

# Contents

<b>1 Basic</b>	<b>1</b>
1.1 vimrc	1
1.2 IncreaseStackSize	1
1.3 Default Code	2
<b>2 Data Structure</b>	<b>2</b>
2.1 Bigint	2
2.2 unordered_map	3
2.3 extc_balance_tree	3
2.4 Disjoint Set	4
2.5 Treap	4
2.6 Link-Cut Tree	4
<b>3 Graph</b>	<b>5</b>
3.1 BCC Edge	5
3.2 BCC Vertex	6
3.3 Strongly Connected Components	6
3.4 DMST_with_sol	7
3.5 Maximum Clique	7
3.6 MinimumMeanCycle	7
<b>4 Flow</b>	<b>8</b>
4.1 ISAP	8
4.2 Dinic	8
4.3 Cost Flow	8
4.4 Bipartite Matching (Augmenting Path)	9
4.5 Kuhn Munkres	9
4.6 SW-Mincut	9
4.7 Maximum Simple Graph Matching	9
4.8 Minimum Weight Matching (Clique version)	10
4.9 2-Commodity Flow	11
4.10 (+1) SW-mincut $O(NM)$	12
<b>5 Math</b>	<b>13</b>
5.1 ax+by=gcd	13
5.2 Chinese Remainder	13
5.3 Fast Fourier Transform	13
5.4 Fast Linear Recurrence	13
5.5 (+1) ntt	14
5.6 Mod	14
5.7 (+1) Miller Rabin	15
5.8 Pollard Rho	15
5.9 Algorithms about Primes	15
5.10 (+1) PolynomialGenerator	15
5.11 Pseudoinverse of Square matrix	16
5.12 Simplex	16
5.13 Theorem	17
5.13.1 Lucas' Theorem	17
5.13.2 Sum of Two Squares Thm (Legendre)	17
5.13.3 Difference of D1-D3 Thm	17
5.13.4 Krush-Kuhn-Tucker Conditions	17
<b>6 Geometry</b>	<b>17</b>
6.1 Point operators	17
6.2 Intersection of two circles	17
6.3 Intersection of two lines	17
6.4 Half Plane Intersection	17
6.5 Convex Hull	18
6.6 Minimum Covering Circle	18
6.7 KDTree (Nearest Point)	19
6.8 (+1) MinkowskiSum	19
<b>7 Stringology</b>	<b>20</b>
7.1 Suffix Array	20
7.2 Suffix Array (SAIS TWT514)	21
7.3 Aho-Corasick Algorithm	21
7.4 KMP	21
7.5 Z value	22
7.6 Z value (palindrome ver.)	22
7.7 palindromic tree	22
7.8 Lexicographically Smallest Rotation	22
7.9 Suffix Automaton	22
<b>8 Problems</b>	<b>22</b>
8.1 Find the maximun tangent (x,y is increasing)	22
8.2 Painter	23

## 1 Basic

### 1.1 vimrc

```

1 colo torte
2 syn on
3 se ai ar sm nu rnu is
4 se mouse=a bs=2 ww+=<, >, [, ] so=6 ts=4 sw=4 tt=100
5 se makeprg=g++\ -Wall\ -Wshadow\ -O2\ -std=c++0x\ -o\
6     %<\ %
7 au BufNewFile *.cpp 0r ~/default.cpp | :1,$-6 fo
8 filetype indent on
9
10 map <F7> <ESC>:wa<CR>:make!<CR>
11 imap <F7> <ESC>:wa<CR>:make!<CR>
12 map <C-F7> <ESC>:tabe %<.in<CR>
13 map <F8> :cope <CR>
14 map <S-F8> :ccl <CR>
15 map <F9> :!./%< <CR>
16 map <C-F9> :!./%< < %<.in <CR>

```

### 1.2 IncreaseStackSize

```

1 //stack resize
2 asm( "mov %0,%esp\n" ::"g"(mem+100000000) );
3 //change esp to rsp if 64-bit system
4
5 //stack resize (linux)
6 #include <sys/resource.h>
7 void increase_stack_size() {
8     const rlim_t ks = 64*1024*1024;
9     struct rlimit rl;
10    int res=getrlimit(RLIMIT_STACK, &rl);
11    if(res==0){
12        if(rl.rlim_cur<ks){
13            rl.rlim_cur=ks;
14            res=setrlimit(RLIMIT_STACK, &rl);
15        }
16    }
17 }

```

### 1.3 Default Code

```
#include<bits/stdc++.h>
#include<unistd.h>
using namespace std;
#define FZ(n) memset((n),0,sizeof(n))
#define FMO(n) memset((n),-1,sizeof(n))
#define F first
#define S second
#define PB push_back
#define ALL(x) begin(x),end(x)
#define SZ(x) ((int)(x).size())
#define IOS ios_base::sync_with_stdio(0); cin.tie(0)
#define REP(i,x) for (int i=0; i<(x); i++)
#define REPI(i,a,b) for (int i=(a); i<=(b); i++)
#ifdef ONLINE_JUDGE
#define FILEIO(name) \
    freopen(name".in", "r", stdin); \
    freopen(name".out", "w", stdout);
#else
#define FILEIO(name)
#endif
template<typename A, typename B>
ostream& operator <<(ostream &s, const pair<A,B> &p) {
    return s<<" "<<p.first<<" "<<p.second<<" ";
}
template<typename T>
ostream& operator <<(ostream &s, const vector<T> &c) {
    s<<"[";
    for (auto it : c) s << it << " ";
    s<<"]";
    return s;
}
// Let's Fight!

int main() {
    return 0;
}
```

## 2 Data Structure

### 2.1 Bigint

```
struct Bigint{
    static const int LEN = 60;
    static const int BIGMOD = 10000;

    int s;
    int vl, v[LEN];
    // vector<int> v;
    Bigint() : s(1) { vl = 0; }
    Bigint(long long a) {
        s = 1; vl = 0;
        if (a < 0) { s = -1; a = -a; }
        while (a) {
            push_back(a % BIGMOD);
            a /= BIGMOD;
        }
    }
    Bigint(string str) {
        s = 1; vl = 0;
        int stPos = 0, num = 0;
        if (!str.empty() && str[0] == '-') {
            stPos = 1;
            s = -1;
        }
        for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
            num += (str[i] - '0') * q;
            if ((q *= 10) >= BIGMOD) {
                push_back(num);
                num = 0; q = 1;
            }
        }
        if (num) push_back(num);
    }

    int len() const {
        return vl;
        // return SZ(v);
    }
    bool empty() const { return len() == 0; }
    void push_back(int x) {
        v[vl++] = x;
        // v.PB(x);
    }
    void pop_back() {
        vl--;
        // v.pop_back();
    }
    int back() const {
        return v[vl-1];
        // return v.back();
    }
    void n() {
        while (!empty() && !back()) pop_back();
    }
    void resize(int nl) {
        vl = nl;
        fill(v, v+vl, 0);
        // v.resize(nl);
        // fill(ALL(v), 0);
    }

    void print() const {
        if (empty()) { putchar('0'); return; }
        if (s == -1) putchar('-');
        printf("%d", back());
        for (int i=len()-2; i>=0; i--) printf("%.4d",v[i]);
    }
    friend ostream& operator << (std::ostream& out,
        const Bigint &a) {
        if (a.empty()) { out << "0"; return out; }
        if (a.s == -1) out << "-";
        out << a.back();
        for (int i=a.len()-2; i>=0; i--) {
            char str[10];
            snprintf(str, 5, "%.4d", a.v[i]);
            out << str;
        }
    }
}
```

```

    return out;
}

int cp3(const Bigint &b) const {
    if (s != b.s) return s - b.s;
    if (s == -1) return -(*this).cp3(-b);
    if (len() != b.len()) return len() - b.len(); //int
    for (int i=len()-1; i>=0; i--)
        if (v[i] != b.v[i]) return v[i] - b.v[i];
    return 0;
}

bool operator < (const Bigint &b) const { return cp3(b) < 0; }
bool operator <= (const Bigint &b) const { return cp3(b) <= 0; }
bool operator == (const Bigint &b) const { return cp3(b) == 0; }
bool operator != (const Bigint &b) const { return cp3(b) != 0; }
bool operator > (const Bigint &b) const { return cp3(b) > 0; }
bool operator >= (const Bigint &b) const { return cp3(b) >= 0; }

Bigint operator - () const {
    Bigint r = (*this);
    r.s = -r.s;
    return r;
}

Bigint operator + (const Bigint &b) const {
    if (s == -1) return -(*this) + (-b);
    if (b.s == -1) return (*this) - (-b);
    Bigint r;
    int nl = max(len(), b.len());
    r.resize(nl + 1);
    for (int i=0; i<nl; i++) {
        if (i < len()) r.v[i] += v[i];
        if (i < b.len()) r.v[i] += b.v[i];
        if (r.v[i] >= BIGMOD) {
            r.v[i+1] += r.v[i] / BIGMOD;
            r.v[i] %= BIGMOD;
        }
    }
    r.n();
    return r;
}

Bigint operator - (const Bigint &b) const {
    if (s == -1) return -(*this) - (-b);
    if (b.s == -1) return (*this) + (-b);
    if ((*this) < b) return -(b - (*this));
    Bigint r;
    r.resize(len());
    for (int i=0; i<len(); i++) {
        r.v[i] += v[i];
        if (i < b.len()) r.v[i] -= b.v[i];
        if (r.v[i] < 0) {
            r.v[i] += BIGMOD;
            r.v[i+1]--;
        }
    }
    r.n();
    return r;
}

Bigint operator * (const Bigint &b) {
    Bigint r;
    r.resize(len() + b.len() + 1);
    r.s = s * b.s;
    for (int i=0; i<len(); i++) {
        for (int j=0; j<b.len(); j++) {
            r.v[i+j] += v[i] * b.v[j];
            if (r.v[i+j] >= BIGMOD) {
                r.v[i+j+1] += r.v[i+j] / BIGMOD;
                r.v[i+j] %= BIGMOD;
            }
        }
    }
    r.n();
    return r;
}

Bigint operator / (const Bigint &b) {
    Bigint r;

```

```

    r.resize(max(1, len()-b.len()+1));
    int oriS = s;
    Bigint b2 = b; // b2 = abs(b)
    s = b2.s = r.s = 1;
    for (int i=r.len()-1; i>=0; i--) {
        int d=0, u=BIGMOD-1;
        while(d<u) {
            int m = (d+u+1)>>1;
            r.v[i] = m;
            if((r*b2) > (*this)) u = m-1;
            else d = m;
        }
        r.v[i] = d;
    }
    s = oriS;
    r.s = s * b.s;
    r.n();
    return r;
}

Bigint operator % (const Bigint &b) {
    return (*this) - (*this) / b * b;
}
};

```

## 2.2 unordered\_map

```

struct Key {
    int first, second;
    Key () {}
    Key (int _x, int _y) : first(_x), second(_y) {}
    bool operator == (const Key &b) const {
        return tie(F,S) == tie(b.F,b.S);
    }
};

struct KeyHasher {
    size_t operator()(const Key& k) const {
        return k.first + k.second*1000000;
    }
};

typedef unordered_map<Key,int,KeyHasher> map_t;

int main(int argc, char** argv){
    map_t mp;
    for (int i=0; i<10; i++)
        mp[Key(i,0)] = i+1;
    for (int i=0; i<10; i++)
        printf("%d\n", mp[Key(i,0)]);

    return 0;
}

```

## 2.3 extc\_balance\_tree

```

#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
            tree_order_statistics_node_update> set_t;

int main()
{
    // Insert some entries into s.
    set_t s;
    s.insert(12);
    s.insert(505);

    // The order of the keys should be: 12, 505.
    assert(*s.find_by_order(0) == 12);
    assert(*s.find_by_order(3) == 505);

    // The order of the keys should be: 12, 505.
    assert(s.order_of_key(12) == 0);
    assert(s.order_of_key(505) == 1);

    // Erase an entry.
    s.erase(12);

    // The order of the keys should be: 505.

```

```

assert(*s.find_by_order(0) == 505);

// The order of the keys should be: 505.
assert(s.order_of_key(505) == 0);
}

```

## 2.4 Disjoint Set

```

struct DisjointSet {
    // save() is like recursive
    // undo() is like return
    int n, fa[MXN], sz[MXN];
    vector<pair<int*,int>> h;
    vector<int> sp;
    void init(int tn) {
        n=tn;
        for (int i=0; i<n; i++) {
            fa[i]=i;
            sz[i]=1;
        }
        sp.clear(); h.clear();
    }
    void assign(int *k, int v) {
        h.PB({k, *k});
        *k=v;
    }
    void save() { sp.PB(SZ(h)); }
    void undo() {
        assert(!sp.empty());
        int last=sp.back(); sp.pop_back();
        while (SZ(h)!=last) {
            auto x=h.back(); h.pop_back();
            *x.F=x.S;
        }
    }
    int f(int x) {
        while (fa[x]!=x) x=fa[x];
        return x;
    }
    void uni(int x, int y) {
        x=f(x); y=f(y);
        if (x==y) return;
        if (sz[x]<sz[y]) swap(x, y);
        assign(&sz[x], sz[x]+sz[y]);
        assign(&fa[y], x);
    }
}djs;

```

## 2.5 Treap

```

const int MEM = 16000004;
struct Treap {
    static Treap nil, mem[MEM], *pmem;
    Treap *l, *r;
    char val;
    int size;
    Treap () : l(&nil), r(&nil), size(0) {}
    Treap (char _val) :
        l(&nil), r(&nil), val(_val), size(1) {}
} Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap::
    mem;

int size(const Treap *t) { return t->size; }
void pull(Treap *t) {
    if (!size(t)) return;
    t->size = size(t->l) + size(t->r) + 1;
}
Treap* merge(Treap *a, Treap *b) {
    if (!size(a)) return b;
    if (!size(b)) return a;
    Treap *t;
    if (rand() % (size(a) + size(b)) < size(a)) {
        t = new (Treap::pmem++) Treap(*a);
        t->r = merge(a->r, b);
    } else {
        t = new (Treap::pmem++) Treap(*b);
        t->l = merge(a, b->l);
    }
}

```

```

}
pull(t);
return t;
}
void split(Treap *t, int k, Treap *&a, Treap *&b) {
    if (!size(t)) a = b = &Treap::nil;
    else if (size(t->l) + 1 <= k) {
        a = new (Treap::pmem++) Treap(*t);
        split(t->r, k - size(t->l) - 1, a->r, b);
        pull(a);
    } else {
        b = new (Treap::pmem++) Treap(*t);
        split(t->l, k, a, b->l);
        pull(b);
    }
}

int nv;
Treap *rt[50005];

void print(const Treap *t) {
    if (!size(t)) return;
    print(t->l);
    cout << t->val;
    print(t->r);
}

int main(int argc, char** argv) {
    IOS;
    rt[nv=0] = &Treap::nil;
    Treap::pmem = Treap::mem;
    int Q, cmd, p, c, v;
    string s;
    cin >> Q;
    while (Q--) {
        cin >> cmd;
        if (cmd == 1) {
            // insert string s after position p
            cin >> p >> s;
            Treap *tl, *tr;
            split(rt[nv], p, tl, tr);
            for (int i=0; i<SZ(s); i++)
                tl = merge(tl, new (Treap::pmem++) Treap(s[i]));
            rt[++nv] = merge(tl, tr);
        } else if (cmd == 2) {
            // remove c characters starting at position
            Treap *tl, *tm, *tr;
            cin >> p >> c;
            split(rt[nv], p-1, tl, tm);
            split(tm, c, tm, tr);
            rt[++nv] = merge(tl, tr);
        } else if (cmd == 3) {
            // print c characters starting at position p, in
            // version v
            Treap *tl, *tm, *tr;
            cin >> v >> p >> c;
            split(rt[v], p-1, tl, tm);
            split(tm, c, tm, tr);
            print(tm);
            cout << "\n";
        }
    }
    return 0;
}

```

## 2.6 Link-Cut Tree

```

const int MXN = 100005;
const int MEM = 100005;

struct Splay {
    static Splay nil, mem[MEM], *pmem;
    Splay *ch[2], *f;
    int val, rev, size;
    Splay () : val(-1), rev(0), size(0) {
        f = ch[0] = ch[1] = &nil;
    }
    Splay (int _val) : val(_val), rev(0), size(1) {
        f = ch[0] = ch[1] = &nil;
    }
}

```

```

}
bool isr() {
    return f->ch[0] != this && f->ch[1] != this;
}
int dir() {
    return f->ch[0] == this ? 0 : 1;
}
void setCh(Splay *c, int d) {
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
}
void push() {
    if (rev) {
        swap(ch[0], ch[1]);
        if (ch[0] != &nil) ch[0]->rev ^= 1;
        if (ch[1] != &nil) ch[1]->rev ^= 1;
        rev=0;
    }
}
void pull() {
    size = ch[0]->size + ch[1]->size + 1;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
}
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::
    mem;
Splay *nil = &Splay::nil;

void rotate(Splay *x) {
    Splay *p = x->f;
    int d = x->dir();
    if (!p->isr()) p->f->setCh(x, p->dir());
    else x->f = p->f;
    p->setCh(x->ch[!d], d);
    x->setCh(p, !d);
    p->pull(); x->pull();
}

vector<Splay*> splayVec;
void splay(Splay *x) {
    splayVec.clear();
    for (Splay *q=x;; q=q->f) {
        splayVec.push_back(q);
        if (q->isr()) break;
    }
    reverse(begin(splayVec), end(splayVec));
    for (auto it : splayVec) it->push();
    while (!x->isr()) {
        if (x->f->isr()) rotate(x);
        else if (x->dir()==x->f->dir()) rotate(x->f), rotate
            (x);
        else rotate(x), rotate(x);
    }
}

Splay* access(Splay *x) {
    Splay *q = nil;
    for (;x!=nil;x=x->f) {
        splay(x);
        x->setCh(q, 1);
        q = x;
    }
    return q;
}

void evert(Splay *x) {
    access(x);
    splay(x);
    x->rev ^= 1;
    x->push(); x->pull();
}

void link(Splay *x, Splay *y) {
    // evert(x);
    access(x);
    splay(x);
    evert(y);
    x->setCh(y, 1);
}

void cut(Splay *x, Splay *y) {
    // evert(x);
    access(y);
    splay(y);

```

```

y->push();
y->ch[0] = y->ch[0]->f = nil;
}

int N, Q;
Splay *vt[MXN];

int ask(Splay *x, Splay *y) {
    access(x);
    access(y);
    splay(x);
    int res = x->f->val;
    if (res == -1) res=x->val;
    return res;
}

int main(int argc, char** argv) {
    scanf("%d%d", &N, &Q);
    for (int i=1; i<=N; i++)
        vt[i] = new (Splay::pmem++) Splay(i);
    while (Q--) {
        char cmd[105];
        int u, v;
        scanf("%s", cmd);
        if (cmd[1] == 'i') {
            scanf("%d%d", &u, &v);
            link(vt[u], vt[v]);
        } else if (cmd[0] == 'c') {
            scanf("%d", &v);
            cut(vt[1], vt[v]);
        } else {
            scanf("%d%d", &u, &v);
            int res=ask(vt[u], vt[v]);
            printf("%d\n", res);
        }
    }

    return 0;
}

```

## 3 Graph

### 3.1 BCC Edge

```

struct BccEdge {
    static const int MXN = 100005;
    struct Edge { int v, eid; };
    int n, m, step, par[MXN], dfn[MXN], low[MXN];
    vector<Edge> E[MXN];
    DisjointSet djs;
    void init(int _n) {
        n = _n; m = 0;
        for (int i=0; i<n; i++) E[i].clear();
        djs.init(n);
    }
    void add_edge(int u, int v) {
        E[u].PB({v, m});
        E[v].PB({u, m});
        m++;
    }
    void DFS(int u, int f, int f_eid) {
        par[u] = f;
        dfn[u] = low[u] = step++;
        for (auto it:E[u]) {
            if (it.eid == f_eid) continue;
            int v = it.v;
            if (dfn[v] == -1) {
                DFS(v, u, it.eid);
                low[u] = min(low[u], low[v]);
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }
    void solve() {
        step = 0;
        memset(dfn, -1, sizeof(int)*n);
        for (int i=0; i<n; i++) {
            if (dfn[i] == -1) DFS(i, i, -1);

```

```

    }
    djs.init(n);
    for (int i=0; i<n; i++) {
        if (low[i] < dfn[i]) djs.uni(i, par[i]);
    }
}
}graph;

```

## 3.2 BCC Vertex

```

struct BccVertex {
    int n,nScc,step,dfn[MXN],low[MXN];
    vector<int> E[MXN],sccv[MXN];
    int top,stk[MXN];
    void init(int _n) {
        n = _n;
        nScc = step = 0;
        for (int i=0; i<n; i++) E[i].clear();
    }
    void add_edge(int u, int v) {
        E[u].PB(v);
        E[v].PB(u);
    }
    void DFS(int u, int f) {
        dfn[u] = low[u] = step++;
        stk[top++] = u;
        for (auto v:E[u]) {
            if (v == f) continue;
            if (dfn[v] == -1) {
                DFS(v,u);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    int z;
                    sccv[nScc].clear();
                    do {
                        z = stk[--top];
                        sccv[nScc].PB(z);
                    } while (z != v);
                    sccv[nScc].PB(u);
                    nScc++;
                }
            } else {
                low[u] = min(low[u],dfn[v]);
            }
        }
    }
    vector<vector<int>> solve() {
        vector<vector<int>> res;
        for (int i=0; i<n; i++) {
            dfn[i] = low[i] = -1;
        }
        for (int i=0; i<n; i++) {
            if (dfn[i] == -1) {
                top = 0;
                DFS(i,i);
            }
        }
        REP(i,nScc) res.PB(sccv[i]);
        return res;
    }
}graph;

```

## 3.3 Strongly Connected Components

```

struct Scc{
    int n, nScc, vst[MXN], bln[MXN];
    vector<int> E[MXN], rE[MXN], vec;
    void init(int _n){
        n = _n;
        for (int i=0; i<MXN; i++){
            E[i].clear();
            rE[i].clear();
        }
    }
    void add_edge(int u, int v){
        E[u].PB(v);
        rE[v].PB(u);
    }
}

```

```

void DFS(int u){
    vst[u]=1;
    for (auto v : E[u])
        if (!vst[v]) DFS(v);
    vec.PB(u);
}
void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    for (auto v : rE[u])
        if (!vst[v]) rDFS(v);
}
void solve(){
    nScc = 0;
    vec.clear();
    FZ(vst);
    for (int i=0; i<n; i++)
        if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    FZ(vst);
    for (auto v : vec){
        if (!vst[v]){
            rDFS(v);
            nScc++;
        }
    }
}
};

```

## 3.4 DMST\_with\_sol

```

const int INF = 1029384756;

struct edge_t{
    int u,v,w;
    set< pair<int,int> > add, sub;
    edge_t() : u(-1), v(-1), w(0) {}
    edge_t(int _u, int _v, int _w) {
        u = _u; v = _v; w = _w;
        add.insert({u, v});
    }
    edge_t& operator += (const edge_t& obj) {
        w += obj.w;
        FOR (it, obj.add) {
            if (!sub.count(*it)) add.insert(*it);
            else sub.erase(*it);
        }
        FOR (it, obj.sub) {
            if (!add.count(*it)) sub.insert(*it);
            else add.erase(*it);
        }
        return *this;
    }
    edge_t& operator -= (const edge_t& obj) {
        w -= obj.w;
        FOR (it, obj.sub) {
            if (!sub.count(*it)) add.insert(*it);
            else sub.erase(*it);
        }
        for (auto it : obj.add) {
            if (!add.count(it)) sub.insert(it);
            else add.erase(it);
        }
        return *this;
    }
}eg[MXN*MXN],prv[MXN],EDGE_INF(-1,-1,INF);
int N,M;
int cid,incyc[MXN],contracted[MXN];
vector<int> E[MXN];

edge_t dmst(int rt){
    edge_t cost;
    for (int i=0; i<N; i++){
        contracted[i] = incyc[i] = 0;
        prv[i] = EDGE_INF;
    }
    cid = 0;
    int u,v;
    while (true){
        for (v=0; v<N; v++){

```

```

    if (v != rt && !contracted[v] && prv[v].w == INF)
        break;
}
if (v >= N) break; // end
for (int i=0; i<M; i++){
    if (eg[i].v == v && eg[i].w < prv[v].w)
        prv[v] = eg[i];
}
if (prv[v].w == INF) // not connected
    return EDGE_INF;
cost += prv[v];
for (u=prv[v].u; u!=v && u!=-1; u=prv[u].u);
if (u == -1) continue;
incyc[v] = ++cid;
for (u=prv[v].u; u!=v; u=prv[u].u){
    contracted[u] = 1;
    incyc[u] = cid;
}
for (int i=0; i<M; i++){
    if (incyc[eg[i].u] != cid && incyc[eg[i].v] == cid){
        eg[i] -= prv[eg[i].v];
    }
}
for (int i=0; i<M; i++){
    if (incyc[eg[i].u] == cid) eg[i].u = v;
    if (incyc[eg[i].v] == cid) eg[i].v = v;
    if (eg[i].u == eg[i].v) eg[i--] = eg[--M];
}
for (int i=0; i<N; i++){
    if (contracted[i]) continue;
    if (prv[i].u>=0 && incyc[prv[i].u] == cid)
        prv[i].u = v;
}
prv[v] = EDGE_INF;
}
return cost;
}

void solve(){
    edge_t cost = dmst(0);
    for (auto it : cost.add){ // find a solution
        E[it.F].PB(it.S);
        prv[it.S] = edge_t(it.F, it.S, 0);
    }
}

```

### 3.5 Maximum Clique

```

class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;

    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }

    /* Zero Base */
    void addEdge(int u, int v) {
        if (u > v) swap(u, v);
        if (u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }

    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for (int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if (k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if (c == 0) {

```

```

            if (k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for (int i=0; i<(V+31)/32; i++) {
            for (int a = s[k][i]; a ; a >> 1) {
                if (k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while (lb!=1) {
                    lb = (unsigned int)(lb) >> 1;
                    lg++;
                }
                int u = i*32 + lg;
                if (k + dp[u] <= ans) return 0;
                if (dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }

    int solve() {
        for (int i=V-1; i>=0; i--) {
            dfs(i, 1);
            dp[i] = ans;
        }
        return ans;
    }
};

```

### 3.6 MinimumMeanCycle

```

/* minimum mean cycle */
const int MAXE = 1805;
const int MAXN = 35;
const double inf = 1029384756;
const double eps = 1e-6;
struct Edge {
    int v,u;
    double c;
};
int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];
Edge e[MAXE];
vector<int> edgeID, cycle, rho;
double d[MAXN][MAXN];
inline void bellman_ford() {
    for (int i=0; i<n; i++) d[0][i]=0;
    for (int i=0; i<n; i++) {
        fill(d[i+1], d[i+1]+n, inf);
        for (int j=0; j<m; j++) {
            int v = e[j].v, u = e[j].u;
            if (d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
                d[i+1][u] = d[i][v]+e[j].c;
                prv[i+1][u] = v;
                prve[i+1][u] = j;
            }
        }
    }
}

double karp_mmc() {
    // returns inf if no cycle, mmc otherwise
    double mmc=inf;
    int st = -1;
    bellman_ford();
    for (int i=0; i<n; i++) {
        double avg=-inf;
        for (int k=0; k<n; k++) {
            if (d[n][i]<inf-eps) avg=max(avg, (d[n][i]-d[k][i])/(n-k));
            else avg=max(avg, inf);
        }
        if (avg < mmc) tie(mmc, st) = tie(avg, i);
    }
}

```

```

FZ(vst); edgeID.clear(); cycle.clear(); rho.clear();
for (int i=n; !vst[st]; st=prv[i--][st]) {
    vst[st]++;
    edgeID.PB(prve[i][st]);
    rho.PB(st);
}
while (vst[st] != 2) {
    int v = rho.back(); rho.pop_back();
    cycle.PB(v);
    vst[v]++;
}
reverse(ALL(edgeID));
edgeID.resize(SZ(cycle));
return mmc;
}

```

## 4 Flow

### 4.1 ISAP

```

struct Isap{
    static const int MXN = 10000;
    struct Edge{ int v,f,re; };
    int n,s,t,h[MXN],gap[MXN];
    vector<Edge> E[MXN];
    void init(int _n, int _s, int _t){
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
    }
    void add_edge(int u, int v, int f){
        E[u].PB({v,f,SZ(E[v])});
        E[v].PB({u,0,SZ(E[u])-1});
    }
    int DFS(int u, int nf, int res=0){
        if (u == t) return nf;
        for (auto &it : E[u]){
            if (h[u]==h[it.v]+1 && it.f>0){
                int tf = DFS(it.v,min(nf,it.f));
                res += tf; nf -= tf; it.f -= tf;
                E[it.v][it.re].f += tf;
                if (nf == 0) return res;
            }
        }
        if (nf){
            if (--gap[h[u]] == 0) h[s]=n;
            gap[++h[u]]++;
        }
        return res;
    }
    int flow(int res=0){
        FZ(h); FZ(gap);
        gap[0] = n;
        while (h[s] < n) res += DFS(s,2147483647);
        return res;
    }
}flow;

```

### 4.2 Dinic

```

struct Dinic{
    static const int MXN = 10000;
    struct Edge{ int v,f,re; };
    int n,s,t,level[MXN];
    vector<Edge> E[MXN];
    void init(int _n, int _s, int _t){
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
    }
    void add_edge(int u, int v, int f){
        E[u].PB({v,f,SZ(E[v])});
        E[v].PB({u,0,SZ(E[u])-1});
    }
    bool BFS(){
        FMO(level);
        queue<int> que;
        que.push(s);
    }

```

```

    level[s] = 0;
    while (!que.empty()){
        int u = que.front(); que.pop();
        for (auto it : E[u]){
            if (it.f > 0 && level[it.v] == -1){
                level[it.v] = level[u]+1;
                que.push(it.v);
            }
        }
    }
    return level[t] != -1;
}
int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
        if (it.f > 0 && level[it.v] == level[u]+1){
            int tf = DFS(it.v, min(nf,it.f));
            res += tf; nf -= tf; it.f -= tf;
            E[it.v][it.re].f += tf;
            if (nf == 0) return res;
        }
    }
    if (!res) level[u] = -1;
    return res;
}
int flow(int res=0){
    while (BFS())
        res += DFS(s,2147483647);
    return res;
}
}flow;

```

### 4.3 Cost Flow

```

typedef pair<long long, long long> pll;
struct CostFlow {
    static const int MXN = 205;
    static const long long INF = 102938475610293847LL;
    struct Edge {
        int v, r;
        long long f, c;
    };
    int n, s, t, prv[MXN], prvL[MXN], inq[MXN];
    long long dis[MXN], fl, cost;
    vector<Edge> E[MXN];
    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
        fl = cost = 0;
    }
    void add_edge(int u, int v, long long f, long long c) {
        E[u].PB({v, SZ(E[v]), f, c});
        E[v].PB({u, SZ(E[u])-1, 0, -c});
    }
    pll flow() {
        while (true) {
            for (int i=0; i<n; i++) {
                dis[i] = INF;
                inq[i] = 0;
            }
            dis[s] = 0;
            queue<int> que;
            que.push(s);
            while (!que.empty()) {
                int u = que.front(); que.pop();
                inq[u] = 0;
                for (int i=0; i<SZ(E[u]); i++) {
                    int v = E[u][i].v;
                    long long w = E[u][i].c;
                    if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
                        prv[v] = u; prvL[v] = i;
                        dis[v] = dis[u] + w;
                        if (!inq[v]) {
                            inq[v] = 1;
                            que.push(v);
                        }
                    }
                }
            }
        }
    }
}

```



```

    }
    if (dis[t] == INF) break;
    long long tf = INF;
    for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        tf = min(tf, E[u][l].f);
    }
    for (int v=t, u, l; v!=s; v=u) {
        u=prv[v]; l=prvL[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
    }
    cost += tf * dis[t];
    fl += tf;
}
return {fl, cost};
}
}flow;

```

#### 4.4 Bipartite Matching (Augmenting Path)

```

bool DFS(int u){
    for (auto v : E[u]){
        if (!vst[v]){
            vst[v]=1;
            if (match[v] == -1 || DFS(match[v])){
                match[v] = u; match[u] = v;
                return true;
            }
        }
    }
    return false;
}

int DoMatch(int res=0){
    memset(match,-1,sizeof(match));
    for (int i=1; i<=N; i++){
        if (match[i] == -1){
            memset(vst,0,sizeof(vst));
            DFS(i);
        }
    }
    for (int i=1; i<=N; i++){
        if (match[i] != -1) res++;
    }
    return res;
}

```

#### 4.5 Kuhn Munkres

```

struct KM{
    // Maximum Bipartite Weighted Matching (Perfect Match)
    static const int MXN = 650;
    static const int INF = 2147483647; // long long
    int n,match[MXN],vx[MXN],vy[MXN];
    int edge[MXN][MXN],lx[MXN],ly[MXN],slack[MXN];
    // ^^^^ long long
    void init(int _n){
        n = _n;
        for (int i=0; i<n; i++){
            for (int j=0; j<n; j++){
                edge[i][j] = 0;
            }
        }
    }
    void add_edge(int x, int y, int w){ // long long
        edge[x][y] = w;
    }
    bool DFS(int x){
        vx[x] = 1;
        for (int y=0; y<n; y++){
            if (vy[y]) continue;
            if (lx[x]+ly[y] > edge[x][y]){
                slack[y] = min(slack[y], lx[x]+ly[y]-edge[x][y]);
            }
        }
        else {
            vy[y] = 1;
            if (match[y] == -1 || DFS(match[y])){
                match[y] = x;
                return true;
            }
        }
    }
}

```

```

    }
    return false;
}

int solve(){
    fill(match,match+n,-1);
    fill(lx,lx+n,-INF);
    fill(ly,ly+n,0);
    for (int i=0; i<n; i++){
        for (int j=0; j<n; j++){
            lx[i] = max(lx[i], edge[i][j]);
        }
    }
    for (int i=0; i<n; i++){
        fill(slack,slack+n,INF);
        while (true){
            fill(vx,vx+n,0);
            fill(vy,vy+n,0);
            if (DFS(i)) break;
            int d = INF; // long long
            for (int j=0; j<n; j++){
                if (!vy[j]) d = min(d, slack[j]);
            }
            for (int j=0; j<n; j++){
                if (vx[j]) lx[j] -= d;
                if (vy[j]) ly[j] += d;
                else slack[j] -= d;
            }
        }
    }
    int res=0;
    for (int i=0; i<n; i++){
        res += edge[match[i]][i];
    }
    return res;
}
}graph;

```

#### 4.6 SW-Mincut

```

struct SW{ // O(V^3)
    static const int MXN = 514;
    int n,vst[MXN],del[MXN];
    int edge[MXN][MXN],wei[MXN];
    void init(int _n){
        n = _n;
        FZ(edge);
        FZ(del);
    }
    void add_edge(int u, int v, int w){
        edge[u][v] += w;
        edge[v][u] += w;
    }
    void search(int &s, int &t){
        FZ(vst); FZ(wei);
        s = t = -1;
        while (true){
            int mx=-1, cur=0;
            for (int i=0; i<n; i++){
                if (!del[i] && !vst[i] && mx<wei[i]){
                    cur = i, mx = wei[i];
                }
            }
            if (mx == -1) break;
            vst[cur] = 1;
            s = t;
            t = cur;
            for (int i=0; i<n; i++){
                if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
            }
        }
    }
    int solve(){
        int res = 2147483647;
        for (int i=0,x,y; i<n-1; i++){
            search(x,y);
            res = min(res,wei[y]);
            del[y] = 1;
            for (int j=0; j<n; j++){
                edge[x][j] = (edge[j][x] += edge[y][j]);
            }
        }
        return res;
    }
}graph;

```

#### 4.7 Maximum Simple Graph Matching

```

struct GenMatch { // 1-base
    static const int MAXN = 250;
    int V;
    bool el[MAXN][MAXN];
    int pr[MAXN];
    bool inq[MAXN], inp[MAXN], inb[MAXN];
    queue<int> qe;
    int st, ed;
    int nb;
    int bk[MAXN], djs[MAXN];
    int ans;
    void init(int _V) {
        V = _V;
        FZ(el); FZ(pr);
        FZ(inq); FZ(inp); FZ(inb);
        FZ(bk); FZ(djs);
        ans = 0;
    }
    void add_edge(int u, int v) {
        el[u][v] = el[v][u] = 1;
    }
    int lca(int u, int v) {
        memset(inp, 0, sizeof(inp));
        while(1) {
            u = djs[u];
            inp[u] = true;
            if(u == st) break;
            u = bk[pr[u]];
        }
        while(1) {
            v = djs[v];
            if(inp[v]) return v;
            v = bk[pr[v]];
        }
        return v;
    }
    void upd(int u) {
        int v;
        while(djs[u] != nb) {
            v = pr[u];
            inb[djs[u]] = inb[djs[v]] = true;
            u = bk[v];
            if(djs[u] != nb) bk[u] = v;
        }
    }
    void blo(int u, int v) {
        nb = lca(u, v);
        memset(inb, 0, sizeof(inb));
        upd(u); upd(v);
        if(djs[u] != nb) bk[u] = v;
        if(djs[v] != nb) bk[v] = u;
        for(int tu = 1; tu <= V; tu++)
            if(inb[djs[tu]]) {
                djs[tu] = nb;
                if(!inq[tu]){
                    qe.push(tu);
                    inq[tu] = 1;
                }
            }
    }
    void flow() {
        memset(inq, false, sizeof(inq));
        memset(bk, 0, sizeof(bk));
        for(int i = 1; i <= V; i++)
            djs[i] = i;

        while(qe.size()) qe.pop();
        qe.push(st);
        inq[st] = 1;
        ed = 0;
        while(qe.size()) {
            int u = qe.front(); qe.pop();
            for(int v = 1; v <= V; v++)
                if(el[u][v] && (djs[u] != djs[v]) && (pr[u] != v)) {
                    if((v == st) || ((pr[v] > 0) && bk[pr[v]] > 0))
                        blo(u, v);
                    else if(bk[v] == 0) {
                        bk[v] = u;
                        if(pr[v] > 0) {
                            if(!inq[pr[v]]) qe.push(pr[v]);

```

```

                        } else {
                            ed = v;
                            return;
                        }
                    }
                }
            }
        }
        void aug() {
            int u, v, w;
            u = ed;
            while(u > 0) {
                v = bk[u];
                w = pr[v];
                pr[v] = u;
                pr[u] = v;
                u = w;
            }
        }
        int solve() {
            memset(pr, 0, sizeof(pr));
            for(int u = 1; u <= V; u++)
                if(pr[u] == 0) {
                    st = u;
                    flow();
                    if(ed > 0) {
                        aug();
                        ans++;
                    }
                }
            return ans;
        }
    };

    int main() {
        gp.init(V);
        for(int i=0; i<E; i++) {
            int u, v;
            cin >> u >> v;
            gp.edge(u, v);
        }
        cout << gp.solve() << endl;
    }
}

```

#### 4.8 Minimum Weight Matching (Clique version)

```

struct Graph {
    // Minimum General Weighted Matching (Perfect Match)
    static const int MXN = 105;

    int n, edge[MXN][MXN];
    int match[MXN], dis[MXN], onstk[MXN];
    vector<int> stk;

    void init(int _n) {
        n = _n;
        FZ(edge);
    }
    void add_edge(int u, int v, int w) {
        edge[u][v] = edge[v][u] = w;
    }
    bool SPFA(int u) {
        if (onstk[u]) return true;
        stk.PB(u);
        onstk[u] = 1;
        for (int v=0; v<n; v++){
            if (u != v && match[u] != v && !onstk[v]){
                int m = match[v];
                if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
                    dis[m] = dis[u] - edge[v][m] + edge[u][v];
                    onstk[v] = 1;
                    stk.PB(v);
                    if (SPFA(m)) return true;
                }
                stk.pop_back();
                onstk[v] = 0;
            }
        }
        onstk[u] = 0;
        stk.pop_back();
    }
}

```

```

    return false;
}

int solve() {
    // find a match
    for (int i=0; i<n; i+=2){
        match[i] = i+1;
        match[i+1] = i;
    }
    while (true){
        int found = 0;
        FZ(dis); FZ(onstk);
        for (int i=0; i<n; i++){
            stk.clear();
            if (!onstk[i] && SPFA(i)){
                found = 1;
                while (SZ(stk)>=2){
                    int u = stk.back(); stk.pop_back();
                    int v = stk.back(); stk.pop_back();
                    match[u] = v;
                    match[v] = u;
                }
            }
        }
        if (!found) break;
    }
    int ret = 0;
    for (int i=0; i<n; i++){
        ret += edge[i][match[i]];
    }
    ret /= 2;
    return ret;
}
}graph;

```

## 4.9 2-Commodity Flow

```

const int MAXN = 64;
const int INF = 1029384756;

int N;
int s1, s2, t1, t2, d1, d2, S, T;
int edge[MAXN][MAXN];
int cap[MAXN][MAXN];

int h[MAXN], gap[MAXN];
bool vis[MAXN];

int isap(int v, int f)
{
    if(v == T) return f;

    if(vis[v]) return 0;
    vis[v] = true;

    for(int i=0; i<N+2; i++)
    {
        if(cap[v][i] <= 0) continue;
        if(h[i] != h[v] - 1) continue;
        int res = isap(i, min(cap[v][i], f));
        if(res > 0)
        {
            cap[v][i] -= res;
            cap[i][v] += res;
            return res;
        }
    }

    gap[h[v]]--;
    if(gap[h[v]] <= 0) h[S] = N + 4;
    h[v]++;
    gap[h[v]]++;

    return 0;
}

int get_flow()
{
    for(int i=0; i<MAXN; i++)
    {
        h[i] = gap[i] = 0;
    }
}

```

```

}
gap[0] = N + 2;

int flow = 0;

while(h[S] <= N + 3)
{
    for(int i=0; i<N+2; i++)
    {
        vis[i] = false;
    }

    int df = isap(S, INF);
    flow += df;
}

return flow;
}

int main()
{
    ios_base::sync_with_stdio(0);

    int TT;
    cin>>TT;
    while(TT-->0)
    {
        cin>>N;
        cin>>s1>>t1>>d1>>s2>>t2>>d2;

        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                edge[i][j] = 0;
            }
        }

        for(int i=0; i<N; i++)
        {
            string s;
            cin>>s;
            for(int j=0; j<N; j++)
            {
                if(s[j] == 'X') edge[i][j] = 0;
                else if(s[j] == 'O') edge[i][j] = 1;
                else if(s[j] == 'N') edge[i][j] = INF;
            }
        }

        int ans = 0;

        S = N;
        T = N + 1;

        //first
        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                cap[i][j] = edge[i][j];
            }
        }

        cap[S][s1] = cap[t1][T] = d1;
        cap[S][s2] = cap[t2][T] = d2;

        ans = get_flow();

        //second
        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                cap[i][j] = edge[i][j];
            }
        }

        cap[S][s1] = cap[t1][T] = d1;
        cap[S][t2] = cap[s2][T] = d2;

        ans = min(ans, get_flow());
    }
}

```

```

    cout<<(ans == d1 + d2 ? "Yes" : "No")<<endl;
}

return 0;
}

```

#### 4.10 (+1) SW-mincut $O(NM)$

```

// {{{ StoerWagner
const int inf=1000000000;
// should be larger than max.possible mincut
class StoerWagner {
public:
    int n,mc; // node id in [0,n-1]
    vector<int> adj[MAXN];
    int cost[MAXN][MAXN];
    int cs[MAXN];
    bool merged[MAXN],sel[MAXN];
    // --8<-- include only if cut is explicitly needed
    DisjointSet djs;
    vector<int> cut;
    // --8<-----
    StoerWagner(int _n):n(_n),mc(inf),djs(_n) {
        for(int i=0;i<n;i++)
            merged[i]=0;
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                cost[i][j]=cost[j][i]=0;
    }
    void append(int v,int u,int c) {
        if(v==u) return;
        if(!cost[v][u]&&c) {
            adj[v].PB(u);
            adj[u].PB(v);
        }
        cost[v][u]+=c;
        cost[u][v]+=c;
    }
    void merge(int v,int u) {
        merged[u]=1;
        for(int i=0;i<n;i++)
            append(v,i,cost[u][i]);
        // --8<-- include only if cut is explicitly
        // needed
        djs.merge(v,u);
        // --8<-----
    }
    void phase() {
        priority_queue<pii> pq;
        for(int v=0;v<n;v++) {
            if(merged[v]) continue;
            cs[v]=0;
            sel[v]=0;
            pq.push({0,v});
        }
        int v,s,pv;
        while(pq.size()) {
            if(cs[pq.top().S]>pq.top().F) {
                pq.pop();
                continue;
            }
            pv=v;
            v=pq.top().S;
            s=pq.top().F;
            pq.pop();
            sel[v]=1;
            for(int i=0;i<adj[v].size();i++) {
                int u=adj[v][i];
                if(merged[u]||sel[u]) continue;
                cs[u]+=cost[v][u];
                pq.push({cs[u],u});
            }
        }
        if(s<mc) {
            mc=s;
            // --8<-- include only if cut is explicitly
            // needed -----

```

```

        cut.clear();
        for(int i=0;i<n;i++)
            if(djs.getrep(i)==djs.getrep(v)) cut.PB(i);
        // --8<-----
    }
    merge(v,pv);
}
int mincut() {
    if(mc==inf) {
        for(int t=0;t<n-1;t++)
            phase();

        return mc;
    }
    // --8<-- include only if cut is explicitly needed
    -----
    vector<int> getcut() { // return one side of the
        // cut
        mincut();
        return cut;
    }
    // --8<-----
};
// }}}

```

## 5 Math

### 5.1 $ax+by=gcd$

```
typedef pair<int, int> pii;

pii gcd(int a, int b){
    if(b == 0) return make_pair(1, 0);
    else{
        int p = a / b;
        pii q = gcd(b, a % b);
        return make_pair(q.second, q.first - q.second * p);
    }
}
```

### 5.2 Chinese Remainder

```
int pfn; // number of distinct prime factors
int pf[MAXNUM]; // prime factor powers
int rem[MAXNUM]; // corresponding remainder
int pm[MAXNUM];
inline void generate_primes() {
    int i, j;
    pnum=1;
    prime[0]=2;
    for(i=3; i<MAXVAL; i+=2) {
        if(!nprime[i]) continue;
        prime[pnum++]=i;
        for(j=i*i; j<MAXVAL; j+=i) nprime[j]=1;
    }
}
inline int inverse(int x, int p) {
    int q, tmp, a=x, b=p;
    int a0=1, a1=0, b0=0, b1=1;
    while(b) {
        q=a/b; tmp=b; b=a-b*q; a=tmp;
        tmp=b0; b0=a0-b0*q; a0=tmp;
        tmp=b1; b1=a1-b1*q; a1=tmp;
    }
    return a0;
}
inline void decompose_mod() {
    int i, p, t=mod;
    pfn=0;
    for(i=0; i<pnum&&prime[i]<=t; i++) {
        p=prime[i];
        if(t%p==0) {
            pf[pfn]=1;
            while(t%p==0) {
                t/=p;
                pf[pfn]*=p;
            }
            pfn++;
        }
    }
    if(t>1) pf[pfn]=t;
}
inline int chinese_remainder() {
    int i, m, s=0;
    for(i=0; i<pfn; i++) {
        m=mod/pf[i];
        pm[i]=(long long)m*inverse(m, pf[i])%mod;
        s=(s+(long long)pm[i]*rem[i])%mod;
    }
    return s;
}
```

### 5.3 Fast Fourier Transform

```
// const int MAXN = 262144;
// (must be 2^k)

typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acos(-1);
const cplx I(0, 1);

cplx omega[MAXN+1];
void pre_fft() {
    for(int i=0; i<=MAXN; i++)
        omega[i] = exp(i * 2 * PI / MAXN * I);
}
void fft(int n, cplx a[], bool inv=false) {
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >= 1) {
        int mh = m >> 1;
        for (int i = 0; i < mh; i++) {
            cplx w = omega[inv ? MAXN-(i*theta%MAXN) : i*theta%MAXN];
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                cplx x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
        }
        theta = (theta * 2) % MAXN;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++) {
        for (int k = n >> 1; k > (i ^ k); k >>= 1);
        if (j < i) swap(a[i], a[j]);
    }
    if (inv)
        for (i = 0; i < n; i++)
            a[i] /= n;
}
```

### 5.4 Fast Linear Recurrence

```
typedef long long ll;

const int MAXN = 514;
const int LG = 61;
const ll MOD = 1000000007;

ll N;
int M;
ll f[LG][MAXN];
ll tmp[MAXN*2];
ll g[MAXN];

void calc_f(ll f0[MAXN], ll f1[MAXN], ll f2[MAXN]) {
    FZ(tmp);
    for(int i=0; i<M; i++)
        for(int j=0; j<M; j++)
            tmp[i+j] = (tmp[i+j] + f0[i] * f1[j]) % MOD;
    for(int i=2*M-2; i>=M; i--) {
        ll mrat = (MOD - tmp[i]) % MOD;
        tmp[i] = 0;
        for(int j=0; j<M; j++)
            tmp[i+j-M+1] = (tmp[i+j-M+1] + mrat * f0[j]) % MOD;
        tmp[i-M] = (tmp[i-M] + mrat * (MOD-1)) % MOD;
    }
    for(int i=0; i<M; i++)
        f2[i] = tmp[i];
}

void make() {
    for(int i=1; i<LG; i++)
        calc_f(f[i-1], f[i-1], f[i]);

    for(int i=0; i<10; i++) {
```

```

    for(int j=0; j<M; j++)
        cout<<f[i][j]<<" ";
    cout<<endl;
}
}
ll calc(ll x) {
    bool any = false;
    for(int i=0; i<LG; i++) {
        if(x&(1LL<<i)) {
            if(!any) {
                any = true;
                for(int j=0; j<M; j++)
                    g[j] = f[i][j];
            } else {
                calc_f(g, f[i], g);
            }
        }
    }

    return g[0];
}
int main() {
    cin>>N>>M;
    for(int i=0; i<M; i++)
        cin>>f[0][i];

    // f[n] = \sum_{0 <= i < M} f[n-i-1]

    make();
    ll ans = calc(N);

    cout<<ans<<endl;

    return 0;
}

```

## 5.5 (+1) ntt

```

int P=605028353,root=3,MAXNUM=262144;
// Remember coefficient are mod P
/*
p=a*2^n+1
n  2^n      p      a      root
5   32      97      3      5
6   64     193      3      5
7  128     257      2      3
8  256     257      1      3
9  512    7681     15     17
10 1024   12289     12     11
11 2048   12289      6     11
12 4096   12289      3     11
13 8192   40961      5      3
14 16384  65537      4      3
15 32768  65537      2      3
16 65536  65537      1      3
17 131072 786433      6     10
18 262144 786433      3     10 (605028353,
    2308, 3)
19 524288 5767169     11      3
20 1048576 7340033      7      3
21 2097152 23068673     11      3
22 4194304 104857601     25      3
23 8388608 167772161     20      3
24 16777216 167772161     10      3
25 33554432 167772161      5      3 (1107296257, 33,
    10)
26 67108864 469762049      7      3
27 134217728 2013265921     15     31
*/
int bigmod(long long a,int b){
    if(b==0)return 1;
    return (bigmod((a*a)%P,b/2)*(b%2?a:1ll))%P;
}
int inv(int a,int b){
    if(a==1)return 1;
    return (((long long)(a-inv(b%a,a))*b+1)/a)%b;
}
std::vector<long long> ps(MAXNUM);
std::vector<int> rev(MAXNUM);
struct poly{

```

```

std::vector<unsigned int> co;
int n;//polynomial degree = n
poly(int d){n=d;co.resize(n+1,0);}
void trans2(int NN){
    int r=0,st,N;
    unsigned int a,b;
    while((1<<r)<(NN>>1))++r;
    for(N=2;N<=NN;N<=1,--r){
        for(st=0;st<NN;st+=N){
            int i,ss=st+(N>>1);
            for(i=(N>>1)-1;i>=0;--i){
                a=co[st+i]; b=(ps[i<<r]*co[ss+i])%P;
                co[st+i]=a+b; if(co[st+i]>=P)co[st+i]-=P;
                co[ss+i]=a+P-b; if(co[ss+i]>=P)co[ss+i]-=P;
            }
        }
    }
}
void trans1(int NN){
    int r=0,st,N;
    unsigned int a,b;
    for(N=NN;N>1;N>=1,++r){
        for(st=0;st<NN;st+=N){
            int i,ss=st+(N>>1);
            for(i=(N>>1)-1;i>=0;--i){
                a=co[st+i]; b=co[ss+i];
                co[st+i]=a+b; if(co[st+i]>=P)co[st+i]-=P;
                co[ss+i]=(a+P-b)*ps[i<<r]%P;
            }
        }
    }
}
poly operator*(const poly& _b)const{
    poly a=*this,b=_b;
    int k=n+b.n,i,N=1;
    while(N<=k)N*=2;
    a.co.resize(N,0); b.co.resize(N,0);
    int r=bigmod(root,(P-1)/N),Ni=inv(N,P);
    ps[0]=1;
    for(i=1;i<N;++i)ps[i]=(ps[i-1]*r)%P;
    a.trans1(N);b.trans1(N);
    for(i=0;i<N;++i)a.co[i]=((long long)a.co[i]*b.co[i]
        )%P;
    r=inv(r,P);
    for(i=1;i<N/2;++i)std::swap(ps[i],ps[N-i]);
    a.trans2(N);
    for(i=0;i<N;++i)a.co[i]=((long long)a.co[i]*Ni)%P;
    a.n=n+_b.n; return a;
}
};

```

## 5.6 Mod

```

// _fd(a,b) floor(a/b).
// _rd(a,m) a-floor(a/m)*m.
// _pv(a,m,r) largest x s.t x<=a && x%m == r.
// _nx(a,m,r) smallest x s.t x>=a && x%m == r.
// _ct(a,b,m,r) |A| , A = { x : a<=x<=b && x%m == r }.

int _fd(int a,int b){ return a<0?(-~a/b-1):a/b; }
int _rd(int a,int m){ return a-_fd(a,m)*m; }
int _pv(int a,int m,int r)
{
    r=(r%m+m)%m;
    return _fd(a-r,m)*m+r;
}
int _nt(int a,int m,int r)
{
    m=abs(m);
    r=(r%m+m)%m;
    return _fd(a-r-1,m)*m+r+m;
}
int _ct(int a,int b,int m,int r)
{
    m=abs(m);
    a=_nt(a,m,r);
    b=_pv(b,m,r);
    return (a>b)?0:((b-a+m)/m);
}

```

## 5.7 (+1) Miller Rabin

```
// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : pimes <= 13
// n < 2^64              7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
long long power(long long x, long long p, long long mod) {
    long long s=1, m=x;
    while(p) {
        if(p&1) s=mult(s, m, mod);
        p>>=1;
        m=mult(m, m, mod);
    }
    return s;
}
bool witness(long long a, long long n, long long u, int t)
{
    long long x=power(a, u, n);
    for(int i=0; i<t; i++) {
        long long nx=mult(x, x, n);
        if(nx==1&&x!=1&&x!=n-1) return 1;
        x=nx;
    }
    return x!=1;
}
bool miller_rabin(long long n, int s=100) {
    // iterate s times of witness on n
    // return 1 if prime, 0 otherwise
    if(n<2) return 0;
    if(!(n&1)) return n==2;
    long long u=n-1;
    int t=0;
    // n-1 = u*2^t
    while(!(u&1)) {
        u>>=1;
        t++;
    }
    while(s--) {
        long long a=randll()%(n-1)+1;
        if(witness(a, n, u, t)) return 0;
    }
    return 1;
}
```

## 5.8 Pollard Rho

```
// 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;
    }
}
```

```
}
}
```

## 5.9 Algorithms about Primes

```
/*
 * 12721
 * 13331
 * 14341
 * 75577
 * 123457
 * 222557
 * 556679
 * 999983
 * 1097774749
 * 1076767633
 * 100102021
 * 999997771
 * 1001010013
 * 1000512343
 * 987654361
 * 999991231
 * 999888733
 * 98789101
 * 987777733
 * 999991921
 * 1010101333
 * 1010102101
 * 1000000000039
 * 100000000000037
 * 2305843009213693951
 * 4611686018427387847
 * 9223372036854775783
 * 18446744073709551557
 */
int mu[MX], p_tbl[MX];
vector<int> primes;
void sieve() {
    mu[1] = p_tbl[1] = 1;
    for (int i=2; i<MX; i++) {
        if (!p_tbl[i]) {
            p_tbl[i] = i;
            primes.PB(i);
            mu[i] = -1;
        }
        for (auto p : primes) {
            int x = i*p;
            if (x >= M) break;
            p_tbl[x] = p;
            mu[x] = -mu[i];
            if (i%p==0) {
                mu[x] = 0;
                break;
            }
        }
    }
}
vector<int> factor(int x) {
    vector<int> fac{1};
    while (x > 1) {
        int fn=SZ(fac), p=p_tbl[x], pos=0;
        while (x%p == 0) {
            x /= p;
            for (int i=0; i<fn; i++)
                fac.PB(fac[pos++] * p);
        }
    }
    return fac;
}
```

## 5.10 (+1) PolynomialGenerator

```
class PolynomialGenerator {
    /* for a nth-order polynomial f(x), *
     * given f(0), f(1), ..., f(n) *
     * express f(x) as sigma_i{c_i*C(x,i)} */
}
```

```

public:
    int n;
    vector<long long> coef;
    // initialize and calculate f(x), vector _fx should
    // be
    // filled with f(0) to f(n)
    PolynomialGenerator(int _n, vector<long long> _fx)
        : n(_n)
        , coef(_fx) {
        for(int i=0; i<n; i++)
            for(int j=n; j>i; j--)
                coef[j] -= coef[j-1];
        }
    // evaluate f(x), runs in O(n)
    long long eval(int x) {
        long long m=1, ret=0;
        for(int i=0; i<=n; i++) {
            ret += coef[i]*m;
            m = m*(x-i)/(i+1);
        }
        return ret;
    }
};

```

## 5.11 Pseudoinverse of Square matrix

```

Mat pinv(Mat m)
{
    Mat res = I;

    FZ(used);
    for(int i=0; i<W; i++)
    {
        int piv = -1;
        for(int j=0; j<W; j++)
        {
            if(used[j]) continue;
            if(abs(m.v[j][i]) > EPS)
            {
                piv = j;
                break;
            }
        }
        if(piv == -1)
            continue;
        used[i] = true;
        swap(m.v[piv], m.v[i]);
        swap(res.v[piv], res.v[i]);

        ld rat = m.v[i][i];
        for(int j=0; j<W; j++)
        {
            m.v[i][j] /= rat;
            res.v[i][j] /= rat;
        }

        for(int j=0; j<W; j++)
        {
            if(j == i) continue;
            rat = m.v[j][i];
            for(int k=0; k<W; k++)
            {
                m.v[j][k] -= rat * m.v[i][k];
                res.v[j][k] -= rat * res.v[i][k];
            }
        }

        for(int i=0; i<W; i++)
        {
            if(used[i]) continue;
            for(int j=0; j<W; j++)
                res.v[i][j] = 0;
        }

        return res;
    }
}

```

## 5.12 Simplex

```

const int maxn = 111;
const int maxm = 111;
const double eps = 1E-10;

double a[maxn][maxm], b[maxn], c[maxn], d[maxn][maxm];
double x[maxn];
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[maxn], 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;
}

```



## 5.13 Theorem

### 5.13.1 Lucas' Theorem

For non-negative integer  $n, m$  and prime  $p$ ,  $\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p}$  where  $m_i$  is the  $i$ -th digit of  $m$  in base  $p$ .

### 5.13.2 Sum of Two Squares Thm (Legendre)

For a given positive integer  $n$ , let  
 $D_1 = (\# \text{ of positive integers } d \text{ dividing } N \text{ that } 1 \equiv d \pmod{4})$   
 $D_3 = (\# \text{ of positive integers } d \text{ dividing } N \text{ that } 3 \equiv d \pmod{4})$   
 then  $n$  can be written as a sum of two squares in exactly  
 $R(n) = 4(D_1 - D_3)$  ways.

### 5.13.3 Difference of D1-D3 Thm

let  $n = 2^t \cdot (p_1^{e_1} \cdot \dots \cdot p_r^{e_r}) \cdot \dots \cdot (q_1^{f_1} \cdot \dots \cdot q_s^{f_s})$   
 where  $p_i, q_i$  are primes and  $1 \equiv p_i \pmod{4}, 3 \equiv q_i \pmod{4}$   
 then  $D_1 - D_3 = \begin{cases} (e_1 + 1)(e_2 + 1) \dots (e_r + 1), & \text{if } (f_i)s \text{ all even} \\ 0, & \text{if any } f_i \text{ is odd} \end{cases}$

### 5.13.4 Krush-Kuhn-Tucker Conditions

#### Stationarity

For maximizing  $f(x)$ :  $\nabla f(x^*) = \sum_{i=1}^m \mu_i \nabla g_i(x^*) + \sum_{j=1}^l \lambda_j \nabla h_j(x^*)$   
 For minimizing  $f(x)$ :  $-\nabla f(x^*) = \sum_{i=1}^m \mu_i \nabla g_i(x^*) + \sum_{j=1}^l \lambda_j \nabla h_j(x^*)$

#### Primal feasibility

$g_i(x^*) \leq 0$ , for all  $i = 1, \dots, m$   
 $h_j(x^*) = 0$ , for all  $j = 1, \dots, l$

#### Dual feasibility

$\mu_i \geq 0$ , for all  $i = 1, \dots, m$

#### Complementary slackness

$\mu_i g_i(x^*) = 0$ , for all  $i = 1, \dots, m$

## 6 Geometry

### 6.1 Point operators

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

#define _x first
#define _y second
typedef pair<double, double> pdd;

pdd operator + (const pdd p1, const pdd p2){
    return pdd(p1._x + p2._x, p1._y + p2._y);
}
pdd operator - (const pdd p1, const pdd p2){
    return pdd(p1._x - p2._x, p1._y - p2._y);
}
pdd operator * (const double c, const pdd p){
    return pdd(p._x * c, p._y * c);
}
pdd operator - (const pdd p){
    return (-1.0) * p;
}
double operator * (const pdd p1, const pdd p2){
    return p1._x * p2._x + p1._y * p2._y;
}
double operator % (const pdd p1, const pdd p2){
    return p1._x * p2._y - p2._x * p1._y;
}
```

### 6.2 Intersection of two circles

Let  $O_1 = (x_1, y_1), O_2 = (x_2, y_2)$  be two centers of circles,  $r_1, r_2$  be the radius. If:

$$d = |O_1 - O_2|, u = \frac{1}{2}(O_1 + O_2) + \frac{(r_2^2 - r_1^2)}{2d^2}(O_1 - O_2)$$

$v = \frac{\sqrt{(r_1 + r_2 + d)(r_1 - r_2 + d)(r_1 + r_2 - d)(-r_1 + r_2 + d)}}{2d^2}(y_1 - y_2, -x_1 + x_2)$  then  $u + v, u - v$  are the two intersections of the circles, provided that  $d < r_1 + r_2$ .

```
vector<pdd> interCircle(pdd o1, double r1, pdd o2,
    double r2) {
    ld d2 = (o1 - o2) * (o1 - o2);
    ld d = sqrt(d2);
    if (d > r1+r2) return {};
    pdd u = 0.5*(o1+o2) + ((r2*r2-r1*r1)/(2*d2))*(o1-o2);
    double A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d) *
        (-r1+r2+d));
    pdd v = A / (2*d2) * pdd(o1.S-o2.S, -o1.F+o2.F);
    return {u+v, u-v};
}
```

### 6.3 Intersection of two lines

```
#include<bits/stdc++.h>

using namespace std;
const double EPS = 1e-9;

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2){
    double f1 = (p2 - p1) % (q1 - p1);
    double f2 = (p2 - p1) % (p1 - q2);
    double f = (f1 + f2);

    if(fabs(f) < EPS) return pdd(nan(""), nan(""));

    return (f2 / f) * q1 + (f1 / f) * q2;
}
```

### 6.4 Half Plane Intersection

```
#include<bits/stdc++.h>

using namespace std;

#define PB push_back
#define SZ(x) ((int)(x).size())
#define ALL(x) begin(x), end(x)
#define F first
#define S second
#define x first
#define y second
#define IOS ios::sync_with_stdio(0);cin.tie(0)

const double EPS = 1E-9;

struct point : public pair<double, double> {
    point() {}
    point(double _x, double _y) : pair<double, double>(_x, _y) {}

    point operator + (const point he) const {
        return {x+he.x, y+he.y};
    }
    point operator - (const point he) const {
        return {x-he.x, y-he.y};
    }
    friend point operator * (const double c, const point &p) {
        return {c*p.x, c*p.y};
    }
    point operator * (const double c) const {
        return {c*x, c*y};
    }
    double operator * (const point he) const {
        return x*he.x + y*he.y;
    }
    double cross (const point &he) const {
        return x*he.y - y*he.x;
    }
    friend ostream& operator << (ostream& o, const point &p) {
        cout << "(" << p.x << ", " << p.y << ")";
        return o;
    }
};
```

```

typedef pair<point, point> Line;

point interPnt(Line l1, Line l2, bool &res){
    point p1, p2, q1, q2;
    tie(p1, p2) = l1;
    tie(q1, q2) = l2;
    double f1 = (p2 - p1).cross(q1 - p1);
    double f2 = (p2 - p1).cross(p1 - q2);
    double f = (f1 + f2);

    if(fabs(f) < EPS) {
        res = false;
        return {0, 0};
    }

    res = true;
    return (f2 / f) * q1 + (f1 / f) * q2;
}

bool isin(Line l0, Line l1, Line l2) {
    // Check inter(l1, l2) in l0
    bool res;
    point p = interPnt(l1, l2, res);
    return (l0.S - l0.F).cross(p - l0.F) > 1e-9;
}

/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.S - l.F).cross(p - l.F) >
 * 0
 */
vector<Line> halfPlaneInter(vector<Line> lines) {
    int sz = lines.size();
    vector<double> ata(sz), ord(sz);
    for (int i=0; i<sz; i++) {
        ord[i] = i;
        point d = lines[i].S - lines[i].F;
        ata[i] = atan2(d.y, d.x);
    }
    sort(ALL(ord), [&](int i, int j) {
        return ata[i] < ata[j];
    });
    vector<Line> fin;
    for (int i=0; i<sz; i++) {
        if (i and fabs(ata[ord[i]] - ata[ord[i-1]]) <
            EPS) {
            Line li = lines[ord[i]];
            Line lj = lines[ord[i-1]];
            if ((li.S - li.F).cross(lj.S - lj.F) >= 0)
                continue;
            else
                fin.back() = li;
        } else {
            fin.PB(lines[ord[i]]);
        }
    }

    deque<Line> dq;
    for (int i=0; i<SZ(fin); i++) {
        while(SZ(dq) >= 2 and
            not isin(fin[i], dq[SZ(dq)-2], dq[SZ(dq)-1])) {
            dq.pop_back();
        }
        while(SZ(dq) >= 2 and
            not isin(fin[i], dq[0], dq[1])) {
            dq.pop_front();
        }
        dq.push_back(fin[i]);
    }

    while (SZ(dq) >= 3 and
        not isin(dq[0], dq[SZ(dq)-2], dq[SZ(dq)-1])) {
        dq.pop_back();
    }

    while (SZ(dq) >= 3 and
        not isin(dq[SZ(dq)-1], dq[0], dq[1])) {
        dq.pop_front();
    }
}

```

```

vector<Line> res(SZ(dq));
copy(ALL(dq), res.begin());
return res;
}

```

## 6.5 Convex Hull

```

double cross(pdd o, pdd a, pdd b){
    return (a-o) % (b-o);
}

vector<pdd> convex_hull(vector<pdd> pt){
    sort(pt.begin(), pt.end());
    int top=0;
    vector<pdd> stk(2*pt.size());
    for (int i=0; i<(int)pt.size(); i++){
        while (top >= 2 && cross(stk[top-2], stk[top-1], pt[i])
            <= 0)
            top--;
        stk[top++] = pt[i];
    }
    for (int i=pt.size()-2, t=top+1; i>=0; i--){
        while (top >= t && cross(stk[top-2], stk[top-1], pt[i])
            <= 0)
            top--;
        stk[top++] = pt[i];
    }
    stk.resize(top-1);
    return stk;
}

```

## 6.6 Minimum Covering Circle

```

struct Mcc{
    // return pair of center and r^2
    static const int MAXN = 1000100;
    int n;
    pdd p[MAXN], cen;
    double r2;

    void init(int _n, pdd _p[]){
        n = _n;
        memcpy(p, _p, sizeof(pdd)*n);
    }

    double sqr(double a){ return a*a; }
    double abs2(pdd a){ return a.a; }
    pdd center(pdd p0, pdd p1, pdd p2) {
        pdd a = p1-p0;
        pdd b = p2-p0;
        double c1=abs2(a)*0.5;
        double c2=abs2(b)*0.5;
        double d = a % b;
        double x = p0.x + (c1 * b.y - c2 * a.y) / d;
        double y = p0.y + (a.x * c2 - b.x * c1) / d;
        return pdd(x,y);
    }

    pair<pdd, double> solve(){
        random_shuffle(p, p+n);
        r2=0;
        for (int i=0; i<n; i++){
            if (abs2(cen-p[i]) <= r2) continue;
            cen = p[i];
            r2 = 0;
            for (int j=0; j<i; j++){
                if (abs2(cen-p[j]) <= r2) continue;
                cen = 0.5 * (p[i]+p[j]);
                r2 = abs2(cen-p[j]);
                for (int k=0; k<j; k++){
                    if (abs2(cen-p[k]) <= r2) continue;
                    cen = center(p[i], p[j], p[k]);
                    r2 = abs2(cen-p[k]);
                }
            }
            return {cen, r2};
        }
    }
}mcc;

```

## 6.7 KDTree (Nearest Point)

```
const int MXN = 100005;

struct KDTree {
    struct Node {
        int x,y,x1,y1,x2,y2;
        int id,f;
        Node *L, *R;
    }tree[MXN];
    int n;
    Node *root;

    long long dis2(int x1, int y1, int x2, int y2) {
        long long dx = x1-x2;
        long long dy = y1-y2;
        return dx*dx+dy*dy;
    }

    static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
    static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
    void init(vector<pair<int,int>> ip) {
        n = ip.size();
        for (int i=0; i<n; i++) {
            tree[i].id = i;
            tree[i].x = ip[i].first;
            tree[i].y = ip[i].second;
        }
        root = build_tree(0, n-1, 0);
    }

    Node* build_tree(int L, int R, int dep) {
        if (L>R) return nullptr;
        int M = (L+R)/2;
        tree[M].f = dep%2;
        nth_element(tree+L, tree+M, tree+R+1, tree[M].f ?
            cmpx : cmpy);
        tree[M].x1 = tree[M].x2 = tree[M].x;
        tree[M].y1 = tree[M].y2 = tree[M].y;

        tree[M].L = build_tree(L, M-1, dep+1);
        if (tree[M].L) {
            tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
        }

        tree[M].R = build_tree(M+1, R, dep+1);
        if (tree[M].R) {
            tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
        }

        return tree+M;
    }

    int touch(Node* r, int x, int y, long long d2){
        long long dis = sqrt(d2)+1;
        if (x<r->x1-dis || x>r->x2+dis || y<r->y1-dis || y>
            r->y2+dis)
            return 0;
        return 1;
    }

    void nearest(Node* r, int x, int y, int &mID, long
        long &md2) {
        if (!r || !touch(r, x, y, md2)) return;
        long long d2 = dis2(r->x, r->y, x, y);
        if (d2 < md2 || (d2 == md2 && mID < r->id)) {
            mID = r->id;
            md2 = d2;
        }
        // search order depends on split dim
        if ((r->f == 0 && x < r->x) ||
            (r->f == 1 && y < r->y)) {
            nearest(r->L, x, y, mID, md2);
            nearest(r->R, x, y, mID, md2);
        } else {
            nearest(r->R, x, y, mID, md2);
            nearest(r->L, x, y, mID, md2);
        }
    }
}
```

```
int query(int x, int y) {
    int id = 1029384756;
    long long d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
}

}tree;
```

## 6.8 (+1) MinkowskiSum

```
/* convex hull Minkowski Sum*/
#define INF 1000000000000000LL
class PT{ public:
    long long x,y;
    int POS(){
        if(y==0) return x>0?0:1;
        return y>0?0:1;
    }
};

PT pt[300000],qt[300000],rt[300000];
long long Lx,Rx;
int dn,un;
inline bool cmp(PT a,PT b){
    int pa=a.POS(),pb=b.POS();
    if(pa==pb) return (a^b)>0;
    return pa<pb;
}

int minkowskiSum(int n,int m){
    int i,j,r,p,q,fi,fj;
    for(i=1,p=0;i<n;i++){
        if(pt[i].y<pt[p].y || (pt[i].y==pt[p].y && pt[i].x<
            pt[p].x)) p=i;
    }
    for(i=1,q=0;i<m;i++){
        if(qt[i].y<qt[q].y || (qt[i].y==qt[q].y && qt[i].x<
            qt[q].x)) q=i;
    }
    rt[0]=pt[p]+qt[q];
    r=1; i=p; j=q; fi=fj=0;
    while(1){
        if(((fj&j==q) || ((!fi||i!=p) && cmp(pt[(p+1)%n]-pt
            [p],qt[(q+1)%m]-qt[q]))){
            rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
            p=(p+1)%n;
            fi=1;
        }else{
            rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
            q=(q+1)%m;
            fj=1;
        }
        if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))!=0)
            r
            ++;
        else rt[r-1]=rt[r];
        if(i==p && j==q) break;
    }
    return r-1;
}

void initInConvex(int n){
    int i,p,q;
    long long Ly,Ry;
    Lx=INF; Rx=-INF;
    for(i=0;i<n;i++){
        if(pt[i].x<Lx) Lx=pt[i].x;
        if(pt[i].x>Rx) Rx=pt[i].x;
    }
    Ly=Ry=INF;
    for(i=0;i<n;i++){
        if(pt[i].x==Lx && pt[i].y<Ly){ Ly=pt[i].y; p=i; }
        if(pt[i].x==Rx && pt[i].y<Ry){ Ry=pt[i].y; q=i; }
    }
    for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
    qt[dn]=pt[q]; Ly=Ry=-INF;
    for(i=0;i<n;i++){
        if(pt[i].x==Lx && pt[i].y>Ly){ Ly=pt[i].y; p=i; }
        if(pt[i].x==Rx && pt[i].y>Ry){ Ry=pt[i].y; q=i; }
    }
    for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
    rt[un]=pt[q];
}

inline int inConvex(PT p){
```

```

int L,R,M;
if(p.x<Lx || p.x>Rx) return 0;
L=0;R=dn;
while(L<R-1){ M=(L+R)/2;
  if(p.x<qt[M].x) R=M; else L=M; }
if(tri(qt[L],qt[R],p)<0) return 0;
L=0;R=un;
while(L<R-1){ M=(L+R)/2;
  if(p.x<rt[M].x) R=M; else L=M; }
if(tri(rt[L],rt[R],p)>0) return 0;
return 1;
}
int main(){
  int n,m,i;
  PT p;
  scanf("%d",&n);
  for(i=0;i<n;i++) scanf("%I64d %I64d",&pt[i].x,&pt[i].y);
  scanf("%d",&m);
  for(i=0;i<m;i++) scanf("%I64d %I64d",&qt[i].x,&qt[i].y);
  n=minkowskiSum(n,m);
  for(i=0;i<n;i++) pt[i]=rt[i];
  scanf("%d",&m);
  for(i=0;i<m;i++) scanf("%I64d %I64d",&qt[i].x,&qt[i].y);
  n=minkowskiSum(n,m);
  for(i=0;i<n;i++) pt[i]=rt[i];
  initInConvex(n);
  scanf("%d",&m);
  for(i=0;i<m;i++){
    scanf("%I64d %I64d",&p.x,&p.y);
    p.x*=3; p.y*=3;
    puts(inConvex(p)?"YES":"NO");
  }
}

```

## 7 Stringology

### 7.1 Suffix Array

```

const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX], 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+2;j++) ct[j]+=ct[j-1];
    for(int j=0;j<len;j++) tsa[ct[tp[j][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[j][0]]+=j]=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;
  }
}

```

## 7.2 Suffix Array (SAIS TWT514)

```

struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
    static const int MXN = 300010;
    bool _t[MXN*2];
    int _s[MXN*2], _sa[MXN*2], _c[MXN*2], x[MXN], _p[MXN], _q[MXN*2], hei[MXN], r[MXN];
    int operator [] (int i){ return _sa[i]; }
    void build(int *s, int n, int m){
        memcpy(_s, s, sizeof(int) * n);
        sais(_s, _sa, _p, _q, _t, _c, n, m);
        mkhei(n);
    }
    void mkhei(int n){
        REP(i,n) r[_sa[i]] = i;
        hei[0] = 0;
        REP(i,n) if(r[i]) {
            int ans = i>0 ? max(hei[r[i-1]] - 1, 0) : 0;
            while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans++;
            hei[r[i]] = ans;
        }
    }
    void sais(int *s, int *sa, int *p, int *q, bool *t, int *c, int n, int z){
        bool uniq = t[n-1] = true, neq;
        int nn = 0, nmz = -1, *nsa = sa + n, *ns = s + n, lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
        memcpy(x, c, sizeof(int) * z); \
        XD; \
        memcpy(x + 1, c, sizeof(int) * (z - 1)); \
        REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i]-1]]++] = sa[i]-1; \
        memcpy(x, c, sizeof(int) * z); \
        for(int i = n - 1; i >= 0; i--) if(sa[i] && t[sa[i]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
        MS0(c, z);
        REP(i,n) uniq &= ++c[s[i]] < 2;
        REP(i,z-1) c[i+1] += c[i];
        if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return; }
        for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s[i+1] ? t[i+1] : s[i]<s[i+1]);
        MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[s[i]]]=p[q[i]=nn++]=i);
        REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1]) {
            neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa[i])*sizeof(int));
            ns[q[lst=sa[i]]]=nmz+=neq;
        }
        sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmz + 1);
        MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[p[nsa[i]]]] = p[nsa[i]]);
    }
}sa;

void suffix_array(int* ip, int len) {
    // should padding a zero in the back
    // s is int array, n is array length
    // s[0..n-1] != 0, and s[n] = 0
    // resulting SA will be length n+1
    ip[len++] = 0;
    sa.build(ip, len, 128);
    // original 1-base
    for (int i=0; i<l; i++) {
        hei[i] = sa.hei[i + 1];
        sa[i] = sa._sa[i + 1];
    }
}

```

## 7.3 Aho-Corasick Algorithm

```

struct AAutomata{
    struct Node{
        int cnt,dp;
        Node *go[26], *fail;
        Node (){
            cnt = 0;
            dp = -1;
            memset(go,0,sizeof(go));
            fail = 0;
        }
    };

    Node *root, pool[1048576];
    int nMem;

    Node* new_Node(){
        pool[nMem] = Node();
        return &pool[nMem++];
    }
    void init(){
        nMem = 0;
        root = new_Node();
    }
    void add(const string &str){
        insert(root,str,0);
    }
    void insert(Node *cur, const string &str, int pos){
        if (pos >= (int)str.size()){
            cur->cnt++;
            return;
        }
        int c = str[pos]-'a';
        if (cur->go[c] == 0){
            cur->go[c] = new_Node();
        }
        insert(cur->go[c],str,pos+1);
    }
    void make_fail(){
        queue<Node*> que;
        que.push(root);
        while (!que.empty()){
            Node* fr=que.front();
            que.pop();
            for (int i=0; i<26; i++){
                if (fr->go[i]){
                    Node *ptr = fr->fail;
                    while (ptr && !ptr->go[i]) ptr = ptr->fail;
                    if (!ptr) fr->go[i]->fail = root;
                    else fr->go[i]->fail = ptr->go[i];
                    que.push(fr->go[i]);
                }
            }
        }
    }
};

```

## 7.4 KMP

```

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

void build_fail_function(string B, int *fail) {
    int len = B.length(), pos;
    pos = fail[0] = -1;
    for (int i = 1; i < len; i++) {
        while (pos != -1 and B[pos + 1] != B[i])
            pos = fail[pos];
        if (B[pos + 1] == B[i]) pos++;
        fail[i] = pos;
    }
}

void match(string A, string B, int *fail) {
    int lenA = A.length(), lenB = B.length();
    int pos = -1;
    for (int i = 0; i < lenA; i++) {
        while (pos != -1 and B[pos + 1] != A[i])

```

```

        pos = fail[pos];

        if (B[pos + 1] == A[i]) pos ++;

        if (pos == lenB - 1) {
            // Match ! A[i - lenB + 1, i] = B
            pos = fail[pos];
        }
    }
}

```

## 7.5 Z value

```

void Zval(const char *s, int len, int *z) {
    z[0] = len;
    for (int b=0, i=1; i<len; i++) {
        z[i] = max(min(z[i-b], z[b] + b - i), 0);
        while (s[i + z[i]] == s[z[i]]) z[i] ++;
        if (i+z[i] > b+z[b]) b=i;
    }
}

```

## 7.6 Z value (palindrome ver.)

```

void Zpal(const char *s, int len, int *z) {
    // Only odd palindrome len is considered
    // z[i] means that the longest odd palindrom
    // centered at
    // i is [i-z[i] .. i+z[i]]
    z[0] = 0;
    for (int b=0, i=1; i<len; i++) {
        if (z[b] + b >= i) z[i] = min(z[2*b-i], b+z[b]-i);
        else z[i] = 0;
        while (i+z[i]+1 < len and i-z[i]-1 >= 0 and
            s[i+z[i]+1] == s[i-z[i]-1]) z[i] ++;
        if (z[i] + i > z[b] + b) b = i;
    }
}

```

## 7.7 palindromic tree

```

int len[maxn];
int suffLink[maxn];
int to[maxn][2];
int cnt[maxn];
int numV;
char str[maxn];

int v;

void addLetter(int n)
{
    while (str[n - len[v] - 1] != str[n] )
        v = suffLink[v];
    int u = suffLink[v];
    while (str[n - len[u] - 1] != str[n] )
        u = suffLink[u];
    int u_ = to[u][str[n] - 'a'];
    int v_ = to[v][str[n] - 'a'];
    if (v_ == -1)
    {
        v_ = to[v][str[n] - 'a'] = numV;
        len[numV++] = len[v] + 2;
        suffLink[v_] = u_;
    }
    v = v_;
    cnt[v]++;
}

void init()
{
    memset(to, -1, sizeof to);
    str[0] = '#';
    len[0] = -1;
    len[1] = 0;
}

```

```

len[2] = len[3] = 1;
suffLink[1] = 0;
suffLink[0] = 0;
suffLink[2] = 1;
suffLink[3] = 1;
to[0][0] = 2;
to[0][1] = 3;
numV = 4;
}

```

## 7.8 Lexicographically Smallest Rotation

```

string mcp(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);
}

```

## 7.9 Suffix Automaton

```

// par : fail link
// val : a topological order ( useful for DP )
// go[x] : automata edge ( x is integer in [0,26] )

struct SAM{
    struct State{
        int par, go[26], val;
        State () : par(0), val(0){ FZ(go); }
        State (int _val) : par(0), val(_val){ FZ(go); }
    };
    vector<State> vec;
    int root, tail;

    void init(int arr[], int len){
        vec.resize(2);
        vec[0] = vec[1] = State(0);
        root = tail = 1;
        for (int i=0; i<len; i++)
            extend(arr[i]);
    }
    void extend(int w){
        int p = tail, np = vec.size();
        vec.PB(State(vec[p].val+1));
        for ( ; p && vec[p].go[w]==0; p=vec[p].par)
            vec[p].go[w] = np;
        if (p == 0){
            vec[np].par = root;
        } else {
            if (vec[vec[p].go[w]].val == vec[p].val+1){
                vec[np].par = vec[p].go[w];
            } else {
                int q = vec[p].go[w], r = vec.size();
                vec.PB(vec[q]);
                vec[r].val = vec[p].val+1;
                vec[q].par = vec[np].par = r;
                for ( ; p && vec[p].go[w] == q; p=vec[p].par)
                    vec[p].go[w] = r;
            }
        }
        tail = np;
    }
};

```

## 8 Problems

### 8.1 Find the maximum tangent (x,y is increasing)

```

typedef long long LL;
const int MAXN = 100010;
struct Coord{
    LL x, y;
    Coord operator - (Coord ag) const{
        Coord res;
        res.x = x - ag.x;
        res.y = y - ag.y;
        return res;
    }
}sum[MAXN], pnt[MAXN], ans, calc;

inline bool cross(Coord a, Coord b, Coord c){
    return (c.y - a.y) * (c.x - b.x) > (c.x - a.x) * (c.y - b.y);
}

int main(){
    int n, l, np, st, ed, now;
    scanf("%d %d\n", &n, &l);
    sum[0].x = sum[0].y = np = st = ed = 0;
    for (int i = 1, v; i <= n; i++){
        scanf("%d", &v);
        sum[i].y = sum[i - 1].y + v;
        sum[i].x = i;
    }
    ans.x = now = 1;
    ans.y = -1;
    for (int i = 0; i <= n - l; i++){
        while (np > 1 && cross(pnt[np - 2], pnt[np - 1], sum[i]))
            np--;
        if (np < now && np != 0) now = np;
        pnt[np++] = sum[i];
        while (now < np && !cross(pnt[now - 1], pnt[now], sum[i + l]))
            now++;
        calc = sum[i + l] - pnt[now - 1];
        if (ans.y * calc.x < ans.x * calc.y){
            ans = calc;
            st = pnt[now - 1].x;
            ed = i + l;
        }
    }
    double res = (sum[ed].y - sum[st].y) / (sum[ed].x - sum[st].x);
    printf("%f\n", res);
    return 0;
}

```

## 8.2 Painter

```

#include <bits/stdc++.h>
using namespace std;
#define F first
#define S second
#define PB push_back
#define IOS ios_base::sync_with_stdio(0); cin.tie(0);
#define SZ(x) ((int)((x).size()))
#define ALL(x) begin(x), end(x)
#define REP(i, x) for (int i = 0; i < (x); i++)
#define REP1(i, a, b) for (int i = (a); i <= (b); i++)

typedef long long ll;
typedef pair<ll, ll> pll;

typedef pll Point;
const int MXN = 100005;

Point operator + (const Point &a, const Point &b) {
    return Point(a.F + b.F, a.S + b.S);
}
Point operator - (const Point &a, const Point &b) {
    return Point(a.F - b.F, a.S - b.S);
}
ll operator * (const Point &a, const Point &b) { return a.F * b.F + a.S * b.S; }
ll operator % (const Point &a, const Point &b) { return a.F * b.S - a.S * b.F; }

struct Segment {
    int v, id;

```

```

    Point p, q;
    Segment () {}
    Segment (int _v, int _id, Point _p, Point _q) :
        v(_v), id(_id), p(_p), q(_q) {}
};

bool operator < (const Segment &a, const Segment &b) {
    if (a.p == b.q) return false;
    if (a.q == b.p) return true;
    if (a.p == b.p) return (a.q - a.p) % (b.q - a.p) > 0;
    if (a.q == b.q) return (a.p - a.q) % (b.p - a.q) < 0;
    if (a.p.F == b.p.F) return a.p.S < b.p.S;
    if (a.q.F == b.q.F) return a.q.S < b.q.S;
    if (a.p.F < b.p.F) return (a.q - a.p) % (b.p - a.p) > 0;
    else return (b.q - b.p) % (a.p - b.p) < 0;
}

bool operator == (const Segment &a, const Segment &b) {
    return tie(a.v, a.id, a.p, a.q) == tie(b.v, b.id, b.p, b.q);
}

struct Triangle {
    Point pt[3];
}ip[MXN];

const int MEM = 350004;
struct Treap {
    static Treap nil, mem[MEM], *pmem;
    Treap *l, *r;
    int sum, presum, size;
    Segment seg;
    Treap () : l(&nil), r(&nil), sum(0), presum(0), size(0), seg() {}
    Treap (Segment _val) :
        l(&nil), r(&nil), sum(_val.v), presum(max(_val.v, 0)), size(1), seg(_val) {}
} Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap::mem;

int size(const Treap *t) { return t->size; }
void pull(Treap *t) {
    if (!size(t)) return;
    t->size = size(t->l) + size(t->r) + 1;
    t->sum = t->l->sum + t->seg.v + t->r->sum;
    t->presum = max(t->l->presum, t->l->sum + t->seg.v);
    t->presum = max(t->presum, t->l->sum + t->seg.v + t->r->presum);
}

Treap* merge(Treap *a, Treap *b) {
    if (!size(a)) return b;
    if (!size(b)) return a;
    Treap *t;
    if (rand() % (size(a) + size(b)) < size(a)) {
        t = a;
        t->r = merge(a->r, b);
    } else {
        t = b;
        t->l = merge(a, b->l);
    }
    pull(t);
    return t;
}

void split(Treap *t, int k, Treap *&a, Treap *&b) {
    if (!size(t)) a = b = &Treap::nil;
    else if (size(t->l) + 1 <= k) {
        a = t;
        split(t->r, k - size(t->l) - 1, a->r, b);
        pull(a);
    } else {
        b = t;
        split(t->l, k, a, b->l);
        pull(b);
    }
}

int get_rank(Treap *t, Segment x) {
    if (!size(t)) return 0;
    if (x < t->seg) return get_rank(t->l, x);
    return get_rank(t->r, x) + size(t->l) + 1;
}

Treap* find_leftist(Treap *t) {
    while (size(t->l)) t = t->l;
    return t;
}

Treap* find_rightist(Treap *t) {

```



```

while (size(t->r)) t = t->r;
return t;
}

int N;
vector<int> allx;
vector<Segment> _seg[3*MXN];
#define seg(x) _seg[(x)+100000]

inline void add_seg(Segment s) {
    seg(s.p.F).PB(s);
    if (s.q.F != s.p.F) seg(s.q.F).PB(s);
}

void predo() {
    allx.clear();
    REP(i,N) REP(j,3) {
        seg(ip[i].pt[j].F).clear();
        allx.PB(ip[i].pt[j].F);
    }
    sort(ALL(allx));
    allx.resize(unique(ALL(allx))-begin(allx));
    REP(i,N) {
        sort(ip[i].pt, ip[i].pt+3);
        Point *pt = ip[i].pt;
        Segment seg1 = Segment(1,i,pt[0],pt[1]);
        Segment seg2 = Segment(1,i,pt[0],pt[2]);
        Segment seg3 = Segment(1,i,pt[1],pt[2]);
        if (seg2 < seg1) seg1.v = -1;
        else seg2.v = -1;
        seg3.v = seg1.v;
        add_seg(seg1);
        add_seg(seg2);
        add_seg(seg3);
    }
}

inline int sgn(ll x) { return x < 0 ? -1 : x > 0; }
bool interPnt(Point p1, Point p2, Point q1, Point q2){
    ll c1 = (p2-p1)*(q1-p1), c2 = (p2-p1)*(q2-p1);
    ll c3 = (q2-q1)*(p1-q1), c4 = (q2-q1)*(p2-q1);
    return sgn(c1) * sgn(c2) <= 0 and sgn(c3) * sgn(c4)
        <= 0;
}

bool check_error(Segment a, Segment b) {
    if (a.id == b.id) return false;
    return interPnt(a.p,a.q,b.p,b.q);
}

int solve() {
    Treap::pmem = Treap::mem;
    Treap *rt = &Treap::nil;
    int res = 0;
    for (auto i:allx) {
        for (auto l:seg(i)) {
            int k = get_rank(rt, l);
            Treap *t,*tl,*tm,*tr;
            split(rt,k,tl,tr);
            t = find_rightist(tl);
            if (size(t) and check_error(t->seg,l)) return -1;
            t = find_leftist(tr);
            if (size(t) and check_error(t->seg,l)) return -1;
            rt = merge(tl,tr);
            if (l.p.F == i and l.p.F != l.q.F) {
                k = get_rank(rt, l);
                split(rt,k,tl,tr);
                tm = new (Treap::pmem++) Treap(l);
                rt = merge(merge(tl,tm),tr);
            }
        }
        for (auto l:seg(i)) {
            if (l.q.F == i and l.p.F != l.q.F) {
                Treap *tl,*tm,*tr;
                int k = get_rank(rt, l);
                split(rt,k-1,tl,tm);
                split(tm,1,tm,tr);
                Treap *t1=find_rightist(tl),*t2=find_leftist(tr);
                if (size(t1) and size(t2) and check_error(t1->
                    seg,t2->seg)) return -1;
                rt = merge(tl,tr);
            }
        }
        res = max(res, rt->presum);
    }
}

res++;
return res;
}

int main() {
    IOS;
    int cas = 0;
    while (cin >> N) {
        if (N == -1) break;
        REP(i,N) {
            REP(j,3) cin >> ip[i].pt[j].F >> ip[i].pt[j].S;
        }
        predo();
        int ans = solve();
        cas++;
        cout << "Case " << cas << ": ";
        if (ans == -1) cout << "ERROR\n";
        else cout << ans << " shades\n";
    }

    return 0;
}

```