起手板子

```
#pragma GCC optimize(2)

#include <bits/stdc++.h>

#define fi first
#define se second
#define mkp(x, y) make_pair((x), (y))
#define all(x) (x).begin(), (x).end()

using namespace std;

typedef long long LL;
typedef pair<int, int> PII;

void solve() {
    //
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout << fixed;  // << setprecision(20); // double
    // freopen("i.txt", "r", stdin);
    // freopen("o.txt", "w", stdout);
    // time_t t1 = clock();
    int Tcase = 1;
    // cin >> Tcase; // scanf("%d", &Tcase);
    while (Tcase--)
        solve();
    // cout << "time: " << 1000.0 * ((clock() - t1) / CLOCKS_PER_SEC) << "ms\n";
    return 0;
}
```

图论

Dinic

```
int idx, h[N], ne[M], ver[M], e[M];
int n, m, S, T;
int d[N], cur[N];

void add(int a, int b, int c) {
    ver[idx] = b, e[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}
bool bfs() {
    queue<int> q;
    memset(d, -1, sizeof d);
    q.push(S);
    d[S] = 0, cur[S] = h[S];
```

```
    while(q.size()) {
        int t = q.front(); q.pop();
        for(int i = h[t]; ~i; i = ne[i]) {
            int v = ver[i];
            if(d[v] == -1 && e[i]) {
                d[v] = d[t]+1;
                cur[v] = h[v];
                if(v == T) return true;
                q.push(v);
            }
        }
    }
    return false;
}
int update(int u, int limit) {
    if(u == T) return limit;
    int flow = 0;
    for(int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int v = ver[i];
        if(d[v] == d[u]+1 && e[i]) {
            int t = update(v, min(e[i], limit-flow));
            if(!t) d[v] = -1;
            flow += t;
            e[i] -= t;
            e[i^1] += t;
        }
    }
    return flow;
}
int dinic() {
    int res = 0, flow;
    while(bfs())
        while(flow = update(S, INF))
            res += flow;
    return res;
}
/*
******临时加边/容量***************
int tx = h[x], ty = h[y];
for(int z = 0; z < idx; z += 2)
    e[z] += e[z^1], e[z^1] = 0;
add(x, y, C), add(y, x, 0);
LL tres = dinic();
idx -= 2;
h[x] = tx, h[y] = ty;
******也可以先记录下来，然后记得用bool判定只加了一次
bool ok = false;
for(int z = 0; z < idx; z += 2) { // .....
    e[z] = rec[z] + (!ok && ver[z] == y && ver[z^1] == x ? C : 0);
    e[z^1] = 0;
    if(ver[z] == y && ver[z^1] == x) ok = true;
}
*/
```

数据结构

DSU

```
struct DSU {
    vector<int> fa, siz;

    void Init(int n) {
        fa.resize(n+1);
        siz.resize(n+1);
        for(int i = 1; i <= n; i ++)
            fa[i] = i, siz[i] = 1;
    }
    int GetFa(int x) {
        if(x == fa[x]) return x;
        return fa[x] = GetFa(fa[x]);
    }
    bool Same(int x, int y) {
        return GetFa(x) == GetFa(y);
    }
    bool Merge(int x, int y) {
        x = GetFa(x), y = GetFa(y);
        if(x == y) return false;
        siz[x] += siz[y];
        fa[y] = x;
        return true;
    }
    int Size(int x) {
        return siz[GetFa(x)];
    }
}dsu;
```

Fenwick Tree

```
template <typename T>
struct FenwickTree {
    int n;
    vector<T> a;
    void Init(int n) {
        this->n = n;
        a.assign(n+1, 0);
    }
    int Lowbit(int x) {
        return x & -x;
    }
    void Add(int x, T v) {
        for( ; x <= n; x += Lowbit(x))
            a[x] += v;
    }
    T Sum(int x) {
        auto ans = T();
        for( ; x; x -= Lowbit(x))
            ans += a[x];
        return ans;
    }
```

```
    T RangeSum(int l, int r) {
        return Sum(r) - Sum(l-1);
    }
};
FenwickTree<int> fen; // int / LL
```

线段树 & 倍增LCA

```
void BuildPhiTree(int n) {
    dep[1] = 1;
    for(int k = 0; k < 6; k ++) fa[1][k] = 1;
    for(int i = 2; i <= n; i ++) {
        dep[i] = dep[phi[i]] + 1;
        fa[i][0] = phi[i];
        for(int k = 1; k < 6; k ++)
            fa[i][k] = fa[fa[i][k-1]][k-1];
    }
}
int LCA(int x, int y) {
    if(dep[x] < dep[y]) swap(x, y);
    for(int k = 5; k >= 0; k --)
        if(dep[fa[x][k]] >= dep[y])
            x = fa[x][k];
    if(x == y) return x;
    for(int k = 5; k >= 0; k --)
        if(fa[x][k] != fa[y][k])
            x = fa[x][k], y = fa[y][k];
    return fa[x][0];
}
struct Info { // int / LL
    int l, r;
    int lca, ans;
    bool root;
};
struct SegTree {
    Info tr[N<<2];
    #define ls(u) (u<<1)
    #define rs(u) (u<<1|1)

    void PushUp(int u) {
        tr[u].lca = LCA(tr[ls(u)].lca, tr[rs(u)].lca);
        tr[u].ans = tr[ls(u)].ans + tr[rs(u)].ans +
                    (tr[ls(u)].r-tr[ls(u)].l+1) * (dep[tr[ls(u)].lca]-
dep[tr[u].lca]) +
                    (tr[rs(u)].r-tr[rs(u)].l+1) * (dep[tr[rs(u)].lca]-
dep[tr[u].lca]);
        tr[u].root = tr[ls(u)].root && tr[rs(u)].root;
    }
    void Build(int u, int l, int r) {
        if(l == r) {
            tr[u] = {l, r, a[r], 0, a[r] == 1};
            return;
        }
        tr[u] = {l, r};
        int mid = l+r >> 1;
        Build(ls(u), l, mid), Build(rs(u), mid+1, r);
        PushUp(u);
```

```
    }
    void Modify(int u, int l, int r) {
        if(tr[u].root) return;
        if(tr[u].l == tr[u].r) {
            tr[u].lca = fa[tr[u].lca][0];
            tr[u].root = tr[u].lca == 1;
            return;
        }
        int mid = tr[u].l+tr[u].r >> 1;
        if(l <= mid) Modify(ls(u), l, r);
        if(r > mid) Modify(rs(u), l, r);
        PushUp(u);
    }
    int QueryLCA(int u, int l, int r) {
        if(l <= tr[u].l && tr[u].r <= r) return tr[u].lca;
        int mid = tr[u].l+tr[u].r >> 1;
        int lca1 = 0, lca2 = 0;
        if(l <= mid) lca1 = QueryLCA(ls(u), l, r);
        if(r > mid) lca2 = QueryLCA(rs(u), l, r);
        // cout << tr[u].l << ' ' << tr[u].r << " : " << lca1 << ' ' << lca2 <<
' ' << LCA(lca1, lca2) << '\n';
        if(!lca1) return lca2;
        if(!lca2) return lca1;
        return LCA(lca1, lca2);
    }
    int QueryAns(int u, int l, int r, int lca) {
        if(l <= tr[u].l && tr[u].r <= r) return tr[u].ans + (tr[u].r-tr[u].l+1)*
(dep[tr[u].lca]-dep[lca]);
        int mid = tr[u].l+tr[u].r >> 1;
        int ans = 0;
        if(l <= mid) ans += QueryAns(ls(u), l, r, lca);
        if(r > mid) ans += QueryAns(rs(u), l, r, lca);
        return ans;
    }
    #undef ls
    #undef rs
}sgt;
```

树链剖分

```
struct HeavyLightDecompotion {
    int n, tms;
    vector<int> siz, top, dep, fa, dfni, dfno, rnk;
    vector<vector<int>> e;

    void Init(int n) {
        this->n = n;
        siz.resize(n+1);
        top.resize(n+1);
        dep.resize(n+1);
        fa.resize(n+1);
        dfni.resize(n+1);
        dfno.resize(n+1);
        rnk.resize(n+1);
        tms = 0;
        e.assign(n+1, {});
    }
```

```
    void AddEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    void Build(int root = 1) {
        top[root] = root;
        dep[root] = 1;
        fa[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (fa[u] != -1) {
            e[u].erase(find(all(e[u]), fa[u]));
        }
        siz[u] = 1;
        for (auto &v : e[u]) {
            fa[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[e[u][0]]) {
                swap(v, e[u][0]);
            }
        }
    }
    void dfs2(int u) {
        dfni[u] = ++ tms;
        rnk[dfni[u]] = u;
        for (auto v : e[u]) {
            top[v] = (v == e[u][0]) ? top[u] : v;
            dfs2(v);
        }
        dfno[u] = tms+1;
    }
    int LCA(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = fa[top[u]];
            }
            else {
                v = fa[top[v]];
            }
        }
        return (dep[u] < dep[v]) ? u : v;
    }
    int GetDist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[LCA(u, v)];
    }
    int Jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }
        int d = dep[u] - k;
        while (dep[top[u]] > d) {
            u = fa[top[u]];
        }
        return rnk[dfni[u] - dep[u] + d];
```

```
    }
}hld;
```

```
// 路径查询和维护
/**********************************
void add_x2y(int x, int y, int c) {
    while(top[x] != top[y]) {
        if(d[top[x]] < d[top[y]]) swap(x, y);
        modify/query/ ... (1, dfn[top[x]], dfn[x], c);
        x = fa[top[x]];
    }
    if(d[x] < d[y]) swap(x, y);
    modify/query/ ... (1, dfn[y], dfn[x], c);
}
**********************************/
```

莫队

```
struct Rec {
    int l, r, qid;
};

LL tans;
int bNum, bSize;
vector<int> a, belong, cnt;
vector<Rec> query;

LL f(int c) {
    return 1LL*c*(c-1)*(c-2)/6;
}
void add(int x) {
    auto &c = cnt[a[x]];
    tans -= f(c);
    c ++;
    tans += f(c);
}
void del(int x) {
    auto &c = cnt[a[x]];
    tans -= f(c);
    c --;
    tans += f(c);
}
void solve() {
    int n, m;
    cin >> n >> m;
    a.resize(n+1), belong.resize(n+1), query.resize(m+1);
    // bSize在 m < n时，取n/sqrt(m)，但在 m > n时，这样取可能会下取整导致bSize = 0，导致
RE。
    if(n > m) bSize = n/sqrt(m);
    else bSize = sqrt(n);
    bNum = (n-1)/bSize + 1;
    int mx = 0;
    for(int i = 1; i <= n; i ++) {
        cin >> a[i];
        mx = max(mx, a[i]);
        belong[i] = (i-1)/bSize + 1;
```

```
    }
    cnt.resize(mx+1);
    for(int i = 1; i <= m; i ++) {
        auto &[l, r, qid] = query[i];
        cin >> l >> r;
        qid = i;
    }
    sort(1+all(query), [&](const Rec &a, const Rec &b) {
        int abl = belong[a.l], bbl = belong[b.l];
        if(abl != bbl) {
            return abl < bbl;
        }
        else {
            if(abl & 1) return a.r < b.r;
            else return a.r > b.r;
        }
    });

    vector<LL> ans(m+1);
    int l = 1, r = 0;
    for(int i = 1; i <= m; i ++) {
        auto [ll, rr, qid] = query[i];
        // 整体加减顺序，现在遇到的都没关系，但建议想一想。
        while(l < ll) del(l ++);
        while(l > ll) add(-- l);
        while(r < rr) add(++ r);
        while(r > rr) del(r --);
        ans[qid] = tans;
    }
    for(int i = 1; i <= m; i ++) {
        cout << ans[i] << '\n';
    }
}
```

字符串

字符串双哈希

```
typedef pair<int,int> hashv;
const LL mod1=1000000007;
const LL mod2=1000000009;

hashv operator + (hashv a, hashv b) {
    int c1=a.fi+b.fi,c2=a.se+b.se;
    if (c1>=mod1) c1-=mod1;
    if (c2>=mod2) c2-=mod2;
    return mkp(c1,c2);
}

hashv operator - (hashv a, hashv b) {
    int c1=a.fi-b.fi,c2=a.se-b.se;
    if (c1<0) c1+=mod1;
    if (c2<0) c2+=mod2;
    return mkp(c1,c2);
```

```
}

hashv operator * (hashv a, hashv b) {
    return mkp(1ll*a.fi*b.fi%mod1,1ll*a.se*b.se%mod2);
}

hashv pw[N], Hx[N];

hashv base=mkp(13331, 23333);

pw[0] = mkp(1,1);
for (int i=1;i<=n;i++)
    pw[i] = pw[i-1]*base, Hx[i] = Hx[i-1]*base+mkp(s[i],s[i]);

// 前面的才是高位，需要抵消掉
hashv hx1 = Hx[r]-Hx[l-1]*pw[r-l+1];
hashv hx2 = Hx[x]-Hx[x-len]*pw[len];

if (s1==s2) {
    // 双哈希相等
}
```

数学

快速幂 & ExGcd 求逆元 & 线性求逆元

```
int qpm(int a, int b, const int &c) { // int/LL
    int ans = 1 % c;
    while(b) {
        if(b & 1) ans = 1LL*ans*a % c;
        a = 1LL*a*a % c;
        b >>= 1;
    }
    return ans;
}
int ExGcd(int a, int b, int &x, int &y) {
    if(!b) {
        x = 1, y = 0;
        return a;
    }
    int d = ExGcd(b, a%b, x, y);
    int t = x;
    x = y, y = t-a/b*y;
    return d;
}
// a关于b的逆元，gcd(a, b) != 1时，逆元不存在。
int ExGcdInv(int a, int b) {
    int x, y;
    int d = ExGcd(a, b, x, y);
    assert(d != 1);
    x = (x%b + b) % b;
    return x;
}
vector<int> inv;
```

```
void GetInvs(int n) {
    inv.resize(n+1);
    inv[1] = 1;
    for(int i = 2; i <= n; i ++)
        inv[i] = (LL)(MO - MO / i) * inv[MO % i] % MO;
}
```

组合数

```
const int MO = 998244353;

int qpm(int a, int b, const int &c = MO) {
    int ans = 1%c;
    while(b) {
        if(b & 1) ans = 1LL*ans*a%MO;
        a = 1LL*a*a%MO;
        b >>= 1;
    }
    return ans;
}
struct Comb {
    int n;
    vector<int> _fac, _facInv, _inv;
    Comb() : n{0}, _fac{1}, _facInv{1}, _inv{0} {}
    Comb(int n) : Comb() {
        Init(n);
    }
    void Init(int m) {
        if(m <= n) return;
        _fac.resize(m+1);
        _facInv.resize(m+1);
        _inv.resize(m+1);
        for(int i = n+1; i <= m; i ++) {
            _fac[i] = 1LL*_fac[i-1]*i%MO;
        }
        _facInv[m] = qpm(_fac[m], MO-2);
        for(int i = m; i > n; i --) {
            _facInv[i-1] = 1LL*_facInv[i]*i%MO;
            _inv[i] = 1LL*_facInv[i]*_fac[i-1]%MO;
        }
        n = m;
    }
    int Fac(int m) {
        if(m > n) Init(2*m);
        return _fac[m];
    }
    int FacInv(int m) {
        if(m > n) Init(2*m);
        return _facInv[m];
    }
    int Inv(int m) {
        if(m > n) Init(2*m);
        return _inv[m];
    }
    int Permu(int n, int m) {
        if(n < m || m < 0) return 0;
        return 1LL*Fac(n) * FacInv(n-m) % MO;
```

```
    }
    int Bimon(int n, int m) {
        if(n < m || m < 0) return 0;
        return 1LL*Fac(n) * FacInv(n-m) % MO * FacInv(m) %MO;
    }
} comb;
```

复数

```
struct Complex{
    double x, y;
    Complex(double xx = 0.0, double yy = 0.0) {
        x = xx, y = yy;
    }
    Complex operator + (const Complex &o) const {
        return Complex(x+o.x, y+o.y);
    }
    Complex operator - (const Complex &o) const {
        return Complex(x-o.x, y-o.y);
    }
    Complex operator * (const Complex &o) const {
        return Complex(x*o.x-y*o.y, x*o.y+y*o.x);
    }
};
```

FFT

```
const double Pi = acos(-1.0);

struct Complex{
    double x, y;
    Complex(double xx = 0.0, double yy = 0.0) {
        x = xx, y = yy;
    }
    Complex operator + (const Complex &o) const {
        return Complex(x+o.x, y+o.y);
    }
    Complex operator - (const Complex &o) const {
        return Complex(x-o.x, y-o.y);
    }
    Complex operator * (const Complex &o) const {
        return Complex(x*o.x-y*o.y, x*o.y+y*o.x);
    }
};

int n, m, limit = 1;
Complex a[N], b[N];

// 递归版本
void FFT(int tlimit, Complex a[], int on) {
    if(tlimit == 1) return;
    Complex a1[tlimit>>1], a2[tlimit>>1];
    for(int i = 0; i < tlimit; i += 2)
        a1[i>>1] = a[i], a2[i>>1] = a[i+1];
    FFT(tlimit>>1, a1, on), FFT(tlimit>>1, a2, on);
    Complex Wn(cos(2.0*Pi/tlimit), on*sin(2.0*Pi/tlimit)), w(1, 0);
```

```
    for(int i = 0; i < (tlimit>>1); i ++, w = w*Wn) {
        auto t = w * a2[i];
        a[i] = a1[i] + t;
        a[i+(tlimit>>1)] = a1[i] - t;
    }
}
// 迭代版本 Faster
void FFT(Complex a[], int on) {
    for(int i = 0; i < limit; i ++)
        if(i < R[i]) swap(a[i], a[R[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        Complex Wn(cos(Pi/mid), on*sin(Pi/mid));
        for(int j = 0, r = mid<<1; j < limit; j += r) {
            Complex w(1, 0);
            for(int k = 0; k < mid; k ++, w = w*Wn) {
                Complex x = a[j+k], y = w*a[j+mid+k];
                a[j+k] = x+y;
                a[j+mid+k] = x-y;
            }
        }
    }
}

void solve() {
    cin >> n >> m;
    for(int i = 0; i <= n; i ++)
        cin >> a[i].x;
    for(int i = 0; i <= m; i ++)
        cin >> b[i].x;
    limit = 1;
    while(limit <= n+m) limit <<= 1;
    FFT(limit, a, 1), FFT(limit, b, 1);
    for(int i = 0; i < limit; i ++)
        a[i] = a[i]*b[i];
    FFT(limit, a, -1);
    for(int i = 0; i <= n+m; i ++){
        cout << (int)(a[i].x/limit+0.5) << " \n"[i == n+m];
    }
}
```

NTT

```
const int N = 4e6+10, MO = 998244353, G = 3, Gi = 332748118;

int n, m, limit = 1, rr;
int R[N];
int a[N], b[N];

int qpm(int a, int b, const int &c = MO) {
    int ans = 1;
    while(b) {
        if(b & 1) ans = 1LL*ans*a%MO;
        a = 1LL*a*a%MO;
        b >>= 1;
    }
    return ans;
}
```

```
void NTT(int a[], int on) {
    for(int i = 0; i < limit; i ++)
        if(i < R[i]) swap(a[i], a[R[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        int Wn = qpm((on == 1 ? G : Gi), (MO-1)/(mid<<1));
        for(int j = 0, r = mid<<1; j < limit; j += r) {
            int w = 1;
            for(int k = 0; k < mid; k ++, w = 1LL*w*Wn%MO) {
                int x = a[j+k], y = 1LL*w*a[j+mid+k]%MO;
                a[j+k] = (1LL*x+y)%MO;
                a[j+mid+k] = (1LL*x-y)%MO;
            }
        }
    }
}
void solve() {
    cin >> n >> m;
    for(int i = 0; i <= n; i ++) {
        cin >> a[i];
        a[i] %= MO;
    }
    for(int i = 0; i <= m; i ++) {
        cin >> b[i];
        b[i] %= MO;
    }
    limit = 1, rr = 0;
    while(limit <= n+m) limit <<= 1, rr ++;
    for(int i = 0; i < limit; i ++)
        R[i] = (R[i>>1]>>1) | ((i&1)<<(rr-1));
    NTT(a, 1), NTT(b, 1);
    for(int i = 0; i <= limit; i ++)
        a[i] = 1LL*a[i]*b[i]%MO;
    NTT(a, -1);
    int inv = qpm(limit, MO-2);
    for(int i = 0; i <= n+m; i ++){
        cout << (1LL*a[i]*inv%MO+MO)%MO << " \n"[i == n+m];
    }
}
```

线性筛

```
vector<int> primes, minp, phi;

void Sieve(int n) {
    primes.clear();
    minp.assign(n+1, 0);
    phi.resize(n+1);
    phi[1] = 0;
    for(int i = 2; i <= n; i ++) {
        if(!minp[i]) {
            minp[i] = i;
            primes.push_back(i);
            phi[i] = i-1;
        }
        for(auto p : primes) {
            if(p > n/i) {
                break;
```

```
            }
            minp[i*p] = p;
            if(p == minp[i]) {
                phi[t] = phi[i]*p;
                break;
            }
            phi[t] = phi[i]*(p-1);
        }
    }
}
```

计算几何

```
const double eps = 1e-8;
const double PI = acos(-1);

// Remind: LL in Point/abs2/dot/cross?
int sign(double a) {
    return (a < -eps ? -1 : (a > eps));
}
int dcmp(double a, double b) {
    return sign(a-b);
}
// Point
struct Point {
    int x, y;
    Point(int x = 0, int y = 0) : x(x), y(y) {}
    Point operator - (const Point &p) {
        return Point(-p.x, -p.y);
    }
    friend Point operator + (const Point &a, const Point &b) {
        return Point(a.x+b.x, a.y+b.y);
    }
    friend Point operator - (const Point &a, const Point &b) {
        return Point(a.x-b.x, a.y-b.y);
    }
    Point operator * (const int &v) const {
        return Point(x*v, y*v);
    }
    Point operator / (const int &v) const {
        assert(v != 0);
        return Point(x/v, y/v);
    }
    bool operator < (const Point &o) const {
        int c = dcmp(x, o.x);
        if(c) return c == -1;
        return dcmp(y, o.y) == -1;
    }
    bool operator == (const Point &o) const {
        return dcmp(x, o.x) == 0 && dcmp(y, o.y) == 0;
    }
    double Dist(const Point &o) {
        return (*this-o).abs();
    }
    double alpha() {
```

```
        return atan2(y, x);
    }
    void read() {
        cin >> x >> y;
    }
    void write() {
        cout << '(' << x << ',' << y << ')' << '\n';
    }
    double abs() {
        return sqrt(abs2());
    }
    double abs2() {
        return x*x+y*y;
    }
    Point rot90() {
        return Point(-y, x);
    }
    Point unit() {
        return *this/abs();
    }
};
LL dot(const Point &a, const Point &b) {
    return 1LL*a.x*b.x + 1LL*a.y*b.y;
}
LL cross(const Point &a, const Point &b) {
    return 1LL*a.x*b.y - 1LL*a.y*b.x;
}
// Line
struct Line {
    // a -> b
    Point a, b;
    Line(Point _a = Point(), Point _b = Point()) : a(_a), b(_b) {}
};
bool PointOnLineLeft(const Point &p, const Line &l) {
    return cross(l.b-l.a, p-l.a) > 0;
}
Point LineIntersection(const Line &l1, const Line &l2) {
    return l1.a + (l1.b-l1.a) * (cross(l2.b-l2.a, l1.a-l2.a) / cross(l2.b-l2.a,
l1.a-l1.b));
}
bool PointOnSegment(const Point &p, const Line &l) {
    return sign(cross(p-l.a, l.b-l.a)) == 0 && min(l.a.x, l.b.x) <= p.x && p.x
<= max(l.a.x, l.b.x)
        && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
}
// Polygon
double area(const vector<Point> &poly) {
    double res = 0;
    int n = poly.size();
    for(int i = 0; i < n; i ++) {
        res += cross(poly[i], poly[(i+1)%n]);
    }
    return res / 2;
}
// 射线法
int PointInPolygonLine(const Point &p, const vector<Point> &poly) {
    // -1: out, 0: on, 1: in
    int n = poly.size(), res = 0;
```

```
    for(int i = 0; i < n; i ++) {
        Point u = poly[i], v = poly[(i+1)%n];
        if(PointOnSegment(p, Line(u, v))) {
            return 0;
        }
        if(dcmp(u.y, v.y) <= 0) {
            swap(u, v);
        }
        if(dcmp(p.y, u.y) > 0 || dcmp(p.y, v.y) <= 0) {
            continue;
        }
        res ^= PointOnLineLeft(p, Line(u, v));
    }
    if(res == 1) return 1;
    return -1;
}
// 转角法（优化）
int PointInPolygonAngle(const Point &p, const vector<Point> &poly) {
    // -1: out, 0: on, 1: in
    int n = poly.size(), wn = 0;
    for(int i = 0; i < n; i++) {
        Point u = poly[i], v = poly[(i+1)%n];
        if(PointOnSegment(p, Line(u, v))) {
            return 0;
        }
        int k = PointOnLineLeft(p, Line(u, v));
        int d1 = sign(u.y - p.y);
        int d2 = sign(v.y - p.y);
        if(k == 0 && d1 > 0 && d2 <= 0) wn ++;
        if(k > 0 && d1 <= 0 && d2 > 0) wn --;
    }
    if(wn != 0) return 1;
    return -1;
}
int PointInConvex(const Point &p, const vector<Point> &poly) {
    // counter-clockwise
    // -1: out, 0: on, 1: in
    if(PointOnLineLeft(p, Line(poly[1], poly[0])) ||
        PointOnLineLeft(p, Line(poly[0], poly.back())))
        return -1;
    if(PointOnSegment(p, Line(poly[0], poly.back()))) return 0;
    int n = poly.size();
    int l = 1, r = n-1;
    while(l < r) {
        int mid = l+r >> 1;
        if(PointOnLineLeft(p, Line(poly[0], poly[mid]))) l = mid+1;
        else r = mid;
    }
    if(PointOnSegment(p, Line(poly[r-1], poly[r]))) return 0;
    if(PointOnLineLeft(p, Line(poly[r-1], poly[r]))) return 1;
    return -1;
}
```

两球体积的交

```
double GetVol(Point a, Point b) {
    double dist2 = GetDist2(a, b);
    if(dist2 >= a.r+b.r) return 0;

    if(dist2+a.r <= b.r) return 4.0/3.0*PI*a.r*a.r*a.r;
    if(dist2+b.r <= a.r) return 4.0/3.0*PI*b.r*b.r*b.r;

    /*
    double dx = (a.r*a.r-b.r*b.r)/dist2;
    double L = (dist2+dx)/2.0, l = dist2-L;
    double x1 = a.r - L, x2 = b.r - l;
    double res = PI*x1*x1*(a.r - x1 / 3.0);
    res += PI*x2*x2*(b.r - x2 / 3.0);
    */

    double cosA = (a.r*a.r+dist2*dist2-b.r*b.r)/(2.0*dist2*a.r);
    double hA = a.r*(1.0-cosA);
    double res = PI*hA*hA*(3.0*a.r-hA)/3.0;

    double cosB = (b.r*b.r+dist2*dist2-a.r*a.r)/(2.0*dist2*b.r);
    double hB = b.r*(1.0-cosB);
    res += PI*hB*hB*(3.0*b.r-hB)/3.0;

    return res;
}
```

杂

不怎么用，但是需要用的语法

```
// 赋值函数
// void的递归auto有问题。有引用，带引用
std::function<void(int, int, int &)>
function<返回值(类型1, 类型2)> dfs = [&](类型1 a, 类型2 b) {
    ...
}
// 好用的库函数
to_string()
stoi(), stol, stoll(string, [idx], [base]) // string -> int, long, long long
iota(a, a+n, 0) // 0, 1 ... n-1
```

快读快写

```
 __int128 read() {
    __int128 x = 0,f = 1;
    char ch = cin.get(); // getchar();
    while(ch < '0' || ch > '9') {
        if(ch == '-') f = -1;
        ch = cin.get(); // getchar();
    }
```

```
    while('0' <= ch && ch <= '9')
        x = (x<<3)+(x<<1)+ch-'0', ch = cin.get();  // getchar();
    return x * f;
}
void write(__int128 x) {
    static int _stk_[40];
    int top = 0;
    do {
        _stk_[++ top] = x % 10, x /= 10;
    } while(x);
    while(top)
        putchar(_stk_[top --] + '0');
}
```

Meet in the Middle

```
// 区间放
void dfs(int cur, int goal, int sum, vector<int> &ans) { // int ans[], int &id
    if(goal == cur) { // >
        ans.push_back(sum);
        return;
    }
    for() {
        // 枚举
    }
}
// 根据集合大小动态放
void step(vector<LL> &ans, int p) {
    int siz = ans.size();
    for(int i = 0; i < siz; i ++) {
        LL x = ans[i];
        while(x*p <= n) {
            x *= p;
            ans.push_back(x);
        }
    }
};

// 上面 == 用开区间
dfs(0, n/2, 0, ans1);
dfs(n/2, n, 0, ans2);
// 上面 >  用闭区间
dfs(0, n/2-1, 0, ans1);
dfs(n/2, n-1, 0, ans2);
/**********或者*************/
dfs(1, n/2, 0, ans1);
dfs(n/2+1, n, 0, ans2);
/* 不然会造成RE/MLE等情况，因为不是一半分
    就记忆下面的这种就行，上面是都-1。
    ！最终是为了状态数平分，所以划分可以一个一个进来，
    判断两个集合的大小，看放在哪里，或者先跑，
    然后找到一个不错的划分线                        */
```

模拟退火

```
const int N = 110;

mt19937 rnd(time(0));
int n;
vector<PDD> p;
PDD ansp;
double ans = 1e8;

double Rand(double l, double r) {
    return (double)rnd() / RAND_MAX * (r-l) + l;
}
double GetDist(PDD a, PDD b) {
    double dx = a.fi-b.fi, dy = a.se-b.se;
    return sqrt(dx*dx + dy*dy);
}
double Calc(PDD a) {
    double res = 0;
    for(auto x : p) {
        res += GetDist(a, x);
    }
    ans = min(ans, res);
    return res;
}
void SimulateAnneal() {
    double t = 1e4;
    PDD curp = ansp;
    while(t > 1e-4) {
        PDD newp(Rand(curp.fi-t, curp.fi+t), Rand(curp.se-t, curp.se+t));
        double delta = Calc(newp) - Calc(curp);
        if(exp(-delta/t) > Rand(0, 1))
            curp = newp;
        t *= 0.985;
    }
}
void solve() {
    srand(time(0));
    cin >> n;
    p.resize(n);
    for(auto &[x, y] : p) {
        cin >> x >> y;
        ansp.fi += x, ansp.se += y;
    }
    ansp.fi /= n, ansp.se /= n;
    // for(int i = 0; i < 100; i++)
    while ((double)clock()/CLOCKS_PER_SEC < 0.8)
        SimulateAnneal();
    cout << ans << '\n';
}
/*
    T 一般状态数的一半，结束T一般为精度要求-2位
    1. 普通类函数SA，可以当前最优为起点，开始可以取中心。
    2. 分组SA - 先random_shuffle序列，在加贪心/DP
    3. 排序SA - swap两个变化较小，算有连续性，check下满足性质。。
    4. 生成树SA - 先生成，然后改边。
*/
```