

## Contents

1 Basic	1
1.1 vimrc	1
2 Flow	1
2.1 Dinic	1
2.2 MCMF	2
3 DataStructure	2
3.1 KDTree	2
3.2 BIT	4
3.3 DisjointSet	4
3.4 HeavyLightDecomposition	4
3.5 LCA	4
3.6 MO	5
3.7 PartitionTree	5
3.8 PersistentSegmentTree	6
3.9 PersistentTreap	6
3.10 SparseTable	7
3.11 pbds_heap	7
3.12 pbds_tree	7
3.13 unordered_map	7
4 Geometry	8
4.1 ClosestPair	8
4.2 Geometry	8
5 DP	12
5.1 KnapsackLimit	12
5.2 Digit_Count	12
6 Graph	12
6.1 BCC	12
6.2 Blossom	12
6.3 CutBridge	13
6.4 Dijkstra	13
6.5 MaximumClique	14
6.6 MinMeanCycle	14
6.7 SCC	14
6.8 KM	15
6.9 GridMST	15
7 Math	16
7.1 bigN	16
7.2 BSGS	17
7.3 CRT	17
7.4 ExtgcdModInv	17
7.5 FFT	17
7.6 Matrix	18
7.7 Mobius	18
7.8 PHITable	18
7.9 MillerRabin	18
7.10 PrimitiveRoot	19
7.11 Simplex	19
7.12 PollardRho	19
8 String	20
8.1 ACAutomaton	20
8.2 Eertree	20
8.3 KMP	21
8.4 minRotation	21
8.5 SA	21
8.6 SAM	21
8.7 Z	21
8.8 BWT	22
8.9 IBWT	22
9 Other	22
9.1 Python	22

## Basic

### vimrc

```
set nu ai si cin ts=4 sw=4 sts=4 mouse=a expandtab
syn on
imap {<CR> {<CR>}<Esc>ko
map <F5> :w<LF>:!g++ -O2 -std=c++11 % && echo "-----
Start-----" && ./a.out<LF>
map <F6> :w<LF>:!g++ -O2 -std=c++11 % && echo "-----
Start-----" && time ./a.out < input.in<LF>
map <F9> :tabe input.in<LF>
```

## Flow

### Dinic

```
const LL INF = 0x3f3f3f3f3f3f3f3f;
const int MAXN = 1e3 + 5;
const int MAXM = (MAXN * MAXN) / 2;
struct Graph{
    struct Node; struct Edge;
    int V;
    struct Node : vector<Edge*>{
        iterator cur; int d;
        Node(){ clear(); }
    }_memN[MAXN], *node[MAXN];
    struct Edge{
        Node *u, *v;
        Edge *rev;
        LL c, f;
        Edge(){
            Edge(Node *u, Node *v, LL c, Edge *rev) : u(u),
                v(v), c(c), f(0), rev(rev){}
        }
    }_memE[MAXM], *ptrE;
    Graph(int _V) : V(_V) {
        for (int i = 0; i < V; i++)
            node[i] = _memN + i;
        ptrE = _memE;
    }
    void addEdge(int _u, int _v, LL _c){
        *ptrE = Edge(node[_u], node[_v], _c, ptrE + 1);
        node[_u]->PB(ptrE++);
        *ptrE = Edge(node[_v], node[_u], _c, ptrE - 1);
        // 有向: 0, 無向: _c
        node[_v]->PB(ptrE++);
    }
    Node *s, *t;
    LL maxFlow(int _s, int _t){
        s = node[_s], t = node[_t];
        LL flow = 0;
        while (bfs()) {
            for (int i = 0; i < V; i++)
                node[i]->cur = node[i]->begin();
            flow += dfs(s, INF);
        }
        return flow;
    }
    bool bfs(){
        for (int i = 0; i < V; i++) node[i]->d = -1;
        queue<Node*> q; q.push(s); s->d = 0;
        while (q.size()) {
            Node *u = q.front(); q.pop();
            for (auto e : *u) {
                Node *v = e->v;
                if (!~v->d && e->c > e->f)
                    q.push(v), v->d = u->d + 1;
            }
        }
        return ~t->d;
    }
    LL dfs(Node *u, LL a){
        if (u == t || !a) return a;
        LL flow = 0, f;
        for (; u->cur != u->end(); u->cur++) {
            auto &e = *u->cur; Node *v = e->v;
            if (u->d + 1 == v->d && (f = dfs(v, min(a,
                e->c - e->f))) > 0) {
                e->f += f; e->rev->f -= f;
                flow += f; a -= f;
                if (!a) break;
            }
        }
        return flow;
    }
};
```

## MCMF

```

const int MAXN = 300;
const int MAXM = MAXN * MAXN * 2;
const LL INF = 0x3f3f3f3f3f3f3f3f;
struct Graph {
    struct Node; struct Edge; int V;
    struct Node : vector<Edge*> {
        bool inq; Edge *pa; LL a, d;
        Node() { clear(); }
    }_memN[MAXN], *node[MAXN];
    struct Edge{
        Node *u, *v; Edge *rev;
        LL c, f, _c; Edge() {}
        Edge(Node *u, Node *v, LL c, LL _c, Edge *rev)
            : u(u), v(v), c(c), f(0), _c(_c), rev(rev) {}
    }_memE[MAXM], *ptrE;
    Graph(int _V) : V(_V) {
        for (int i = 0; i < V; i++)
            node[i] = _memN + i;
        ptrE = _memE;
    }
    void addEdge(int u, int v, LL c, LL _c) {
        *ptrE = Edge(node[u], node[v], c, _c, ptrE + 1);
        node[u]->PB(ptrE++);
        *ptrE = Edge(node[v], node[u], 0, -_c, ptrE - 1);
        node[v]->PB(ptrE++);
    }
    Node *s, *t;
    bool SPFA() {
        for (int i = 0; i < V; i++) node[i]->d = INF,
            node[i]->inq = false;
        queue<Node*> q; q.push(s); s->inq = true;
        s->d = 0, s->pa = NULL, s->a = INF;
        while (q.size()) {
            Node *u = q.front(); q.pop(); u->inq = false;
            for (auto &e : *u) {
                Node *v = e->v;
                if (e->c > e->f && v->d > u->d + e->_c) {
                    v->d = u->d + e->_c;
                    v->pa = e; v->a = min(u->a, e->c - e->f);
                    if (!v->inq) q.push(v), v->inq = true;
                }
            }
        }
        return t->d != INF;
    }
    pLL maxFlowMinCost(int _s, int _t) {
        s = node[_s], t = node[_t];
        pLL res = MP(0, 0);
        while (SPFA()) {
            res.F += t->a;
            res.S += t->d * t->a;
            for (Node *u = t; u != s; u = u->pa->u) {
                u->pa->f += t->a;
                u->pa->rev->f -= t->a;
            }
        }
        return res;
    }
};

```

## DataStructure

## KDTree

```
#define MAXN 50100
```

```

inline long long sq(long long x){return x*x;}
const double alpha=0.75;
int W,H,rx[MAXN],ry[MAXN];
namespace KDTree{
    struct Point {
        int x,y;
        int index;
        long long distance(const Point &b)const{
            return sq(x-b.x) + sq(y-b.y);
        }
        bool operator==(const Point& rhs){return index==rhs.index;}
    };
    struct qnode{
        Point p;
        long long dis;
        qnode(){
            qnode(Point _p,long long _dis){
                p = _p;
                dis = _dis;
            }
        }
        bool operator <(const qnode &b)const{
            if(dis != b.dis)return dis < b.dis;
            else return p.index < b.p.index;
        }
    };
    priority_queue<qnode>q;
    inline bool cmpX(const Point &a,const Point &b){
        return a.x < b.x || (a.x == b.x && a.y < b.y) || (a.x == b.x && a.y == b.y && a.index < b.index);
    }
    inline bool cmpY(const Point &a,const Point &b){
        return a.y < b.y || (a.y == b.y && a.x < b.x) || (a.y == b.y && a.x == b.x && a.index < b.index);
    }
    bool cmp(const Point &a,const Point &b,bool div){
        return div?cmpY(a,b):cmpX(a,b);
    }
    struct Node{
        Point e;
        Node *lc,*rc;
        int size;
        bool div;
        inline void pull(){
            size = 1 + lc->size + rc->size;
        }
        inline bool isBad(){
            return lc->size > alpha*size || rc->size > alpha*size;
        }
    };
    pool[MAXN],*tail,*root,*recycle[MAXN],*null;
    int rc_cnt;
    void init(){
        tail = pool;
        null = tail++;
        null->lc = null->rc = null;
        null->size = 0;
        rc_cnt = 0;
        root = null;
    }
    Node *newNode(Point e){
        Node *p;
        if(rc_cnt)p = recycle[--rc_cnt];
        else p = tail++;
        p->e = e;
        p->lc = p->rc = null;
        p->size = 1;
        return p;
    }
    Node *build(Point *a,int l,int r,bool div){
        if(l >= r)return null;
        int mid = (l+r)/2;
        nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
        Node *p = newNode(a[mid]);
        p->div = div;
        p->lc = build(a,l,mid,!div);
        p->rc = build(a,mid+1,r,!div);
    }
}

```

```

    p->pull();
    return p;
}
void getTree(Node *p,vector<Point>& v){
    if(p==null) return;
    getTree(p->lc,v);
    v.push_back(p->e);
    recycle[rc_cnt++]=p;
    getTree(p->rc,v);
}
Node *rebuild(vector<Point>& v,int l,int r,bool div){
    if(l>=r) return null;
    int mid = (l+r)/2;
    nth_element(v.begin()+l,v.begin()+mid,v.begin()+r,
        div?cmpY:cmpX);
    Node *p = newNode(v[mid]);
    p->div = div;
    p->lc = rebuild(v,l,mid,!div);
    p->rc = rebuild(v,mid+1,r,!div);
    p->pull();
    return p;
}
void rebuild(Node *p){
    vector<Point> v;
    getTree(p,v);
    p = rebuild(v,0,v.size(),p->div);
}
Node **insert(Node *p,Point a,bool div){
    if(p==null){
        p = newNode(a);
        p->div = div;
        return &p;
    }
    else{
        Node **res;
        if(cmp(a,p->e,div)) res=insert(p->lc,a,!div);
        else res=insert(p->rc,a,!div);
        p->pull();
        if(p->isBad()) res=&p;
        return res;
    }
}
void insert(Point e){
    Node **p = insert(root,e,0);
    if(*p!=null) rebuild(*p);
}
Node **get_min(Node *p,bool div){
    if(p->div==div){
        if(p->lc!=null) return get_min(p->lc,div);
        else return &p;
    }
    else{
        Node **res=&p,**tmp;
        if(p->lc!=null){
            tmp = get_min(p->lc,div);
            if(cmp((*tmp)->e,(*res)->e,div)) res=tmp;
        }
        if(p->rc!=null){
            tmp = get_min(p->rc,div);
            if(cmp((*tmp)->e,(*res)->e,div)) res=tmp;
        }
        return res;
    }
}
void del(Node *p){
    Node **nxt;
    if(p->rc!=null){
        nxt = get_min(p->rc,p->div);
        p->e = (*nxt)->e;
        del(*nxt);
    }
    else if(p->lc!=null){
        nxt = get_min(p->lc,p->div);
        p->e = (*nxt)->e;
        del(*nxt);
        p->rc = p->lc;
        p->lc = null;
    }
}

```

```

    }
    else{
        recycle[rc_cnt++]=p;
        p=null;
    }
}
void del(Node *p,Point d){
    if(p->e==d){
        del(p);
    }
    else if(cmp(d,p->e,p->div)) del(p->lc,d);
    else del(p->rc,d);
}
void search(Point p,Node *t,bool div,int m){
    if(!t) return;
    if(cmp(p,t->e,div)){
        search(p,t->lc,!div,m);
        if(q.size() < m){
            q.push(qnode(t->e,p.distance(t->e)));
            search(p,t->rc,!div,m);
        }
    }
    else {
        if(p.distance(t->e) <= q.top().dis){
            q.push(qnode(t->e,p.distance(t->e)));
            q.pop();
        }
        if(!div){
            if(sq(t->e.x-p.x) <= q.top().dis)
                search(p,t->rc,!div,m);
        }
        else {
            if(sq(t->e.y-p.y) <= q.top().dis)
                search(p,t->rc,!div,m);
        }
    }
}
else {
    search(p,t->rc,!div,m);
    if(q.size() < m){
        q.push(qnode(t->e,p.distance(t->e)));
        search(p,t->lc,!div,m);
    }
}
else {
    if(p.distance(t->e) <= q.top().dis){
        q.push(qnode(t->e,p.distance(t->e)));
        q.pop();
    }
    if(!div){
        if(sq(t->e.x-p.x) <= q.top().dis)
            search(p,t->lc,!div,m);
    }
    else {
        if(sq(t->e.y-p.y) <= q.top().dis)
            search(p,t->lc,!div,m);
    }
}
}
}
void search(Point p,int m){
    while(!q.empty())q.pop();
    search(p,root,0,m);
}
void getRange(Node *p,vector<Point>& v,int x1,int x2,
    int y1,int y2){
    if(p==null) return;
    if(x1<=p->e.x && p->e.x<=x2 && y1<=p->e.y && p->e.y
        <=y2) v.push_back(p->e);
    if(p->div ? y1<=p->e.y : x1<=p->e.x) getRange(p->lc,
        v,x1,x2,y1,y2);
    if(p->div ? y2>=p->e.y : x2>=p->e.x) getRange(p->rc,
        v,x1,x2,y1,y2);
}
void solve(Point p){
    del(root,p);
    insert(p);
}
};

```

```

KDTree::Point p[MAXN];
int main(){
    KDTree::init();
    KDTree::root = KDTree::build(p,0,n,0);
    while(q--){
        KDTree::Point tmp,p1,p2;
        scanf("%d%d",&tmp.x,&tmp.y);
        search(tmp,2);
        p1=KDTree::q.top().p;
        KDTree::q.pop();
        p2=KDTree::q.top().p;
        KDTree::q.pop();
    }
    return 0;
}

```

## BIT

```

// ONE BASE!!
const int MAXN = 5e4 + 5;
struct BIT{
    int data[MAXN], n;
    BIT(int *arr, int _n){ n = _n;
        memset(data, 0, sizeof(data));
        for (int i = 1 ; i <= n ; i++)
            add(i, arr[i]);
    }
    int lowbit(int x) { return x & (-x); }
    int sum(int x){
        int res = 0;
        while (x > 0) res += data[x], x -= lowbit(x);
        return res;
    }
    void add(int x, int d){
        while (x <= n) data[x] += d, x += lowbit(x);
    }
};

```

## DisjointSet

```

struct djs {
    vector<int> pa; int n;
    djs(int _n) : n(_n) { pa.resize(n, -1); }
    int find(int x) { return pa[x] < 0 ? x : pa[x] = find(pa[x]); }
    bool Union(int u, int v) {
        int x = find(u), y = find(v);
        if (x == y) return false;
        if (pa[x] < pa[y]) swap(x, y);
        pa[y] += pa[x], pa[x] = y;
        return true;
    }
};

```

## HeavyLightDecomposition

```

const int MAXN = 1e3 + 5;
struct Tree{
    struct Node; struct Edge; int V;
    struct Node : vector<Node*> {
        int sz, dep, v, id;
        Node *pa, *top, *hc;
    }_memN[MAXN], *node[MAXN], *rt;
    Tree(int _V) : V(_V) {
        for (int i = 0 ; i < V ; i++)
            node[i] = _memN + i;
        rt = node[0];
    }
    void addEdge(int u, int v) {
        node[u]->push_back(node[v]);
        node[v]->push_back(node[u]);
    }
};

```

```

}
int stamp;
void HLD() {
    stamp = 0;
    dfs_size(rt);
    dfs_link(rt, rt);
}
void dfs_size(Node *u) {
    u->sz = 1; u->hc = NULL;
    for (auto v : *u) {
        if (v == u->pa) continue;
        v->pa = u;
        v->dep = u->dep + 1;
        dfs_size(v);
        if (!u->hc || v->sz > u->hc->sz)
            u->hc = v;
        u->sz += v->sz;
    }
}
void dfs_link(Node *u, Node *_top) {
    u->id = stamp++;
    u->top = _top;
    if (!u->hc) return;
    dfs_link(u->hc, _top);
    for (auto v : *u) {
        if (v == u->hc || v == u->pa) continue;
        dfs_link(v, v);
    }
}
Node* query(int _u, int _v) {
    Node *u = node[_u], *v = node[_v];
    Node *uTop = u->top, *vTop = v->top;
    while (uTop != vTop) {
        if (uTop->dep < vTop->dep)
            swap(u, v), swap(uTop, vTop);
        // query [uTop->id, u->id + 1)
        uTop = (u = uTop->pa)->top;
    }
    // if (u != v) query[u->id + 1, v->id + 1)
    return u->dep < v->dep ? u : v; // LCA
};

```

## LCA

```

const int MAXN = 1e5 + 5;
const int lgN = __lg(MAXN) + 5;
struct Tree {
    struct Node : vector<Node*>{
        int dep, v;
        Node* pa[lgN];
        int maxV[lgN];
        Node() {
            clear(), dep = -1;
            for (int i = 0 ; i < lgN ; i++)
                maxV[i] = -INF;
        }
    }_memN[MAXN], *node[MAXN];
    int V;
    Tree(int _V) : V(_V) {
        for (int i = 0 ; i < V ; i++)
            node[i] = _memN + i;
    }
    inline void addEdge(int u, int v) {
        node[u]->push_back(node[v]);
        node[v]->push_back(node[u]);
    }
    void solve() {
        dfs(node[0], node[0], 0);
    }
    void dfs(Node *u, Node *p, int dep) {
        u->pa[0] = p; u->dep = dep;
        u->maxV[0] = max(u->v, p->v);
        for (int i = 1 ; i < lgN ; i++)
            u->pa[i] = u->pa[i - 1]->pa[i - 1],

```

```

        u->maxV[i] = max(u->maxV[i - 1], u->pa[i - 1]->maxV[i - 1]);
    for (auto v : *u)
        if (!v->dep)
            dfs(v, u, dep + 1);
}
int query(int _u, int _v) {
    Node *u = node[_u], *v = node[_v];
    int ans = max(u->v, v->v);
    if (u->dep < v->dep) swap(u, v);
    for (int i = lgN - 1; ~i; i--)
        if (u->pa[i]->dep >= v->dep)
            ans = max(ans, u->maxV[i]), u = u->pa[i];
    if (u == v) return ans;
    for (int i = lgN - 1; ~i; i--)
        if (u->pa[i] != v->pa[i])
            ans = max({ans, u->maxV[i], v->maxV[i]}),
            u = u->pa[i], v = v->pa[i];
    return ans = max({ans, u->maxV[0], v->maxV[0]});
};

```

## MO

```

const int MAXN = 1e5 + 5;
const int MAXV = 1e5 + 5;
const int MAXQ = 1e6 + 5;
struct MO {
    struct Q {
        int l, r, id, b;
        Q(int _l, int _r, int _id, int _b) : l(_l), r(_r), id(_id), b(_b) {}
        bool operator < (const Q &q) const {
            return b == q.b ? r < q.r : l < q.l;
        }
    };
    int qn, sqn;
    vector<int> data; vector<Q> qs;
    pii ans; int cnt[MAXN], val_cnt[MAXN];
    MO(vector<int> &_data, vector<pii> &_qs) : data(_data) {
        qn = _qs.size(), sqn = (int)(sqrt(qn) + 1e-6);
        for (int i = 0; i < _qs.size(); i++)
            qs.emplace_back(_qs[i].F, _qs[i].S, i, _qs[i].F / sqn);
        ans = make_pair(0, 0);
        memset(cnt, 0, sizeof(cnt));
        memset(val_cnt, 0, sizeof(val_cnt));
    }
    vector<pii> solve() {
        vector<pii> ret(qn);
        sort(qs.begin(), qs.end());
        int l = 0, r = 0;
        for (auto q : qs) {
            while (r < q.r) update(data[r++], 1);
            while (r > q.r) update(data[--r], -1);
            while (l > q.l) update(data[--l], 1);
            while (l < q.l) update(data[l++], -1);
            ret[q.id] = ans;
        }
        return ret;
    }
    void update(int num, int op) {
        if (op == 1) {
            if (cnt[num]) val_cnt[cnt[num]]--;
            val_cnt[++cnt[num]]++;
            if (ans.F == cnt[num]) ans.S++;
            if (ans.F < cnt[num]) ans.F++, ans.S = 1;
        }
        if (op == -1) {
            val_cnt[cnt[num]]--;
            val_cnt[--cnt[num]]++;
            if (ans.F == cnt[num] + 1)

```

```

        if (ans.S == 1)
            ans.F--, ans.S = val_cnt[cnt[num]];
        else ans.S--;
    }
};
int main() { ios_base::sync_with_stdio(false); cin.tie(0);
    int n, q; cin >> n >> q;
    vector<int> data(n);
    vector<pii> qs(q);
    for (auto &num : data) cin >> num;
    for (auto &p : qs) { cin >> p.F >> p.S; p.F--; }
    MO *sol = new MO(data, qs);
    vector<pii> ans = sol->solve();
    for (auto p : ans) cout << p.F << ' ' << p.S << '\n';
}

```

## PartitionTree

```

const int MAXN = 50005;
const int lgN = __log(MAXN) + 5;
struct PT{
    int sorted[MAXN];
    int tree[lgN][MAXN];
    int toleft[lgN][MAXN];
    int n;
    void build(int l, int r, int dep){
        if (l == r) return;
        int mid = (l+r) >> 1;
        int same = mid - l + 1;
        for (int i = l; i <= r; i++)
            if (tree[dep][i] < sorted[mid])
                same--;
        int lpos = l;
        int rpos = mid+1;
        for (int i = l; i <= r; i++){
            if (tree[dep][i] < sorted[mid])
                tree[dep+1][lpos++] = tree[dep][i];
            else if (tree[dep][i] == sorted[mid] && same){
                tree[dep+1][lpos++] = tree[dep][i];
                same--;
            }else
                tree[dep+1][rpos++] = tree[dep][i];
            toleft[dep][i] = toleft[dep][l-1] + lpos - 1;
        }
        build(l, mid, dep+1);
        build(mid+1, r, dep+1);
    }
    int query(int L, int R, int l, int r, int dep, int k){
        if (l == r) return tree[dep][l];
        int mid = (L+R) >> 1;
        int cnt = toleft[dep][r] - toleft[dep][l-1];
        if (cnt >= k){
            int newl = L + toleft[dep][l-1] - toleft[dep][L-1];
            int newr = newl + cnt - 1;
            return Query(L, mid, newl, newr, dep+1, k);
        }else{
            int newr = r + toleft[dep][R] - toleft[dep][r];
            int newl = newr - (r - l - cnt);
            return Query(mid + 1, R, newl, newr, dep+1, k-cnt);
        }
    }
    void Insert(int _n){
        n = _n;
        for (int i = 0; i < n; i++){
            cin >> tree[0][i];
            sorted[i] = tree[0][i];
        }
    }
}

```

```

        sort(sorted, sorted + n);
        build(0, n-1, 0);
    }
    int query(int l, int r, int k){
        return query(0, n-1, l, r, 0, k);
    }
} _PT;
int main(){
    int n, q; cin >> n >> q;
    _PT.Insert(n);
    for (int i = 0 ; i < q; i++){
        int x, y, k; cin >> x >> y >> k;
        cout << _PT.query(x-1, y-1, k) << '\n';
    }
}

```

## PersistentSegmentTree

```

// SmartPointer
template <typename T>
struct _ptrCntr{
    T v; int cnt;
    _ptrCntr(const T& _v = 0) : v(_v), cnt(0){}
};
template <typename T>
struct Sptr{
    _ptrCntr<T> *p;
    T& operator->(){ return &p->v; }
    T& operator*(){ return p->v; }
    operator _ptrCntr<T>*(){ return p; }
    Sptr& operator = (const Sptr& t){
        if (p && !--p->cnt) delete p;
        (p = t.p) && ++p->cnt; return *this;
    }
    Sptr(_ptrCntr<T> *t = NULL) : p(t){p && ++p->cnt;}
    Sptr(const Sptr& t) : p(t.p){p && ++p->cnt;}
    ~Sptr(){ if (p && !--p->cnt) delete p; }
};
template <typename T>
inline Sptr<T> _new(const T& u){
    return Sptr<T>(new _ptrCntr<T>(u));
}
// PersistentSegmentTree
const int MAXN = 1e5 + 5;
const int lgN = __lg(MAXN) + 5;
const int MAXK = 100;
struct PersistentSegmentTree{
    struct Node{
        Sptr<Node> l, r;
        int L, R;
        // data
        // tag
        Node(int _L, int _R) : l(NULL), r(NULL){
            L = _L, R = _R;
            // data tag init
        }
        int len(){ return R - L; }
        int mid(){ return (R + L) >> 1; }
    };
    Sptr<Node> rt[MAXK];
    int *arr, n, kCnt;
    PersistentSegmentTree(int *_arr, int _n){
        arr = _arr, n = _n; kCnt = 0;
        rt[0] = build(0, n);
    }
    Sptr<Node> copy(Sptr<Node> &u){
        return _new(*u);
    }
    Sptr<Node> build(int L, int R){
        Sptr<Node> u = _new(Node(L, R));
        if (u->len() == 1){
            // base data
            return u;
        }
        int M = u->mid();
        u->l = build(L, M);

```

```

        u->r = build(M, R);
        return pull(u);
    }
    Sptr<Node> pull(Sptr<Node> &u, Sptr<Node> &l, Sptr<Node> &r){
        if (!l || !r) return l ? l : r;
        push(l), push(r);
        // pull function
        return u;
    }
    void push(Sptr<Node> &u){
        if (!u) return ;
        // push function
    }
    Sptr<Node> pull(Sptr<Node> &u){
        return pull(u, u->l, u->r);
    }
    Sptr<Node> modify(int mL, int mR, int v, Sptr<Node> &u){
        if (u->R <= mL || mR <= u->L) return u;
        Sptr<Node> _u = copy(u);
        if (mL <= u->L && u->R <= mR) {
            // tag (on copy node)
            return _u;
        }
        push(u);
        int M = u->mid();
        _u->l = modify(mL, mR, v, u->l);
        _u->r = modify(mL, mR, v, u->r);
        return pull(_u);
    }
    Sptr<Node> query(int qL, int qR, Sptr<Node> &u){
        if (u->R <= qL || qR <= u->L) return Sptr<Node>(NULL);
        if (qL <= u->L && u->R <= qR) return u;
        push(u); int M = u->mid();
        Sptr<Node> res = _new(Node(u->L, u->R));
        Sptr<Node> l = query(qL, qR, u->l);
        Sptr<Node> r = query(qL, qR, u->r);
        return pull(res, l, r);
    }
    void modify(int mL, int mR, int v){
        rt[kCnt + 1] = modify(mL, mR, v, rt[kCnt]);
        kCnt++;
    }
    Sptr<Node> query(int qL, int qR, int k){
        return query(qL, qR, rt[k]);
    }
};
int main(){
    int arr[MAXN], n;
    cin >> n;
    for (int i = 0 ; i < n; i++) cin >> arr[i];
    Sptr<PersistentSegmentTree> sol = _new(
        PersistentSegmentTree(arr, n));
}

```

## PersistentTreap

```

template <typename T>
struct _ptrCntr{
    T v; int c;
    _ptrCntr(const T& _v):v(_v){ c = 0;}
};
template <typename T>
struct Sptr{
    _ptrCntr<T> *p;
    T& operator->(){ return &p->v; }
    T& operator*(){ return p->v; }
    operator _ptrCntr<T>*(){ return p; }
    Sptr& operator = (const Sptr<T>& t){
        if (p && !--p->c) delete p;
        (p = t.p) && ++p->c;
        return *this;
    }
    Sptr(_ptrCntr<T> *t = 0) : p(t){ p && ++p->c; }
}

```

```

    Sptr(const Sptr& t) : p(t.p){ p && ++p->c; }
    ~Sptr(){ if (p && !--p->c) delete p;}
};
template <typename T>
inline Sptr<T> _new(const T& u){
    return Sptr<T>(new _ptrCntr<T>(u));
}
#define PNN pair<Sptr<Node>, Sptr<Node> >
#define MP make_pair
#define F first
#define S second
const int MAXK = 5e4 + 5;
int d;
struct PersistentTreap{
    struct Node{
        Sptr<Node> l, r;
        int sz;
        // data
        // tag
        Node() : l(NULL), r(NULL){
            sz = 1;
        }
    };
    Sptr<Node> ver[MAXK];
    int verCnt;
    PersistentTreap(){ verCnt = 0; }
    inline int size(Sptr<Node> &u){
        return u ? u->sz : 0;
    }
    inline void push(Sptr<Node> &u){
        // push function
        // copy a new one and modify on it
    }
    inline Sptr<Node> pull(Sptr<Node> &u){
        u->sz = 1 + size(u->l) + size(u->r);
        // pull function
        return u;
    }
    inline Sptr<Node> copy(Sptr<Node> &u){
        return _new(*u);
    }
    Sptr<Node> merge(Sptr<Node> &T1, Sptr<Node> &T2){
        if (!T1 || !T2) return T1 ? T1 : T2;
        Sptr<Node> res;
        if (rand() % (size(T1) + size(T2)) < size(T1)){
            push(T1);
            res = copy(T1);
            res->r = merge(T1->r, T2);
        }else{
            push(T2);
            res = copy(T2);
            res->l = merge(T1, T2->l);
        }
        return pull(res);
    }
    PNN split(Sptr<Node> &T, int k){
        if (!T) return MP(Sptr<Node>(NULL), Sptr<Node>(
            NULL));
        push(T);
        Sptr<Node> res = copy(T);
        if (size(T->l) < k){
            PNN tmp = split(T->r, k - 1 - size(T->l));
            res->r = tmp.F;
            return MP(pull(res), tmp.S);
        }else{
            PNN tmp = split(T->l, k);
            res->l = tmp.S;
            return MP(tmp.F, pull(res));
        }
    }
    /* create a version : verCnt++, ver[verCnt] = ver
       [verCnt - 1]
       * Treap operator
       * Query dont need to merge
       */
};
int main(){

```

## SparseTable

```

struct SparseTable{
    vector<vector<int> > data;
    SparseTable(int *arr, int n){
        int lgN = ceil(__lg(n)) + 1;
        data.resize(lgN);
        for (int i = 0 ; i < n ; i++) data[0].PB(arr[i]
        );
        for (int h = 1 ; h < lgN ; h++){
            int len = 1 << (h-1), i = 0;
            for (; i + len < n ; i++){
                data[h].PB(max(data[h-1][i], data[h-1][
                    i+len]));
            }
            if (!i) break;
            for (; i < n ; i++)
                data[h].PB(data[h-1][i]);
        }
    }
    int query(int l, int r){
        int h = __lg(r - l);
        int len = 1 << h;
        return op(data[h][l], data[h][r-len]);
    }
};

```

## pbds\_heap

```

#include <bits/extc++.h>
typedef __gnu_pbds::priority_queue<int> heap_t;
heap_t a, b;
int main() {
    a.clear(); b.clear();
    a.push(1); a.push(3);
    b.push(2); b.push(4);
    assert(a.top() == 3);
    assert(b.top() == 4);
    a.join(b);
    assert(a.top() == 4);
    assert(b.empty());
}

```

## pbds\_tree

```

#include <bits/extc++.h>
using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> set_t;
typedef cc_hash_table<int, int> umap_t;
int main() {
    set_t s; s.insert(12); s.insert(505);
    assert(*s.find_by_order(0) == 12);
    assert(s.find_by_order(2) == end(s));
    assert(s.order_of_key(12) == 0);
    assert(s.order_of_key(505) == 1);
    s.erase(12);
    assert(*s.find_by_order(0) == 505);
    assert(s.order_of_key(505) == 0);
}

```

## unordered\_map

```

struct Key {
    int F, S;
    Key() {}
    Key(int _x, int _y) : F(_x), S(_y) {}
}

```



```

    bool operator == (const Key &b) const {
        return tie(F, S) == tie(b.F, b.S);
    }
};
struct KeyHasher {
    size_t operator() (const Key &b) const {
        return k.F + k.S * 100000;
    }
};
typedef unordered_map<Key, int, KeyHasher> map_t;

```

## Geometry

### ClosestPair

```

template<typename T>
struct point{
    T x,y;
    point(){}
    point(const T&dx,const T&dy):x(dx),y(dy){}
    inline const point operator-(const point &b)const{
        return point(x-b.x,y-b.y);
    }
    inline const T dot(const point &b)const{
        return x*b.x+y*b.y;
    }
    inline const T abs2()const{/*向量長度的平方*/
        return dot(*this);
    }
    static bool x_cmp(const point<T>& a,const point<T>& b)
    ){
        return a.x<b.x;
    }
    static bool y_cmp(const point<T>& a,const point<T>& b)
    ){
        return a.y<b.y;
    }
};
#define INF LLONG_MAX/*預設是long long最大值*/
template<typename T>
T closest_pair(vector<point<T> >&v,vector<point<T> >&t,
    int l,int r){
    T dis=INF,tmd;
    if(l==r)return dis;
    int mid=(l+r)/2;
    if((tmd=closest_pair(v,t,l,mid))<dis)dis=tmd;
    if((tmd=closest_pair(v,t,mid+1,r))<dis)dis=tmd;
    t.clear();
    for(int i=l;i<=r;++i)
        if((v[i].x-v[mid].x)*(v[i].x-v[mid].x)<dis)t.
            push_back(v[i]);
    sort(t.begin(),t.end(),point<T>::y_cmp);/*如果用
        merge_sort的方式可以O(n)*/*
    for(int i=0;i<(int)t.size();++i)
        for(int j=1;j<=3&&i+j<(int)t.size();++j)
            if((tmd=(t[i]-t[i+j]).abs2())<dis)dis=tmd;
    return dis;
}
template<typename T>
inline T closest_pair(vector<point<T> > &v){
    vector<point<T> >t;
    sort(v.begin(),v.end(),point<T>::x_cmp);
    return closest_pair(v,t,0,v.size()-1);/*最近點對距離
        */
}

```

## Geometry

```

#define EPS 1e-12
#define LEFT_TOP POS(1000, 1000)
#define NO_INTERSECT POS(-1234, -1234)
#define PARALLEL POS(-1001, -1001)

```

```

#define COLINE POS(1234, 1234)
const double PI = acos(-1.0);

typedef double T;

class POS {
public:
    T x, y;
    POS(const T& x = 0, const T& y = 0) : x(x), y(y) {}
    POS(const POS& x) : x(x.x), y(x.y) {}

    bool operator==(const POS& rhs) const {
        return x == rhs.x && y == rhs.y;
    }

    POS& operator+=(const POS& rhs) {
        x += rhs.x;
        y += rhs.y;
        return *this;
    }

    POS operator -() {
        POS tmp(-x, -y);
        return tmp;
    }

    POS const operator+(const POS& rhs) const {
        return POS(*this) += rhs;
    }

    POS const operator-(const POS& rhs) const {
        POS tmp = rhs;
        tmp = -tmp;
        return POS(*this) += (tmp);
    }

    POS operator * (T c) const { return POS(x*c, y*c); }

    POS operator / (T c) const { return POS(x/c, y/c); }

    double dist(const POS& rhs) const {
        T tmp_x = x-rhs.x, tmp_y = y-rhs.y;
        return sqrt(tmp_x*tmp_x+tmp_y*tmp_y);
    }

    friend ostream& operator<<(ostream& out, const POS&
        pos) {
        out << pos.x << " " << pos.y;
        return out;
    }
};

T dot(POS p, POS q) { return p.x*q.x+p.y*q.y; }

T dist2(POS p, POS q) { return dot(p-q,p-q); }
double dist(POS p, POS q) { return sqrt(dist2(p, q)); }

// rotate a point CCW or CW around the origin
POS RotateCCW90(POS p) { return POS(-p.y,p.x); }

POS RotateCW90(POS p) { return POS(p.y,-p.x); }

POS RotateCCW(POS p, double t) {
    return POS(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(
        t));
}

// project point c onto line through a and b
// assuming a != b
POS ProjectPointLine(POS a, POS b, POS c) {
    return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

```



```

// project point c onto line segment through a and b
POS ProjectPointSegment(POS a, POS b, POS c) {
    double r = dot(b-a, b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
T DistancePointPlane(T x, T y, T z, T a, T b, T c, T d)
{
    return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

bool cmp_convex(const POS& lhs, const POS& rhs) {
    return (lhs.x < rhs.x) || ( (lhs.x == rhs.x) && (lhs.y < rhs.y) );
}

inline T cross(const POS& o, const POS& a, const POS& b)
{
    double value = (a.x-o.x)*(b.y-o.y) - (a.y-o.y)*(b.x-o.x);
    if (fabs(value) < EPS) return 0;
    return value;
}

void convex_hull(POS* points, POS* need, int& n) {
    sort(points, points+n, cmp_convex);
    int index = 0;
    for (int i = 0; i < n; ++i) {
        while (index >= 2 && cross(need[index-2], need[index-1], points[i]) <= 0) index--;
        need[index++] = points[i];
    }
    int half_point = index+1;
    for (int i = n-2; i >= 0; --i) {
        while (index >= half_point && cross(need[index-2], need[index-1], points[i]) <= 0) index--;
        need[index++] = points[i];
    } /* be careful that start point will appear in first and last in need array */
    n = index;
}

class LINE {
public:
    POS start, end, vec;
    double angle;
    LINE() {}
    LINE(const T& st_x, const T& st_y, const T& ed_x, const T& ed_y) :
        start(st_x, st_y), end(ed_x, ed_y), vec(end - start), angle(atan2(vec.x, vec.y)) {}

    LINE(const POS& start, const POS& end) :
        start(start), end(end), vec(end - start), angle(atan2(vec.x, vec.y)) {}

    LINE(const POS& end) : /* start point is origin */
        start(0, 0), end(end), vec(end), angle(atan2(vec.x, vec.y)) {}

    LINE(const T a, const T b, const T c) : /* given line by ax+by+c = 0 */
        start(0, 0), end(0, 0), vec(-b, a) {
            if (a == 0) {
                start.y = end.y = -c/b;
                end.x = -b;
            }
            else if (b == 0) {
                start.x = end.x = -c/a;
                end.y = a;
            }
            else if (c == 0) {
                end.x = -b; end.y = a;
            }
            else {
                start.y = -c/b; end.x = -c/a;
                vec.x = -c/a; vec.y = c/b;
            }
            angle = atan2(vec.x, vec.y);
        }

    LINE build_orthogonal(const POS& point) const {
        T c = -(vec.x*point.x + vec.y*point.y);
        return LINE(vec.x, vec.y, c);
    }

    T length2() const { /* square */
        T x = start.x - end.x, y = start.y - end.y;
        return x*x + y*y;
    }

    void modify(T x, T y) {
        this->end.x += x;
        this->end.y += y;
        this->vec.x += x;
        this->vec.y += y;
    }

    bool on_line(const POS& a) const {
        if (vec.x == 0) {
            if (start.x != a.x) return false;
            return true;
        }
        if (vec.y == 0) {
            if (start.y != a.y) return false;
            return true;
        }
        return fabs((a.x-start.x)/vec.x*vec.y + start.y - a.y) < EPS;
    }

    bool operator/(const LINE& rhs) const { /* to see if this line parallel to LINE rhs */
        return (vec.x*rhs.vec.y == vec.y*rhs.vec.x);
    }

    bool operator==(const LINE& rhs) const { /* to see if they are same line */
        return (*this/rhs) && (rhs.on_line(start));
    }

    POS intersect(const LINE& rhs) const {
        if (*this==rhs) return COLINE; /* return co-line */
        if (*this/rhs) return PARALLEL; /* return parallel */

        double A1 = vec.y, B1 = -vec.x, C1 = end.x*start.y - start.x*end.y;
        double A2 = rhs.vec.y, B2 = -rhs.vec.x, C2 = rhs.end.x*rhs.start.y - rhs.start.x*rhs.end.y;
        return POS( (B2*C1-B1*C2)/(A2*B1-A1*B2), (A1*C2-A2*C1)/(A2*B1-A1*B2) ); /* sometimes has -0 */
    }

    double dist(const POS& a) const {
        return fabs(vec.y*a.x - vec.x*a.y + vec.x*start.y - vec.y*start.x)/sqrt(vec.y*vec.y+vec.x*vec.x);
    }

    double dist(const LINE& rhs) const {
        POS intersect_point = intersect(rhs);
        if (intersect_point == PARALLEL) {
            return dist(rhs.start);
        }
    }
}

```

```

    }
    return 0;
}

friend ostream& operator<<(ostream& out, const LINE
& line) {
    out << line.start << "-->" << line.end << " vec
        : " << line.vec;
    return out;
}
};

POS ComputeCircleCenter(POS a, POS b, POS c) {
    POS ret;
    double A1 = b.x - a.x, B1 = b.y - a.y, C1 = (A1 * A1
        + B1 * B1) / 2;
    double A2 = c.x - a.x, B2 = c.y - a.y, C2 = (A2 * A2
        + B2 * B2) / 2;
    double D = A1 * B2 - A2 * B1;
    ret.x = a.x + (C1 * B2 - C2 * B1) / D;
    ret.y = a.y + (A1 * C2 - A2 * C1) / D;
    return ret;
}

class LINESEG : public LINE {
public:
    LINESEG() : LINE(POS(0, 0)) {}
    LINESEG(const LINE& input) : LINE(input) {}
    LINESEG(const POS& start, const POS& end) : LINE(
        start, end) {}

    bool on_lineseg(const POS& a) const {
        if (!on_line(a)) return false;
        bool first, second;
        if (vec.x >= 0) first = (a.x >= start.x)&&(a.x
            <= end.x);
        else first = (a.x <= start.x)&&(a.x >= end.x);
        if (vec.y >= 0) second = (a.y >= start.y)&&(a.y
            <= end.y);
        else second = (a.y <= start.y)&&(a.y >= end.y);
        return first&&second;
    }

    bool operator==(const LINESEG& rhs) const {
        return ( (rhs.start == start && rhs.end == end)
            ||
            (rhs.start == end && rhs.end == start) );
    }

    bool operator==(const LINE& rhs) const {
        return this->LINE::operator==(rhs);
    }

    T dot(const LINESEG& rhs) const {
        return vec.x*rhs.vec.x + vec.y*rhs.vec.y;
    }

    T cross(const LINESEG& rhs) const {
        return vec.x*rhs.vec.y - vec.y*rhs.vec.x;
    }

    bool clockwise(const LINE& a) const { /* to see if
        LINE a is in b's clockwise way */
        return cross(a) > 0;
    }

    double dist(const POS& a) const {
        double ortho_dist = this->LINE::dist(a);
        LINE ortho_line = build_orthogonal(a);
        POS intersect_point = this->LINE::intersect(
            ortho_line);
        if (on_lineseg(intersect_point)) return
            ortho_dist;
        else return min(a.dist(this->start), a.dist(
            this->end));
    }
}

```

```

double dist(const LINE& line) const {
    POS intersect_point = this->LINE::intersect(
        line);
    if (intersect_point == COLINE) return 0;
    if (intersect_point == PARALLEL) return dist(
        line.start);
    if (on_lineseg(intersect_point)) return 0;
    return min(line.dist(start), line.dist(end));
}

double dist(const LINESEG& line) const {
    return min( min(dist(line.start), dist(line.end
        )),
        min(line.dist(start), line.dist(end
        )) );
}

POS intersect(const LINESEG& rhs) const {
    LINE a1b1(start, rhs.start);
    LINE a1b2(start, rhs.end);
    LINE b1a1(rhs.start, start);
    LINE b1a2(rhs.start, end);

    POS tmp(this->LINE::intersect(rhs));

    if (tmp == COLINE) {
        if ( (start==rhs.start) && (!rhs.on_lineseg
            (end)) && (!on_lineseg(rhs.end)) )
            return start;
        if ( (start==rhs.end) && (!rhs.on_lineseg(
            end)) && (!on_lineseg(rhs.start)) )
            return start;
        if ( (end==rhs.start) && (!rhs.on_lineseg(
            start)) && (!on_lineseg(rhs.end)) )
            return end;
        if ( (end==rhs.end) && (!rhs.on_lineseg(
            start)) && (!on_lineseg(rhs.start)) )
            return end;
        if (on_lineseg(rhs.start) || on_lineseg(rhs
            .end) || rhs.on_lineseg(start) || rhs.
            on_lineseg(end)) return COLINE;
        return NO_INTERSECT;
    }

    bool intersected = ( (cross(a1b1)*cross(a1b2)
        <=0) && (rhs.cross(b1a1)*rhs.cross(b1a2)
        <=0) );
    if (!intersected) return NO_INTERSECT;
    if (!on_lineseg(tmp) || !rhs.on_lineseg(tmp))
        return NO_INTERSECT;
    return tmp;
}

};

inline bool cmp_half_plane(const LINE &a, const LINE &b)
{
    if(fabs(a.angle-b.angle) < EPS) return cross(a.
        start, a.end, b.start) < 0;
    return a.angle > b.angle;
}

void half_plane_intersection(LINE* a, LINE* need, POS*
    answer, int &n){
    int m = 1, front = 0, rear = 1;
    sort(a, a+n, cmp_half_plane);
    for(int i = 1; i < n; ++i){
        if( fabs(a[i].angle-a[m-1].angle) > EPS ) a[m
            ++] = a[i];
    }
    need[0] = a[0], need[1] = a[1];
    for(int i = 2; i < m; ++i){
        while (front<rear&&cross(a[i].start, a[i].end,
            need[rear].intersect(need[rear-1]))<0) rear
            --;
        while (front<rear&&cross(a[i].start, a[i].end,
            need[front].intersect(need[front+1]))<0)
            front++;
    }
}

```

```

    need[++rear] = a[i];
}
while (front < rear && cross(need[front].start, need[
front].end, need[rear].intersect(need[rear-1]))
< 0) rear--;
while (front < rear && cross(need[rear].start, need[rear
].end, need[front].intersect(need[front+1])) < 0)
front++;
if (front == rear) return;

n = 0;
for (int i = front; i < rear; ++i) answer[n++] = need[i
].intersect(need[i+1]);
if (rear > front+1) answer[n++] = need[front].
intersect(need[rear]);
}

void rotating_calipers(int& ans, POS* need, int& n) {
--n;
if (n == 2) {
ans = need[0].dist(need[1]);
return;
}

int now = 2;
for (int i = 0; i < n; ++i) {
LINE target(need[i], need[i+1]);
double pre = target.dist(need[now]);
for (; now != i; now = (now+1)%n) {
double tmp = target.dist(need[now]);
if (tmp < pre) break;
pre = tmp;
}
now = (now-1+n)%n;
ans = max(ans, pre);
}
}

// determine if point is in a possibly non-convex
// polygon (by William
// Randolph Franklin); returns 1 for strictly interior
// points, 0 for
// strictly exterior points, and 0 or 1 for the
// remaining points.
// Note that it is possible to convert this into an *
// exact* test using
// integer arithmetic by taking care of the division
// appropriately
// (making sure to deal with signs properly) and then
// by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<POS> &p, POS q) {
bool c = 0;
for (int i = 0; i < p.size(); i++) {
int j = (i+1)%p.size();
if ((p[i].y <= q.y && q.y < p[j].y ||
p[j].y <= q.y && q.y < p[i].y) &&
q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y)
/ (p[j].y - p[i].y))
c = !c;
}
return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<POS> &p, POS q) {
for (int i = 0; i < p.size(); i++)
if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size
()], q), q) < EPS)
return true;
return false;
}

// compute intersection of line through points a and b
// with
// circle centered at c with radius r > 0

```

```

vector<POS> CircleLineIntersection(POS a, POS b, POS c,
double r) {
vector<POS> ret;
b = b-a;
a = a-c;
double A = dot(b, b);
double B = dot(a, b);
double C = dot(a, a) - r*r;
double D = B*B - A*C;
if (D < -EPS) return ret;
ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
if (D > EPS)
ret.push_back(c+a+b*(-B-sqrt(D))/A);
return ret;
}

// compute intersection of circle centered at a with
// radius r
// with circle centered at b with radius R
vector<POS> CircleCircleIntersection(POS a, POS b,
double r, double R) {
vector<POS> ret;
double d = sqrt(dist2(a, b));
if (d > r+R || d+min(r, R) < max(r, R)) return ret;
double x = (d*d-R*R+r*r)/(2*d);
double y = sqrt(r*r-x*x);
POS v = (b-a)/d;
ret.push_back(a+v*x + RotateCCW90(v)*y);
if (y > 0)
ret.push_back(a+v*x - RotateCCW90(v)*y);
return ret;
}

// This code computes the area or centroid of a (
// possibly nonconvex)
// polygon, assuming that the coordinates are listed in
// a clockwise or
// counterclockwise fashion. Note that the centroid is
// often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<POS> &p) {
double area = 0;
for (int i = 0; i < p.size(); i++) {
int j = (i+1) % p.size();
area += p[i].x*p[j].y - p[j].x*p[i].y;
}
return area / 2.0;
}

double ComputeArea(const vector<POS> &p) {
return fabs(ComputeSignedArea(p));
}

POS ComputeCentroid(const vector<POS> &p) {
POS c(0,0);
double scale = 6.0 * ComputeSignedArea(p);
for (int i = 0; i < p.size(); i++) {
int j = (i+1) % p.size();
c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y)
;
}
return c / scale;
}

// tests whether or not a given polygon (in CW or CCW
// order) is simple
bool IsSimple(const vector<POS> &p) {
for (int i = 0; i < p.size(); i++) {
for (int k = i+1; k < p.size(); k++) {
int j = (i+1) % p.size();
int l = (k+1) % p.size();
if (i == 1 || j == k) continue;
LINESEG l1 = LINESEG(p[i], p[j]), l2 = LINESEG(p[
k], p[l]);
POS res = l1.intersect(l2);
if (!res == NO_INTERSECT)
return false;
}
}
}

```

```

        //if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        //    return false;
    }
}
return true;
}

```

## DP

### KnapsackLimit

```

int v[100 + 1], w[100 + 1], m[100 + 1];
int dp[10000 + 1];

int knapsack(int N, int W)
{
    int ans = 0;
    for(int i = 0; i < N; ++i) cin >> v[i] >> w[i] >> m[i];
    for(int i = 0; i < N; ++i)
    {
        for(int j = 0; m[i] > 0; ++j)
        {
            int take = min(m[i], (1 << j));
            m[i] -= take;
            for(int k = W; k >= take * w[i]; --k) dp[k]
                = max(dp[k], dp[k - take * w[i]] + take * v[i]);
        }
    }
    for(int i = W; i >= 0; --i) ans = max(ans, dp[i]);
    return ans;
}

```

### Digit\_Count

```

int DP[10000][10];
void Build() {
    int i, tn;
    memset(DP, 0, sizeof(DP));
    for(i = 1; i < 10000; i++) {
        memcpy(DP[i], DP[i-1], 40);
        tn = i;
        while(tn) DP[i][tn%10]++, tn /= 10;
    }
}

```

## Graph

### BCC

```

const int MAXN = 1e3 + 5;
struct Graph {
    int V;
    struct Node : vector<Node*> { // if it is a cut,
        then bcc is not true;
        int dfn, low, bcc;
        bool is_cut;
        Node () { clear(); dfn = low = bcc = -1; is_cut = false; }
    } _memN[MAXN], *node[MAXN];
    Graph(int _V) : V(_V) {
        for (int i = 0; i < V; i++)
            node[i] = _memN + i;
    }
    void addEdge(int u, int v) {
        node[u]->push_back(node[v]);
        node[v]->push_back(node[u]);
    }
}

```

```

int stamp, bcc_num, child;
stack<Node*> stk;
vector<Node*> BCC[MAXN];
void findBCC() {
    stamp = bcc_num = child = 0;
    Tarjan(node[0], NULL);
}
void Tarjan(Node *u, Node *pa) {
    u->low = u->dfn = stamp++;
    stk.push(u);
    for (auto to : *u) {
        if (!to->dfn) {
            Tarjan(to, u); child++;
            u->low = min(u->low, to->low);
            if (u->dfn <= to->low) {
                u->is_cut = true;
                BCC[bcc_num].clear();
                Node *v;
                do{
                    v = stk.top(); stk.pop();
                    BCC[v->bcc = bcc_num].push_back(v);
                }while (v != to);
                u->bcc = bcc_num;
                BCC[bcc_num++].push_back(u);
            }
        }else if (to->dfn < u->dfn && to != pa)
            u->low = min(u->low, to->dfn);
    }
    if (!pa && child < 2) u->is_cut = false;
}
int solve() {
    findBCC();
    int out_degree[MAXN]; memset(out_degree, 0, sizeof(out_degree));
    for (int _bcc = 0; _bcc < bcc_num; _bcc++) {
        bool all_cut = true, inBCC[MAXN];
        memset(inBCC, false, sizeof(inBCC));
        for (auto u : BCC[_bcc]) {
            inBCC[u - _memN] = true;
            if (!u->is_cut)
                all_cut = false;
        }
        if (all_cut) continue;
        for (auto u : BCC[_bcc]) {
            for (auto to : *u) {
                if (inBCC[to - _memN]) continue;
                out_degree[_bcc]++;
            }
        }
    }
    int ans = 0;
    for (int i = 0; i < bcc_num; i++)
        if (out_degree[i] == 1)
            ans++;
    return (ans + 1) >> 1;
}

};
int main() {
    int n, m; cin >> n >> m;
    Graph *G = new Graph(n);
    while (m--) {
        int u, v; cin >> u >> v;
        G->addEdge(u - 1, v - 1);
    }
    cout << G->solve() << '\n';
}

```

### Blossom

```

const int MAXN = 250 + 5;
const int MAXM = MAXN * MAXN / 2;
struct Graph {
    struct Node; struct Edge;
}

```

```

int V;
struct Node : vector<Edge*> {
    Node *p, *s, *m;
    int S, v;
    Node() {
        clear(), S = v = -1, s = p = m = NULL;
    }
} _memN[MAXN], *node[MAXN];
struct Edge {
    Node *v;
    Edge(Node *v = NULL) : v(v) {}
} _memE[MAXM], *ptrE;
Graph(int _V) : V(_V) {
    for (int i = 0 ; i < V ; i++)
        node[i] = _memN + i;
    ptrE = _memE;
}
void addEdge(int u, int v) {
    node[u]->PB(new (ptrE++) Edge(node[v]));
    node[v]->PB(new (ptrE++) Edge(node[u]));
}
inline int maxMatch() {
    int ans = 0;
    for (int i = 0 ; i < V ; i++)
        if (!node[i]->m && bfs(node[i]))
            ans++;
    return ans;
}
inline bool bfs(Node *u) {
    for (int i = 0 ; i < V ; i++)
        node[i]->s = node[i], node[i]->S = -1;
    queue<Node*> q; q.push(u), u->S = 0;
    while (q.size()) {
        u = q.front(); q.pop();
        for (auto e : *u) {
            Node *v = e->v;
            if (!v->S) {
                v->p = u; v->S = 1;
                if (!v->m) return augment(u, v);
                q.push(v->m), v->m->S = 0;
            } else if (!v->S && v->s != u->s) {
                Node *l = LCA(v->s, u->s);
                flower(v, u, l, q);
                flower(u, v, l, q);
            }
        }
    }
    return false;
}
inline bool augment(Node *u, Node *v) {
    for (Node *l; u; v = l, u = v ? v->p : NULL) {
        l = u->m;
        u->m = v;
        v->m = u;
    }
    return true;
}
inline Node* LCA(Node *u, Node *v) {
    static int t = 0;
    for (++t ; ; swap(u, v)) {
        if (!u) continue;
        if (u->v == t) return u;
        u->v = t;
        u = u->m; if (!u) continue;
        u = u->p; if (!u) continue;
        u = u->s;
    }
}
inline void flower(Node *u, Node *v, Node *l, queue
<Node*> &q) {
    while (u->s != l) {
        u->p = v;
        v = u->m;
        if (v->S == 1) q.push(v), v->S = 0;
        u->s = v->s = l;
        u = v->p;
    }
}

```

```

    }
};

```

## CutBridge

```

const int MAXN = 1e2 + 5;
struct Graph{
    struct Node : vector<Node*> {
        int low, dfn;
        bool is_cut;
        Node *pa;
        Node () {
            clear(), low = dfn = -1;
            is_cut = false; pa = NULL;
        }
    } _memN[MAXN], *node[MAXN];
    int V;
    Graph(int _V) : V(_V) {
        for (int i = 0 ; i < V ; i++)
            node[i] = _memN + i;
    }
    void addEdge(int u, int v){
        node[u]->push_back(node[v]);
        node[v]->push_back(node[u]);
    }

    int stamp;
    int findCutAndBridge(){
        stamp = 0; int root_son = 0;
        int ans = 0;
        Tarjan(node[0], NULL);
        for (int i = 1 ; i < V ; i++){
            Node *pa = node[i]->pa;
            if (pa == node[0]) root_son++;
            else {
                if (node[i]->low >= pa->dfn)
                    pa->is_cut = true;
            }
        }
        if (root_son > 1) node[0]->is_cut = true;
        for (int i = 0 ; i < V ; i++)
            if (node[i]->is_cut);
            /* node[i] is a cut */
        for (int i = 0 ; i < V ; i++){
            Node *pa = node[i]->pa;
            if (pa && node[i]->low > pa->dfn);
            /* pa and node[i] is a bridge */
        }
    }
    void Tarjan(Node *u, Node *pa){
        u->pa = pa;
        u->dfn = u->low = stamp++;
        for (auto to : *u){
            if (!to->dfn) {
                Tarjan(to, u);
                u->low = min(u->low, to->low);
            } else if (pa != to)
                u->low = min(u->low, to->dfn);
        }
    }
};

```

## Dijkstra

```

typedef struct Edge {
    int v; LL w;
    bool operator > (const Edge &b) const {
        return w > b.w;
    }
} State;
const LL INF = 0x3f3f3f3f3f3f3f3fLL;
void Dijkstra(int n, vector<vector<Edge> > &G, vector<
LL> &d, int s, int t = -1) {

```

```

static priority_queue<State, vector<State>, greater
<State> > pq;
d.clear(); d.resize(n);
while (pq.size()) pq.pop();
for (auto &num : d) num = INF;
d[s] = 0; pq.push({s, d[s]});
while (pq.size()) {
    auto p = pq.top(); pq.pop();
    int u = p.v;
    if (d[u] < p.w) continue;
    if (u == t) return ;
    for (auto &e : G[u]) {
        if (d[e.v] > d[u] + e.w) {
            d[e.v] = d[u] + e.w;
            pq.push({e.v, d[e.v]});
        }
    }
}
}
}

```

## MaximumClique

```

const int MAXN = 105;
int best;
int n;
int num[MAXN];
int path[MAXN];
int G[MAXN][MAXN];

bool dfs( int *adj, int total, int cnt ){
    int t[MAXN];
    if (total == 0){
        if( best < cnt ){
            best = cnt;
            return true;
        }
        return false;
    }
    for(int i = 0; i < total; i++){
        if( cnt+(total-i) <= best ) return false;
        if( cnt+num[adj[i]] <= best ) return false;
        int k=0;
        for(int j=i+1; j<total; j++)
            if(G[ adj[i] ][ adj[j] ])
                t[k++] = adj[j];
        if (dfs(t, k, cnt+1)) return true;
    }
    return false;
}

int MaximumClique(){
    int adj[MAXN];
    if (n <= 0) return 0;
    best = 0;
    for(int i = n-1; i >= 0; i--){
        int k=0;
        for(int j = i+1; j < n; j++)
            if (G[i][j]) adj[k++] = j;
        dfs( adj, k, 1 );
        num[i] = best;
    }
    return best;
}

```

## MinMeanCycle

```

const int MAXN = 55;
const double INF = 0x3f3f3f3f;
const double EPS = 1e-4;
double min_mean_cycle(vector<vector<pii> > &G) {
    int n = G.size(); G.resize(n + 1);
    for (int i = 0 ; i < n ; i++)
        G[n].push_back(MP(i, 0));
    double d[MAXN][MAXN];

```

```

int s = n++;
for (int i = 0 ; i <= n ; i++)
    for (int j = 0 ; j < n ; j++)
        d[i][j] = INF;
d[0][s] = 0;
for (int k = 0 ; k < n ; k++)
    for (int i = 0 ; i < n ; i++)
        for (auto p : G[i])
            if (d[k][i] + p.S < d[k + 1][p.F])
                d[k + 1][p.F] = d[k][i] + p.S;

double ans = INF;
for (int i = 0 ; i < n ; i++) {
    if (fabs(d[n][i] - INF) < EPS) continue;
    double maxW = -INF;
    for (int k = 0 ; k < n - 1 ; k++) {
        maxW = max(maxW, (d[n][i] - d[k][i]) / (n - k));
    }
    ans = min(ans, maxW);
}
return ans;
}

int main() {
    int kase = 0;
    int t; cin >> t; while (t--) {
        cout << "Case #" << ++kase << ": ";
        int n, m; cin >> n >> m;
        vector<vector<pii> > G(n);
        while (m--) {
            int a, b, c;
            cin >> a >> b >> c;
            a--, b--;
            G[a].push_back(MP(b, c));
        }
        double ans = min_mean_cycle(G);
        if (fabs(ans - INF) < EPS) cout << "No cycle found.\n";
        else printf("%f\n", ans + EPS);
    }
}

```

## SCC

```

const int MAXN = 1e5 + 5;
struct Graph{
    struct Node : vector<Node*> {
        int dfn, low, scc;
        bool in_stk;
        Node () { clear();
            dfn = low = scc = -1;
            in_stk = false;
        }
    } _memN[MAXN], *node[MAXN];
    int V;
    Graph(int _V) : V(_V) {
        for (int i = 0 ; i < V ; i++)
            node[i] = _memN + i;
    }
    void addEdge(int u, int v){
        node[u]->push_back(node[v]);
    }

    int stamp, scc_num; stack<Node*> stk;
    int findSCC(){
        stamp = scc_num = 0;
        for (auto u : node)
            if (!u->dfn)
                Tarjan(u);
        return scc_num;
    }
    void Tarjan(Node *u) {
        u->dfn = u->low = stamp++;
        stk.push(u); u->in_stk = true;
        for (auto to : *u){
            if (!to->dfn) {

```

```

        Tarjan(to);
        u->low = min(u->low, to->low);
    }else if (to->in_stk)
        u->low = min(u->low, to->dfn);
    }
    if (u->dfn == u->low){
        Node *v;
        do {
            v = stk.top(); stk.pop();
            v->scc = scc_num;
            v->in_stk = false;
        }while (v != u);
        scc_num++;
    }
};

```

## KM

```

const int MAX_N = 400 + 10;
const ll INF64 = 0x3f3f3f3f3f3f3fLL;
int n1, nr;
int pre[MAX_N];
ll slack[MAX_N];
ll W[MAX_N][MAX_N];
ll lx[MAX_N], ly[MAX_N];
int mx[MAX_N], my[MAX_N];
bool vx[MAX_N], vy[MAX_N];
void augment(int u) {
    if(!u) return;
    augment(mx[pre[u]]);
    mx[pre[u]] = u;
    my[u] = pre[u];
}
inline void match(int x) {
    queue<int> que;
    que.push(x);
    while(1) {
        while(!que.empty()) {
            x = que.front();
            que.pop();
            vx[x] = 1;
            REP1(y, 1, nr) {
                if(vy[y]) continue;
                ll t = lx[x] + ly[y] - W[x][y];
                if(t > 0) {
                    if(slack[y] >= t) slack[y] = t,
                        pre[y] = x;
                    continue;
                }
                pre[y] = x;
                if(!my[y]) {
                    augment(y);
                    return;
                }
                vy[y] = 1;
                que.push(my[y]);
            }
        }
        ll t = INF64;
        REP1(y, 1, nr) if(!vy[y]) t = min(t, slack[y]);
        REP1(x, 1, n1) if(vx[x]) lx[x] -= t;
        REP1(y, 1, nr) {
            if(vy[y]) ly[y] += t;
            else slack[y] -= t;
        }
        REP1(y, 1, nr) {
            if(vy[y] || slack[y]) continue;
            if(!my[y]) {
                augment(y);
                return;
            }
            vy[y] = 1;
            que.push(my[y]);
        }
    }
}

```

```

    }
}
int main() {
    int m;
    RI(n1, nr, m);
    nr = max(n1, nr);
    while(m--) {
        int x, y;
        ll w;
        RI(x, y, w);
        W[x][y] = w;
        lx[x] = max(lx[x], w);
    }
    REP1(i, 1, n1) {
        REP1(x, 1, n1) vx[x] = 0;
        REP1(y, 1, nr) vy[y] = 0, slack[y] = INF64;
        match(i);
    }
    ll ans = 0LL;
    REP1(x, 1, n1) ans += W[x][mx[x]];
    PL(ans);
    REP1(x, 1, n1) printf("%d%c", W[x][mx[x]] ? mx[x] : 0, "\n"[x == n1]);
    return 0;
}

```

## GridMST

```

#define REP(i,n) for(int i=0;i<n;i++)
const int N=200100;
int n,m;
struct PT {int x,y,z,w,id;}p[N];
inline int dis(const PT &a,const PT &b){return abs(a.x-b.x)+abs(a.y-b.y);}
inline bool cpx(const PT &a,const PT &b){return a.x!=b.x? a.x>b.x:a.y>b.y;}
inline bool cpz(const PT &a,const PT &b){return a.z<b.z;}
struct E{int a,b,c;}e[8*N];
bool operator<(const E&a,const E&b){return a.c<b.c;}
struct Node{
    int L,R,key;
}node[4*N];
int s[N];
int F(int x){return s[x]==x?s[x]:s[x]=F(s[x]);}
void U(int a,int b){s[F(b)]=F(a);}
void init(int id,int L,int R) {
    node[id]=(Node){L,R,-1};
    if(L==R)return;
    init(id*2,L,(L+R)/2);
    init(id*2+1,(L+R)/2+1,R);
}
void ins(int id,int x) {
    if(node[id].key==-1 || p[node[id].key].w>p[x].w)node[id].key=x;
    if(node[id].L==node[id].R)return;
    if(p[x].z<=(node[id].L+node[id].R)/2)ins(id*2,x);
    else ins(id*2+1,x);
}
int Q(int id,int L,int R){
    if(R<node[id].L || L>node[id].R)return -1;
    if(L<=node[id].L && node[id].R<=R)return node[id].key;
    int a=Q(id*2,L,R),b=Q(id*2+1,L,R);
    if(b==-1 || (a!=-1 && p[a].w<p[b].w)) return a;
    else return b;
}
void calc() {
    REP(i,n) {
        p[i].z=p[i].y-p[i].x;
        p[i].w=p[i].x+p[i].y;
    }
    sort(p,p+n,cpz);
    int cnt=0,j,k;
    for(int i=0;i<n;i=j){
        for(j=i+1;p[j].z==p[i].z && j<n;j++);
    }
}

```



```

    for(k=i,cnt++;k<j;k++)p[k].z=cnt;
}
init(1,1,cnt);
sort(p,p+n,cpx);
REP(i,n) {
    j=Q(1,p[i].z,cnt);
    if(j!=-1)e[m++]=(E){p[i].id,p[j].id,dis(p[i],p[j])};
    ins(1,i);
}
}
LL MST() {
    LL r=0;
    sort(e,e+m);
    REP(i,m) {
        if(F(e[i].a)==F(e[i].b))continue;
        U(e[i].a,e[i].b);
        r+=e[i].c;
    }
    return r;
}
int main(){
    m = 0;
    scanf("%d",&n);
    REP(i,n) {
        scanf("%d%d",&p[i].x,&p[i].y);
        p[i].id=s[i]=i;
    }
    calc();
    REP(i,n)p[i].y= -p[i].y;
    calc();
    REP(i,n)swap(p[i].x,p[i].y);
    calc();
    REP(i,n)p[i].x=-p[i].x;
    calc();
    printf("%lld\n",MST());
    return 0;
}

```

## Math

### bigN

```

const int BASE = 1e9 + 0.5;
const int WIDTH = log10(BASE) + 0.5;
template <typename T>
inline string to_string(const T &x) {
    stringstream ss;
    return ss << x, ss.str();
}
typedef long long LL;
struct bigN : vector<LL> {
    bool neg;
    bigN(string s) {
        if (s.empty()) return ;
        if (s[0] == '-') neg = true, s = s.substr(1);
        else neg = false;
        for (int i = s.size() - 1 ; i >= 0 ; i -= WIDTH) {
            LL t = 0;
            for (int j = max(0, i - WIDTH + 1) ; j <= i ; j++)
                t = t * 10 + s[j] - '0';
            push_back(t);
        }
        trim();
    }
    template <typename T>
    bigN(const T &x) : bigN(to_string(x)) {}
    bigN() : neg(false) {}
    friend istream& operator >> (istream &in, bigN &b)
    {
        string s;
        return in >> s, b = s, in;
    }
}

```

```

}
friend ostream& operator << (ostream &out, const
    bigN &b) {
    if (b.neg) out << '-';
    out << (b.empty() ? 0 : b.back());
    for (int i = b.size() - 2 ; i >= 0 ; i--)
        out << setw(WIDTH) << setfill('0') << b[i];
    return out;
}
inline void trim() {
    while (size() && !back()) pop_back();
    if (empty()) neg = false;
}
bigN operator - () const {
    bigN res = *this;
    return res.neg = !neg, res.trim(), res;
}
bigN operator + (const bigN &b) const {
    if (neg) return -(*this) + (-b);
    if (b.neg) return *this - (-b);
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (int i = 0 ; i < b.size() ; i++) res[i] +=
        b[i];
    return res.carry(), res.trim(), res;
}
bigN operator - (const bigN &b) const {
    if (neg) return -(*this) - (-b);
    if (b.neg) return *this + (-b);
    if (abscmp(b) < 0) return -(b-*this);
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (int i = 0 ; i < b.size() ; i++) res[i] -=
        b[i];
    return res.carry(), res.trim(), res;
}
inline void carry() {
    for (int i = 0 ; i < size() ; i++) {
        if (at(i) >= 0 && at(i) < BASE) continue;
        if (i + 1 == size()) push_back(0);
        int r = at(i) % BASE;
        if (r < 0) r += BASE;
        at(i + 1) += (at(i) - r) / BASE;
        at(i) = r;
    }
}
int abscmp(const bigN &b) const {
    if (size() > b.size()) return 1;
    if (size() < b.size()) return -1;
    for (int i = size() - 1 ; i >= 0 ; i--) {
        if (at(i) > b[i]) return 1;
        if (at(i) < b[i]) return -1;
    }
    return 0;
}
bigN operator * (const bigN &b) const {
    bigN res;
    res.neg = neg != b.neg;
    res.resize(size() + b.size());
    for (int i = 0 ; i < size() ; i++)
        for (int j = 0 ; j < b.size() ; j++)
            if ((res[i + j] += at(i) * b[j]) >=
                BASE) {
                res[i + j + 1] += res[i + j] / BASE;
                res[i + j] %= BASE;
            }
    return res.trim(), res;
}
bigN operator / (const bigN &b) const {
    int norm = BASE / (b.back() + 1);
    bigN x = abs() * norm;
    bigN y = b.abs() * norm;
    bigN q, r;
    q.resize(x.size());
    for (int i = x.size() - 1 ; i >= 0 ; i--) {
        r = r * BASE + x[i];
    }
}

```

```

        int s1 = r.size() <= y.size() ? 0 : r[y.size()];
        int s2 = r.size() < y.size() ? 0 : r[y.size() - 1];
        int d = (LL(BASE) * s1 + s2) / y.back();
        r = r - y * d;
        while (r.neg) r = r + y, d--;
        q[i] = d;
    }
    q.neg = neg != b.neg;
    return q.trim(), q;
}
bigN abs() const {
    bigN res = *this;
    return res.neg = false, res;
}
bigN operator % (const bigN &b) const {
    return *this - (*this / b) * b;
}
int cmp(const bigN &b) const {
    if (neg != b.neg) return neg ? -1 : 1;
    return neg ? -abscmp(b) : abscmp(b);
}
bool operator < (const bigN &b) const { return cmp(b) < 0; }
bool operator > (const bigN &b) const { return cmp(b) > 0; }
bool operator <= (const bigN &b) const { return cmp(b) <= 0; }
bool operator >= (const bigN &b) const { return cmp(b) >= 0; }
bool operator == (const bigN &b) const { return cmp(b) == 0; }
bool operator != (const bigN &b) const { return cmp(b) != 0; }
template <typename T>
operator T() {
    stringstream ss;
    ss << *this;
    T res;
    return ss >> res, res;
}
};

```

## BSGS

```

LL extgcd(LL a, LL b, LL &x, LL &y){
    if (!b) return x = 1, y = 0, a;
    LL res = extgcd(b, a%b, y, x);
    return y -= a / b * x, res;
}
LL modInv(LL a, LL m){
    LL x, y, d = extgcd(a, m, x, y);
    return d == 1 ? (x + m) % m : -1;
}
LL BSGS(LL B, LL N, LL P) { // B^L = N mod P
    unordered_map<LL, int> R;
    LL sq = (LL)(sqrt(P) + 1e-6), t = 1;
    for (int i = 0; i < sq; i++) {
        if (t == N) return i;
        if (!R.count(t)) R[t] = i;
        t = (t * B) % P;
    }
    LL f = modInv(t, P);
    for (int i = 0; i <= sq + 1; i++) {
        if (R.count(N)) return i * sq + R[N];
        N = (N * f) % P;
    }
    return -1;
}
int main() {
    int a, b, n; while (cin >> a >> b >> n) {
        LL L = BSGS(a, b, n);
        if (L == -1) cout << "NOT FOUND\n";
        else cout << L << '\n';
    }
}

```

```

    }
}

```

## CRT

```

LL extgcd(LL a, LL b, LL &x, LL &y){
    LL d = a;
    if (b != 0){
        d = extgcd(b, a % b, y, x);
        y -= (a / b) * x;
    }else x = 1, y = 0;
    return d;
}
LL modInv(LL a, LL m){
    LL x, y, d = extgcd(a, m, x, y);
    return d == 1 ? (m + x % m) % m : -1;
}
LL gcd(LL x, LL y){ return y ? gcd(y, x % y) : x; }
typedef pair<LL, LL> pLL;
pLL CRT(LL *A, LL *B, LL *M, int n){
    // A[i]x = B[i] (mod M[i]); F : ans, S : lcm of M;
    LL x = 0, m = 1;
    for (int i = 0; i < n; i++){
        LL a = A[i] * m, b = B[i] - A[i] * x, d = gcd(M[i], a);
        if (b % d) return pLL(0, -1);
        LL t = b / d * modInv(a / d, M[i] / d) % (M[i] / d);
        x = x + m * t;
        m *= M[i] / d;
    }
    x = (x % m + m) % m;
    return pLL(x, m);
}

```

## ExtgcdModInv

```

LL extgcd(LL a, LL b, LL &x, LL &y){
    if (!b) return x = 1, y = 0, a;
    LL res = extgcd(b, a%b, y, x);
    return y -= a / b * x, res;
}
LL modInv(LL a, LL m){
    LL x, y, d = extgcd(a, m, x, y);
    return d == 1 ? (x + m) % m : -1;
}

```

## FFT

```

typedef double D;
const D PI = acos(-1.0);
struct C{
    D x, y; C(){x=0, y=0;} C(D x, D y):x(x), y(y){}
    C operator+(const C&c){return C(x+c.x, y+c.y);}
    C operator-(const C&c){return C(x-c.x, y-c.y);}
    C operator*(const C&c){return C(x*c.x - y*c.y, x*c.y + y*c.x);}
};
void FFT(vector<C> &c, int t) {
    int n = c.size();
    for (int i = 1, j = 0; i < n; i++) {
        for (int k = (n >> 1); k > (j ^ k); k >>= 1);
        if (i < j) swap(c[i], c[j]);
    }
    for (int m = 2; m <= n; m <= 1) {
        C wm(cos(2 * PI * t / m), sin(2 * PI * t / m));
        for (int k = 0; k < n; k += m) {
            C w(1.0, 0.0);
            for (int j = 0; j < (m >> 1); j++) {
                C u = c[k + j];
                C t = w * c[k + j + (m >> 1)];
            }
        }
    }
}

```

```

        c[k + j] = u + t;
        c[k + j + (m >> 1)] = u - t;
        w = w * wm;
    }
}
}
if (~t) return;
for (int i = 0 ; i < n ; i++)
    c[i].x /= n, c[i].y /= n;
}
vector<int> multi(vector<int> &a, vector<int> &b) {
    int maxlen = max(a.size(), b.size());
    int n = 1; while (n < 2 * maxlen) n <= 1;
    vector<C> A(n), B(n), R(n);
    for (int i = 0 ; i < a.size() ; i++) A[i].x = a[i];
    for (int i = 0 ; i < b.size() ; i++) B[i].x = b[i];
    FFT(A, 1); FFT(B, 1);
    for (int i = 0 ; i < n ; i++) R[i] = A[i] * B[i];
    FFT(R, -1);
    vector<int> ret(n);
    for (int i = 0 ; i < n ; i++) ret[i] = int(R[i].x +
        .5);
    return ret;
}

```

## Matrix

```

template <typename T>
struct Matrix {
    using vt = vector<T>;
    using mt = vector<vt>;
    using matrix = Matrix<T>;
    int r, c;
    mt m;
    Matrix(int r, int c) : r(r), c(c), m(r, vt(c)){}
    vt& operator [](int i) { return m[i]; }
    matrix operator + (const matrix &a) {
        matrix ret(r, c);
        for (int i = 0 ; i < r ; i++)
            for (int j = 0 ; j < c ; j++)
                ret[i][j] = m[i][j] + a.m[i][j];
        return ret;
    }
    matrix operator - (const matrix &a) {
        matrix ret(r, c);
        for (int i = 0 ; i < r ; i++)
            for (int j = 0 ; j < c ; j++)
                ret[i][j] = m[i][j] - a.m[i][j];
        return ret;
    }
    matrix operator * (const matrix &a) {
        matrix ret(r, a.c);
        for (int i = 0 ; i < r ; i++)
            for (int j = 0 ; j < a.c ; j++)
                for (int k = 0 ; k < c ; k++)
                    ret.m[i][j] += m[i][k] * a.m[k][j];
        return ret;
    }
    T gas() {
        T det = 1;
        for (int i = 0 ; i < r ; i++) {
            for (int j = i + 1 ; j < r ; j++) {
                int a = i, b = j;
                while (m[b][i]) {
                    T q = m[a][i] / m[b][i];
                    for (int k = 0 ; k < c ; k++)
                        m[a][k] -= m[b][k] * q;
                    swap(a, b);
                }
                if (a != i) {
                    swap(m[i], m[j]);
                    det *= -1;
                }
            }
        }
        for (int i = 0 ; i < r ; i++)

```

```

        det *= m[i][i];
        return det;
    }
};

```

## Mobius

```

const int MAXN = 1e5 + 5;
vector<bool> isPrime(MAXN, true);
vector<int> mu(MAXN), prime;
void mobius() {
    mu[1] = 1;
    for (int i = 2 ; i < MAXN ; i++) {
        if (isPrime[i]) prime.push_back(i), mu[i] = -1;
        for (auto p : prime) {
            if (i * p >= MAXN) break;
            isPrime[i * p] = mu[i * p] = false;
            if (i % p == 0) break;
            mu[i * p] = -mu[i];
        }
    }
}

```

## PHITable

```

const int MAXN = 1000;
long long int PHI[MAXN + 1];
void PHITable(){
    for (int i = 1 ; i <= MAXN ; i++) PHI[i] = i;
    for (int i = 1 ; i <= MAXN ; i++)
        for (int j = i * 2 ; j <= MAXN ; j += i)
            PHI[j] -= PHI[i];
}

```

## MillerRabin

```

LL modMul(LL a, LL b, LL m){
    a %= m, b %= m;
    LL y = (LL)((double)a * b / m + .5);
    LL r = (a * b - y * m) % m;
    return r < 0 ? r + m : r;
}
template <typename T>
inline T pow(T a, T b, T mod){
    T ans = 1;
    for (; b; a = modMul(a, a, mod), b >>= 1)
        if (b%2) ans = modMul(ans, a, mod);
    return ans;
}
int sprp[3] = {2, 7, 61};
int llsprp[7] = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};
template <typename T>
inline bool isPrime(T n, int *sprp, int num){
    if (n == 2) return true;
    if (n < 2 || n % 2 == 0) return false;
    int t = 0;
    T u = n - 1;
    for (; u % 2 == 0; t++) u >>= 1;
    for (int i = 0 ; i < num ; i++){
        T a = sprp[i] % n;
        if (a == 0 || a == 1 || a == n-1) continue;
        T x = pow(a, u, n);
        if (x == 1 || x == n-1) continue;
        for (int j = 1 ; j < t ; j++){
            x = modMul(x, x, n);
            if (x == 1) return false;
            if (x == n - 1) break;
        }
        if (x == n - 1) continue;
        return false;
    }
}

```

```

    }
    return true;
}

```

## PrimitiveRoot

```

LL modPow(LL a, LL x, LL m){
    if (x == 0) return 1;
    LL k = modPow(a, x / 2, m);
    if (x & 1) return k * k % m * a % m;
    else return k * k % m;
}

const int MAXN = 1e9 + 5;
const int sqrtN = sqrt(MAXN) + 5;
vector<bool> isPrime(sqrtN, true);
vector<int> Prime;
void linearPrime(){
    isPrime[0] = isPrime[1] = false;
    for (int i = 2; i < sqrtN; i++){
        if (isPrime[i]){
            Prime.push_back(i);
            for (int j = 2 * i; j < sqrtN; j += i)
                isPrime[j] = false;
        }
    }
}

bool isPrimitiveRoot(int a, int x){
    vector<int> primeFactor;
    int target = x - 1;
    for (auto p : Prime){
        if (target < p) break;
        bool _find = false;
        while (target % p == 0) target /= p, _find = true;
        if (_find) primeFactor.push_back(p);
    }
    for (auto p : primeFactor)
        if (modPow(a, (x - 1) / p, x) == 1) return false;
    return true;
}

int main(){ ios_base::sync_with_stdio(false); cin.tie(0);
    int n; cin >> n; linearPrime();
    int ans = 0; while (1){
        ans++;
        if (!isPrimitiveRoot(ans, n)) continue;
        cout << ans << '\n'; break;
    }
}

```

## Simplex

```

const int maxn = 222;
const int maxm = 222;
const double eps = 1E-10;
double a[maxn][maxm], b[maxn], c[maxm], d[maxn][maxm];
double x[maxm];
int ix[maxn + maxm]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
//
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[maxn][maxm], double b[maxn],
    double c[maxm], int n, int m) {
    ++m;
    int r = n, s = m - 1;
    memset(d, 0, sizeof(d));
    for (int i = 0; i < n + m; ++i) ix[i] = i;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j)

```

```

        d[i][j] = -a[i][j];
        d[i][m - 1] = 1;
        d[i][m] = b[i];
        if (d[r][m] > d[i][m]) r = i;
    }
    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
    d[n + 1][m - 1] = -1;
    for (double dd;; ) {
        if (r < n) {
            int t = ix[s];
            ix[s] = ix[r + m]; ix[r + m] = t;
            d[r][s] = 1.0 / d[r][s];
            for (int j = 0; j <= m; ++j)
                if (j != s) d[r][j] *= -d[r][s];
            for (int i = 0; i <= n + 1; ++i)
                if (i != r) {
                    for (int j = 0; j <= m; ++j)
                        if (j != s)
                            d[i][j] += d[r][j] * d[i][s];
                    d[i][s] *= d[r][s];
                }
        }
        r = -1; s = -1;
        for (int j = 0; j < m; ++j)
            if (s < 0 || ix[s] > ix[j]) {
                if (d[n + 1][j] > eps || (d[n + 1][j] >
                    -eps && d[n][j] > eps)) s = j;
            }
        if (s < 0) break;
        for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
            if (r < 0 || (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s]) < -eps || (dd < eps &&
                ix[r + m] > ix[i + m])) r = i;
        }
        if (r < 0) return -1; // not bounded
    }
    if (d[n + 1][m] < -eps) return -1; // not executable
    double ans = 0;
    for (int i = 0; i < m; i++) x[i] = 0;
    for (int i = m; i < n + m; ++i) { // the missing enumerated x[i] = 0
        if (ix[i] < m - 1)
        {
            ans += d[i - m][m] * c[ix[i]];
            x[ix[i]] = d[i - m][m];
        }
    }
    return ans;
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(0);
    int n, m; while (cin >> n >> m) {
        for (int i = 0; i < n; i++) cin >> c[i];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++)
                cin >> a[i][j];
            cin >> b[i];
        }
        cout << "Nasa can spend " << ceil(simplex(a, b,
            c, m, n) * m) << " taka.\n";
    }
}

```

## PollardRho

```

// does not work when n is prime
long long modit(long long x, long long mod) {
    if (x >= mod) x -= mod;
    //if (x < 0) x += mod;
    return x;
}

long long mult(long long x, long long y, long long mod) {
    long long s = 0, m = x % mod;
    while (y) {
        if (y & 1) s = modit(s + m, mod);

```

```

    y>>=1;
    m=modit(m+m,mod);
}
return s;
}
long long f(long long x,long long mod) {
    return modit(mult(x,x,mod)+1,mod);
}
long long pollard_rho(long long n) {
    if(!(n&1)) return 2;
    while (true) {
        long long y=2, x=rand()%(n-1)+1, res=1;
        for (int sz=2; res==1; sz*=2) {
            for (int i=0; i<sz && res<=1; i++) {
                x = f(x, n);
                res = __gcd(abs(x-y), n);
            }
            y = x;
        }
        if (res!=0 && res!=n) return res;
    }
}
}

```

## String

### ACAAutomaton

```

const int SIGMA = 26;
const int MAXLEN = 1e5;
struct ACAutomaton{
    struct Node{
        Node *n[SIGMA], *f;
        int dp;
        Node(){
            memset(n, 0, sizeof(n));
            dp = 0; f = NULL;
        }
    }*r, *o;
    ACAutomaton(int n){
        o = new Node();
        r = new Node();
        for (int i = 0 ; i < n ; i++){
            char input[MAXLEN]; cin >> input;
            buildTrie(input);
        }
        buildAC();
    }
    ~ACAutomaton(){
        remove(r);
        delete o;
    }
    void remove(Node *u){
        if (!u) return ;
        for (int i = 0 ; i < SIGMA ; i++)
            remove(u->n[i]);
        delete u;
    }
    inline int idx(char c){
        // mapping function;
        return c - 'a';
    }
    void buildTrie(char *s){
        Node *u = r;
        for (int i = 0 ; s[i] ; i++){
            int c = idx(s[i]);
            if (!u->n[c])
                u->n[c] = new Node();
            u = u->n[c];
        }
        u->dp++;
    }
    void buildAC(){
        static queue<Node*> q;
        for (int i = 0 ; i < SIGMA ; i++)

```

```

        o->n[i] = r;
        r->f = o; q.push(r);
        while (q.size()){
            Node *u = q.front(); q.pop();
            for (int i = 0 ; i < SIGMA ; i++){
                if (!u->n[i]) continue;
                u->n[i]->f = trans(u->f, i);
                q.push(u->n[i]);
            }
            // u->dp += u->f->dp;
        }
    }
    Node* trans(Node *u, int c){
        while (!u->n[c]) u = u->f;
        return u->n[c];
    }
    int search(char *s){
        int ans = 0;
        Node *u = r;
        for (int i = 0 ; i < s[i] ; i++){
            u = trans(u, idx(s[i]));
            ans += u->dp;
        }
        return ans;
    }
}

```

```
};
```

### Eertree

```

const int SIGMA = 26;
inline int idx(char c){ return c - 'a'; }
struct Eertree{
    struct Node{
        Node *n[SIGMA], *f;
        int len;
        Node (int _len = 0){
            len = _len, f = NULL;
            memset(n, 0, sizeof(n));
        }
    }*last, *rt;
    vector<char> s;
    int n, maxlen, sz;
    Eertree(char *input){
        s.clear(), s.PB(-1); n = 0;
        rt = new Node(0); maxlen = -1;
        last = new Node(-1); sz = 0;
        rt->f = last; last->f = last;
        for (int i = 0 ; input[i] ; i++) add(input[i]);
    }
    ~Eertree(){
        clear(rt->f); clear(rt);
    }
    void clear(Node *u){
        if (!u) return ;
        for (int i = 0 ; i < SIGMA ; i++)
            clear(u->n[i]);
        delete u;
    }
    inline Node* getFail(Node *u){
        while (s[n - u->len - 1] != s[n]) u = u->f;
        return u;
    }
    inline void add(char c){
        s.PB(c); n++;
        Node *u = getFail(last);
        if (!u->n[idx(c)]){
            Node *v = new Node(u->len + 2);
            maxlen = max(maxlen, v->len);
            sz++;
            v->f = getFail(u->f->n[idx(c)]);
            if (!v->f) v->f = rt;
            u->n[idx(c)] = v;
        }
        last = u->n[idx(c)];
    }
}

```

};

## KMP

```

int F[MAXLEN];
void build(char *s){
    F[0] = -1;
    for (int i = 1, pos = -1; s[i] ; i++){
        while (~pos && s[i] != s[pos + 1]) pos = F[pos];
        if (s[i] == s[pos + 1]) pos++;
        F[i] = pos;
    }
}
bool match(char *_find, char *content){
    int findlen = strlen(_find);
    for (int i = 0, pos = -1; content[i] ; i++){
        while (~pos && content[i] != _find[pos + 1])
            pos = F[pos];
        if (content[i] == _find[pos + 1]) pos++;
        if (pos + 1 == findlen) return true;
    }
    return false;
}

```

## minRotation

```

string minStringRotate(string s){
    int n = s.length();
    s += s;
    int i=0, j=1;
    while (i<n && j<n){
        int k = 0;
        while (k < n && s[i+k] == s[j+k]) k++;
        if (s[i+k] <= s[j+k]) j += k+1;
        else i += k+1;
        if (i == j) j++;
    }
    int ans = i < n ? i : j;
    return s.substr(ans, n);
}

```

## SA

```

const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX];
int sa[MAX], tsa[MAX], tp[MAX][2];
void suffix_array(char *ip){
    int len = strlen(ip);
    int alp = 256;
    memset(ct, 0, sizeof(ct));
    for(int i=0;i<len;i++) ct[ip[i]+1]++;
    for(int i=1;i<alp;i++) ct[i]+=ct[i-1];
    for(int i=0;i<len;i++) rk[i]=ct[ip[i]];
    for(int i=1;i<len;i*=2){
        for(int j=0;j<len;j++){
            if(j+i>len) tp[j][1]=0;
            else tp[j][1]=rk[j+i]+1;
            tp[j][0]=rk[j];
        }
        memset(ct, 0, sizeof(ct));
        for(int j=0;j<len;j++) ct[tp[j][1]+1]++;
        for(int j=1;j<len+1;j++) ct[j]+=ct[j-1];
        for(int j=0;j<len;j++) tsa[ct[tp[j][1]]+1]=j;
        memset(ct, 0, sizeof(ct));
        for(int j=0;j<len;j++) ct[tp[j][0]+1]++;
        for(int j=1;j<len+1;j++) ct[j]+=ct[j-1];
        for(int j=0;j<len;j++)
            sa[ct[tp[tsa[j]][0]]+1]=tsa[j];
        rk[sa[0]]=0;
        for(int j=1;j<len;j++){

```

```

            if( tp[sa[j]][0] == tp[sa[j-1]][0] &&
                tp[sa[j]][1] == tp[sa[j-1]][1] )
                rk[sa[j]] = rk[sa[j-1]];
            else
                rk[sa[j]] = j;
        }
    }
    for(int i=0,h=0;i<len;i++){
        if(rk[i]==0) h=0;
        else{
            int j=sa[rk[i]-1];
            h=max(0,h-1);
            for(;ip[i+h]==ip[j+h];h++);
        }
        he[rk[i]]=h;
    }
}

```

## SAM

```

const int SIGMA = 26;
struct SAM {
    struct Node {
        Node *f, *ch[SIGMA];
        int len;
        Node(int _len) {
            len = _len; f = 0;
            memset(ch, 0, sizeof(ch));
        }
    } *rt, *la;
    inline int idx(char c) { return c - 'a'; }
    SAM(char *s) {
        rt = la = new Node(0);
        for (int i = 0 ; s[i] ; i++) extend(idx(s[i]));
    }
    void extend(int c) {
        Node *u = la; la = new Node(u->len + 1);
        for (; u && !u->ch[c] ; u = u->f) u->ch[c] = la;
        if (!u) la->f = rt;
        else {
            Node *pf = u->ch[c];
            if (pf->len == u->len + 1) la->f = pf;
            else {
                Node *cn = new Node(u->len + 1);
                for (; u && u->ch[c] == pf; u = u->f) u->ch[c] = cn;
                for (int i = 0 ; i < SIGMA ; i++) cn->ch[i] = pf->ch[i];
                cn->f = pf->f;
                pf->f = la->f = cn;
            }
        }
    }
    bool search(char *s) {
        Node *u = rt;
        for (int i = 0 ; s[i] ; i++) {
            u = u->ch[idx(s[i])];
            if (!u) return false;
        }
        return true;
    }
};

```

## Z

```

void ZAlg(char *s, int *Z){
    Z[0] = strlen(s);
    for (int L = 0, R = 0, i = 1; s[i] ; i++){
        if (i <= R && Z[i - L] <= R - i) Z[i] = Z[i - L];
        else{
            L = i; if (i > R) R = i;

```

```

        while (R < Z[0] && s[R - L] == s[R]) R++;
        Z[i] = (R--) - L;
    }
}

```

## BWT

```

const int N = 8;
int s[N+N+1] = "suffixes";
int sa[N];
int pivot;

int cmp(const void* i, const void* j)
{
    return strcmp(s+(int*)i, s+(int*)j, N);
}

void BWT()
{
    strcpy(s + N, s, N);
    for (int i=0; i<N; ++i) sa[i] = i;
    qsort(sa, N, sizeof(int), cmp);

    for (int i=0; i<N; ++i)
        cout << s[(sa[i] + N-1) % N];

    for (int i=0; i<N; ++i)
        if (sa[i] == 0)
        {
            pivot = i;
            break;
        }
}

```

```

# output
print( x, sep = ' ')
print( ''.join( str(x) + ' ' for x in a ) )
print( '{:5d}'.format(x) )

# sort
a.sort()

# list
a = [ x for x in range(n) ]
a.append(x)

# stack
stack = [3, 4, 5] # C++
stack.append(6)   # push(6)
stack.pop()       # pop()
stack[-1]         # top()
len(stack)        # size()

# queue
for collections import deque
queue = deque([3, 4, 5])
queue.append(6)   # push(6)
queue.popleft()   # pop()
queue[0]          # front()
len(queue)       # size()

```

## IBWT

```

const int N = 8; // 字串長度
char t[N+1] = "xuffessi"; // 字串
int pivot;
int next[N];

void IBWT()
{
    vector<int> index[256];
    for (int i=0; i<N; ++i)
        index[t[i]].push_back(i);

    for (int i=0, n=0; i<256; ++i)
        for (int j=0; j<index[i].size(); ++j)
            next[n++] = index[i][j];

    int p = pivot;
    for (int i=0; i<N; ++i)
        cout << t[p = next[p]];
}

```

## Other

### Python

```

# input
n = int( input() )

# EOF
while True:
    try:
        solve()
    except:
        break

```