.

Base Julios GOD

Contents

1 Data structures

1.1 Segment tree

```
#define M ((l + r) >> 1)
#define op(x, y) (x + y)
int st[4*N], values[N];
void build(int l, int r, int i) {
    if(l == r) {
        st[i] = values[l];
        return;
    }
    build(l, M, 2*i+1);
    build(M+1, r, 2*i+2);
    st[i] = op(st[2*i+1], st[2*i+2]);  // #!
}
// Point update - range query
void update(int l, int r, int idx, int x, int i) {
    if(l == r) {
        st[i] += x;
        return;
    }
    if(idx <= M) update(l, M, idx, x, 2*i+1);
    else update(M+1, r, idx, x, 2*i+2);
    st[i] = op(st[2*i+1], st[2*i+2]);
}
int query(int l, int r, int a, int b, int i) {
    if(a > r || b < l) return 0;        // change for min/max
    if(a <= l && r <= b) return st[i];
    return op(query(l, M, a, b, 2*i+1), query(M+1, r, a, b, 2*i+2));
}
// l = 0, r = n-1, i = 0
```

1.2 Segment tree - Range update, point query

```
#define M ((l + r) >> 1)
#define op(x, y) (x + y)
int st[4*N], values[N];
void build(int l, int r, int i) {
    if(l == r) {
        st[i] = values[l];
        return;
    }
    build(l, M, 2*i+1);
    build(M+1, r, 2*i+2);
    st[i] = op(st[2*i+1], st[2*i+2]);  // #!
}
// Point update - range query
void update(int l, int r, int idx, int x, int i) {
    if(l == r) {
        st[i] += x;
        return;
    }
    if(idx <= M) update(l, M, idx, x, 2*i+1);
    else update(M+1, r, idx, x, 2*i+2);
    st[i] = op(st[2*i+1], st[2*i+2]);
}
int query(int l, int r, int a, int b, int i) {
    if(a > r || b < l) return 0;        // change for min/max
    if(a <= l && r <= b) return st[i];
    return op(query(l, M, a, b, 2*i+1), query(M+1, r, a, b, 2*i+2));
}
// l = 0, r = n-1, i = 0
```

1.3 Segment tree - Lazy propagation

```
#define M ((l + r) >> 1)
#define op(x, y) (x + y)
ll st[4*N], lazy[4*N], arr[N];
void build(int l, int r, int i) {
    lazy[i] = 0;
    if (l == r) {st[i] = arr[l]; return;}
    build(l, M, 2*i+1);
    build(M+1, r, 2*i+2);
    st[i] = op(st[2*i+1], st[2*i+2]);
}
void push(int l, int r, int i) {
    if (!lazy[i]) return;
    st[i] += (r-l+1) * lazy[i];   // change for min/max
    if (l != r) {
        lazy[2*i+1] += lazy[i];
        lazy[2*i+2] += lazy[i];
    }
    lazy[i] = 0;   // change for multiplication
}
void update(int l, int r, int a, int b, ll x, int i) {
```

```
    push(l, r, i);
    if (a > r || b < l) return;
    if (a <= l && r <= b) {
        lazy[i] += x;
        push(l, r, i);
        return;
    }
    update(l, M, a, b, x, 2*i+1);
    update(M+1, r, a, b, x, 2*i+2);
    st[i] = op(st[2*i+1], st[2*i+2]);
}
ll query(int l, int r, int a, int b, int i) {
    if (a > r || b < l) return 0;   // change for min/max
    push(l, r, i);
    if (a <= l && r <= b) return st[i];
    return op(query(l, M, a, b, 2*i+1), query(M+1, r, a, b, 2*i+2));
}  // i=0, l=0, r=n-1, x=value, a,b=range query
```

1.4 Segment tree - Persistence

```
#define M ((l + r) >> 1)
struct Node{
    Node *left, *right;
    ll val;
    Node(ll x) : left(NULL), right(NULL), val(x) {}        // Add value
    Node(Node *l, Node *r) : left(l), right(r), val(0) {    // Update
        values
        if(l) val += l->val;
        if(r) val += r->val;
    }
    Node(Node *root) : left(root->left), right(root->right), val(root->
        val) {}    // Make copy
};
Node *build(int l, int r, vector<int> &values) {
    if(l == r) return new Node(values[l]);
    return new Node(build(l, M, values), build(M+1, r, values));
}
Node *update(Node *node, int l, int r, int idx, ll k) {
    if(l == r) return new Node(k);
    if(idx <= M) return new Node(update(node->left, l, M, idx, k), node
        ->right);
    return new Node(node->left, update(node->right, M+1, r, idx, k));
}
ll query(Node *node, int l, int r, int a, int b) {
    if(a > r || b < l) return 0;
    if(a <= l && r <= b) return node->val;
    return query(node->left, l, M, a, b) + query(node->right, M+1, r, a,
         b);
}
Node *roots[N];
// 0 based indexing
// roots[copy++] = build(0, n-1);
// roots[copy++] = new Node(roots[--idx]);
// roots[--copy] = update(roots[--copy], 0, n-1, --idx, x);
// query(roots[--copy], 0, n-1, --a, --b)
```

1.5 Fenwick tree

```
struct FT{
    int n;
    vector<int> ft;
    FT(int _n) : n(_n), ft(_n+1) {}
    void add(int idx, int k) {
        for(; idx<=n; idx+=idx&-idx)
            ft[idx] += k;
    }
    int query(int idx) {
        int sum = 0;
        for(; idx>0; idx-=idx&-idx)
            sum += ft[idx];
        return sum;
    }
    int query(int l, int r) {
        return query(r) - query(l-1);
    }
    int lower_bound(int k) { // LOG = log2(n) + 1
        int sum = 0, idx = 0;
        for(int i=LOG-1; i>=0; i--) {
            if(idx + (1 << i) <= n && sum + ft[idx + (1 << i)] < k) {
                sum += ft[idx + (1 << i)];
                idx += 1 << i;
            }
        }
        return idx + 1;
    }
}; // 1-based indexing
```

1.6 DSU

```
struct DSU{
    int n;
    vector<int> parent, rank;
    DSU(int _n) : n(_n), parent(_n), rank(_n) {
        for(int i=0; i<n; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }
    int find_set(int v) {
        if (v == parent[v]) return v;
        return parent[v] = find_set(parent[v]);
    }
    void union_sets(int a, int b) {
        a = find_set(a), b = find_set(b);
        if (a != b) {
            if (rank[a] < rank[b]) swap(a, b);
            parent[b] = a;
            if (rank[a] == rank[b]) rank[a]++;
        }
    }
    int components() {
        int cnt = 0;
        for(int i=0; i<n; i++) {
            if(find_set(i) == i) cnt++;
        }
        return cnt;
    }
};
```

1.7 SQRT decomposition

```
struct SQ{
    int n, b;
    vector<int> values, blocks;
    SQ(int _n) : n(_n), values(_n) {
        b = (int)sqrt(n) + 1;
        blocks = vector<int> (b);
    }
    // Basic update / query
    void operation(int l, int r, int k) {
        int bl = l/b, br = r/b;
        // operation lies in same block
        if(bl == br) {
            for(int i=l; i<=r; i++) {
                blocks[bl] -= values[i];
                values[i] += k;
                blocks[bl] += values[i];
            }
        }
        // operation on different blocks
        else {
            for(int i=l; i<(bl+1)*b; i++) {
                blocks[bl] -= values[i];
                values[i] += k;
                blocks[bl] += values[i];
            }
            for(int i=br*b; i<=r; i++) {
                blocks[br] -= values[i];
                values[i] += k;
                blocks[br] += values[i];
            }
            for(int i=(bl+1)*b; i<br; i++) {
                blocks[i] += k * b;
            }
        }
    }
};
```

1.8 Trie

```
struct Node{
    vector<Node*> ocu;
    bool flag;
    Node() : ocu(26), flag(false) {}
};
struct Trie{
    Node *root;
    Trie() : root(new Node()) {}
    void insert(string word) {
        Node *curr = root;
        for(auto &it : word) {
            if(!curr -> ocu[it - 'a'])
                curr -> ocu[it - 'a'] = new Node();
```

```
            curr = curr -> ocu[it - 'a'];
        }
        curr -> flag = true;
    }
    bool search(string word) {
        Node *curr = root;
        for(auto &it : word) {
            if(!curr -> ocu[it - 'a']) return false;
            curr = curr -> ocu[it - 'a'];
        }
        return curr -> flag;
    }
};
```

1.9 Trie XOR

```
struct Node{
    vector<Node*> ocu;
    Node() : ocu(2) {}
};
struct Trie {
    Node *root;
    Trie() : root(new Node()) { insert(0); }

    void insert(long long x) {
        Node *curr = root;
        for(int mask=63; mask>=0; mask--) {
            bool currBit = (x >> mask) & 1;
            if(!curr -> ocu[currBit])
                curr -> ocu[currBit] = new Node();
            curr = curr -> ocu[currBit];
        }
    }
    long long query(long long prefix) {
        Node *curr = root;
        long long res = 0;
        for(int mask=63; mask>=0; mask--) {
            bool currBit = (prefix >> mask) & 1;
            if(curr -> ocu[currBit ^ 1]) {
                res |= (1LL << mask);
                curr = curr -> ocu[currBit ^ 1];
            }
            else curr = curr -> ocu[currBit];
        }
        return res;
    }
};
```

1.10 Map custom hash

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

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now()
            .time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
gp_hash_table<int, int, custom_hash> freq;
```

1.11 Ordered set

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;
// find_by_order(i) -> iterator to ith element
// order_of_key(k) -> position (int) of lower_bound of k
```

1.12 Merge Sort Tree

```
#define M ((l + r) >> 1)
struct ST{
    int n;
    vector<vector<int>> st;
    ST(int _n) : n(_n), st(4*_n) {}

    void build(int l, int r, int i, vector<int> &values) {
```

```
        if(l == r) {
            st[i].pb(values[l]);
            return;
        }
        build(l, M, 2*i+1, values);
        build(M+1, r, 2*i+2, values);
        st[i].resize(st[2*i+1].size() + st[2*i+2].size());
        merge(st[2*i+1].begin(), st[2*i+1].end(), st[2*i+2].begin(), st
            [2*i+2].end(), st[i].begin());
    }
    int query(int l, int r, int a, int b, int k, int i) {
        if(a > r || b < l) return 0;
        if(a <= l && r <= b) return st[i].end() - upper_bound(st[i].
            begin(), st[i].end(), k);
        return query(l, M, a, b, k, 2*i+1) + query(M+1, r, a, b, k, 2*i
            +2);
    }
};
```

1.13 Wavelet Tree

```
// indexed in 1
// from pointer to first element and to to end
// x and y The minimun element and y the max element
// If you need only one function or more erase the others
// If you need tu construct other function you only required to
    undertand the limit, this
// are the same
struct wavelet_tree{
  int lo, hi;
  wavelet_tree *l, *r;
  vector<int> b;
  wavelet_tree(int *from, int *to, int x, int y){
    lo = x, hi = y;
    if(lo == hi or from >= to) return;
    int mid = (lo+hi)/2;
    auto f = [mid](int x){ return x <= mid;};
    b.reserve(to-from+1);
    b.pb(0);
    for(auto it = from; it != to; it++)
      b.push_back(b.back() + f(*it));
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree(from, pivot, lo, mid);
    r = new wavelet_tree(pivot, to, mid+1, hi);
  }
  //kth smallest element in [l, r]
  int kth(int l, int r, int k){
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l-1];
    int lb = b[l-1];
    int rb = b[r];
    if(k <= inLeft) return this->l->kth(lb+1, rb , k);
    return this->r->kth(l-lb, r-rb, k-inLeft);
  }
  //count of nos in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l-1], rb = b[r];
    return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
  }
  //count of nos in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r or k < lo or k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
    if(k <= mid) return this->l->count(lb+1, rb, k);
    return this->r->count(l-lb, r-rb, k);
  }
};
```

1.14 Monotonic stack

```
int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    int n; cin >> n;
    vector<int> v(n), l(n), r(n);
    for(auto &it : v) cin >> it;

    stack<pair<int, int>> st;
    for(int i=n-1; i>=0; i--) {
        while(!st.empty() && v[i] < st.top().first) {
            l[st.top().second] = i;
            st.pop();
        }
```

```
        st.push({v[i], i});
    }

    while(!st.empty()) l[st.top().second] = -1, st.pop();

    rep(i,0,n) {
        while(!st.empty() && v[i] <= st.top().first) {
            r[st.top().second] = i;
            st.pop();
        }
        st.push({v[i], i});
    }

    while(!st.empty()) r[st.top().second] = n, st.pop();

    vector<int> res(n);
    rep(i,0,n) {
        int curr = r[i] - l[i] - 1;
        res[curr-1] = max(res[curr-1], v[i]);
    }
    for(int i=n-2; i>=0; i--) {
        res[i] = max(res[i], res[i+1]);
    }

    for(auto &it : res) cout << it << "␣";
    cout << endl;

    return 0;
}
```

1.15 Matrix operations

```
struct Matrix {
  ll a[N][N];
  Matrix() {memset(a,0,sizeof(a));}
  Matrix operator *(Matrix other) {  // Product of a matrix
    Matrix product=Matrix();
        rep(i,0,N) rep(j,0,N) rep(k,0,N) {
            product.a[i][k]+=a[i][j]*other.a[j][k];
            product.a[i][k]%=MOD;
        }
    return product;
  }
};
Matrix expo_power(Matrix a, ll n) {  // Matrix exponentiation
  Matrix res=Matrix();
    rep(i,0,N) res.a[i][i]=1;  // Matriz identidad
  while(n){  // binpow
        if(n & 1) res = res * a;
        n >>= 1;
        a = a * a;
  }
  return res;
} // Ej. Matrix M=Matrix();  M.a[0][0]=1;  M=M*M;   Matrix res=
    expo_power(M,k);
```

1.16 Heavy Light Decomposition

```
int sz[N], in[N], out[N], timer, head[N], parent[N];
void dfs_sz(int u = 0, int p = -1) {
    sz[u] = 1;
    parent[u] = p;
    for(auto &v : adj[u]) {
        if(v == p) continue;
        dfs_sz(v, u);
        sz[u] += sz[v];
        if(adj[u][0] == p || sz[v] > sz[adj[u][0]]) {
            swap(v, adj[u][0]);
        }
    }
}
void dfs_hld(int u = 0, int p = -1) {
    in[u] = timer++;
    for(auto &v : adj[u]) {
        if(v == p) continue;
        head[v] = (adj[u][0] == v ? head[u] : v);
        dfs_hld(v, u);
    }
    out[u] = timer;
}
ll query_path(int u, int v) {
    ll res = 0;
    while(head[u] != head[v]) {
        if(in[u] < in[v]) swap(u, v);
        res += query(0, timer-1, in[head[u]], in[u], 0);
        u = parent[head[u]];
```

```
    }
    if(in[u] > in[v]) swap(u, v);
    res += query(0, timer-1, in[u]+1, in[v], 0);
    // in[u] (node)  / in[u]+1 (edge)
    return res;
}
```

2 Graphs

2.1 Euler tour

```
vector<int> in(n+1), out(n+1), euler(n+1);
int timer = 0;
auto dfs = [&](auto self, int u, int p = -1) -> void {
    in[u] = ++timer;
    euler[timer] = u;
    for(auto &v : adj[u]) {
        if(v == p) continue;
        self(self, v, u);
    }
    out[u] = timer;
};
dfs(dfs, 0);
queries[i] = {in[u], out[u], k, i};
int x = values[euler[idx]];
```

2.2 Find bridges

```
int in[N], low[N], timer;
/* Articulation bridges */
void dfs(int u, int p = -1) {
    in[u] = low[u] = ++timer;
    for(auto &v : adj[u]) {
        if(v == p) continue;
        if(!in[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] > in[u]) IS_BRIDGE(U, V);
        }
        else low[u] = min(low[u], in[v]);
    }
}

/* Bridge tree */
// Use dfs to find bridges, but change
if(low[v] <= in[u]) dsu.union_sets(u, v);

// Iterate over the edges
for(auto [u, v] : edges) {
    u = dsu.find_set(u);
    v = dsu.find_set(v);
    if(u != v) {
        bridgeTree[u].pb(v);
        bridgeTree[v].pb(u);
    }
}

/* Articulation points */
// Use same code as finding bridges
if(low[v] >= in[u]) IS_ARTICULATION_POINT(U, V);
```

2.3 Dijkstra

```
void dijkstra(int start, int n) {
    priority_queue<pair<ll, int>, vector<pair<ll,int>>, greater<pair<ll,
        int>>> pq;
    for(int i=0; i<n; i++) {
        dist[i] = 1e14;
    }

    dist[start] = 0;
    pq.push({0, start});

    while(!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();

        if(d > dist[u]) continue;

        for(auto &[v, w] : adj[u]) {
            if(dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
```

```
}
```

2.4 Binary Lifting LCA

```
int depth[N], up[N][LOG];
void dfs1(int u, int p) {
    up[u][0] = p;
    rep(i,1,LOG) {
        if(up[u][i-1] != -1)
            up[u][i] = up[up[u][i-1]][i-1];
        else
            up[u][i] = -1;
    }
    for(auto &v : adj[u]) {
        if(v == p) continue;
        depth[v] = depth[u] + 1;
        dfs1(v, u);
    }
}
int lift(int u, int k) {
    for(int i=LOG-1; i>=0; i--) {
        if(k & (1 << i)) {
            u = up[u][i];
            if(u == -1) return 0;
        }
    }
    return u;
}
int lca(int u, int v) {
    if(depth[u] < depth[v]) swap(u, v);
    u = lift(u, depth[u] - depth[v]);
    if(u == v) return u;

    for(int i=LOG-1; i>=0; i--) {
        if(up[u][i] != -1 && up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0] == -1 ? 0 : up[u][0];
}
```

2.5 Centroid decomposition

```
void dfs(int u, int p) {
    sz[u] = 1;
    for(auto &v : adj[u]) {
        if(v == p || centroid[v]) continue;
        dfs(v, u);
        sz[u] += sz[v];
    }
}
int getCentroid(int u, int p, int n) {
    for(auto &v : adj[u]) {
        if(v == p || centroid[v]) continue;
        if(2 * sz[v] > n) return getCentroid(v, u, n);
    }
    return u;
}
void getCount(int u, int p, int depth) {
    if(depth > k) return;
    res += 0ll + cnt[k - depth];
    for(auto &v : adj[u]) {
        if(v == p || centroid[v]) continue;
        getCount(v, u, depth+1);
    }
}
void getPaths(int u, int p, int depth, int x) {
    if(depth > k) return;
    cnt[depth] += x;
    for(auto &v : adj[u]) {
        if(v == p || centroid[v]) continue;
        getPaths(v, u, depth+1, x);
    }
}
void decompose(int u, int p) {
    dfs(u, p);
    int c = getCentroid(u, p, sz[u]);
    centroid[c] = 1;
    cnt[0] = 1;

    for(auto &v : adj[c]) {
        if(centroid[v]) continue;
        getCount(v, c, 1);
        getPaths(v, c, 1, 1);
    }

```

```
    getPaths(c, p, 0, -1);

    for(auto &v : adj[c]) {
        if(v == c || centroid[v]) continue;
        decompose(v, c);
    }
}
```

2.6 Edmonds karp

```
const int N = 1e3+5;
vector<int> adj[N];
int parent[N], cap[N][N];

void add_edge(int u, int v, int c) {
    adj[u].pb(v);
    adj[v].pb(u);
    cap[u][v] += c;
}
int bfs(int s, int t) {
    memset(parent, -1, sizeof(parent));
    parent[s] = -2;
    queue<pair<int, int>> q;
    q.push({s, (int)1e9});

    while(!q.empty()) {
        auto [u, f] = q.front();
        q.pop();

        for(auto &v : adj[u]) {
            if(parent[v] != -1 || !cap[u][v]) continue;

            parent[v] = u;
            int flow = min(cap[u][v], f);
            if(v == t) return flow;

            q.push({v, flow});
        }
    }
    return 0;
}
int maxFlow(int s, int t) {
    int mx = 0, flow;
    while(flow = bfs(s, t)) {
        mx += flow;
        int curr = t;
        while(curr != s) {
            int prev = parent[curr];
            cap[prev][curr] -= flow;
            cap[curr][prev] += flow;
            curr = prev;
        }
    }
    return mx;
}
```

2.7 Dinics

```
struct Dinic {
    int nodes, src, dst;
    vector<int> dist, q, work;
    struct edge{int to, rev; ll f, cap;};
    vector<vector<edge>> g;
    Dinic(int x) : nodes(x), g(x), dist(x), q(x), work(x){}
    void add_edge(int s, int t, ll cap) {
        g[s].pb((edge) {t, SZ(g[t]), 0, cap});
        g[t].pb((edge) {s, SZ(g[s])-1, 0, 0});
    }
    bool dinic_bfs() {
        fill(all(dist), -1); dist[src]=0;
        int qt=0; q[qt++]=src;
        for(int qh=0; qh<qt; qh++) {
            int u=q[qh];
            rep(i, 0, SZ(g[u])) {
                edge &e = g[u][i]; int v=g[u][i].to;
                if(dist[v]<0&&e.f<e.cap) dist[v]=dist[u]+1, q[qt++]=v;
            }
        }
        return dist[dst]>=0;
    }
    ll dinic_dfs(int u, ll f) {
        if(u==dst) return f;
        for(int &i=work[u]; i<SZ(g[u]); i++) {
            edge &e=g[u][i];
            if(e.cap <= e.f) continue;
            int v=e.to;
```

```
            if(dist[v]==dist[u]+1) {
                ll df=dinic_dfs(v, min(f, e.cap-e.f));
                if(df > 0) {e.f+=df; g[v][e.rev].f-=df; return df;}
            }
        }
        return 0;
    }
    ll max_flow(int _src, int _dst) {
        src=_src, dst=_dst;
        ll result=0;
        while(dinic_bfs()) {
            fill(all(work), 0);
            while(ll delta=dinic_dfs(src, 1e12)) result+=delta;
        }
        return result;
    }
};

int main() {io
    int n,m,sl,el,s,t,u,v;
    cin>>n>>sl>>el;
    s=n+n, t=s+1;
    Dinic nf(n+n+2);
    rep(i,0,sl){
        cin>>u; u--;
        nf.add_edge(s,u,1);
    }
    rep(i,0,el){
        cin>>u; u--;
        nf.add_edge(u+n,t,1);
    }
    rep(i,0,n) nf.add_edge(i,i+n,1);
    cin>>m;
    rep(i,0,m){
        int u,v;
        cin>>u>>v;
        u--,v--;
        nf.add_edge(u+n,v,1);
    }
  cout<<nf.max_flow(s,t)<<endl;
    return 0;
}
```

2.8 Bipartite check

```
int color[N];
bool f = 1;
// memset(color, -1, sizeof(color));
bool dfs(int u = 0, int c = 0) {
    color[u] = c;
    for(auto &v : adj[u]) {
        if(color[v] == -1 && !dfs(v, 1-c)) return false;
        else if(color[v] == c) return false;
    }
}
```

2.9 Has cycle?

```
int color[N];
// directed graph
bool dfs(int u, int p = -1) {
    color[u] = 1;
    for(auto &v : adj[u]) {
        if(!color[v]) if(!dfs(v, u)) return false;
        if(color[v] == 1) return false;
    }
    color[u] = 2;
    return true;
}

// undirected graph
bool dfs(int u, int p = -1) {
    color[u] = 1;
    parent[u] = p;
    for(auto &v : adj[u]) {
        if(v == p) continue;
        if(!color[v]) {
            if(dfs(v, u)) return true;
        }
        else {
            t = u, s = v;
            return true;
        }
    }
    return false;
}
```

2.10 Kuhn - matching

```
vector<int> g[MAXN]; // [0,n)->[0,m)
int n,m;
int mat[MAXM];bool vis[MAXN];
int match(int x){
  if(vis[x])return 0;
  vis[x]=true;
  for(int y:g[x]) if(mat[y]<0||match(mat[y])) {mat[y]=x;return 1;}
  return 0;
}
vector<pair<int,int> > max_matching(){
  vector<pair<int,int> > r;
  memset(mat,-1,sizeof(mat));
  rep(i,0,n) memset(vis,false,sizeof(vis)),match(i);
  rep(i,0,m) if(mat[i]>=0)r.pb({mat[i],i});
  return r;
}
```

2.11 Hopcroft Karp - matching

```
vector<int> g[MAXN]; // [0,n)->[0,m)
int n,m;
int mt[MAXN],mt2[MAXN],ds[MAXN];
bool bfs(){
  queue<int> q;
  memset(ds,-1,sizeof(ds));
  rep(i,0,n)if(mt2[i]<0)ds[i]=0,q.push(i);
  bool r=false;
  while(!q.empty()){
    int x=q.front();q.pop();
    for(int y:g[x]){
      if(mt[y]>=0&&ds[mt[y]]<0)ds[mt[y]]=ds[x]+1,q.push(mt[y]);
      else if(mt[y]<0)r=true;
    }
  }
  return r;
}
bool dfs(int x){
  for(int y:g[x])if(mt[y]<0||ds[mt[y]]==ds[x]+1&&dfs(mt[y])){
    mt[y]=x;mt2[x]=y;
    return true;
  }
  ds[x]=1<<30;
  return false;
}
int mm(){
  int r=0;
  memset(mt,-1,sizeof(mt));memset(mt2,-1,sizeof(mt2));
  while(bfs()){
    rep(i,0,n)if(mt2[i]<0)r+=dfs(i);
  }
  return r;
}
```

2.12 Min cost - max flow

```
typedef ll tf;
typedef ll tc;
const tf INFFLOW=1e9;
const tc INFCOST=1e9;
struct MCF{
  int n;
  vector<tc> prio, pot; vector<tf> curflow; vector<int> prevedge,
      prevnode;
  priority_queue<pair<tc, int>, vector<pair<tc, int>>, greater<pair<tc,
      int>>> q;
  struct edge{int to, rev; tf f, cap; tc cost;};
  vector<vector<edge>> g;
  MCF(int n):n(n),prio(n),curflow(n),prevedge(n),prevnode(n),pot(n),g(n)
      {}
  void add_edge(int s, int t, tf cap, tc cost) {
    g[s].pb((edge){t,SZ(g[t]),0,cap,cost});
    g[t].pb((edge){s,SZ(g[s])-1,0,0,-cost});
  }
  pair<tf,tc> get_flow(int s, int t) {
    tf flow=0; tc flowcost=0;
    while(1){
      q.push({0, s});
      fill(ALL(prio),INFCOST);
      prio[s]=0; curflow[s]=INFFLOW;
      while(!q.empty()) {
        auto cur=q.top();
        tc d=cur.fst;
        int u=cur.snd;
        q.pop();
```

```
        if(d!=prio[u]) continue;
        for(int i=0; i<SZ(g[u]); ++i) {
          edge &e=g[u][i];
          int v=e.to;
          if(e.cap<=e.f) continue;
          tc nprio=prio[u]+e.cost+pot[u]-pot[v];
          if(prio[v]>nprio) {
            prio[v]=nprio;
            q.push({nprio, v});
            prevnode[v]=u; prevedge[v]=i;
            curflow[v]=min(curflow[u], e.cap-e.f);
          }
        }
      }
      if(prio[t]==INFCOST) break;
      rep(i,0,n) pot[i]+=prio[i];
      tf df=min(curflow[t], INFFLOW-flow);
      flow+=df;
      for(int v=t; v!=s; v=prevnode[v]) {
        edge &e=g[prevnode[v]][prevedge[v]];
        e.f+=df; g[v][e.rev].f-=df;
        flowcost+=df*e.cost;
      }
    }
    return {flow,flowcost};
  }
};
```

2.13 2sat

```
struct two_sat {
    int n;
    vector<vector<int>> g, gr;
    vector<int> comp, topological_order, answer;
    vector<bool> vis;

    two_sat(int _n) {
        n = _n;
        g.assign(2 * n, vector<int>());
        gr.assign(2 * n, vector<int>());
        comp.resize(2 * n);
        vis.resize(2 * n);
        answer.resize(2 * n);
    }

    void add_edge(int u, int v) {
        g[u].pb(v);
        gr[v].pb(u);
    }

    void dfs(int u) {
        vis[u] = true;
        each(v, g[u]) if (!vis[v]) dfs(v);
        topological_order.push_back(u);
    }

    void scc(int u, int id) {
        vis[u] = true, comp[u] = id;
        each(v, gr[u]) if (!vis[v]) scc(v, id);
    }

    bool satisfiable() {
        fill(vis.begin(), vis.end(), false);
        for (int i = 0; i < 2 * n; i++) if (!vis[i]) dfs(i);
        fill(vis.begin(), vis.end(), false);
        reverse(topological_order.begin(), topological_order.end());
        int id = 0;
        for(const auto &v : topological_order) if (!vis[v]) scc(v, id++)
            ;
        for(int i=0; i<n; i++) {
            if (comp[i] == comp[i + n]) return false;
            answer[i] = (comp[i] > comp[i + n] ? 1 : 0);
        }
        return true;
    }

    // Conditions
    void add_clause(int a, int b, string op, int c) {
        if(op=="=") {
            if(c==0) add_nor(a, b);
            else if(c==1) add_01_10(a, b);
            else add_and(a, b);
        }
        else if(op=="!=") {
            if(c==0) add_or(a, b);
            else if(c==1) add_same(a, b);
```

```
            else add_nand(a, b);
        }
        else if(op=="<") {
            if(c==0) {
                cout<<"No"<<endl;
                exit(0);
            }
            else if(c==1) add_nor(a, b);
            else add_nand(a, b);
        }
        else if(op==">") {
            if(c==0) add_or(a, b);
            else if(c==1) add_and(a, b);
            else {
                cout<<"No"<<endl;
                exit(0);
            }
        }
        else if(op=="<=") {
            if(c==0) add_nor(a, b);
            else if(c==1) add_nand(a, b);
            else return;
        }
        else {
            if(c==0) return;
            else if(c==1) add_or(a, b);
            else add_and(a, b);
        }
    }

    void add_nor(int a, int b) {
        add_edge(a, a+n);
        add_edge(b, b+n);
    }
    void add_01_10(int a, int b){
        add_or(a, b);
        add_nand(a, b);
    }
    void add_and(int a, int b) {
        add_edge(a+n, a);
        add_edge(b+n, b);
    }
    void add_or(int a, int b){
        add_edge(a+n, b);
        add_edge(b+n, a);
    }
    void add_same(int a, int b) {
        add_edge(a, b);
        add_edge(b+n, a+n);
        add_edge(a+n, b+n);
        add_edge(b, a);
    }
    void add_nand(int a, int b) {
        add_edge(a, b+n);
        add_edge(b, a+n);
    }
};

int main() { io
    int n, m; cin>>n>>m;
    two_sat ts(n);
    rep(i,0,m) {
        int a, b, c; string op;
        cin>>a>>b>>op>>c;
        ts.add_clause(a, b, op, c);
    }
    cout<< (ts.satisfiable() ? "Yes" : "No")<<endl;
    return 0;
}
```

3 Math

3.1 Binpow

```
ll binpow(ll a, ll b, ll m) {
    a %= m;
    ll res = 1;
    while (b > 0) {
        if (b & 1) res = res * a % m;
        a = a * a % m;
        b >>= 1;
    }
    return res;
}
```

3.2 Modular inverse

```
tuple<int, int, int> extendedGCD(int mod, int a) {
  if (!a) return {mod, 1, 0};
  auto[r, x, y] = extendedGCD(a, mod % a);
  return {r, y, x - mod / a * y};
}
int modInverse(int mod, int a) {
    auto[r, x, y] = extendedGCD(mod, a);
    if(r != 1) {
        return -1;
    }
    return ((y < 0) ? y + mod : y);
}
```

3.3 Linear Sieve

```
const int N = 1e7+5;  // N: range to get primes
int p[N];
vector<int> pr;
void linearSieve() {
    for(int i=2; i<N; i++) {
        if(!p[i]) {
            p[i] = i;
            pr.push_back(i);
        }
        for(int j=0; i*pr[j] < N; j++) {
            p[i*pr[j]] = pr[j];
            if(pr[j] == p[i])
                break;
        }
    }
}
```

3.4 Factorials

```
void pre() {
    fact[0] = 1;
    for(int i=1; i<N; i++) {
        fact[i] = 1LL * fact[i-1] * i % MOD;
    }

    ifact[N-1] = binpow(fact[N-1], MOD-2, MOD);
    for(int i=N-2; i>=0; i--) {
        ifact[i] = 1LL * ifact[i+1] * (i+1) % MOD;
    }
}
```

3.5 Prime factorization

```
set<int> factors;
for(int i=2; i*i<=n; i++) {
    while(!(n % i)) {
        factors.insert(i);
        n /= p;
    }
}
if(n > 1) factors.insert(n);
```

3.6 Divisors

```
vector<int> div;
for(int i=1; i*i<=n; i++) {
    if(!(n % i)) {
        div.pb(d);
        if(d != n/d) div.pb(n / d);
    }
}
sort(all(div));
```

3.7 Identities

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$
$$C_n = \frac{1}{n+1}\binom{2n}{n}$$
$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$
$$\sigma(n) = O(\log(\log(n))) \text{ (number of divisors of } n)$$
$$F_{2n+1} = F_n^2 + F_{n+1}^2$$
$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$
$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$
$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_i F_j$$
(Fermat's little theorem) $a^p \, mod(p) = a \quad a^{p-1} \, mod(p) = 1 \quad a^{p-2} \, mod(p) = a^{-1}$
(Möbius Inv. Formula) Let $g(n) = \sum_{d|n} f(d)$, then $f(n) = \sum d \mid n g(d)\mu\left(\frac{n}{d}\right)$.

4 Strings

4.1 Rolling Hash

```
#define rep(i,a,b) for(int i=a;i<b;i++)
using ll = long long;
```

```

const int base = 251, MOD = 1e9+7;
vector<int> rolling_hash(string text, string pattern) {
    int n = text.length(), m = pattern.length();
    vector<ll> hashes(n+1), p(n+1);

    p[0] = 1;
    rep(i,1,n+1) p[i] = p[i-1] * base % MOD;

    rep(i,1,n+1) hashes[i] = (hashes[i-1] + (text[i-1]-'a'+1) * p[i-1] %
         MOD) % MOD;

    ll patternHash = 0;
    rep(i,1,m+1) patternHash = (patternHash + (pattern[i-1]-'a'+1) * p[i
        -1] % MOD) % MOD;

    vector<int> ocu;
    rep(i,0,n-m+1) {
        ll curr = (hashes[i+m] - hashes[i] + MOD) % MOD;
        if(curr == patternHash * p[i] % MOD) ocu.push_back(i);
    }
    return ocu;
}

// 0. Define 2 prime numbers: base and mod
// 1. Precompute powers of base
// 2. Compute prefix hashes of text
// 3. Compute hash of pattern
// 4. Sliding window of pattern size over the text to find matches
```

5 Geometry

5.1 Convex Hull

```
const double eps = 1e-9;
bool leq(double a, double b){return b-a >= -eps;}
bool le(double a, double b){return b-a > eps;}
bool eq(double a, double b){return fabs(a-b) <= eps;}

struct point{
    double x, y;
    int idx = -1;
    point(): x(0), y(0){}
    point(double x, double y): x(x), y(y){}
    point operator-(const point &p) const{return point(x - p.x, y - p.y)
        ;}
    point operator*(const int &k) const{return point(x * k, y * k);}
    bool operator<(const point &p) const{return le(x, p.x) || (eq(x, p.x
        )&& le(y, p.y));}
    bool operator==(const point &p) const{return eq(x, p.x) && eq(y, p.y
        );}
    double cross(const point &p) const{return x * p.y - y * p.x;}
};

istream &operator>>(istream &is, point &p){return is >> p.x >> p.y;}
ostream &operator<<(ostream &os, const point &p){return os << "(" << p.x
     << ",␣" << p.y << ")";}

vector<point> convexHull(vector<point> P) {
    sort(P.begin(), P.end());
    vector<point> L, U;
    for (int i = 0; i < P.size(); i++) {
        while (L.size() >= 2 && le((L[L.size() - 2] - P[i]).cross(L[L.
            size() - 1] - P[i]), 0)) {
            L.pop_back();
        }
        L.push_back(P[i]);
    }
    for (int i = P.size() - 1; i >= 0; i--) {
        while (U.size() >= 2 && le((U[U.size() - 2] - P[i]).cross(U[U.
            size() - 1] - P[i]), 0)) {
            U.pop_back();
        }
        U.push_back(P[i]);
    }
    L.pop_back();
    U.pop_back();
    L.insert(L.end(), U.begin(), U.end());
    return L;
}

bool pointInConvexHull(const vector<point> &poly, point p) {
    int n = poly.size();
    if(n < 3) return false;
    rep(i,0,n) {
        point a = poly[i], b = poly[(i+1) % n];
```

```
        double cp = (b - a).cross(p - a);
        if(!le(0, cp)) return false;
    }
    return true;
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    int n;
    double h;
    cin >> n >> h;

    vector<point> islands(n);
    vector<double> time(n), heights(n);
    rep(i,0,n) {
        cin >> islands[i] >> heights[i];
        islands[i].idx = i;
        time[i] = max(0.0, h - heights[i]);
    }

    double l = 0.0, r = *max_element(all(time));
    point c;
    cin >> c;
    auto valid = [&](double t) {
        vector<point> aux;
        rep(i,0,n) {
            if(t >= time[i]) aux.pb(islands[i]);
        }
        if(aux.size() < 3) return false;
        aux = convexHull(aux);
        return pointInConvexHull(aux, c);
    };

    if(!valid(r)) {
        cout << -1 << endl;
        return 0;
    }

    for(int it=0; it<100; it++) {
        double m = (l + r) / 2.0;
        if(valid(m)) {
            r = m;
        }
        else {
            l = m;
        }
    }
    cout << fixed << setprecision(12) << r << endl;

    return 0;
}
```

6 Techniques

6.1 MO's algorithm

```
const int N = 3e5+5, B = 550;
int values[N], freq[N], res[N], ans, has[B];
void add(int idx) {
    has[values[idx]/B]++;
    freq[values[idx]]++;
}
void remove(int idx) {
    has[values[idx]/B]--;
    freq[values[idx]]--;
}
int getAns(int k) {
    rep(i,0,B) if(has[i] > k) rep(j,i*B,(i+1)*B)
        if(freq[j] > k) return j;
    return -1;
}
void mos(vector<array<int, 4>> &queries) {
    sort(all(queries), [](array<int, 4> &a, array<int, 4> &b) {
        if(a[0]/B != b[0]/B) return a[0] < b[0];
        return (a[0]/B & 1 ? a[1] < b[1] : a[1] > b[1]);
    });

    int l = 0, r = -1;
    for(auto &[ql, qr, k, idx] : queries) {
        while(l > ql) add(--l);
        while(r < qr) add(++r);
        while(l < ql) remove(l++);
        while(r > qr) remove(r--);
        res[idx] = getAns(k);
    }
}
```

6.2 Parallel binary search

```
void parallel_bs() {
    vector<array<int, 2>> queries(n+1);
    rep(i,1,n+1) queries[i] = {1, q};

    bool f = 1;
    while(f) {
        f = 0;
        vector<vector<int>> mids(m+2);
        for(auto &it : queries) {
            if(it.l <= it.r) {
                int mid = (it.l + it.r) >> 1;
                mids[mid].pb(it.idx);
                f = 1;
            }
        }

        obj.reset();

        rep(mid,1,q+1) {
            auto &[l, r, k] = values[mid];
            obj.update(l, r, k);

            for(auto &idx : mids[mid]) {
                Queries &q = queries[idx];
                if(ok) q.r = mid - 1;
                else q.l = mid + 1;
            }
        }
    }
}
```

6.3 Split objects into light and heavy

```
vector<pair<int, int>> adj[N];
bitset<N> heavy;
const int B = 2050;
vector<pair<int, int>> heavyVertices[N];
int values[N], deg[N];

void solve() {
    ll res = 0;
    auto dfs = [&](auto self, int u, int p = -1) -> void {
        for(auto &[v, x] : adj[u]) {
            if(v == p) continue;
            if(values[u] != values[v]) res += 1ll * x;
            deg[u]++, deg[v]++;
            self(self, v, u);
        }
    };
    dfs(dfs, 0);

    rep(i,0,n) if(deg[i] >= B) heavy[i] = 1;

    rep(u,0,n) {
        if(heavy[u]) {
            for(auto &[v, x] : adj[u]) {
                colors[u][values[v]] += 1ll * x;
                if(heavy[v])
                    heavyVertices[u].pb({v, x});
                    heavyVertices[v].pb({u, x});
            }
        }
    }

    while(q--) {
        int u, c;
        cin >> u >> c;
        u--;

        if(heavy[u]) {
            res += 1ll * colors[u][values[u]];
            res -= 1ll * colors[u][c];
            for(auto &[v, x] : heavyVertices[u]) {
                colors[v][values[u]] -= 1ll * x;
                colors[v][c] += 1ll * x;
            }
            values[u] = c;
        }
        else {
            for(auto &[v, x] : adj[u]) {
                if(values[u] != values[v]) {
                    if(c == values[v]) res -= 1ll * x;
                }
                else {
```

```
                    if(c != values[v]) {
                        res += 1ll * x;
                    }
                }
                if(heavy[v]) {
                    colors[v][values[u]] -= 1ll * x;
                    colors[v][c] += 1ll * x;
                }
            }
            values[u] = c;
        }

        cout << res << endl;
    }
}
```

6.4 Meet in the middle

```
int main() {
    ios::sync_with_stdio(0); cout.tie(0); cin.tie(0);
    int n; cin >> n;
    vector<long long> a(n), b(n);
    for(auto &it : a) cin >> it;
    for(auto &it : b) cin >> it;

    auto subsets = [](vector<long long> &a, vector<long long> &b, bool f
        ){
        int n = a.size();
        vector<long long> aux;
        for(int mask=0; mask<(1<<n); mask++) {
            long long sum = 0;
            for(int j=0; j<n; j++) {
                if((1 << j) & mask) sum += a[j];
                else sum -= b[j];
            }
            if(f) sum = abs(sum);
            aux.push_back(sum);
        }
        return aux;
    };

    if(n <= 20) {
        vector<long long> res = subsets(a, b, 1);
        sort(res.begin(), res.end());
        for(auto &it : res) cout << it << "␣";
        cout<< endl;
        return 0;
    }

    vector<long long> leftA(a.begin(), a.begin()+n/2),
                      leftB(b.begin(), b.begin()+n/2),
                      rightA(a.begin()+n/2, a.end()),
                      rightB(b.begin()+n/2, b.end());

    vector<long long> left = subsets(leftA, leftB, 0),
                      right = subsets(rightA, rightB, 0);
    sort(right.begin(), right.end());

    priority_queue<tuple<long long, int, int, int>, vector<tuple<long
        long, int, int, int>>, greater<tuple<long long, int, int, int>>>
         pq;
    for(int i=0; i<left.size(); i++) {
        int idx = lower_bound(right.begin(), right.end(), -left[i]) -
            right.begin();
        long long best = LONG_LONG_MAX, curr;
        int idxBest;
        for(int j=-1; j<=1; j++) {
            if(idx + j >= 0 && idx + j < right.size()) {
                curr = abs(left[i] + right[idx+j]);
                if(curr < best) {
                    best = curr;
                    idxBest = idx+j;
                }
            }
        }
        pq.push({best, idxBest, i, 0});
    }

    int req = 1 << 20, cnt = 0;
    while(!pq.empty() && cnt < req) {
        auto [sum, idx, i, event] = pq.top();
        pq.pop();
        cout << sum << "␣";
        cnt++;
        if(event <= 0 && idx-1 >= 0) {
            long long lsum = abs(left[i] + right[idx-1]);
```

```
            pq.push({lsum, idx-1, i, -1});
        }
        if(event >=0 && idx+1 < right.size()) {
            long long rsum = abs(left[i] + right[idx+1]);
            pq.push({rsum, idx+1, i, 1});
        }
    }
    cout << endl;

    return 0;
}
```

6.5 Dijkstra on ST

```
vector<pair<int, int>> adj[2*4*N];
ll dist[2*4*N];
int mp[N], vis[2*4*N];
void build(int l, int r, int i) {
    if (l == r) {
        mp[l] = i;
        adj[i].pb({i+4*N, 0});
        adj[i+4*N].pb({i, 0});
        return;
    }
    int m = (l+r)>>1;
    build(l, m, 2*i+1);
    build(m+1, r, 2*i+2);
    adj[i].pb({2*i+1, 0});
    adj[i].pb({2*i+2, 0});
    adj[2*i+1+4*N].pb({i+4*N, 0});
    adj[2*i+2+4*N].pb({i+4*N, 0});
}
void add(int l, int r, int u, int a, int b, int w, int op, int i) {
    if (a > r || b < l) return;
    if (a <= l && r <= b) {
        if(op == 2) adj[mp[u]].pb({i, w});
        else adj[i+4*N].pb({mp[u]+4*N, w});
        return;
    }
    int m = l+r>>1;
    add(l, m, u, a, b, w, op, 2*i+1);
    add(m+1, r, u, a, b, w, op, 2*i+2);
}
```

6.6 Venice set

```
struct VeniceSet {
    multiset<int> st;
    int global = 0;

    void add(int x) {
        st.insert(x + global);
    }
    void remove(int x) {
        st.erase(st.find(x + global));
    }
    void updateAll(int x) {
        global += x;
    }
    int min(int x) {
        return *st.begin() - global;
    }
};
```

7 Basic techniques

7.1 Difference array

```
// If we only need to get the final result, we can increase l and r+1
// to simulate range update and use a prefix sum to get the answer.
diff[l] += k;
diff[r+1] -= k;

// If we have initial values, we need to subtract the previous values:
diff[i] -= values[i - 1];
```

7.2 Prefix Sum 2D

```
// Compute prefix sum 2D
rep(i,1,n+1) rep(j,1,m+1)
    prefix[i][j] = prefix[i-1][j] + prefix[i][j-1]
                                  - prefix[i-1][j-1]
                                  + matrix[i-1][j-1]

// Query prefix sum 2D
query = prefix[i][j] - prefix[i-1][j] - prefix[i][j-1] + prefix[i-1][j
    -1]
```

```

// Specific parts
query = prefix[r2][c2] - prefix[r1-1][c2] - prefix[r2][c1-1] + prefix[r1
    -1][c1-1];
```

7.3 Random number generator

```
// Secure random seed
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

// Generate values using distribution
uniform_int_distribution<int> dist(0, N);       // [0, N]
randval = dist(rng);
```

7.4 Coordinate compression

```
// Push to an array all possible values
coord.pb(x);
sort(all(coord));
coord.resize(unique(all(coord)) - coord.begin());

auto get = [&](int x) = {
    return lower_bound(all(coord), x) - coord.begin();
};
```

7.5 Digit DP

```
int dp[11][2][2];
auto calc = [&](auto self, int idx, bool f1, bool f2) -> int {
    if(idx >= n) return 0;

    int &x = dp[idx][f1][f2];
    if(x != -1) return x;

    int num1 = a[idx] - '0', num2 = b[idx] - '0';
    int l = (f1 ? 0 : num1);
    int r = (f2 ? 9 : num2);
    int mn = 1e9+5;

    rep(i,l,r+1) {
        mn = min(mn, self(self, idx+1, f1 | (i > num1), f2 | (i < num2))
             + (num1==i) + (num2==i));
    }
    return x = mn;
};
memset(dp, -1, sizeof(dp));
cout << calc(calc, 0, 0, 0) << endl;
```

7.6 Intersection [L1, R1] and [L2, R2]

```
We need to check whether there is any number common to both ranges:
max(L1, L2) <= min(R1, R2)
```