Team Note of kumayuC

kuming19, mathjuny, yuntaek

Compiled on November 20, 2025

Contents

1 Setup

1.1 Linux Commands

```
$ cd Desktop
$ touch main.cpp
$ mkdir A B C D ... M
$ alias run='g++ main.cpp -o main && ./main'
$ alias debug='g++ main.cpp -fsanitize=undefined -o main &&
./main'
```

1.2 Template Code

```cpp
#include <bits/stdc++.h>
#pragma warning(disable:4996)
#pragma comment(linker, "/STACK:336777216")
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx,avx2,fma")
using namespace std;
using ll = long long;
using pll = pair<ll,ll>;
using ld = long double;
using pld = pair<ld,ld>;
using vl = vector<ll>;

template<typename T>
ostream& operator<<(ostream& out, vector<T> v) {
    string _;
    out << '(';
    for (T x : v) out << _ << x, _ = " ";
    out << ')';
    return out;
}

void pre() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
}
void pre2 () {
    freopen("input.txt","r",stdin);
    freopen("output.txt","w",stdout);
}

void solve() {
    ll i, j, k;
}

int main() {
    pre();
    ll t = 1;
    //cin >> t;
    while(t--) solve();
    return 0;
}
```

2 STL, PBDS

2.1 Bitset

```
bitset<8> a; // 00000000
bitset<8> b("10101010"); // 10101010
bitset<8> c(42); // base 10, 00101010

b[2] = 1; // 0 index, least bit is index 0
b.set(); // set every bit to 1
b.set(1); // b[1] = 1
b.reset(); // set every bit to 0
b.reset(2); // b[2] = 0
b.flip(); // flip every bit
b.flip(0); // flip b[0]
b.count(); // # of turned on(i.e. 1) bits
b.all(); // true if all bits are 1
b.any(); // true if least one bit is 1
b.none(); // true if all bits are 0

ull v = b.to_ullong();
string s = b.to_string();

bitset<8> d = b & c; // bit operator applicable
```

2.2 Tuple, Tuplehash

```
tuple<ll, string> t = make_tuple(1, "asdf");
get<0>(t); // t[0]
int a; string b; tie(a, b) = t; // a = 1, b = "asdf"
tuple_cat(t, make_tuple(3)); // {1, "asdf", 3}

struct TupleHash {
    size_t operator()(const tuple<ll,ll,ll>& t) const noexcept
    {
        auto [a,b,c] = t;
        size_t h1 = hash<ll>()(a);
        size_t h2 = hash<ll>()(b);
        size_t h3 = hash<ll>()(c);
        return ((h1 ^ (h2 << 1)) >> 1) ^ (h3 << 1);
    }
}; //unordered_set<tuple<ll, ll, ll>, TupleHash> memo;
struct PairHash {
    size_t operator()(const pair<ll, ll>& p) const noexcept {
        size_t h1 = hash<ll>()(p.first);
        size_t h2 = hash<ll>()(p.second);
        return h1 ^ (h2 + 0x9e3779b97f4a7c15ULL + (h1<<6) +
        (h1>>2));
    }
}; //unordered_set<pair<ll, ll>, PairHash> memo;
```

2.3 Rope

```
#include<ext/rope>
using namespace __gnu_cxx;

main(){
    string inp;
    cin >> inp;
    crope r(inp.c_str());  // rope<char> r("") also works

    r.insert(pos, str); // str -> c_string || crope, O(logN)
    // r = "abc", pos = 1, str = "de" -> r = "adebc"
    r.erase(pos, len);  // O(logN)
    r += r2; // = r.append(r2) = r.concat(r2), O(logN +
    len(r2))
    crope r2 = r.substr(pos, len); // O(logN + len)
    r[idx]; // = r.at(idx), O(logN)
    size(r); // = length(r)
    cout << r;
}
```

2.4 unordered_map

```
struct Point {
    ll x, y;
    ll idx;
    bool operator==(const Point &r) const {
        return r.x==x&&r.y==y;
    }
};

struct PointHash {
    size_t operator()(const Point& p) const {
        // 해시 조합 (간단 버전)
        return std::hash<ll>()(p.x) ^ (std::hash<ll>()(p.y) <<
        1);
    }
};
unordered_map<Point,ll,PointHash> M1;  //== operator 와 hash
함수 필요
```

2.5 ordered_set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

ordered_set<int> os;
os.insert(10);
os.insert(20);
os.insert(30);

*os.find_by_order(1); // 20, kth smallest element, 0-indexed

int count = os.order_of_key(25); // 2, * of elements strictly
less than k

os.erase(10); // erase element

template<typename T>
using ordered_multiset = tree<T, null_type, less_equal<T>,
rb_tree_tag, tree_order_statistics_node_update>;

void erase_one(ordered_multiset &ms, int val) {
    int idx = ms.order_of_key(val);
    auto it = ms.find_by_order(idx);
    if (it != ms.end() && *it == val) ms.erase(it);
}

ordered_multiset::iterator find_one(ordered_multiset &ms, int
val) {
    int idx = ms.order_of_key(val);
    auto it = ms.find_by_order(idx);
    if (it != ms.end() && *it == val) return it;
    return ms.end();
}
```

2.6 priority_queue(PBDS)

```
#include <ext/pb_ds/priority_queue.hpp>

typedef __gnu_pbds::priority_queue<ll, less<ll>,
pairing_heap_tag> pq_ll;
pq_ll pq;

auto it = pq.push(10); // it : note iterator
// save node iterator while push if needed
int t = pq.top(); // gets maximum element
pq.pop(); // removes maximum element
pq.modify(it, 5); // *it.set(5)
pq.erase(it); // remove iterator
pq_ll pq2;
pq2.push(100);
pq.join(pq2); // pq = (10, 100), pq2 = ()
pq_ll pq3;
```

2.7 gp_hash_table

```
gp_hash_table<ll, ll> table;
// key type must be in {ll, ld, char, string, pair, tuple}
table[42] = 13;
table[100] = 99;

// Access
if (table.find(42) != table.end()) {
    int x = table[42];
}

// Erase
table.erase(100);
```

2.8 trie_set

```
typedef trie<std::string, null_type,
trie_string_access_traits<>, pat_trie_tag,
trie_prefix_search_node_update> trie_set;

trie_set t;
t.insert("apple");
t.insert("app");
t.insert("banana");

// Search for exact string
if (t.find("app") != t.end()) {
    // found "app"
}

auto range = t.prefix_range("ba"); // find all string with
prefix "ba"
for (auto it = range.first; it != range.second; it++)
    std::cout << *it << "\n"; // banana

*t.lower_bound("app"); // app
*t.upper_bound("app"); // apple

t.erase("apple");
```

3 Data Structure

3.1 Segment Tree

```
struct SegmentTree {
    // zero-index
    int n;
    vector<ll> tree, data;

    explicit SegmentTree(int size) {
        n = size;
        tree.assign(n * 4, 0);
    }

    void init_(int node, int l, int r) {
        if (l == r) {
            tree[node] = data[l];
            return;
        }
        int mid = (l + r) >> 1;
        init_(node << 1, l, mid);
        init_((node << 1) | 1, mid + 1, r);
        tree[node] = tree[node << 1] + tree[(node << 1) | 1];
    }

    void init(const vector<ll>& arr) {
        data = arr;
        init_(1, 0, n - 1);
    }

    void update_(int node, int l, int r, ll idx, ll val) {
        if (l == r) {
            tree[node] += val;
            return;
        }
        int mid = (l + r) >> 1;
        if (mid >= idx) {
            update_(node << 1, l, mid, idx, val);
        }
        else {
            update_((node << 1) | 1, mid + 1, r, idx, val);
        }
        tree[node] = tree[node << 1] + tree[(node << 1) | 1];
    }

    void update(ll idx, ll val) {
        update_(1, 0, n - 1, idx, val);
    }

    ll query_(int node, int l, int r, int s, int e) {
        if (e < l || r < s) return 0;
        if (s <= l && r <= e) return tree[node];
        int mid = (l + r) >> 1;
        return query_(node << 1, l, mid, s, e) + query_((node
        << 1) | 1, mid + 1, r, s, e);
    }

    ll query(ll s, ll e) {
        return query_(1, 0, n - 1, s, e);
    }
};
```

3.2 Lazy Segment Tree

```
struct LazySegmentTree {
    // zero-index
    int n;
    vector<ll> tree, lazy, data;

    explicit LazySegmentTree(int size) {
        n = size;
        tree.assign(n * 4, 0);
        lazy.assign(n * 4, 0);
    }

    void init_(int node, int l, int r) {
        if (l == r) {
            tree[node] = data[l];
            return;
        }
        int mid = (l + r) >> 1;
        init_(node << 1, l, mid);
        init_((node << 1) | 1, mid + 1, r);
        tree[node] = tree[node << 1] + tree[(node << 1) | 1];
    }

    void init(const vector<ll>& arr) {
        data = arr;
        init_(1, 0, n - 1);
    }

    void push(int node, int l, int r) {
        if (lazy[node] != 0) {
            tree[node] += (r - l + 1) * lazy[node];
            if (l != r) {
                lazy[node << 1] += lazy[node];
                lazy[(node << 1) | 1] += lazy[node];
            }
            lazy[node] = 0;
        }
    }

    void update_(int node, int l, int r, int s, int e, ll val)
    {
        push(node, l, r);
        if (r < s || l > e) return;
        if (s <= l && r <= e) {
            lazy[node] += val;
            push(node, l, r);
            return;
        }
        int mid = (l + r) >> 1;
        update_(node << 1, l, mid, s, e, val);
        update_((node << 1) | 1, mid + 1, r, s, e, val);
        tree[node] = tree[node << 1] + tree[(node << 1) | 1];
    }

    void update(int s, int e, ll val) {
        update_(1, 0, n - 1, s, e, val);
    }

    ll query_(int node, int l, int r, int s, int e) {
        push(node, l, r);
        if (r < s || l > e) return 0;
        if (s <= l && r <= e) return tree[node];
        int mid = (l + r) >> 1;
        return query_(node << 1, l, mid, s, e) + query_((node
        << 1) | 1, mid + 1, r, s, e);
    }

    ll query(int s, int e) {
        return query_(1, 0, n - 1, s, e);
    }
};
```

3.3 Fenwick Tree

```
struct Fenwick {
```

```
    int n;
    vector<long long> bit;

    Fenwick(int n = 0) {init(n);}

    void init(int n_) {
        n = n_;
        bit.assign(n + 1, 0);
    }

    void update(int idx, long long val) { //업데이트
        for (; idx <= n; idx += idx & -idx)
            bit[idx] += val;
    }

    long long sum(int idx) const {
        long long s = 0;
        for (; idx > 0; idx -= idx & -idx)
            s += bit[idx];
        return s;
    }

    long long range_sum(int l, int r) const { //구간합 쿼리
        return sum(r) - sum(l - 1);
    }
};
```

3.4 2D Fenwick Tree

```
struct Fenwick2D {
    int n, m;
    vector<vector<long long>> bit;

    Fenwick2D(int n = 0, int m = 0) {
        init(n, m);
    }

    void init(int n_, int m_) {
        n = n_;
        m = m_;
        bit.assign(n + 1, vector<long long>(m + 1, 0));
    }

    void update(int x, int y, long long val) {
        for (int i = x; i <= n; i += i & -i)
            for (int j = y; j <= m; j += j & -j)
                bit[i][j] += val;
    }

    long long sum(int x, int y) const {
        long long s = 0;
        for (int i = x; i > 0; i -= i & -i)
            for (int j = y; j > 0; j -= j & -j)
                s += bit[i][j];
        return s;
    }

    long long range_sum(int x1, int y1, int x2, int y2) const
    {
        return sum(x2, y2)
             - sum(x1 - 1, y2)
             - sum(x2, y1 - 1)
             + sum(x1 - 1, y1 - 1);
    }
};
```

3.5 Trie

```
class Node {
public:
    Node* next[26];
    bool isEnd;
    Node() : isEnd(false) {
        for (int i = 0; i < 26; i++) {
            next[i] = nullptr;
        }
    }
};
class Trie {
public:
    Node* root;
    Trie() {
        root = new Node();
    }
    ~Trie() {
        freeNode(root);
    }
    void insert(const string& s) { // 삽입
        Node* cur = root;
        for (char c : s) {
            int idx = c - 'a';
            if (cur->next[idx] == nullptr) {
                cur->next[idx] = new Node();
            }
            cur = cur->next[idx];
        }
        cur->isEnd = true;
    }
    bool search(const string& s) const { // 완전 일치 검색
        Node* cur = root;
        for (char c : s) {
            int idx = c - 'a';
            if (cur->next[idx] == nullptr) return false;
            cur = cur->next[idx];
        }
        return cur->isEnd;
    }
    bool startsWith(const string& prefix) const { //prefix
    존재 검사
        Node* cur = root;
        for (char c : prefix) {
            int idx = c - 'a';
            if (cur->next[idx] == nullptr) return false;
            cur = cur->next[idx];
        }
        return true;
    }
private:
    void freeNode(Node* node) {
        if (node == nullptr) return;
        for (int i = 0; i < 26; i++) {
            freeNode(node->next[i]);
        }
        delete node;
    }
};
```

3.6 Persistent Segment Tree

```
class PST {
    PST *l,*r;
    ll val;
public:
    PST() {
        l = nullptr;
        r = nullptr;
        val=0;
    }
    ~PST() {
        delete(l);
        delete(r);
    }
    void MakeNode() {
        l = new PST();
        r = new PST();
    }
    ll GetVal() {return val;}
    ll SetVal(ll t) {val = t;}
    PST* SetLNode(PST* t) {l=t;};
    PST* SetRNode(PST* t) {r=t;};
    PST* NextLNode() {return l;};
    PST* NextRNode() {return r;};
};

int main() {
    pre2();
    ll N;
    cin>>N;
    vl fruit(N+1);
    vector<vl> P(N+1);
    for (int i=1;i<=N;i++)cin>>fruit[i];
    for (int i=1;i<N;i++) {
        ll s,e;
        cin>>s>>e;
```

```
        P[s].push_back(e);
        P[e].push_back(s);
    }

    ll two=1;
    while (two<N)two*=2;

    vector<PST*> root(N+1,nullptr);
    root[1] = new PST();
    auto Build = [&](PST* node,ll s,ll e) {
        if (s==e)return;
        node->MakeNode();
        ll m = (s+e)/2;
        Build(node->NextLNode(),s,m);
        Build(node->NextRNode(),m+1,e);
    };

    vl check(N+1);

    auto Update = [&](auto self,PST* Node,ll s,ll e,ll
    tar)->PST* {
        if (e<tar||tar<s)return Node;
        if (s==e&&e==tar) {
            PST* newnode =  new PST();
            newnode->SetVal(Node->GetVal()+1);
            return newnode;
        }

        ll m = (s+e)/2;
        Node->SetLNode(self(self,Node->NextLNode(),s,m,tar));

        Node->SetRNode(self(self,Node->NextRNode(),m+1,e,tar));
    };

    auto Query = [&](auto self,ll g,ll p) {
        check[g] = 1;
        root[g] = Update(Update,root[p],1,two,fruit[g]);
        for (auto i : P[g]) {
            if (check[i]==1)continue;
            self(self,i,g);
        }
    };
}
```

3.7 Splay Tree

```
struct Data {
    ll size;
    ll value;
    ll sum;
    ll min;
    ll max;
    ll lazy;
    bool flip;
};

struct Node {
    Node *l, *r, *p;
    Data x;
};
Node* tree;
vector<Node*> ptrs;


void push(Node* n) { // lazy propagation
    if (n == nullptr || !n->x.flip) return;
    swap(n->l, n->r);
    if (n->l) n->l->x.flip ^= n->x.flip;
    if (n->r) n->r->x.flip ^= n->x.flip;
    n->x.flip = false;
}

void update(Node* n) {
    if (n == nullptr) return;
    n->x.sum = n->x.value;
    n->x.size = 1;
    n->x.min = n->x.max = n->x.value;
    if (n->l) {
        push(n->l);
        n->x.sum += n->l->x.sum;
        n->x.size += n->l->x.size;
        n->x.min = min(n->x.min, n->l->x.min);
        n->x.max = max(n->x.max, n->l->x.max);
    }
    if (n->r) {
        push(n->r);
        n->x.sum += n->r->x.sum;
        n->x.size += n->r->x.size;
        n->x.min = min(n->x.min, n->r->x.min);
        n->x.max = max(n->x.max, n->r->x.max);
    }
}

void rotate(Node* n) {
    if (n->p == nullptr) return;
    Node *p = n->p, *gp = p->p;
    push(gp);
    push(p);
    push(n);
    if (p->l == n) {
        p->l = n->r;
        if (n->r != nullptr) {
            n->r->p = p;
        }
        n->r = p;
    }
    else {
        p->r = n->l;
        if (n->l != nullptr) {
            n->l->p = p;
        }
        n->l = p;
    }
    n->p = gp;
    p->p = n;
    if (gp != nullptr) {
        if (gp->l == p) gp->l = n;
        else gp->r = n;
    }
    update(p);
    update(n);
}

void splay(Node* n, Node* target = nullptr) {
    while (n->p != target) {
        Node *p = n->p, *gp = p->p;
        if (gp != target) {
            if ((n == p->l) == (p == gp->l)) rotate(p);
            else rotate(n);
        }
        rotate(n);
    }
}

void push_back(ll x) {
    Node* ptr = tree;
    Data val{};
    val.size = 1;
    val.value = x;
    val.sum = x;
    val.min = x;
    val.max = x;
    val.lazy = 0;
    val.flip = false;

    if (!ptr) {
        Node* n = new Node;
        n->p = n->l = n->r = nullptr;
        n->x = val;
        tree = n;
        ptrs.push_back(tree);
        return;
    }
    while (ptr->r) {
        push(ptr);
        ptr = ptr->r;
    }
    push(ptr);
    splay(ptr);
    tree = ptr;
    Node* n = new Node;
    n->l = n->r = nullptr;
    n->p = ptr;
    n->x = val;
```

```
    tree->r = n;

    splay(n);
    tree = n;
    ptrs.push_back(tree);
}

Node* find(Data val, bool do_splay = true) {
    // find node by value
    Node* root = tree;
    if (!root) return nullptr;
    Node* ptr = tree;
    while (true) {
        if (ptr->x.value == val.value) {
            if (do_splay) {
                splay(ptr);
                tree = ptr;
            }
            return ptr;
        }
        if (ptr->x.value < val.value && ptr->r) {
            ptr = ptr->r;
            continue;
        }
        if (ptr->x.value > val.value && ptr->l) {
            ptr = ptr->l;
            continue;
        }
        return nullptr;
    }
}

void del(Data val) {
    // do not use with ptrs
    if (!find(val)) return;
    Node* root = tree;
    if (root->l && root->r) {
        Node* l = root->l;
        Node* r = root->r;
        tree = l;
        l->p = nullptr;
        delete root;

        Node* m = tree;
        while (m->r) m = m->r;
        splay(m);
        tree = m;
        m->r = r;
        m->r->p = m;
        update(m);
    }
    else if (root->l) {
        tree = root->l;
        root->l->p = nullptr;
        delete root;
    }
    else if (root->r) {
        tree = root->r;
        root->r->p = nullptr;
        delete root;
    }
    else {
        tree = nullptr;
        delete root;
    }
}

Node* find_kth(ll k) {
    // find kth node from left
    Node* x = tree;
    while (true) {
        if (x == nullptr) return nullptr;
        push(x);
        ll l = (x->l) ? x->l->x.size : 0, r = (x->r) ?
        x->r->x.size : 0;
        if (l + r + 1 < k) return nullptr;
        if (k == 1 + l) {
            break;
        }
        if (k <= l) {
            x = x->l;
        }
        else {
            k -= l + 1;
            x = x->r;
        }
    }
    return x;
}

Node* interval(ll l, ll r) {
    // gather node which index is [l, r]
    if (!tree) return nullptr;
    ll n = tree ? tree->x.size : 0;
    Node* x = nullptr;
    if (l > 1) {
        x = find_kth(l - 1);
        if (x) {
            splay(x);
            tree = x;
        }
    }

    Node* y = nullptr;
    if (r < n) {
        y = find_kth(r + 1);
        if (y) {
            if (x) {
                splay(y, x);
            }
            else {
                splay(y);
                tree = y;
            }
        }
    }
    Node* res = nullptr;
    if (!x && !y) {
        res = tree;
    } else if (!x) {
        res = tree->l;
    } else if (!y) {
        res = tree->r;
    } else {
        res = tree->r->l;
    }
    push(res);
    return res;
}

void print(Node* n) {
    push(n);
    if (n->l) print(n->l);
    cout << n->x.value << " ";
    if (n->r) print(n->r);
}

void print_tree_helper(Node* n, int depth, string prefix) {
    if (!n) return;

    for (int i = 0; i < depth; i++) cout << "  ";
    cout << prefix << ": " << n->x.value
         << " (size=" << n->x.size
         << ", sum=" << n->x.sum;
    cout << ", lazy=" << n->x.flip;
    cout << ")\n";

    if (n->l || n->r) {
        if (n->l) {
            print_tree_helper(n->l, depth + 1, "L");
        } else {
            for (int i = 0; i <= depth; i++) cout << "  ";
            cout << "L: null\n";
        }

        if (n->r) {
            print_tree_helper(n->r, depth + 1, "R");
        } else {
            for (int i = 0; i <= depth; i++) cout << "  ";
            cout << "R: null\n";
        }
    }
}
```

```
void print_tree() {
    // for debug
    cout << "=== Tree Structure ===\n";
    if (!tree) {
        cout << "Empty tree\n";
        return;
    }
    print_tree_helper(tree, 0, "ROOT");
    cout << "\n";
}


void q1(ll l, ll r) {
    // [l, l+1, ..., r-1, r] = [r, r-1, ... l+1, l]
    Node* res = interval(l, r);
    cout << res->x.min << " " << res->x.max << " " <<
    res->x.sum << "\n";
    res->x.flip ^= true;
    push(res);
}

void q2(ll l, ll r, ll x) {
    // range shift in [l, r]
    Node* res = interval(l, r);
    cout << res->x.min << " " << res->x.max << " " <<
    res->x.sum << "\n";
    x += (r - l + 1) * tree->x.size;
    x %= (r - l + 1);
    if (x == 0) return;
    Node* r1 = interval(l, r);
    r1->x.flip ^= true;
    push(r1);
    Node* r2 = interval(l, l + x - 1);
    r2->x.flip ^= true;
    push(r2);
    Node* r3 = interval(l + x, r);
    r3->x.flip ^= true;
    push(r3);
}

void q3(ll i) {
    // find v[i]
    Node* res = find_kth(i);
    cout << res->x.value << "\n";
}

void q4(ll i) {
    // find i s.t. v[i] = n
    splay(ptrs[i - 1]);
    tree = ptrs[i - 1];
    push(tree);
    if (tree->l) cout << tree->l->x.size + 1 << "\n";
    else cout << "1\n";
}
```

4 Math

4.1 Template code

```
using LL = __int128_t;
using pLL = pair<LL, LL>;

inline LL modmul(LL a, LL b, LL c) {
    return a * b % c;
}
inline LL modadd(LL a, LL b, LL c) {
    return (a + b) % c;
}
ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
}
ll lcm(ll a, ll b) {
    if(a && b) return a * (b / gcd(a, b));
    return a+b;
}
LL POW(LL a, LL b, LL rem){
    LL p = 1;
    a %= rem;
    for(; b; b >>= 1,a = (a * a) % rem){
        if(b & 1) p = (p * a) % rem;
    }
    return p;
}
ll isqrt(ll x) {
    ll r1 = sqrt(x);
    for (ll i = r1 - 2; i < r1 + 3; i++) {
        if (i * i <= x && (i + 1) * (i + 1) > x) return i;
    }
    return -1;
}
tuple<ll, ll, ll> extended_gcd(ll a, ll b){
    // return (g, x, y) s.t. ax + by = g
    if (a == 0) return {b, 0, 1};
    auto [g, x, y] = extended_gcd(b % a, a);
    return {g, y - (b / a) * x, x};
}
ll modinverse(ll a, ll m) {
    return (get<1>(extended_gcd(a, m))%m+m)%m;
}
```

4.2 Primality test, Pollard rho, Euler phi

```
bool is_prime(ll n) {
    if (n < 2 || n % 2 == 0 || n % 3 == 0) return n == 2 || n
    == 3;
    ll k = __builtin_ctzll(n - 1), d = n - 1 >> k;
    for (ll a : { 2, 325, 9375, 28178, 450775, 9780504,
    1795265022 }) {
        LL p = POW(a % n, d, n);
        ll i = k;
        while (p != 1 && p != n - 1 && a % n && i--) p = p * p
        % n;
        if (p != n - 1 && i != k) return false;
    }
    return true;
}

ll pollard(ll n) {
    auto f = [n](ll x) { return modadd(modmul(x, x, n), 3, n);
    };
    ll x = 0, y = 0, t = 30, p = 2, i = 1, q;
    while (t++ % 40 || gcd(p, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if (q = modmul(p, abs(x - y), n)) p = q;
        x = f(x), y = f(f(y));
    }
    return gcd(p, n);
}

vector<ll> factor(ll n) {
    if (n == 1) return {};
    if (is_prime(n)) return { n };
    ll x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    sort(l.begin(), l.end());
    return l;
}

ll euler_phi(ll n){
    vector<ll> v = factor(n);
    for (ll i = 0; i < v.size(); i++) {
        if (i == 0 || v[i - 1] < v[i]) {
            n /= v[i];
            n *= v[i] - 1;
        }
    }
    return n;
}
```

4.3 Chinese Remainder theorem

```
// find (a, b) s.t. a + bi = r[j] (mod m[j]) for all i, j
pll crt(vector<ll> r, vector<ll> m){
    const int n = r.size(); LL r0 = 0, m0 = 1;
    for (int i = 0; i < n; i++) {
        LL r1 = r[i], m1 = m[i];
        if (m0 < m1) swap(r0, r1), swap(m0, m1);
        if (m0 % m1 == 0 && r0 % m1 != r1) return {0, 0};
        if (m0 % m1 == 0) continue;
        LL g = gcd(m0, m1);
        if ((r1 - r0) % g) return {0, 0};
        LL u0 = m0 / g, u1 = m1 / g;
        LL x = (r1 - r0) / g % u1 * modinverse(u0, u1) % u1;
```

```
        r0 += x * m0, m0 *= u1; if (r0 < 0) r0 += m0;
    }
    return {r0, m0};
};
```

4.4 Linear Sieve

```
struct sieve {
    const ll MAXN = 101010;
    vector<ll> sp, e, phi, mu, tau, sigma, primes;
    // sp : smallest prime factor, e : exponent, phi : euler
    phi, mu : mobius
    // tau : num of divisors, sigma : sum of divisors
    sieve(ll sz) {
        sp.resize(sz + 1), e.resize(sz + 1), phi.resize(sz +
        1), mu.resize(sz + 1),
            tau.resize(sz + 1), sigma.resize(sz + 1);
        phi[1] = mu[1] = tau[1] = sigma[1] = 1;
        for (ll i = 2; i <= sz; i++) {
            if (!sp[i]) {
                primes.push_back(i), e[i] = 1, phi[i] = i - 1,
                mu[i] = -1, tau[i] = 2;
                sigma[i] = i + 1;
            }
            for (auto j : primes) {
                if (i * j > sz) break;
                sp[i * j] = j;
                if (i % j == 0) {
                    e[i * j] = e[i] + 1, phi[i * j] = phi[i] *
                    j, mu[i * j] = 0,
                          tau[i * j] = tau[i] / e[i * j] *
                          (e[i * j] + 1),
                          sigma[i * j] = sigma[i] * (j - 1) /
                          (POW(j, e[i * j], 1e18) - 1) *
                                        (POW(j, e[i * j] + 1,
                                        1e18) - 1) / (j - 1);
                    break;
                }
                e[i * j] = 1, phi[i * j] = phi[i] * phi[j],
                mu[i * j] = mu[i] * mu[j],
                      tau[i * j] = tau[i] * tau[j], sigma[i *
                      j] = sigma[i] * sigma[j];
            }
        }
    }
};
```

4.5 FFT

```
#define sz(v) ((int)(v).size())
#define all(v) (v).begin(), (v).end()
using complex_t = complex<double>;
void fft(vector<complex_t>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, {1, 0});
    static vector<complex_t> rt(2, {1, 0});
    // (^ 10% faster if double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        for (int i=k;i<k+k;i++) rt[i] = R[i] = i&1 ? R[i/2] *
        x : R[i/2];
    }
    vector<int> rev(n);
    for (int i=0;i<n;i++) rev[i] = (rev[i / 2] | (i & 1) << L)
    / 2;
    for (int i=0;i<n;i++) if (i < rev[i]) swap(a[i],
    a[rev[i]]);
    for (int k = 1; k < n; k *= 2){
        for (int i = 0; i < n; i += 2 * k) for (int
        j=0;j<k;j++) {
            // complex_t z = rt[j+k] * a[i+j+k]; // (25%
            faster if hand-rolled) /// include-line
            auto x = (double *)&rt[j+k], y = (double
            *)&a[i+j+k]; /// exclude-line
            complex_t z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] +
            x[1]*y[0]); /// exclude-line
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
    }
}
```

```
template <typename T>
vector<T> conv_fft(const vector<T>& a, const vector<T>& b) {
    if (a.empty() || b.empty()) return {};
    vector<T> res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<complex_t> in(n), out(n);
    copy(all(a), begin(in));
    for (int i=0;i<sz(b);i++) in[i].imag(b[i]);
    fft(in);
    for (complex_t& x: in) x *= x;
    for (int i=0;i<n;i++) out[i] = in[-i & (n - 1)] -
    conj(in[i]);
    fft(out);
    for (int i=0;i<sz(res);i++) {
        res[i] = static_cast<T>(imag(out[i]) / (4 * n) +
        (is_integral_v<T> ? (imag(out[i]) > 0 ? 0.5 : -0.5) :
        0));
    }
    return res;
}

const int BASE = 1<<16;
vector<ll> conv_mod(const vector<ll>& a, const vector<ll>& b,
ll M) {
    int n = sz(a), m = sz(b);
    vector<int> a0(n), a1(n), b0(m), b1(m);
    for (int i = 0; i < n; i++) a0[i] = a[i] % BASE, a1[i] =
    a[i] / BASE;
    for (int i = 0; i < m; i++) b0[i] = b[i] % BASE, b1[i] =
    b[i] / BASE;

    auto z0 = conv_fft(a0, b0);
    auto z1 = conv_fft(a0, b1);
    auto z2 = conv_fft(a1, b0);
    auto z3 = conv_fft(a1, b1);

    vector<ll> res(sz(z0));
    for (int i = 0; i < sz(res); i++) {
        ll val = z0[i];
        val = (val + ((z1[i] + z2[i]) % M) * BASE) % M;
        val = (val + (z3[i] % M) * BASE % M * BASE) % M;
        res[i] = val;
    }
    return res;
}
```

4.6 NTT

```
using poly = vector<ll>;
const int mod = 998244353, w = 3;
ll pw(ll a, ll b){ll p=1; a%=mod; for
(;b;b>>=1,a=(a*a)%mod)if(b&1)p=(p*a)%mod;return p;}

void ntt(poly& f, bool inv = 0) {
    int n = f.size(), j = 0;
    vector<ll> root(n >> 1);
    for (int i = 1; i < n; i++) {
        int bit = (n >> 1);
        while (j >= bit) {
            j -= bit;
            bit >>= 1;
        }
        j += bit;
        if (i < j) swap(f[i], f[j]);
    }
    ll ang = pw(w, (mod - 1) / n);
    if (inv) ang = pw(ang, mod - 2);
    root[0] = 1;
    for (int i = 1; i < (n >> 1); i++) root[i] = root[i - 1] *
    ang % mod;
    for (int i = 2; i <= n; i <<= 1) {
        int step = n / i;
        for (int j = 0; j < n; j += i) {
            for (int k = 0; k < (i >> 1); k++) {
                ll u = f[j | k], v = f[j | k | i >> 1] *
                root[step * k] % mod;
                f[j | k] = (u + v) % mod;
                f[j | k | i >> 1] = (u - v) % mod;
                if (f[j | k | i >> 1] < 0) f[j | k | i >> 1]
                += mod;
            }
        }
```

```
    }
    ll t = pw(n, mod - 2);
    if (inv)
        for (int i = 0; i < n; i++) f[i] = f[i] * t % mod;
}

vector<ll> conv_ntt(poly& _a, poly& _b) {
    auto a = _a, b = _b;
    int n = 2;
    while (n < a.size() + b.size()) n <<= 1;
    a.resize(n);
    b.resize(n);
    ntt(a);
    ntt(b);
    for (int i = 0; i < n; i++) a[i] = a[i] * b[i] % mod;
    ntt(a, 1);
    return a;
}
```

4.7 Gauss-Jordan Elimination

```
/*
[[2, 3, 5],  (2x + 3y = 5)
 [4, 5, 9]]  (4x + 5y = 9)
=> ans = {1, 1}, a is rref
*/
int gauss_jordan(vector<vector<ll>>& a, ll mod, vector<ll>&
ans) {
    // return value  0 : no sol  1 : unique sol  2 : infinite
    sol
    int n = a.size();
    int m = a[0].size() - 1;
    vector<int> pos(m, -1);
    for (int col = 0, row = 0; col < m && row < n; col++) {
        int pivot = -1;
        for (int i = row; i < n; ++i) {
            if (a[i][col] % mod != 0) {
                pivot = i;
                break;
            }
        }
        if (pivot == -1) continue;
        swap(a[pivot], a[row]);
        pos[col] = row;

        ll inv = modinverse(a[row][col], mod);
        for (int j = col; j <= m; ++j) a[row][j] = (a[row][j]
        * inv) % mod;

        for (int i = 0; i < n; i++) {
            if (i != row && a[i][col] != 0) {
                ll factor = a[i][col];
                for (int j = col; j <= m; j++) {
                    a[i][j] = (a[i][j] - factor * a[row][j] %
                    mod + mod) % mod;
                }
            }
        }
        ++row;
    }

    ans.assign(m, 0);
    for (int i = 0; i < m; i++) {
        if (pos[i] != -1) {
            ans[i] = a[pos[i]][m];
        }
    }

    for (int i = 0; i < n; i++) {
        ll sum = 0;
        for (int j = 0; j < m; j++) {
            sum = (sum + ans[j] * a[i][j]) % mod;
        }
        if (sum != a[i][m]) {
            return 0;
        }
    }

    for (int i = 0; i < m; ++i) {
        if (pos[i] == -1) {
            return 2;
        }
    }

    return 1;
}
```

4.8 Sum of floor of linear

```
// returns sum(floor((ai + b) / c)) for i in [0, n]
ll fsum(ll a, ll b, ll c, ll n) {
    if (a == 0) return (b / c) * (n + 1);
    if (a >= c || b >= c) return ((n * n + n) / 2) * (a / c) +
    (n + 1) * (b / c) + fsum(a % c, b % c, c, n);
    return (a * n + b) / c * n - fsum(c, c - b - 1, a, (a * n
    + b) / c - 1);
}
```

4.9 Tonelli-Shanks Algorithm

```
ll tonelliShanks(ll n, ll p) {
    // find x s.t. x^2 = n (mod p), O((logP)^2)
    auto legendre = [&](ll a) {
        ll result = POW(a, (p - 1) / 2, p);
        return result == p - 1 ? -1 : (ll)result;
    };

    auto findNonResidue = [&]() {
        for (ll z = 2; z < p; z++) {
            if (legendre(z) == -1) return z;
        }
        return 2LL;
    };

    if (n == 0) return 0;
    if (p == 2) return n % 2;
    if (legendre(n) != 1) return -1;

    ll Q = p - 1, S = 0;
    while (Q % 2 == 0) {
        Q /= 2;
        S++;
    }

    if (S == 1) return POW(n, (p + 1) / 4, p);

    ll z = findNonResidue();
    ll M = S;
    ll c = POW(z, Q, p);
    ll t = POW(n, Q, p);
    ll R = POW(n, (Q + 1) / 2, p);

    while (t != 1) {
        ll i = 1;
        ll temp = (t * t) % p;
        while (temp != 1) {
            temp = (temp * temp) % p;
            i++;
        }
        ll b = POW(c, 1LL << (M - i - 1), p);
        M = i;
        c = (b * b) % p;
        t = (t * c) % p;
        R = (R * b) % p;
    }
    return R;
}
```

5 Geometry

5.1 Template code

```
const ld eps = 1e-12;
const ld MX = 1e18;
bool eq(ld x, ld y) {
    if (abs(x - y) <= eps) return true;
    return false;
}
struct Point {
    ld x, y;
    bool operator<(const Point& other) const {
        if (eq(x, other.x)) {
            return y < other.y;
        }
```

```
        return x < other.x;
    }
    bool operator==(const Point& other) const {
        return eq(x, other.x) && eq(y, other.y);
    }
};
struct Line {   // line segment
    Point a, b;
};
struct SLine {  // full line, y = ax + b
    ld a, b;
    bool ver;   // a == inf
    ld p;       // if ver then x = p
};
struct VLine {
    Point pos, dir;
    pair<ld, ld> range;
};
struct Ray {
    ld x, y, dir;
};

inline ld dist(Point p1, Point p2) {
    return sqrtl((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y)
    * (p1.y - p2.y));
}

inline ld ccw(Point p1, Point p2, Point p3) {
    return (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) *
    (p3.x - p1.x);
}

inline ld ccw_line(Line l1, Line l2) {
    pld v1 = {l1.b.x - l1.a.x, l1.b.y - l1.a.y}, v2 = {l2.b.x
    - l2.a.x, l2.b.y - l2.a.y};
    return v1.first * v2.second - v2.first * v1.second;
}

bool cross(Line l1, Line l2) {
    ld res1 = ccw(l1.a, l1.b, l2.a), res2 = ccw(l1.a, l1.b,
    l2.b);
    ld res3 = ccw(l2.a, l2.b, l1.a), res4 = ccw(l2.a, l2.b,
    l1.b);
    if (res1 * res2 == 0 && res3 * res4 == 0) {
        if (l1.a.x > l1.b.x || (l1.a.x == l1.b.x && l1.a.y >
        l1.b.y)) swap(l1.a, l1.b);
        if (l2.a.x > l2.b.x || (l2.a.x == l2.b.x && l2.a.y >
        l2.b.y)) swap(l2.a, l2.b);
        if (eq(l1.a.x, l1.b.x)) return max(min(l1.a.y,
        l1.b.y), min(l2.a.y, l2.b.y)) <= min(max(l1.a.y,
        l1.b.y), max(l2.a.y, l2.b.y)) + eps;
        return max(min(l1.a.x, l1.b.x), min(l2.a.x, l2.b.x))
        <= min(max(l1.a.x, l1.b.x), max(l2.a.x, l2.b.x)) +
        eps;
    }
    return res1 * res2 <= eps && res3 * res4 <= eps;
}

SLine ExtendLine(Line l) {
    SLine res{};
    if (eq(l.a.x, l.b.x)) {
        res.ver = true;
        res.a = res.b = MX;
        res.p = l.a.x;
    }
    else {
        res.ver = false;
        res.a = (l.a.y - l.b.y) / (l.a.x - l.b.x);
        res.b = l.a.y - res.a * l.a.x;
        res.p = 0;
    }
    return res;
}

Point proj(Point p, SLine l) {
    if (l.ver) return Point{l.p, p.y};
    if (eq(l.a, 0)) return Point{p.x, l.b};
    ld x = (l.a / (l.a * l.a + 1)) * (p.x / l.a + p.y - l.b);
    return Point{x, l.a * x + l.b};
}

Point reflect(Point p, SLine l) {
    Point q = proj(p, l);
    return Point{q.x * 2 - p.x, q.y * 2 - p.y};
}

Point Sintersection(SLine l1, SLine l2) {
    Point res{};
    if (!eq(l1.a, l2.a)) {
        if (!l1.ver && !l2.ver) {
            ld a = l1.a, b = l1.b, c = l2.a, d = l2.b;
            res.x = (d - b) / (a - c);
            res.y = a * res.x + b;
        }
        else if (!l1.ver) {
            res.x = l2.p;
            res.y = l1.a * res.x + l1.b;
        }
        else if (!l2.ver) {
            res.x = l1.p;
            res.y = l2.a * res.x + l2.b;
        }
    }
    else;
    return res;
}

Point intersection(Line l1, Line l2) {
    Point res{};
    if (cross(l1, l2)) {
        SLine L1 = ExtendLine(l1), L2 = ExtendLine(l2);
        if (L1.a == L2.a) {
            if (l1.a < l1.b) swap(l1.a, l1.b);
            if (l2.a < l2.b) swap(l2.a, l2.b);
            if (l1.a == l2.b) return l1.a;
            if (l1.b == l2.a ) return l1.b;
            // infinite intersection points
        }
        else {
            res = Sintersection(L1, L2);
        }
    }
    return res;
}

inline ld grad(Point a, Point b) {
    return (b.y - a.y) / (b.x - a.x);
}

void sortbyPoint(vector<Point>& v, Point p) {
    sort(v.begin(), v.end(), [p](Point x, Point y) {return
    atan2(x.y - p.y, x.x - p.x) < atan2(y.y - p.y, y.x -
    p.x);});
}

using Polygon = vector<Point>;

ld area(const Polygon& v) {
    if (v.size() < 3) return 0;
    ld a = v[v.size() - 1].x * v[0].y, b = v[v.size() - 1].y *
    v[0].x;
    for (int i = 0; i < v.size() - 1; i++) {
        a += v[i].x * v[i + 1].y;
        b += v[i].y * v[i + 1].x;
    }
    return (ld)abs(a - b) / 2;
}
```

5.2 Convex Hull(Graham Scan)

```
vector<Point> ConvexHull(vector<Point> v) {
    sortbyPoint(v, v[0]);
    vector<Point> convex;
    for (auto i : v) {
        ll len = convex.size();
        while (len>=2&&ccw(convex[len-2],i,convex[len-1])) {
            len--;
            convex.pop_back();
        }
        convex.push_back(i);
    }
    return convex;
}
```

5.3 Convex Hull(Monotone Chain)

```
vector<Point> ConvexHull(vector<Point> v, bool inclusive =
false) {
    sort(v.begin(), v.end());
    vector<Point> lower, upper;
    ll ltop = 0, utop = 0;
    ld ep = eps;
    if (inclusive) ep = -ep;
    for (auto p : v) {
        while (ltop >= 2 && ccw(lower[ltop - 2], lower[ltop -
        1], p) <= ep) {
            lower.pop_back();
            ltop --;
        }
        while (utop >= 2 && ccw(upper[utop - 2], upper[utop -
        1], p) >= -ep) {
            upper.pop_back();
            utop --;
        }
        lower.push_back(p); ltop ++;
        upper.push_back(p); utop ++;
    }
    reverse(upper.begin(), upper.end());
    if (utop > 2) lower.insert(lower.end(), upper.begin() + 1,
    upper.end() - 1);
    if (ltop >= 2) lower.erase(unique(lower.begin(),
    lower.end()), lower.end());
    return lower;
}
```

5.4 Rotating Callipers

```
tuple<ll, ll, ld> FurthestPoint(vector<Point> inp) {
    // returns <index of p1, index of p2, dist(p1, p2)> where
    p1, p2 are furthest points
    auto v = ConvexHull(inp);
    if (v.size() < 2) return {0, 1, 0};
    ll a = 0, c = 1, n = v.size();
    ll r1 = 0, r2 = 1;
    ld mx = (v[a].x - v[c].x) * (v[a].x - v[c].x) + (v[a].y -
    v[c].y) * (v[a].y - v[c].y);
    if (v.size() > 2) {
        v.push_back(v[0]);
        while (a < n) {
            ll b = a + 1;
            ll d = c + 1;
            Line l1 = {v[a], v[b]}, l2 = {v[c], v[d]};
            if (ccw_line(l1, l2) > -eps) {
                c ++;
                if (c >= n) c = 0;
            }
            else {
                a ++;
            }
            if (mx < (v[a].x - v[c].x) * (v[a].x - v[c].x) +
            (v[a].y - v[c].y) * (v[a].y - v[c].y)) {
                mx = (v[a].x - v[c].x) * (v[a].x - v[c].x) +
                (v[a].y - v[c].y) * (v[a].y - v[c].y);
                r1 = a;
                r2 = c;
            }
        }
    }

    for (ll i = 0; i < inp.size(); i++) {
        if (inp[i] == v[r1]) {
            r1 = i;
            break;
        }
    }
    for (ll i = 0; i < inp.size(); i++) {
        if (r1 != i && inp[i] == v[r2]) {
            r2 = i;
            break;
        }
    }
    return {r1, r2, mx};
}
```

5.5 Point in a Convex Polygon

```
bool inside_polygon(const Polygon& v, Point p) {
    // works only if polygon is convex and sorted by
    counterclockwise, O(logN)
    ll n = v.size(), i, j;
    if (ccw(v[n - 1], v[0], p) < 0 || ccw(v[1], v[0], p) > 0)
    return false;
    ll l = 1, r = n - 1;
    while (l + 1 < r) {
        ll mid = (l + r) / 2;
        if (ccw(v[mid], v[0], p) > 0) r = mid;
        else l = mid;
    }
    return ccw(v[l], p, v[r]) < 0;
}
```

5.6 Intersection of Convex Polygon

```
Polygon intersection(const Polygon& a, const Polygon& b) {
    Polygon res = a;
    ll s = b.size();
    for (ll i = 0; i < s; i++) {
        Point p = b[i], q = b[(i + 1) % s];
        Polygon tmp;
        ll sz = res.size();
        for (ll j = 0; j < sz; j++) {
            Point x = res[j], y = res[(j + 1) % sz];
            bool b1 = ccw(x, p, q) > -eps, b2 = ccw(y, p, q) >
            -eps;
            if (b1 && b2) {
                tmp.push_back(y);
            }
            else if (b1) {
                SLine l1 = ExtendLine(Line{x, y}), l2 =
                ExtendLine(Line{p, q});
                tmp.push_back(Sintersection(l1, l2));
            }
            else if (b2) {
                SLine l1 = ExtendLine(Line{x, y}), l2 =
                ExtendLine(Line{p, q});
                tmp.push_back(Sintersection(l1, l2));
                tmp.push_back(y);
            }
        }
        res = tmp;
    }
    return res;
}
```

5.7 Minimum Enclosing Circle

```
pair<Point, ld> MinimumEnclosingCircle(vector<Point> v) {
    // amortized O(n) if shuffled
    random_device rd;
    mt19937 gen(rd());
    shuffle(v.begin(), v.end(), gen);

    int i, j, k, n = v.size();
    Point p = v[0];
    ld r = 0;
    for (i = 0; i < n; i++) {
        if (dist(p, v[i]) <= r) continue;
        p = v[i], r = 0;
        for (j = 0; j < i; j++) {
            if (dist(p, v[j]) <= r) continue;
            p = mid(v[i], v[j]);
            r = dist(v[i], v[j]) / 2;
            for (k = 0; k < j; k++) {
                if (dist(p, v[k]) <= r) continue;
                p = Sintersection(perp_SLine(Line{v[i],
                v[j]}), perp_SLine(Line{v[j], v[k]}));
                r = dist(p, v[i]);
            }
        }
    }
    return {p, r};
}
```

6 DP

6.1 Longest Increasing Subsequence

```
vector<ll> lis(vector<ll>& arr) {
    int n = arr.size();
```

```
    vector<ll> tmp = vector<ll>();
    vector<ll> from = vector<ll>();
    for (int x : arr) {
        int loc = lower_bound(tmp.begin(), tmp.end(), x) -
        tmp.begin();
        if (loc == tmp.size()) {
            tmp.push_back(x);
        } else {
            tmp[loc] = x;
        }
        from.push_back(loc);
    }
    vector<ll> lis = vector<ll>(tmp.size());
    int target = tmp.size() - 1;
    for (int i = n - 1; i >= 0; i--) {
        if (target == from[i]) {
            lis[target--] = arr[i];
        }
    }
    return lis;
}
```

6.2 Convex Hull Optimization

$O(n^2) \rightarrow O(n \log n)$
DP 점화식 꼴
$D[i] = \max_{j<i}(D[j] + b[j] * a[i])$ $(b[k] \leq b[k+1])$
$D[i] = \min_{j<i}(D[j] + b[j] * a[i])$ $(b[k] \geq b[k+1])$
특수조건) $a[i] \leq a[i+1]$ 도 만족하는 경우, 마지막 쿼리의 위치를 저장해두면 이분검색이 필요없어지기 때문에 amortized $O(n)$ 에 해결할 수 있음

```
struct CHTLinear {
    struct Line {
        long long a, b;
        long long y(long long x) const { return a * x + b; }
    };
    vector<Line> stk;
    int qpt;
    CHTLinear() : qpt(0) { }
    // when you need maximum :  (previous l).a < (now l).a
    // when you need minimum :  (previous l).a > (now l).a
    void pushLine(const Line& l) {
        while (stk.size() > 1) {
            Line& l0 = stk[stk.size() - 1];
            Line& l1 = stk[stk.size() - 2];
            if ((l0.b - l.b) * (l0.a - l1.a) > (l1.b - l0.b) *
            (l.a - l0.a)) break;
            stk.pop_back();
        }
        stk.push_back(l);
    }
    // (previous x) <= (current x)
    // it calculates max/min at x
    long long query(long long x) {
        while (qpt + 1 < stk.size()) {
            Line& l0 = stk[qpt];
            Line& l1 = stk[qpt + 1];
            if (l1.a - l0.a > 0 && (l0.b - l1.b) > x * (l1.a -
            l0.a)) break;
            if (l1.a - l0.a < 0 && (l0.b - l1.b) < x * (l1.a -
            l0.a)) break;
            ++qpt;
        }
        return stk[qpt].y(x);
    }
};
```

6.3 Divide & Conquer Optimization

$O(kn^2) \rightarrow O(kn \log n)$
조건 1) DP 점화식 꼴
$D[t][i] = \min_{j<i}(D[t-1][j] + C[j][i])$
조건 2) $A[t][i]$는 $D[t][i]$의 답이 되는 최소의 j라 할 때, 아래의 부등식을 만족해야 함
$A[t][i] \leq A[t][i+1]$
조건 2-1) 비용C가 다음의 사각부등식을 만족하는 경우도 조건 2)를 만족하게 됨
$C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$ $(a \leq b \leq c \leq d)$

```
//To get D[t][s...e] and range of j is [l, r]
void f(int t, int s, int e, int l, int r){
  if(s > e) return;
  int m = s + e >> 1;
  int opt = l;
  for(int i=l; i<=r; i++){
    if(D[t-1][opt] + C[opt][m] > D[t-1][i] + C[i][m]) opt = i;
  }
  D[t][m] = D[t-1][opt] + C[opt][m];
  f(t, s, m-1, l, opt);
  f(t, m+1, e, opt, r);
}
```

6.4 Knuth Optimization

$O(n^3) \rightarrow O(n^2)$
조건 1) DP 점화식 꼴
$D[i][j] = \min_{i<k<j}(D[i][k] + D[k][j]) + C[i][j]$
조건 2) 사각 부등식
$C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$ $(a \leq b \leq c \leq d)$
조건 3) 단조성
$C[b][c] \leq C[a][d]$ $(a \leq b \leq c \leq d)$
결론) 조건 2, 3을 만족한다면 $A[i][j]$를 $D[i][j]$의 답이 되는 최소의 k라 할 때, 아래의 부등식을 만족하게 됨
$A[i][j-1] \leq A[i][j] \leq A[i+1][j]$
3중 루프를 돌릴 때 위 조건을 이용하면 최종적으로 시간복잡도가 $O(n^2)$이 됨

```
for (i = 1; i <= n; i++) {
  cin >> a[i];
  s[i] = s[i - 1] + a[i];
  dp[i - 1][i] = 0;
  assist[i - 1][i] = i;
}
for (i = 2; i <= n; i++) {
  for (j = 0; j <= n - i; j++) {
    dp[j][i + j] = 1e9 + 7;
    for (k = assist[j][i + j - 1]; k <= assist[j + 1][i + j];
    k++) {
      if (dp[j][i + j] > dp[j][k] + dp[k][i + j] + s[i + j] -
      s[j]) {
        dp[j][i + j] = dp[j][k] + dp[k][i + j] + s[i + j] -
        s[j];
        assist[j][i + j] = k;
      }
    }
  }
}
```

7 Graph

7.1 Dijkstra Algorithm

```
vector<ll> dijk(ll n, ll s){
    vector<ll>dis(n,INF);
    priority_queue<pll, vector<pll>, greater<pll> > q; //
    pair(dist, v)
    dis[s] = 0;
    q.push({dis[s], s});
    while (!q.empty()){
        while (!q.empty() && visit[q.top().second]) q.pop();
        if (q.empty()) break;
        ll next = q.top().second; q.pop();
        visit[next] = 1;
        for (ll i = 0; i < adj[next].size(); i++)
            if (dis[adj[next][i].first] > dis[next] +
            adj[next][i].second){
            dis[adj[next][i].first] = dis[next] +
            adj[next][i].second;
            q.push({dis[adj[next][i].first],
            adj[next][i].first});}}
    for(ll i=0;i<n;i++)if(dis[i]==INF)dis[i]=-1;
    return dis;
}
```

7.2 Lowest Common Ancestor

```
const ll MAX_PARENT = log(N);
vl level(N+1);
vector<vl> lca(N+1,vl(MAX_PARENT+1));
```

```
auto BuildLCA = [&](auto self,ll g,ll p,ll deep) ->void {
    check[g] = 1;
    level[g] = deep;
    ll cnt = 0;
    while(p!=0){
        lca[g][cnt] = p;
        p = lca[p][cnt++];
    }

    for (auto i : P[g]) {
        if (check[i]==1)continue;
        self(self,i,g,deep+1);
    }
};

auto FindLCA = [&](ll t1,ll t2)->ll {
    if(level[t1] < level[t2])swap(t1,t2);

    for(int i=MAX_PARENT-1;i>=0;i--){
        if(level[lca[t1][i]] >= level[t2]){
            t1 = lca[t1][i];
        }
    }
    if(t1==t2)return t1;

    for(int i=MAX_PARENT-1;i>=0;i--){
        if(lca[t1][i] != lca[t2][i]){
            t1 = lca[t1][i];
            t2 = lca[t2][i];
        }
    }
    return lca[t1][0];
};
```

7.3 Union Find

```
ll find(ll i, vector<ll>& p) {
    if (p[i] == i) {
        return i;
    }
    p[i] = find(p[i], p);
    return p[i];
}
void uni(ll i, ll j, vector<ll>& p, vector<ll>& r) {
    ll root_i = find(i, p);
    ll root_j = find(j, p);
    if (root_i != root_j) {
        if (r[root_i] >= r[root_j]) {
            p[root_j] = root_i;
            r[root_i] += r[root_j];
        }
        else {
            p[root_i] = root_j;
            r[root_j] += r[root_i];
        }
    }
    //현재 uni는 사이즈 방식, 랭크 방식으로 하고 싶으면 r에 +1만
    하면 됨.
}

int main(){
    vector<ll> p(100001, 0);
    vector<ll> r(100001, 0);
    for(ll i=0; i<100001; i++) {
        p[i] = i;
        r[i] = 1;
    }

}
```

7.4 SCC (Kosaraju's Algorithm)

```
void dfs_1(ll n, vector<ll>& stack, vector<ll>& memo,
vector<vector<ll>>& graph) {
    if (memo[n] != 0) {
        return;
    }
    memo[n] = 1;
    for (ll i=0; i<graph[n].size(); i++) {
        dfs_1(graph[n][i], stack, memo, graph);
    }
    stack.push_back(n);
}
void dfs_2(ll n, vector<ll>& temp, vector<ll>& memo,
vector<vector<ll>>& graph) {
    if (memo[n] != 0) {
        return;
    }
    memo[n] = 1;
    temp.push_back(n);
    for (ll i=0; i<graph[n].size(); i++) {
        dfs_2(graph[n][i], temp, memo, graph);
    }
}
int main(){
    ll V, E;
    cin >> V >> E;
    vector<vector<ll>> graph_1(V + 1);
    vector<vector<ll>> graph_2(V + 1);
    ll a, b;
    for (ll i=0; i<E; i++) {
        cin >> a >> b;
        graph_1[a].push_back(b);
        graph_2[b].push_back(a);
    }
    vector<ll> stack;
    vector<ll> memo(V + 1, 0);
    vector<vector<ll>> SCC(V + 1);
    for (ll i=1; i<=V; i++) {
        if (memo[i] == 0) {
            dfs_1(i, stack, memo, graph_1);
        }
    }
    ll count = 0;
    memo = vector<ll>(V + 1, 0);
    while (!stack.empty()) {
        ll n = stack.back();
        stack.pop_back();
        if (memo[n] != 0) {
            continue;
        }
        count += 1;
        vector<ll> temp;
        dfs_2(n, temp, memo, graph_2);
        sort(temp.begin(), temp.end());
        SCC[temp[0]] = temp;
    }
    cout << count << "\n";
    for (ll i=1; i<=V; i++) {
        if (!SCC[i].empty()) {
            for (ll j=0; j<SCC[i].size(); j++) {
                cout << SCC[i][j] << " ";
            }
            cout << "-1\n";
        }
    }
}
```

7.5 Maximum Flow (Dinic's Algorithm)

```
struct Maxflow {
    ll v, s, t;
    struct Edge {
        ll n, cap, rev;
    };
    vector<ll> level, work;
    vector<vector<Edge>> graph;

    void init(ll V, ll S, ll T) {
        v = V; s = S; t = T;
        graph.resize(v);
        level.resize(v);
        work.resize(v);
        for (ll i = 0; i < v; i++) graph[i].clear();
    }

    void add_edge(ll start, ll end, ll cap) {
        graph[start].push_back(Edge(end, cap,
        graph[end].size()));
        graph[end].push_back(Edge(start, 0,
        graph[start].size() - 1));
    }

    bool bfs() {
```

```
        fill(level.begin(), level.end(), -1);
        queue<ll> q;
        q.push(s);
        level[s] = 0;
        while (!q.empty()) {
            ll cur = q.front();
            q.pop();
            for (auto a : graph[cur]) {
                if (level[a.n] == -1 && a.cap > 0) {
                    level[a.n] = level[cur] + 1;
                    q.push(a.n);
                }
            }
        }
        return level[t] != -1;
    }

    ll dfs(ll pos, ll flow) {
        if (pos == t) return flow;
        for (ll &i = work[pos]; i < graph[pos].size(); i++) {
            Edge &e = graph[pos][i];
            if (level[pos] + 1 == level[e.n] && e.cap > 0) {
                ll res = dfs(e.n, min(flow, e.cap));
                if (res > 0) {
                    e.cap -= res;
                    graph[e.n][e.rev].cap += res;
                    return res;
                }
            }
        }
        return 0;
    }

    ll dinic() {
        ll res = 0;
        while (bfs()) {
            fill(work.begin(), work.end(), 0);
            while (ll r = dfs(s, 1e18)) {
                res += r;
            }
        }
        return res;
    }
};
```

7.6 Heavy-Light Decomposition

```
// (*) <- use only if the graph is edge-weighted
// val[v] is weight of edge (p[v], v)
vector<vector<ll>> adj(n, vector<ll>());
vector<ll> sz(n), dep(n), p(n), top(n), idx(n);
// size of subtree, depth, parent, root of heavy chain
// xth vertex corresponds to idx[x] in segtree
vector<ll> val(n);
vector<vector<ll>> weight(n, vector<ll>()); // (*)
vector<pll> vertice; // (*)

vector<ll> visited(n);
static function <void(ll)> init1 = [&](ll v) {
    visited[v] = 1;
    sz[v] = 1;
    for (ll i = 0; i < adj[v].size(); i++) {
        ll a = adj[v][i];
        if (visited[a]) continue;
        p[a] = v;
        dep[a] = dep[v] + 1;
        init1(a);
        sz[v] += sz[a];
        val[a] = weight[v][i]; // (*)
        if (dep[adj[v][0]] < dep[a] || sz[a] > sz[adj[v][0]])
        {
            swap(adj[v][i], adj[v][0]);
            swap(weight[v][i], weight[v][0]); // (*)
        }
    }
};

ll cnt = -1;
static function <void(ll)> init2 = [&](ll v) {
    visited[v] = 1;
    idx[v] = ++cnt;
    for (ll i = 0; i < adj[v].size(); i++) {
        ll a = adj[v][i];
        if (visited[a]) continue;
        if (i == 0) top[a] = top[v];
        else top[a] = a;
        init2(a);
    }
};

// input example
for (i = 0; i < n - 1; i++) {
    ll a, b, c;
    cin >> a >> b;
    cin >> c; // (*)
    adj[a].push_back(b);
    adj[b].push_back(a);
    weight[a].push_back(c); // (*)
    weight[b].push_back(c); // (*)
    vertice.emplace_back(a, b); // (*)
}

init1(0);
for (i = 0; i < n; i++) visited[i] = 0;
init2(0);

for (i = 0; i < n - 1; i++) { // (*)
    auto [l, r] = vertice[i];
    if (dep[l] > dep[r]) vertice[i] = make_pair(r, l);
}

LazySegmentTree seg(n);
vector<ll> arr(n);
for (i = 0; i < n; i++) arr[i] = weight_of_vertex_i; //
vertex-weighted
for (i = 0; i < n; i++) arr[idx[i]] = val[i]; // (*)
edge-weighted
seg.init(arr);

function <ll(ll, ll)> query = [&](ll v, ll u) -> ll {
    ll res = 0;
    while (top[v] != top[u]) {
        if (dep[top[v]] > dep[top[u]]) swap(v, u);
        res += seg.query(idx[top[u]], idx[u]);
        u = p[top[u]];
    }
    if (v == u) return res; // (*)
    if (dep[v] > dep[u]) swap(v, u);
    res += seg.query(idx[v], idx[u]); // vertex-weighted
    res += seg.query(idx[adj[v][0]], idx[u]); // (*)
    edge-weighted
    return res;
};

function <void(ll, ll, ll)> update = [&](ll v, ll u, ll x) {
    while (top[v] != top[u]) {
        if (dep[top[v]] > dep[top[u]]) swap(v, u);
        seg.update(idx[top[u]], idx[u], x);
        u = p[top[u]];
    }
    if (v == u) return; // (*)
    if (dep[v] > dep[u]) swap(v, u);
    //seg.update(idx[v], idx[u], x); // vertex-weighted
    seg.update(idx[adj[v][0]], idx[u], x); // (*)
    edge-weighted
};

auto subtree_query = [&](ll v) -> ll { // lazy query
    return seg.query(idx[v], idx[v] + sz[v] - 1); //
    vertex-weighted
    return (sz[v] == 1) ? 0 : seg.query(idx[v] + 1, idx[v] +
    sz[v] - 1); // (*) edge-weighted
};

auto subtree_update = [&](ll v, ll x) -> void { // lazy update
    seg.update(idx[v], idx[v] + sz[v] - 1, x); //
    vertex-weighted
    if (sz[v] > 1) seg.update(idx[v] + 1, idx[v] + sz[v] - 1,
    x); // (*) edge-weighted
};
```

7.7 Faker's Algorithm

```
// for any topological order p in tree, minimize sum(a[i] *
b[j]) for all p[i] < p[j]
ll faker_algorithm(ll n, const vector<ll>& tree, vector<ll> a,
vector<ll> b, ll root = 0) {
    ll i, j, k, res = 0;
    vector<ll> ds;
    for (i = 0; i < n; i++) {
        ds.push_back(i);
    }
    struct Data {
        pll val;
        ll idx;
    };
    priority_queue<Data, vector<Data>, decltype([](Data xx,
    Data yy) {
        pll x = xx.val, y = yy.val;
        if (y.first == 0) return true;
        if (x.first == 0) return false;
        if (x.first * y.first > 0) return x.second * y.first <
        x.first * y.second;
        return x.second * y.first > x.first * y.second;
    })> pq;

    auto Push = [&](ll x, ll y, ll z) -> void {
        pq.push(Data(make_pair(x, y), z));
    };

    for (i = 0; i < n; i++) {
        if (i == root) continue;
        Push(a[i], b[i], i);
    }

    function <ll(ll)> find = [&](ll x) {
        if (x == ds[x]) return x;
        return ds[x] = find(ds[x]);
    };

    while (!pq.empty()) {
        auto [dat, idx] = pq.top();
        pq.pop();
        auto [x, y] = dat;
        if (a[idx] != x || b[idx] != y || idx == root)
        continue;

        ll p = find(tree[idx]);
        ds[idx] = p;
        res += a[p] * b[idx];
        a[p] += a[idx];
        b[p] += b[idx];
        Push(a[p], b[p], p);
    }

    return res;
}
```

8 String

8.1 KMP, Z

```
void calculate_pi(vector<ll>& pi, const string& str) {
    pi[0] = -1;
    for (ll i = 1, j = -1; i < str.size(); i++) {
        while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
        if (str[i] == str[j + 1]) pi[i] = ++j;
        else pi[i] = -1;
    }
}

vector<ll> kmp(const string& text, const string& pattern) {
    vector<ll> pi(pattern.size()), ans;
    if (pattern.empty()) return ans;
    calculate_pi(pi, pattern);
    for (ll i = 0, j = -1; i < text.size(); i++) {
        while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
        if (text[i] == pattern[j + 1]) {
            j++;
            if (j + 1 == pattern.size()) ans.push_back(i - j),
            j = pi[j];
        }
    }
    return ans;
}

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

8.2 Manacher's Algorithm

```
vector<ll> manacher(string s) {
    string t = "#";
    for (char c : s) {t += c; t += '#';}
    s = t;
    ll n = s.size(), r = 0, l = 0;
    vector<ll> p(n);
    for (ll i = 1; i < n; i++) {
        if (r < i) p[i] = 0;
        else p[i] = min(r - i, p[l * 2 - i]);
        while (i + p[i] + 1 < n && i - p[i] - 1 >= 0 && s[i +
        p[i] + 1] == s[i - p[i] - 1]) p[i] ++;
        if (r < i + p[i]) {
            r = i + p[i];
            l = i;
        }
    }
    return p;
}
```

8.3 Suffix Array, LCP Array

```
string str;
// str input is needed

ll len = str.size();
vl rank(len), suf(len), lcp(len);

auto build_suffix = [&]() { // 시간 복잡도  O(Nlog(N))
    for (ll i = 0; i < len; i++) {
        rank[i] = str[i] - 'a'; // 'a' should be changed
        unless input is only composed of lowercase
        suf[i] = i;
    }

    for (ll k = 1; k < len; k *= 2) {
        vl tmp(len), cnt(max(len + 1, 27LL), 0); // '27LL'
        should be changed unless input is only composed of
        lowercase

        for (ll i = 0; i < len; i++) {
            cnt[(i + k < len ? rank[i + k] : -1) + 1]++;
        }
        for (ll i = 1; i < cnt.size(); i++) cnt[i] += cnt[i -
        1];

        ll ptr = 0;
        for (ll i = 0; i < len; i++) {
            if (suf[i] + k < len) tmp[cnt[rank[suf[i] + k]]++]
            = suf[i];
            else tmp[ptr++] = suf[i];
        }

        fill(cnt.begin(), cnt.end(), 0);
        for (ll i = 0; i < len; i++) cnt[rank[i] + 1]++;
        for (ll i = 1; i < cnt.size(); i++) cnt[i] += cnt[i -
        1];
        for (ll i = 0; i < len; i++) suf[cnt[rank[tmp[i]]]++]
        = tmp[i];

        vl new_rank(len);
        new_rank[suf[0]] = 0;
        ll classes = 1;
        for (ll i = 1; i < len; i++) {
            bool diff = rank[suf[i]] != rank[suf[i - 1]] ||
```

```
                        (suf[i] + k < len ? rank[suf[i] + k] :
                        -1) != (suf[i - 1] + k < len ?
                        rank[suf[i - 1] + k] : -1);
            classes += diff;
            new_rank[suf[i]] = classes - 1;
        }
        rank = new_rank;
    }

    for (ll i = 0; i < len; i++) {
        suf[rank[i]] = i;
    }
};

auto build_suffix2 = [&]() { // 시간 복잡도  O(Nlog^2(N))
    for (ll i = 0; i < len; i++) {
        suf[i] = i;
        rank[i] = str[i];
    }

    for (ll k = 1; k < len; k *= 2) {
        auto cmp = [&](ll i, ll j) {
            if (rank[i] != rank[j]) return rank[i] < rank[j];
            ll ri = (i + k < len ? rank[i + k] : -1);
            ll rj = (j + k < len ? rank[j + k] : -1);
            return ri < rj;
        };

        sort(suf.begin(), suf.end(), cmp);

        vl new_rank(len);
        new_rank[suf[0]] = 0;
        for (ll i = 1; i < len; i++) {
            new_rank[suf[i]] = new_rank[suf[i - 1]] +
            cmp(suf[i - 1], suf[i]);
        }
        rank = new_rank;
    }
};

auto build_lcp = [&]() {
    ll h = 0;
    for (ll i = 0; i < len; i++) {
        if (rank[i] == 0) continue;
        ll j = suf[rank[i] - 1];
        while (i + h < len && j + h < len && str[i + h] ==
        str[j + h]) h++;
        lcp[i] = h;
        if (h > 0) h--;
    }
};

build_suffix();
build_lcp();
```

9 Misc

9.1 Barrett Reduction (Fastmod)

```
constexpr ll MOD = 1000000007;
constexpr ll IMOD = 0xFFFFFFFFFFFFFFFF / MOD + 1;
inline ll modular(ull n) {
    ull x = ull((__uint128_t(n) * IMOD) >> 64);
    unsigned int r = n - x * m;
    return m <= r ? x - 1 : x;
}
```

9.2 Permutation Cycle Decomposition

```
vector<vector<int>> decompose(vector<ll>& v) {
    // v must be a 0-index permutation  ex. {3, 2, 1, 4, 0}
    vector<vector<int>> res;
    int n = v.size();
    vector<int> visited(n);
    for (int i = 0; i < n; i++) {
        if (visited[i]) continue;
        int j = i;
        vector<int> tmp;
        do {
            tmp.push_back(j);
            visited[j] = 1;
            j = v[j];
        } while (visited[j] == 0);
        res.push_back(tmp);
    }
    return res;
}

void apply(vector<ll>& v, vector<ll>& order, ll c) {
    // v = {5, 6, 7, 8, 9}, order = {3, 2, 1, 4, 0}, c =
    1(mod6), then v = {9, 7, 6, 5, 8}
    auto res = decompose(order);
    vector<ll> tmp = v;
    for (auto a : res) {
        ll n = a.size(), t = c % n;
        for (int i = 0; i < n; i++) {
            v[order[a[(i + t) % n]]] = tmp[order[a[i]]];
        }
    }
}
```

9.3 Knuth's X

```
struct Node {
    ll size;
    ll name;
    Node* l;
    Node* r;
    Node* u;
    Node* d;
    Node* c;
};

ll N = 12, C = 72;
Node h{};
h.l = h.r = h.u = h.d = &h;
vector<Node*> o(N); // N : # of rows that can be chosen
vector<Node> cols(C); // C : # of columns

function <void(Node*)> cover = [](Node* n) {
    n->l->r = n->r;
    n->r->l = n->l;
    for (Node* i = n->d; i != n; i = i->d) {
        for (Node* j = i->r; j != i; j = j->r) {
            j->u->d = j->d;
            j->d->u = j->u;
            j->c->size --;
        }
    }
};

function <void(Node*)> uncover = [](Node* n) {
    for (Node* i = n->u; i != n; i = i->u) {
        for (Node* j = i->l; j != i; j = j->l) {
            j->u->d = j;
            j->d->u = j;
            j->c->size ++;
        }
    }
    n->l->r = n;
    n->r->l = n;
};

function <void(ll)> search = [&](ll k) {
    if (h.r == &h) {
        return; // success
    }
    Node* c = &h;
    ll mn = h.r->size + 1;
    for (Node* j = h.r; j != &h; j = j->r) {
        if (mn > j->size) {
            mn = j->size;
            c = j;
        }
    }
    if (mn == 0) return; // empty column
    cover(c);

    for (Node* r = c->d; r != c; r = r->d) {
        o[k] = r;
        for (Node* j = r->r; j != r; j = j->r) {
            cover(j->c);
        }
        search(k + 1);
```

```
        r = o[k];
        c = r->c;
        for (Node* j = r->l; j != r; j = j->l) {
            uncover(j->c);
        }
    }
    uncover(c);
};

function <void(vector<vector<ll>>, ll, ll)> init = [&h,
&cols](const vector<vector<ll>>& data, ll n, ll m) {
    ll i, j;
    for (i = 0; i < m; i++) {
        cols[i].size = 0;
        if (i == 0) cols[i].l = &h;
        else cols[i].l = &cols[i-1];
        if (i == m - 1) cols[i].r = &h;
        else cols[i].r = &cols[i+1];
        cols[i].u = &cols[i];
        cols[i].d = &cols[i];
        cols[i].c = &cols[i];
    }
    h.r = &cols[0];
    h.l = &cols[m - 1];
    for (i = 0; i < n; i++) {
        Node row{};     // not included in the real structure
        for (j = 0; j < m; j++) {
            if (data[i][j]) {
                Node* node = new Node;

                node->c = &cols[j];

                if (cols[j].size == 0) {
                    cols[j].u = node;
                    cols[j].d = node;
                    node->u = &cols[j];
                    node->d = &cols[j];
                }
                else {
                    node->u = cols[j].u;
                    node->d = &cols[j];
                    cols[j].u->d = node;
                    cols[j].u = node;
                }
                cols[j].size ++;

                if (row.size == 0) {
                    node->l = &row;
                    node->r = &row;
                    row.l = node;
                    row.r = node;
                    row.size = 1;
                }
                else {
                    node->l = row.l;
                    node->r = &row;
                    row.l->r = node;
                    row.l = node;
                }
                node->name = i;
            }
        }
        if (row.size > 0) {
            row.l->r = row.r;
            row.r->l = row.l;
        }
    }
};

/* init(data, R, C);
   where data is an R * C matrix(vector<vector<>>) consists of
   0 and 1
   search(0);*/
```

9.4 Kitamasa & Berlekamp Massey

```
const int mod = 1e9+7; ll pw(ll a, ll b){/*a^b mod M*/}
vector<int> berlekamp_massey(vector<int> x){
  int n = x.size(),L=0,m=0; ll b=1; if(!n) return {};
  vector<int> C(n), B(n), T; C[0]=B[0]=1;
  for(int i=0; ++m && i<n; i++){ ll d = x[i] % mod;
    for(int j=1; j<=L; j++) d = (d + 1LL * C[j] * x[i-j]) %
    mod;
    if(!d) continue; T=C; ll c = d * pw(b, mod-2) % mod;
    for(int j=m; j<n; j++) C[j] = (C[j] - c * B[j-m]) % mod;
    if(2 * L <= i) L = i-L+1, B = T, b = d, m = 0;
  }
  C.resize(L+1); C.erase(C.begin());
  for(auto &i : C) i = (mod - i) % mod; return C;
} // O(NK + N log mod)
int get_nth(vector<int> rec, vector<int> dp, ll n){
  int m = rec.size(); vector<int> s(m), t(m); ll ret=0;
  s[0] = 1; if(m != 1) t[1] = 1; else t[0] = rec[0];
  auto mul = [&rec](vector<int> v, vector<int> w){
    int m = v.size(); vector<int> t(2*m);
    for(int j=0; j<m; j++) for(int k=0; k<m; k++){
      t[j+k] = (t[j+k] + 1LL * v[j] * w[k]) % mod;
    }
    for(int j=2*m-1; j>=m; j--) for(int k=1; k<=m; k++){
      t[j-k] = (t[j-k] + 1LL * t[j] * rec[k-1]) % mod;
    }
    t.resize(m); return t;
  };
  for(; n; n>>=1, t=mul(t,t)) if(n & 1) s=mul(s,t);
  for(int i=0; i<m; i++) ret += 1LL * s[i] * dp[i] % mod;
  return ret % mod;
} // O(N2 log X)
int guess_nth_term(vector<int> x, ll n){
  if(n < x.size()) return x[n];
  vector<int> v = berlekamp_massey(x);
  return v.empty() ? 0 : get_nth(v, x, n);
}
struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no
duplicate please..
vector<int> get_min_poly(int n, vector<elem> M){
  // smallest poly P such that A^i = sum_{j < i} {A^j \times
  P_j}
  vector<int> rnd1, rnd2, gobs; mt19937 rng(0x14004);
  auto gen = [&rng](int lb, int ub){ return
  uniform_int_distribution<int>(lb, ub)(rng); };
  for(int i=0; i<n; i++) rnd1.push_back(gen(1, mod-1)),
  rnd2.push_back(gen(1, mod-1));
  for(int i=0; i<2*n+2; i++){ int tmp = 0;
    for(int j=0; j<n; j++) tmp = (tmp + 1LL * rnd2[j] *
    rnd1[j]) % mod;
    gobs.push_back(tmp); vector<int> nxt(n);
    for(auto &j : M) nxt[j.x] = (nxt[j.x] + 1LL * j.v *
    rnd1[j.y]) % mod;
    rnd1 = nxt;
  } auto v = berlekamp_massey(gobs);
  return vector<int>(v.rbegin(), v.rend());
}
ll det(int n, vector<elem> M){
  vector<int> rnd; mt19937 rng(0x14004);
  auto gen = [&rng](int lb, int ub){ return
  uniform_int_distribution<int>(lb, ub)(rng); };
  for(int i=0; i<n; i++) rnd.push_back(gen(1, mod-1));
  for(auto &i : M) i.v = 1LL * i.v * rnd[i.y] % mod;
  auto sol = get_min_poly(n, M)[0]; if(n % 2 == 0) sol = mod -
  sol;
  for(auto &i : rnd) sol = 1LL * sol * pw(i, mod-2) % mod;
  return sol;
}
```

9.5 Ternary Search

```
ll f(ll x, vector<pair<ll, ll>>& data) {
    ll result = 0;
    for (auto [a, b] : data) {
        result += abs(a) + abs(b - x);
        result += abs(a - x) + abs(b - x);
        result += abs(a - x) + abs(b);
        result += abs(a - x) + abs(b + x);
    }
    return result;
} //아무 함수
ll ternary_search(ll lo, ll hi, vector<pair<ll, ll>>& data) {
    while (hi - lo >= 3) {
        ll m1 = lo + (hi - lo) / 3;
        ll m2 = hi - (hi - lo) / 3;
        if (f(m1, data) >= f(m2, data)) { //최솟값 >=, 최댓값
        <=
            lo = m1;
```

```
        } else {
            hi = m2;
        }
    }

    // 마지막 2~3개 남은 구간은 직접 비교
    ll best_x = lo;
    ll best_val = f(lo, data);
    for (ll x = lo + 1; x <= hi; ++x) {
        ll val = f(x, data);
        if (val < best_val) {
            best_val = val;
            best_x = x;
        }
    }
    //best_x는 위치, best_val은 값
    return best_val;
}
```

10 Notes

10.1 Basic Number Theory

Euler's Theorem

$$\text{If } \gcd(a,n)=1, \quad \text{then } a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Fermat's Little Theorem

$$\text{If } p \text{ is prime and } p \nmid a, \quad \text{then } a^{p-1} \equiv 1 \pmod{p}.$$

Wilson's Theorem

$$\text{For an integer } p>1, \quad p \text{ is prime } \iff (p-1)! \equiv -1 \pmod{p}.$$

$$\text{Let } p \text{ be an odd prime. If } a^2 \equiv 1 \pmod{p},$$
$$\text{then } a \equiv 1 \pmod{p} \text{ or } a \equiv -1 \pmod{p}.$$

Euler phi Function

$$\text{For } n \in \mathbb{N},\ \varphi(n) = \#\{1 \le k \le n : \gcd(k,n)=1\}.$$

$$\text{If } n = p_1^{a_1} p_2^{a_2} \cdots p_r^{a_r}, \text{ then } \varphi(n) = n \prod_{p \mid n} \left(1 - \frac{1}{p}\right).$$

Quadratic Reciprocity Law

Let p, q be distinct odd primes. Define the Legendre symbol:

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

$$\text{Then: } \left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}.$$

Möbius Function

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n \text{ has a squared prime factor} \\ (-1)^k & \text{if } n \text{ is a product of } k \text{ distinct primes} \end{cases}$$

Möbius Inversion:

$$\text{If } g(n) = \sum_{d \mid n} f(d), \text{ then } f(n) = \sum_{d \mid n} \mu(d)\, g(n/d).$$

$$\sum_{d \mid n} \mu(d) = \begin{cases} 1 & \text{if } n = 1, \\ 0 & \text{otherwise.} \end{cases}$$

10.2 Basic Geometry

Sine Law

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

Cosine Law

$$a^2 = b^2 + c^2 - 2bc\cos A$$
$$a = b\cos C + c\cos B$$

Heron's Formula

$$s = \frac{a+b+c}{2}$$
$$S = \sqrt{s(s-a)(s-b)(s-c)}$$

Angle addition/subtraction Formula

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$
$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$
$$\tan(\alpha \pm \beta) = \frac{\tan\alpha \pm \tan\beta}{1 \mp \tan\alpha\tan\beta}$$
$$\sin\alpha + \sin\beta = 2\sin(\frac{\alpha+\beta}{2})\cos(\frac{\alpha-\beta}{2})$$
$$\cos\alpha + \cos\beta = 2\cos(\frac{\alpha+\beta}{2})\cos(\frac{\alpha-\beta}{2})$$

10.3 Lifting The Exponent Lemma

Let p be an odd prime, and let $x, y \in \mathbb{Z},\ n \in \mathbb{N}^+$. Suppose $p \mid x-y$ and $p \nmid x,\ p \nmid y$. Then,

$$v_p(x^n - y^n) = v_p(x-y) + v_p(n)$$

For $p=2$, if x and y are odd integers and $n \ge 1$:
If n is even and $x \equiv y \pmod 4$, then

$$v_2(x^n - y^n) = v_2(x-y) + v_2(x+y) + v_2(n) - 1$$

Here, $v_p(k)$ denotes the exponent of the prime p in the prime factorization of k.

10.4 Pick's Theorem

격자점 위 점들로 이루어진 단순 다각형에 대해, 다각형의 넓이를 A, 다각형 내부의 격자점의 수를 i, 다각형 둘레 위의 격자점의 수를 b라고 하면 다음이 성립한다.

$$A = i + \frac{b}{2} - 1$$

10.5 Catalan, Derangement

$C_n = \frac{1}{n+1}\binom{2n}{n}, C_0 = 1, C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}, C_{n+1} = \frac{2(2n+1)}{n+2} C_n$
$D_n = (n-1)(D_{n-1} + D_{n-2}) = n! \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i!}$

10.6 Special Nim Game

Subtraction Game : 한 번에 k개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 $k+1$로 나눈 나머지를 XOR 합하여 판단한다.
Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 $k+1$로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.