```
// Ukkonen's algorithm O(n)
const int A = 27; // Alphabet size
struct SuffixTree {
    struct Node { // [l, r) !!!
        int l, r, link, par;
        int nxt[A];
        Node(): l(-1), r(-1), link(-1), par(-1) { fill(nxt, nxt + A, -1); }
        Node(int _l, int _r, int _link, int _par):
        l(_l), r(_r), link(_link), par(_par) { fill(nxt, nxt + A, -1); }
        int &next(int c) { return nxt[c]; }
        int get_len() const { return r - l; }
    };
    struct State { int v, len; };
    vec< Node > t;
    State cur_state;
    vec< int > s;
    SuffixTree(): cur_state({0, 0}) { t.push_back(Node()); }
    // v -> v + s[l, r) !!!
    State go(State st, int l, int r) {
        while(l < r) {
            if(st.len == t[st.v].get_len()) {
                State nx = State({ t[st.v].next(s[l]), 0 });
                if(nx.v == -1) return nx;
                st = nx;
                continue;
            }
            if(s[ t[st.v].l + st.len ] != s[l]) return State({-1, -1});
            if(r - l < t[st.v].get_len() - st.len)
                return State({st.v, st.len + r - l});
            l += t[st.v].get_len() - st.len;
            st.len = t[st.v].get_len();
        }
        return st;
    }
    int get_vertex(State st) {
        if(t[st.v].get_len() == st.len) return st.v;
        if(st.len == 0) return t[st.v].par;
        Node &v = t[st.v];
        Node &pv = t[v.par];
        Node add(v.l, v.l + st.len, -1, v.par);
        // nxt
        pv.next(s[v.l]) = (int)t.size();
        add.next(s[v.l + st.len]) = st.v;
        // par
        v.par = (int)t.size();
        // [l, r)
        v.l += st.len;
        t.push_back(add); // !!!
        return (int)t.size() - 1;
    }
    int get_link(int v) {
        if(t[v].link != -1) return t[v].link;
        if(t[v].par == -1) return 0;
        int to = get_link(t[v].par);
        to = get_vertex(
            go(State({to, t[to].get_len()}), t[v].l + (t[v].par == 0), t[v].r)
        );
        return t[v].link = to;
    }
    void add_symbol(int c) {
        assert(0 <= c && c < A);
        s.push_back(c);
        while(1) {
            State hlp = go( cur_state, (int)s.size() - 1, (int)s.size() );
            if(hlp.v != -1) { cur_state = hlp; break; }
            int v = get_vertex(cur_state);
            Node add((int)s.size() - 1, +inf, -1, v);
            t.push_back(add);
            t[v].next(c) = (int)t.size() - 1;
            cur_state.v = get_link(v);
            cur_state.len = t[cur_state.v].get_len();
            if(!v) break;
        }
    }
};
```

```
const int LOG = 21;
struct SuffixArray {
    string s;
    int n;
    vec< int > p;
    vec< int > c[LOG];
    SuffixArray(): n(0) { }
    SuffixArray(string ss): s(ss) {
        s.push_back(0);
        n = (int)s.size();
        vec< int > pn, cn;
        vec< int > cnt;
        p.resize(n);
        for(int i = 0;i < LOG;i++) c[i].resize(n);
        pn.resize(n);
        cn.resize(n);
        cnt.assign(300, 0);
        for(int i = 0;i < n;i++) cnt[s[i]]++;
        for(int i = 1;i < (int)cnt.size();i++) cnt[i] += cnt[i - 1];
        for(int i = n - 1;i >= 0;i--) p[--cnt[s[i]]] = i;
        for(int i = 1;i < n;i++) {
            c[0][p[i]] = c[0][p[i - 1]];
            if(s[p[i]] != s[p[i - 1]]) c[0][p[i]]++;
        }
        for(int lg = 0, k = 1;k < n;k <<= 1, lg++) {
            for(int i = 0;i < n;i++) {
                if((pn[i] = p[i] - k) < 0) pn[i] += n;
            }
            cnt.assign(n, 0);
            for(int i = 0;i < n;i++) cnt[c[lg][pn[i]]]++;
            for(int i = 1;i < (int)cnt.size();i++) cnt[i] += cnt[i - 1];
            for(int i = n - 1;i >= 0;i--) p[--cnt[c[lg][pn[i]]]] = pn[i];
            for(int l1, r1, l2, r2, i = 1;i < n;i++) {
                cn[p[i]] = cn[p[i - 1]];
                l1 = p[i - 1];
                l2 = p[i];
                if((r1 = l1 + k) >= n) r1 -= n;
                if((r2 = l2 + k) >= n) r2 -= n;
                if(c[lg][l1] != c[lg][l2] || c[lg][r1] != c[lg][r2]) cn[p[i]]++;
            }
            c[lg + 1] = cn;
        }
        p.erase(p.begin(), p.begin() + 1);
        n--;
    }
    int get_lcp(int i, int j) {
        int res = 0;
        for(int lg = LOG - 1;lg >= 0;lg--) {
            if(i + (1 << lg) > n || j + (1 << lg) > n) continue;
            if(c[lg][i] == c[lg][j]) {
                i += (1 << lg);
                j += (1 << lg);
                res += (1 << lg);
            }
        }
        return res;
    }
};
```

```
struct suf_auto {
    vector<int> base, suf, len;
    vector<vector<int>> g;
    int last, sz;
    suf_auto(): base(26, -1), g(1, base), suf(1, -1), len(1, 0), last(0), sz(1){}
    void add_string(const string &s) {
        for (char c : s) {
            c -= 'a';
            int cur = last;
            last = sz++;
            g.push_back(base);
            suf.emplace_back();
            len.push_back(len[cur] + 1);
            while (cur != -1 && g[cur][c] == -1)
                g[cur][c] = last, cur = suf[cur];
            if (cur == -1) {
                suf[last] = 0;
                continue;
            }
            int nx = g[cur][c];
            if (len[nx] == len[cur] + 1) {
                suf[last] = nx;
                continue;
            }
            int cl = sz++;
            g.push_back(g[nx]);
            suf.push_back(suf[nx]);
            len.push_back(len[cur] + 1);
            suf[last] = suf[nx] = cl;
            while (cur != -1 && g[cur][c] == nx)
                g[cur][c] = cl, cur = suf[cur];
        }
    }
};
```

```
vector<int> get_lcp(const string& s, const vector<int>& suf){
    int n = (int)suf.size();
    vector<int> back(n);
    for(int i = 0; i < n; i++) back[suf[i]] = i;
    vector<int> lcp(n - 1);
    for(int i = 0, k = 0; i < n; i++){
        int x = back[i]; k = max(0, k - 1);
        if(x == n - 1){k = 0; continue;}
        while(s[suf[x] + k] == s[suf[x + 1] + k]) k++;
        lcp[x] = k;
    }
    return lcp;
}
```

```
struct Edge {
    int fr, to, id;
    int get(int v) { return v == fr ? to : fr;}};
void dfs(const vector<vector<Edge>> &g, vector<int> &fup, vector<int> &tin,
    vector<int> &used, int &timer, int v, int par = -1) {
    tin[v] = fup[v] = timer++; used[v] = 1;
    for (Edge e : g[v]) {
        int to = e.get(v);
        if (to == par) continue;
        if (used[to]) {fup[v] = min(fup[v], tin[to]);
        } else { dfs(g, fup, tin, used, timer, to, v);
            fup[v] = min(fup[v], fup[to]);}}}
void paintEdges(const vector<vector<Edge>> &g, vector<int> &fup,
    vector<int> &tin, vector<int> &used,
    vector<int> &colors, int v, int curColor, int &maxColor, int par = -1) {
    used[v] = 1;
    for (Edge e : g[v]) {
        int to = e.get(v);
        if (to == par) continue;
        if (!used[to]) {
            if (tin[v] <= fup[to]) { int tmpColor = maxColor++;
                colors[e.id] = tmpColor;
                paintEdges(g, fup, tin, used, colors, to, tmpColor, maxColor, v);
            } else { colors[e.id] = curColor;
                paintEdges(g, fup, tin, used, colors, to, curColor, maxColor, v);}
        } else if (tin[to] < tin[v]) { colors[e.id] = curColor;}}}
vector<vector<Edge>> get2components(const vector<vector<Edge>> &g,
    int m, const vector<Edge> &es) {
    int n = (int)g.size(); vector<int> fup(n), tin(n), used(n);
    vector<int> colors(m);
    int timer; used.assign(n, 0); timer = 0;
    for (int v = 0; v < n; v++) { if (used[v]) continue;
        dfs(g, fup, tin, used, timer, v);}
    used.assign(n, 0);    timer = 0; for (int v = 0; v < n; v++) {
        if (used[v]) continue;
        paintEdges(g, fup, tin, used, colors, v, timer, timer, -1); }
    vector<vector<Edge>> res(timer);
    for (int i = 0; i < m; i++) { res[colors[i]].push_back(es[i]); }
    return res;}
```

```
vector<int> get_z(const string& s) {
    int n = (int) s.size(); vector<int> z(n);
    for (int l = 0, r = -1, i = 1; i < n; i++) {
        z[i] = i <= r ? min(r - i, z[i - l]) : 0;
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
        if (i + z[i] > r) l = i, r = i + z[i];} return z;}
```

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

```
pair<vector<int>, vector<int>> manacker(const string& s) { // -> {d0, d1}. RUN test!
    int n = (int) s.size();
    vector<int> d0(n), d1(n);
    for (int l = 0, r = -1, i = 0; i < n; i++) { // d1
        d1[i] = i <= r ? min(r - i, d1[l + r - i]) : 0;
        while (i >= d1[i] && i + d1[i] < n && s[i - d1[i]] == s[i + d1[i]]) d1[i]++;
        d1[i]--; if (i + d1[i] > r) l = i - d1[i], r = i + d1[i];}
    for (int l = 0, r = -1, i = 0; i < n; i++) {
        d0[i] = i < r ? min(r - i, d0[l + r - i - 1]) : 0;
        while (i >= d0[i] && i + d0[i] + 1 < n && s[i - d0[i]] == s[i + d0[i] + 1]) d0[i]++;
        if (d0[i] > 0 && i + d0[i] > r) l = i - d0[i] + 1, r = i + d0[i];}
    return {d0, d1};}
```

```
const int A = 300; // alphabet size
struct Aho {
    struct Node {
        int nxt[A], go[A];
        int par, pch, link;
        int good;
        Node(): par(-1), pch(-1), link(-1), good(-1) {
            fill(nxt, nxt + A, -1); fill(go, go + A, -1); }
    };
    vec< Node > a;
    Aho() { a.push_back(Node()); }
    void add_string(const string &s) {
        int v = 0;
        for(char c : s) {
            if(a[v].nxt[c] == -1) {
                a[v].nxt[c] = (int)a.size();
                a.push_back(Node());
                a.back().par = v;
                a.back().pch = c;
            }
            v = a[v].nxt[c];
        }
        a[v].good = 1;
    }
    int go(int v, int c) {
        if(a[v].go[c] == -1) {
            if(a[v].nxt[c] != -1) {
                a[v].go[c] = a[v].nxt[c];
            }else {
                a[v].go[c] = v ? go(get_link(v), c) : 0;
            }
        }
        return a[v].go[c];
    }
    int get_link(int v) {
        if(a[v].link == -1) {
            if(!v || !a[v].par) a[v].link = 0;
            else a[v].link = go(get_link(a[v].par), a[v].pch);
        }
        return a[v].link;
    }
    bool is_good(int v) {
        if(!v) return false;
        if(a[v].good == -1) {
            a[v].good = is_good(get_link(v));
        }
        return a[v].good;
    }
    bool is_there_substring(const string &s) {
        int v = 0;
        for(char c : s) {
            v = go(v, c);
            if(is_good(v)) {
                return true;
            }
        }
        return false;
    }
};
```

```
vector<int> Hungarian(const vector< vector<int> >& a){ // ALARM: INT everywhere
    int n = (int)a.size();
    vector<int> row(n), col(n), pair(n, -1), back(n, -1), prev(n, -1);
    auto get = [&](int i, int j){ return a[i][j] + row[i] + col[j];};
    for(int v = 0; v < n; v++){
        vector<int> min_v(n, v), A_plus(n), B_plus(n);
        A_plus[v] = 1; int jb;
        while(true){
            int pos_i = -1, pos_j = -1;
            for(int j = 0; j < n; j++){
                if(!B_plus[j] && (pos_i == -1 ||
                    get(min_v[j], j) < get(pos_i, pos_j))) {
                    pos_i = min_v[j], pos_j = j;
                }
            }
            int weight = get(pos_i, pos_j);
            for(int i = 0; i < n; i++) if(!A_plus[i]) row[i] += weight;
            for(int j = 0; j < n; j++) if(!B_plus[j]) col[j] -= weight;
            B_plus[pos_j] = 1, prev[pos_j] = pos_i;
            int x = back[pos_j];
            if(x == -1) { jb = pos_j; break;}
            A_plus[x] = 1;
            for(int j = 0; j < n; j++)
                if(get(x, j) < get(min_v[j], j))
                    min_v[j] = x;
        }
        while(jb != -1){
            back[jb] = prev[jb];
            swap(pair[prev[jb]], jb);
        }
    }
    return pair;
}
```

```
struct HopcroftKarp {
    int n, m;
    vec< vec< int > > g;
    vec< int > pl, pr, dist;
    HopcroftKarp(): n(0), m(0) { }
    HopcroftKarp(int _n, int _m): n(_n), m(_m) { g.resize(n); }
    void add_edge(int u, int v) { g[u].push_back(v); }
    bool bfs() {
        dist.assign(n + 1, inf);
        queue< int > q;
        for(int u = 0;u < n;u++) {
            if(pl[u] < m) continue;
            dist[u] = 0;
            q.push(u);
        }
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            if(dist[u] >= dist[n]) continue;
            for(int v : g[u]) {
                if(dist[ pr[v] ] > dist[u] + 1) {
                    dist[ pr[v] ] = dist[u] + 1;
                    q.push(pr[v]);
                }
            }
        }
        return dist[n] < inf;
    }
    bool dfs(int v) {
        if(v == n) return 1;
        for(int to : g[v]) {
            if(dist[ pr[to] ] != dist[v] + 1) continue;
            if(!dfs(pr[to])) continue;
            pl[v] = to;
            pr[to] = v;
            return 1;
        }
        return 0;
    }
    int find_max_matching() {
        pl.resize(n, m);
        pr.resize(m, n);
        int result = 0;
        while(bfs()) {
            for(int u = 0;u < n;u++) {
                if(pl[u] < m) continue;
                result += dfs(u);
            }
        }
        return result;
    }
};
```

```
struct Line {
    ll k, b;
    int type;
    ld x;
    Line(): k(0), b(0), type(0), x(0) { }
    Line(ll _k, ll _b, ld _x = 1e18, int _type = 0):
        k(_k), b(_b), x(_x), type(_type) { }
    bool operator<(const Line& other) const {
        if(type + other.type > 0) { return x < other.x;
        }else { return k < other.k; }
    }
    ld intersect(const Line& other) const {
        return ld(b - other.b) / ld(other.k - k);
    }
    ll get_func(ll x0) const {
        return k * x0 + b;
    }
 };
struct CHT {
    set< Line > qs;
    set< Line > :: iterator fnd, help;
    bool hasr(const set< Line > :: iterator& it) {
        return it != qs.end() && next(it) != qs.end(); }
    bool hasl(const set< Line > :: iterator& it) {
        return it != qs.begin(); }
    bool check(const set< Line > :: iterator& it) {
        if(!hasr(it)) return true;
        if(!hasl(it)) return true;
        return it->intersect(*prev(it)) < it->intersect(*next(it)); }
    void update_intersect(const set< Line > :: iterator& it) {
        if(it == qs.end()) return;
        if(!hasr(it)) return;
        Line tmp = *it;
        tmp.x = tmp.intersect(*next(it));
        qs.insert(qs.erase(it), tmp);
    }
    void add_line(Line L) {
        if(qs.empty()) { qs.insert(L); return; }
        {   fnd = qs.lower_bound(L);
            if(fnd != qs.end() && fnd->k == L.k) {
                if(fnd->b >= L.b) return;
                else qs.erase(fnd);
            }
        }
        fnd = qs.insert(L).first;
        if(!check(fnd)) { qs.erase(fnd); return; }
        while(hasr(fnd) && !check(help = next(fnd))) { qs.erase(help); }
        while(hasl(fnd) && !check(help = prev(fnd))) { qs.erase(help); }
        if(hasl(fnd)) { update_intersect(prev(fnd)); }
        update_intersect(fnd);
    }
    ll get_max(ld x0) {
        if(qs.empty()) return -inf64;
        fnd = qs.lower_bound(Line(0, 0, x0, 1));
        if(fnd == qs.end()) fnd--;
        ll res = -inf64; int i = 0;
        while(i < 2 && fnd != qs.end()) {
            res = max(res, fnd->get_func(x0));
            fnd++;
            i++;
        }
        while(i-- > 0) fnd--;
        while(i < 2) {
            res = max(res, fnd->get_func(x0));
            if(hasl(fnd)) {
                fnd--; i++;
            }else {
                break;
            }
        }
        return res;
    }
```

```
};
```

```
const int mod = 998244353;
const int root = 31;
const int LOG = 23;
const int N = 1e5 + 5;
vec< int > G[LOG + 1];
vec< int > rev[LOG + 1];
inline void _add(int &x, int y);
inline int _sum(int a, int b);
inline int _sub(int a, int b);
inline int _mul(int a, int b);
inline int _binpow(int x, int p);
inline int _rev(int x);
void precalc() {
    for(int start = root, lvl = LOG;lvl >= 0;lvl--, start = _mul(start, start)) {
        int tot = 1 << lvl;
        G[lvl].resize(tot);
        for(int cur = 1, i = 0;i < tot;i++, cur = _mul(cur, start)) {
            G[lvl][i] = cur;
        }
    }
    for(int lvl = 1;lvl <= LOG;lvl++) {
        int tot = 1 << lvl;
        rev[lvl].resize(tot);
        for(int i = 1;i < tot;i++) {
            rev[lvl][i] = ((i & 1) << (lvl - 1)) | (rev[lvl][i >> 1] >> 1);
        }
    }
}
void fft(vec< int > &a, int sz, bool invert) {
    int n = 1 << sz;
    for(int j, i = 0;i < n;i++) {
        if((j = rev[sz][i]) < i) {
            swap(a[i], a[j]);
        }
    }
    for(int f1, f2, lvl = 0, len = 1;len < n;len <<= 1, lvl++) {
        for(int i = 0;i < n;i += (len << 1)) {
            for(int j = 0;j < len;j++) {
                f1 = a[i + j];
                f2 = _mul(a[i + j + len], G[lvl + 1][j]);
                a[i + j] = _sum(f1, f2);
                a[i + j + len] = _sub(f1, f2);
            }
        }
    }
    if(invert) {
        reverse(a.begin() + 1, a.end());
        int rn = _rev(n);
        for(int i = 0;i < n;i++) {
            a[i] = _mul(a[i], rn);
        }
    }
}
vec< int > multiply(const vec< int > &a, const vec< int > &b) {
    vec< int > fa(ALL(a));
    vec< int > fb(ALL(b));
    int n = (int)a.size();
    int m = (int)b.size();
    int maxnm = max(n, m), sz = 0;
    while((1 << sz) < maxnm) sz++; sz++;
    fa.resize(1 << sz);
    fb.resize(1 << sz);
    fft(fa, sz, false);
    fft(fb, sz, false);
    int SZ = 1 << sz;
    for(int i = 0;i < SZ;i++) { fa[i] = _mul(fa[i], fb[i]); }
    fft(fa, sz, true);
    while((int)fa.size() > 1 && !fa.back()) fa.pop_back();
    return fa;
}
```

```
struct Complex {
    ld rl = 0, im = 0;
    Complex() = default;
    Complex(ld x, ld y = 0): rl(x), im(y) { }
    Complex & operator = (const Complex &o) {
        if(this != &o) {
            rl = o.rl;
            im = o.im;
        }
        return *this;
    }
    Complex operator + (const Complex &o) const { return {rl + o.rl, im + o.im}; }
    Complex operator - (const Complex &o) const { return {rl - o.rl, im - o.im}; }
    Complex operator * (const Complex &o) const {
        return {rl * o.rl - im * o.im, rl * o.im + im * o.rl}; }
    Complex & operator *= (const Complex &o) {
        (*this) = (*this) * o;
        return *this;
    }
    Complex operator / (const Complex &o) const {
        ld md = o.rl * o.rl + o.im * o.im;
        return {(rl * o.rl + im * o.im) / md, (im * o.rl - rl * o.im) / md};
    }
    Complex & operator /= (int k) {
        rl /= k;
        im /= k;
        return *this;
    }
    ld real() const { return rl; }
    ld imag() const { return im; }
};
typedef Complex base;
const int LOG = 20;
const int N = 1 << LOG;
int rev[N];
vec< base > PW[LOG + 1];
void precalc() {
    for(int i = 1;i < N;i++) {
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (LOG - 1)); }
    for(int lvl = 0;lvl <= LOG;lvl++) {
        int sz = 1 << lvl;
        ld alpha = 2 * pi / sz;
        base root(cos(alpha), sin(alpha));
        base cur = 1;
        PW[lvl].resize(sz);
        for(int j = 0;j < sz;j++) {
            PW[lvl][j] = cur;
            cur *= root;
        }
    }
}
void fft(base *a, bool invert = 0) {
    for(int j, i = 0;i < N;i++) {
        if((j = rev[i]) > i) swap(a[i], a[j]);
    }
    base u, v;
    for(int lvl = 0;lvl < LOG;lvl++) {
        int len = 1 << lvl;
        for(int i = 0;i < N;i += (len << 1)) {
            for(int j = 0;j < len;j++) {
                u = a[i + j];
                v = a[i + j + len] *
                    (invert?PW[lvl+1][j?(len << 1)-j:0]:PW[lvl+1][j]);
                a[i + j] = u + v;
                a[i + j + len] = u - v;
            }
        }
    }
    if(invert) {
        for(int i = 0;i < N;i++) {
            a[i] /= N;
        }
    }
}
```

```
int fact[N];
int rfact[N];

void precalc2() {
    fact[0] = 1;
    for (int i = 1; i < N; i++) {
        fact[i] = _mul(fact[i - 1], i);
    }
    rfact[N - 1] = _rev(fact[N - 1]);
    for (int i = N - 2; i >= 0; i--) {
        rfact[i] = _mul(rfact[i + 1], i + 1);
    }
}

int getMulOnSegment(int l, int r) {
    assert(l <= r);
    if (l == 0 && r == 0) return 1;
    if (r <= 0) {
        int res = getMulOnSegment(-r, -l);
        int cnt = r - l + 1;
        if (cnt % 2) {
            res = (-res % mod + mod) % mod;
        }
        return res;
    }
    if (l < 0) {
        int resl = getMulOnSegment(0, -l);
        if (l % 2) {
            resl = (-resl % mod + mod) % mod;
        }
        int resr = getMulOnSegment(0, r);
        return _mul(resl, resr);
    }
    assert(l >= 0);
    int res = fact[r];
    if (l > 0) {
        res = _mul(res, rfact[l - 1]);
    }
    return res;
}

vector<int> extrapolate(vector<int> y, int m) {
    vector<int> yy = y;
    int n = (int)y.size() - 1;
    for (int i = 0; i <= n; i++) {
        yy[i] = _mul(y[i], _rev(getMulOnSegment(i - n, i - 0)));
    }
    vector<int> ff(n + m + 1);
    for (int i = 1; i <= n + m; i++) {
        ff[i] = _mul(fact[i - 1], rfact[i]);
    }
    vector<int> ss = multiply(yy, ff);
    for (int i = 1; i <= m; i++) {
        int cc = getMulOnSegment(i, n + i);
        int Si = ss[n + i];
        y.push_back(_mul(cc, Si));
    }
    return y;
}
```

```
struct pal_tree {
    int sz;
    vector<unordered_map<int,int>> g;
    vector<int> len, suf, cn, base;
    vector<int> fin;
    int cur, v;
    pal_tree(): g(2),sz(2),len({-1, 0}),cn({0, 0}),suf({0, 0}),cur(0),v(1){}
    void add_string (const string &s) {
        for (int i = 0; i < s.length(); i++) {
            char c = s[i] - 'a';
            while (i - len[v] - 1 < 0 || s[i - len[v] - 1] - 'a' != c)
                v = suf[v];
            if (!g[v].count(c)) {
                g.emplace_back();
                int t = len[v];
                len.push_back(t + 2);
                int u = v;
                do {
                    u = suf[u];
                } while (u && s[i - len[u] - 1] - 'a' != c);
                suf.push_back(!g[u].count(c) ? 1 : g[u][c]);
                t = cn[suf.back()];
                cn.push_back(t + 1);
                g[v][c] = sz++;
                cur++;
            }
            v = g[v][c];
            fin.push_back(cn[v]);
        }
        return;
    }
};
```

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
typedef
    tree<
        pair< int, int >,
        null_type,
        less< pair< int, int > >,
        rb_tree_tag,
        tree_order_statistics_node_update
    > stat_set;
// ...
ordered_set X;
```

```
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);
    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true
    cout<<X.order_of_key(-5)<<endl;  // 0
    cout<<X.order_of_key(1)<<endl;   // 0
    cout<<X.order_of_key(3)<<endl;   // 2
    cout<<X.order_of_key(4)<<endl;   // 2
    cout<<X.order_of_key(400)<<endl; // 5
// ...
// pbds
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
using namespace __gnu_pbds;
gp_hash_table<int, int> table; // the usage is the same as the unordered_map
```

```
struct Dinic {
    struct Edge { int fr, to, cp, id, fl; };
    int n, S, T;
    vec< Edge > es;
    vec< vec< int > > g;
    vec< int > dist, res, ptr;
    Dinic(int _n, int _S, int _T):
        n(_n), S(_S), T(_T) { g.resize(n); }
    void add_edge(int fr, int to, int cp, int id) {
        g[fr].push_back((int)es.size());
        es.push_back({fr, to, cp, id, 0});
        g[to].push_back((int)es.size());
        es.push_back({to, fr, 0, -1, 0});
    }
    bool bfs(int K) {
        dist.assign(n, inf);
        dist[S] = 0;
        queue< int > q;
        q.push(S);
        while(!q.empty()) {
            int v = q.front();
            q.pop();
            for(int ps : g[v]) {
                Edge &e = es[ps];
                if(e.fl + K > e.cp) continue;
                if(dist[e.to] > dist[e.fr] + 1) {
                    dist[e.to] = dist[e.fr] + 1;
                    q.push(e.to);
                }
            }
        }
        return dist[T] < inf;
    }
    int dfs(int v, int _push = INT_MAX) {
        if(v == T || !_push) return _push;
        for(int &iter = ptr[v];iter < (int)g[v].size();iter++) {
            int ps = g[v][ ptr[v] ];
            Edge &e = es[ps];
            if(dist[e.to] != dist[e.fr] + 1) continue;
            int tmp = dfs(e.to, min(_push, e.cp - e.fl));
            if(tmp) {
                e.fl += tmp;
                es[ps ^ 1].fl -= tmp;
                return tmp;
            }
        }
        return 0;
    }
    ll find_max_flow() {
        ptr.resize(n);
        ll max_flow = 0, add_flow;
        for(int K = 1 << 30;K > 0;K >>= 1) {
            while(bfs(K)) {
                ptr.assign(n, 0);
                while((add_flow = dfs(S))) {
                    max_flow += add_flow;
                }
            }
        }
        return max_flow;
    }
    void assign_result() {
        res.resize(es.size());
        for(Edge e : es) if(e.id != -1) res[e.id] = e.fl; }
    int get_flow(int id) { return res[id]; }
    bool go(int v, vec< int > &F, vec< int > &path) {
        if(v == T) return 1;
        for(int ps : g[v]) {
            if(F[ps] <= 0) continue;
            if(go(es[ps].to, F, path)) {
                path.push_back(ps);
                return 1;
            }
        }
        return 0;
    }
    vec< pair< int, vec< int > > > decomposition() {
        find_max_flow();
        vec< int > F((int)es.size()), path, add;
        vec< pair< int, vec< int > > > dcmp;
        for(int i = 0;i < (int)es.size();i++) F[i] = es[i].fl;
        while(go(S, F, path)) {
            int mn = INT_MAX;
            for(int ps : path) mn = min(mn, F[ps]);
            for(int ps : path) F[ps] -= mn;
            for(int ps : path) add.push_back(es[ps].id);
            reverse(ALL(add));
            dcmp.push_back({mn, add});
            add.clear();
            path.clear();
        }
        return dcmp;
    }
};
```

```
struct MCMF {
    struct Edge { int fr, to, cp, fl, cs, id; };
    int n, S, T;
    vec< Edge > es;
    vec< vec< int > > g;
    vec< ll > dist, phi;
    vec< int > from;
    MCMF(int _n, int _S, int _T): n(_n), S(_S), T(_T)
    { g.resize(n); }
    void add_edge(int fr, int to, int cp, int cs, int id) {
        g[fr].push_back((int)es.size());
        es.push_back({fr, to, cp, 0, cs, id});
        g[to].push_back((int)es.size());
        es.push_back({to, fr, 0, 0, -cs, -1});
    }
    void init_phi() {
        dist.assign(n, LLONG_MAX);
        dist[S] = 0;
        for(int any, iter = 0;iter < n - 1;iter++) { // Ford Bellman
            any = 0;
            for(Edge e : es) {
                if(e.fl == e.cp) continue;
                if(dist[e.to] - dist[e.fr] > e.cs) {
                    dist[e.to] = dist[e.fr] + e.cs;
                    any = 1;
                }
            }
            if(!any) break;
        }
        phi = dist;
    }
    bool Dijkstra() {
        dist.assign(n, LLONG_MAX);
        from.assign(n, -1);
        dist[S] = 0;
        priority_queue< pair< ll, int >, vec< pair< ll, int > >,
            greater< pair< ll, int > > > pq;
        pq.push({dist[S], S});
        while(!pq.empty()) {
            int v;
            ll di;
            tie(di, v) = pq.top();
            pq.pop();
            if(di != dist[v]) continue;
            for(int ps : g[v]) {
                Edge &e = es[ps];
                if(e.fl == e.cp) continue;
                if(dist[e.to] - dist[e.fr] > e.cs + phi[e.fr] - phi[e.to]) {
                    dist[e.to] = dist[e.fr] + e.cs + phi[e.fr] - phi[e.to];
                    from[e.to] = ps;
                    pq.push({dist[e.to], e.to});
                }
            }
        }
        for(int v = 0;v < n;v++) {
            phi[v] += dist[v];
        }
        return dist[T] < LLONG_MAX;
    }
    pll find_mcmf() {
        init_phi();
        ll flow = 0, cost = 0;
        while(Dijkstra()) {
            int mn = INT_MAX;
            for(int v = T;v != S;v = es[ from[v] ].fr) {
                mn = min(mn, es[from[v]].cp - es[from[v]].fl);
            }
            flow += mn;
            for(int v = T;v != S;v = es[ from[v] ].fr) {
                es[ from[v] ].fl += mn;
                es[ from[v] ^ 1 ].fl -= mn;
            }
        }
        for(Edge &e : es) {
            if(e.fl >= 0)
                cost += 1ll * e.fl * e.cs;
        }
        return make_pair(flow, cost);
    }
    bool go(int v, vec< int > &F, vec< int > &path, vec< int > &used) {
        if(used[v]) return 0;
        used[v] = 1;
        if(v == T) return 1;
        for(int ps : g[v]) {
            if(F[ps] <= 0) continue;
            if(go(es[ps].to, F, path, used)) {
                path.push_back(ps);
                return 1;
            }
        }
        return 0;
    }
    vec< pair< int, vec< int > > > decomposition(ll &_flow, ll &_cost) {
        tie(_flow, _cost) = find_mcmf();
        vec< int > F((int)es.size()), path, add, used(n);
        vec< pair< int, vec< int > > > dcmp;
        for(int i = 0;i < (int)es.size();i++) F[i] = es[i].fl;
        while(go(S, F, path, used)) {
            used.assign(n, 0);
            int mn = INT_MAX;
            for(int ps : path) mn = min(mn, F[ps]);
            for(int ps : path) F[ps] -= mn;
            for(int ps : path) add.push_back(es[ps].id);
            reverse(ALL(add));
            dcmp.push_back({mn, add});
            add.clear();
            path.clear();
        }
        return dcmp;
    }
};
```

```
namespace FACTORIZE {
    const ll MAXX = 1000;
    const int FERMA_ITER = 30;
    // const int POLLARD_PO_ITER = 10000;
    int POLLARD_PO_ITER;
    inline ll sqr(ll n) { return n * n; }
    ll check_small(ll n) {
        for(ll x = 1;sqr(x) <= n && x <= MAXX;x++) {
            if(x > 1 && n % x == 0) {
                return x;
            }else if(sqr(x + 1) > n) {
                return -1;
            }
        }
        return -1;
    }
    ll check_square(ll n) {
        ll bl = 0;
        ll br = 3e9+1;
        ll bm;
        while(br - bl > 1) {
            bm = (bl + br) / 2;
            if(sqr(bm) <= n) {
                bl = bm;
            }else {
                br = bm;
            }
        }
        if(sqr(bl) == n && bl > 1) {
            return bl;
        }else {
            return -1;
        }
    }
    inline ll _mul(ll a, ll b, ll m) {
        static __int128 xa = 1;
        static __int128 xb = 1;
        static __int128 xm = 1;
        xa = a;
        xb = b;
        xm = m;
        return ll(xa * xb % xm);
    }
    /*
    ll _mul(ll x, ll y, ll mod) {
        ll q = ld(x) * ld(y) / ld(mod);
        ll r = x * y - q * mod;
        return (r % mod + mod) % mod;
    }*/
    inline ll _binpow(ll x, ll p, ll m) {
        static ll res = 1;
        static ll tmp = 1;
        res = 1;
        tmp = x;
        while(p > 0) {
            if(p & 1ll) {
                res = _mul(res, tmp, m);
            }
            tmp = _mul(tmp, tmp, m);
            p >>= 1;
        }
        return res;
    }
    mt19937_64 next_rand(179);
    ll gcd(ll x, ll y) { return !x ? y : gcd(y % x, x); }
    bool is_prime(ll n) {
        if(n <= 1) return false;
        if(n == 2) return true;
        ll a, g;
        for(int iter = 0;iter < FERMA_ITER;iter++) {
            a = next_rand() % (n - 2);
            if(a < 0) a += n - 2;
            a += 2;
            assert(1 < a && a < n);
            g = gcd(a, n);
            if(g != 1) { return false; }
            if(_binpow(a, n - 1, n) != 1) { return false; }
        }
        return true;
    }
    inline ll _func(ll x, ll n) {
        static ll result = 1;
        result = _mul(x, x, n);
        return result + 1 < n ? result + 1 : 0;
    }
    ll pollard_po(ll n) {
        POLLARD_PO_ITER = 5 + 3 * pow(n, 0.25);
        ll a, b, x, g;
        while(1) {
            a = next_rand() % n;
            if(a < 0) a += n;
            b = next_rand() % n;
            if(b < 0) b += n;
            for(int iter = 0;iter < POLLARD_PO_ITER;iter++) {
                x = a >= b ? a - b : b - a;
                g = gcd(x, n);
                if(1 < g && g < n) {
                    return g;
                }
                a = _func(a, n);
                b = _func(_func(b, n), n);
            }
        }
    }
    ll get_div(ll n) {
        ll res;
        res = check_small(n);
        if(res != -1) { return res; }
        res = check_square(n);
        if(res != -1) { return res; }
        if(is_prime(n)) { return n; }
        return pollard_po(n);
    }
}
```

```
class EulerTourTrees {
/*graph - forest 1 .. n get = is connected?
no memory leaks 1 <= n, q <= 10^5 0.7 sec*/
private:
    struct Node {
        Node *l; Node *r; Node *p;
        int prior; int cnt; int rev;
    };
    void do_rev(Node *v) {if(v) v->rev ^= 1, swap(v->l, v->r);}
    int get_cnt(Node *v) const {return v ? v->cnt : 0;}
    void update(Node *v) {if(!v) return;
        v->cnt = 1 + get_cnt(v->l) + get_cnt(v->r);
        v->p = nullptr;if(v->l) v->l->p = v;if(v->r) v->r->p = v;}
    void push(Node *v) {
        if(!v) return; if(v->rev) {do_rev(v->l);do_rev(v->r);v->rev ^= 1;}}
    void merge(Node *& v, Node *l, Node *r) {
        if(!l || !r) {v = l ? l : r;return;}
        push(l);push(r);
        if(l->prior < r->prior) {merge(l->r, l->r, r);v = l;}else {
            merge(r->l, l, r->l);v = r;}update(v);}
    void split_by_cnt(Node *v, Node *& l, Node *& r, int x) {
        if(!v) {l = r = nullptr;return;}push(v);
        if(get_cnt(v->l) + 1 <= x) {
            split_by_cnt(v->r, v->r, r, x - get_cnt(v->l) - 1);l = v;
        }else {split_by_cnt(v->l, l, v->l, x);r = v;}
        update(l);update(r);
    }
    void push_path(Node *v) {
        if(!v) return;push_path(v->p);push(v);}
    int get_pos(Node *v) {
        push_path(v);int res = 0, ok = 1;
        while(v) {if(ok) res += get_cnt(v->l) + 1;
            ok = v->p && v->p->r == v;v = v->p;}
        return res;}
    Node *get_root(Node *v) const{while(v && v->p) v = v->p;return v;}
    Node *shift(Node *v) {
        if(!v) return v;int pos = get_pos(v);
        Node *nl = nullptr, *nr = nullptr;Node *root = get_root(v);
        split_by_cnt(root, nl, nr, pos - 1);do_rev(nl);do_rev(nr);
        merge(root, nl, nr);do_rev(root);return root;}
public:EulerTourTrees() = default;
    EulerTourTrees(int _n): n(_n) {ptr.resize(_n + 1);
        where_edge.resize(_n + 1);}
    bool get(int u, int v) const {if(u == v) return true;
        Node *ru = get_root(ptr[u].empty() ? nullptr : *ptr[u].begin());
        Node *rv = get_root(ptr[v].empty() ? nullptr : *ptr[v].begin());
        return ru && ru == rv;}
    void link(int u, int v) {
        Node *ru = shift(ptr[u].empty() ? nullptr : *ptr[u].begin());
        Node *rv = shift(ptr[v].empty() ? nullptr : *ptr[v].begin());
        Node *uv = new Node();Node *vu = new Node();
        ptr[u].insert(uv);ptr[v].insert(vu);
        where_edge[u][v] = uv;where_edge[v][u] = vu;
        merge(ru, ru, uv);merge(ru, ru, rv);merge(ru, ru, vu);}
    void cut(int u, int v) {
        Node *uv = where_edge[u][v];Node *vu = where_edge[v][u];
        ptr[u].erase(uv);ptr[v].erase(vu);
        Node *root = shift(uv);Node *nl = nullptr, *nm = nullptr, *nr = nullptr;
        int pos1 = get_pos(uv);int pos2 = get_pos(vu);
        if(pos1 < pos2) {split_by_cnt(root, nl, nr, pos2);
            split_by_cnt(nl, nl, vu, pos2 - 1);split_by_cnt(nl, nl, nm, pos1);
            split_by_cnt(nl, nl, uv, pos1 - 1);merge(nl, nl, nr);}else {
            split_by_cnt(root, nl, nr, pos1);split_by_cnt(nl, nl, uv, pos1 - 1);
            split_by_cnt(nl, nl, nm, pos2);split_by_cnt(nl, nl, vu, pos2 - 1);
            merge(nl, nl, nm);}delete uv;delete vu;}
```

```
private:int n = 0;vec< set< Node* > > ptr;
    vec< unordered_map< int, Node* > > where_edge; // ptr to node
};
```

```
struct Edge {
    int fr, to, w, id;
    bool operator < (const Edge& o) const { return w < o.w; }
};

// find oriented mst (tree)
// there are no edge --> root (root is 0)
// 0 .. n - 1, weights and vertices will be changed, but ids are ok
vector<Edge> work(const vector<vector<Edge>>& graph) {
    int n = (int) graph.size();
    vector<int> color(n), used(n, -1);
    for (int i = 0; i < n; i++)
        color[i] = i;
    vector<Edge> e(n);
    for (int i = 0; i < n; i++) {
        if (graph[i].empty()) {
            e[i] = {-1, -1, -1, -1};
        } else {
            e[i] = *min_element(graph[i].begin(), graph[i].end());
        }
    }
    vector<vector<int>> cycles;
    used[0] = -2;
    for (int s = 0; s < n; s++) {
        if (used[s] != -1)
            continue;
        int x = s;
        while (used[x] == -1) {
            used[x] = s;
            x = e[x].fr;
        }
        if (used[x] != s)
            continue;
        vector<int> cycle = {x};
        for (int y = e[x].fr; y != x; y = e[y].fr)
            cycle.push_back(y), color[y] = x;
        cycles.push_back(cycle);
    }
    if (cycles.empty())
        return e;
    vector<vector<Edge>> next_graph(n);
    for (int s = 0; s < n; s++) {
        for (const Edge& edge : graph[s]) {
            if (color[edge.fr] != color[s])
                next_graph[color[s]].push_back({
                    color[edge.fr], color[s], edge.w - e[s].w, edge.id
                });
        }
    }
    vector<Edge> tree = work(next_graph);
    for (const auto& cycle : cycles) {
        int cl = color[cycle[0]];
        Edge next_out = tree[cl], out{};
        int from = -1;
        for (int v : cycle) {
            tree[v] = e[v];
            for (const Edge& edge : graph[v])
                if (edge.id == next_out.id)
                    from = v, out = edge;
        }
        tree[from] = out;
    }
    return tree;
}
```

Gomory-Hu tree (Gusfield's algorithm): label nodes from 0 to $(|V|-1)$ and set $p_i = 0 \forall i > 0$. $\forall\, i > 0$: find min-cut (S, T) between i and p_i, where $i \in S, p_i \in T$; for each node j, s.t. $i < j, j \in S, p_j = p_i$ set $p_j = i$