代码模板

zdq

SCUT

August 27, 2021

Contents

开始

宏定义

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
typedef __int128 LLL;
typedef unsigned u32;
typedef unsigned long long u64;
typedef long double LD;
#define il inline
#define pln putchar('\n')
#define For(i,a,b)  for(int i=(a),(i##i)=(b);i<=(i##i);++i)
#define Rep(i,n)    for(int i=0,(i##i)=(n);i<(i##i);++i)
#define Fodn(i,a,b) for(int i=(a),(i##i)=(b);i>=(i##i);--i)
const int M=1000000007,INF=0x3f3f3f3f;
const long long INFLL=0x3f3f3f3f3f3f3f3fLL;
const int N=1000009;
```

快读

```cpp
ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
```

```cpp
template <typename T>
inline bool read(T &x) {
    x = 0; char c = getchar(); int f = 1;
    while (!isdigit(c) && (c != '-') && (c != EOF)) c = getchar();
    if (c == EOF) return 0;
    if (c == '-') f = -1, c = getchar();
    while (isdigit(c)) { x = x * 10 + (c & 15); c = getchar();}
    x *= f; return 1;
}

template <typename T, typename... Args>
inline bool read(T &x, Args &...args) {
    bool res = 1;
    res &= read(x);
    res &= read(args...);
    return res;
}
```

对拍

```cpp
//in.txt
//AC.exe std.txt
//MY.exe my.txt

void init(){
    FILE*F=fopen("int.txt","w");

    //srand(time(0));
    //int a=(long long)rand()*rand()%1001;
    //fscanf(F,"%d",&a);fprintf(F,"%d\n",a);

    fclose(F);
}

int main(){
    init();
    while(1){
        system("AC.exe < in.txt > std.txt");

        system("MY.exe < in.txt > my.txt");

        if(system("fc std.txt my.txt")){
            puts("WA");
            return 0;
        }else puts("AC\n\n");

```

```
        init();
    }
}
```

数学

模乘模幂

- longlong 范围用 fpl

```
inline LL mul(LL a, LL b, LL p) {
    LL res = a * b - ((LL)((LD)a * b / p) * p);
    return res < 0 ? res + p : (res < p ? res : res - p);
}

inline LL fp(LL a, LL b, LL Mod) {
    LL res = (Mod != 1);
    for (; b; b >>= 1, a = a * a % Mod)
        if (b & 1)
            res = res * a % Mod;
    return res;
}

inline LL fpl(LL a, LL b, LL Mod) {
    LL res = (Mod != 1);
    for (; b; b >>= 1, a = mul(a, a, Mod))
        if (b & 1)
            res = mul(res, a, Mod);
    return res;
}
```

GCD

```
template <typename T>
inline T gcd(T a, T b) {
    while (b){
        T t = b;
        b = a % b;
        a = t;
    }
    return a;
}

template <typename T>
inline T lcm(T a, T b) { return a / gcd(a, b) * b; }

template<typename T>
inline T exgcd(T a,T b,T&x,T&y){
    x=1,y=0;T m=0,n=1;
    while(b){
        const T q=a/b;
        tie(x,m)=make_tuple(m,x-q*m);
        tie(y,n)=make_tuple(n,y-q*n);
        tie(a,b)=make_tuple(b,a-q*b);
    }
    return a;
}
```

CRT

- 需要 GCD 64 位模乘
- 用来合并同余方程
- 返回最小正数解或最小非负解无解返回-1

```
inline LL Crt(LL a1, LL a2, LL mod1, LL mod2) {
    LL u, v;
    LL g = exgcd(mod1, mod2, u, v);
    if ((a2 - a1) % g) return -1;
```

```cpp
    LL m12 = abs(lcm(mod1, mod2));
    LL res = (mul(mod1, mul(u, ((a2 - a1) / g), m12), m12) + a1) % m12;
    return res <= 0 ? res + m12 : res; /* 求最小正数解还是非负解 */
}
```

高次同余方程

- 以下 p 均为正整数

BSGS

- 求最小非负整数 x 满足 `a^x=b(mod p)`
- 要求 `(a,p)=1` p 不必是质数无解返回 -1
- 需要 `fp` 和 `map/unorded_map` 如果 p 超过 1e9 用 LL 版本
- 期望复杂度 $O(\sqrt{p})$
- 自带哈希可以被卡 TLE (luogu P4192)
- – 用 `map` 复杂度多个 log 慢
- – 手写整数哈希 `unordered_map<int,int,custom_hash>h;`
- 若多组询问 a,p 不变且始终满足 (b,p)=1 可以复用 h 降低复杂度

```cpp
inline int log_p(int a, int b, int p){
    if ((b - 1) % p == 0)return 0;
    if ((b - a) % p == 0)return 1;
    //在%p 前特判 此后满足 p>1 且 result>1
    a %= p; (a < 0) && (a += p);
    b %= p; (b < 0) && (b += p);
    //assert(abs(gcd(a,p))==1);//调用时保证 a⊠p 则此行可省
    unordered_map<int, int> h; //仔细判断出题人卡不卡
    int t = (int)sqrt(p) + 1;
    for (int j = 0; j < t; ++j, b = (LL)b * a % p) h[b] = j;
    int at = (a = fp(a, t, p));
    for (int i = 1, j = t; i <= t; ++i, a = (LL)a * at % p, j += t){
        auto bat = h.find(a);
        if (bat != h.end()) return j - (*bat).second;
    } return -1;
}
```

EXBSGS

- 求最小非负整数 x 满足 `a^x=b(mod p)`
- 不要求正整数 a,p 互质无解返回 -1
- 注意乘法溢出

```cpp
inline int getinv(int a, int p){
    int m = 0, n = 1, x = 1, y = 0, b = p;
    while (b){
        int q = a / b;
        tie(x, m) = make_tuple(m, x - q * m);
        tie(y, n) = make_tuple(n, y - q * n);
        tie(a, b) = make_tuple(b, a - q * b);
    }
    x %= p; (x < 0) && (x += p);
    return x;
}

inline int log(int a,int b,int p){
    if ((b - 1) % p == 0)return 0;
    if ((b - a) % p == 0)return 1;
    a %= p; (a < 0) && (a += p);
    b %= p; (b < 0) && (b += p);
    int _a = a, _b = b, _p = p;
    int d = gcd(a, p), ax = 1, t = 0;
```

```
    while (d > 1){
        if (b % d != 0)break; //若 b 不能再提出 d 说明 result<=t
        b /= d; p /= d; ax = (LL)ax * (a / d) % p;
        ++t;
        d = gcd(a, p);
    }
    for (int i = 2, at = (LL)_a * _a % _p; i <= t; ++i, at = (LL)at * _a % _p)
        if (at == _b) return i;
    if (d > 1) return -1;
    int res = log_p(a, (LL)b * getinv(ax, p) % p, p);
    return res < 0 ? -1 : res + t;
}
```

二次剩余

判定定理

- p 是奇素数且 n 和 p 互质
- $n^{\frac{p-1}{2}} mod\, p = \begin{cases} 1 & n \\ -1 & n \end{cases}$

Cipolla 算法

- 寻找非负整数 x 满足 $x^2 \equiv n\,(mod\, p)$ 其中 p 为质数
- 期望复杂度 O(log p)
- 需要用到快速幂 `fp()` 若 p 大于 1e9 要换 `mul()` 和 `fpl()`
- 注意乘法溢出
-   – 先找到 a 使 $a^2 - n$ 是非二次剩余
-   – 定义 $w = i^2 = a^2 - n$ 则 $x = (a+i)^{\frac{p+1}{2}}$
- 无解返回 -1

```
namespace Cipolla{
    LL w,p,d;

    struct dat{
        LL x,y;//复数 x+yi
        dat operator*(const dat&a)const{
            return(dat){(x*a.x+y*a.y%p*w)%p,(x*a.y+y*a.x)%p};
        }
    };

    inline LL ask(LL n,LL P){//n 为非二次剩余返回-1
        p=P;n%=p;if(n==0||p==2)return n;
        d=(p-1)/2;
        if(fp(n,d,p)!=1)return -1;//检验是否有解
        LL a;
        while(1){
            a=(LL)rand()*rand()%p;
            w=(a*a-n)%p;(w<0)&&(w+=p);
            if(fp(w,d,p)==(p-1))break;//找到了非二次剩余 w
        }
        dat res={1,0};++d;
        for(dat u={a,1};d;d>>=1,u=u*u)if(d&1)res=res*u;
        return res.x;
    }
}

signed main(){//luogu P5491
    int t;scanf("%d",&t);
    while(t--){
        LL n,p;scanf("%lld%lld",&n,&p);
        LL ans=Cipolla::ask(n,p);
        if(ans==-1)puts("Hola!");
```

```
        else if(ans==0)puts("0");
        else printf("%lld %lld\n",min(ans,p-ans),max(ans,p-ans));
    }
    return 0;
}
```

线性筛

```
struct primenumberlist{
#define MAXN (100000000)
    int cnt, pri[10000000];
    bool np[MAXN + 10];
    primenumberlist(){
        np[1] = 1; cnt = 0;
        for (int i = 2; i <= MAXN; ++i) {
            if (!np[i]) pri[++cnt] = i;
            for (int j = 1; j <= cnt; ++j) {
                LL t = pri[j] * i;
                if (t > MAXN) break;
                np[t] = 1;
                if (!(i % pri[j])) break;
            }
        }
    }
} prime;
```

ϕ 单点欧拉函数

```
template <typename T>
inline T phi(T x) {
    T res = x;
    for (T 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;
}
```

Miller-Rabin 素性测试

- $n <= 10^{18}$
- 需要 64 位模乘 64 位模幂
- base 少的非常容易 hack (luogu 数据太弱 loj 不错
- http://miller-rabin.appspot.com/
- 此实现仍有被 hack 可能 WA 了就换 base:
-   - 2^64 以内保证正确的 7 个 base: 2, 325, 9375, 28178, 450775, 9780504, 1795265022

```
inline bool MR(LL a,const LL&d,const int&t,const LL&n){//注意 a%n==0 时应判定通过测试
    a%=n;if(a==0)return 1;
    a=fpl(a,d,n);
    if(a==1||a==(n-1))return 1;
    for(int i=1;i<t;++i){
        a=mul(a,a,n);
        if(a==(n-1))return 1;
        if(a==1)return 0;
    }
    return 0;
}

constexpr LL a[]={2, 123635709730000, 9233062284813009, 43835965440333360, 761179012939631437, 1263739024124850375};
inline bool isPrime(LL n){
    if(n==2||n==3)return 1;if(n<2||(n%2==0)||(n%3==0))return 0;//特判最好别省
    if(n==585226005592931977LL)return 0;//这 6 个基的最小反例 暂不清楚 1e18 内有无其它反例
```

```
    LL d=n-1;int t=0;while((d&1)==0)d>>=1,++t;
    for(int i=0;i<6;++i)if(!MR(a[i],d,t,n))return 0;
    return 1;
}
```

Pollard-Rho 分解质因数

- 需要 64 位模乘 gcd
- 返回 n 的一个大于 1 的真因子不存在就死循环 gg
- 务必在调用前确认 n 是合数 (>1)

```
mt19937 mt(time(0)); //随机化
inline LL PR(LL n) {
    LL x = uniform_int_distribution<LL>(0, n - 1)(mt), s, t, c = uniform_int_distribution<LL>(1, n - 1)(mt); //随机化
    for (int gol = 1; 1; gol <<= 1, s = t, x = 1) {
        for (int stp = 1; stp <= gol; ++stp) {
            t = (mul(t, t, n) + c) % n;
            x = mul(x, abs(s - t), n);
            if ((stp & 127) == 0) {
                LL d = gcd(x, n);
                if (d > 1) return d;
            }
        }
        LL d = gcd(x, n);
        if (d > 1) return d;
    }
}
```

组合数

- 数较小模数为较大质数求逆元
- – 如果模数固定可以 O(n) 预处理阶乘的逆元
- 数较大模数为较小质数用 $Lucas$ 定理
- –
$$C_n^m \equiv C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n\ mod\ p}^{m\ mod\ p} (mod\ p)$$
- 数较大模数较小用 $exLucas$ 定理求 $C_n^m mod\ P$

exLucas

- 需要模乘 CRT
- O(P log P)
- 不要求 P 为质数

```
namespace EXLUCAS {
    inline LL idxp(LL n, LL p) {
        LL nn = n;
        while (n > 0) nn -= (n % p), n /= p;
        return nn / (p - 1);
    }

    LL facp(LL n, LL p, LL pk) {
        if (n == 0) return 1;
        LL res = 1;
        if (n >= pk) {
            LL t = n / pk, k = 1, els = n - t * pk;
            for (LL i = 1; i <= els; ++i) if (i % p) k = k * i % pk;
            res = k;
            for (LL i = els + 1; i < pk; ++i) if (i % p) k = k * i % pk;
            res = res * fp(k, n / pk, pk) % pk;
        }
        else for (LL i = 1; i <= n; ++i) if (i % p) res = res * i % pk;
```

```
        return res * facp(n / p, p, pk) % pk;
    }

    inline LL exlucas(LL n, LL m, LL p, LL pk, LL k) {
        LL a = facp(n, p, pk) * fp(facp(n - m, p, pk) * facp(m, p, pk) % pk, pk / p * (p - 1) - 1, pk) % pk;
        LL b = idxp(n, p) - idxp(m, p) - idxp(n - m, p);
        if (b >= k) return 0;
        while (b--) a *= p;
        return a % pk;
    }

    /* 接口 */ inline LL exlucas(LL n, LL m, LL p) {
        LL a = 0, b = 1;
        for (LL i = 2; i * i <= p; ++i) {
            if (p % i) continue;
            LL t = 0, pk = 1;
            while (p % i == 0) ++t, p /= i, pk *= i;
            a = Crt(a, exlucas(n, m, i, pk, t), b, pk);
            b *= pk;
        }
        return (p > 1) ? Crt(a, exlucas(n, m, p, p, 1), b, p) : a;
    }
}
```

类欧几里得算法

洛谷模板题

- 计算直线下整点数
- f=∑[(ai+b)/c] g=∑i[(ai+b)/c] h=∑[(ai+b)/c]^2 i=0..n a,b,n⊠N c⊠N*
- 复杂度 log(MAX{a,c})

```
struct dat{LL f, g, h;};
const LL i2 = 499122177, i3 = 332748118, M = 998244353; //预处理出模 M 意义下 2 和 3 的逆元
dat f(LL a, LL b, LL c, LL n){
    LL ac = a / c, bc = b / c;
    LL n2 = (n * (n + 1) % M) * i2 % M, n3 = n2 * (2ll * n + 1) % M * i3 % M;
    dat res = {
        (n2 * ac % M + (n + 1) * bc % M) % M,
        (ac * n3 % M + bc * n2 % M) % M,
        (ac * ac % M * n3 % M +
            bc * bc % M * (n + 1) % M + ac * bc % M * n2 % M * 2ll) % M};
    a %= c; b %= c; if (a == 0)return res;
    LL m = (a * n + b) / c;
    dat p = f(c, c - b - 1, a, m - 1);
    LL fc = (n * m % M - p.f + M) % M, gc = (n2 * m % M - i2 * (p.f + p.h) % M + M) % M;
    return{(res.f + fc) % M,    (res.g + gc) % M,
    (res.h + 2ll * (bc * fc % M + ac * gc % M) % M +
        n * m % M * m % M - 2ll * p.g - p.f + 3ll * M) % M};}
```

LOJ 模板题

- 在学了

多项式乘法 FFT

- fft 计算卷积复杂度 O(nlogn) luogu P3803

```
#include<bits/stdc++.h>
using namespace std;
const int N=2100009;

//fft 计算卷积 复杂度 O(nlogn)  luogu P3803
namespace FFT{
    const double PI=acos(-1.0);
    struct comp{
        double x,y;
        comp(double _x=0,double _y=0):x(_x),y(_y){}
```

```
        comp operator+(const comp&a)const{return comp(x+a.x,y+a.y);}
        comp operator-(const comp&a)const{return comp(x-a.x,y-a.y);}
        comp operator*(const comp&a)const{return comp(x*a.x-y*a.y,x*a.y+y*a.x);}
    };
    int rev[N];//位逆序置换
    inline void getrev(int len){//每次 len 变化应重新计算 rev
        const int d=len>>1;
        for(int i=1;i<len;++i)rev[i]=(rev[i>>1]>>1)|((i&1)*d);
    }

    inline void dft(comp y[],int len,int on){//on==1 DFT   on==-1 IDFT
        for(int i=1;i<len;++i)if(i<rev[i])swap(y[i],y[rev[i]]);
        for(int mid=1;mid<len;mid<<=1){
            comp wn(cos(PI/mid),on*sin(PI/mid));
            for(int st=0,t=(mid<<1);st<len;st+=t){
                comp w(1,0);
                for(int i=st,j=st+mid;j<st+t;++i,++j,w=w*wn){
                    comp u=y[i],v=w*y[j];
                    y[i]=u+v;y[j]=u-v;
                }
            }
        }
        if(on==-1)for(int i=0;i<len;++i)y[i].x/=len,y[i].y/=len;//保证结果在 R 上可只除实部
    }
}

int n,m;
FFT::comp f1[N],f2[N];
signed main(){
    scanf("%d%d",&n,&m);
    for(int i=0;i<=n;++i)scanf("%lf",&f1[i].x);
    for(int i=0;i<=m;++i)scanf("%lf",&f2[i].x);
    int len=1;while(len<=m+n)len<<=1;FFT::getrev(len);
    FFT::dft(f1,len,1);FFT::dft(f2,len,1);
    for(int i=0;i<len;++i)f1[i]=f1[i]*f2[i];
    FFT::dft(f1,len,-1);
    for(int i=0;i<=n+m;++i)printf("%d ",(int)(f1[i].x+0.5));
    return 0;
}
```

数值积分

- 自适应辛普森算法
- 复杂度 O(玄学) (调参够好 || 数据够巧)==AC
- luogu p4542

```
#include<bits/stdc++.h>
using namespace std;

struct integration{//∫f(x)dx≈(r-l)*(f(l)+f(r)+4*f((l+r)/2))/6
    typedef double T;
    const T EPS=(1e-6)*15;

    T aa,bb,cc,dd;
    integration(T _a,T _b,T _c,T _d):aa(_a),bb(_b),cc(_c),dd(_d){}

    T f(T x)const{//被积函数 f(x)
        return (cc*x+dd)/(aa*x+bb);
    }

    T simpson(T len,T l,T mid,T r)const{//时间优化 传入算好的函数值
        return len*(l+4*mid+r)/6;
    }
    T asr(T l,T r,T a,T c,T e,int stp)const{
        const T len=r-l,mid=l+len/2,lmid=l+len/4,rmid=mid+len/4;
        const T b=f(lmid),d=f(rmid);
        const T L=simpson(len/2,a,b,c),R=simpson(len/2,c,d,e),o=L+R-simpson(len,a,c,e);
        return (abs(o)<=EPS&&stp<0)?(L+R+o/15):asr(l,mid,a,b,c,stp-1)+asr(mid,r,c,d,e,stp-1);
    }
```

```
    T operator()(T l,T r)const{return asr(l,r,f(l),f(l+(r-l)/2),f(r),6);}//最小迭代次数
};

signed main(){
    double a,b,c,d,l,r;
    scanf("%lf%lf%lf%lf%lf%lf",&a,&b,&c,&d,&l,&r);
    integration F(a,b,c,d);
    printf("%.6lf\n",F(l,r));
    return 0;
}
```

日期操作

用于跳转的常量

```
const LL year_1[2]={365, 366};
const LL year_400=1460097;
const LL m_day[13]={(LL)0x3f3f3f, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

辅助函数

```
inline bool isLeap(LL t){return (t % 400 == 0) ||((t % 4 == 0) && (t % 100));}
inline bool pick(LL a, LL b){return ((isLeap(a) && b <= 2) ||(isLeap(a + 1) && b > 2));}
inline LL dayThisMonth(LL y, LL m){return m_day[m] + isLeap(y) * (m == 2);}
```

日期和整数的一一对应

- LL 可以改成 int

```
struct MY_DATE{
    LL year, month, day;
    MY_DATE(LL y = 2021, LL m = 1, LL d = 1) : year(y), month(m), day(d){};
    LL p(MY_DATE op = {0, 0, 0}){//日期转换为整数
        LL y = year - op.year, m = month - op.month, d = day - op.day;
        if (m <= 2){ y--; m += 12;}
        return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307;
    }
    MY_DATE run(LL k){//当前日期过 k 天
        k += p();
        LL x = k + 1789995, n = 4 * x / 146097, i, j, d;
        x -= (146097 * n + 3) / 4;
        i = (4000 * (x + 1)) / 1461001;
        x -= 1461 * i / 4 - 31;
        j = 80 * x / 2447;
        d = x - 2447 * j / 80;
        x = j / 11;
        return MY_DATE(100 * (n - 49) + i + x, j + 2 - 12 * x, d);
    }
};
```

二维计算几何

- Point 直接支持整型和浮点型
- 部分函数可以对整型改写
- 多边形 (凸包) 按逆时针存在下标 1..n

点向量基本运算

```
template <typename T>
struct Point {
    T x, y;
    Point() {}
    Point(T u, T v) : x(u), y(v) {}
    Point operator+(const Point &a) const { return Point(x + a.x, y + a.y); }
    Point operator-(const Point &a) const { return Point(x - a.x, y - a.y); }
    Point operator*(const T &a) const { return Point(x * a, y * a); }
    T operator*(const Point &a) const { return x * a.x + y * a.y; }
```

```
    T operator%(const Point &a) const { return x * a.y - y * a.x; }
    double len() const { return hypot(x, y); }
    double operator^(const Point &a) const { return (a - (*this)).len(); }
    double angle() const { return atan2(y, x); }
    bool id() const { return y < 0 || (y == 0 && x < 0); }
    bool operator<(const Point &a) const { return id() == a.id() ? (*this) % a > 0 : id() < a.id(); }
};
typedef Point<double> point;

#define sqr(x) ((x) * (x))
const point O(0, 0);
const double PI(acos(-1.0)), EPS(1e-8);
inline bool dcmp(const double &x, const double &y) { return fabs(x - y) < EPS; }
inline int sgn(const double &x) { return fabs(x) < EPS ? 0 : ((x < 0) ? -1 : 1); }
inline double mul(point p1, point p2, point p0) { return (p1 - p0) % (p2 - p0); }
```

位置关系

```
inline bool in_same_seg(point p, point a, point b) {
    if (fabs(mul(p, a, b)) < EPS) {
        if (a.x > b.x) swap(a, b);
        return (a.x <= p.x && p.x <= b.x && ((a.y <= p.y && p.y <= b.y) || (a.y >= p.y && p.y >= b.y)));
    } else return 0;
}

inline bool is_right(point st, point ed, point a) {
    return ((ed - st) % (a - st)) < 0;
}

inline point intersection(point s1, point t1, point s2, point t2) {
    return s1 + (t1 - s1) * (((s1 - s2) % (t2 - s2)) / ((t2 - s2) % (t1 - s1)));
}

inline bool parallel(point a, point b, point c, point d) {
    return dcmp((b - a) % (d - c), 0);
}

inline double point2line(point p, point s, point t) {
    return fabs(mul(p, s, t) / (t - s).len());
}

inline double point2seg(point p, point s, point t) {
    return sgn((t - s) * (p - s)) * sgn((s - t) * (p - t)) > 0 ? point2line(p, s, t) : min((p ^ s), (p ^ t));
}
```

多边形

求多边形面积

```
inline double area(int n, point s[]) {
    double res = 0;
    s[n + 1] = s[1];
    for (int i = 1; i <= n; ++i)
        res += s[i] % s[i + 1];
    return fabs(res / 2);
}
```

判断点是否在多边形内

- 特判边上的点
- 使用了 a[1]...a[n+1] 的数组

```
inline bool in_the_area(point p, int n, point area[]) {
    bool ans = 0; double x;
    area[n + 1] = area[1];
    for (int i = 1; i <= n; ++i) {
        point p1 = area[i], p2 = area[i + 1];
        if (in_same_seg(p, p1, p2)) return 1; //特判边上的点
        if (p1.y == p2.y) continue;
```

```
        if (p.y < min(p1.y, p2.y)) continue;
        if (p.y >= max(p1.y, p2.y)) continue;
        ans ^= (((p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x) > p.x);
    }
    return ans;
}
```

凸包

- *Andrew* 算法
- O(n log n)
- 可以应对凸包退化成直线/单点的情况但后续旋转卡壳时应注意特判
- 注意是否应该统计凸包边上的点

```
inline bool pcmp1(const point &a, const point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }

inline int Andrew(int n, point p[], point ans[]) { //ans[] 逆时针存凸包
    sort(p + 1, p + 1 + n, pcmp1);
    int m = 0;
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    int k = m;
    for (int i = n - 1; i >= 1; --i) {
        while (m > k && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    return m - (n > 1); //返回凸包有多少个点
}
```

凸包直径·平面最远点对

- 旋转卡壳算法
- O(n)
- 凸包的边上只能有端点，否则不满足严格单峰
- 凸包不能退化成直线，调用前务必检查 n>=3
- 使用了 a[1]...a[n+1] 的数组

```
inline double Rotating_Caliper(int n, point a[]) {
    a[n + 1] = a[1];
    double ans = 0;
    int j = 2;
    for (int i = 1; i <= n; ++i) {
        while (fabs(mul(a[i], a[i + 1], a[j])) < fabs(mul(a[i], a[i + 1], a[j + 1]))) j = (j % n + 1);
        ans = max(ans, max((a[j] ^ a[i]), (a[j] ^ a[i + 1])));
    }
    return ans;
}
```

平面最近点对

- 分治 + 归并
- O(n log n)

```
namespace find_the_closest_pair_of_points {
    const int N = 200010; //maxn
    inline bool cmp1(const point &a, const point &b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
    inline bool operator>(const point &a, const point &b) { return a.y > b.y || (a.y == b.y && a.x > b.x); }

    point a[N], b[N];
    double ans;
    inline void upd(const point &i, const point &j) { ans = min(ans, i ^ j); }

    void find(int l, int r) {
        if (l == r) return;
```

```
        if (l + 1 == r) {
            if (a[l] > a[r]) swap(a[l], a[r]);
            upd(a[l], a[r]); return;
        }
        int mid = (l + r) >> 1;
        double mx = (a[mid + 1].x + a[mid].x) / 2;
        find(l, mid); find(mid + 1, r);
        int i = l, j = mid + 1;
        for (int k = l; k <= r; ++k) b[k] = a[((j > r) || (i <= mid && a[j] > a[i])) ? (i++) : (j++)];
        for (int k = l; k <= r; ++k) a[k] = b[k];
        int tot = 0;
        for (int k = l; k <= r; ++k) if (fabs(a[k].x - mx) <= ans) {
            for (int j = tot; j >= 1 && (a[k].y - b[j].y <= ans); --j) upd(a[k], b[j]);
            b[++tot] = a[k];
        }
    }

    //接口
    inline double solve(int n, point ipt[]){
        ans = 0x3f3f3f3f3f3f3f3fll; //max distance
        for (int i = 1; i <= n; ++i) a[i] = ipt[i];
        sort(a + 1, a + 1 + n, cmp1);
        find(1, n);
        return ans;
    }
}
```

圆

三点垂心

```
inline point geto(point p1, point p2, point p3) {
    double a = p2.x - p1.x;
    double b = p2.y - p1.y;
    double c = p3.x - p2.x;
    double d = p3.y - p2.y;
    double e = sqr(p2.x) + sqr(p2.y) - sqr(p1.x) - sqr(p1.y);
    double f = sqr(p3.x) + sqr(p3.y) - sqr(p2.x) - sqr(p2.y);
    return {(f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2};
}
```

最小覆盖圆

- 随机增量 O(n)

```
inline void min_circlefill(point &o, double &r, int n, point a[]) {
    mt19937 myrand(20011224); shuffle(a + 1, a + 1 + n, myrand); //越随机越难 hack
    o = a[1];
    r = 0;
    for (int i = 1; i <= n; ++i) if ((a[i] ^ o) > r + EPS) {
        o = a[i];
        r = 0;
        for (int j = 1; j < i; ++j) if ((o ^ a[j]) > r + EPS) {
            o = (a[i] + a[j]) * 0.5;
            r = (a[i] ^ a[j]) * 0.5;
            for (int k = 1; k < j; ++k) if ((o ^ a[k]) > r + EPS) {
                o = geto(a[i], a[j], a[k]);
                r = (o ^ a[i]);
            }
        }
    }
}
```

图论

存图

- 前向星

- 注意边数开够

```cpp
int Head[N], Ver[N*2], Next[N*2], Ew[N*2], Gtot=1;
inline void graphinit(int n) {Gtot=1; for(int i=1; i<=n; ++i) Head[i]=0;}
inline void edge(int u, int v, int w=1) {Ver[++Gtot]=v; Next[Gtot]=Head[u]; Ew[Gtot]=w; Head[u]=Gtot;}
#define go(i,st,to) for (int i=Head[st], to=Ver[i]; i; i=Next[i], to=Ver[i])
```

最短路

Dijkstra

- 非负权图

```cpp
namespace DIJK{//适用非负权图 满足当前 dist 最小的点一定不会再被松弛
    typedef pair<long long,int> pii;
    long long dist[N];//存最短路长度
    bool vis[N];//记录每个点是否被从队列中取出  每个点只需第一次取出时扩展
    priority_queue<pii,vector<pii>,greater<pii> >pq;//维护当前 dist[] 最小值及对应下标 小根堆

    inline void dijk(int s,int n){//s 是源点 n 是点数
        while(pq.size())pq.pop();for(int i=1;i<=n;++i)dist[i]=INFLL,vis[i]=0;//所有变量初始化
        dist[s]=0;pq.push(make_pair(0,s));
        while(pq.size()){
            int now=pq.top().second;pq.pop();
            if(vis[now])continue;vis[now]=1;
            go(i,now,to){
                const long long relx(dist[now]+Ew[i]);
                if(dist[to]>relx){dist[to]=relx;pq.push(make_pair(dist[to],to));}//松弛
            }
        }
    }
}
```

LCA

- 倍增求 lca
- 数组开够

```cpp
namespace LCA_Log{
    int fa[N][22],dep[N];
    int t,now;
    void dfs(int x){
        dep[x]=dep[fa[x][0]]+1;
        go(i,x,to){
            if(dep[to])continue;
            fa[to][0]=x;for(int j=1;j<=t;++j)fa[to][j]=fa[fa[to][j-1]][j-1];
            dfs(to);
        }
    }

    //初始化接口
    inline void lcainit(int n,int rt){//记得初始化全部变量
        now=1;t=0;while(now<n)++t,now<<=1;
        for(int i=1;i<=n;++i)dep[i]=0,fa[i][0]=0;
        for(int i=1;i<=t;++i)fa[rt][i]=0;
        dfs(rt);
    }

    //求 lca 接口
    inline int lca(int u,int v){
        if(dep[u]>dep[v])swap(u,v);
        for(int i=t;~i;--i)if(dep[fa[v][i]]>=dep[u])v=fa[v][i];
        if(u==v)return u;
        for(int i=t;~i;--i)if(fa[u][i]!=fa[v][i])u=fa[u][i],v=fa[v][i];
        return fa[u][0];
    }
}
```

连通性

有向图强联通分量

- tarjan $O(n)$

```
namespace SCC{
    int dfn[N],clk,low[N];
    bool ins[N];int sta[N],tot; //栈  存正在构建的强连通块
    vector<int>scc[N];int c[N],cnt;//cnt 为强联通块数 scc[i] 存放每个块内点 c[i] 为原图每个结点属于的块
    void dfs(int x){
        dfn[x]=low[x]=(++clk);//low[] 在这里初始化
        ins[x]=1;sta[++tot]=x;
        go(i,x,to){
            if(!dfn[to]){dfs(to);low[x]=min(low[x],low[to]);}//走树边
            else if(ins[to])low[x]=min(low[x],dfn[to]);//走返祖边
        }
        if(dfn[x]==low[x]){//该结点为块的代表元
            ++cnt;int u;
            do{u=sta[tot--];ins[u]=0;c[u]=cnt;scc[cnt].push_back(u);}while(x!=u);
        }
    }
    inline void tarjan(int n){//n 是点数
        for(int i=1;i<=cnt;++i)scc[i].clear();//清除上次的 scc 防止被卡 MLE
        for(int i=1;i<=n;++i)dfn[i]=ins[i]=0;tot=clk=cnt=0;//全部变量初始化
        for(int i=1;i<=n;++i)if(!dfn[i])dfs(i);
        for(int i=1;i<=n;++i)c[i]+=n;//此行 (可以省略) 便于原图上加点建新图 加新点前要初始化 Head[]=0
    }
}
```

二分图匹配

匈牙利算法求二分图无权最大匹配

- 复杂度 O(nm)

```
#include<bits/stdc++.h>
using namespace std;
namespace Hungary{
    const int N=1000009;
    //图中应储存左部到右部的边  左点编号 1..n  右点编号 1..m
    int Head[N],Ver[N*2],Next[N*2],Gtot=1;  //注意边数开够
    inline void graphinit(int n){Gtot=1;for(int i=1;i<=n;++i)Head[i]=0;}
    inline void edge(int u,int v){Ver[++Gtot]=v,Next[Gtot]=Head[u],Head[u]=Gtot;}
    #define go(i,st,to) for(int i=Head[st],to=Ver[i];i;i=Next[i],to=Ver[i]) //st 是左点, to 是 st 能到达的右点

    int match[N],vis[N];//右点的的匹配点和访问标记

    bool dfs(int x){//x 是左点
        go(i,x,to)if(!vis[to]){
            vis[to]=1;
            if(!match[to]||dfs(match[to])){
                match[to]=x;return 1;
            }
        }
        return 0;
    }

    inline int ask(int n,int m){//左点数和右点数 返回最大匹配数
        for(int i=1;i<=m;++i)match[i]=0;int res=0;
        for(int i=1;i<=n;++i){
            for(int j=1;j<=m;++j)vis[j]=0;
            res+=dfs(i);
        }
        return res;
    }
}
```

KM 算法求二分图带权最大匹配

- 要求该图存在完美匹配该算法将最大化完美匹配的权值和

- 复杂度 O(n^3)
- naive 的写法复杂度为 O((n^2)m) 在完全图上会退化至 O(n^4) luogu P6577

```cpp
#include<bits/stdc++.h>
using namespace std;
namespace KM{
    const int N=509;
    const long long INFLL=0x3f3f3f3f3f3f3f3fll;//注意 INF 应足够大 至少远大于 n*n*Max{W}

    int n;//左右点数均为 n 标号均为 1..n
    int w[N][N];//邻接矩阵存边权
    bool r[N][N];//记录 i j 之间是否有边连接 完全图/非 0 权图则不必要

    inline void init(){//记得重写初始化
        int m;scanf("%d%d",&n,&m);
        for(int i=1;i<=m;++i){
            int u,v,wt;scanf("%d%d%d",&u,&v,&wt);
            w[u][v]=wt;r[u][v]=1;
        }
    }

    long long la[N],lb[N],upd[N];//左右顶标 每个右点对应的最小 delta 要开 longlong
    int last[N];//每个右点对应的回溯右点
    bool va[N],vb[N];
    int match[N];//每个右点对应的左匹配点
    bool dfs(int x,int fa){
        va[x]=1;
        for(int y=1;y<=n;++y)if(r[x][y]&&!vb[y]){
            const long long dt=la[x]+lb[y]-w[x][y];
            if(dt==0){//相等子图
                vb[y]=1;last[y]=fa;
                if(!match[y]||dfs(match[y],y)){
                    match[y]=x;
                    return 1;
                }
            }else if(upd[y]>dt){//下次 dfs 直接从最小 delta 处开始
                upd[y]=dt;
                last[y]=fa;//用 last 回溯该右点的上一个右点以更新增广路
            }
        }
        return 0;
    }

    inline void KM(){
        for(int i=1;i<=n;++i){//初始化顶标
            la[i]=-INFLL;lb[i]=0;
            for(int j=1;j<=n;++j)if(r[i][j])la[i]=max(la[i],(long long)w[i][j]);
        }
        for(int i=1;i<=n;++i){//尝试给每一个左点匹配上右点 匹配失败则扩展相等子图重试至成功
            memset(va,0,sizeof va);//注意复杂度
            memset(vb,0,sizeof vb);
            memset(last,0,sizeof last);
            memset(upd,0x3f,sizeof upd);
            int st=0;match[0]=i;//给起点 i 连一个虚右点 标号为 0
            //不断尝试将右点 st 从已有的匹配中解放 以获得增广路
            while(match[st]){//当 st 到达非匹配右点直接退出
                long long delta=INFLL;
                if(dfs(match[st],st))break;//st 的左点匹配到了新的右点则退出
                for(int j=1;j<=n;++j)
                    if(!vb[j]&&upd[j]<delta){//下次从最小的 delta 处开始 DFS
                        delta=upd[j];
                        st=j;
                    }
                for(int j=1;j<=n;++j){//将交错树上的左顶标加 delta 右顶标减 delta 使更多的边转为相等边
                    if(va[j])la[j]-=delta;
                    if(vb[j])lb[j]+=delta;
                        else upd[j]-=delta;//小问题: 这里在干啥 每次修改顶标后重新计算 upd 是否可行
                }
                vb[st]=1;
            }
            while(st){//更新增广路
```

```
                match[st]=match[last[st]];
                st=last[st];
            }
        }

        long long ans=0;
        for(int i=1;i<=n;++i)ans+=w[match[i]][i];
        printf("%lld\n",ans);
        for(int i=1;i<=n;++i)printf("%d%c",match[i]," \n"[i==n]);
    }
    //signed main(){init();KM();return 0;}
}
```

数据结构

手写整数哈希

- 防止自带哈希被卡 T

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

二维树状数组

- LOJ135 区间修改 + 区间查询单次操作复杂度 log^2 空间复杂度 N^2
- 对原数组差分 S(n,m) = ∑ ∑A_ij 用树状数组维护差分数组 A_ij 转化为单点修改
- 区间查询 SS(n,m) = ∑ ∑S(i,j) 推式子转化为单点查询问题
- `SS(n,m) = ∑∑∑∑A_ij = (n+1)*(m+1)*∑∑A_ij - (n+1)*∑∑jA_ij - (m+1)*∑∑iA_ij + ∑∑ijA_ij`

```
//省略了文件头
const int N=2051;
int n,m;
struct dat{
    LL c,cx,cy,cxy;
    void operator+=(const dat&a){c+=a.c;cx+=a.cx;cy+=a.cy;cxy+=a.cxy;}
}c[N][N];

inline void add(int a,int b,int k){
    const dat d={k,k*a,k*b,k*a*b};
    for(int i=a;i<=n;i+=(i&(-i)))
    for(int j=b;j<=m;j+=(j&(-j)))c[i][j]+=d;
}

inline LL f(int a,int b){
    dat r={0,0,0,0};
    for(int i=a;i>0;i-=(i&(-i)))
    for(int j=b;j>0;j-=(j&(-j)))r+=c[i][j];
    return (a+1)*(b+1)*r.c-(b+1)*r.cx-(a+1)*r.cy+r.cxy;
}

signed main(){
    read(n,m);
    int op;while(read(op))if(op==1){
        int a,b,c,d,k;read(a,b,c,d,k);
        add(a,b,k);add(c+1,b,-k);add(a,d+1,-k);add(c+1,d+1,k);
    }else{
```

```
        int a,b,c,d;read(a,b,c,d);
        printf("%lld\n",f(c,d)-f(a-1,d)-f(c,b-1)+f(a-1,b-1));
    }
    return 0;
}
```

堆式线段树

- 区间求和区间修改
- 空间估算
- 所有数组务必初始化

```
struct SegmentTree_Heap{
    #define TreeLen (N<<2)          //N
    #define lc(x)   ((x)<<1)
    #define rc(x)   ((x)<<1|1)
    #define sum(x)  (tr[x].sum)     //
    #define t(x)    (t[x])          //

    struct dat{
        LL sum;
        /* 这里写区间加法 */
        dat operator+(const dat&brother){
            dat result;
            result.sum=sum+brother.sum;
            return result;
        }
    }tr[TreeLen];
    LL t[TreeLen];  //lazy tag

    /* 单区间修改 */
    inline void change(const int&x,const int&l,const int&r,const LL&d){
        tr[x].sum=tr[x].sum+d*(r-l+1);
        t[x]=t[x]+d;
    }

    inline void pushup(int x){tr[x]=tr[lc(x)]+tr[rc(x)];}

    inline void pushdown(int x,const int&l,const int&r,const int&mid){
        if(t(x)){//注意区间修改细节
            change(lc(x),l,mid,t(x));
            change(rc(x),mid+1,r,t(x));
            t(x)=0;
        }
    }

    void build(int x,int l,int r){
        t(x)=0;   // 记得初始化!!!
        if(l==r){
            sum(x)=0;
            return;
        }
        int mid=(l+r)>>1;
        build(lc(x),l,mid);
        build(rc(x),mid+1,r);
        pushup(x);
    }

    void add(int x,int l,int r,const int&L,const int&R,const LL&d){
        if(L<=l&&r<=R){
            change(x,l,r,d);
            return;
        }
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        if(L<=mid)add(lc(x),l,mid,L,R,d);
        if(R>mid)add(rc(x),mid+1,r,L,R,d);
        pushup(x);
    }

    LL ask(int x,int l,int r,const int&L,const int&R){
```

```
        if(L<=l&&r<=R)return sum(x);
        int mid=(l+r)>>1;pushdown(x,l,r,mid);
        LL res=0;
        if(L<=mid)res=(res+ask(lc(x),l,mid,L,R));
        if(mid<R)res=(res+ask(rc(x),mid+1,r,L,R));
        return res;
    }
};
```

小根堆

```
namespace MyPQ{
    typedef int pqdat;    ////
    pqdat q[N];
    int tot;
    void up(int x){
        while(x>1)
        if(q[x]<q[x/2]){
            swap(q[x],q[x/2]);
            x/=2;
        }else return;
    }
    void down(int x){
        int ls=x*2;
        while(ls<=tot){
            if(ls<tot&&q[ls+1]<q[ls])++ls;
            if(q[ls]<q[x]){
                swap(q[x],q[ls]);x=ls;ls=x*2;
            }else return;
        }
    }
    void push(pqdat x){q[++tot]=x;up(tot);}
    pqdat top(){return q[1];}
    void pop(){if(!tot)return;q[1]=q[tot--];down(1);}
    void pop(int k){if(!tot)return;q[k]=q[tot--];up(k);down(k);}
}
```

Treap

```
#include<bits/stdc++.h>
using namespace std;
const int N=2000007,INF=(1ll<<30)+7;

//Treap 维护升序多重集
//支持操作: 数 <-> 排名 查询某数前驱后继
//操作数 x 可以不在集合中
//x 的排名: 集合中 <x 的数的个数 +1
//排名 x 的数: 集合中排名 <=x 的数中的最大数
//x 的前驱 比 x 小的最大数

struct treap{//所有点值不同 用副本数实现多重集
    int l,r;
    int v,w;//v 是数据 w 是维护堆的随机值
    int num,sz;//num 是该点副本数 sz 是该子树副本总数
}tr[N];int tot,rt;//tr[0] 始终全 0 使用范围 tr[1..n]
#define lc(x) tr[x].l
#define rc(x) tr[x].r
#define sz(x) tr[x].sz
#define num(x) tr[x].num
#define val(x) tr[x].v
#define wt(x) tr[x].w

inline int New(int x){
    val(++tot)=x; wt(tot)=rand();
    num(tot)=sz(tot)=1; return tot;
}

inline void upd(int p){sz(p)=sz(lc(p))+sz(rc(p))+num(p);}

inline void build(){//初始化 INF 和-INF 两个点
```

```
    srand(time(0));
    rt=1;tot=2;
    rc(1)=2;val(1)=-INF;wt(1)=rand();num(1)=1;sz(1)=2;
    val(2)=INF;wt(2)=rand();num(2)=1;sz(2)=1;
}

//调用时记得减一 askrk(rt,x)-1
int askrk(int p,int x){//当前子树中查询 x 的排名
    if(p==0)return 1;//说明某子树所有数均比 x 大
    if(x==val(p))return sz(lc(p))+1;
    return x<val(p)?askrk(lc(p),x):askrk(rc(p),x)+sz(lc(p))+num(p);
}

//调用时记得加一 kth(rt,++rank)
int kth(int p,int rk){//当前子树中查询排名 rk 的数
    if(p==0)return INF;//说明集合大小 <rk
    if(sz(lc(p))>=rk)return kth(lc(p),rk);
    rk-=sz(lc(p))+num(p);
    return (rk>0)?kth(rc(p),rk):val(p);
}

inline void zig(int &p){//与左子节点交换位置
    int q=lc(p);lc(p)=rc(q);rc(q)=p;
    upd(p);p=q;upd(p);
}

inline void zag(int &p){//与右子节点交换位置
    int q=rc(p);rc(p)=lc(q);lc(q)=p;
    upd(p);p=q;upd(p);
}

//insert(rt,x)
void insert(int &p,int x){//当前子树中插入 x
    if(p==0){p=New(x);return;}//x 首次插入
    if(x==val(p)){++num(p);++sz(p);return;}
    if(x<val(p)){
        insert(lc(p),x);
        if(wt(p)<wt(lc(p)))zig(p);//维护大根堆
    }else{
        insert(rc(p),x);
        if(wt(p)<wt(rc(p)))zag(p);//维护大根堆
    }
    upd(p);
}

//erase(rt,x)
void erase(int &p,int x){//当前子树中删除一个 x
    if(p==0)return;//已经无需删除
    if(val(p)==x){//如果找到了 x 的位置
        if(num(p)>1){//无需删点
            --num(p);--sz(p);return;//如果有多个 x 维护副本数即可
        }
        if(lc(p)||rc(p)){//该点不是叶子节点 则不断向下调整至叶子节点
            if(rc(p)==0||wt(lc(p))>wt(rc(p)))zig(p),erase(rc(p),x);//由于 rand() 的值域 & 大根堆的实现 故省略左子树为空的判断
            else zag(p),erase(lc(p),x);
            upd(p);
        }else p=0;//是叶子节点则直接删除
        return;
    }
    x<val(p)?erase(lc(p),x):erase(rc(p),x);upd(p);
}

int askpre(int x){
    int id=1;//-INF 若没有前驱则返回-INF
    //尝试自顶向下寻找 x 则 x 的前驱有两种情况
    //1) 未找到 x 或 x 没有左子树 则前驱在搜索路径上
    //2) 前驱是 x 的左子树中最大值 即 x 的左子树一直向右走
    int p=rt;
    while(p){
        if(x==val(p)){//找到 x
            if(lc(p)){p=lc(p);while(rc(p))p=rc(p);id=p;}
```

```
                break;
            }
            if(val(p)<x&&val(p)>val(id))id=p;//每经过一个点尝试更新前驱
            p=(val(p)>x?lc(p):rc(p));//找 x
        }
        return val(id);
}

int asknxt(int x){
    int id=2;//INF
    int p=rt;
    while(p){
        if(x==val(p)){
            if(rc(p)){p=rc(p);while(lc(p))p=lc(p);id=p;}
            break;
        }
        if(val(p)>x&&val(p)<val(id))id=p;
        p=(val(p)>x?lc(p):rc(p));
    }
    return val(id);
}
```

```
#include <bits/stdc++.h>
using namespace std;

struct MY_IO{
#define DEBUG 1///本地调试
#define MAXSIZE (1 << 20)
    inline bool isdigit(const char &x) { return x >= '0' && x <= '9'; }//字符集 看情况改
    inline bool blank(const char &c) { return c == ' ' || c == '\n' || c == '\r' || c == '\t'; }
#if DEBUG //
#else     //
    char buf[MAXSIZE + 3], *p1, *p2, pbuf[MAXSIZE + 3], *pp;
    MY_IO() : p1(buf), p2(buf), pp(pbuf) {}
    ~MY_IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }
#endif    //

    inline char gc(){
#if DEBUG                  //
        return getchar(); //
#else                      //
        if (p1 == p2)
            p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
        return p1 == p2 ? EOF : *p1++;
#endif                     //
    }

    inline void pc(const char &c){
#if DEBUG            //
        putchar(c); //
#else                //
        if (pp - pbuf == MAXSIZE)
            fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
        *pp++ = c;
#endif               //
    }

    template<typename T>inline bool read(T &x){
        x = 0; char c = gc(); int f = 1;
        while (!isdigit(c) && (c != '-') && (c != EOF)) c = gc();
        if (c == EOF) return 0;
        if (c == '-') f = -1, c = gc();
        while (isdigit(c)) {x = x * 10 + (c & 15); c = gc();}
        x *= f; return 1;
    }

    template<typename T, typename... Args>inline bool read(T &x, Args &...args){
        bool res = 1; res &= read(x); res &= read(args...); return res;
    }

    inline int gets(char *s){
        char c = gc(); while (blank(c) && c != EOF) c = gc();
```

```cpp
            if (c == EOF) return 0;
            int len = 0;
            while (!blank(c) && c != EOF) *s++ = c, c = gc(), ++len;
            *s = 0; return len;
        }

        inline void getc(char &c){for (c = gc(); blank(c) && c != EOF; c = gc());}

        /* 不能输出 (int)(-2^31)*/template<typename T>inline void write(T x){
            if (x < 0) x = -x, pc('-');
            static T sta[233];
            int top = 0;
            do{
                sta[top++] = x % 10, x /= 10;
            } while (x);
            while (top) pc(sta[--top] + '0');
        }

        template <typename T>inline void write(T x, const char &Lastchar){write(x); pc(Lastchar);}

        inline void puts(char *s){while ((*s) != 0)pc(*s++);}

        inline int getline(char *s){
            char c = gc();
            int len = 0;
            while (c != '\n' && c != EOF) *s++ = c, c = gc(), ++len;
            *s = 0; return len;
        }

        inline void putline(char *s){while ((*s) != 0)pc(*s++); pc('\n');}
}IO;
#define read IO.read
#define write IO.write
#define gc IO.gc
#define pc IO.pc
#define gets IO.gets
#define getc IO.getc
#define puts IO.puts
#define getl IO.getline
#define putl IO.putline
```