Contents

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>
```

DataStructure

BIT

```
#include <bits/stdc++.h>
using namespace std;
// 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);
    }
};
int main(){
    int t; cin >> t; while (t--){
        int n; cin >> n;
        int arr[MAXN];
        for (int i = 1 ; i <= n ; i++) cin >> arr[i];
        BIT *sol = new BIT(arr, n);
        char op[10];
        while (cin >> op){
            int a, b;
            if (op[0] == 'E') break;
            if (op[0] == 'Q'){
                cin >> a >> b;
                cout << sol->sum(b) - sol->sum(a-1) <<
                    '\n';
            }
            if (op[0] == 'A'){
                cin >> a >> b;
                sol->add(a, b);
            }
            if (op[0] == 'S'){
                cin >> a >> b;
                sol->add(a, -b);
            }
        }
    }
}
```

DisjointSet

```
#include <bits/stdc++.h>
using namespace std;
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;
    }
};
int main() {

}
```

HeavyLightDecomposition

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

```cpp
#define PB push_back
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);
        }
    }
    void Print() {
        cout << "\tid\tsz\tdep\tpa\ttop\thc\n";
        for (int i = 0; i < V ; i++) {
            Node *u = node[i];
            cout << "G[" << i << "]:\t" << u->id << '\t
                ' << u->sz
                << '\t' << u->dep << '\t' << ( u->pa ?
                    u->pa - _memN : -1 )
                << '\t' << ( u->top ? u->top - _memN :
                    -1 ) << '\t'
                << ( u->hc ? u->hc - _memN : -1 ) << '\
                    n';
        }
    }
    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
    }
};
int main() {
    int n; cin >> n;
    Tree *G = new Tree(n);
    for (int i = 0 ; i < n - 1 ; i++){
        int u, v; cin >> u >> v;
        G->addEdge(u, v);
    }
    G->HLD();
    G->Print();
}
```

LCA

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 5;
const int lgN  = __lg(MAXN) + 5;
const int INF = 0x3f3f3f3f;
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]})
            ;
    }
};
int main() { ios_base::sync_with_stdio(false); cin.tie
    (0);
    int t; cin >> t; while (t--) {
        int n; cin >> n;
        Tree *T = new Tree(n);
        for (int i = 0 ; i < n - 1 ; i++) {
            int u, v; cin >> u >> v;
            T->addEdge(u - 1, v - 1);
        }
        for (int i = 0 ; i < n ; i++)
```

```
            cin >> T->node[i]->v;
        T->solve();
        int q; cin >> q;
        while (q--) {
            int u, v; cin >> u >> v;
            cout << T->query(u - 1, v - 1) << '\n';
        }
        delete T;
    }
}
```

```
    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
        ';
}
```

MO

```
#pragma GCC optimize ("O3")
#include <bits/stdc++.h>
#define F first
#define S second
using namespace std;
const int MAXN = 1e5 + 5;
const int MAXV = 1e5 + 5;
const int MAXQ = 1e6 + 5;
typedef pair<int, int> pii;
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[MAXV], val_cnt[MAXV];
    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);
```

PartitionTree

```
#include <bits/stdc++.h>
using namespace std;
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 -
                l;
        }
        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;
    int 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

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

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

```
#include <bits/stdc++.h>
using namespace std;
#define PB push_back
struct SparseTable{
    vector<vector<int> > data;
    int (*op)(int a, int b);
    SparseTable(int *arr, int n, int (*_op)(int a, int
        b)){
        op = _op;
        int lgN = ceil(__lg(n));
        data.resize(lgN + 2);
        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(op(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]);
    }
};
int getMin(int a, int b){
    return a < b ? a : b;
}
const int MAXN = 1000;
int main(){
    int arr[MAXN], n;
    cin >> n;
    for (int i = 0 ; i < n ; i++) cin >> arr[i];
    SparseTable *sol = new SparseTable(arr, n, getMin);
    int l, r;
    while (cin >> l >> r)
        cout << sol->query(l, r) << '\n';
    delete sol;
}
```

Flow

Dinic

```
#include <bits/stdc++.h>
using namespace std;
#define PB push_back
typedef long long LL;
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;
    }
};
int main(){ ios_base::sync_with_stdio(false); cin.tie
    (0);
    int kase = 0, n; while (cin >> n && n) {
        cout << "Network " << ++kase << '\n';
        Graph *G = new Graph(n);
        int s, t, m; cin >> s >> t >> m;
        while (m--) {
            int u, v; LL c;
            cin >> u >> v >> c;
            G->addEdge(u - 1, v - 1, c);
        }
        cout << "The bandwidth is " << G->maxFlow(s -
            1, t - 1) << ".\n\n";
    }
}
```

MCMF

```
#include <bits/stdc++.h>
using namespace std;
#define PB push_back
#define MP make_pair
#define F first
#define S second
typedef long long LL;
typedef pair<LL, LL> pLL;
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;
    }
};
int main() {

}
```

Geometry

CH

```
#include "Point.cpp"
using namespace std;
vector<P> CH(vector<P> &ps) {   // front() back() is
    same
    sort(ps.begin(), ps.end());
    vector<P> ret; int m = 0;
    for (int i = 0 ; i < ps.size() ; i++) {
        while (m >= 2 && (ps[i] - ret[m - 2]) % (ret[m
            -1] - ret[m - 2]) < -EPS) ret.pop_back(), m
            --;
        ret.push_back(ps[i]), m++;
    }
    for (int i = ps.size() - 2 ; ~i ; i--) {
        while (m >= 2 && (ps[i] - ret[m - 2]) % (ret[m
            - 1] - ret[m - 2]) < -EPS) ret.pop_back(),
            m--;
        ret.push_back(ps[i]); m++;
    }
    return ret;
}
int main() {

}
```

ClosestPair

```
#include "Point.cpp"
using namespace std;
const double INF = 0x3f3f3f3f;
bool cmpx(P a, P b) { return a.x < b.x; }
bool cmpy(P a, P b) { return a.y < b.y; }
pair<P, P> DnC(vector<P> &p, int L, int R) {
    if (R - L <= 1) return make_pair(P(-INF, -INF), P(
        INF, INF));
    int M = (L + R) >> 1;
    pair<P, P> l = DnC(p, L, M);
    pair<P, P> r = DnC(p, M, R);


}
pair<P, P> closestPair(vector<P> &p) {
    sort(p.begin(), p.end(), cmpx);
    return DnC(p, 0, p.size());
}
int main() {

}
```

HalfPlaneInter

```
#include "Point.cpp"
using namespace std;
typedef vector<P> Pg;

int main() {

}
```

ClosestPair

```
#include "Point.cpp"
using namespace std;
const double INF = 0x3f3f3f3f;
bool cmpx(P a, P b) { return a.x < b.x; }
bool cmpy(P a, P b) { return a.y < b.y; }
pair<P, P> DnC(vector<P> &p, int L, int R) {
    if (R - L <= 1) return make_pair(P(-INF, -INF), P(
        INF, INF));
    int M = (L + R) >> 1;
    pair<P, P> l = DnC(p, L, M);
    pair<P, P> r = DnC(p, M, R);


}
pair<P, P> closestPair(vector<P> &p) {
    sort(p.begin(), p.end(), cmpx);
    return DnC(p, 0, p.size());
}
int main() {

}
```

Point

```
#include <bits/stdc++.h>
#define MP make_pair
using namespace std;
using T = double;
const double EPS = 1e-9;
int dcmp(double x) {
    if (fabs(x) < EPS) return 0;
    return x < 0 ? -1 : 1;
}
struct P {
    T x, y; P(T _x=0.0,T _y=0.0):x(_x),y(_y){}
    P operator+(const P&p){return P(x+p.x,y+p.y);}
    P operator-(const P&p){return P(x-p.x,y-p.y);}
    T operator*(const P&p){return x*p.x+y*p.y;}
    T operator%(const P&p){return x*p.y-y*p.x;}
    P operator*(const T c){return P(x*c,y*c);}
    P operator/(const T c){return P(x/c,y/c);}
    bool operator == (const P &p){
        return dcmp(x - p.x) == 0 && dcmp(y - p.y);
    }
    bool operator<(const P&p)const{
        return MP(x,y)<MP(p.x,p.y);
        // return atan2(x,y)<atan2(p.x,p.y);
    }
    T len(){return sqrt(*this**this);}
    T operator^(P&p){return acos(*this*p/len()/p.len())
        ;}
    P normal() { return P(-y, x)/len(); }
    P rotate(double rad) {
        return P(x*cos(rad)-y*sin(rad), x*sin(rad)+y*
            cos(rad));
    }
};
T area(P a, P b, P c) {
    return (a - b) % (a - c) / 2.0;
}
typedef P V;
struct L {
    P p1, p2;
    L(P _p1,P _p2):p1(_p1), p2(_p2){}
    V getV(){return p1-p2;}
};
P LLintersect(L: l1, L l2) {
    assert(dcmp(l1.getV() % l2.getV()) != 0);
    return l1.p1 + l1.getV() * (l2.getV() % (l1.p1 - l2
        .p1)) / (l1.getV() % l2.getV());
}
typedef L S;
```

PointInPoly

```
#include "Point.cpp"
using namespace std;
double interpolate_x(double y, P p1, P p2) {
    if (p1.y == p2.y) return p1.x;
```

```
    return p1.x + (p2.x - p1.x) * (y - p1.y) / (p2.y -
        p1.y);
}
bool PinPs(vector<P> &ps, P p) {
    bool c = false;
    for (int i = ps.size() - 1, j = 0; j < ps.size() ;
        i = j++)
        if ((ps[i].y > p.y) != (ps[j].y > p.y) && p.x <
             interpolate_x(p.y, ps[i], ps[j]))
            c = !c;
    return c;
}
int main() {

}
```

RotatingCaliper

```
#include "Point.cpp"
using namespace std;

int main() {

}
```

SSinter

```
#include "Point.cpp"
using namespace std;
bool PSinter(P p, S s) {
    V v1=p-s.p1,v2=p-s.p2;
    return (fabs(v1%v2)<EPS)&&(v1*v2<=EPS);
}
bool SSinter(S s1, S s2) {
    T c1 = s1.getV() % (s1.p1 - s2.p1);
    T c2 = s1.getV() % (s1.p1 - s2.p2);
    T c3 = s2.getV() % (s2.p1 - s1.p1);
    T c4 = s2.getV() % (s2.p1 - s1.p2);
    if (dcmp(c1 * c2) < 0 && dcmp(c3 * c4) < 0)return
        true;
    if (dcmp(c1) == 0 && PSinter(s2.p1, s1))return true
        ;
    if (dcmp(c2) == 0 && PSinter(s2.p2, s1))return true
        ;
    if (dcmp(c3) == 0 && PSinter(s1.p1, s2))return true
        ;
    if (dcmp(c4) == 0 && PSinter(s1.p2, s2))return true
        ;
    return false;
}
int main() {

}
```

Graph

BCC

```
// #include <bits/stdc++.h>
#include <iostream>
#include <cstring>
#include <vector>
#include <stack>
using namespace std;
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

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 250 + 5;
const int MAXM = MAXN * MAXN / 2;
#define PB push_back
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;
        }
    }
};
int main() {

}
```

CutBridge

```
#include <bits/stdc++.h>
using namespace std;
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);
        }
    }
};
int main() {

}
```

MaximumClique

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 35;
bool G[MAXN][MAXN];
struct Set {
    bool s[MAXN]; int size;
    Set() { memset(s, false, sizeof(s)); size = 0; }
};
int n, m, maximum_clique;
Set intersect(Set S, int u) {
    for (int i = 0 ; i < n ; i++) {
        if (S.s[i] && !G[u][i]) {
            S.s[i] = false;
            S.size--;
        }
    }
    return S;
}
void backtrack(Set R, Set P, Set X) {
    if (P.size == 0) {
        if (X.size == 0) {
            maximum_clique = max(maximum_clique, R.size
                );
        }
        return ;
    }
    int pivot;
    for (pivot = 0 ; pivot < n ; pivot++)
        if (P.s[pivot] || X.s[pivot])
            break;
    for (int i = 0 ; i < n ; i++) {
        if (P.s[i] && !G[pivot][i]) {
            R.s[i] = true; R.size++;
            backtrack(R, intersect(P, i), intersect(X,
                i));
            R.s[i] = false; R.size--;
            P.s[i] = false; P.size--;
            if (!X.s[i]) X.s[i] = true, X.size++;
        }
    }
}
void BK() {
    for (int i = 0 ; i < n ; i++) G[i][i] = false;
    Set R, P, X;
    for (int i = 0 ; i < n ; i++) R.s[i] = false;
    for (int i = 0 ; i < n ; i++) P.s[i] = true;
    for (int i = 0 ; i < n ; i++) X.s[i] = false;
    R.size = 0;
    P.size = n;
    X.size = 0;
    backtrack(R, P, X);
}
int main() {
    while (cin >> n >> m) {
        memset(G, false, sizeof(G));
        maximum_clique = 0;
        for (int i = 0 ; i < m ; i++) {
            int u, v; cin >> u >> v;
            G[u][v] = G[v][u] = true;
        }
        BK();
        cout << maximum_clique << '\n';
    }
}
```

SCC

```
// #include <bits/stdc++.h>
#include <iostream>
#include <stack>
#include <cstring>
#include <vector>
using namespace std;
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++;
        }
    }
}
int main() {

}
```

TreeDiameter

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e4 + 5;
struct Tree {
    int V;
    struct Node : vector<Node*> {
    }_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 diam;
    int diameter() {
        diam = 0;
        dfs(rt, NULL);
        return diam;
    }
    int dfs(Node *u, Node *pa) {
        int h1 = 0, h2 = 0;
        for (auto to : *u) {
            if (pa != to) {
                int h = dfs(to, u) + 1;
                if (h > h1) h2 = h1, h1 = h;
                else if (h > h2) h2 = h;
            }
        }
        diam = max(diam, h1 + h2);
        return h1;
    }
};
int main() {
    int n; cin >> n;
    Tree *G = new Tree(n);
    for (int i = 0 ; i < n - 1 ; i++) {
        int u, v; cin >> u >> v;
        G->addEdge(u - 1, v - 1);
    }
    cout << G->diameter() << '\n';
}
```

Math

bigN

```
#include <bits/stdc++.h>
using namespace std;
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;
    }
};
int main() {

}
```

BSGS

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

```
//#include <bits/stdc++.h>
#include <iostream>
#include <utility>
using namespace std;
typedef long long LL;
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);
}
int main(){

}
```

ExtgcdModInv

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
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;
}

int main(){

}
```

FFT

```
#include <bits/stdc++.h>
using namespace std;
const double pi = atan(1.0)*4;
struct Complex {
    long double x,y;
    Complex(long double _x=0,long double _y=0)
```

```
        :x(_x),y(_y) {}
    Complex operator + (Complex &tt) { return Complex(x
        +tt.x,y+tt.y); }
    Complex operator - (Complex &tt) { return Complex(x
        -tt.x,y-tt.y); }
    Complex operator * (Complex &tt) { return Complex(x
        *tt.x-y*tt.y,x*tt.y+y*tt.x); }
};
void FFT(Complex *a, int n, int rev) {
    // n是大于等于相乘的两个数组长度的2的幂次
    // 从0开始表示长度，对a进行操作
    // rev==1进行DFT，==-1进行IDFT
    for (int i = 1,j = 0; i < n; ++ i) {
        for (int k = n>>1; k > (j^=k); k >>= 1);
        if (i<j) swap(a[i],a[j]);
    }
    for (int m = 2; m <= n; m <<= 1) {
        Complex wm(cos(2*pi*rev/m),sin(2*pi*rev/m));
        for (int i = 0; i < n; i += m) {
            Complex w(1.0,0.0);
            for (int j = i; j < i+m/2; ++ j) {
                Complex t = w*a[j+m/2];
                a[j+m/2] = a[j] - t;
                a[j] = a[j] + t;
                w = w * wm;
            }
        }
    }
    if (rev==-1) {
        for (int i = 0; i < n; ++ i) a[i].x /= n,a[i].y
             /= n;
    }
}
int main(){

}
```

Karatsuba

```
#include <bits/stdc++.h>
using namespace std;
template <typename T>
void karatsuba(int n, T* A, T* B, T* R){ // n = (1<<k)
    memset(R, 0, sizeof(T) * 2 * n);
    if (n <= 16) {
        for (int i =0 ; i < n ; i++)
            for (int j = 0 ; j < n ; j++)
                R[i + j] += A[i] * B[j];
        return ;
    }
    int m = n >> 1;
    karatsuba(m, A, B, R);
    karatsuba(m, A + m, B + m, R + n);
    T* a = new T[m], *b = new T[m], *r = new T[n];
    for (int i = 0 ; i < m ; i++) a[i] = A[i] + A[i + m
        ], b[i] = B[i] + B[i + m];
    karatsuba(m, a, b, r);
    for (int i = 0 ; i < n ; i++) r[i] -= R[i], r[i] -=
         R[i + n];
    for (int i = 0 ; i < n ; i++) R[i + m] += r[i];
    delete [] a; delete [] b; delete [] r;
}
const int MAXV = (1 << 16) + 5;
typedef long long LL;
int main(){

}
```

Matrix

```
template<typename T>
struct Matrix{
  using rt = std::vector<T>;
  using mt = std::vector<rt>;
  using matrix = Matrix<T>;
  int r,c;
  mt m;
  Matrix(int r,int c):r(r),c(c),m(r,rt(c)){}
  rt& operator[](int i){return m[i];}
  matrix operator+(const matrix &a){
    matrix rev(r,c);
    for(int i=0;i<r;++i)
      for(int j=0;j<c;++j)
        rev[i][j]=m[i][j]+a.m[i][j];
    return rev;
  }
  matrix operator-(const matrix &a){
    matrix rev(r,c);
    for(int i=0;i<r;++i)
      for(int j=0;j<c;++j)
        rev[i][j]=m[i][j]-a.m[i][j];
    return rev;
  }
  matrix operator*(const matrix &a){
    matrix rev(r,a.c);
    matrix tmp(a.c,a.r);
    for(int i=0;i<a.r;++i)
      for(int j=0;j<a.c;++j)
        tmp[j][i]=a.m[i][j];
    for(int i=0;i<r;++i)
      for(int j=0;j<a.c;++j)
        for(int k=0;k<c;++k)
          rev.m[i][j]+=m[i][k]*tmp[j][k];
    return rev;
  }
  bool inverse(){
    Matrix t(r,r+c);
    for(int y=0;y<r;y++){
      t.m[y][c+y] = 1;
      for(int x=0;x<c;++x)
        t.m[y][x]=m[y][x];
    }
    if( !t.gas() )
      return false;
    for(int y=0;y<r;y++)
      for(int x=0;x<c;++x)
        m[y][x]=t.m[y][c+x]/t.m[y][y];
    return true;
  }
  T gas(){
    vector<T> lazy(r,1);
    bool sign=false;
    for(int i=0;i<r;++i){
      if( m[i][i]==0 ){
        int j=i+1;
        while(j<r&&!m[j][i])j++;
        if(j==r)continue;
        m[i].swap(m[j]);
        sign=!sign;
      }
      for(int j=0;j<r;++j){
        if(i==j)continue;
        lazy[j]=lazy[j]*m[i][i];
        T mx=m[j][i];
        for(int k=0;k<c;++k)
          m[j][k]=m[j][k]*m[i][i]-m[i][k]*mx;
      }
    }
    T det=sign?-1:1;
    for(int i=0;i<r;++i){
      det = det*m[i][i];
      det = det/lazy[i];
      for(auto &j:m[i])j/=lazy[i];
    }
    return det;
  }
};
```

MillerRabin

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
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;
}

int main(){
    for (int i = 1 ; i < 100 ; i++)
        if (isPrime(i, llsprp, 7))
            cout << i << '\n';
}
```

String

ACAutomaton

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

};
int main(){

}
```

Eertree

```cpp
#include <bits/stdc++.h>
using namespace std;
#define PB push_back
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)];
    }
};
const int MAXLEN = 100;
int main(){
    char input[MAXLEN];
    while (cin >> input){
        Eertree *sol = new Eertree(input);
        cout << sol->maxLen << '\n';
        cout << sol->sz << '\n';
    }
}
```

KMP

```
#include <bits/stdc++.h>
using namespace std;
const int MAXLEN = 1e6 + 5;
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;
}
int main(){
    while (1){
        char input[MAXLEN], search[MAXLEN];
        cin >> input >> search;
        build(input);
        cout << match(input, search) << '\n';
    }
}
```

minRotation

```
#include <bits/stdc++.h>
using namespace std;
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);
}
int main() {
    string s; while (cin >> s) {
        cout << minStringRotate(s) << '\n';
    }
}
```

SAM

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

```
    }
};
const int MAXLEN = 1e5 + 5;
int main() {

}
```

Z

```
#include <bits/stdc++.h>
using namespace std;
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;
        }
    }
}
int main(){
}
```