

# 100% É Pouco, Pagode Importa D+

University of Brasilia

September 14, 2018

## Contents

<b>1</b>	<b>Data Structures</b>	<b>3</b>
1.1	Merge Sort Tree . . . . .	3
1.2	Wavelet Tree . . . . .	3
1.3	Ordered Set . . . . .	3
1.4	Convex Hull Trick . . . . .	4
1.5	Min queue . . . . .	4
1.6	Sparse Table . . . . .	4
1.7	Treap . . . . .	4
<b>2</b>	<b>Math</b>	<b>5</b>
2.1	Euclides Extendido . . . . .	5
2.2	Prefix inverse . . . . .	5
2.3	Pollard Rho . . . . .	5
2.4	Miller Rabin . . . . .	5
2.5	Totiente . . . . .	5
2.6	Mobius Function . . . . .	5
2.7	Mulmod TOP . . . . .	5
2.8	Determinant . . . . .	5
2.9	FFT . . . . .	6
2.10	NTT . . . . .	6
<b>3</b>	<b>Graphs</b>	<b>6</b>
3.1	Dinic . . . . .	6
3.2	Min Cost Max Flow . . . . .	7
3.3	Small to Large . . . . .	8
3.4	Junior e Falta de Ideias . . . . .	8
3.5	Kosaraju . . . . .	8
3.6	Tarjan . . . . .	9
3.7	Max Clique . . . . .	9
3.8	Dominator Tree . . . . .	9
3.9	Min Cost Matching . . . . .	10
<b>4</b>	<b>Strings</b>	<b>10</b>
4.1	Aho Corasick . . . . .	10
4.2	Suffix Array . . . . .	10
4.3	Z Algorithm . . . . .	11
4.4	Prefix function/KMP . . . . .	11
4.5	Min rotation . . . . .	11
4.6	All palindrome . . . . .	11
4.7	Palindromic Tree . . . . .	11
<b>5</b>	<b>Geometry</b>	<b>12</b>
5.1	2D basics . . . . .	12
5.2	Nearest Points . . . . .	14
5.3	Convex Hull . . . . .	14
5.4	Check point inside polygon, borders included . . . . .	14

<b>6</b>	<b>Miscellaneous</b>	<b>15</b>
6.1	LIS . . . . .	15
6.2	DSU rollback . . . . .	15
6.3	Buildings . . . . .	16
6.4	Burnside's Lemma . . . . .	16

```
set ts=4 sw=4 sta nu rnu sc stl+=%F cindent
imap {<CR> {<CR>}<Esc>O
nmap <F2> 0v$%d
syntax on
alias cmp='g++ -Wall -Wformat=2 -Wshadow -Wconversion -
fsanitize=address -fsanitize=undefined -fno-sanitize-
recover -std=c++11'
```

## Data Structures

### Merge Sort Tree

```
struct MergeTree{
    int n;
    vector<vector<int>> st;

    void build(int p, int L, int R, const int v[]){
        if(L == R){
            st[p].push_back(v[L]);
            return;
        }
        int mid = (L+R)/2;
        build(2*p, L, mid, v);
        build(2*p+1, mid+1, R, v);
        st[p].resize(R-L+1);
        merge(st[2*p].begin(), st[2*p].end(),
            st[2*p+1].begin(), st[2*p+1].end(),
            st[p].begin());
    }

    int query(int p, int L, int R, int i, int j, int x)
    const{
        if(L > j || R < i) return 0;
        if(L >= i && R <= j){
            int id = lower_bound(st[p].begin(), st[p].end(),
                x) - st[p].begin();
            return int(st[p].size()) - id;
        }
        int mid = (L+R)/2;
        return query(2*p, L, mid, i, j, x) +
            query(2*p+1, mid+1, R, i, j, x);
    }
};

public:
    MergeTree(int sz, const int v[]): n(sz), st(4*sz){
        build(1, 1, n, v);
    }

    //number of elements >= x on segment [i, j]
    int query(int i, int j, int x) const{
        if(i > j) swap(i, j);
        return query(1, 1, n, i, j, x);
    }
};
```

### Wavelet Tree

```
template<typename T>
class wavelet{
    T L, R;
    vector<int> l;
    vector<T> sum; // <<
    wavelet *lef, *rig;

    int r(int i) const{ return i - l[i]; }

public:
    template<typename ITER>
```

```
    wavelet(ITER bg, ITER en){
        lef = rig = nullptr;
        L = *bg, R = *bg;

        for(auto it = bg; it != en; it++){
            L = min(L, *it), R = max(R, *it);
            if(L == R) return;
        }

        T mid = L + (R - L)/2;
        l.reserve(std::distance(bg, en) + 1);
        sum.reserve(std::distance(bg, en) + 1);
        l.push_back(0), sum.push_back(0);
        for(auto it = bg; it != en; it++){
            l.push_back(l.back() + (*it <= mid)),
            sum.push_back(sum.back() + *it);
        }

        auto tmp = stable_partition(bg, en, [mid](T x){
            return x <= mid;
        });

        if(bg != tmp) lef = new wavelet(bg, tmp);
        if(tmp != en) rig = new wavelet(tmp, en);
    }

    ~wavelet(){
        delete lef;
        delete rig;
    }

    // 1 index, first is 1st
    T kth(int i, int j, int k) const{
        if(L >= R) return L;
        int c = l[j] - l[i-1];
        if(c >= k) return lef->kth(l[i-1]+1, l[j], k);
        else return rig->kth(r(i-1)+1, r(j), k - c);
    }

    // # elements > x on [i, j]
    int cnt(int i, int j, T x) const{
        if(L > x) return j - i + 1;
        if(R <= x || L == R) return 0;
        int ans = 0;
        if(lef) ans += lef->cnt(l[i-1]+1, l[j], x);
        if(rig) ans += rig->cnt(r(i-1)+1, r(j), x);
        return ans;
    }

    // sum of elements <= k on [i, j]
    T sumk(int i, int j, T k){
        if(R <= k) return sum[j] - sum[i-1];
        if(L == R || L > k) return 0;
        int ans = 0;
        if(lef) ans += lef->sumk(l[i-1]+1, l[j], k);
        if(rig) ans += rig->sumk(r(i-1)+1, r(j), k);
        return ans;
    }

    // swap (i, i+1) just need to update "array" l[i]
};
```

### Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#include <ext/pb_ds/detail/standard_policies.hpp>

using namespace __gnu_pbds; // or pb_ds;
```

```
template<typename T, typename B = null_type>
using oset = tree<T, B, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// find_by_order / order_of_key
```

## Convex Hull Trick

```
const ll is_query = -(1LL<<62);
struct Line{
    ll m, b;
    mutable function<const Line*> succ;
    bool operator<(const Line& rhs) const{
        if(rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct Cht : public multiset<Line>{ // maintain max
    bool bad(iterator y){
        auto z = next(y);
        if(y == begin()){
            if(z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if(z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b){
        auto y = insert({ m, b });
        y->succ = [=]{ return next(y) == end() ? 0 : &*next(y); };
        if(bad(y)){ erase(y); return; }
        while(next(y) != end() && bad(next(y))) erase(next(y));
        while(y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x){
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

## Min queue

```
template<typename T>
class minQ{
    deque<tuple<T, int, int> > p;
    T delta;
    int sz;
public:
    minQ() : delta(0), sz(0) {}
    inline int size() const{ return sz; }
    inline void add(T x){ delta += x; }
    inline void push(T x, int id){
        x -= delta, sz++;
        int t = 1;
        while(p.size() > 0 && get<0>(p.back()) >= x)
            t += get<1>(p.back()), p.pop_back();
        p.emplace_back(x, t, id);
    }
    inline void pop(){
        get<1>(p.front())--, sz--;
```

```
        if(!get<1>(p.front())) p.pop_front();
    }
    T getmin() const{ return get<0>(p.front())+delta; }
    int getid() const{ return get<2>(p.front()); }
};
```

## Sparse Table

```
const int N = 100005;

int v[N], n;
int dn[N][20];
int fn(int i, int j){
    if(j == 0) return v[i];
    if(~dn[i][j]) return dn[i][j];
    return dn[i][j] = min(fn(i, j-1), fn(i + (1 << (j-1)), j-1));
}

int lg(int x){ return 31 - __builtin_clz(x); }
```

```
int getmn(int l, int r){ // [l, r]
    int lz = lg(r - l + 1);
    return min(fn(l, lz), fn(r - (1 << lz) + 1, lz));
}
```

## Treap

```
// source: https://github.com/victorsenam/caderno/blob/master/code/treap.cpp
//const int N = ; typedef int num;
num X[N]; int en = 1, Y[N], sz[N], L[N], R[N];
void calc (int u) { // update node given children info
    sz[u] = sz[L[u]] + 1 + sz[R[u]];
    // code here, no recursion
}
void unlaze (int u) {
    if(!u) return;
    // code here, no recursion
}
void split_val(int u, num x, int &l, int &r) { // l gets <= x, r gets > x
    unlaze(u); if(!u) return (void) (l = r = 0);
    if(X[u] <= x) { split_val(R[u], x, l, r); R[u] = l; l = u; }
    else { split_val(L[u], x, l, r); L[u] = r; r = u; }
    calc(u);
}
void split_sz(int u, int s, int &l, int &r) { // l gets first s, r gets remaining
    unlaze(u); if(!u) return (void) (l = r = 0);
    if(sz[L[u]] < s) { split_sz(R[u], s - sz[L[u]] - 1, l, r); R[u] = l; l = u; }
    else { split_sz(L[u], s, l, r); L[u] = r; r = u; }
    calc(u);
}
int merge(int l, int r) { // els on l <= els on r
    unlaze(l); unlaze(r); if(!l || !r) return l + r; int u;
    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
    else { L[r] = merge(l, L[r]); u = r; }
    calc(u); return u;
}
void init(int n=N-1) { // XXX call before using other funcs
    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i] = 1; L[i] = R[i] = 0; }
    random_shuffle(Y + 1, Y + n + 1);
}
```

## Math

### Euclides Extendido

```
// a*x + b*y = gcd(a, b), <gcd, x, y>
tuple<int, int> euclidesExt(int a, int b) {
    if(b == 0) return make_tuple(a, 1, 0);
    int q, w, e;
    tie(q, w, e) = euclidesExt(b, a % b);
    return make_tuple(q, e, w - e * (a / b));
}
```

### Prefix inverse

```
inv[1] = 1;
for(int i = 2; i < p; i++)
    inv[i] = (p - (p/i) * inv[p%i] % p) % p;
```

### Pollard Rho

```
ll rho(ll n){
    if(n % 2 == 0) return 2;

    ll d, c, x, y;
    do{
        c = llrand() % n, x = llrand() % n, y = x;
        do{
            x = add(mul(x, x, n), c, n);
            y = add(mul(y, y, n), c, n);
            y = add(mul(y, y, n), c, n);
            d = __gcd(abs(x - y), n);
        }while(d == 1);
    }while(d == n);
    return d;
}
```

```
ll pollard_rho(ll n){
    ll x, c, y, d, k;
    int i;
    do{
        i = 1;
        x = llrand() % n, c = llrand() % n;
        y = x, k = 4;
        do{
            if(++i == k) y = x, k *= 2;
            x = add(mul(x, x, n), c, n);
            d = __gcd(abs(x - y), n);
        }while(d == 1);
    }while(d == n);
    return d;
}
```

```
void factorize(ll val, map<ll, int> &fac){
    if(rabin(val)) fac[ val ]++;
    else{
        ll d = pollard_rho(val);
        factorize(d, fac);
        factorize(val / d, fac);
    }
}
```

```
map<ll, int> factor(ll val){
    map<ll, int> fac;
    if(val > 1) factorize(val, fac);
    return fac;
}
```

### Miller Rabin

```
bool rabin(ll n){
```

```
    if(n <= 1) return 0;
    if(n <= 3) return 1;
    ll s = 0, d = n - 1;
    while(d % 2 == 0) d /= 2, s++;
    for(int k = 0; k < 64; k++){
        ll a = (llrand() % (n - 3)) + 2;
        ll x = fexp(a, d, n);
        if(x != 1 && x != n-1){
            for(int r = 1; r < s; r++){
                x = mul(x, x, n);
                if(x == 1) return 0;
                if(x == n-1) break;
            }
            if(x != n-1) return 0;
        }
    }
    return 1;
}
```

### Totiente

```
ll totiente(ll n){
    ll ans = n;
    for(ll i = 2; i*i <= n; i++){
        if(n % i == 0){
            ans = ans / i * (i - 1);
            while(n % i == 0) n /= i;
        }
    }

    if(n > 1) ans = ans / n * (n - 1);
    return ans;
}
```

### Mobius Function

```
memset(mu, 0, sizeof mu);
mu[1] = 1;
for(int i = 1; i < N; i++){
    for(int j = i + i; j < N; j += i)
        mu[j] -= mu[i];
}
// g(n) = sum{f(d)} => f(n) = sum{mu(d)*g(n/d)}
```

### Mulmod TOP

```
constexpr uint64_t mod = (1ull<<61) - 1;
uint64_t modmul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (
        uint32_t)b, h2 = b>>32;
    uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
    uint64_t ret = (l&mod) + (l>>61) + (h << 3) + (m >>
        29) + (m << 35 >> 3) + 1;
    ret = (ret & mod) + (ret>>61);
    ret = (ret & mod) + (ret>>61);
    return ret-1;
}
```

### Determinant

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
    }
}
```

```

        break;
    }
    swap(a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs(a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;

FFT

struct base{
    double r, i;
    base(double _r = 0, double _i = 0) : r(_r), i(_i) {}
    base operator*(base &o) const{
        return {r*_o.r - i*_o.i, r*_o.i + o.r*i};
    }
    double real() const{ return r; }
    void operator*=(const base &o){
        (*this) = {r*_o.r - i*_o.i, r*_o.i + o.r*i};
    }
    void operator+=(const base &o){r += o.r, i += o.i; }
    void operator/=(const double &o){ r /= o, i /= o; }
    void operator-=(const base &o){r -= o.r, i -= o.i; }
    base operator+(const base &o){return {r+o.r,i+o.i};}
    base operator-(const base &o){return {r-o.r,i-o.i};}
};

double PI = acos(-1);

void fft(vector<base> &a, bool inv){
    int n = (int)a.size();

    for(int i = 1, j = 0; i < n; i++){
        int bit = n >> 1;
        for(; j >= bit; bit >>= 1) j -= bit;
        j += bit;
        if(i < j) swap(a[i], a[j]);
    }

    for(int sz = 2; sz <= n; sz <= 1) {
        double ang = 2*PI/sz * (inv ? -1 : 1);
        base wlen(cos(ang), sin(ang));
        for(int i = 0; i < n; i += sz){
            base w(1);
            for(int j = 0; j < sz/2; j++){
                base u = a[i+j], v = a[i+j+sz/2] * w;
                a[i+j] = u + v;
                a[i+j+sz/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv) for(int i = 0; i < n; i++) a[i] /= 1.0 * n;
}

void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res){
    vector<base> fa(a.begin(), a.end());
    vector<base> fb(b.begin(), b.end());
    size_t n = 1;

```

```

    while(n < a.size()) n <= 1;
    while(n < b.size()) n <= 1;
    n <= 1;
    fa.resize(n), fb.resize(n);

    fft(fa, false), fft(fb, false);
    for(size_t i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    res.resize(n);
    for(size_t i = 0; i < n; i++)
        res[i] = int(fa[i].real() + 0.5);
}

```

## NTT

```

const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft(vector<int> &a, bool invert) {
    int n = (int) a.size();

    for (int i=1, j=0; i<n; ++i) {
        int bit = n >> 1;
        for (; j>=bit; bit>>=1)
            j -= bit;
        j += bit;
        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len=2; len<=n; len<=1) {
        int wlen = invert ? root_1 : root;
        for (int i=len; i<root_pw; i<=1)
            wlen = int(wlen * 111 * wlen % mod);
        for (int i=0; i<n; i+=len) {
            int w = 1;
            for (int j=0; j<len/2; ++j) {
                int u = a[i+j], v = int(a[i+j+len/2] * 1
                    11 * w % mod);
                a[i+j] = u+v < mod ? u+v : u+v-mod;
                a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
                w = int(w * 111 * wlen % mod);
            }
        }
    }
    if (invert) {
        int nrev = reverse(n, mod);
        for (int i=0; i<n; ++i)
            a[i] = int(a[i] * 111 * nrev % mod);
    }
}

```

## Graphs

### Dinic

```

const int N = 100005;
const int E = 2000006;
vector<int> g[N];

```

```

int ne;
struct Edge{
    int from, to;
    ll flow, cap;
} edge[E];

```

```

int lvl[N], vis[N], pass, start = N-2, target = N-1;
int qu[N], qt, px[N];

ll run(int s, int sink, ll minE){
    if(s == sink) return minE;

    ll ans = 0;

    for(; px[s] < (int)g[s].size(); px[s]++){
        int e = g[s][ px[s] ];
        auto &v = edge[e], &rev = edge[e^1];
        if(lvl[v.to] != lvl[s]+1 || v.flow >= v.cap)
            continue;
        ll tmp = run(v.to, sink, min(minE, v.cap-v.flow));
        v.flow += tmp, rev.flow -= tmp;
        ans += tmp, minE -= tmp;
        if(minE == 0) break;
    }
    return ans;
}

bool bfs(int source, int sink){
    qt = 0;
    qu[qt++] = source;
    lvl[source] = 1;
    vis[source] = ++pass;

    for(int i = 0; i < qt; i++){
        int u = qu[i];
        px[u] = 0;
        if(u == sink) return true;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.flow >= v.cap || vis[v.to] == pass)
                continue;
            vis[v.to] = pass;
            lvl[v.to] = lvl[u]+1;
            qu[qt++] = v.to;
        }
    }
    return false;
}

ll flow(int source = start, int sink = target){
    ll ans = 0;
    while(bfs(source, sink))
        ans += run(source, sink, oo);
    return ans;
}

void addEdge(int u, int v, ll c = 1, ll rc = 0){
    edge[ne] = {u, v, 0, c};
    g[u].push_back(ne++);
    edge[ne] = {v, u, 0, rc};
    g[v].push_back(ne++);
}

void reset_flow(){
    for(int i = 0; i < ne; i++)
        edge[i].flow = 0;
}

```

## Min Cost Max Flow

```

const ll oo = 1e18;
const int N = 505;

```

```

const int E = 30006;

vector<int> g[N];

int ne;

struct Edge{
    int from, to;
    ll cap, cost;
} edge[E];

int lvl[N], vis[N], pass, source, target, p[N], px[N];

ll d[N];

ll back(int s, ll minE){
    if(s == source) return minE;

    int e = p[s];

    ll f = back(edge[e].from, min(minE, edge[e].cap));
    edge[e].cap -= f;
    edge[e^1].cap += f;
    return f;
}

int dijkstra(){
    forn(i, N) d[i] = oo;

    priority_queue<pair<ll, int> > q;

    d[source] = 0;

    q.emplace(0, source);

    while(!q.empty()){
        ll dis = -q.top().ff;
        int u = q.top().ss; q.pop();

        if(dis > d[u]) continue;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0) continue;
            if(d[u] + v.cost < d[v.to]){
                d[v.to] = d[u] + v.cost;
                p[v.to] = e;
                q.emplace(-d[v.to], v.to);
            }
        }
    }
    return d[target] != oo;
}

pair<ll, ll> mincost(){
    ll ans = 0, mf = 0;
    while(dijkstra()){
        ll f = back(target, oo);
        mf += f;
        ans += f * d[target];
    }
    return {mf, ans};
}

void addEdge(int u, int v, ll c, ll cost){
    edge[ne] = {u, v, c, cost};
    g[u].pb(ne++);
}

```

```
}
```

## Small to Large

```
void cnt_sz(int u, int p = -1){
    sz[u] = 1;

    for(int v : g[u]) if(v != p)
        cnt_sz(v, u), sz[u] += sz[v];
}

void add(int u, int p, int big = -1){
    // Update info about this vx in global answer

    for(int v : g[u]) if(v != p && v != big)
        add(v, u);
}

void dfs(int u, int p, int keep){

    int big = -1, mmx = -1;

    for(int v : g[u]) if(v != p && sz[v] > mmx)
        mmx = sz[v], big = v;

    for(int v : g[u]) if(v != p && v != big)
        dfs(v, u, 0);

    if(big != -1) dfs(big, u, 1);

    add(u, p, big);

    for(auto x : q[u]){
        // answer all queries for this vx
    }

    if(!keep){
        // Remove data from this subtree
    }
}
```

## Junior e Falta de Ideias

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

#define ff first
#define ss second
#define mp make_pair

using namespace std;

typedef long long ll;

vector<pair<int,int>> G[500005];
int subtree[500005], treesize, k;
bool vis[500005];
ll dist[500005], ans;

int dfs(int v, int p){
    subtree[v] = 1;
    for(pair<int,int> x : G[v])
        if(x.ff != p && !vis[x.ff]) subtree[v] += dfs(x.ff, v);
    return subtree[v];
}

int centroid(int v, int p){
    for(pair<int,int> x : G[v]){
        if(x.ff == p || vis[x.ff]) continue;
```

```
        if(subtree[x.ff]*2 > treesize) return centroid(x.ff, v);
    }
    return v;
}

void procurar_ans(int v, int p, int d_atual, ll custo){
    ans = min(ans, dist[k-d_atual] + custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            procurar_ans(x.ff, v, d_atual+1, custo+x.ss);
    }
}

void atualiza_distancia(int v, int p, int d_atual, ll custo){
    dist[d_atual] = min(dist[d_atual], custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            atualiza_distancia(x.ff, v, d_atual+1, custo+x.ss);
    }
}

void decomp(int v, int p){
    treesize = dfs(v, v);
    // if(treesize < k) return;
    int cent = centroid(v, v);
    vis[cent] = 1;

    for(int i = 1; i <= treesize; i++)
        dist[i] = 1e18;

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff]){
            procurar_ans(x.ff, cent, 1, x.ss);
            atualiza_distancia(x.ff, cent, 1, x.ss);
        }
    }

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff])
            decomp(x.ff, cent);
    }
}

int main(){
    int n, i, a, b;

    scanf("%d%d", &n, &k);
    for(i = 2; i <= n; i++){
        scanf("%d%d", &a, &b);
        G[i].push_back(mp(a, b));
        G[a].push_back(mp(i, b));
    }
    ans = 1e18;
    decomp(1, -1);

    printf("%lld\n", ans == 1e18 ? -1 : ans);

    return 0;
}

Kosaraju

vector<int> g[N], gt[N], S;
```



```

int vis[N], cor[N], tempo = 1;

void dfs(int u){
    vis[u] = 1;
    for(int v : g[u]) if(!vis[v]) dfs(v);
    S.push_back(u);
}

int e;
void dfst(int u){
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v]) dfst(v);
}

int main(){

    for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);

    e = 0;
    reverse(S.begin(), S.end());
    for(int u : S) if(!cor[u])
        e++, dfst(u);

    return 0;
}

```

## Tarjan

```

int cnt = 0, root;
void dfs(int u, int p = -1){
    low[u] = num[u] = ++t;
    for(int v : g[u]){
        if(!num[v]){
            dfs(v, u);
            if(v == root) cnt++;
            if(low[v] >= num[u]) u PONTO DE ARTICULACAO;
            if(low[v] > num[u]) ARESTA u->v PONTE;
            low[u] = min(low[u], low[v]);
        }
        else if(v != p) low[u] = min(low[u], num[v]);
    }
}

root PONTO DE ARTICULACAO <=> cnt > 1

```

```

void tarjanSCC(int u){
    low[u] = num[u] = cnt++;
    vis[u] = 1;
    S.push_back(u);
    for(int v : g[u]){
        if(!num[v]) tarjanSCC(v);
        if(vis[v]) low[u] = min(low[u], low[v]);
    }
    if(low[u] == num[u]){
        ssc[u] = ++ssc_cnt; int v;
        do{
            v = S.back(); S.pop_back(); vis[v] = 0;
            ssc[v] = ssc_cnt;
        }while(u != v);
    }
}

```

## Max Clique

```

long long adj[N], dp[N];

for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        int x;

```

```

        scanf("%d",&x);
        if(x || i == j)
            adj[i] |= 1LL << j;
    }
}

int resto = n - n/2;
int C = n/2;
for(int i = 1; i < (1 << resto); i++){
    int x = i;
    for(int j = 0; j < resto; j++){
        if(i & (1 << j))
            x &= adj[j + C] >> C;
    }
    if(x == i){
        dp[i] = __builtin_popcount(i);
    }
}

for(int i = 1; i < (1 << resto); i++){
    for(int j = 0; j < resto; j++){
        if(i & (1 << j))
            dp[i] = max(dp[i], dp[i ^ (1 << j)]);
    }
}

int maxCliq = 0;
for(int i = 0; i < (1 << C); i++){
    int x = i, y = (1 << resto) - 1;
    for(int j = 0; j < C; j++){
        if(i & (1 << j))
            x &= adj[j] & ((1 << C) - 1), y &= adj[j] >> C;
    }
    if(x != i) continue;
    maxCliq = max(maxCliq, __builtin_popcount(i) + dp[y]);
}

```

## Dominator Tree

```

vector<int> g[N], gt[N], T[N];
vector<int> S;
int dsu[N], label[N];
int sdom[N], idom[N], dfs_time, id[N];

vector<int> bucket[N];
vector<int> down[N];

void prep(int u){
    S.push_back(u);
    id[u] = ++dfs_time;
    label[u] = sdom[u] = dsu[u] = u;

    for(int v : g[u]){
        if(!id[v])
            prep(v), down[u].push_back(v);
        gt[v].push_back(u);
    }
}

int fnd(int u, int flag = 0){
    if(u == dsu[u]) return u;
    int v = fnd(dsu[u], 1), b = label[ dsu[u] ];
    if(id[ sdom[b] ] < id[ sdom[ label[u] ] ]){
        label[u] = b;
        dsu[u] = v;
    }
    return flag ? v : label[u];
}

void build_dominator_tree(int root, int sz){
    // memset(id, 0, sizeof(int) * (sz + 1));

```

```
// for(int i = 0; i <= sz; i++) T[i].clear();
prep(root);

reverse(S.begin(), S.end());

int w;
for(int u : S){
    for(int v : gt[u]){
        w = fnd(v);
        if(id[ sdom[w] ] < id[ sdom[u] ])
            sdom[u] = sdom[w];
    }
    gt[u].clear();

    if(u != root) bucket[ sdom[u] ].push_back(u);

    for(int v : bucket[u]){
        w = fnd(v);
        if(sdom[w] == sdom[v]) idom[v] = sdom[v];
        else idom[v] = w;
    }
    bucket[u].clear();

    for(int v : down[u]) dsu[v] = u;
    down[u].clear();
}

reverse(S.begin(), S.end());

for(int u : S) if(u != root){
    if(idom[u] != sdom[u]) idom[u] = idom[ idom[u] ];
    T[ idom[u] ].push_back(u);
}

S.clear();
}
```

## Min Cost Matching

```
// Min cost matching
// O(n^2 * m)
// n == nro de linhas
// m == nro de colunas
// n <= m | flow == n
// a[i][j] = custo pra conectar i a j
vector<int> u(n + 1), v(m + 1), p(m + 1), way(m + 1);
for(int i = 1; i <= n; ++i){
    p[0] = i;
    int j0 = 0;
    vector<int> minv(m + 1, oo);
    vector<char> used(m + 1, false);
    do{
        used[j0] = true;
        int i0 = p[j0], delta = oo, j1;
        for(int j = 1; j <= m; ++j)
            if(!used[j]){
                int cur = a[i0][j] - u[i0] - v[j];
                if(cur < minv[j])
                    minv[j] = cur, way[j] = j0;
                if(minv[j] < delta)
                    delta = minv[j], j1 = j;
            }
        for(int j = 0; j <= m; ++j)
            if(used[j])
                u[p[j]] += delta, v[j] -= delta;
            else
                minv[j] -= delta;
    }
}
```

```
j0 = j1;
}while(p[j0] != 0);

do{
    int j1 = way[j0];
    p[j0] = p[j1];
    j0 = j1;
}while(j0);
}

// match[i] = coluna escolhida para linha i
vector<int> match(n + 1);
for(int j = 1; j <= m; ++j)
    match[p[j]] = j;
```

```
int cost = -v[0];
```

## Strings

### Aho Corasick

```
void init_aho(){
    queue<int> q;

    q.push(0);

    while(!q.empty()){
        int t = q.front(); q.pop();

        for(int i = 0; i < 52; i++) if(trie[t][i]){
            int x = trie[t][i];
            Q.push(x);

            if(t){
                fn[x] = fn[t];

                while(fn[x] && trie[fn[x]][i] == 0) fn[x]
                    = fn[fn[x]];
                if(trie[fn[x]][i]) fn[x] = trie[fn[x]][i];
            }
        }
    }
}
```

### Suffix Array

```
char s[N];
int n, sa[N], tsa[N], lcp[N], r[N], nr[N], c[N];

void sort(int k, int mx){
    mx++;
    memset(c, 0, sizeof(int) * mx);
    for(int i = 0; i < n; i++) c[i + k < n ? r[i+k]+1 : 1]++;
    partial_sum(c, c+mx, c);
    int t;
    for(int i = 0; i < n; i++)
        t = sa[i]+k < n ? r[ sa[i]+k ] : 0,
        tsa[ c[t]++ ] = sa[i];
    memcpy(sa, tsa, sizeof(int) * n);
}

void build_sa(){
    for(int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];

    int t = 300, a, b;
    for(int sz = 1; sz < n; sz *= 2){
```

```

sort(sz, t), sort(0, t);
t = nr[ sa[0] ] = 0;
for(int i = 1; i < n; i++){
    a = sa[i]+sz < n ? r[ sa[i]+sz ] : -1;
    b = sa[i-1]+sz < n ? r[ sa[i-1]+sz ] : -1;
    nr[ sa[i] ] = r[ sa[i] ] == r[ sa[i-1] ] && a
        == b ? t : ++t;
}
if(t == n-1) break;
memcpy(r, nr, sizeof(int) * n);
}
}

```

```

void build_lcp(){ // lcp[i] = lcp(s[:i], s[:i+1])
    int k = 0;
    for(int i = 0; i < n; i++) r[ sa[i] ] = i;

    for(int i = 0; i < n; i++){
        if(r[i] == n-1) k = 0;
        else{
            int j = sa[r[i]+1];
            while(i+k < n && j+k < n && s[i+k] == s[j+k])
                k++;
        }
        lcp[r[i]] = k;
        if(k) k--;
    }
}

```

## Z Algorithm

```

vector<int> z_algo(const string &s) {
    int n = s.size(), L = 0, R = 0;
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i <= R) z[i] = min(z[i-L], R - i + 1);
        while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ])
            z[i]++;
        if(i+z[i]-1 > R) L = i, R = i + z[i] - 1;
    }
    return z;
}

```

## Prefix function/KMP

```

vector<int> prefix_function(const string &s){
    int n = s.size();
    vector<int> b(n+1);
    b[0] = -1;
    int i = 0, j = -1;
    while(i < n){
        while(j >= 0 && s[i] != s[j]) j = b[j];
        b[++i] = ++j;
    }
    return b;
}

```

```

void kmp(const string &t, const string &p){
    vector<int> b = prefix_function(p);
    int n = t.size(), m = p.size();
    int j = 0;
    for(int i = 0; i < n; i++){
        while(j >= 0 && t[i] != p[j]) j = b[j];
        j++;
        if(j == m){
            //patern of p found on t
            j = b[j];
        }
    }
}

```

```

}

```

## Min rotation

```

int min_rotation(int *s, int N) {
    REP(i, N) s[N+i] = s[i];

    int a = 0;
    REP(b, N) REP(i, N) {
        if (a+i == b || s[a+i] < s[b+i]) { b += max(0, i-1);
            break; }
        if (s[a+i] > s[b+i]) { a = b; break; }
    }
    return a;
}

```

## All palindrome

```

void manacher(char *s, int N, int *rad) {
    static char t[2*MAX];
    int m = 2*N - 1;

    REP(i, m) t[i] = -1;
    REP(i, N) t[2*i] = s[i];

    int x = 0;
    FOR(i, 1, m) {
        int &r = rad[i] = 0;
        if (i <= x+rad[x]) r = min(rad[x+x-i], x+rad[x]-i);
        while (i-r-1 >= 0 && i+r+1 < m && t[i-r-1] == t[i+r+1]) ++r;
        if (i+r >= x+rad[x]) x = i;
    }

    REP(i, m) if (i-rad[i] == 0 || i+rad[i] == m-1) ++rad[i];
    REP(i, m) rad[i] /= 2;
}

```

## Palindromic Tree

```

const int MAXN = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
char s[MAXN];
node tree[MAXN];
int num; // node 1 - root with len -1, node 2 - root
           with len 0
int suff; // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while(true){
        curlen = tree[cur].len;
        if (pos-1 - curlen >= 0 && s[pos-1 - curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {

```

```

    suff = tree[cur].next[let];
    return false;
}

num++;
suff = num;
tree[num].len = tree[cur].len + 2;
tree[cur].next[let] = num;

if (tree[num].len == 1){
    tree[num].sufflink = 2;
    tree[num].num = 1;
    return true;
}

while (true){
    cur = tree[cur].sufflink;
    curlen = tree[cur].len;
    if(pos-1 - curlen >= 0 && s[pos-1 - curlen] == s[
        pos]){
        tree[num].sufflink = tree[cur].next[let];
        break;
    }
}

tree[num].num = 1 + tree[tree[num].sufflink].num;

return true;
}

void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}

int main() {

    initTree();

    for (int i = 0; i < len; i++) {
        addLetter(i);
    }

    return 0;
}

```

## Geometry

### 2D basics

```

typedef double coord;
double eps = 1e-7;
bool eq(coord a, coord b){ return abs(a - b) <= eps; }

struct vec{
    coord x, y; int id;
    vec(coord a = 0, coord b = 0) : x(a), y(b) {}
    vec operator+(const vec &o) const{
        return {x + o.x, y + o.y};
    }
    vec operator-(const vec &o) const{
        return {x - o.x, y - o.y};
    }
    vec operator*(coord t) const{
        return {x * t, y * t};
    }
    vec operator/(coord t) const{

```

```

        return {x / t, y / t};
    }
    coord operator*(const vec &o) const{ // cos
        return x * o.x + y * o.y;
    }
    coord operator^(const vec &o) const{ // sin
        return x * o.y - y * o.x;
    }
    bool operator==(const vec &o) const{
        return eq(x, o.x) && eq(y, o.y);
    }
    bool operator<(const vec &o) const{
        if(!eq(x, o.x)) return x < o.x;
        return y < o.y;
    }
    coord cross(const vec &a, const vec &b) const{
        return (a-(*this)) ^ (b-(*this));
    }
    int ccw(const vec &a, const vec &b) const{
        coord tmp = cross(a, b);
        return (tmp > eps) - (tmp < -eps);
    }
    coord dot(const vec &a, const vec &b) const{
        return (a-(*this)) * (b-(*this));
    }
    coord len() const{
        return sqrt(x * x + y * y); // <
    }
    double angle(const vec &a, const vec &b) const{
        return atan2(cross(a, b), dot(a, b));
    }
    double tan(const vec &a, const vec &b) const{
        return cross(a, b) / dot(a, b);
    }
    vec unit() const{
        return operator/(len());
    }
    int quad() const{
        if(x > 0 && y >=0) return 0;
        if(x <=0 && y > 0) return 1;
        if(x < 0 && y <=0) return 2;
        return 3;
    }
    bool comp(const vec &a, const vec &b) const{
        return (a - *this).comp(b - *this);
    }
    bool comp(vec b){
        if(quad() != b.quad()) return quad() < b.quad();
        if(!eq(operator^(b), 0)) return operator^(b) > 0;
        return (*this) * (*this) < b * b;
    }
    template<class T>
    void sort_by_angle(T first, T last) const{
        std::sort(first, last, [=](const vec &a, const
            vec &b){
                return comp(a, b);
            });
    }
    vec rot90() const{ return {-y, x}; }
    vec rot(double a) const{
        return {cos(a)*x -sin(a)*y, sin(a)*x +cos(a)*y};
    }
};

struct line{
    coord a, b, c; vec n;
    line(vec q, vec w){ // q.cross(w, (x, y)) = 0

```

```

    a = -(w.y-q.y);
    b = w.x-q.x;
    c = -(a * q.x + b * q.y);
    n = {a, b};
}
coord dist(const vec &o) const{
    return abs(eval(o)) / n.len();
}
bool contains(const vec &o) const{
    return eq(a * o.x + b * o.y + c, 0);
}
coord dist(const line &o) const{
    if(!parallel(o)) return 0;
    if(!eq(o.a * b, o.b * a)) return 0;
    if(!eq(a, 0))
        return abs(c - o.c * a / o.a) / n.len();
    if(!eq(b, 0))
        return abs(c - o.c * b / o.b) / n.len();
    return abs(c - o.c);
}
bool parallel(const line &o) const{
    return eq(n ^ o.n, 0);
}
bool operator==(const line &o) const{
    if(!eq(a*o.b, b*o.a)) return false;
    if(!eq(a*o.c, c*o.a)) return false;
    if(!eq(c*o.b, b*o.c)) return false;
    return true;
}
bool intersect(const line &o) const{
    return !parallel(o) || *this == o;
}
vec inter(const line &o) const{
    if(parallel(o)){
        if(*this == o){ }
        else{ /* dont intersect */ }
    }

    auto tmp = n ^ o.n;
    return {(o.c*b -c*o.b)/tmp, (o.a*c -a*o.c)/tmp};
}
vec at_x(coord x) const{
    return {x, (-c-a*x)/b};
}
vec at_y(coord y) const{
    return {(-c-b*y)/a, y};
}
coord eval(const vec &o) const{
    return a * o.x + b * o.y + c;
}
};

struct segment{
    vec p, q;
    segment(vec a = vec(), vec b = vec()): p(a), q(b) {}
    bool onstrip(const vec &o) const{ // onstrip strip
        return p.dot(o, q) >= -eps && q.dot(o, p) >= -eps;
    }
    coord len() const{
        return (p-q).len();
    }
    coord dist(const vec &o) const{
        if(onstrip(o)) return line(p, q).dist(o);
        return min((o-q).len(), (o-p).len());
    }
    bool contains(const vec &o) const{
        return eq(p.cross(q, o), 0) && onstrip(o);
    }
}

bool intersect(const segment &o) const{
    if(contains(o.p)) return true;
    if(contains(o.q)) return true;
    if(o.contains(q)) return true;
    if(o.contains(p)) return true;
    return p.ccw(q, o.p) * p.ccw(q, o.q) == -1
        && o.p.ccw(o.q, q) * o.p.ccw(o.q, p) == -1;
}
bool intersect(const line &o) const{
    return o.eval(p) * o.eval(q) <= 0;
}
coord dist(const segment &o) const{
    if(line(p, q).parallel(line(o.p, o.q))){
        if(onstrip(o.p) || onstrip(o.q)
            || o.onstrip(p) || o.onstrip(q))
            return line(p, q).dist(line(o.p, o.q));
        else if(intersect(o)) return 0;
        return min(min(dist(o.p), dist(o.q)),
            min(o.dist(p), o.dist(q)));
    }
    coord dist(const line &o) const{
        if(line(p, q).parallel(o))
            return line(p, q).dist(o);
        else if(intersect(o)) return 0;
        return min(o.dist(p), o.dist(q));
    }
};

struct hray{
    vec p, q;
    hray(vec a = vec(), vec b = vec()): p(a), q(b){}
    bool onstrip(const vec &o) const{ // onstrip strip
        return p.dot(q, o) >= -eps;
    }
    coord dist(const vec &o) const{
        if(onstrip(o)) return line(p, q).dist(o);
        return (o-p).len();
    }
    bool intersect(const segment &o) const{
        if(!o.intersect(line(p,q))) return false;
        if(line(o.p, o.q).parallel(line(p,q)))
            return contains(o.p) || contains(o.q);
        return contains(line(p,q).inter(line(o.p,o.q)));
    }
    bool contains(const vec &o) const{
        return eq(line(p, q).eval(o), 0) && onstrip(o);
    }
    coord dist(const segment &o) const{
        if(line(p, q).parallel(line(o.p, o.q))){
            if(onstrip(o.p) || onstrip(o.q))
                return line(p, q).dist(line(o.p, o.q));
            return o.dist(p);
        }
        else if(intersect(o)) return 0;
        return min(min(dist(o.p), dist(o.q)),
            o.dist(p));
    }
    bool intersect(const hray &o) const{
        if(!line(p, q).parallel(line(o.p, o.q)))
            return false;
        auto pt = line(p, q).inter(line(o.p, o.q));
        return contains(pt) && o.contains(pt); // <<
    }
    bool intersect(const line &o) const{
        return eq(p.cross(q, o), 0) && onstrip(o);
    }
}

```

```

    if(line(p, q).parallel(o)) return line(p, q) == o;
    if(o.contains(p) || o.contains(q)) return true;
    return (o.eval(p) >= -eps)^(o.eval(p) < o.eval(q));
    return contains(o.inter(line(p, q)));
}
coord dist(const line &o) const{
    if(line(p,q).parallel(o))
        return line(p,q).dist(o);
    else if(intersect(o)) return 0;
    return o.dist(p);
}
coord dist(const hray &o) const{
    if(line(p, q).parallel(line(o.p, o.q))) {
        if(onstrip(o.p) || o.onstrip(p))
            return line(p,q).dist(line(o.p, o.q));
        return (p-o.p).len();
    }
    else if(intersect(o)) return 0;
    return min(dist(o.p), o.dist(p));
}
};

```

```

double heron(coord a, coord b, coord c){
    coord s = (a + b + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}

```

## Nearest Points

```

struct pt {
    int x, y, id;
};

inline bool cmp_x (const pt & a, const pt & b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

inline bool cmp_y (const pt & a, const pt & b) {
    return a.y < b.y;
}

pt a[MAXN];

double mindist;
int ansa, ansb;

inline void upd_ans (const pt & a, const pt & b) {
    double dist = sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)
        *(a.y-b.y) + .0);
    if (dist < mindist)
        mindist = dist, ansa = a.id, ansb = b.id;
}

void rec (int l, int r) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                upd_ans (a[i], a[j]);
        sort (a+l, a+r+1, &cmp_y);
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec (l, m), rec (m+1, r);
    static pt t[MAXN];
    merge (a+l, a+m+1, a+m+1, a+r+1, t, &cmp_y);
    copy (t, t+r-l+1, a+l);
}

```

```

int tsz = 0;
for (int i=l; i<=r; ++i)
    if (abs (a[i].x - midx) < mindist) {
        for (int j=tsz-1; j>=0 && a[i].y - t[j].y <
            mindist; --j)
            upd_ans (a[i], t[j]);
        t[tsz++] = a[i];
    }
}

```

```

sort (a, a+n, &cmp_x);
mindist = 1E20;
rec (0, n-1);

```

## Convex Hull

```

vector<vec> monotone_chain_ch(vector<vec> P){
    sort(P.begin(), P.end());

    vector<vec> L, U;
    for(auto p : P){
        while(L.size() >= 2 && L[L.size() - 2].cross(L.
            back(), p) < 0)
            L.pop_back();

        L.push_back(p);
    }

    reverse(P.begin(), P.end());
    for(auto p : P){
        while(U.size() >= 2 && U[U.size() - 2].cross(U.
            back(), p) < 0)
            U.pop_back();

        U.push_back(p);
    }

    L.pop_back(), U.pop_back();

    L.reserve(L.size() + U.size());
    L.insert(L.end(), U.begin(), U.end());

    return L;
}

```

## Check point inside polygon, borders included

```

bool below(const vector<vec> &vet, vec p){
    auto it = lower_bound(vet.begin(), vet.end(), p);
    if(it == vet.begin())
        return vet.back().cross(*it, p) < 0;
    if(it == vet.end())
        return prev(it)->cross(vet[0], p) < 0;
    return prev(it)->cross(*it, p) < 0;
}

bool above(const vector<vec> &vet, vec p){
    auto it = lower_bound(vet.begin(), vet.end(), p);
    if(it == vet.begin())
        return vet.back().cross(*it, p) > 0;
    if(it == vet.end())
        return prev(it)->cross(vet[0], p) > 0;
    return prev(it)->cross(*it, p) > 0;
}

// lowerhull, upperhull and point
bool inside_poly(const vector<vec> &lo, const vector<vec>
    &hi, vec p){
}

```

```
    return below(hi, p) && above(lo, p);
}
```

## Miscellaneous

### LIS

```
multiset<int> S;
for(int i = 0; i < n; i++){
    auto it = S.upper_bound(a[i]); // low for inc
    if(it != S.end()) S.erase(it);
    S.insert(a[i]);
}
ans = S.size();
```

### DSU rollback

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

using namespace std;

struct DSU{
    vector<int> sz, p, change;
    vector<tuple<int, int, int>> modifications;
    vector<size_t> saves;
    bool bipartite;

    DSU(int n): sz(n+1, 1), p(n+1), change(n+1),
        bipartite(true){
        iota(p.begin(), p.end(), 0);
    }

    void add_edge(int u, int v){
        if(!bipartite) return;
        int must_change = get_colour(u) == get_colour(v);
        int a = rep(u), b = rep(v);
        if(sz[a] < sz[b]) swap(a, b);
        if(a != b){
            p[b] = a;
            modifications.emplace_back(b, change[b],
                bipartite);
            change[b] ^= must_change;
            sz[a] += sz[b];
        }
        else if(must_change){
            modifications.emplace_back(0, change[0],
                bipartite);
            bipartite = false;
        }
    }

    int rep(int u){
        return p[u] == u ? u : rep(p[u]);
    }

    int get_colour(int u){
        if(p[u] == u) return change[u];
        return change[u] ^ get_colour(p[u]);
    }

    void reset(){
        modifications.clear();
        saves.clear();
        iota(p.begin(), p.end(), 0);
        fill(sz.begin(), sz.end(), 1);
        fill(change.begin(), change.end(), 0);
        bipartite = true;
    }
}
```

```
void rollback(){
    int u = get<0>(modifications.back());
    tie(ignore, change[u], bipartite) = modifications
        .back();
    sz[ p[u] ] -= sz[u];
    p[u] = u;
    modifications.pop_back();
}

void reload(){
    while(modifications.size() > saves.back())
        rollback();
    saves.pop_back();
}

void save(){
    saves.push_back(modifications.size());
}
};

const int N = 100005;
const int B = 318;

int n, m, q;
int x[N], y[N], l[N], r[N], ans[N];

vector<int> qu[N];

int brute(int lef, int rig, DSU &s){
    s.save();
    for(int i = lef; i <= rig; i++)
        s.add_edge(x[i], y[i]);
    int ret = s.bipartite;
    s.reload();
    return ret;
}

int main(){
    scanf("%d %d %d", &n, &m, &q);

    for(int i = 1; i <= m; i++)
        scanf("%d %d", x+i, y+i);

    DSU s(n);
    for(int i = 0; i < q; i++){
        scanf("%d %d", l+i, r+i);
        if(r[i] - l[i] <= B + 10)
            ans[i] = brute(l[i], r[i], s);
        else qu[l[i] / B].push_back(i);
    }

    for(int i = 0; i <= m / B; i++){
        sort(qu[i].begin(), qu[i].end(), [](int a, int b){
            return r[a] < r[b];
        });
        s.reset();

        int R = (i+1)*B-1;

        for(int id : qu[i]){
            while(R < r[id]) ++R, s.add_edge(x[R], y[R]);
            s.save();
            for(int k = l[id]; k < (i+1)*B; k++)
                s.add_edge(x[k], y[k]);
            ans[id] = s.bipartite;
            s.reload();
        }
    }
}
```

```

    }
}

for(int i = 0; i < q; i++)
    printf("%s\n", ans[i] ? "Possible": "Impossible");
}

```

## Buildings

```

// count the number of circular arrays
// of size m, with elements on range
// [1, c**(x*x)]

```

```

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

```

```

#define debug(x) cerr << fixed << #x << " = " << x << endl;
#define ll long long

```

```

const int MOD = 1e9 + 7;
const int MAX = 1e5 + 5;
int dp[MAX];

```

```

inline int add(int a, int b) {
    a += b;
    if(a >= MOD) {
        a -= MOD;
    }
    return a;
}

```

```

inline int sub(int a, int b) {
    a -= b;
    if(0 > a) {
        a += MOD;
    }
    return a;
}

```

```

inline int mult(int a, int b) {
    return (1LL * a * b) % MOD;
}

```

```

int f_exp(int x, int exp) {
    if(exp == 0) {
        return 1;
    }
    else if(exp & 1) {

```

```

        return mult(x, f_exp(x, exp - 1));
    }
    return f_exp(mult(x, x), exp / 2);
}

```

```

inline int inv(int x) {
    return f_exp(x, MOD - 2);
}

```

```

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n, m, c;
    cin >> n >> m >> c;

    int x = f_exp(c, n * n);
    int ans = f_exp(x, m);
    for(int i = 1; i <= m; i++) {
        if(m % i == 0) {
            int y = f_exp(x, i);
            for(int j = 1; j < i; j++) {
                if(i % j == 0) {
                    y = sub(y, mult(j, dp[j]));
                }
            }
            dp[i] = mult(y, inv(i));
            ans = sub(ans, mult(i - 1, dp[i]));
        }
    }

    cout << ans << '\n';

    return 0;
}

```

## Burnside's Lemma

Let  $G$  be a finite group that acts on a set  $X$ . For each  $g$  in  $G$  let  $X^g$  denote the set of elements in  $X$  that are fixed by  $g$  (also said to be left invariant by  $g$ ), i.e.  $X^g = \{x \in X \mid g * x = x\}$ .  
Number of orbits

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

$$\text{Number of necklaces: } N_k(n) = \frac{1}{n} \sum_{d|n} \varphi(d) k^{n/d}.$$

$$\text{Number of bracelets: } B_k(n) = \begin{cases} \frac{1}{2} N_k(n) + \frac{1}{4} (k+1) k^{\frac{n}{2}}, & n \text{ even} \\ \frac{1}{2} N_k(n) + \frac{1}{2} k^{\frac{n+1}{2}}, & n \text{ odd} \end{cases}$$