
    ifwt(f);
    return f;
}
```

快速 AND 卷积

```
Poly and_convolution(const Poly &F,const Poly &G)
{
    Poly f=F,g=G;
    int m=max(f.size()-1,g.size()-1),n=1;
    while(n<=m) n<<=1;
    f.resize(n),g.resize(n);
    function<void(Poly &)> fwt=[&](Poly &F)
    {
        int n=F.size();
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+(len>>1);k++)
                    F[k]=add(F[k],F[k+(len>>1)]);
        return;
    };
    fwt(f);
    fwt(g);
    for(int i=0;i<n;i++)
        f[i]=(long long)f[i]*g[i]%MOD;
    function<void(Poly &)> ifwt=[&](Poly &F)
    {
        int n=F.size();
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+(len>>1);k++)
                    F[k]=sub(F[k],F[k+(len>>1)]);
        return;
    };
    ifwt(f);
    return f;
}
```

快速 XOR 卷积

```
Poly xor_convolution(const Poly &F,const Poly &G)
{
    Poly f=F,g=G;
    int m=max(f.size()-1,g.size()-1),n=1;
    while(n<=m) n<<=1;
    f.resize(n),g.resize(n);
    function<void(Poly &)> fwt=[&](Poly &F)
    {
        int n=F.size();
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+(len>>1);k++)
                {
                    int l=F[k],r=F[k+(len>>1)];
                    F[k]=add(l,r);
```

```
                    F[k+(len>>1)]=sub(l,r);
                }
        return;
    };
    fwt(f);
    fwt(g);
    for(int i=0;i<n;i++)
        f[i]=(long long)f[i]*g[i]%MOD;
    function<void(Poly &)> ifwt=[&](Poly &F)
    {
        int n=F.size();
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+(len>>1);k++)
                {
                    int l=F[k],r=F[k+(len>>1)];
                    F[k]=add(l,r);
                    F[k+(len>>1)]=sub(l,r);
                }
        int invn=getinv(n);
        for(int i=0;i<(int)F.size();i++)
            F[i]=(long long)F[i]*invn%MOD;
        return;
    };
    ifwt(f);
    return f;
}
```

子集卷积

```
Poly subset_convolution(const Poly &F,const Poly &G)
{
    int m=max(F.size()-1,G.size()-1),n=1,c=1;
    while(n<=m) n<<=1,c++;
    vector<Poly> f(c+1,Poly(n,0)),g(c+1,Poly(n,0));
    for(int i=0;i<(int)F.size();i++)
        f[__builtin_popcount(i)][i]=F[i];
    for(int i=0;i<(int)G.size();i++)
        g[__builtin_popcount(i)][i]=G[i];
    function<void(Poly &)> fwt=[&](Poly &F)
    {
        int n=F.size();
        vector<unsigned long long>f(n);
        for(int i=0;i<n;i++)
            f[i]=F[i];
        for(int len=2;len<=n;len<<=1)
        {
            if((len>>BIT)&1)
            {
                for(int i=0;i<n;i++)
                    f[i]%=MOD;
            }
            for(int i=0;i<n;i+=len)
```

```
                for(int k=i;k<i+(len>>1);k++)
                {
                    unsigned long long l=F[k];
                    int r=F[k+(len>>1)]%MOD;
                    F[k]=l+r;
                    F[k+(len>>1)]=l+MOD-r;
                }
        }
        for(int i=0;i<n;i++)
            F[i]=f[i]%MOD;
        return;
    };
    for(int i=0;i<=c;i++)
        fwt(f[i]),fwt(g[i]);
    vector<Poly> h(c+1,Poly(n,0));
    for(int i=0;i<=c;i++)
        for(int j=0;j<=i;j++)
        {
            int k=i-j;
            for(int s=0;s<n;s++)
                h[i][s]=(h[i][s]+(long long)f[j][s]*g[k][s])%MOD;
        }
    function<void(Poly &)> ifwt=[&](Poly &F)
    {
        int n=F.size();
        for(int len=2;len<=n;len<<=1)
            for(int i=0;i<n;i+=len)
                for(int k=i;k<i+(len>>1);k++)
                {
                    int l=F[k],r=F[k+(len>>1)];
                    F[k]=add(l,r);
                    F[k+(len>>1)]=sub(l,r);
                }
        int invn=getinv(n);
        for(int i=0;i<(int)F.size();i++)
            F[i]=(long long)F[i]*invn%MOD;
        return;
    };
    for(int i=0;i<=c;i++)
        ifwt(h[i]);
    Poly res(n);
    for(int i=0;i<n;i++)
        res[i]=h[__builtin_popcount(i)][i];
    return res;
}
```

线性代数

行列式

```
int det(vector<vector<int>> &a)
{
    int n=a.size();
    int d=1;
    for(int i=0;i<n;i++)
    {
        int k=i;
        for(int j=i;j<n;j++)
            if(a[j][i])
            {
                k=j;
                break;
            }
        if(!a[k][i])
        {
            d=0;
            break;
        }
        if(i!=k) swap(a[i],a[k]),d=(MOD-d)%MOD;
        d=(long long)d*a[i][i]%MOD;
        int invi=getinv(a[i][i]);
        for(int j=i+1;j<n;j++)
            a[i][j]=(long long)a[i][j]*invi%MOD;
        for(int j=0;j<n;j++)
            if(j!=i&&a[j][i])
            {
                for(int k=i+1;k<n;k++)
                    a[j][k]=(a[j][k]-(long long)a[i][k]*a[j][i]%MOD+MOD)%MOD;
                a[j][i]=0;
            }
    }
    return d;
}
```

线性基

```
struct Basis
{
private:
    static const int N=63;
    long long a[N];
    vector<long long>b;
    int cnt;
public:
    Basis()
    {
        cnt=0;
        memset(a,0,sizeof(a));
        b.clear();
        return;
    }
```

```
    void clear()
    {
        cnt=0;
        memset(a,0,sizeof(a));
        b.clear();
        return;
    }
    void insert(long long x)
    {
        cnt++;
        for(int i=N-1;i>=0;i--)
            if((x>>i)&1)
            {
                if(a[i]) x^=a[i];
                else
                {
                    for(int j=0;j<i;j++)
                        if((x>>j)&1) x^=a[j];
                    for(int j=i+1;j<N;j++)
                        if((a[j]>>i)&1) a[j]^=x;
                    a[i]=x;
                    break;
                }
            }
        b.clear();
        for(int i=0;i<N;i++)
            if(a[i]) b.push_back(i);
        return;
    }
    bool exist(long long x)
    {
        for(int i=N-1;i>=0;i--)
            if((x>>i)&1)
            {
                if(a[i]) x^=a[i];
                else return false;
            }
        return true;
    }
    int size()const
    {
        return b.size();
    }
    long long max_xor()const
    {
        long long res=0;
        for(int i=N-1;i>=0;i--)
            res^=a[i];
        return res;
    }
    long long kth_xor(long long k)const
    {
        if(size()!=cnt) k--;
        if(k>(1ll<<size())-1) return -1;
        long long res=0;
        for(int i=0;i<(int)b.size();i++)
```

```
            if((k>>i)&1) res^=a[b[i]];
        return res;
    }
    long long query_rank(const int &x)const
    {
        long long res=0;
        for(int i=0;i<(int)b.size();i++)
            if((x>>b[i])&1) res+=1ll<<i;
        return res;
    }
};
```

字符串

字符串哈希

```
struct Hash
{
    static constexpr int MOD1=1000000007,MOD2=1000000009;
    static constexpr int BASE=23;
    int x,y;
    Hash():x(0),y(0){}
    Hash(int v):x(v),y(v){}
    Hash(int _x,int _y):x(_x),y(_y){}
    long long to_ll()const
    {
        return (((long long)x)<<31)+y;
    }
    friend bool operator == (const Hash &a,const Hash &b)
    {
        return a.x==b.x&&a.y==b.y;
    }
    friend Hash operator + (const Hash &a,const Hash &b)
    {
        int x=a.x+b.x,y=a.y+b.y;
        if(x>=MOD1) x-=MOD1;
        if(y>=MOD2) y-=MOD2;
        return Hash(x,y);
    }
    friend Hash operator - (const Hash &a,const Hash &b)
    {
        int x=a.x-b.x,y=a.y-b.y;
        if(x<0) x+=MOD1;
        if(y<0) y+=MOD2;
        return Hash(x,y);
    }
    friend Hash operator * (const Hash &a,const Hash &b)
    {
        return Hash((long long)a.x*b.x%MOD1,(long long)a.y*b.y%MOD2);
    }
};
Hash pw[N];
```

```
void init_hash(int n)
{
    pw[0]=1;
    for(int i=1;i<=n;i++)
        pw[i]=pw[i-1]*Hash::BASE;
    return;
}
vector<Hash>hsh[N];
vector<Hash>get_hash(const string &s)
{
    vector<Hash>val(s.size());
    val[0]=s[0]-'a'+1;
    for(int i=1;i<(int)s.size();i++)
        val[i]=val[i-1]*Hash::BASE+s[i]-'a'+1;
    return val;
}
Hash calc_hash(const vector<Hash> &val,int l,int r)
{
    if(l>r) return 0;
    Hash res=val[r];
    if(l-1>=0) res=res-val[l-1]*pw[r-l+1];
    return res;
}
```

字典树

```
struct Trie
{
    static const int C=128;
    int trie[N][C],tot;
    int ed[N],siz[N];
    Trie():tot(1)
    {
        memset(trie[1],0,sizeof(trie[1]));
        ed[1]=siz[1]=0;
    }
    void clear()
    {
        tot=1;
        memset(trie[1],0,sizeof(trie[1]));
        ed[1]=siz[1]=0;
        return;
    }
    int new_node()
    {
        int now=++tot;
        memset(trie[now],0,sizeof(trie[now]));
        ed[now]=siz[now]=0;
        return now;
    }
    int get(char c)
    {
        return (int)c;
```

```
    }
    void insert(const string &s)
    {
        int len=s.size();
        int u=1;
        for(int i=0;i<len;i++)
        {
            int c=get(s[i]);
            if(!trie[u][c]) trie[u][c]=new_node();
            u=trie[u][c];
        }
        ed[u]++;
        return;
    }
    int query(const string &s)
    {
        int len=s.size();
        int u=1;
        for(int i=0;i<len;i++)
        {
            int c=get(s[i]);
            if(!trie[u][c]) return 0;
            u=trie[u][c];
        }
        return siz[u];
    }
    void dfs(int u)
    {
        if(!u) return;
        siz[u]=ed[u];
        for(int c=0;c<C;c++)
            dfs(trie[u][c]),siz[u]+=siz[trie[u][c]];
        return;
    }
};
```

前缀函数 / KMP

```
vector<int> prefix_function(const string &s)
{
    int n=s.size();
    vector<int>pi(n,0);
    for(int i=1;i<n;i++)
    {
        int j=pi[i-1];
        while(j>0&&s[i]!=s[j]) j=pi[j-1];
        if(s[i]==s[j]) j++;
        pi[i]=j;
    }
    return pi;
}
```

Z 函数（扩展 KMP）

```
vector<int> z_function(const string &s)
{
    int n=s.size();
    vector<int> z(n);
    for(int i=1,l=0,r=0;i<n;i++)
    {
        if(i<=r&&z[i-l]<r-i+1) z[i]=z[i-l];
        else
        {
            z[i]=max(0,r-i+1);
            while(i+z[i]<n&&s[z[i]]==s[i+z[i]]) z[i]++;
        }
        if(i+z[i]-1>r) l=i,r=i+z[i]-1;
    }
    return z;
}
```

Manacher

```
vector<int>manacher(const string &s)
{
    int n=s.size();
    vector<int>p(n);
    int l=0,r=-1;
    for(int i=0;i<n;i++)
    {
        int k=1;
        if(i<=r) k=min(p[l+r-i],r-i+1);
        while(0<=i-k&&i+k<n&&s[i-k]==s[i+k]) k++;
        p[i]=k;
        k--;
        if(i+k>r) l=i-k,r=i+k;
    }
    return p;
}
```

后缀数组 (SA)

```
vector<int>suffix_array(const string &s)
{
    int n=s.size();
    vector<int>rk(n),sa(n);
    for(int i=0;i<n;i++)
        sa[i]=i;
    sort(sa.begin(),sa.end(),[&](const int &x,const int &y){return s[x]<s[y];});
    rk[sa[0]]=0;
    for(int i=1;i<n;i++)
        if(s[sa[i]]==s[sa[i-1]]) rk[sa[i]]=rk[sa[i-1]];
        else rk[sa[i]]=rk[sa[i-1]]+1;
```

```
    int m=rk[sa[n-1]]+1;
    for(int w=1;w<n;w<<=1)
    {
        vector<int>id;
        for(int i=n-w;i<n;i++)
            id.emplace_back(i);
        for(int i=0;i<n;i++)
            if(sa[i]>=w) id.emplace_back(sa[i]-w);
        vector<int>cnt(m,0);
        for(int i=0;i<n;i++)
            cnt[rk[i]]++;
        for(int i=1;i<m;i++)
            cnt[i]+=cnt[i-1];
        for(int i=n-1;i>=0;i--)
            sa[--cnt[rk[id[i]]]]=id[i];
        vector<int>oldrk=rk;
        rk[sa[0]]=0;
        for(int i=1;i<n;i++)
        {
            if(oldrk[sa[i]]==oldrk[sa[i-1]]&&(sa[i]+w>=n?-1:oldrk[sa[i]+w])==
(sa[i-1]+w>=n?-1:oldrk[sa[i-1]+w])) rk[sa[i]]=rk[sa[i-1]];
            else rk[sa[i]]=rk[sa[i-1]]+1;
        }
        m=rk[sa[n-1]]+1;
        if(m==n) break;
    }
    return sa;
}
vector<int> get_rank(const vector<int> &sa)
{
    vector<int>rk(sa.size());
    for(int i=0;i<(int)sa.size();i++)
        rk[sa[i]]=i;
    return rk;
}
vector<int> get_height(const string &s)
{
    int n=s.size();
    vector<int>sa=suffix_array(s),rk=get_rank(sa);
    vector<int>height(n);
    for(int i=0,k=0;i<n;i++)
    {
        if(rk[i]==0)
        {
            height[rk[i]]=k=0;
            continue;
        }
        if(k>0) 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;
    }
    return height;
}
```

后缀自动机 (SAM)

```
struct SAM
{
    struct state
    {
        int len,link;
        int siz;
        int next[26];
    };
    state st[N*2];
    int tot,last;
    SAM()
    {
        st[0].len=0;
        st[0].link=-1;
        tot=last=0;
    }
    void clear()
    {
        st[0].len=0;
        st[0].siz=0;
        st[0].link=-1;
        tot=last=0;
        return;
    }
    int new_node()
    {
        int now=++tot;
        st[now].len=0,st[now].link=-1;
        st[now].siz=0;
        memset(st[now].next,0,sizeof(st[now].next));
        return now;
    }
    int convert(char c)
    {
        return c-'a';
    }
    void extend(char ch)
    {
        int c=convert(ch);
        int cur=new_node();
        st[cur].len=st[last].len+1;
        st[cur].siz=1;
        int p=last;
        while(p!=-1&&!st[p].next[c])
            st[p].next[c]=cur,p=st[p].link;
        if(p==-1) st[cur].link=0;
        else
        {
            int q=st[p].next[c];
            if(st[p].len+1==st[q].len) st[cur].link=q;
            else
            {
                int clone=new_node();
```

```
                st[clone]=st[q];
                st[clone].siz=0;
                st[clone].len=st[p].len+1;
                while(p!=-1&&st[p].next[c]==q)
                    st[p].next[c]=clone,p=st[p].link;
                st[q].link=st[cur].link=clone;
            }
        }
        last=cur;
        return;
    }
    void build(const string &s)
    {
        clear();
        for(char ch:s)
            extend(ch);
        return;
    }
    void dfs(int u)
    {
        for(int v:G[u])
        {
            dfs(v);
            st[u].siz+=st[v].siz;
        }
        return;
    }
    vector<int>G[N*2];
    void rebuild()
    {
        for(int i=1;i<=tot;i++)
            G[st[i].link].emplace_back(i);
        dfs(0);
        return;
    }
};
```

数据结构

ZKW 线段树

```
struct ZKW_Tree
{
    struct Node
    {
        int mn,tag;
    }tree[N*3];
    int p;
    void push_up(int i)
    {
        tree[i].mn=min(tree[i<<1].mn,tree[i<<1|1].mn)+tree[i].tag;
        return;
```

```
    }
    void init(int n,int *w)
    {
        for(p=1;p<n+2;p<<=1);
        for(int i=1;i<=p;i++)
            tree[i].tag=0;
        for(int i=1;i<=n;i++)
            tree[p+i].mn=w[i];
        for(int i=(p+n)>>1;i>0;i--)
            push_up(i);
        return;
    }
    void add(int l,int r,int d)
    {
        for(l=p+l-1,r=p+r+1;l^1^r;)
        {
            if(~l&1) tree[l^1].mn+=d,tree[l^1].tag+=d;
            if(r&1) tree[r^1].mn+=d,tree[r^1].tag+=d;
            l>>=1,r>>=1;
            push_up(l),push_up(r);
        }
        for(l>>=1;l>0;l>>=1)
            push_up(l);
        return;
    }
    int query(int l,int r)
    {
        int resl=INF,resr=INF;
        for(l=p+l-1,r=p+r+1;l^1^r;)
        {
            if(~l&1) resl=min(resl,tree[l^1].mn);
            if(r&1) resr=min(resr,tree[r^1].mn);
            l>>=1,r>>=1;
            resl+=tree[l].tag;
            resr+=tree[r].tag;
        }
        resl=min(resl,resr);
        for(l>>=1;l>0;l>>=1)
            resl+=tree[l].tag;
        return resl;
    }
};
```

李超线段树

```
#include<bits/stdc++.h>
using namespace std;
constexpr int N=200005;
constexpr long long INF=4e18;
struct Line
{
    long long k,b;
    long long calc(long long x)
```

```
    {
        return k*x+b;
    }
};
struct Li_Chao_Tree
{
    struct Node
    {
        int ls,rs;
        Line l;
    }tree[N*120];
    int rt,tot;
    Li_Chao_Tree():rt(0),tot(0){}
    int new_node()
    {
        int now=++tot;
        assert(now<=N*60);
        tree[now].ls=0,tree[now].rs=0;
        tree[now].l={0,INF};
        return now;
    }
    long long query(int i,int l,int r,int u)
    {
        if(!i) return INF;
        if(l==r) return tree[i].l.calc(u);
        int mid=(l+r)>>1;
        long long res=tree[i].l.calc(u);
        if(u<=mid) res=min(res,query(tree[i].ls,l,mid,u));
        else res=min(res,query(tree[i].rs,mid+1,r,u));
        return res;
    }
    void insert(int &i,int l,int r,Line pos)
    {
        if(!i) i=new_node();
        if(tree[i].l.b>=INF)
        {
            tree[i].l=pos;
            return;
        }
        if(tree[i].l.calc(l)>pos.calc(l)) swap(tree[i].l,pos);
        if(tree[i].l.calc(r)<=pos.calc(r)) return;
        if(l==r) return;
        int mid=(l+r)>>1;
        if(tree[i].l.calc(mid)>pos.calc(mid))
insert(tree[i].ls,l,mid,tree[i].l),tree[i].l=pos;
        else insert(tree[i].rs,mid+1,r,pos);
        return;
    }
    void add(int &i,int l,int r,int ql,int qr,Line pos)
    {
        if(!i) i=new_node();
        if(ql<=l&&r<=qr) return insert(i,l,r,pos);
        int mid=(l+r)>>1;
        if(ql<=mid) add(tree[i].ls,l,mid,ql,qr,pos);
        if(qr>mid) add(tree[i].rs,mid+1,r,ql,qr,pos);
        return;
```

```
    }
    #undef ls
    #undef rs
};
Li_Chao_Tree T;
int n,m,q;
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    cin>>n>>q;
    m=1e9;
    for(int i=0;i<n;i++)
    {
        int l,r,a;
        long long b;
        cin>>l>>r>>a>>b;
        r--;
        T.add(T.rt,-m,m,l,r,Line(a,b));
    }
    while(q--)
    {
        int op;
        cin>>op;
        if(op==0)
        {
            int l,r,a;
            long long b;
            cin>>l>>r>>a>>b;
            r--;
            T.add(T.rt,-m,m,l,r,Line(a,b));
        }
        else if(op==1)
        {
            int p;
            cin>>p;
            long long ans=T.query(T.rt,-m,m,p);
            if(ans>=INF) cout<<"NO\n";
            else cout<<ans<<"\n";
        }
    }
    return 0;
}
```

平衡树

普通平衡树

FHQ-Treap

```
#include<iostream>
#include<cstdio>
#include<chrono>
#include<random>
using namespace std;
const int N=1100005;
mt19937 rnd(chrono::system_clock::now().time_since_epoch().count());
struct FHQ_Treap
{
    int rt,tot;
    struct Node
    {
        int ch[2];
        int val;
        unsigned int rd;
        int size;
    }tree[N];
    FHQ_Treap():tot(0),rt(0){}
    void clear()
    {
        tot=rt=0;
        return;
    }
    int new_node(int val)
    {
        int now=++tot;
        tree[now].ch[0]=tree[now].ch[1]=0;
        tree[now].val=val,tree[now].rd=rnd();
        tree[now].size=1;
        return now;
    }
    void push_up(int u)
    {
        tree[u].size=tree[tree[u].ch[0]].size+tree[tree[u].ch[1]].size+1;
        return;
    }
    int merge(int u,int v)
    {
        if(!u) return v;
        if(!v) return u;
        if(tree[u].rd<tree[v].rd)
        {
            tree[u].ch[1]=merge(tree[u].ch[1],v);
            push_up(u);
            return u;
        }
        else
        {
            tree[v].ch[0]=merge(u,tree[v].ch[0]);
            push_up(v);
            return v;
        }
    }
```

```
    void split(int now,int k,int &x,int &y)
    {
        if(!now)
        {
            x=y=0;
            return;
        }
        if(tree[now].val<=k)
        {
            x=now;
            split(tree[now].ch[1],k,tree[x].ch[1],y);
        }
        else
        {
            y=now;
            split(tree[now].ch[0],k,x,tree[y].ch[0]);
        }
        push_up(now);
        return;
    }
    void insert(int val)
    {
        int x,y;
        split(rt,val,x,y);
        rt=merge(merge(x,new_node(val)),y);
        return;
    }
    void erase(int val)
    {
        int x,y,z;
        split(rt,val,x,z);
        split(x,val-1,x,y);
        y=merge(tree[y].ch[0],tree[y].ch[1]);
        rt=merge(merge(x,y),z);
        return;
    }
    int find(int val)
    {
        int x,y;
        split(rt,val-1,x,y);
        int res=tree[x].size+1;
        rt=merge(x,y);
        return res;
    }
    int findkth(int pos,int x)
    {
        int now=pos;
        while(now)
        {
            if(tree[now].ch[0]&&x<=tree[tree[now].ch[0]].size)
            {
                now=tree[now].ch[0];
                continue;
            }
            if(tree[now].ch[0]) x-=tree[tree[now].ch[0]].size;
            if(x<=1) return tree[now].val;
```

```
            x--;
            now=tree[now].ch[1];
        }
        return 0;
    }
    int findkth(int x)
    {
        return findkth(rt,x);
    }
    int prev(int val)
    {
        int x,y;
        split(rt,val-1,x,y);
        int res=findkth(x,tree[x].size);
        rt=merge(x,y);
        return res;
    }
    int next(int val)
    {
        int x,y;
        split(rt,val,x,y);
        int res=findkth(y,1);
        rt=merge(x,y);
        return res;
    }
};
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    FHQ_Treap T;
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        int x;
        cin>>x;
        T.insert(x);
    }
    int last=0,ans=0;
    for(int i=1;i<=m;i++)
    {
        int op,x;
        cin>>op>>x;
        x^=last;
        if(op==1) T.insert(x);
        else if(op==2) T.erase(x);
        else if(op==3) last=T.find(x),ans^=last;
        else if(op==4) last=T.findkth(x),ans^=last;
        else if(op==5) last=T.prev(x),ans^=last;
        else if(op==6) last=T.next(x),ans^=last;
    }
    cout<<ans;
    return 0;
}
```

文艺平衡树

FHQ-Treap

```
#include<iostream>
#include<cstdio>
#include<random>
#include<chrono>
using namespace std;
constexpr int N=100005;
mt19937 rnd(chrono::system_clock::now().time_since_epoch().count());
struct FHQ_Treap
{
    struct Node
    {
        int ch[2];
        int val;
        unsigned int rd;
        int siz;
        int tag;
        void clear()
        {
            ch[0]=ch[1]=0;
            val=0;
            rd=0;
            siz=0,tag=0;
            return;
        }
    }tree[N];
    int rt,tot;
    FHQ_Treap():rt(0),tot(0){}
    void clear()
    {
        rt=tot=0;
        tree[0].clear();
        return;
    }
    int new_node(int val)
    {
        int now=++tot;
        tree[now].clear();
        tree[now].val=val,tree[now].rd=rnd();
        tree[now].siz=1;
        return now;
    }
    void reverse(int u)
    {
        if(!u) return;
        tree[u].tag^=1;
        swap(tree[u].ch[0],tree[u].ch[1]);
        return;
    }
    void push_down(int u)
```

```
    {
        if(!tree[u].tag) return;
        reverse(tree[u].ch[0]);
        reverse(tree[u].ch[1]);
        tree[u].tag=0;
        return;
    }
    void push_up(int u)
    {
        tree[u].siz=tree[tree[u].ch[0]].siz+tree[tree[u].ch[1]].siz+1;
        return;
    }
    int merge(int u,int v)
    {
        if(!u) return v;
        if(!v) return u;
        push_down(u);
        push_down(v);
        if(tree[u].rd<tree[v].rd)
        {
            tree[u].ch[1]=merge(tree[u].ch[1],v);
            push_up(u);
            return u;
        }
        else
        {
            tree[v].ch[0]=merge(u,tree[v].ch[0]);
            push_up(v);
            return v;
        }
    }
    void split_size(int now,int k,int &x,int &y)
    {
        if(!now)
        {
            x=y=0;
            return;
        }
        push_down(now);
        if(k>tree[tree[now].ch[0]].siz)
        {
            x=now;
            split_size(tree[now].ch[1],k-tree[tree[now].ch[0]].siz-
1,tree[x].ch[1],y);
            push_up(x);
        }
        else
        {
            y=now;
            split_size(tree[now].ch[0],k,x,tree[y].ch[0]);
            push_up(y);
        }
        return;
    }
    void insert(int val)
    {
```

```
        int x,y;
        split_size(rt,val,x,y);
        rt=merge(merge(x,new_node(val)),y);
        return;
    }
    void reverse(int l,int r)
    {
        int x,y,z;
        split_size(rt,r,x,z);
        split_size(x,l-1,x,y);
        reverse(y);
        rt=merge(merge(x,y),z);
        return;
    }
    int findkth(int u)
    {
        int x,y,z;
        split_size(rt,u,x,z);
        split_size(x,u-1,x,y);
        int res=tree[y].val;
        rt=merge(merge(x,y),z);
        return res;
    }
};
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    FHQ_Treap T;
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        T.insert(i);
    for(int i=1;i<=m;i++)
    {
        int l,r;
        cin>>l>>r;
        T.reverse(l,r);
    }
    for(int i=1;i<=n;i++)
        cout<<T.findkth(i)<<" ";
    return 0;
}
```

动态树

Link Cut Tree

```
#include<iostream>
#include<cstdio>
using namespace std;
const int N=150005;
```

```
struct LCT
{
    struct Node
    {
        int fa,ls,rs;
        int val,sum;
        int rev;
    }tree[N];
    void push_up(int u)
    {
        tree[u].sum=tree[tree[u].ls].sum^tree[tree[u].rs].sum^tree[u].val;
        return;
    }
    void reverse(int u)
    {
        swap(tree[u].ls,tree[u].rs);
        tree[u].rev^=1;
        return;
    }
    void push_down(int u)
    {
        if(!tree[u].rev) return;
        if(tree[u].ls) reverse(tree[u].ls);
        if(tree[u].rs) reverse(tree[u].rs);
        tree[u].rev=0;
        return;
    }
    int get(int u)
    {
        if(tree[tree[u].fa].ls==u) return 0;
        if(tree[tree[u].fa].rs==u) return 1;
        return -1;
    }
    bool isroot(int u)
    {
        return get(u)==-1;
    }
    void connect(int u,int v,int op)
    {
        if(u) tree[u].fa=v;
        if(v)
        {
            if(op==0) tree[v].ls=u;
            if(op==1) tree[v].rs=u;
        }
        return;
    }
    void rotate(int u)
    {
        int f=tree[u].fa,gf=tree[f].fa,r=get(u),gr=get(f);
        int v=0;
        if(r==1) v=tree[u].ls;
        if(r==0) v=tree[u].rs;
        connect(u,gf,gr);
        connect(v,f,r);
        connect(f,u,r^1);
```

```
        push_up(f);
        push_up(u);
        return;
    }
    void pushall(int u)
    {
        if(!isroot(u)) pushall(tree[u].fa);
        push_down(u);
        return;
    }
    void splay(int u)
    {
        pushall(u);
        while(!isroot(u))
        {
            int f=tree[u].fa;
            if(!isroot(f)) rotate(get(u)==get(f)?f:u);
            rotate(u);
        }
        return;
    }
    int access(int u)
    {
        int v;
        for(v=0;u;v=u,u=tree[u].fa)
        {
            splay(u);
            tree[u].rs=v;
            push_up(u);
        }
        return v;
    }
    void makeroot(int u)
    {
        access(u);
        splay(u);
        reverse(u);
        return;
    }
    int findroot(int u)
    {
        access(u);
        splay(u);
        push_down(u);
        while(tree[u].ls) u=tree[u].ls,push_down(u);
        splay(u);
        return u;
    }
    void split(int x,int y)
    {
        makeroot(x);
        access(y);
        splay(y);
        return;
    }
    bool link(int x,int y)
```

```
    {
        makeroot(x);
        if(findroot(y)==x) return false;
        tree[x].fa=y;
        return true;
    }
    bool cut(int x,int y)
    {
        if(findroot(x)!=findroot(y)) return false;
        split(x,y);
        if(tree[x].fa!=y||tree[x].rs) return false;
        tree[x].fa=tree[y].ls=0;
        push_up(y);
        return true;
    }
    int query(int x,int y)
    {
        makeroot(x);
        if(findroot(y)!=x) return -1;
        split(x,y);
        return tree[y].sum;
    }
    void modify(int u,int v)
    {
        splay(u);
        tree[u].val=v;
        return;
    }
}T;
int n,m;
int a[N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>a[i];
    for(int i=1;i<=n;i++)
        T.tree[i].sum=T.tree[i].val=a[i];
    for(int i=1;i<=m;i++)
    {
        int op,x,y;
        cin>>op>>x>>y;
        if(op==0) cout<<T.query(x,y)<<"\n";
        else if(op==1) T.link(x,y);
        else if(op==2) T.cut(x,y);
        else if(op==3) T.modify(x,y);
    }
    return 0;
}
```

堆

左偏树

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
constexpr int N=100005;
struct Leftist_Tree
{
    int n;
    struct Node
    {
        int lc,rc;
        int dis,val;
    }tree[N];
    int fa[N];
    bool deleted[N];
    Leftist_Tree(int _n):n(_n)
    {
        for(int i=0;i<=_n;i++)
            fa[i]=i;
        fill(deleted+1,deleted+_n+1,false);
        memset(tree,0,sizeof(tree));
        tree[0].dis=-1;
    }
    void init(int _n)
    {
        n=_n;
        for(int i=0;i<=_n;i++)
            fa[i]=i;
        fill(deleted+1,deleted+_n+1,false);
        memset(tree,0,sizeof(tree));
        tree[0].dis=-1;
        return;
    }
    int getf(int v)
    {
        if(v==fa[v]) return v;
        fa[v]=getf(fa[v]);
        return fa[v];
    }
    int merge(int u,int v)
    {
        if(!u) return v;
        if(!v) return u;
        if(tree[u].val>tree[v].val||(tree[u].val==tree[v].val&&u>v)) swap(u,v);
        tree[u].rc=merge(tree[u].rc,v);
        if(tree[tree[u].lc].dis<tree[tree[u].rc].dis)
swap(tree[u].lc,tree[u].rc);
        fa[u]=fa[tree[u].lc]=fa[tree[u].rc]=u;
        tree[u].dis=tree[tree[u].rc].dis+1;
        return u;
    }
    void unity(int x,int y)
```

```
    {
        if(deleted[x]||deleted[y]) return;
        x=getf(x),y=getf(y);
        if(x!=y) fa[x]=fa[y]=merge(x,y);
        return;
    }
    int top(int x)
    {
        if(deleted[x]) return -1;
        return tree[getf(x)].val;
    }
    void pop(int x)
    {
        if(deleted[x]) return;
        x=getf(x);
        deleted[x]=true;
        fa[tree[x].lc]=tree[x].lc,fa[tree[x].rc]=tree[x].rc;
        fa[x]=merge(tree[x].lc,tree[x].rc);
        return;
    }
};
int n,m;
int a[N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    cin>>n>>m;
    Leftist_Tree T(n);
    for(int i=1;i<=n;i++)
        cin>>a[i];
    for(int i=1;i<=n;i++)
        T.tree[i].val=a[i];
    for(int i=1;i<=m;i++)
    {
        int op,x,y;
        cin>>op;
        if(op==1)
        {
            cin>>x>>y;
            T.unity(x,y);
        }
        else if(op==2)
        {
            cin>>x;
            int res=T.top(x);
            T.pop(x);
            cout<<res<<"\n";
        }
    }
    return 0;
}
```

图论

最短路

Dijkstra + pb_ds O(nlogn + m)

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<ext/pb_ds/priority_queue.hpp>
using namespace std;
using namespace __gnu_pbds;
const int N=2505;
const long long INF=4557430888798830399;
int n,m;
vector<pair<int,int>>G[N];
vector<long long> dijkstra(int s)
{
    vector<long long>dis(n+1,INF);
    __gnu_pbds::priority_queue<pair<long long,int>,greater<pair<long
long,int>>,pairing_heap_tag> q;
    vector<__gnu_pbds::priority_queue<pair<long long,int>,greater<pair<long
long,int>>,pairing_heap_tag>::point_iterator> id(n+1,nullptr);
    dis[s]=0;
    static bool vis[N];
    fill(vis+1,vis+n+1,false);
    id[s]=q.push(make_pair(dis[s],s));
    while(!q.empty())
    {
        int u=q.top().second;
        q.pop();
        if(vis[u]) continue;
        vis[u]=true;
        for(auto [v,w]:G[u])
        {
            if(dis[v]>dis[u]+w)
            {
                dis[v]=dis[u]+w;
                if(id[v]==nullptr) id[v]=q.push(make_pair(dis[v],v));
                else q.modify(id[v],make_pair(dis[v],v));
            }
        }
    }
    return dis;
}
int main()
{
    int s,t;
    scanf("%d%d%d%d",&n,&m,&s,&t);
    for(int i=1;i<=m;i++)
    {
        int x,y,z;
```

```
        scanf("%d%d%d",&x,&y,&z);
        G[x].emplace_back(y,z);
        G[y].emplace_back(x,z);
    }
    vector<long long>dis=dijkstra(s);
    printf("%lld",dis[t]);
    return 0;
}
```

Johnson 全源最短路 O(n^2logm) ~ O(nmlogm)

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<queue>
#include<algorithm>
#include<ext/pb_ds/priority_queue.hpp>
using namespace std;
using namespace __gnu_pbds;
const int N=3005;
const long long INF=4557430888798830399;
int n,m;
vector<pair<int,int>>G[N];
long long h[N];
bool inq[N];
int num[N];
bool spfa(int s)
{
    fill(h,h+n+1,INF);
    h[s]=0;
    fill(num,num+n+1,0);
    fill(inq,inq+n+1,false);
    queue<int>q;
    q.emplace(s);
    inq[s]=true;
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        inq[u]=false;
        for(auto [v,w]:G[u])
            if(h[v]>h[u]+w)
            {
                h[v]=h[u]+w;
                if(!inq[v])
                {
                    q.emplace(v);
                    inq[v]=true;
                    num[v]=num[u]+1;
                    if(num[v]>n) return false;
                }
            }
    }
    return true;
```

```
}
vector<long long> dijkstra(int s)
{
    vector<long long>dis(n+1,INF);
    __gnu_pbds::priority_queue<pair<long long,int>,greater<pair<long
long,int>>,pairing_heap_tag> q;
    vector<__gnu_pbds::priority_queue<pair<long long,int>,greater<pair<long
long,int>>,pairing_heap_tag>::point_iterator> id(n+1,nullptr);
    dis[s]=0;
    static bool vis[N];
    fill(vis+1,vis+n+1,false);
    id[s]=q.push(make_pair(dis[s],s));
    while(!q.empty())
    {
        int u=q.top().second;
        q.pop();
        if(vis[u]) continue;
        vis[u]=true;
        for(auto [v,w]:G[u])
        {
            if(dis[v]>dis[u]+w)
            {
                dis[v]=dis[u]+w;
                if(id[v]==nullptr) id[v]=q.push(make_pair(dis[v],v));
                else q.modify(id[v],make_pair(dis[v],v));
            }
        }
    }
    return dis;
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
    {
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        G[x].emplace_back(y,z);
    }
    for(int i=1;i<=n;i++)
        G[0].emplace_back(i,0);
    if(!spfa(0))
    {
        printf("-1");
        return 0;
    }
    for(int u=1;u<=n;u++)
        for(auto &[v,w]:G[u])
            w+=h[u]-h[v];
    for(int i=1;i<=n;i++)
    {
        vector<long long>dis=dijkstra(i);
        for(int j=1;j<=n;j++)
            if(dis[j]<INF) dis[j]+=h[j]-h[i];
        long long ans=0;
        for(int j=1;j<=n;j++)
```

```
        {
            if(dis[j]>=INF) ans+=j*1000000000LL;
            else ans+=j*dis[j];
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

连通性相关

强连通分量

Trajan O(n+m)

```
struct Trajan_SCC
{
    int n;
    Trajan_SCC(int _n=0):n(_n){}
    void init(int _n)
    {
        n=_n;
        return;
    }
    vector<int>G[N];
    void add_edge(int u,int v)
    {
        G[u].emplace_back(v);
        return;
    }
    int dfn[N],low[N],idx;
    bool ins[N];
    stack<int>s;
    int bel[N],tot;
    vector<int>block[N];
    void dfs(int u)
    {
        dfn[u]=low[u]=++idx;
        ins[u]=true;
        s.push(u);
        for(int v:G[u])
        {
            if(!dfn[v])
            {
                dfs(v);
                low[u]=min(low[u],low[v]);
            }
            else if(ins[v]) low[u]=min(low[u],dfn[v]);
        }
        if(dfn[u]==low[u])
        {
            tot++;
```

```
            block[tot].clear();
            while(!s.empty()&&s.top()!=u)
            {
                int v=s.top();
                s.pop();
                ins[v]=false;
                bel[v]=tot;
                block[tot].push_back(v);
            }
            s.pop();
            ins[u]=false;
            bel[u]=tot;
            block[tot].push_back(u);
        }
        return;
    }
    void tarjan()
    {
        fill(dfn+1,dfn+n+1,0);
        fill(low+1,low+n+1,0);
        idx=0;
        fill(bel+1,bel+n+1,0);
        tot=0;
        fill(ins+1,ins+n+1,false);
        for(int i=1;i<=n;i++)
            if(!dfn[i]) dfs(i);
        return;
    }
};
```

Kosaraju O(n^2/w)

```
struct Kosaraju_SCC
{
    int n;
    Kosaraju_SCC(int _n=0):n(_n){}
    void init(int _n)
    {
        n=_n;
        return;
    }
    bitset<N>mp[N],rmp[N];
    vector<int>block[N];
    int bel[N],tot;
    void add_edge(int u,int v)
    {
        mp[u][v]=1;
        rmp[v][u]=1;
        return;
    }
    bitset<N>vis;
    vector<int>out;
    void dfs1(int x)
    {
```

```
        vis[x]=1;
        for(auto now=mp[x]&(~vis);now.any();now=now&(~vis))
            dfs1(now._Find_first());
        out.push_back(x);
        return;
    }
    void dfs2(int x)
    {
        block[tot].push_back(x);
        vis[x]=1;
        bel[x]=tot;
        for(auto now=rmp[x]&(~vis);now.any();now=now&(~vis))
            dfs2(now._Find_first());
        return;
    }
    void kosaraju()
    {
        out.clear();
        vis.reset();
        for(int i=1;i<=n;i++)
            if(!vis[i]) dfs1(i);
        vis.reset();
        fill(bel+1,bel+n+1,0);
        tot=0;
        for(int i=out.size()-1;i>=0;i--)
        {
            int u=out[i];
            if(!vis[u])
            {
                tot++;
                block[tot].clear();
                dfs2(u);
            }
        }
        return;
    }
};
```

割点

```
struct Tarjan_Cut
{
    int n;
    Tarjan_Cut(int _n=0):n(_n){}
    void init(int _n)
    {
        n=_n;
        return;
    }
    vector<int>G[N];
    void add_edge(int u,int v)
    {
        G[u].emplace_back(v);
        G[v].emplace_back(u);
```

```
        return;
    }
    int dfn[N],low[N],idx;
    bool iscut[N];
    void dfs(int u,int father)
    {
        int children=0;
        dfn[u]=low[u]=++idx;
        for(int v:G[u])
        {
            if(!dfn[v])
            {
                children++;
                dfs(v,u);
                low[u]=min(low[u],low[v]);
                if(u!=father&&low[v]>=dfn[u]) iscut[u]=true;
                else if(u==father&&children>=2) iscut[u]=true;
            }
            else if(v!=father) low[u]=min(low[u],dfn[v]);
        }
        return;
    }
    int cut[N],tot;
    void tarjan()
    {
        fill(dfn+1,dfn+n+1,0);
        fill(low+1,low+n+1,0);
        idx=0;
        fill(iscut+1,iscut+n+1,false);
        for(int i=1;i<=n;i++)
            if(!dfn[i]) dfs(i,i);
        tot=0;
        for(int i=1;i<=n;i++)
            if(iscut[i]) cut[++tot]=i;
        return;
    }
};
```

桥

```
struct Tarjan_Bridge
{
    int n,m;
    Tarjan_Bridge(int _n=0):n(_n),m(0){}
    void init(int _n)
    {
        n=_n,m=0;
        return;
    }
    vector<pair<int,int>>G[N];
    pair<int,int>edge[M];
    void add_edge(int u,int v)
    {
        m++;
```

```
        edge[m]=make_pair(u,v);
        G[u].emplace_back(v,m);
        G[v].emplace_back(u,m);
        return;
    }
    int dfn[N],low[N],idx;
    bool isbridge[M];
    void dfs(int u,int prev)
    {
        low[u]=dfn[u]=++idx;
        for(auto [v,id]:G[u])
        {
            if(!dfn[v])
            {
                dfs(v,id);
                low[u]=min(low[u],low[v]);
                if(low[v]>dfn[u]) isbridge[id]=true;
            }
            else if(dfn[v]<dfn[u]&&id!=prev) low[u]=min(low[u],dfn[v]);
        }
        return;
    }
    pair<int,int>bridge[M];
    int tot;
    void tarjan()
    {
        fill(dfn+1,dfn+n+1,0);
        fill(low+1,low+n+1,0);
        idx=0;
        fill(isbridge+1,isbridge+m+1,false);
        for(int i=1;i<=n;i++)
            if(!dfn[i]) dfs(i,0);
        tot=0;
        for(int i=1;i<=m;i++)
            if(isbridge[i]) bridge[++tot]=edge[i];
        return;
    }
};
```

边双连通分量

```
struct Trajan_EBC
{
    int n,m;
    Trajan_EBC(int _n=0):n(_n),m(0){}
    void init(int _n)
    {
        n=_n,m=0;
        return;
    }
    vector<pair<int,int>>G[N];
    void add_edge(int u,int v)
    {
        m++;
```

```
        G[u].emplace_back(v,m);
        G[v].emplace_back(u,m);
        return;
    }
    int dfn[N],low[N],idx;
    bool ins[N];
    stack<int>s;
    int bel[N],tot;
    vector<int>block[N];
    void dfs(int u,int prev)
    {
        dfn[u]=low[u]=++idx;
        ins[u]=true;
        s.push(u);
        for(auto [v,id]:G[u])
        {
            if(id==prev) continue;
            if(!dfn[v])
            {
                dfs(v,id);
                low[u]=min(low[u],low[v]);
            }
            else if(ins[v]) low[u]=min(low[u],dfn[v]);
        }
        if(dfn[u]==low[u])
        {
            tot++;
            block[tot].clear();
            while(!s.empty()&&s.top()!=u)
            {
                int v=s.top();
                s.pop();
                ins[v]=false;
                bel[v]=tot;
                block[tot].push_back(v);
            }
            s.pop();
            ins[u]=false;
            bel[u]=tot;
            block[tot].push_back(u);
        }
        return;
    }
    void tarjan()
    {
        fill(dfn+1,dfn+n+1,0);
        fill(low+1,low+n+1,0);
        idx=0;
        fill(bel+1,bel+n+1,0);
        tot=0;
        fill(ins+1,ins+n+1,false);
        for(int i=1;i<=n;i++)
            if(!dfn[i]) dfs(i,0);
        return;
    }
};
```

点双连通分量

```
constexpr int N=500005;
struct Trajan_PBC
{
    int n;
    Trajan_PBC(int _n=0):n(_n){}
    void init(int _n)
    {
        n=_n;
        return;
    }
    vector<int>G[N];
    void add_edge(int u,int v)
    {
        G[u].emplace_back(v);
        G[v].emplace_back(u);
        return;
    }
    int dfn[N],low[N],idx;
    vector<int>block[N];
    int tot;
    stack<int>s;
    void dfs(int u,int father)
    {
        int son=0;
        dfn[u]=low[u]=++idx;
        s.emplace(u);
        for(int v:G[u])
        {
            if(!dfn[v])
            {
                son++;
                dfs(v,u);
                low[u]=min(low[u],low[v]);
                if(low[v]>=dfn[u])
                {
                    tot++;
                    block[tot].clear();
                    while(!s.empty()&&s.top()!=v)
                    {
                        block[tot].emplace_back(s.top());
                        s.pop();
                    }
                    block[tot].emplace_back(v);
                    s.pop();
                    block[tot].push_back(u);
                }
            }
            else if(v!=father) low[u]=min(low[u],dfn[v]);
        }
        if(father==0&&son==0)
        {
```

```
            tot++;
            block[tot].clear();
            s.pop();
            block[tot].emplace_back(u);
        }
        return;
    }
    void tarjan()
    {
        fill(dfn+1,dfn+n+1,0);
        fill(low+1,low+n+1,0);
        tot=0;
        idx=0;
        for(int i=1;i<=n;i++)
            if(!dfn[i]) dfs(i,0);
        return;
    }
};
```

2-SAT

有 n 个布尔变量 $x_1, x_2, \ldots, x_n$。有 m 个限制，每个限制形如：$x_a = b \vee x_c = d$。

构造一组合法的取值方案，使得所有的限制都被满足。

```
struct Two_Sat
{
    static constexpr int N=100005;
    int n,tot;
    array<int,2>id[N];
    Trajan_SCC scc;
    bool val[N];
    Two_Sat(int _n=0)
    {
        n=_n;
        tot=0;
        for(int i=1;i<=_n;i++)
            id[i][0]=++tot,id[i][1]=++tot;
        scc.resize(tot);
    }
    int new_node()
    {
        n++;
        id[n][0]=++tot,id[n][1]=++tot;
        scc.resize(tot);
        return n;
    }
    void add(int u,bool a,int v,bool b)
    {
        if(u==v)
        {
            if(a!=b) return;
            scc.add_edge(id[u][a^1],id[v][a]);
            return;
```

```
        }
        scc.add_edge(id[u][a^1],id[v][b]);
        scc.add_edge(id[v][b^1],id[u][a]);
        return;
    }
    bool twosat()
    {
        scc.tarjan();
        for(int i=1;i<=n;i++)
            if(scc.bel[id[i][0]]==scc.bel[id[i][1]]) return false;
        for(int i=1;i<=n;i++)
            val[i]=scc.bel[id[i][0]]>scc.bel[id[i][1]];
        return true;
    }
};
```

圆方树

[APIO2018] 铁人两项

> 给定一张简单无向图，问有多少对三元组 $\langle s,c,f\rangle$（s,c,f 互不相同）使得存在一条简单路径从 s 出发，经过 c 到达 f。

考虑两圆点在圆方树上的路径，与路径上经过的方点相邻的圆点的集合，就等于原图中两点简单路径上的点集。

回到题目，考虑固定 s 和 f，求合法的 c 的数量，显然有合法 c 的数量等于 s,f 之间简单路径的并集的点数减 2（去掉 s,f 本身）。

那么，对原图建出圆方树后，两点之间简单路径的点数，就和它们在圆方树上路径经过的方点（点双）和圆点的个数有关。

接下来是圆方树的一个常用技巧：路径统计时，点赋上合适的权值。
本题中，每个方点的权值为对应点双的大小，而每个圆点权值为 -1。

这样赋权后则有两圆点间圆方树上路径点权和，恰好等于原图中简单路径并集大小减 2。

问题转化为统计圆方树上 $\sum$ 两圆点路径权值和。

换个角度考虑，改为统计每一个点对答案的贡献，即权值乘以经过它的路径条数，这可以通过简单的树形 DP 求出。

最后，不要忘记处理图不连通的情况。下面是对应代码：

```
#include<iostream>
#include<cstdio>
#include<stack>
#include<vector>
using namespace std;
const int N=100005,M=200005;
int n,m,q;
struct Edge
{
    int to,next;
}edge[M*2];
int head[N],cnt;
```

```
void add_edge(int u,int v)
{
    cnt++;
    edge[cnt].to=v;
    edge[cnt].next=head[u];
    head[u]=cnt;
    return;
}
int dfn[N],low[N],idx;
int a[N*2],tot;
stack<int>s;
vector<int>G[N*2];
void tarjan(int u)
{
    dfn[u]=low[u]=++idx;
    s.emplace(u);
    for(int i=head[u];i;i=edge[i].next)
    {
        int v=edge[i].to;
        if(!dfn[v])
        {
            tarjan(v);
            low[u]=min(low[u],low[v]);
            if(low[v]==dfn[u])
            {
                tot++;
                while(!s.empty()&&s.top()!=v)
                {
                    G[tot].emplace_back(s.top());
                    G[s.top()].emplace_back(tot);
                    a[tot]++;
                    s.pop();
                }
                G[tot].emplace_back(v);
                G[v].emplace_back(tot);
                a[tot]++;
                s.pop();
                G[tot].emplace_back(u);
                G[u].emplace_back(tot);
                a[tot]++;
            }
        }
        else low[u]=min(low[u],dfn[v]);
    }
    return;
}
int siz[N+N];
void init(int u,int father)
{
    if(u<=n) siz[u]=1;
    else siz[u]=0;
    for(int v:G[u])
    {
        if(v==father) continue;
        init(v,u);
        siz[u]+=siz[v];
```

```
    }
    return;
}
long long ans;
void dfs(int u,int father,int topf)
{
    int s=0;
    if(u<=n) s=1;
    for(int v:G[u])
    {
        if(v==father) continue;
        dfs(v,u,topf);
        ans+=2ll*s*siz[v]*a[u];
        s+=siz[v];
    }
    ans+=2ll*s*(siz[topf]-s)*a[u];
    return;
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        add_edge(x,y);
        add_edge(y,x);
    }
    idx=0,tot=n;
    for(int i=1;i<=n;i++)
        if(!dfn[i])
        {
            tarjan(i);
            while(!s.empty()) s.pop();
        }
    for(int i=1;i<=n;i++)
        a[i]=-1;
    for(int i=1;i<=n;i++)
        if(!siz[i])
        {
            init(i,0);
            dfs(i,0,i);
        }
    printf("%lld",ans);
    return 0;
}
```

树上问题

最近公共祖先

标准 RMQ

```
#include<iostream>
#include<cstdio>
#include<vector>
using namespace std;
const int N=500005;
int n,q,s;
vector<int>G[N];
int dfn[N],idx;
int mn[N][20],lg2[N];
void dfs(int u,int father)
{
    dfn[u]=++idx;
    mn[idx][0]=father;
    for(int v:G[u])
    {
        if(v==father) continue;
        dfs(v,u);
    }
    return;
}
int lca(int u,int v)
{
    if(u==v) return u;
    u=dfn[u],v=dfn[v];
    if(u>v) swap(u,v);
    u++;
    int d=lg2[v-u+1];
    if(dfn[mn[u][d]]<dfn[mn[v-(1<<d)+1][d]]) return mn[u][d];
    else return mn[v-(1<<d)+1][d];
}
void init_lca(int n)
{
    lg2[0]=-1;
    for(int i=1;i<=n;i++)
        lg2[i]=lg2[i/2]+1;
    for(int j=1;(1<<j)<=n;j++)
        for(int i=1;i+(1<<j)-1<=n;i++)
            if(dfn[mn[i][j-1]]<dfn[mn[i+(1<<(j-1))][j-1]]) mn[i][j]=mn[i][j-1];
            else mn[i][j]=mn[i+(1<<(j-1))][j-1];
    return;
}
int main()
{
    scanf("%d%d%d",&n,&q,&s);
    for(int i=1;i<n;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        G[x].emplace_back(y);
        G[y].emplace_back(x);
```

```
    }
    dfs(s,0);
    init_lca(n);
    while(q--)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        printf("%d\n",lca(u,v));
    }
    return 0;
}
```

树链剖分

重链剖分

[LibreOJ #139 树链剖分]

> 给定一棵 n 个节点的树，初始时该树的根为 1 号节点，每个节点有一个给定的权值。下面依次进行 m 个操作，操作分为如下五种类型：
>
> - 换根：将一个指定的节点设置为树的新根。
> - 修改路径权值：给定两个节点，将这两个节点间路径上的所有节点权值（含这两个节点）增加一个给定的值。
> - 修改子树权值：给定一个节点，将以该节点为根的子树内的所有节点权值增加一个给定的值。
> - 询问路径：询问某条路径上节点的权值和。
> - 询问子树：询问某个子树内节点的权值和。

```
#include<iostream>
#include<cstdio>
#include<vector>
using namespace std;
const int N=100005;
const long long INF=4557430888798830399;
int n,m;
vector<int>G[N];
int dep[N],fa[N];
int f[N][17],son[N],siz[N];
int a[N],w[N];
void dfs1(int u,int father)
{
    siz[u]=1;
    dep[u]=dep[father]+1;
    fa[u]=father;
    f[u][0]=father;
    for(int i=1;(1<<i)<=n;i++)
        f[u][i]=f[f[u][i-1]][i-1];
    for(int v:G[u])
    {
        if(v==father) continue;
```

```
        dfs1(v,u);
        siz[u]+=siz[v];
        if(siz[v]>siz[son[u]]) son[u]=v;
    }
    return;
}
int top[N],Index,dfn[N];
void dfs2(int u,int father)
{
    top[u]=father;
    dfn[u]=++Index;
    w[Index]=a[u];
    if(!son[u]) return;
    dfs2(son[u],father);
    for(int v:G[u])
    {
        if(v==fa[u]||v==son[u]) continue;
        dfs2(v,v);
    }
    return;
}
struct Segment_Tree
{
    #define ls i*2
    #define rs i*2+1
    struct Node
    {
        int l,r;
        long long sum,tag;
    }tree[N<<2];
    void push_up(int i)
    {
        tree[i].sum=tree[ls].sum+tree[rs].sum;
        return;
    }
    void add(int i,long long val)
    {
        tree[i].tag+=val;
        tree[i].sum+=val*(tree[i].r-tree[i].l+1);
        return;
    }
    void push_down(int i)
    {
        if(!tree[i].tag) return;
        add(ls,tree[i].tag);
        add(rs,tree[i].tag);
        tree[i].tag=0;
        return;
    }
    void build(int i,int l,int r)
    {
        tree[i].l=l,tree[i].r=r;
        tree[i].tag=0;
        if(l==r)
        {
            tree[i].sum=w[l];
```

```
            return;
        }
        int mid=(l+r)/2;
        build(ls,l,mid);
        build(rs,mid+1,r);
        push_up(i);
        return;
    }
    void add(int i,int l,int r,long long val)
    {
        if(l<=tree[i].l&&tree[i].r<=r) return add(i,val);
        push_down(i);
        if(l<=tree[ls].r) add(ls,l,r,val);
        if(r>=tree[rs].l) add(rs,l,r,val);
        push_up(i);
        return;
    }
    long long query(int i,int l,int r)
    {
        if(l<=tree[i].l&&tree[i].r<=r) return tree[i].sum;
        push_down(i);
        long long res=0;
        if(l<=tree[ls].r) res+=query(ls,l,r);
        if(r>=tree[rs].l) res+=query(rs,l,r);
        return res;
    }
}T;

int rt;
void modify(int u,int v,long long val)
{
    if(dep[u]<dep[v]) swap(u,v);
    while(top[u]!=top[v])
    {
        if(dep[top[u]]<dep[top[v]]) swap(u,v);
        T.add(1,dfn[top[u]],dfn[u],val);
        u=fa[top[u]];
    }
    if(dep[u]>dep[v]) swap(u,v);
    T.add(1,dfn[u],dfn[v],val);
    return;
}
long long getsum(int u,int v)
{
    if(dep[u]<dep[v]) swap(u,v);
    long long res=0;
    while(top[u]!=top[v])
    {
        if(dep[top[u]]<dep[top[v]]) swap(u,v);
        res+=T.query(1,dfn[top[u]],dfn[u]);
        u=fa[top[u]];
    }
    if(dep[u]>dep[v]) swap(u,v);
    res+=T.query(1,dfn[u],dfn[v]);
    return res;
}
```

```
int getfa(int u,int k)
{
    for(int i=__lg(n);i>=0;i--)
        if((k>>i)&1) u=f[u][i];
    return u;
}
void modify(int u,long long val)
{
    if(u==rt)
    {
        T.add(1,1,n,val);
        return;
    }
    else if(dep[u]<dep[rt])
    {
        int v=getfa(rt,dep[rt]-dep[u]-1);
        if(fa[v]!=u)
        {
            T.add(1,dfn[u],dfn[u]+siz[u]-1,val);
            return;
        }
        else if(dfn[v]+siz[v]<=n)
        {
            T.add(1,1,dfn[v]-1,val);
            T.add(1,dfn[v]+siz[v],n,val);
            return;
        }
        else
        {
            T.add(1,1,dfn[v]-1,val);
            return;
        }
    }
    else
    {
        T.add(1,dfn[u],dfn[u]+siz[u]-1,val);
        return;
    }
}
long long getsum(int u)
{
    if(u==rt) return T.query(1,1,n);
    else if(dep[u]<dep[rt])
    {
        int v=getfa(rt,dep[rt]-dep[u]-1);
        if(fa[v]!=u) return T.query(1,dfn[u],dfn[u]+siz[u]-1);
        else if(dfn[v]+siz[v]<=n) return
T.query(1,1,dfn[v]-1)+T.query(1,dfn[v]+siz[v],n);
        else return T.query(1,1,dfn[v]-1);
    }
    else return T.query(1,dfn[u],dfn[u]+siz[u]-1);
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
```

```
    cin>>n;
    for(int i=1;i<=n;i++)
        cin>>a[i];
    for(int i=2;i<=n;i++)
    {
        int f;
        cin>>f;
        G[i].emplace_back(f);
        G[f].emplace_back(i);
    }
    dfs1(1,0);
    dfs2(1,1);
    T.build(1,1,n);
    rt=1;
    cin>>m;
    while(m--)
    {
        int op;
        cin>>op;
        if(op==1) cin>>rt;
        else if(op==2)
        {
            int x,y;
            long long k;
            cin>>x>>y>>k;
            modify(x,y,k);
        }
        else if(op==3)
        {
            int x;
            long long k;
            cin>>x>>k;
            modify(x,k);
        }
        else if(op==4)
        {
            int x,y;
            cin>>x>>y;
            cout<<getsum(x,y)<<"\n";
        }
        else if(op==5)
        {
            int x;
            cin>>x;
            cout<<getsum(x)<<"\n";
        }
    }
    return 0;
}
```

长链剖分

优化 DP

[Luogu P5904 [POI2014] HOT-Hotels 加强版]

> 给定一棵树，在树上选 3 个点，要求两两距离相等，求方案数。

设 $f_{i,j}$ 为满足 x 在 i 的子树中且 $d(x,i)=j$ 的 x 的个数， $g_{i,j}$ 为满足 x,y 在 i 的子树中且 $d(\operatorname{lca}(x,y),x)=d(\operatorname{lca}(x,y),y)=d(\operatorname{lca}(x,y),i)+j$ 的无序数对 (x,y) 的个数。

有转移:

$$
\begin{aligned}
ans &\leftarrow g_{i,0}\\
\text{ans} &\leftarrow \sum_{x,y\in \text{son}(i),x\neq y} f_{x,j-1}\times g_{y,j+1}\\
g_{i,j} &\leftarrow \sum_{x,y\in \text{son}(i),x\neq y} f_{x,j-1}\times f_{y,j-1}\\
g_{i,j} &\leftarrow \sum_{x\in \text{son}(i)} g_{x,j+1}\\
f_{i,j} &\leftarrow \sum_{x\in \text{son}(i)} f_{x,j-1}
\end{aligned}
$$

显然这可以利用前缀和，或者说是所有儿子「向 i 合并」，做到 $\mathcal{O}(n)$ 转移，总时间复杂度 $\mathcal{O}\left(n^2\right)$。注意到这里的信息都是以深度为下标的，那么可以利用长链剖分将复杂度降为均摊 $\mathcal{O}(1)$，总时间复杂度即可降为 $\mathcal{O}(n)$。

```cpp
#include<iostream>
#include<cstdio>
#include<vector>
using namespace std;
const int N=100005;
int n;
vector<int>G[N];
int dep[N],son[N],len[N];
void dfs1(int u,int father)
{
    for(int v:G[u])
    {
        if(v==father)continue;
        dfs1(v,u);
        len[u]=max(len[u],len[v]);
        if(len[v]>len[son[u]])son[u]=v;
    }
    len[u]=len[son[u]]+1;
    return;
}
long long *f[N],*g[N];
long long tmp[N<<2],*id=tmp;
long long ans;
void dfs(int u,int father)
{
    f[u][0]=1;
    if(!son[u]) return;
```

```
    f[son[u]]=f[u]+1;
    g[son[u]]=g[u]-1;
    dfs(son[u],u);
    ans+=g[u][0];
    for(int v:G[u])
    {
        if(v==father||v==son[u])continue;
        f[v]=id;id+=len[v]<<1;
        g[v]=id;id+=len[v]<<1;
        dfs(v,u);
        for(int j=0;j<len[v];j++)
        {
            if(j) ans+=f[u][j-1]*g[v][j];
            ans+=g[u][j+1]*f[v][j];
        }
        for(int j=0;j<len[v];j++)
        {
            g[u][j+1]+=f[u][j+1]*f[v][j];
            if(j) g[u][j-1]+=g[v][j];
            f[u][j+1]+=f[v][j];
        }
    }
    return;
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    cin>>n;
    for(int i=1;i<n;i++)
    {
        int u,v;
        cin>>u>>v;
        G[u].emplace_back(v);
        G[v].emplace_back(u);
    }
    dfs1(1,0);
    f[1]=id,id+=len[1]<<1;
    g[1]=id,id+=len[1]<<1;
    dfs(1,0);
    cout<<ans;
    return 0;
}
```

求 k 级祖先

```
const int N=500005;
int n;
vector<int>G[N];
int f[N][19],dep[N];
int dis[N],top[N],son[N],len[N],fa[N];
void dfs1(int u,int father)
{
    dis[u]=dep[u]=dep[father]+1;
```

```
    f[u][0]=father;
    fa[u]=father;
    for(int i=1;(1<<i)<=n;i++)
        f[u][i]=f[f[u][i-1]][i-1];
    for(int v:G[u])
    {
        if(v==father) continue;
        dfs1(v,u);
        if(dis[v]>dis[son[u]]) son[u]=v;
    }
    if(son[u]) dis[u]=dis[son[u]];
    return;
}
void dfs2(int u,int father)
{
    top[u]=father;
    len[u]=dis[u]-dep[top[u]]+1;
    if(!son[u]) return;
    dfs2(son[u],father);
    for(int v:G[u])
    {
        if(v==fa[u]||v==son[u]) continue;
        dfs2(v,v);
    }
    return;
}
vector<int> up[N],down[N];
int kth_ancestor(int x,int k)
{
    if(k>=dep[x]) return 0;
    if(k==0) return x;
    int h=__lg(k);
    x=f[x][h];
    k^=1<<h;
    if(k==0) return x;
    if(k>dep[x]-dep[top[x]]) return up[top[x]][k-dep[x]+dep[top[x]]];
    else if(k<dep[x]-dep[top[x]]) return down[top[x]][dep[x]-dep[top[x]]-k];
    else return top[x];
}
void init(int rt)
{
    dfs1(rt,0);
    dfs2(rt,rt);
    for(int u=1;u<=n;u++)
        if(u==top[u])
        {
            for(int i=1,x=u;i<=len[u]&&x;i++) up[u].push_back(x),x=fa[x];
            for(int i=1,x=u;i<=len[u];i++) down[u].push_back(x),x=son[x];
        }
    return;
}
```

树哈希

UOJ #763. 树哈希

```
const unsigned long long
mask=chrono::steady_clock::now().time_since_epoch().count();
const unsigned long long S=1;
unsigned long long shift(unsigned long long x)
{
    x^=mask;
    x^=x<<13;
    x^=x>>7;
    x^=x<<17;
    x^=mask;
    return x;
}
int n;
vector<int>G[N];
unsigned long long f[N];
void dfs(int u,int father)
{
    f[u]=S;
    for(int v:G[u])
    {
        if(v==father) continue;
        dfs(v,u);
        f[u]+=shift(f[v]);
    }
    return;
}
```

虚树

```
void build(vector<int>h)
{
    sort(h.begin(),h.end(),[&](const int &x,const int &y){return dfn[x]
<dfn[y];});
    static int s[N],top;
    top=0;
    s[++top]=1;
    E[1].clear();
    for(int i=0;i<(int)h.size();i++)
        if(h[i]!=1)
        {
            int l=lca(h[i],s[top]);
            if(l!=s[top])
            {
                while(dfn[l]<dfn[s[top-1]])
                {
                    E[s[top]].emplace_back(s[top-1]),E[s[top-
1]].emplace_back(s[top]);
                    top--;
                }
```

```
                if(dfn[l]>dfn[s[top-1]])
                {
                    E[l].clear();
                    E[l].emplace_back(s[top]),E[s[top]].emplace_back(l);
                    top--;
                    s[++top]=l;
                }
                else
                {
                    E[s[top-
1]].emplace_back(s[top]),E[s[top]].emplace_back(s[top-1]);
                    top--;
                }
            }
            E[h[i]].clear();
            s[++top]=h[i];
        }
    for(int i=1;i<top;i++)
        E[s[i]].emplace_back(s[i+1]),E[s[i+1]].emplace_back(s[i]);
    return;
}
```

树分治

点分治

Luogu P3806 【模板】点分治 1

> 给定一棵有 n 个点的树，询问树上距离为 k 的点对是否存在。

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
using namespace std;
const int N=30005;
const int INF=1061109567;
int n,m;
int k[N];
vector<pair<int,int>>G[N];
int rt,tot;
int siz[N],mx[N];
bool book[N];
void getroot(int u,int father)
{
    siz[u]=1,mx[u]=0;
    for(auto [v,w]:G[u])
    {
        if(book[v]) continue;
        if(v==father) continue;
        getroot(v,u);
        siz[u]+=siz[v];
        mx[u]=max(mx[u],siz[v]);
```

```
    }
    mx[u]=max(mx[u],tot-siz[u]);
    if(rt==0||mx[u]<mx[rt]) rt=u;
    return;
}
vector<int>d;
int dis[N];
void getdis(int u,int father)
{
    d.push_back(dis[u]);
    for(auto [v,w]:G[u])
    {
        if(book[v]) continue;
        if(v==father) continue;
        dis[v]=dis[u]+w;
        getdis(v,u);
    }
    return;
}
int ans[N];
void calc(int u,int len,int op)
{
    dis[u]=len;
    d.clear();
    getdis(u,0);
    sort(d.begin(),d.end());
    for(int i=1;i<=m;i++)
    {
        int l=0,r=d.size()-1;
        while(l<r)
        {
            if(d[l]+d[r]<=k[i])
            {
                if(d[l]+d[r]==k[i]) ans[i]+=op;
                l++;
            }
            else r--;
        }
    }
    return;
}
void dfs(int u)
{
    book[u]=true;
    calc(u,0,1);
    for(auto [v,w]:G[u])
    {
        if(book[v]) continue;
        calc(v,w,-1);
        tot=siz[v],rt=0;
        getroot(v,0);
        dfs(rt);
    }
    return;
}
int main()
```

```
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    cin>>n>>m;
    for(int i=1;i<n;i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        G[u].emplace_back(v,w);
        G[v].emplace_back(u,w);
    }
    for(int i=1;i<=m;i++)
        cin>>k[i];
    rt=0,tot=n;
    getroot(1,0);
    dfs(rt);
    for(int i=1;i<=m;i++)
        if(ans[i]>0) cout<<"Yes\n";
        else cout<<"No\n";
    return 0;
}
```

动态树分治

点分树

Luogu P6329【模板】点分树 | 震波

> 给定一棵有 n 个结点的树，树上每个结点都有一个权值 $v[x]$。实现以下两种操作：
>
> 1. 询问与结点 x 距离不超过 y 的结点权值和；
> 2. 修改结点 x 的点权为 y，即 $v[x]=y$。

```
#include<iostream>
#include<cstdio>
#include<cassert>
#include<vector>
using namespace std;
template<typename T>
struct Fenwick_Tree
{
private:
    size_t n;
    vector<T>C;
public:
    Fenwick_Tree(size_t _n=0):n(_n),C(vector<T>(_n)){}
    size_t size()const
    {
        return n;
    }
    int lowbit(int x)const
    {
        return x&-x;
```

```
    }
    void init(size_t _n)
    {
        n=_n;
        C=vector<T>(_n);
        return;
    }
    void add(size_t x,T y)
    {
        x++;
        assert(1<=x&&x<=n);
        for(size_t i=x;i<=n;i+=lowbit(i))
            C[i-1]=C[i-1]+y;
        return;
    }
    T getsum(size_t x)const
    {
        x++;
        assert(1<=x&&x<=n);
        T res=C[x-1];
        for(size_t i=x-lowbit(x);i>0;i-=lowbit(i))
            res=res+C[i-1];
        return res;
    }
    int query(size_t l,size_t r)const
    {
        assert(l<=r);
        T res=getsum(r);
        if(l>0) res=res-getsum(l-1);
        return res;
    }
};
const int N=100005;
int n,m;
vector<int>G[N];
int dep[N];
int dfn[N],Index;
int mn[N][20],lg2[N];
void dfs_lca(int u,int father)
{
    dfn[u]=++Index;
    mn[Index][0]=father;
    dep[u]=dep[father]+1;
    for(int v:G[u])
    {
        if(v==father) continue;
        dfs_lca(v,u);
    }
    return;
}
void init_lca()
{
    Index=0;
    dfs_lca(1,0);
    lg2[0]=-1;
    for(int i=1;i<=Index;i++)
```

```
        lg2[i]=lg2[i/2]+1;
    for(int j=1;(1<<j)<=Index;j++)
        for(int i=1;i+(1<<j)-1<=Index;i++)
            if(dfn[mn[i][j-1]]<dfn[mn[i+(1<<(j-1))][j-1]]) mn[i][j]=mn[i][j-1];
            else mn[i][j]=mn[i+(1<<(j-1))][j-1];
    return;
}
int lca(int u,int v)
{
    if(u==v) return u;
    u=dfn[u],v=dfn[v];
    if(u>v) swap(u,v);
    u++;
    int d=lg2[v-u+1];
    if(dfn[mn[u][d]]<dfn[mn[v-(1<<d)+1][d]]) return mn[u][d];
    else return mn[v-(1<<d)+1][d];
}
int dis(int u,int v)
{
    return dep[u]+dep[v]-dep[lca(u,v)]*2;
}

int a[N];
int rt,tot;
int siz[N],mx[N];
bool book[N];
void getroot(int u,int father)
{
    siz[u]=1,mx[u]=0;
    for(int v:G[u])
    {
        if(book[v]) continue;
        if(v==father) continue;
        getroot(v,u);
        siz[u]+=siz[v];
        mx[u]=max(mx[u],siz[v]);
    }
    mx[u]=max(mx[u],tot-siz[u]);
    if(!rt||mx[u]<mx[rt]) rt=u;
    return;
}
int fa[N];
Fenwick_Tree<long long>f[N],g[N];
void add(int u,int father,int d)
{
    f[fa[rt]].add(d,a[u]);
    g[rt].add(d,a[u]);
    for(int v:G[u])
    {
        if(v==father) continue;
        if(book[v]) continue;
        add(v,u,d+1);
    }
    return;
}
void dfs(int u)
```

```
{
    book[u]=true;
    f[u].add(0,a[u]);
    for(int v:G[u])
    {
        if(book[v]) continue;
        tot=siz[v],rt=0;
        getroot(v,0);
        fa[rt]=u;
        f[rt].init(siz[v]+1),g[rt].init(siz[v]+1);
        add(v,u,1);
        dfs(rt);
    }
    return;
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    for(int i=1;i<n;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        G[x].emplace_back(y);
        G[y].emplace_back(x);
    }
    init_lca();
    rt=0,tot=n;
    getroot(1,0);
    f[rt].init(n),g[rt].init(n);
    fa[rt]=0;
    dfs(rt);
    long long lastans=0;
    while(m--)
    {
        int op;
        scanf("%d",&op);
        if(op==0)
        {
            long long x,k;
            scanf("%lld%lld",&x,&k);
            x^=lastans,k^=lastans;
            long long ans=f[x].query(0,min<int>(k,f[x].size()-1));
            for(int u=fa[x],v=x;u;u=fa[u],v=fa[v])
            {
                int d=dis(u,x);
                if(k>=d) ans+=f[u].query(0,min<int>(k-d,f[u].size()-1))-
g[v].query(0,min<int>(k-d,g[v].size()-1));
            }
            printf("%lld\n",ans);
            lastans=ans;
        }
        else if(op==1)
        {
            long long x,y;
```

```
            scanf("%lld%lld",&x,&y);
            x^=lastans,y^=lastans;
            for(int u=x;u;u=fa[u])
            {
                f[u].add(dis(x,u),y-a[x]);
                if(fa[u]) g[u].add(dis(x,fa[u]),y-a[x]);
            }
            a[x]=y;
        }
    }
    return 0;
}
```

网络流

最大流

```
struct Max_Flow
{
    static const int N=105,M=5005;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long flow;
    }edge[M*2];
    int cur[N],head[N],cnt;
    int tot;
    Max_Flow():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
    }
    void add_edge(int u,int v,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long f)
    {
        add_edge(u,v,f);
        add_edge(v,u,0);
        return;
    }
    int dep[N];
    bool bfs(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            dep[i]=-1;
```

```
        queue<int>q;
        q.push(s);
        dep[s]=0;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
};
```

最小费用最大流

```
struct Min_Cost_Max_Flow
{
    static const int N=405,M=15005;
    static const long long INF=4557430888798830399;
    struct Edge
```

```
{
    int to,next;
    int cost;
    long long flow;
}edge[M*2];
int cur[N],head[N],cnt;
int tot;
Min_Cost_Max_Flow():cnt(1),tot(0)
{
    memset(head,0,sizeof(head));
}
void add_edge(int u,int v,int c,long long f)
{
    cnt++;
    edge[cnt].to=v;
    edge[cnt].cost=c;
    edge[cnt].flow=f;
    edge[cnt].next=head[u];
    head[u]=cnt;
    return;
}
void add(int u,int v,int c,long long f)
{
    add_edge(u,v,c,f);
    add_edge(v,u,-c,0);
    return;
}
long long dis[N];
bool spfa(int s,int t)
{
    static bool vis[N];
    for(int i=1;i<=tot;i++)
        vis[i]=false;
    for(int i=1;i<=tot;i++)
        dis[i]=INF;
    queue<int>q;
    vis[s]=true;
    dis[s]=0;
    q.push(s);
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        vis[u]=false;
        for(int i=head[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(edge[i].flow<=0) continue;
            if(dis[v]>dis[u]+edge[i].cost)
            {
                dis[v]=dis[u]+edge[i].cost;
                if(!vis[v])
                {
                    vis[v]=true;
                    q.push(v);
                }
```

```
                }
            }
        }
        return dis[t]!=INF;
    }
    bool book[N];
    pair<long long,long long> dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return make_pair(flow,0);
        book[u]=true;
        long long used=0,res=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(book[v]||dis[v]!=dis[u]+edge[i].cost||edge[i].flow<=0) continue;
            pair<long long,long long>val=dfs(v,t,min(flow,edge[i].flow));
            long long now=val.first;
            res+=val.second+now*edge[i].cost;
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        book[u]=false;
        return make_pair(used,res);
    }
    pair<long long,long long> ssp(int s,int t)
    {
        long long ans=0,cost=0;
        for(int i=1;i<=tot;i++)
            book[i]=false;
        while(spfa(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            pair<long long,long long> res=dfs(s,t,INF);
            ans+=res.first,cost+=res.second;
        }
        return make_pair(ans,cost);
    }
};
```

最小费用可行流

```
struct Min_Cost_Feasible_Flow
{
    static const int N=405,M=15005;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        int cost;
        long long flow;
```

```
    }edge[M*2];
    int cur[N],head[N],cnt;
    int tot;
    Min_Cost_Feasible_Flow():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
    }
    void add_edge(int u,int v,int c,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].cost=c;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,int c,long long f)
    {
        add_edge(u,v,c,f);
        add_edge(v,u,-c,0);
        return;
    }
    long long dis[N];
    bool vis[N];
    bool spfa(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            vis[i]=false;
        for(int i=1;i<=tot;i++)
            dis[i]=INF;
        queue<int>q;
        vis[s]=true;
        dis[s]=0;
        q.push(s);
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            vis[u]=false;
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(edge[i].flow<=0) continue;
                if(dis[v]>dis[u]+edge[i].cost)
                {
                    dis[v]=dis[u]+edge[i].cost;
                    if(!vis[v])
                    {
                        vis[v]=true;
                        q.push(v);
                    }
                }
            }
        }
        return dis[t]!=INF;
```

```
    }
    bool book[N];
    pair<long long,long long> dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return make_pair(flow,0);
        book[u]=true;
        long long used=0,res=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(book[v]||dis[v]!=dis[u]+edge[i].cost||edge[i].flow<=0) continue;
            pair<long long,long long>val=dfs(v,t,min(flow,edge[i].flow));
            long long now=val.first;
            res+=val.second+now*edge[i].cost;
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        book[u]=false;
        return make_pair(used,res);
    }
    pair<long long,long long> ssp(int s,int t)
    {
        long long ans=0,cost=0;
        for(int i=1;i<=tot;i++)
            book[i]=false;
        while(spfa(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            pair<long long,long long> res=dfs(s,t,INF);
            if(cost<0) break;
            ans+=res.first,cost+=res.second;
        }
        return make_pair(ans,cost);
    }
};
```

无源汇上下界可行流

```
struct Bounded_Feasible_Flow_Without_Source_Sink
{
    static const int N=205,M=10205;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long flow;
    }edge[M*2+N*2];
    int cur[N],head[N],cnt;
    long long extra[N];
    vector<long long>flow;
```

```
int tot;
Bounded_Feasible_Flow_Without_Source_Sink():cnt(1),tot(0)
{
    memset(head,0,sizeof(head));
    memset(extra,0,sizeof(extra));
}
void add_edge(int u,int v,long long f)
{
    cnt++;
    edge[cnt].to=v;
    edge[cnt].flow=f;
    edge[cnt].next=head[u];
    head[u]=cnt;
    return;
}
void add(int u,int v,long long f)
{
    add_edge(u,v,f);
    add_edge(v,u,0);
    return;
}
void add(int u,int v,long long lower,long long upper)
{
    add(u,v,upper-lower);
    extra[v]+=lower,extra[u]-=lower;
    flow.emplace_back(lower);
    return;
}
int dep[N];
bool bfs(int s,int t)
{
    for(int i=1;i<=tot;i++)
        dep[i]=-1;
    queue<int>q;
    q.push(s);
    dep[s]=0;
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        for(int i=head[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=-1||edge[i].flow<=0) continue;
            dep[v]=dep[u]+1;
            q.push(v);
        }
    }
    return dep[t]!=-1;
}
long long dfs(int u,int t,long long flow)
{
    if(u==t||flow==0) return flow;
    long long used=0;
    for(int &i=cur[u];i;i=edge[i].next)
    {
```

```
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    vector<long long> solve()
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        if(dinic(s,t)!=sum) return {};
        for(int i=0;i<(int)flow.size();i++)
            flow[i]+=edge[i*2+3].flow;
        return flow;
    }
}bounded_feasible_flow_without_source_sink;
```

有源汇上下界可行流

```
struct Bounded_Feasible_Flow_With_Source_Sink
{
    static const int N=205,M=10205;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long flow;
    }edge[M*2+N*2];
```

```
    int cur[N],head[N],cnt;
    long long extra[N];
    vector<long long>flow;
    int tot;
    Bounded_Feasible_Flow_With_Source_Sink():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
        memset(extra,0,sizeof(extra));
    }
    void add_edge(int u,int v,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long f)
    {
        add_edge(u,v,f);
        add_edge(v,u,0);
        return;
    }
    void add(int u,int v,long long lower,long long upper)
    {
        add(u,v,upper-lower);
        extra[v]+=lower,extra[u]-=lower;
        flow.emplace_back(lower);
        return;
    }
    int dep[N];
    bool bfs(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            dep[i]=-1;
        queue<int>q;
        q.push(s);
        dep[s]=0;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return flow;
```

```
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    vector<long long> solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        add(T,S,INF);
        if(dinic(s,t)!=sum) return {};
        for(int i=0;i<(int)flow.size();i++)
            flow[i]+=edge[i*2+3].flow;
        return flow;
    }
};
```

有源汇上下界最大流

```
struct Bounded_Max_Flow_With_Source_Sink
{
    static const int N=205,M=10005;
    static const long long INF=4557430888798830399;
    struct Edge
```

```
{
    int to,next;
    long long flow;
}edge[M*2+N*2];
int cur[N],head[N],cnt;
long long extra[N];
int tot;
Bounded_Max_Flow_With_Source_Sink():cnt(1),tot(0)
{
    memset(head,0,sizeof(head));
    memset(extra,0,sizeof(extra));
}
void add_edge(int u,int v,long long f)
{
    cnt++;
    edge[cnt].to=v;
    edge[cnt].flow=f;
    edge[cnt].next=head[u];
    head[u]=cnt;
    return;
}
void add(int u,int v,long long f)
{
    add_edge(u,v,f);
    add_edge(v,u,0);
    return;
}
void add(int u,int v,long long lower,long long upper)
{
    add(u,v,upper-lower);
    extra[v]+=lower,extra[u]-=lower;
    return;
}
int dep[N];
bool bfs(int s,int t)
{
    for(int i=1;i<=tot;i++)
        dep[i]=-1;
    queue<int>q;
    q.push(s);
    dep[s]=0;
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        for(int i=head[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=-1||edge[i].flow<=0) continue;
            dep[v]=dep[u]+1;
            q.push(v);
        }
    }
    return dep[t]!=-1;
}
long long dfs(int u,int t,long long flow)
```

```
    {
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    long long solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        add(T,S,INF);
        if(dinic(s,t)!=sum) return -1;
        return dinic(S,T);
    }
};
```

有源汇上下界最小流

```
struct Bounded_Min_Flow_With_Source_Sink
{
    static const int N=50010,M=125010;
    static const long long INF=4557430888798830399;
    struct Edge
    {
```

```
        int to,next;
        long long flow;
    }edge[M*2+N*2];
    int cur[N],head[N],cnt;
    long long extra[N];
    int tot;
    Bounded_Min_Flow_With_Source_Sink():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
        memset(extra,0,sizeof(extra));
    }
    void add_edge(int u,int v,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long f)
    {
        add_edge(u,v,f);
        add_edge(v,u,0);
        return;
    }
    void add(int u,int v,long long lower,long long upper)
    {
        add(u,v,upper-lower);
        extra[v]+=lower,extra[u]-=lower;
        return;
    }
    int dep[N];
    bool bfs(int s,int t)
    {
        for(int i=1;i<=tot;i++)
            dep[i]=-1;
        queue<int>q;
        q.push(s);
        dep[s]=0;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=edge[i].next)
            {
                int v=edge[i].to;
                if(dep[v]!=-1||edge[i].flow<=0) continue;
                dep[v]=dep[u]+1;
                q.push(v);
            }
        }
        return dep[t]!=-1;
    }
    long long dfs(int u,int t,long long flow)
    {
```

```
        if(u==t||flow==0) return flow;
        long long used=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(dep[v]!=dep[u]+1||edge[i].flow<=0) continue;
            long long now=dfs(v,t,min(flow,(long long)edge[i].flow));
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        return used;
    }
    long long dinic(int s,int t)
    {
        long long res=0;
        while(bfs(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            res+=dfs(s,t,INF);
        }
        return res;
    }
    long long solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,-extra[i]);
            }
        add(T,S,INF);
        if(dinic(s,t)!=sum) return -1;
        long long res=edge[cnt].flow;
        edge[cnt].flow=edge[cnt^1].flow=0;
        return res-dinic(T,S);
    }
};
```

有源汇上下界最小费用可行流

```
struct Bounded_Min_Cost_Feasible_Flow_With_Source_Sink
{
    static const int N=305,M=5305;
    static const long long INF=4557430888798830399;
```

```
struct Edge
{
    int to,next;
    int cost;
    long long flow;
}edge[M*2+N*2];
int cur[N],head[N],cnt;
long long extra[N];
int tot;
long long totalcost;
Bounded_Min_Cost_Feasible_Flow_With_Source_Sink():cnt(1),tot(0),totalcost(0)
{
    memset(head,0,sizeof(head));
    memset(extra,0,sizeof(extra));
}
void add_edge(int u,int v,int c,long long f)
{
    cnt++;
    edge[cnt].to=v;
    edge[cnt].cost=c;
    edge[cnt].flow=f;
    edge[cnt].next=head[u];
    head[u]=cnt;
    return;
}
void add(int u,int v,int c,long long f)
{
    add_edge(u,v,c,f);
    add_edge(v,u,-c,0);
    return;
}
void add(int u,int v,int c,long long lower,long long upper)
{
    totalcost+=lower*c;
    add(u,v,c,upper-lower);
    extra[v]+=lower,extra[u]-=lower;
    return;
}
long long dis[N];
bool vis[N];
bool spfa(int s,int t)
{
    for(int i=1;i<=tot;i++)
        vis[i]=false;
    for(int i=1;i<=tot;i++)
        dis[i]=INF;
    queue<int>q;
    vis[s]=true;
    dis[s]=0;
    q.push(s);
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        vis[u]=false;
        for(int i=head[u];i;i=edge[i].next)
```

```
            {
                int v=edge[i].to;
                if(edge[i].flow<=0) continue;
                if(dis[v]>dis[u]+edge[i].cost)
                {
                    dis[v]=dis[u]+edge[i].cost;
                    if(!vis[v])
                    {
                        vis[v]=true;
                        q.push(v);
                    }
                }
            }
        }
        return dis[t]!=INF;
    }
    bool book[N];
    pair<long long,long long> dfs(int u,int t,long long flow)
    {
        if(u==t||flow==0) return make_pair(flow,0);
        book[u]=true;
        long long used=0,res=0;
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(book[v]||dis[v]!=dis[u]+edge[i].cost||edge[i].flow<=0) continue;
            pair<long long,long long>val=dfs(v,t,min(flow,edge[i].flow));
            long long now=val.first;
            res+=val.second+now*edge[i].cost;
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        book[u]=false;
        return make_pair(used,res);
    }
    pair<long long,long long> ssp(int s,int t)
    {
        long long ans=0,cost=0;
        for(int i=1;i<=tot;i++)
            book[i]=false;
        while(spfa(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            pair<long long,long long> res=dfs(s,t,INF);
            ans+=res.first,cost+=res.second;
        }
        return make_pair(ans,cost);
    }
    pair<long long,long long> solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
```

```
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,0,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,0,-extra[i]);
            }
        add(T,S,0,INF);
        pair<long long,long long>res=ssp(s,t);
        if(res.first!=sum) return make_pair(-1,-1);
        totalcost+=res.second;
        return make_pair(res.first,totalcost);
    }
};
```

有源汇上下界最小费用最大流

```
struct Bounded_Min_Cost_Max_Flow_With_Source_Sink
{
    static const int N=1005,M=5005;
    static const long long INF=4557430888798830399;
    struct Edge
    {
        int to,next;
        long long cost,flow;
    }edge[M*2+N*2];
    int cur[N],head[N],cnt;
    long long extra[N];
    int tot;
    long long totalcost;
    Bounded_Min_Cost_Max_Flow_With_Source_Sink():cnt(1),tot(0)
    {
        memset(head,0,sizeof(head));
        memset(extra,0,sizeof(extra));
    }
    void add_edge(int u,int v,long long c,long long f)
    {
        cnt++;
        edge[cnt].to=v;
        edge[cnt].flow=f;
        edge[cnt].cost=c;
        edge[cnt].next=head[u];
        head[u]=cnt;
        return;
    }
    void add(int u,int v,long long c,long long f)
    {
        add_edge(u,v,c,f);
        add_edge(v,u,-c,0);
        return;
    }
```

```
void add(int u,int v,long long c,long long lower,long long upper)
{
    if(c>=0)
    {
        totalcost+=lower*c;
        add(u,v,c,upper-lower);
        extra[v]+=lower,extra[u]-=lower;
    }
    else
    {
        totalcost+=upper*c;
        add(v,u,-c,upper-lower);
        extra[v]+=upper,extra[u]-=upper;
    }
    return;
}
long long dis[N];
bool spfa(int s,int t)
{
    static bool vis[N];
    for(int i=1;i<=tot;i++)
        vis[i]=false;
    for(int i=1;i<=tot;i++)
        dis[i]=INF;
    queue<int>q;
    vis[s]=true;
    dis[s]=0;
    q.push(s);
    while(!q.empty())
    {
        int u=q.front();
        q.pop();
        vis[u]=false;
        for(int i=head[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(edge[i].flow<=0) continue;
            if(dis[v]>dis[u]+edge[i].cost)
            {
                dis[v]=dis[u]+edge[i].cost;
                if(!vis[v])
                {
                    vis[v]=true;
                    q.push(v);
                }
            }
        }
    }
    return dis[t]!=INF;
}
bool book[N];
pair<long long,long long> dfs(int u,int t,long long flow)
{
    if(u==t||flow==0) return make_pair(flow,0);
    book[u]=true;
    long long used=0,res=0;
```

```
        for(int &i=cur[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(book[v]||dis[v]!=dis[u]+edge[i].cost||edge[i].flow<=0) continue;
            pair<long long,long long> val=dfs(v,t,min(flow,edge[i].flow));
            long long now=val.first;
            res+=val.second+now*edge[i].cost;
            flow-=now;
            edge[i].flow-=now;
            edge[i^1].flow+=now;
            used+=now;
            if(flow==0) break;
        }
        book[u]=false;
        return make_pair(used,res);
    }
    pair<long long,long long> ssp(int s,int t)
    {
        long long ans=0,cost=0;
        for(int i=1;i<=tot;i++)
            book[i]=false;
        while(spfa(s,t))
        {
            for(int i=1;i<=tot;i++)
                cur[i]=head[i];
            pair<long long,long long> res=dfs(s,t,INF);
            ans+=res.first,cost+=res.second;
        }
        return make_pair(ans,cost);
    }
    pair<long long,long long> solve(int S,int T)
    {
        int s=++tot,t=++tot;
        long long sum=0;
        for(int i=1;i<=tot-2;i++)
            if(extra[i]>0)
            {
                sum+=extra[i];
                add(s,i,0,extra[i]);
            }
            else if(extra[i]<0)
            {
                add(i,t,0,-extra[i]);
            }
        add(T,S,0,INF);
        pair<long long,long long>res=ssp(s,t);
        if(res.first!=sum) return {-1,-1};
        totalcost+=res.second;
        res=ssp(S,T);
        totalcost+=res.second;
        return make_pair(res.first,totalcost);
    }
};
```

图的匹配

二分图最大匹配

Hopcroft-Karp $O(E\sqrt{V})$

```
struct Hopcroft_Karp
{
    int n,m;
    vector<int>g[N];
    int pa[N],pb[N];
    Hopcroft_Karp(){}
    Hopcroft_Karp(int _n,int _m):n(_n),m(_m){}
    void init(int _n,int _m)
    {
        n=_n,m=_m;
        return;
    }
    void add_edge(int u,int v)
    {
        g[u].push_back(v);
        return;
    }
    int dis[N];
    bool bfs()
    {
        queue<int>q;
        for(int u=1;u<=n;u++)
        {
            if(!pa[u])
            {
                dis[u]=0;
                q.push(u);
            }
            else dis[u]=-1;
        }
        bool found=false;
        while(!q.empty())
        {
            int u=q.front();
            q.pop();
            for(int v:g[u])
            {
                if(pb[v]&&dis[pb[v]]==-1)
                {
                    dis[pb[v]]=dis[u]+1;
                    q.push(pb[v]);
                }
                else if(!pb[v]) found=true;
            }
        }
        return found;
```

```
    }
    bool dfs(int u)
    {
        for(int v:g[u])
            if(!pb[v]||(dis[pb[v]]==dis[u]+1&&dfs(pb[v])))
            {
                pa[u]=v,pb[v]=u;
                return true;
            }
        dis[u]=-1;
        return false;
    }
    int max_matching()
    {
        fill(pa+1,pa+n+1,0);
        fill(pb+1,pb+m+1,0);
        int res=0;
        while(bfs())
        {
            for(int u=1;u<=n;u++)
                if(!pa[u]&&dfs(u)) res++;
        }
        return res;
    }
};
```

二分图最大权匹配

Hungarian Algorithm（Kuhn–Munkres Algorithm） O(n^3)

```
#include<iostream>
#include<cstdio>
#include<queue>
#include<algorithm>
using namespace std;
struct Kuhn_Munkres
{
    static constexpr int N=405;
    static constexpr int INF=1061109567;
    int n;
    int a[N][N];
    int hl[N],hr[N],slk[N];
    int fl[N],fr[N];
    bool vl[N],vr[N];
    int pre[N];
    queue<int>q;
    bool check(int i)
    {
        vl[i]=1;
        if(fl[i]!=0)
        {
            q.push(fl[i]);
            vr[fl[i]]=true;
```

```
            return true;
        }
        while(i!=0)
            fl[i]=pre[i],swap(i,fr[fl[i]]);
        return false;
    }
    void bfs(int s)
    {
        for(int i=1;i<=n;i++)
            slk[i]=INF;
        fill(vl+1,vl+n+1,false);
        fill(vr+1,vr+n+1,false);
        while(!q.empty())
            q.pop();
        q.push(s);
        vr[s]=true;
        while(true)
        {
            while(!q.empty())
            {
                int j=q.front();
                q.pop();
                for(int i=1;i<=n;i++)
                {
                    int d=hl[i]+hr[j]-a[i][j];
                    if(!vl[i]&&slk[i]>=d)
                    {
                        pre[i]=j;
                        if(d) slk[i]=d;
                        else if(!check(i)) return;
                    }
                }
            }
            int d=INF;
            for(int i=1;i<=n;i++)
                if(!vl[i]&&d>slk[i]) d=slk[i];
            for(int i=1;i<=n;i++)
            {
                if(vl[i]) hl[i]+=d;
                else slk[i]-=d;
                if(vr[i]) hr[i]-=d;
            }
            for(int i=1;i<=n;i++)
                if(!vl[i]&&!slk[i]&&!check(i)) return;
        }
    }
    void solve()
    {
        fill(fl+1,fl+n+1,0);
        fill(fr+1,fr+n+1,0);
        fill(hr+1,hr+n+1,0);
        for(int i=1;i<=n;i++)
            hl[i]=*max_element(a[i]+1,a[i]+n+1);
        for(int j=1;j<=n;j++)
            bfs(j);
        return;
```

```
    }
};
```

环计数问题

三元环计数

```
#include<iostream>
#include<cstdio>
#include<vector>
using namespace std;
constexpr int N=100005,M=100005;
constexpr int MOD=998244353;
int n,m;
int x[N];
int u[M],v[M];
int deg[N];
vector<int>G[N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>x[i];
    fill(deg+1,deg+n+1,0);
    for(int i=1;i<=m;i++)
    {
        cin>>u[i]>>v[i];
        u[i]++,v[i]++;
        deg[u[i]]++,deg[v[i]]++;
    }
    for(int i=1;i<=m;i++)
    {
        if(deg[u[i]]>deg[v[i]]||(deg[u[i]]==deg[v[i]]&&u[i]>=v[i]))
swap(u[i],v[i]);
        G[u[i]].emplace_back(v[i]);
    }
    int ans=0;
    for(int u=1;u<=n;u++)
    {
        static bool vis[N];
        for(int v:G[u])
            vis[v]=true;
        for(int v:G[u])
            for(int w:G[v])
                if(vis[w]) ans=(ans+(long long)x[u]*x[v]%MOD*x[w])%MOD;
        for(int v:G[u])
            vis[v]=false;
    }
    cout<<ans;
    return 0;
```

```
}
```

四元环计数

```
#include<iostream>
#include<cstdio>
#include<vector>
using namespace std;
constexpr int N=100005,M=200005;
int n,m;
int u[M],v[M];
int deg[N];
vector<int>E[N],G[N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    cin>>n>>m;
    fill(deg+1,deg+n+1,0);
    for(int i=1;i<=m;i++)
    {
        cin>>u[i]>>v[i];
        E[u[i]].emplace_back(v[i]);
        E[v[i]].emplace_back(u[i]);
        deg[u[i]]++,deg[v[i]]++;
    }
    for(int i=1;i<=m;i++)
    {
        if(deg[u[i]]>deg[v[i]]||(deg[u[i]]==deg[v[i]]&&u[i]>=v[i]))
swap(u[i],v[i]);
        G[u[i]].emplace_back(v[i]);
    }
    long long ans=0;
    for(int a=1;a<=n;a++)
    {
        static int cnt[N];
        for(int b:E[a])
            for(int c:G[b])
            {
                if(deg[a]>deg[c]||(deg[a]==deg[c]&&a>=c)) continue;
                ans+=cnt[c];
                cnt[c]++;
            }
        for(int b:E[a])
            for(int c:G[b])
                cnt[c]=0;
    }
    cout<<ans;
    return 0;
}
```

计算几何

二维计算几何

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cassert>
#include<chrono>
#include<random>
#include<vector>
#include<functional>
#include<iomanip>
#include<algorithm>
using namespace std;
namespace Geometry
{
    const double eps=1e-12;
    const double PI=acosl(-1);
    const double INF=1e18;
    bool equal(double a,double b)
    {
        return abs(a-b)<eps;
    }
    bool less(double a,double b)
    {
        return b-a>=eps;
    }
    bool greater(double a,double b)
    {
        return a-b>=eps;
    }
    bool less_equal(double a,double b)
    {
        return b-a>-eps;
    }
    bool greater_equal(double a,double b)
    {
        return a-b>-eps;
    }
    class Point
    {
    public:
        double x,y;
        Point(){x=0,y=0;}
        Point(const double &_x,const double &_y):x(_x),y(_y) {}
        friend Point operator * (const Point &a,const double &b)
        {
            return Point(a.x*b,a.y*b);
        }
        friend Point operator * (const double &a,const Point &b)
        {
            return Point(a*b.x,a*b.y);
```

```
    }
    friend Point operator / (const Point &a,const double &b)
    {
        return Point(a.x/b,a.y/b);
    }
    friend Point operator + (const Point &a,const Point &b)
    {
        return Point(a.x+b.x,a.y+b.y);
    }
    Point operator += (const Point &b)
    {
        x+=b.x,y+=b.y;
        return *this;
    }
    friend Point operator - (const Point &a,const Point &b)
    {
        return Point(a.x-b.x,a.y-b.y);
    }
    Point operator -= (const Point &b)
    {
        x-=b.x,y-=b.y;
        return *this;
    }
    friend double cross(const Point &a,const Point &b)
    {
        return a.x*b.y-a.y*b.x;
    }
    friend double dot(const Point &a,const Point &b)
    {
        return a.x*b.x+a.y*b.y;
    }
    friend bool operator == (const Point &a,const Point &b)
    {
        return equal(a.x,b.x)&&equal(a.y,b.y);
    }
    friend bool operator != (const Point &a,const Point &b)
    {
        return (!equal(a.x,b.x))||(!equal(a.y,b.y));
    }
    friend bool operator < (const Point &a,const Point &b)
    {
        if(equal(a.x,b.x)) return less(a.y,b.y);
        else return less(a.x,b.x);
    }
    friend bool operator > (const Point &a,const Point &b)
    {
        if(equal(a.x,b.x)) return greater(a.y,b.y);
        else return greater(a.x,b.x);
    }
    friend bool operator <= (const Point &a,const Point &b)
    {
        if(equal(a.x,b.x)) return less_equal(a.y,b.y);
        else return less_equal(a.x,b.x);
    }
    friend bool operator >= (const Point &a,const Point &b)
    {
```

```
            if(equal(a.x,b.x)) return greater_equal(a.y,b.y);
            else return greater_equal(a.x,b.x);
        }
        Point operator - ()const
        {
            return Point(-x,-y);
        }
        double length()const
        {
            return sqrtl(x*x+y*y);
        }
        Point unit()const
        {
            return *this/length();
        }
        double angle()const
        {
            return atan2l(y,x);
        }
        int quadrant()const
        {
            if(x>0&&y>=0) return 1;
            else if(x<=0&&y>0) return 2;
            else if(x<0&&y<=0) return 3;
            else if(x>=0&&y<0) return 4;
            else return 0;
        }
        friend double angle(const Point &a,const Point &b)
        {
            return atan2l(cross(a,b),dot(a,b));
        }
        Point rotate(const double &theta)const
        {
            return Point(x*cosl(theta)-
y*sinl(theta),x*sinl(theta)+y*cosl(theta));
        }
        friend istream &operator>>(istream &in,Point &obj)
        {
            in>>obj.x>>obj.y;
            return in;
        }
        friend ostream &operator<<(ostream &out,const Point &obj)
        {
            out<<obj.x<<" "<<obj.y;
            return out;
        }
    };
    bool cmp_polar_angle(const Point &a,const Point &b)
    {
        int x=a.quadrant(),y=b.quadrant();
        if(x!=y) return x<y;
        return cross(a,b)>0;
    };
    double distance(const Point &a,const Point &b)
    {
        return sqrtl((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
```

```
}
enum Direction
{
    COUNTER_CLOCKWISE,
    CLOCKWISE,
    ONLINE_BACK,
    ONLINE_FRONT,
    ON_SEGMENT
};
istream& operator>>(istream& in,Direction& direction)
{
    string value;
    in>>value;
    if(value=="COUNTER_CLOCKWISE") direction=COUNTER_CLOCKWISE;
    else if(value=="CLOCKWISE") direction=CLOCKWISE;
    else if(value=="ONLINE_BACK") direction=ONLINE_BACK;
    else if(value=="ONLINE_FRONT") direction=ONLINE_FRONT;
    else if(value=="ON_SEGMENT") direction=ON_SEGMENT;
    else in.setstate(ios::failbit);
    return in;
}
ostream& operator<<(ostream& out,const Direction& direction)
{
    if(direction==COUNTER_CLOCKWISE) out<<"COUNTER_CLOCKWISE";
    else if(direction==CLOCKWISE) out<<"CLOCKWISE";
    else if(direction==ONLINE_BACK) out<<"ONLINE_BACK";
    else if(direction==ONLINE_FRONT) out<<"ONLINE_FRONT";
    else if(direction==ON_SEGMENT) out<<"ON_SEGMENT";
    return out;
}
class Line
{
public:
    Point a,b;
    Line(){}
    Line(const Point &_a,const Point &_b):a(_a),b(_b){}
    Point projection(const Point &p)const
    {
        return a+(b-a).unit()*(dot(p-a,b-a)/(b-a).length());
    }
    Point reflection(const Point &p)const
    {
        return projection(p)*2-p;
    }
    Direction direction(const Point &p)const
    {
        double t=cross(b-a,p-a);
        if(greater(t,0)) return COUNTER_CLOCKWISE;
        if(less(t,0)) return CLOCKWISE;
        double l1=dot(p-a,b-a);
        if(less(l1,0)) return ONLINE_BACK;
        double l2=dot(b-a,b-a);
        if(greater(l1,l2)) return ONLINE_FRONT;
        else return ON_SEGMENT;
    }
    double distance(const Point &p)const
```

```
        {
            Point u=projection(p);
            if(direction(u)==ON_SEGMENT) return Geometry::distance(u,p);
            else return min(Geometry::distance(a,p),Geometry::distance(b,p));
        }
        Point middle_point()const
        {
            return (a+b)/2;
        }
        Line perpendicular_bisector()const
        {
            Point p=middle_point();
            return Line(p,p+(b-a).rotate(PI/2));
        }
        double length()const
        {
            return Geometry::distance(a,b);
        }
        friend istream &operator>>(istream &in,Line &obj)
        {
            in>>obj.a>>obj.b;
            return in;
        }
        friend ostream &operator<<(ostream &out,const Line &obj)
        {
            out<<obj.a<<" "<<obj.b;
            return out;
        }
    };
    bool parallel(const Line &x,const Line &y)
    {
        return equal(cross(x.b-x.a,y.b-y.a),0);
    }
    bool orthogonal(const Line &x,const Line &y)
    {
        return equal(dot(x.b-x.a,y.b-y.a),0);
    }
    vector<Point> cross_point(const Line &x,const Line &y)
    {
        if(parallel(x,y)) return {};
        Point u=x.a-y.a,v=x.b-x.a,w=y.b-y.a;
        double t=cross(w,u)/cross(v,w);
        return {x.a+t*v};
    }
    int sgn(double x)
    {
        return greater(x,0)-less(x,0);
    }
    bool intersection(const Line &x,const Line &y)
    {

 if(x.direction(y.a)==ON_SEGMENT||x.direction(y.b)==ON_SEGMENT||y.direction(x.a)==
=ON_SEGMENT||y.direction(x.b)==ON_SEGMENT) return true;
        return sgn(cross(x.b-x.a,y.a-x.a))*sgn(cross(x.b-x.a,y.b-x.a))
<0&&sgn(cross(y.b-y.a,x.a-y.a))*sgn(cross(y.b-y.a,x.b-y.a))<0;
    }
```

```
    double distance(const Line &x,const Line &y)
    {
        if(intersection(x,y)) return 0;
        else return
min({x.distance(y.a),x.distance(y.b),y.distance(x.a),y.distance(x.b)});
    }
    const int IN=2,ON=1,OUT=0;
    mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
    class Polygon
    {
    private:
        vector<Point>g;
    public:
        Polygon(){}
        Polygon(const int &n){g.resize(n);}
        Polygon(const vector<Point> &f):g(f){}
        void clear()
        {
            g.clear();
        }
        void resize(int n)
        {
            g.resize(n);
        }
        void push_back(const Point &x)
        {
            return g.push_back(x);
        }
        void push_back(const vector<Point> &x)
        {
            for(const Point &p:x)
                g.push_back(p);
            return;
        }
        void pop_back()
        {
            return g.pop_back();
        }
        Point& front()
        {
            return g.front();
        }
        const Point& front()const
        {
            return g.front();
        }
        Point& back()
        {
            return g.back();
        }
        const Point& back()const
        {
            return g.back();
        }
        size_t size()const
        {
```

```
        return g.size();
    }
    Point& operator [](const int &i)
    {
        return g[i];
    }
    const Point& operator [](const int &i)const
    {
        return g[i];
    }
    vector<Point>::iterator begin()
    {
        return g.begin();
    }
    vector<Point>::iterator end()
    {
        return g.end();
    }
    vector<Point>::const_iterator begin()const
    {
        return g.begin();
    }
    vector<Point>::const_iterator end()const
    {
        return g.end();
    }
    vector<Point>::reverse_iterator rbegin()
    {
        return g.rbegin();
    }
    vector<Point>::reverse_iterator rend()
    {
        return g.rend();
    }
    vector<Point>::const_reverse_iterator rbegin()const
    {
        return g.rbegin();
    }
    vector<Point>::const_reverse_iterator rend()const
    {
        return g.rend();
    }
    double area()const
    {
        int n=g.size();
        double res=0;
        for(int i=0;i<n;i++)
            res+=cross(g[i],g[(i+1)%n]);
        res/=2;
        return abs(res);
    }
    double perimeter()const
    {
        int n=g.size();
        double sum=0;
        for(int i=0;i<n;i++)
```

```
            sum+=distance(g[i],g[(i+1)%n]);
        return sum;
    }
    bool is_convex()const
    {
        int n=g.size();
        for(int i=0;i<n;i++)
            if(less(cross(g[(i+1)%n]-g[i],g[(i-1+n)%n]-g[i]),0)) return
false;
        return true;
    }
    int point_containment(const Point &a)const
    {
        int n=g.size();
        for(int i=0;i<n;i++)
            if(Line(g[i],g[(i+1)%n]).direction(a)==ON_SEGMENT) return ON;
        function<bool(const Line &)> check=[&](const Line &l)
        {
            for(int i=0;i<n;i++)
                if(parallel(l,Line(g[i],g[(i+1)%n]))) return false;
            for(int i=0;i<n;i++)

 if(l.direction(g[i])==ON_SEGMENT||l.direction(g[i])==ONLINE_FRONT||l.direction(g
[i])==ONLINE_BACK) return false;
            return true;
        };
        Line l=Line(a,Point(rnd(),rnd()));
        while(!check(l))
            l=Line(a,Point(rnd(),rnd()));
        int s=0;
        for(int i=0;i<n;i++)
            if(intersection(l,Line(g[i],g[(i+1)%n]))) s++;
        if(s&1) return IN;
        else return OUT;
    }
    double convex_diamater()const
    {
        int n=g.size();
        assert(n>=2);
        if(n==2) return distance(g[0],g[1]);
        double ans=0;
        for(int i=0,j=2;i<n;i++)
        {
            while(less(cross(g[i]-g[j],g[(i+1)%n]-g[j]),cross(g[i]-
g[(j+1)%n],g[(i+1)%n]-g[(j+1)%n]))) j=(j+1)%n;
            ans=max(ans,max(distance(g[j],g[i]),distance(g[j],g[(i+1)%n])));
        }
        return ans;
    }
    pair<Polygon,Polygon> convex_cut(const Line &l)const
    {
        Polygon res1,res2;
        int n=g.size();
        for(int i=0;i<(int)g.size();i++)
        {
            Point u=g[i],v=g[(i+1)%n];
```

```
                if(greater_equal(cross(l.b-l.a,u-l.a),0))
                {
                    res1.push_back(u);
                    if(less(cross(l.b-l.a,v-l.a),0))
res1.push_back(cross_point(Line(u,v),l));
                }
                else if(greater(cross(l.b-l.a,v-l.a),0))
res1.push_back(cross_point(Line(u,v),l));
            }
            for(int i=0;i<(int)g.size();i++)
            {
                Point u=g[i],v=g[(i+1)%n];
                if(greater_equal(cross(l.a-l.b,u-l.b),0))
                {
                    res2.push_back(u);
                    if(less(cross(l.a-l.b,v-l.b),0))
res2.push_back(cross_point(Line(u,v),l));
                }
                else if(greater(cross(l.a-l.b,v-l.b),0))
res2.push_back(cross_point(Line(u,v),l));
            }
            return make_pair(res1,res2);
        }
        Polygon kernel()const;
    };
    Polygon convex_hull(const vector<Point> &p)
    {
        int n=p.size();
        if(n<=2)
        {
            Polygon res;
            for(int i=0;i<n;i++)
                res.push_back(p[i]);
            return res;
        }
        vector<int>id(n);
        iota(id.begin(),id.end(),0);
        sort(id.begin(),id.end(),[&](const int &a,const int &b){return
p[a].x==p[b].x?p[a].y<p[b].y:p[a].x<p[b].x;});
        vector<int>stk;
        int top=0;
        for(int i=0;i<n;i++)
        {
            while(top>=2&&less_equal(cross(p[stk[top-1]]-p[stk[top-2]],p[id[i]]-
p[stk[top-1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(id[i]),top++;
        }
        int tmp=top;
        for(int i=n-2;i>=0;i--)
        {
            while(top>tmp&&less_equal(cross(p[stk[top-1]]-p[stk[top-2]],p[id[i]]-
p[stk[top-1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(id[i]),top++;
        }
        stk.pop_back(),top--;
        vector<int> hull;
```

```
        for(int i=0;i<top;i++)
            hull.emplace_back(stk[i]);
        Polygon res;
        for(int u:hull)
            res.push_back(p[u]);
        return res;
    }
    Polygon non_strictly_convex_hull(const vector<Point> &p)
    {
        int n=p.size();
        if(n<=2)
        {
            Polygon res;
            for(int i=0;i<n;i++)
                res.push_back(p[i]);
            return res;
        }
        vector<int>id(n);
        iota(id.begin(),id.end(),0);
        sort(id.begin(),id.end(),[&](const int &a,const int &b){return
p[a].x==p[b].x?p[a].y<p[b].y:p[a].x<p[b].x;});
        vector<int>stk;
        int top=0;
        for(int i=0;i<n;i++)
        {
            while(top>=2&&less(cross(p[stk[top-1]]-p[stk[top-2]],p[id[i]]-
p[stk[top-1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(id[i]),top++;
        }
        int tmp=top;
        for(int i=n-2;i>=0;i--)
        {
            while(top>tmp&&less(cross(p[stk[top-1]]-p[stk[top-2]],p[id[i]]-
p[stk[top-1]]),0)) stk.pop_back(),top--;
            stk.emplace_back(id[i]),top++;
        }
        stk.pop_back(),top--;
        vector<int> hull;
        for(int i=0;i<top;i++)
            hull.emplace_back(stk[i]);
        Polygon res;
        for(int u:hull)
            res.push_back(p[u]);
        return res;
    }
    Polygon minkowski_sum(const vector<Point> &a,const vector<Point> &b)
    {
        assert(a.size()!=0&&b.size()!=0);
        Polygon ca=convex_hull(a),cb=convex_hull(b);
        int na=ca.size(),nb=cb.size();
        vector<Point>la,lb;
        for(int i=0;i<na;i++)
            la.emplace_back(ca[(i+1)%na]-ca[i]);
        for(int i=0;i<nb;i++)
            lb.emplace_back(cb[(i+1)%nb]-cb[i]);
        int pa=0,pb=0;
```

```
        vector<Point> l;
        l.emplace_back(ca[0]+cb[0]);
        while(pa<(int)la.size()&&pb<(int)lb.size())
        {
            double val=cross(la[pa],lb[pb]);
            if(greater(val,0)) l.emplace_back(l.back()+la[pa]),pa++;
            else if(less(val,0)) l.emplace_back(l.back()+lb[pb]),pb++;
            else l.emplace_back(l.back()+la[pa]+lb[pb]),pa++,pb++;
        }
        while(pa<(int)la.size())
            l.emplace_back(l.back()+la[pa]),pa++;
        while(pb<(int)lb.size())
            l.emplace_back(l.back()+lb[pb]),pb++;
        Polygon res=convex_hull(l);
        return res;
    }
    Polygon half_plane_intersection(const vector<Line> &l,double x1=-INF,double
y1=-INF,double x2=INF,double y2=INF)
    {
        vector<pair<double,Line>>f;
        for(int i=0;i<(int)l.size();i++)
            f.emplace_back((l[i].b-l[i].a).angle(),l[i]);
        f.emplace_back(0,Line(Point(x1,y1),Point(x2,y1)));
        f.emplace_back(PI/2,Line(Point(x2,y1),Point(x2,y2)));
        f.emplace_back(PI,Line(Point(x2,y2),Point(x1,y2)));
        f.emplace_back(-PI/2,Line(Point(x1,y2),Point(x1,y1)));
        int n=f.size();
        sort(f.begin(),f.end(),[](const pair<double,Line> &a,const
pair<double,Line> &b){return !equal(a.first,b.first)?
a.first<b.first:a.second.direction(b.second.a)==CLOCKWISE;});
        vector<Line>Ql(n);
        vector<Point>Qp(n);
        Polygon res;
        int head=0,tail=-1;
        Ql[++tail]=f[0].second;
        for(int i=1;i<n;i++)
            if(!equal(f[i].first,f[i-1].first))
            {
                while(head<tail&&f[i].second.direction(Qp[tail-1])==CLOCKWISE)
tail--;
                while(head<tail&&f[i].second.direction(Qp[head])==CLOCKWISE)
head++;
                Ql[++tail]=f[i].second;
                if(head<tail)
                {
                    vector<Point> tmp=cross_point(Ql[tail],Ql[tail-1]);
                    if(!tmp.empty()) Qp[tail-1]=tmp[0];
                    else return res;
                }
            }
        while(head<tail&&Ql[head].direction(Qp[tail-1])==CLOCKWISE) tail--;
        while(head<tail&&Ql[tail].direction(Qp[head])==CLOCKWISE) head++;
        vector<Point> tmp=cross_point(Ql[tail],Ql[head]);
        if(tmp.empty()||tail-head+1<=2) return res;
        for(int i=head;i<tail;i++)
            res.push_back(Qp[i]);
```

```
        res.push_back(tmp[0]);
        return res;
    }
    Polygon Polygon::kernel()const
    {
        int n=g.size();
        vector<Line>l;
        for(int i=0;i<n;i++)
            l.emplace_back(Line(g[i],g[(i+1)%n]));
        return half_plane_intersection(l);
    }
    double closest_pair(const vector<Point> &_p)
    {
        vector<Point>p=_p;
        sort(p.begin(),p.end(),[](const Point &a,const Point &b){return
a.x<b.x;});
        function<double(const int &,const int &)> solve=[&](const int &l,const
int &r)
        {
            if(r-l+1<=1) return INF;
            if(r-l+1<=7)
            {
                double ans=INF;
                sort(p.begin()+l,p.begin()+r+1,[](const Point &a,const Point &b)
{return a.y<b.y;});
                for(int i=l;i<=r;i++)
                    for(int j=i+1;j<=r;j++)
                        ans=min(ans,distance(p[i],p[j]));
                return ans;
            }
            int mid=(l+r)/2;
            double w=p[mid].x;
            double d=min(solve(l,mid),solve(mid+1,r));
            inplace_merge(p.begin()+l,p.begin()+mid+1,p.begin()+r+1,[](const
Point &a,const Point &b){return a.y<b.y;});
            vector<Point>q;
            for(int i=l;i<=r;i++)
                if(abs(w-p[i].x)<=d) q.emplace_back(p[i]);
            for(int i=0,j=0;i<(int)q.size();i++)
            {
                while(j<(int)q.size()&&q[j].y<=q[i].y+d) j++;
                for(int k=i+1;k<j;k++)
                    d=min(d,distance(q[i],q[k]));
            }
            return d;
        };
        return solve(0,p.size()-1);
    }
    class Circle
    {
    public:
        Point o;
        double r;
        Circle(){}
        Circle(const Point &_o,const double &_r):o(_o),r(_r){}
        friend istream &operator>>(istream &in,Circle &obj)
```

```
    {
        in>>obj.o>>obj.r;
        return in;
    }
    friend ostream &operator<<(ostream &out,const Circle &obj)
    {
        out<<obj.o<<" "<<obj.r;
        return out;
    }
    friend bool operator==(const Circle &a,const Circle &b)
    {
        return a.o==b.o&&equal(a.r,b.r);
    }
    friend bool operator!=(const Circle &a,const Circle &b)
    {
        return a.o!=b.o||(!equal(a.r,b.r));
    }
    double area()const
    {
        return PI*r*r;
    }
    bool is_tangent(const Line &l)const
    {
        return equal(Geometry::distance(l.projection(o),o),r);
    }
    int point_containment(const Point &p)const
    {
        double d=distance(o,p);
        if(equal(d,r)) return ON;
        else if(less(d,r)) return IN;
        else return OUT;
    }
    vector<Point>cross_point(const Line &l)const
    {
        Point pr=l.projection(o),e=(l.b-l.a).unit();
        double d=distance(pr,o);
        if(greater(d,r)) return {};
        double t=sqrtl(r*r-distance(pr,o)*distance(pr,o));
        if(equal(t,0)) return {pr};
        else return {pr-e*t,pr+e*t};
    }
    vector<Point>cross_point(const Circle &c)const
    {
        double d=distance(o,c.o);
        if(less(d,abs(r-c.r))||greater(d,r+c.r)) return {};
        double x=(r*r-c.r*c.r+d*d)/(d*2),h=sqrtl(r*r-x*x);
        Point p=o+(c.o-o).unit()*x;
        if(equal(d,abs(r-c.r))||equal(d,r+c.r)) return {p};
        Point v=(c.o-o).unit().rotate(PI/2)*h;
        return {p-v,p+v};
    }
    vector<Point>tangent(const Point &p)const
    {
        double d=distance(o,p);
        if(greater(r,d)) return {};
        if(equal(d,r)) return {p};
```

```
            return cross_point(Circle(p,sqrtl(d*d-r*r)));
        }
        vector<Line>common_tangent_out(const Circle &c)const
        {
            assert(*this!=c);
            if(equal(r,c.r))
            {
                Point p=(c.o-o).unit().rotate(PI/2)*r;
                return {Line(o-p,c.o-p),Line(o+p,c.o+p)};
            }
            double d=distance(o,c.o);
            if(less(d,abs(r-c.r))) return {};
            if(equal(d,abs(r-c.r)))
            {
                Point p;
                if(r>c.r) p=o+(c.o-o).unit()*r;
                else p=c.o+(o-c.o).unit()*c.r;
                return {Line(p,p)};
            }
            Point p((o.x*c.r-c.o.x*r)/(c.r-r),(o.y*c.r-c.o.y*r)/(c.r-r));
            vector<Point>p1=tangent(p),p2=c.tangent(p);
            assert((int)p1.size()==2&&(int)p2.size()==2);
            return {Line(p1[0],p2[0]),Line(p1[1],p2[1])};
        }
        vector<Line>common_tangent_in(const Circle &c)const
        {
            assert(*this!=c);
            double d=distance(o,c.o);
            if(less(d,abs(r+c.r))) return {};
            if(equal(d,abs(r+c.r)))
            {
                Point p=o+(c.o-o).unit()*r;
                return {Line(p,p)};
            }
            Point p((o.x*c.r+c.o.x*r)/(r+c.r),(o.y*c.r+c.o.y*r)/(r+c.r));
            vector<Point>p1=tangent(p),p2=c.tangent(p);
            assert((int)p1.size()==2&&(int)p2.size()==2);
            return {Line(p1[0],p2[0]),Line(p1[1],p2[1])};
        }
        vector<Line>common_tangent(const Circle &c)const
        {
            assert(*this!=c);
            vector<Line>f=common_tangent_out(c),g=common_tangent_in(c);
            for(const Line &l:g)
                f.emplace_back(l);
            g.clear();
            sort(f.begin(),f.end(),[](const Line &x,const Line &y){return
x.a.x<y.a.x||(x.a.x==x.a.x&&x.a.y<x.a.y);});
            return f;
        }
        double intersection_area(const Point &a,const Point &b)const
        {
            bool ta=less_equal(distance(o,a),r),tb=less_equal(distance(o,b),r);
            if(ta&&tb) return cross(a-o,b-o)/2;
            vector<Point>t=cross_point(Line(b,a));
```

```
            if(ta&&!tb) return angle(t.front()-o,b-o)*r*r/2+cross(a-o,t.front()-
o)/2;
            if(!ta&&tb) return angle(a-o,t.back()-o)*r*r/2+cross(t.back()-o,b-
o)/2;
            double s=angle(a-o,b-o)*r*r/2;
            if(greater_equal(Line(a,b).distance(o),r)) return s;
            return s+angle(t.front()-o,t.back()-o)*r*r/2-cross(t.front()-
o,t.back()-o)/2;
        }
        double intersection_area(const Polygon &g)const
        {
            int n=g.size();
            double s=0;
            for(int i=0;i<n;i++)
                s+=intersection_area(g[i],g[(i+1)%n]);
            return s;
        }
        double intersection_area(const Circle &c)const
        {
            double d=distance(o,c.o);
            if(greater(d,r+c.r)) return 0;
            if(less_equal(d,abs(r-c.r))) return min(area(),c.area());
            vector<Point>t=cross_point(c);
            double alpha=acosl((d*d+r*r-
c.r*c.r)/(2*d*r))*2,beta=acosl((d*d+c.r*c.r-r*r)/(2*d*c.r))*2;
            double
s1=alpha*r*r/2,s2=beta*c.r*c.r/2,s3=sinl(alpha)*r*r/2+sinl(beta)*c.r*c.r/2;
            return s1+s2-s3;
        }
    };
    const int SEPARATED=4,CIRCUMSCRIBED=3,INTERSECTED=2,INSCRIBED=1,INCLUDED=0;
    int intersection(const Circle &a,const Circle &b)
    {
        double d=distance(a.o,b.o);
        if(greater(d,a.r+b.r)) return SEPARATED;
        else if(equal(d,a.r+b.r)) return CIRCUMSCRIBED;
        else if(greater(d,abs(a.r-b.r))) return INTERSECTED;
        else if(equal(d,abs(a.r-b.r))) return INSCRIBED;
        else return INCLUDED;
    }
    class Triangle
    {
    public:
        Point A,B,C;
        Triangle(){}
        Triangle(const Point &_A,const Point &_B,const Point
&_C):A(_A),B(_B),C(_C){}
        friend istream &operator>>(istream &in,Triangle &obj)
        {
            in>>obj.A>>obj.B>>obj.C;
            return in;
        }
        friend ostream &operator<<(ostream &out,const Triangle &obj)
        {
            out<<obj.A<<" "<<obj.B<<" "<<obj.C;
            return out;
```

```
        }
        Circle inscribed_circle()const
        {
            double a=distance(B,C),b=distance(A,C),c=distance(A,B);
            double p=(a+b+c)/2;
            double s=abs(cross(B-A,C-A))/2;
            double r=s/p;
            Point o((a*A.x+b*B.x+c*C.x)/(a+b+c),(a*A.y+b*B.y+c*C.y)/(a+b+c));
            return Circle(o,r);
        }
        Circle circumscribed_circle()const
        {
            double t1=A.x*A.x+A.y*A.y;
            double t2=B.x*B.x+B.y*B.y;
            double t3=C.x*C.x+C.y*C.y;
            double t=A.x*B.y+B.x*C.y+C.x*A.y-A.x*C.y-B.x*A.y-C.x*B.y;
            Point o((t2*C.y+t1*B.y+t3*A.y-t2*A.y-t3*B.y-t1*C.y)/t/2,
(t3*B.x+t2*A.x+t1*C.x-t1*B.x-t2*C.x-t3*A.x)/t/2);
            double a=distance(B,C),b=distance(A,C),c=distance(A,B);
            double s=abs(cross(B-A,C-A))/2;
            double r=a*b*c/(4*s);
            return Circle(o,r);
        }
    };
    Circle smallest_enclosing_circle(const vector<Point> &_p)
    {
        vector<Point>p=_p;
        shuffle(p.begin(),p.end(),rnd);
        int n=p.size();
        Circle c=Circle(Point(0,0),0);
        for(int i=0;i<n;i++)
            if(c.point_containment(p[i])==OUT)
            {
                c=Circle(p[i],0);
                for(int j=0;j<i;j++)
                    if(c.point_containment(p[j])==OUT)
                    {
                        c=Circle((p[i]+p[j])/2,distance(p[i],p[j])/2);
                        for(int k=0;k<j;k++)
                            if(c.point_containment(p[k])==OUT)

 c=Triangle(p[i],p[j],p[k]).circumscribed_circle();
                    }
            }
        return c;
    }
}
using namespace Geometry;
```

杂项

离线算法

线段树分治

Luogu P5787 二分图 /【模板】线段树分治

> 你需要维护一个 n 个点 m 条边的无向图。第 i 条边为 (x_i, y_i)，出现的时刻为 $[l_i, r_i)$，其余时刻消失。
>
> 对于每一个时刻，若此时该图为二分图，输出 `Yes`，否则输出 `No`。

```
#include<iostream>
#include<cstdio>
#include<tuple>
#include<stack>
#include<vector>
using namespace std;
const int N=100005;
int n,m,k;
int fa[N+N],h[N+N];
int getf(int v)
{
    if(v==fa[v]) return v;
    else return getf(fa[v]);
}
stack<tuple<int,int,int>>stk;
void merge(int u,int v)
{
    int f1=getf(u),f2=getf(v);
    if(f1!=f2)
    {
        if(h[f1]>h[f2]) swap(f1,f2);
        stk.emplace(f1,f2,h[f1]==h[f2]);
        if(h[f1]==h[f2]) h[f2]++;
        fa[f1]=f2;
    }
    return;
}
bool ans[N];
struct Segment_Tree
{
    #define ls i*2
    #define rs i*2+1
    struct Node
    {
        int l,r;
        vector<pair<int,int>>p;
    }tree[N<<2];
    void build(int i,int l,int r)
    {
        tree[i].l=l,tree[i].r=r;
        if(l==r)
        {
            tree[i].p.clear();
```

```
            return;
        }
        int mid=(l+r)/2;
        build(ls,l,mid);
        build(rs,mid+1,r);
        return;
    }
    void modify(int i,int l,int r,pair<int,int> e)
    {
        if(l<=tree[i].l&&tree[i].r<=r)
        {
            tree[i].p.emplace_back(e);
            return;
        }
        if(l<=tree[ls].r) modify(ls,l,r,e);
        if(r>=tree[rs].l) modify(rs,l,r,e);
        return;
    }
    void solve(int i)
    {
        auto pre=make_tuple(-1,-1,-1);
        if(!stk.empty()) pre=stk.top();
        bool flag=true;
        for(auto [u,v]:tree[i].p)
        {
            if(getf(u)==getf(v))
            {
                flag=false;
                break;
            }
            merge(u,v+n);
            merge(u+n,v);
        }
        if(!flag)
        {
            for(int k=tree[i].l;k<=tree[i].r;k++)
                ans[k]=false;
        }
        else
        {
            if(tree[i].l==tree[i].r) ans[tree[i].l]=true;
            else
            {
                solve(ls);
                solve(rs);
            }
        }
        while(!stk.empty()&&stk.top()!=pre)
        {
            auto [f1,f2,t]=stk.top();
            stk.pop();
            fa[f1]=f1;
            if(t) h[f2]--;
        }
        return;
    }
```

```
    #undef ls
    #undef rs
}T;
int main()
{
    scanf("%d%d%d",&n,&m,&k);
    for(int i=1;i<=n+n;i++)
        fa[i]=i,h[i]=1;
    T.build(1,1,k);
    for(int i=1;i<=m;i++)
    {
        int x,y,l,r;
        scanf("%d%d%d%d",&x,&y,&l,&r);
        if(l+1<=r) T.modify(1,l+1,r,make_pair(x,y));
    }
    T.solve(1);
    for(int i=1;i<=k;i++)
        if(ans[i]) printf("Yes\n");
        else printf("No\n");
    return 0;
}
```

莫队

普通莫队

[SPOJ DQUERY - D-query]

> 给出一个长度为 n 的数列 $a_1, a_2, \ldots, a_n$，有 q 个询问，每个询问给出数对 (i, j)，需要你给出 $a_i, a_{i+1}, \ldots, a_j$ 这一段中有多少不同的数字。

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const int N=30005,M=1000005,Q=200005;
int SIZE;
int n,q;
int a[N];
int bel[N],tot;
struct Query
{
    int l,r,id;
}query[Q];
int cnt[M];
int ans[Q];
int now;
void add(int val)
{
    if(cnt[val]==0) now++;
    cnt[val]++;
    return;
```

```
}
void del(int val)
{
    cnt[val]--;
    if(cnt[val]==0) now--;
    return;
}
int main()
{
    scanf("%d",&n);
    SIZE=sqrt(n),tot=(n-1)/SIZE+1;
    for(int i=1;i<=n;i++)
        bel[i]=(i-1)/SIZE+1;
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    scanf("%d",&q);
    for(int i=1;i<=q;i++)
    {
        scanf("%d%d",&query[i].l,&query[i].r);
        query[i].id=i;
    }
    sort(query+1,query+q+1,[&](const Query &x,const Query &y)
    {
        if(bel[x.l]!=bel[y.l]) return bel[x.l]<bel[y.l];
        else return (bel[x.l]&1)?x.r<y.r:x.r>y.r;
    });
    int l=1,r=0;
    for(int i=1;i<=q;i++)
    {
        int ql=query[i].l,qr=query[i].r;
        while(l<ql) del(a[l++]);
        while(l>ql) add(a[--l]);
        while(r<qr) add(a[++r]);
        while(r>qr) del(a[r--]);
        ans[query[i].id]=now;
    }
    for(int i=1;i<=q;i++)
        printf("%d\n",ans[i]);
    return 0;
}
```

带修改莫队

Luogu P1903 [国家集训队] 数颜色 / 维护队列

> 墨墨购买了一套 N 支彩色画笔（其中有些颜色可能相同），摆成一排，你需要回答墨墨的提问。墨墨会向你发布如下指令：
>
> 1. $Q\ L\ R$ 代表询问你从第 L 支画笔到第 R 支画笔中共有几种不同颜色的画笔。
> 2. $R\ P\ C$ 把第 P 支画笔替换为颜色 C。

```
#include<iostream>
#include<cstdio>
#include<cmath>
```

```
#include<algorithm>
using namespace std;
const int N=133338,M=1000005;
int SIZE;
int n,m,q;
int bel[N],tot;
int a[N];
struct Query
{
    int l,r,time,id;
}query[N];
struct Change
{
    int pos,val;
}change[N];
int res[N];
int idx;
int cnt[M],cur;
void add(int val)
{
    if(cnt[val]==0) cur++;
    cnt[val]++;
    return;
}
void del(int val)
{
    cnt[val]--;
    if(cnt[val]==0) cur--;
    return;
}
void can(int now,int i)
{
    if(query[i].l<=change[now].pos&&change[now].pos<=query[i].r)
    {
        del(a[change[now].pos]);
        add(change[now].val);
    }
    swap(change[now].val,a[change[now].pos]);
    return;
}
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    SIZE=pow(n,0.75)+1;
    tot=(n-1)/SIZE+1;
    for(int i=1;i<=n;i++)
        bel[i]=(i-1)/SIZE+1;
    while(m--)
    {
        char ch;
        for(ch=getchar();ch!='Q'&&ch!='R';ch=getchar());
        if(ch=='Q')
        {
            q++;
```

```
            scanf("%d%d",&query[q].l,&query[q].r);
            query[q].time=idx;
            query[q].id=q;
        }
        else if(ch=='R')
        {
            idx++;
            scanf("%d%d",&change[idx].pos,&change[idx].val);
        }
    }
    sort(query+1,query+q+1,[&](const Query &x,const Query &y)
    {
        if(bel[x.l]!=bel[y.l]) return bel[x.l]<bel[y.l];
        else if(bel[x.r]!=bel[y.r]) return bel[x.r]<bel[y.r];
        else return x.time<y.time;
    });
    int l=1,r=0,now=0;
    for(int i=1;i<=q;i++)
    {
        int ql=query[i].l,qr=query[i].r,qtime=query[i].time;
        while(l<ql) del(a[l++]);
        while(l>ql) add(a[--l]);
        while(r<qr) add(a[++r]);
        while(r>qr) del(a[r--]);
        while(now<qtime) can(++now,i);
        while(now>qtime) can(now--,i);
        res[query[i].id]=cur;
    }
    for(int i=1;i<=q;i++)
        printf("%d\n",res[i]);
    return 0;
}
```

回滚莫队

[JOISC 2014 Day1 历史研究]

> 给你一个长度为 n 的数组 A 和 m 个询问 $(1 \leq n, m \leq 10^5)$，每次询问一个区间 $[L, R]$ 内重要度最大的数字，要求 输出其重要度。一个数字 i 重要度的定义为 i 乘上 i 在区间内出现的次数。

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const int N=100005;
int n,q;
int a[N];
int b[N],m;
int bel[N];
int SIZE;
struct Block
{
    int l,r;
```

```
}block[N];
void init()
{
    SIZE=sqrt(n);
    for(int i=1;i<=n;i++)
        bel[i]=(i-1)/SIZE+1;
    for(int i=1;i<=(n-1)/SIZE+1;i++)
        block[i].l=(i-1)*SIZE+1,block[i].r=min(i*SIZE,n);
    return;
}
struct Query
{
    int l,r,id;
}query[N];
int l,r;
long long res;
long long ans[N];
int cnt[N];
void add(int i)
{
    cnt[a[i]]++;
    res=max(res,(long long)b[a[i]]*cnt[a[i]]);
    return;
}
void del(int i)
{
    cnt[a[i]]--;
    return;
}
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++)
        b[++m]=a[i];
    sort(b+1,b+m+1);
    m=unique(b+1,b+m+1)-b-1;
    for(int i=1;i<=n;i++)
        a[i]=lower_bound(b+1,b+m+1,a[i])-b;
    init();
    for(int i=1;i<=q;i++)
        scanf("%d%d",&query[i].l,&query[i].r),query[i].id=i;
    sort(query+1,query+q+1,[=](const Query &a,const Query &b){return
bel[a.l]!=bel[b.l]?bel[a.l]<bel[b.l]:a.r<b.r;});
    int l=1,r=0;
    int last=0;
    for(int t=1;t<=q;t++)
    {
        if(bel[query[t].l]==bel[query[t].r])
        {
            static int c[N];
            long long now=0;
            for(int i=query[t].l;i<=query[t].r;i++)
                c[a[i]]++;
            for(int i=query[t].l;i<=query[t].r;i++)
```

```
                now=max(now,(long long)b[a[i]]*c[a[i]]);
            for(int i=query[t].l;i<=query[t].r;i++)
                c[a[i]]--;
            ans[query[t].id]=now;
            continue;
        }
        if(bel[query[t].l]!=last)
        {
            while(r>block[bel[query[t].l]].r) del(r),r--;
            while(l<=block[bel[query[t].l]].r) del(l),l++;
            res=0;
            last=bel[query[t].l];
        }
        while(r<query[t].r) r++,add(r);
        long long tmp=res;
        int L=l;
        while(L>query[t].l) L--,add(L);
        ans[query[t].id]=res;
        while(L<l) del(L),L++;
        res=tmp;
    }
    for(int i=1;i<=q;i++)
        printf("%lld\n",ans[i]);
    return 0;
}
```