数学

数论

快(龟?)速幂

整数快速幂

```
ll ksm(ll a, ll b , ll m)//a的b次方对m取模
{
    ll ans = 1;
    for (; b; b >>= 1, a = a * a % m)
        if (b & 1)ans = ans * a % m;
    return ans;
}
```

int128版本(牛牛牛)

```
inline ll ksm(__int128 a, ll b, ll m)//a的b次方对m取模
{
    __int128 ans = 1;
    for (; b; b >>= 1, a = a * a % m)
        if (b & 1)ans = ans * a % m;
    return ans;
}
```

矩阵快速幂

```
vector<vector<int>> matric_imul(vector<vector<int>>a, vector<vector<int>>b, int n)
{
    vector<vector<int>>ret(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                ret[i][j] = (ret[i][j] + a[i][k] * b[k][j]) % mod;
    return ret;
}

vector<vector<int>> fast_matrix_pow(vector<vector<int>>& a, int b, int n)
{
    vector<vector<int>>ret(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        ret[i][i] = 1;
    for (; b; b >>= 1, a = matric_imul(a, a, n))
        if (b & 1) ret = matric_imul(a, ret, n);
    return ret;
}
```

龟速乘 &龟速幂

防爆ll(直接用int128好了，没啥用别看)

```
ll gsc(ll a, ll b, ll p)
{
    ll res = 0;
    while (b)
    {
        if (b & 1) res = (res + a) % p;
        a = (a << 1) % p;
        b >>= 1;
    }
    return res;
}
```

```
ll gsm(ll a, ll b , ll m)//a的b次方对m取模
{
    ll ans = 1;
    for (; b; b >>= 1, a = gsc(a,a,m))
        if (b & 1)ans = gsc(ans,a,m);
    return ans;
}
```

欧几里得

求gcd

辗转相除

```
ll my_gcd(ll a, ll b)
{
    while (b != 0)
    {
        ll tmp = a;
        a = b;
        b = tmp % b;
    }
    return a;
}
```

> 另外，对于 C++14，我们可以使用自带的 `__gcd(a,b)` 函数来求最大公约数。而对于 C++ 17，我们可以使用 `<numeric>` 头中的 `std::gcd` 与 `std::lcm` 来求最大公约数和最小公倍数。

lcm=连乘/gcd

拓展欧几里得

常用于解方程

```
ll ex_gcd(ll a, ll b, ll& x, ll& y)//返回值为gcd(a,b)
{
    if (!b)
```

```
    {
        x = 1;
        y = 0;
        return a;
    }
    ll d = ex_gcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - (a / b) * y;
    return d;
}
```

求逆元

费马小定理&欧拉定理

欧拉定理需要保证a,n互质

欧拉定理：

若正整数 a，n 互质，则 $a^{\varphi(n)} \equiv 1 \pmod n$ 其中 $\varphi(n)$ 是欧拉函数（1~n）与 n 互质的数。

费马小定理：

对于质数p，任意整数a，均满足：$a^p \equiv a \pmod p$

欧拉定理的推论：

若正整数a，n互质，那么对于任意正整数b，有 $a^b \equiv a^{b \bmod \varphi(n)} \pmod n$

特别的，如果a，mod不互质，且b>φ（n）时，$a^b \equiv a^{b \bmod \varphi(n) + \varphi(n)} \pmod n$。

```
ll qny(ll a, ll m)//求a在mod m下的乘法逆元
{
    return ksm(a, m - 2, m);
}
```

拓展o吉利的

```
void ex_gcd(ll a, ll b, ll& x, ll& y)
{//求逆元的话这样写: Exgcd(a,p,x,y);//x为结果，求1/a≡x(mod p)
    if (!b) x = 1, y = 0;
    else ex_gcd(b, a % b, y, x), y -= a / b * x;
}
```

线性求一串

```
int ny[3000005];//线性求逆元（求1~n的逆元）
ny[1] = 1;
for (int i = 2; i <= n; i++)
ny[i] = (p - p / i) * ny[p % i] % p;
```

中国剩余定理

crt

求一系列同余方程的最小解，满足m互质

```
ll crt(ll a[], ll b[], int n)//a[]为除数，b[]为余数
{
    ll lc = 1,x=0;//连乘
    for (int i = 0; i < n; ++i)  //算出它们累乘的结果
        lc *= a[i];

    for (int i = 0; i < n; ++i)
    {
        x = (x + (__int128) b[i] * (lc / a[i]) * qny(lc / a[i], a[i])) %
lc;//int128快速幂
    }
    return x;
}
```

ex_crt

同上，但m不互质（）

```
ll ex_crt(ll *cs,ll *ys,ll n)
{
    ll x, y;
    ll M = cs[0], ans = ys[0];
    for (int i = 1; i < n; i++)
    {
        ll a = M, b = cs[i], c = (ys[i] - ans % b + b) % b;
        ll gcd = ex_gcd(a, b, x, y), bg = b / gcd;
        if (c % gcd != 0)
            return -1;
        x = gsc(x, c / gcd, bg);//需要用龟速乘或者int128下的快速幂
        //不会爆的时候x = (x * c / gcd % bg + bg) % bg;
        ans += x * M;
        M *= bg;
        ans = (ans % M + M) % M;
    }
    return (ans % M + M) % M;
}
```

数论分块

整除分块

它的主要功能，就是将一个形如$\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$（当然不是只能在这个式子上应用，但它的确是最经常应用到整除分块的式子，没有之一）的式子的求值时间复杂度从$O(n)$降到$O(\sqrt{n})$。

```
ll fenkuai(ll n)
{
    ll res = 0;
    for (ll l = 1, r; l <= n; l = r + 1)
    {
        r = n / (n / l);
        res += n / l * (r - l + 1);
    }
    return res;
}
```

线性筛

```
bool Mark[N];
int prime[N];

void init()
{
    ll index = 0;
    for (ll i = 2; i <= N; i++)
    {
        if (!Mark[i])
        {
            prime[++index] = i;
        }
        for (ll j = 1; j <= index && prime[j] * i <= N; j++)
        {
            Mark[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                break;
            }
        }
    }
    return;
}
```

欧拉函数

求单个点 O(sqrt(n))

```
int ksm(int a, int b)
{
    int ans = 1;
    for (; b; b >>= 1, a = a * a)
        if (b & 1) ans = ans * a;
    return ans;
}

int Get(int n)//求单个数欧拉函数
{
    int ans = 1;
    for (int i = 2; i * i <= n; ++i)
    {
        int cnt = 0;
        while (n % i == 0) {n /= i; ++cnt;}
        if (cnt != 0) ans = ans * ksm(i, cnt - 1) * (i - 1);//基本性质 4
    }
    if (n != 1) ans = ans * (n - 1);//n^0=1
    return ans;
}
```

求一串 O(n)

使用线性筛:

```
bool Mark[N];//线性筛用的标记
int prime[N];//素数
ll phi[N];//欧拉函数值
void init()
{
    ll index = 0;
    phi[1] = 1;
    for (ll i = 2; i <= N; i++)
    {
        if (!Mark[i])
        {
            prime[++index] = i;
            phi[i] = i - 1;
        }
        for (ll j = 1; j <= index && prime[j] * i <= N; j++)
        {
            Mark[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = prime[j] * phi[i];
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
```

```
    return;
}
```

莫比乌斯函数

求一串法：线性筛

```
bool Mark[N];
int prime[N];
int miu[N];//莫比乌斯函数值
void init()
{
    ll index = 0;
    miu[1] = 1;
    for (ll i = 2; i <= N; i++)
    {
        if (!Mark[i])
        {
            prime[++index] = i;
            miu[i] = -1;
        }
        for (ll j = 1; j <= index && prime[j] * i <= N; j++)
        {
            Mark[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                miu[i * prime[j]] = 0;
                break;
            }
            miu[i * prime[j]] = -miu[i];
        }
    }
    return;
}
```

杜教筛

解释：在非线性时间内求出积性函数的前缀和

预处理理出前$n^{2/3}$的积性函数的前缀和，是最快的

再挂一次核（tao）心（lu）式，全都要靠它：

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

它的关键就是要找到合适的 g 使得这个东西可以快速地算。

理论知识大概就这么多，接下来看几个例子：

(1) μ 的前缀和

考虑到莫比乌斯函数的性质 $\mu * I = \epsilon$，自然想到取 $f = \mu, g = I, f * g = \epsilon$。

其中 I 的前缀和和 ϵ 的前缀和都弱到爆了。。

所以就轻松的解决了。

杜教筛代码：

```
inline ll GetSumu(int n) {
  if(n <= N) return sumu[n]; // sumu是提前筛好的前缀和
  if(Smu[n]) return Smu[n]; // 记忆化
  ll ret = 1ll; // 单位元的前缀和就是 1
  for(int l = 2, r; l <= n; l = r + 1) {
    r = n / (n / l); ret -= (r - l + 1) * GetSumu(n / l);
    // (r - l + 1) 就是 I 在 [l, r] 的和
  } return Smu[n] = ret; // 记忆化
}
```

(2) φ 的前缀和

考虑到 φ 的性质 $\varphi * I = id$，取 $f = \varphi, g = I, f * g = id$

$f * g$ 即 id 的前缀和为 $\frac{n*(n+1)}{2}$

杜教筛代码：

```
inline ll GetSphi(int n) {
  if(n <= N) return sump[n]; // 提前筛好的
  if(Sphi[n]) return Sphi[n]; // 记忆化
  ll ret = 1ll * n * (n + 1) / 2; // f * g = id 的前缀和
  for(int l = 2, r; l <= n; l = r + 1) {
    r = n / (n / l); ret -= (r - l + 1) * GetSphi(n / l);
    // 同上，因为两个的 g 都是 I
  } return Sphi[n] = ret; // 记忆化
}
```

例题：[P4213 【模板】杜教筛（Sum） - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)](luogu.com.cn)

```
#include<bits/stdc++.h>
using namespace std;
//==============================================================================
====//define宏定义区
#define ll long long
#define db double
#define pll pair<long long,long long>
#define PI acos(-1.0)
#define mod 1000000007
#define N 5000005
#define M 1000005
//==============================================================================
====//全局变量区
bool Mark[N];
int prime[N];
ll phi[N];
int miu[N];
```

```
unordered_map<int, int>ans_miu;
unordered_map<int, ll>ans_phi;
//======================================================================
====//函数区
void init()
{
    ll index = 0;
    phi[1] = 1;
    miu[1] = 1;
    for (ll i = 2; i <= N; i++)
    {
        if (!Mark[i])
        {
            prime[++index] = i;
            phi[i] = i - 1;
            miu[i] = -1;
        }
        for (ll j = 1; j <= index && prime[j] * i <= N; j++)
        {
            Mark[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = prime[j] * phi[i];
                miu[i * prime[j]] = 0;
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            miu[i * prime[j]] = -miu[i];
        }
    }
    for (ll i = 1; i <= N; i++)//求N内的前缀和
    {
        miu[i] += miu[i - 1];
        phi[i] += phi[i - 1];
    }
    return;
}
ll get_phi(ll x)
{
    if (x <= N) return phi[x];
    if (ans_phi[x]) return ans_phi[x];
    ll ans = ((1ll + x) * x) / 2ll;
    for (ll l = 2, r; l <= x; l = r + 1)
    {
        r = x / (x / l);
        ans -= 1ll * (r - l + 1) * get_phi(x / l);
    }
    return ans_phi[x] = ans;
}
int get_miu(int x)
{
    if (x <= N) return miu[x];
    if (ans_miu[x]) return ans_miu[x];
    int ans = 1;
    for (ll l = 2, r; l <= x; l = r + 1)
```

```
    {
        r = x / (x / l);
        ans -= (r - l + 1) * get_miu(x / l);
    }
    return ans_miu[x] = ans;
}

void solve()
{
    ll x;
    scanf("%lld", &x);
    printf("%lld %d\n", get_phi(x), get_miu(x));
}
//==================================================================//主函数配置cin&cout加速与测试集区
int main()
{
    //std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);
    ll t = 1;
    scanf("%lld", &t);
    //cin>>t;
    init();
    while (t--)
        solve();
    return 0;
}
```

组合数学

01组合数

用于MOD较大，

一般 0 <= n,m <= 1e6 ，MOD=1e9+7

```
const ll MOD = 1000000007;
const ll MAXN = (ll)1e5;

ll jc[MAXN+3];
ll jcny[MAXN+3];

void Cinit()
{
    jc[0] = 1, jcny[0] = 1;
    for (int i = 1; i <= MAXN; i++)
        jc[i] = jc[i - 1] * i % MOD;
    jcny[MAXN] =qny(jc[MAXN], MOD);
    for (int i = MAXN - 1; i >= 1; i--)
        jcny[i] = jcny[i + 1] * (i + 1) % MOD;
}

ll C(ll m, ll n, ll p)//正常情况下m>=n
{
```

```
    return m < n ? 0 : jc[m] * jcny[n] % p * jcny[m - n] % p;
}
```

Lucas 定理

卢卡斯定理（Lucas's theorem）：

> 对于非负整数 m, n 和质数 p ， $C_m^n \equiv \prod_{i=0}^{k} C_{m_i}^{n_i} \pmod p$ ，其中 $m = m_k p^k + \cdots + m_1 p + m_0$ 、 $n = n_k p^k + \cdots + n_1 p + n_0$ 是 m 和 n 的 p 进制展开。

但其实，我们一般使用的是这个可以与之互推的式子：

> $C_m^n \equiv C_{m \bmod p}^{n \bmod p} \cdot C_{\lfloor m/p \rfloor}^{\lfloor n/p \rfloor} \pmod p$

当 $m < n$ 时，规定 $C_m^n = 0$ （待会儿会将这个规定的意义）。

就像辗转相除法那样，可以利用这个式子递归求解，递归出口是 $n = 0$ 。其实这篇文章只需要这个好记的公式就够了，你甚至可以马上写出卢卡斯定理的板子：

```
// 需要先预处理出fact[]，即阶乘
inline ll C(ll m, ll n, ll p)
{
    return m < n ? 0 : fact[m] * inv(fact[n], p) % p * inv(fact[m - n], p) % p;
}
inline ll lucas(ll m, ll n, ll p)
{
    return n == 0 ? 1 % p : lucas(m / p, n / p, p) * C(m % p, n % p, p) % p;
}
```

网上说卢卡斯定理的复杂度是 $O(p \log_p m)$ ，但如果阶乘和逆元都采取递推的方法预处理，（只需要预处理 p 以内的），每次调用 `c()` 函数应该都是 $O(1)$ 的，一共要调用 $\log_p m$ 次，那么复杂度应该是 $O(p + \log_p m)$ 才对。洛谷上这道模板题的范围才给到 10^5 ，屈才了。

用于p<1e6   0<=n,m<=1e9

(需要O(p)预处理)

```
const ll MOD = 1000000007;
const ll MAXN = (ll)1e5;

ll jc[MAXN+3];
ll jcny[MAXN+3];

void Cinit(ll mod) //处理[1,mod-1]
{
    jc[0] = 1, jcny[0] = 1;
    for (int i = 1; i <= mod-1; i++)
        jc[i] = jc[i - 1] * i % mod;
    jcny[mod-1] =qny(jc[mod-1], mod);
    for (int i = mod-2; i >= 1; i--)
        jcny[i] = jcny[i + 1] * (i + 1) % mod;
}
```

```cpp
ll C(ll m, ll n, ll p)//正常情况下m>=n
{
    return m < n ? 0 : jc[m] * jcny[n] % p * jcny[m - n] % p;
}

ll lucas(ll m, ll n, ll p)
{
    return n == 0 ? 1 : lucas(m / p, n / p, p) * C(m % p, n % p, p) % p;
}
```

线性代数

线性基

线性基是一种擅长处理异或问题的数据结构.设值域为$[1,N]$，就可以用一个长度为$\lceil \log_2 N \rceil$的数组来描述一个线性基。特别地，线性基第i位上的数二进制下最高位也为第i位。

一个线性基满足，对于它所表示的所有数的集合S，S中任意多个数异或所得的结果均能表示为线性基中的元素互相异或的结果，即意，线性基能使用异或运算来表示原数集使用异或运算能表示的所有数。运用这个性质，我们可以极大地缩小异或操作所需的查询次数。

```cpp
ll xxj[N + 5];
bool check0 = false;
void xxj_insert(ll x) {
    for (int i = N; i>=0; i--)
        if (x & (1ll << i))
            if (!xxj[i]) { xxj[i] = x; return; }
            else x ^= xxj[i];
    check0 = true;
}
bool xxj_check(ll x) {
    for (int i = N; i>=0; i--)
        if (x & (1ll << i))
            if (!xxj[i])return false;
            else x ^= xxj[i];
    return true;
}
ll xxj_max(ll res = 0) {
    for (int i = N; i>=0; i--)
        res = max(res, res ^ xxj[i]);
    return res;
}
ll xxj_min() {
    if (check0)return 0;
    for (int i = 0; i <= N; i++)
        if (xxj[i])return xxj[i];
}
```

杂项数学

FFT快速傅里叶变换

O(nlogn)

```
//n次多项式(系数下标0~n)*m多项式(系数下标0~m)
const ll N = 1e7 + 10;

complex<db>a[N], b[N];
int n, m;
int l, r[N];
int limit = 1;

void fft(complex<db>* A, int type)
{
    for (int i = 0; i < limit; i++)
        if (i < r[i]) swap(A[i], A[r[i]]); //求出要迭代的序列
    for (int mid = 1; mid < limit; mid <<= 1) { //待合并区间的长度的一半
        complex Wn(cos(PI / mid), type * sin(PI / mid)); //单位根
        for (int R = mid << 1, j = 0; j < limit; j += R) { //R是区间的长度，j表示前
已经到哪个位置了
            complex w((db)1, (db)0); //幂
            for (int k = 0; k < mid; k++, w = w * Wn) { //枚举左半部分
                complex x = A[j + k], y = w * A[j + mid + k]; //蝴蝶效应
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}

void solve()
{
    scanf("%d%d", &n, &m);
    int tmp;
    for (int i = 0; i <= n; i++)
    {
        scanf("%d", &tmp);
        a[i] = tmp;
    }
    for (int i = 0; i <= m; i++)
    {
        scanf("%d", &tmp);
        b[i] = tmp;
    }
    while (limit <= n + m)
        limit <<= 1, l++;
    for (int i = 0; i < limit; i++)
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    fft(a, 1);
    fft(b, 1);
    for (int i = 0; i <= limit; i++)
        a[i] = a[i] * b[i];
    fft(a, -1);
    for (int i = 0; i <= n + m; i++)
        printf("%d ", (int)(a[i].real() / limit + 0.5));
```

```
}
```

计算几何

有用的函数与约定

```
#define pdd pair<db,db>
const db EPS = 1e-7;
//================================================================================
============================
db cross(pdd vec1, pdd vec2)//cross(vec1,vec2)>0表示a->b左转
{
    return vec1.first * vec2.second - vec2.first * vec1.second;
}
db dot(pdd vec1, pdd vec2)
{
    return vec1.first * vec2.first + vec1.second * vec2.second;
}
db dis(pdd node1, pdd node2)//求(x1,y1),(x2,y2)距离
{
    return sqrt(pow(node2.first - node1.first,2) + pow(node2.second-
node1.second, 2));
}
db S2(pdd node1, pdd node2, pdd node3)//求三角形面积x2
{
    pdd vec1 = { node3.first - node1.first,node3.second - node1.second };
    pdd vec2 = { node3.first - node2.first,node3.second - node2.second };
    return abs(vec1.first * vec2.second - vec1.second * vec2.first);
}
pdd vec_build(pdd node1, pdd node2)//返回node1->node2的向量
{
    return { node2.first - node1.first,node2.second - node1.second };
}
pdd get_node(pair<pdd, pdd> a, pair<pdd, pdd> b)//求两直线交点（定比分点）
{
    db S1 = cross(vec_build(a.first, b.second), vec_build(a.first,a.second));
    db S2 = cross(vec_build(a.first, b.first), vec_build(a.first, a.second));
    return { (S1 * b.first.first - S2 * b.second.first) / (S1 - S2), (S1 *
b.first.second - S2 * b.second.second) / (S1 - S2) };
}
```

二维凸包

Andrew算法

```
pdd stk[200010];
int top = 0;

db cross(pdd vec1, pdd vec2)//cross(vec1,vec2)>0表示a->b左转
{
```

```
    return vec1.first * vec2.second - vec2.first * vec1.second;
}

pdd vec_build(pdd node1, pdd node2)//返回node1->node2的向量
{
    return { node2.first - node1.first,node2.second - node1.second };
}

void qiutubao(vector<pdd>&node)//求凸包
{
    sort(node.begin(), node.end());
    stk[++top] = node[0];
    stk[++top] = node[1];
    for (int i = 2; i < node.size(); i++)
    {
        while (top > 1 && cross(vec_build(stk[top - 1], stk[top]),
vec_build(stk[top], node[i])) <= 0)
            top--;
        stk[++top] = node[i];
    }
    stk[++top] = node[node.size() - 2];
    for (int i = node.size() - 3; i >= 0; i--)
    {
        while (top > 1 && cross(vec_build(stk[top-1],stk[top]),
vec_build(stk[top], node[i])) <= 0 && stk[top] != node[node.size() - 1])
            top--;
        stk[++top] = node[i];
    }
}
```

旋转卡壳

就是求凸包的直径

```
extern pdd stk[200010];
extern int top;//已经求好的凸包(1~n+1)

db dis(pdd node1, pdd node2)//求(x1,y1),(x2,y2)距离
{
    return sqrt(pow(node2.first - node1.first,2) + pow(node2.second-
node1.second, 2));
}

db S2(pdd node1, pdd node2, pdd node3)//求三角形面积x2
{
    pdd vec1 = { node3.first - node1.first,node3.second - node1.second };
    pdd vec2 = { node3.first - node2.first,node3.second - node2.second };
    return abs(vec1.first * vec2.second - vec1.second * vec2.first);
}

db tb_d()//求凸包直径
{
    if (top < 4)
```

```
        return dis(stk[1], stk[2]);
    db ret = -1;
    for (int j = 3, i = 1; i < top; i++)
    {
        //ret = max(ret, dis(stk[i], stk[i + 1]));//没啥用
        while (S2(stk[i], stk[i + 1], stk[j]) / dis(stk[i], stk[i + 1]) <
S2(stk[i], stk[i + 1], stk[j % (top - 1) + 1]) / dis(stk[i], stk[i + 1]))
            j = j % (top - 1) + 1;
        ret= max(ret, max(dis(stk[i],stk[j]),dis(stk[i+1],stk[j])));
    }
    return ret;
}
```

升级版，求凸包的最小矩形

```
//EPS=1e-8
extern pdd stk[200010];
extern int top;//已经求好的凸包(1~n+1)
//================================================================================
========================//
db dot(pdd vec1, pdd vec2)//向量点积
{
    return vec1.first * vec2.first + vec1.second * vec2.second;
}

vector<pdd> tb_jx()//求凸包最小矩形
{
    vector<pdd>ret(5);
    db minS = 1.7e308;
    for (int j = 3, i = 1,r=2,l=2; i < top; i++)
    {
        while (S2(stk[i], stk[i + 1], stk[j]) / dis(stk[i], stk[i + 1]) <
S2(stk[i], stk[i + 1], stk[j % (top - 1) + 1]) / dis(stk[i], stk[i + 1]))
            j = j % (top - 1) + 1;
        while (dot(vec_build(stk[i], stk[i + 1]), vec_build(stk[r], stk[r + 1]))
> 0)
            r = r % (top - 1) + 1;
        if (i == 1)l = j;
        while (dot(vec_build(stk[i], stk[i + 1]), vec_build(stk[l], stk[l + 1]))
< 0)
            l = l % (top - 1) + 1;
        db d1 = dis(stk[i], stk[i + 1]);
        db h = S2(stk[i], stk[i + 1], stk[j]) / dis(stk[i], stk[i + 1]);
        db dr = dot(vec_build(stk[i], stk[i + 1]), vec_build(stk[i + 1],
stk[r])) / d1;
        db dl = dot(vec_build(stk[i + 1], stk[i]), vec_build(stk[i], stk[l])) /
d1;
        db d = d1 + dr + dl;
        db S = d * h;
        if (S<=(minS+EPS))
        {
            minS = S;
            vector<pdd>tmp(5);
            tmp[0] = { S,0 };
```

```
            tmp[1] = { stk[i + 1].first + dr / d1 * (stk[i + 1].first -
stk[i].first),stk[i + 1].second + dr / d1 * (stk[i + 1].second - stk[i].second)
};
            tmp[2] = { tmp[1].first + h / d1 * (stk[i].second -
stk[i+1].second),tmp[1].second + h / d1 * (stk[i+1].first-stk[i].first)};
            tmp[4] = { stk[i].first + dl / d1 * (stk[i].first- stk[i + 1].first
),stk[i].second + dl / d1 * (  stk[i].second-stk[i + 1].second) };
            tmp[3] = { tmp[4].first + h / d1 * (stk[i].second-stk[i +
1].second),tmp[4].second + h / d1 * ( stk[i + 1].first -stk[i].first) };
            ret = tmp;
        }
    }
    return ret;
}
//注意可能输出-0.00000,判一下。
```

半平面交

求多个半平面的交集，S&I算法（代码极为丑陋）

```
const ll N = 10;
const ll M = 50;
const db EPS = 1e-7;
//==============================================================================
============================
pair<pdd, pdd>deq[N * M + 5];//模拟双端队列
ll l = 1;
ll r = 0;
//==============================================================================
============================
bool cmp(pair<pdd, pdd> a, pair<pdd, pdd> b)
{
    db k1 = atan2(a.second.second - a.first.second, a.second.first -
a.first.first);
    db k2 = atan2(b.second.second - b.first.second, b.second.first -
b.first.first);
    if (abs(k1 - k2) > EPS)
        return k1 < k2;
    return cross(vec_build(b.first, b.second), vec_build(b.first, a.first))>0;
}

pdd get_node(pair<pdd, pdd> a, pair<pdd, pdd> b)//定比分点
{
    db S1 = cross(vec_build(a.first, b.second), vec_build(a.first,a.second));
    db S2 = cross(vec_build(a.first, b.first), vec_build(a.first, a.second));
    return { (S1 * b.first.first - S2 * b.second.first) / (S1 - S2), (S1 *
b.first.second - S2 * b.second.second) / (S1 - S2) };
}

db SandI(vector<pair<pdd, pdd>>vecs)
{
    for (int i = 0; i < vecs.size(); i++)
    {
        if (r - l + 1 < 2)
        {
```

```
            r++;
            deq[r] = vecs[i];
            if (r - l + 1 >= 2)
            {
                if (atan2(deq[r].second.second - deq[r].first.second, deq[r].second.first - deq[r].first.first) - atan2(deq[r - 1].second.second - deq[r - 1].first.second, deq[r - 1].second.first - deq[r - 1].first.first) <= EPS)
                {
                    r--;
                    if (cross(vec_build(deq[r+1].first, deq[r+1].second), vec_build(deq[r+1].first, deq[r].first)) < 0)
                        deq[r] = deq[r + 1];
                }
            }
        }
        else
        {
            while (r - l + 1 >= 2 && cross(vec_build(vecs[i].first, get_node(deq[r], deq[r - 1])), vec_build(vecs[i].first, vecs[i].second)) > -EPS)
                r--;
            while (r - l + 1 >= 2 && cross(vec_build(vecs[i].first, get_node(deq[l], deq[l + 1])), vec_build(vecs[i].first, vecs[i].second)) > -EPS)
                l++;
            r++;
            deq[r] = vecs[i];
            if (atan2(deq[r].second.second - deq[r].first.second, deq[r].second.first - deq[r].first.first) - atan2(deq[r - 1].second.second - deq[r - 1].first.second, deq[r - 1].second.first - deq[r - 1].first.first) <= EPS)
            {
                r--;
                if (cross(vec_build(deq[r+1].first, deq[r+1].second), vec_build(deq[r+1].first, deq[r].first)) < 0)
                    deq[r] = deq[r + 1];
            }
        }
    }
    while (r - l + 1 >= 3 && cross(vec_build(deq[l].first, get_node(deq[r], deq[r - 1])), vec_build(deq[l].first, deq[l].second)) > -EPS)
    {
        r--;
    }
    db S = 0;
    pdd node0 = get_node(deq[l], deq[r]);
    for (int i = l; i < r-1; i++)
    {
        pdd node1 = get_node(deq[i], deq[i + 1]);
        pdd node2 = get_node(deq[i+1], deq[i + 2]);
        S += S2(node0, node1, node2) / 2;
    }
    return S;
}

void solve()
```

```
{
    ll n;
    scanf("%lld", &n);
    vector<pair<pdd, pdd>>vecs;
    for (int i = 0; i < n; i++)
    {
        ll nodes;
        scanf("%lld", &nodes);
        pdd node_bp;
        pdd node_tmp;
        scanf("%lf%lf", &node_tmp.first, &node_tmp.second);
        node_bp = node_tmp;
        pdd node_i;
        for (int i = 1; i < nodes; i++)
        {
            scanf("%lf%lf", &node_i.first, &node_i.second);
            vecs.push_back({ node_tmp,node_i });
            node_tmp = node_i;
        }
        vecs.push_back({ node_tmp,node_bp });
    }
    sort(vecs.begin(), vecs.end(), cmp);
    printf("%.3lf\n",SandI(vecs));
}
```

数据结构

线段树

O(logn)区间修改，区间查询

```
const ll N = (ll)1e5;
ll a[N], ans[N << 2+4], tag_mul[N << 2+4],tag_add[N<<2+4];
ll mod=100000007;

inline ll ls(ll x)
{
    return x << 1;
}
inline ll rs(ll x)
{
    return x << 1 | 1;
}

inline void push_up(ll p)//传递上一层
{
    ans[p] = (ans[ls(p)] + ans[rs(p)])%mod;
}
void build(ll p, ll l, ll r)//建树
```

```
{
    tag_mul[p] = 1;
    tag_add[p] = 0;
    if (l == r) { ans[p] = a[l]; return; }
    ll mid = (l + r) >> 1;
    build(ls(p), l, mid);
    build(rs(p), mid + 1, r);
    push_up(p);
}
inline void f(ll p, ll l, ll r, ll num, ll op)//修改单点
{
    if (op == 0)//0为加,1为乘
    {
        tag_add[p] = (tag_add[p] + num)%mod;
        ans[p] = (ans[p]+num* (r - l + 1)%mod)%mod;
    }
    else
    {
        tag_mul[p] = tag_mul[p] * num % mod;
        tag_add[p] = tag_add[p] * num % mod;
        ans[p] = ans[p]*num%mod;
    }
}
inline void push_down(ll p, ll l, ll r)//传递下一层
{
    ll mid = (l + r) >> 1;
    f(ls(p), l, mid, tag_mul[p],1);
    f(ls(p), l, mid, tag_add[p], 0);
    f(rs(p), mid + 1, r, tag_mul[p], 1);
    f(rs(p), mid + 1, r, tag_add[p], 0);
    tag_mul[p] = 1;
    tag_add[p] = 0;
}
inline void update(ll nl, ll nr, ll l, ll r, ll p, ll k,ll op)//修改区间
{
    if (nl <= l && r <= nr)
    {
        if (op == 0)//我要加
        {
            tag_add[p] = (tag_add[p] + k) % mod;
            ans[p] = (ans[p] + k * (r - l + 1) % mod) % mod;
        }
        else
        {
            tag_mul[p] = tag_mul[p] * k % mod;
            tag_add[p] = tag_add[p] * k % mod;
            ans[p] = ans[p] * k % mod;
        }
        return;
    }
    push_down(p, l, r);
    ll mid = (l + r) >> 1;
    if (nl <= mid)update(nl, nr, l, mid, ls(p), k,op);
    if (nr > mid) update(nl, nr, mid + 1, r, rs(p), k,op);
    push_up(p);
```

```
}
ll query(ll q_x, ll q_y, ll l, ll r, ll p)//查询区间
{
    ll res = 0;
    if (q_x <= l && r <= q_y)return ans[p];
    ll mid = (l + r) >> 1;
    push_down(p, l, r);
    if (q_x <= mid)res += query(q_x, q_y, l, mid, ls(p));
    if (q_y > mid) res += query(q_x, q_y, mid + 1, r, rs(p));
    return res%mod;
}
```

图论

字符串

动态规划

其他技巧

快读

```
inline ll read()
{
    ll x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch>'9')
    {
        if (ch == '-')f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
        x = x * 10 + ch - '0', ch = getchar();
    return x * f;
}
```