```
==================  bittrie.cpp  ====================
class BitTrie
{
public:
    static constexpr int BITS = 31;       // for 32-bit ints use 31 down to 0
    vector<array<int,2>> nxt;
    vector<int> cnt;                      // count of numbers ending exactly at this node
    vector<int> pref;                     // count of numbers passing through this node
    int nodes = 0;

    BitTrie(int maxNodes = 1e6+1)
    {
        nxt.reserve(maxNodes);
        cnt.reserve(maxNodes);
        pref.reserve(maxNodes);
        newNode();
    }

    int newNode()
    {
        nxt.push_back({0,0});
        cnt.push_back(0);
        pref.push_back(0);
        return nodes++;
    }

    // insert x (f=+1) or remove x (f=-1)
    void insert(int x, int f = 1)
    {
        int u = 0;
        for (int b = BITS; b >= 0; --b)
        {
            int bit = (x >> b) & 1;
            if (!nxt[u][bit])
                nxt[u][bit] = newNode();
            u = nxt[u][bit];
            pref[u] += f;
        }
        cnt[u] += f;
    }

    void remove(int x) { insert(x, -1); }

    // exact count of x
    int count(int x) const
    {
        int u = 0;
        for (int b = BITS; b >= 0; --b)
        {
            int bit = (x >> b) & 1;
            int v = nxt[u][bit];
            if (!v || pref[v] <= 0) return 0;
            u = v;
        }
        return cnt[u];
    }

    // maximize x^y over all y in the trie
    int maxXor(int x) const
    {
        int u = 0;
        int ans = 0;
        for (int b = BITS; b >= 0; --b)
        {
```

```
            int bit = (x >> b) & 1;
            int want = bit ^ 1;
            if (nxt[u][want] && pref[nxt[u][want]] > 0)
            {
                ans |= (1 << b);
                u = nxt[u][want];
            } else
            {
                u = nxt[u][bit];
            }
        }
        return ans;
    }

    // count of elements     x
    int leq(int x) const
    {
        int u = 0, res = 0;
        for (int b = BITS; b >= 0; --b)
        {
            int bit = (x >> b) & 1;
            if (bit == 1)
            {
                if(nxt[u][0])
                    res += pref[nxt[u][0]];
                u = nxt[u][1];
            }
            else
            {
                u = nxt[u][0];
            }
        }
        if(u)
            res += cnt[u];
        return res;
    }
};


====================  block_decomposition.cpp  ====================
class element_chan
{
public:
};
class block_chan
{
public:
};

template<typename E, typename T, const int B>
class block_decomposition_chan
{
public:
    static int ceil_div(int x, int y)
    {
        return (x + y - 1)/y;
    }
    static int block_id(int i)
    {
        return i/B;
    }
    static int lb(int bid)
    {
        return bid * B;
    }
```

```
    static int rb(int bid)
    {
        return min(n, (bid + 1) * B - 1);
    }

public:
    int n;
    vector<E> element;
    vector<T> block;

    block_decomposition_chan(int n, vector<E> a, vector<T> b) : n(n), element(a), block(b)
    {
    };

    void process(int l, int r, auto block_brute, auto block_quick)
    {
        assert(1 <= l and l <= r and r <= n);
        int bl = block_id(l), br = block_id(r);
        if(bl == br)
            block_brute(l, r);
        else
        {
            block_brute(l, rb(bl));
            for(int b = bl + 1; b < br; b ++)
                block_quick(b);
            block_brute(lb(br), r);
        }
    }
};


====================  bridges.cpp  ====================
vector<bool> visited;
vector<int> tin, low;
int timer;
timer = 0;
visited.assign(n, false);
tin.assign(n, -1);
low.assign(n, -1);

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    bool parent_skipped = false;
    for (int to : adj[v]) {
        if (to == p && !parent_skipped) { // do this for multiple edges
            parent_skipped = true;
            continue;
        }
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}


====================  centroid.cpp  ====================
int nodes = 0;
int subtree[N], parentcentroid[N];
set<int> g[N];
```

```
195
196
197 void dfs(int u, int par){
198         nodes++;
199         subtree[u] = 1;
200         for(auto &it:g[u]){
201                 if(it == par)
202                         continue;
203                 dfs(it, u);
204                 subtree[u] += subtree[it];
205         }
206 }
207
208 int centroid(int k, int parent){
209         for(auto it:g[k]){
210                 if(it==parent)
211                         continue;
212                 if(subtree[it]>(nodes>>1))
213                         return centroid(it,k);
214         }
215         return k;
216 }
217
218 void decompose(int u, int par){
219         nodes = 0;
220         dfs(u, u);
221         int node = centroid(u, u);
222         //do something
223         parentcentroid[node] = par;
224         for(auto &it:g[node]){
225                 g[it].erase(node);
226                 decompose(it, node);
227         }
228 }
229
230
231 /*
232 Properties of Centroid Tree(VERY IMP): https://www.quora.com/q/threadsiiithyderabad/
        Centroid-Decomposition-of-a-Tree
233 */
234
235
236
237 ====================  combinatorics2d.cpp  ====================
238 template<typename T>
239 class combinatorics2d
240 {
241     public:
242     int n;
243     vector<vector<T>> C;
244     combinatorics2d(int n) : n(n), C(n+1, vector<T>(n+1, 0))
245     {
246         for (int i = 0; i <= n; i++)
247         {
248             C[i][0] = C[i][i] = T(1);
249             for (int j = 1; j < i; j++)
250             {
251                 C[i][j] = C[i-1][j-1] + C[i-1][j];
252             }
253         }
254     }
255
256     T ncr(int N, int R)
257     {
258         if (R < 0 || R > N) return T(0);
```

```
        return C[N][R];
    }
};


====================  combinatorics.cpp  ====================
template<typename T, const int P>
class combinatorics
{
    //combinatorics<mint,2> c(200);
    //2 is the number whose power will be precomputed
public:
    int n;
    vector<T> inv, fac, ifac, pw;
    combinatorics (int n) : n(n), inv(n+1), fac(n+1), ifac(n+1), pw(n+1)
    {
        fac[0] = inv[0] = ifac[0] = pw[0] = T(1);

        for(int i = 1; i <= n; i ++)
            inv[i] = T(1)/T(i), fac[i] = fac[i - 1] * T(i), ifac[i] = ifac[i - 1] * inv[i
                ], pw[i] = pw[i - 1] * T(P);
    }

    T ncr(int n, int r)
    {
        if(n < r or r < 0)
            return 0;
        return fac[n] * ifac[r] * ifac[n - r];
    }
};


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

// Extended Euclidean Algorithm
int extended_gcd(int a, int b, int &x, int &y)
{
    if (b == 0) { x = 1; y = 0; return a; }
    int x1, y1;
    int g = extended_gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return g;
}

// Modular inverse using extended Euclid
int modinv(int a, int m)
{
    int x, y;
    int g = extended_gcd(a, m, x, y);
    if (g != 1) return -1; // Inverse doesn't exist
    return (x % m + m) % m;
}

// Chinese Remainder Theorem solver
int crt(vector<int> &a,  vector<int> &m)
{
    int M = 1;
    for (int mi : m) M *= mi;

    int result = 0;
    for (int i = 0; i < a.size(); ++i) {
        int Mi = M / m[i];
```

```
        int inv = modinv(Mi, m[i]);
        result = (result + a[i] * Mi % M * inv % M) % M;
    }

    return (result % M + M) % M; // Ensure positive
}



==========  custom_hash.cpp  ==========
// #include<bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>

struct splitmix64_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        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 = std::chrono::steady_clock::now().
            time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

template <typename K, typename V, typename Hash = splitmix64_hash>
using HashMap = __gnu_pbds::gp_hash_table<K, V, Hash>;

template <typename K, typename Hash = splitmix64_hash>
using HashSet = HashMap<K, __gnu_pbds::null_type, Hash>;


==========  dsu.cpp  ==========
class dsu_chan
{
public:
    int n;
    vector<int> par, siz;
    vector<int> nxt;    // for fast skipping of deleted elements
    vector<bool> dead;  // whether element is deleted

    dsu_chan(int n) : n(n), par(n), siz(n, 1), nxt(n + 1), dead(n, false)
    {
        iota(par.begin(), par.end(), 0);
        iota(nxt.begin(), nxt.end(), 0);
    }

    int get(int x)
    {
        return (par[x] == x ? x : par[x] = get(par[x]));
    }

    void unite(int x, int y)
    {
        x = get(x), y = get(y);
        if (x == y) return;
        if (siz[x] > siz[y]) swap(x, y);
        par[x] = y;
        siz[y] += siz[x];
    }

```

```
    // find next alive element >= x
    int next_alive(int x)
    {
        if (x >= n) return n;
        return nxt[x] == x ? x : nxt[x] = next_alive(nxt[x]);
    }

    // delete all elements from l..r (inclusive)
    void erase_range(int l, int r)
    {
        int cnt = 0;
        for (int i = next_alive(l); i <= r; i = next_alive(i))
        {
            dead[i] = true;
            cnt++;
            nxt[i] = i + 1; // skip this element in future
        }
        cout << cnt << "\n";
    }

    vector<vector<int>> group()
    {
        vector<vector<int>> g(n);
        for (int u = 0; u < n; u++)
            if (!dead[u])
                g[get(u)].push_back(u);
        return g;
    }
};




==========  fenwick2d.cpp  ==========
template <typename T>
class fenwick2d
{
    public:
    int n, m;
    vector<vector<T>> tree;
    fenwick2d(int n, int m)
    {
        this->n = n;
        this->m = m;
        tree.assign(n+1,vector<T>(m+1,0));
    }
    void update(int x, int y, T val)
    {
        for (int i = x; i <= n; i += (i & -i))
        {
            for (int j = y; j <= m; j += (j & -j))
            {
                tree[i][j] += val;
            }
        }
    }
    T query(int x, int y)
    {
        T sum = 0;
        for (int i = x; i > 0; i -= (i & -i))
        {
            for (int j = y; j > 0; j -= (j & -j))
            {
                sum += tree[i][j];
            }
```

```
        }
        return sum;
    }
    T query(int x1, int y1, int x2, int y2)
    {
        return query(x2, y2)
             - query(x1 - 1, y2)
             - query(x2, y1 - 1)
             + query(x1 - 1, y1 - 1);
    }
    // 1 based and [x1..x2]*[y1..y2]
};


====================  fenwick.cpp  ====================
template <typename T>
class fenwick_tree_chan
{
public:
    vector<T> fenw;
    int n;
    int pw;

    fenwick_tree_chan() : n(0) {}
    fenwick_tree_chan(int n) : n(n)
    {
        fenw.resize(n);
        pw = (n == 0 ? 0 : 1ULL << (63 - __builtin_clzll(unsigned(n))));
    }

    // a[x] += v;
    void modify(int x, T v)
    {
        assert(0 <= x && x < n);
        while (x < n)
        {
            fenw[x] += v;
            x |= x + 1;
        }
    }

    /// sum of prefix [0, .. x]
    T query(int x)
    {
        ++ x;
        assert(0 <= x && x <= n);
        T v{};
        while (x > 0)
        {
            v += fenw[x - 1];
            x &= x - 1;
        }
        return v;
    }

    // Returns the length of the longest prefix (0 indexed) with sum <= c
    int max_prefix(T c)
    {
        T v{};
        int at = 0;
        for (int len = pw; len > 0; len >>= 1)
        {
            if (at + len <= n)
            {
                auto nv = v;
```

```
                nv += fenw[at + len - 1];
                if (!(c < nv))
                {
                    v = nv;
                    at += len;
                }
            }
        }
        assert(0 <= at && at <= n);
        return at;
    }
};

struct fenwick_lazy
{
    fenwick_tree_chan<int> B1, B2;
    int n;

    fenwick_lazy(int n) : n(n), B1(n), B2(n) {}

    void range_add(int l, int r, int v)
    {
        B1.modify(l, v);
        if (r + 1 < n) B1.modify(r + 1, -v);
        B2.modify(l, v * (l - 1));
        if (r + 1 < n) B2.modify(r + 1, -v * r);
    }

    int prefix_sum(int x)
    {
        if (x < 0) return 0;
        return B1.query(x) * x - B2.query(x);
    }

    int range_sum(int l, int r)
    {
        return prefix_sum(r) - prefix_sum(l - 1);
    }
};



====================  heavy_light.cpp  ====================
template <class S,
          auto op,
          auto e,
          class F,
          auto mapping,
          auto composition,
          auto id,
          const bool islazy,
          const bool on_edge>
class heavy_light_chan
{
    /*
        info:
            - 0 indexed
            - range [pos[u], out[u]) represents subtree of u
        vars:
            - r = tree root
            - heavy[u] = heavy child v (edge (u,v) is heavy edge)
            - root[u] = It is the starting point of the heavy chain (for u-v it is u, if v
                is heavy and u is not heavy).
            - on_edge = true => values on edges. Internally, value of edge is stored at
                lower node (node more away from root)
```

```
        warning:
            - handle segtree initialization correctly
    */
    using lazy_t = lazy_segtree<S, op, e, F, mapping, composition, id>;
    using simple_t = simple_segtree<S, op, e, F, mapping>;
    using seg_t = std::conditional_t<islazy, lazy_t, simple_t>;

public:
    int n, r;
    vector<int> par, heavy, dep, root, pos, out;
    seg_t seg;
    heavy_light_chan(int n, vector<vector<int>> adj, int r = 0) :
    n(n), r(r), par(n, -1), heavy(n, -1), dep(n), root(n), pos(n), out(n),
    seg(n)
    {
        assert(r < n);
        auto dfs_sz = [&](int u, auto &&dfs) -> int
        {
            //* ensure that in adj list first child is the heavy child
            int sz = 1, mx_sz = 0;
            for(auto &v : adj[u])
            {
                if(v != par[u])
                {
                    par[v] = u, dep[v] = dep[u] + 1;
                    int s = dfs(v, dfs);
                    sz += s;
                    if(s > mx_sz)
                    heavy[u] = v, mx_sz = s, swap(adj[u][0], v);
                }
            }
            return sz;
        };
        int timer = 0;
        auto dfs_hld = [&](int u, auto &&dfs) -> void
        {
            pos[u] = timer++;
            for(auto v : adj[u])
            {
                if(v != par[u])
                {
                    root[v] = (heavy[u] == v ? root[u] : v);
                    dfs(v, dfs);
                }
            }
            out[u] = timer ; // exclusive
        };
        par[r] = -1;
        dep[r] = 0;
        root[r] = r;
        dfs_sz(r, dfs_sz);
        dfs_hld(r, dfs_hld);
    }
    int lca(int u, int v)
    {
        for (; root[u] != root[v]; v = par[root[v]])
        {
            if (dep[root[u]] > dep[root[v]])
            {
                swap(u, v);
            }
        }
        return (dep[u] < dep[v] ? u : v);
    }

```

```
    // *process_path [u, v] in O(logn * logn)
    template <typename O>
    void process_path(int u, int v, O oper)
    {
        for (; root[u] != root[v]; v = par[root[v]])
        {
            if (dep[root[u]] > dep[root[v]])
                swap(u, v);
            oper(pos[root[v]], pos[v]);
        }
        if (dep[u] > dep[v])
            swap(u, v);

        if(!on_edge)
            oper(pos[u], pos[v]);
        else if(u != v)
            oper(pos[u] + 1, pos[v]);
    }

    void set(int v, const S &value)
    {
        seg.set(pos[v], value);
    }

    S get(int v)
    {
        return seg.get(pos[v]);
    }

    void modify_path(int u, int v, const F &f)
    {
        process_path(u, v, [this, &f](int l, int r)
        {
            seg.apply(l, r + 1, f); // convert inclusive [l,r] -> atcoder apply [l, r+1)
        });
    }

    S query_path(int u, int v)
    {
        S res = e();
        process_path(u, v, [this, &res](int l, int r)
        {
            S part = seg.prod(l, r + 1);
            res = op(res, part);
        });
        return res;
    }

    void modify_subtree(int u, const F &f)
    {
        if (!on_edge)
        {
            seg.apply(pos[u], out[u], f); // subtree = [pos[u], out[u])
        }
        else
        {
            // edges stored at children; exclude its parent values that it at u
            if (pos[u] < out[u] - 1)
                seg.apply(pos[u] + 1, out[u], f);
        }
    }
    S query_subtree(int u)
    {
        if (on_edge)
        {
```

```
            if (pos[u] < out[u] - 1)
            {
                return seg.prod(pos[u] + 1, out[u]);
            }
            else
            {
                return e();
            }
        }
        return seg.prod(pos[u], out[u]);
    }

};



====================  lazy_segtree.cpp  ====================
template <class S,
          auto op,
          auto e,
          class F,
          auto mapping,
          auto composition,
          auto id>
struct lazy_segtree
{
public:
    unsigned int bit_ceil(unsigned int n)
    {
        unsigned int x = 1;
        while (x < (unsigned int)(n))
            x *= 2;
        return x;
    }
    int countr_zero(unsigned int n)
    {
        return __builtin_ctz(n);
    }
    lazy_segtree() : lazy_segtree(0) {}
    explicit lazy_segtree(int n) : lazy_segtree(std::vector<S>(n, e())) {}
    explicit lazy_segtree(const std::vector<S> &v) : _n((int)(v.size()))
    {
        size = (int)bit_ceil((unsigned int)(_n));
        log = countr_zero((unsigned int)size);
        d = std::vector<S>(2 * size, e());
        lz = std::vector<F>(size, id());
        for (int i = 0; i < _n; i++)
            d[size + i] = v[i];
        for (int i = size - 1; i >= 1; i--)
        {
            update(i);
        }
    }

    void set(int p, S x)
    {
        assert(0 <= p && p < _n);
        p += size;
        for (int i = log; i >= 1; i--)
            push(p >> i);
        d[p] = x;
        for (int i = 1; i <= log; i++)
            update(p >> i);
    }

```

```
    S get(int p)
    {
        assert(0 <= p && p < _n);
        p += size;
        for (int i = log; i >= 1; i--)
            push(p >> i);
        return d[p];
    }

    S prod(int l, int r)
    {
        assert(0 <= l && l <= r && r <= _n);
        if (l == r)
            return e();

        l += size;
        r += size;

        for (int i = log; i >= 1; i--)
        {
            if (((l >> i) << i) != l)
                push(l >> i);
            if (((r >> i) << i) != r)
                push((r - 1) >> i);
        }

        S sml = e(), smr = e();
        while (l < r)
        {
            if (l & 1)
                sml = op(sml, d[l++]);
            if (r & 1)
                smr = op(d[--r], smr);
            l >>= 1;
            r >>= 1;
        }

        return op(sml, smr);
    }

    S all_prod() { return d[1]; }

    void apply(int p, F f)
    {
        assert(0 <= p && p < _n);
        p += size;
        for (int i = log; i >= 1; i--)
            push(p >> i);
        d[p] = mapping(f, d[p]);
        for (int i = 1; i <= log; i++)
            update(p >> i);
    }
    void apply(int l, int r, F f)
    {
        assert(0 <= l && l <= r && r <= _n);
        if (l == r)
            return;

        l += size;
        r += size;

        for (int i = log; i >= 1; i--)
        {
            if (((l >> i) << i) != l)
                push(l >> i);
```

```
            if (((r >> i) << i) != r)
                push((r - 1) >> i);
        }

        {
            int l2 = l, r2 = r;
            while (l < r)
            {
                if (l & 1)
                    all_apply(l++, f);
                if (r & 1)
                    all_apply(--r, f);
                l >>= 1;
                r >>= 1;
            }
            l = l2;
            r = r2;
        }

        for (int i = 1; i <= log; i++)
        {
            if (((l >> i) << i) != l)
                update(l >> i);
            if (((r >> i) << i) != r)
                update((r - 1) >> i);
        }
    }

    template <bool (*g)(S)>
    int max_right(int l)
    {
        return max_right(l, [](S x)
                         { return g(x); });
    }
    template <class G>
    int max_right(int l, G g)
    {
        assert(0 <= l && l <= _n);
        assert(g(e()));
        if (l == _n)
            return _n;
        l += size;
        for (int i = log; i >= 1; i--)
            push(l >> i);
        S sm = e();
        do
        {
            while (l % 2 == 0)
                l >>= 1;
            if (!g(op(sm, d[l])))
            {
                while (l < size)
                {
                    push(l);
                    l = (2 * l);
                    if (g(op(sm, d[l])))
                    {
                        sm = op(sm, d[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, d[l]);
            l++;
```

```
905         } while ((l & -l) != l);
906         return _n;
907     }
908 
909     template <bool (*g)(S)>
910     int min_left(int r)
911     {
912         return min_left(r, [](S x)
913                         { return g(x); });
914     }
915     template <class G>
916     int min_left(int r, G g)
917     {
918         assert(0 <= r && r <= _n);
919         assert(g(e()));
920         if (r == 0)
921             return 0;
922         r += size;
923         for (int i = log; i >= 1; i--)
924             push((r - 1) >> i);
925         S sm = e();
926         do
927         {
928             r--;
929             while (r > 1 && (r % 2))
930                 r >>= 1;
931             if (!g(op(d[r], sm)))
932             {
933                 while (r < size)
934                 {
935                     push(r);
936                     r = (2 * r + 1);
937                     if (g(op(d[r], sm)))
938                     {
939                         sm = op(d[r], sm);
940                         r--;
941                     }
942                 }
943                 return r + 1 - size;
944             }
945             sm = op(d[r], sm);
946         } while ((r & -r) != r);
947         return 0;
948     }
949 
950 private:
951     int _n, size, log;
952     std::vector<S> d;
953     std::vector<F> lz;
954 
955     void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
956     void all_apply(int k, F f)
957     {
958         d[k] = mapping(f, d[k]);
959         if (k < size)
960             lz[k] = composition(f, lz[k]);
961     }
962     void push(int k)
963     {
964         all_apply(2 * k, lz[k]);
965         all_apply(2 * k + 1, lz[k]);
966         lz[k] = id();
967     }
968 };
969 /*
```

```
    !index is 0 based [l,r)

    S - segment tree node
    op(left, right) - monoid merge of two S values
    e() - identity element for op
    F - lazy tag type
    mapping - apply a tag f to a node value s
    composition - combine two tags (f applied after g)
    id - neutral tag (no-op)

    void set(int p, S x) - ar[p] = x
    S get(int p) - return ar[p]

    S prod(int l, int r) - Combine values in [l, r) via op.
    S all_prod() == prod for whole array

    void apply(int p, F f) - Apply tag f only to index p.
    void apply(int l, int r, F f) - Apply tag f to every index in [l, r).

    int max_right(int l, G g) - max r s.t g(l...r-1) is true and g(l...r) is false; true for e()
    int min_left(int r, G g) - min l s.t g(l...r-1) is true and g(l-1....r-1) is false; true for e()



    struct S { int sum, size; };
    S op(S left, S right) { return {left.sum + right.sum, left.size + right.size}; }
    S e() { return {0, 0}; }
    struct F { int x; bool is_set; }; // lazy tag type
    S mapping(F f, S s)  // apply tag f to segment s
    {
        if (!f.is_set) return s;
        return {f.x * s.size, s.size};
    }

    F composition(F f, F g) { // Compose two tags: new f after old g
        if (f.is_set) return f;
        return g;
    }
    F id() { return {0, false}; }

    * constructor - expects vector<S>

*/


====================  lca.cpp  ====================
class binary_lifter_chan
{
    /*
    tc: O(n logn) preprocessing, O(logn) query
    ml: O(n logn)

    info:
        1 indexed
        n -> number of nodes
        r -> root of the tree
        dep[u] -> depth of node u
        tin[u], tout[u] -> intime, outtime of node u
        up[u][i] -> stores 2^ith ancestor of u
    */
public:
    int n, L, timer;
    vector<int> dep, tin, tout;
```

```
    vector<vector<int>> up;

    binary_lifter_chan(int n, int r, const vector<vector<int>> &adj) :
    n(n), L(ceil(log2(n)) + 1), timer(0), dep(n), tin(n), tout(n), up(n, vector<int> (L, r
        ))
    {
        timer = 0;
        dep[r] = 0;
        dfs(r, r, adj);
    }

    void dfs(int u, int p, const vector<vector<int>> &adj)
    {
        tin[u] = ++ timer;
        up[u][0] = p;

        for(int i = 1; i < L; ++i)
            up[u][i] = up[up[u][i - 1]][i - 1];

        for(auto v : adj[u])
            if (v != p)
                dep[v] = dep[u] + 1, dfs(v, u, adj);

        tout[u] = ++ timer;
    }

    int get_kth(int v, int k)
    {
        if(k != 0)
            for(int i = L - 1; i >= 0 and v > 0; i --)
                if((1 << i) <= k)
                    k -= (1 << i), v = up[v][i];
        return v;
    }

    bool is_anc(int anc, int v)
    {
        return tin[anc] <= tin[v] and tout[v] <= tout[anc];
    }

    int lca(int u, int v)
    {
        if (is_anc(u, v))
            return u;
        if (is_anc(v, u))
            return v;
        for (int i = L - 1; i >= 0; --i)
            if (!is_anc(up[u][i], v))
                u = up[u][i];
        return up[u][0];
    }
};


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

const int N = 10000000;
vector<int> lp(N+1); //lowest prime factor
vector<int> pr; // prime list

void sieve(){
    for(int i = 2; i <= N; ++i){
        if (lp[i] == 0) {
```

```
                lp[i] = i;
                pr.push_back(i);
        }
        for (int j = 0; i * pr[j] <= N  && j < (int)pr.size(); ++j) {
            lp[i * pr[j]] = pr[j];
            if (pr[j] == lp[i]) {
                    break;
                }
        }
    }
}

map<int,int>factorize(int k)
{
    map<int,int> mp;
    while(k >1)
    {
        mp[lp[k]]++;
        k/=lp[k];
    }
    return mp;
}



====================  matrix_op.cpp  ====================
template <typename T, size_t N, size_t M, size_t K>
array<array<T, K>, N> operator*(const array<array<T, M>, N>& a, const array<array<T, K>, M
   >& b) {
  array<array<T, K>, N> c;
  for (size_t i = 0; i < N; i++) {
    for (size_t j = 0; j < K; j++) {
      c[i][j] = 0;
      for (size_t k = 0; k < M; k++) {
        c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return c;
}

template <typename T>
vector<vector<T>> operator*(const vector<vector<T>>& a, const vector<vector<T>>& b) {
  if (a.empty() || b.empty()) {
    return {{}};
  }
  vector<vector<T>> c(a.size(), vector<T>(b[0].size()));
  for (int i = 0; i < static_cast<int>(c.size()); i++) {
    for (int j = 0; j < static_cast<int>(c[0].size()); j++) {
      c[i][j] = 0;
      for (int k = 0; k < static_cast<int>(b.size()); k++) {
        c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return c;
}

template <typename T>
vector<vector<T>>& operator*=(vector<vector<T>>& a, const vector<vector<T>>& b) {
  return a = a * b;
}

template <typename T, typename U>
vector<vector<T>> power(const vector<vector<T>>& a, const U& b) {
```

```
  assert(b >= 0);
  vector<U> binary;
  U bb = b;
  while (bb > 0) {
    binary.push_back(bb & 1);
    bb >>= 1;
  }
  vector<vector<T>> res(a.size(), vector<T>(a.size()));
  for (int i = 0; i < static_cast<int>(a.size()); i++) {
    res[i][i] = 1;
  }
  for (int j = (int) binary.size() - 1; j >= 0; j--) {
    res *= res;
    if (binary[j] == 1) {
      res *= a;
    }
  }
  return res;
}


====================  mint_binpow.cpp  ====================
const int mod = 1e9 + 7;

int add(int a, int b)
{
    a %= mod; b %= mod;
    int res = a + b;
    if (res >= mod) res -= mod;
    return res;
}

int sub(int a, int b)
{
    a %= mod; b %= mod;
    int res = a - b;
    if (res < 0) res += mod;
    return res;
}

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

int power(int a, int b, int m = mod)
{
    a %= m;
    long long res = 1;
    while (b > 0)
    {
        if (b & 1) res = res * a % m;
        a = 1LL * a * a % m;
        b >>= 1;
    }
    return res;
}

int div(int a, int b)
{
    return mult(a, power(b, mod - 2, mod));
}
```

```
====================  offline_lca.cpp  ====================
vector<int> offline_lca_chan(int n, int r, const vector<vector<int>> &adj, vector<pair<int
    , int>> query)
{
    //dsu
    // O(n+m) pre, O(1) query
    vector<int> par(n), siz(n, 1);
    iota(par.begin(), par.end(), 0);
    auto get = [&](int u, auto &&get) -> int
    {
        return (par[u] == u ? u : par[u] = get(par[u], get));
    };
    auto unite = [&](int u, int v) -> void
    {
        u = get(u, get), v = get(v, get);
        if(u == v)
            return;
        if(siz[u] < siz[v])
            swap(u, v);
        par[v] = u, siz[u] += siz[v];
    };

    assert(!query.empty());
    int m = query.size();

    vector<int> ans(m);
    vector<bool> see(m);
    vector<vector<int>> store(n);

    for(int i = 0; i < m; i ++)
    {
        auto [u, v] = query[i];
        store[u].push_back(i), store[v].push_back(i);
    }

    auto dfs = [&](int u, int p, auto &&dfs) -> void
    {
        for(auto i : store[u])
        {
            if(see[i])
                ans[i] = get(query[i].first == u ? query[i].second : query[i].first, get);
            see[i] = true;
        }

        for(auto v : adj[u])
            if(v != p)
                dfs(v, u, dfs);

        if(p != 0)
            unite(u, p);
    };
    dfs(r, 0, dfs);

    return ans;
};



====================  ordered_set.cpp  ====================
// #include<bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template <typename K, typename V, typename Comp = std::less<K>>
```

```cpp
using ordered_map = __gnu_pbds::tree<
    K, V, Comp,
    __gnu_pbds::rb_tree_tag,
    __gnu_pbds::tree_order_statistics_node_update
>;

template<class T> using
ordered_multiset = tree<T, null_type, less_equal<T>, rb_tree_tag,
   tree_order_statistics_node_update>;
template <typename K, typename Comp = std::less<K>>
using ordered_set = ordered_map<K, __gnu_pbds::null_type, Comp>;

// Supports
//  auto iterator = ordered_set().find_by_order(idx); // (0-indexed)
//  int num_strictly_smaller = ordered_set().order_of_key(key);



==========  phi.cpp  ==========
int phi(int n) { // O(sqrt(n))
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            result -= result / i;
        }
    }
    if (n > 1) result -= result / n;
    return result;
}
vector<int> phi; // phi[1] is 1
void phi_1_to_n(int n) { // 1 to n in O(nloglogn)
    phi.resize(n + 1);
    for (int i = 0; i <= n; i++) phi[i] = i;
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i) phi[j] -= phi[j] / i;
        }
    }
}


==========  pragma.cpp  ==========
#include <bits/allocator.h>
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")


==========  rng.cpp  ==========
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());



==========  scc.cpp  ==========
class condenser
{
    /*
    https://github.com/welcome-to-the-sunny-side/libra/

    tc: O(n + m)
    ml: O(n + m)

    info:
        0-indexed
        SCC u occurs before SCC v (u < v) in some topological ordering of SCCs
```

```
    warning:
        there may be multiple edges between different SCCs

    var:

        [n -> number of nodes]
        [c -> number of SCCs]
        [adj] -> G
        [rdj] -> G.T
        [comp[u] -> component number of node u]
        [scc[u] -> outedge list for scc u]
        [grp[u] -> node list for scc u]
    */
public:
    int n, c;
    vector<bool> vis;
    vector<int> stak, comp;
    vector<vector<int>> adj, rdj, scc, grp;

    condenser(int n, const vector<vector<int>> &adj) :
    n(n), c(0), vis(n), adj(adj), rdj(n), scc(n), grp(n), comp(n, -1)
    {
        for(int u = 0; u < n; u ++)
            for(auto v : adj[u])
                rdj[v].push_back(u);
    };

    void condense()
    {
        auto dfs1 = [&](int u, auto &&dfs1) -> void
        {
            vis[u] = true;
            for(auto v : adj[u])
                if(!vis[v])
                    dfs1(v, dfs1);
            stak.push_back(u);
        };
        for(int u = 0; u < n; u ++)
            if(!vis[u])
                dfs1(u, dfs1);

        auto dfs2 = [&](int u, auto &&dfs2) -> void
        {
            comp[u] = c;
            for(auto v : rdj[u])
                if(comp[v] == -1)
                    dfs2(v, dfs2);
        };

        reverse(stak.begin(), stak.end());
        for(auto u : stak)
            if(comp[u] == -1)
                dfs2(u, dfs2), ++ c;

        for(int u = 0; u < n; u ++)
            for(auto v : adj[u])
                if(comp[u] != comp[v])
                    scc[comp[u]].push_back(comp[v]);

        for(int u = 0; u < n; u ++)
            grp[comp[u]].push_back(u);
    }

    void fix()      //remove multiple edges [O(m log(m))]
```

```
    {
        for(auto &v : scc)
        {
            sort(v.begin(), v.end());
            v.erase(unique(v.begin(), v.end()), v.end());
        }
    }
};


====================  segmented_sieve.cpp  ====================
void segmented_sieve()
{

    int l,r;
    cin>>l>>r;

    int lim = sqrt(r)+1;

    vi base_primes;
    vector<bool> is_prime(lim+1,1);
    for(int i=2;i*i<=r;i++){
        if(!is_prime[i]) continue;
        for(int j = i*i;j<=lim;j+=i) {
            // dbg(j);
            is_prime[j] = false;
        }
    }

    for(int i = 2;i*i<=r;i++) if(is_prime[i]) base_primes.pb(i);

    vi is_prime_seg(r-l+1,1);
    for(int p : base_primes){
        int st = ((l+p-1)/p) * p;
        for(int j = st;j<=r;j+=p) is_prime_seg[j-l] = p;
    }
    if(l==1) is_prime_seg[0] = false;
    // is_prime_seg[i] = true ==> l+i is prime
}




====================  simple_segtree.cpp  ====================
template <class S, auto op, auto e, class F, auto mapping>
struct simple_segtree
{
    int _n;
    int size;
    std::vector<S> d;
    simple_segtree() : simple_segtree(0) {}
    explicit simple_segtree(int n) { init(n); }
    explicit simple_segtree(const std::vector<S> &v)
    {
        _n = (int)v.size();
        size = 1;
        while (size < _n) size <<= 1;
        d.assign(2 * size, e());
        for (int i = 0; i < _n; ++i) d[size + i] = v[i];
        for (int i = size - 1; i >= 1; --i) d[i] = op(d[2*i], d[2*i+1]);
    }

    void init(int n)
    {
        _n = n;
```

```
        size = 1;
        while (size < _n) size <<= 1;
        d.assign(2 * size, e());
    }

    void set(int p, S x)
    {
        assert(0 <= p && p < _n);
        int pos = size + p;
        d[pos] = x;
        pos >>= 1;
        while (pos >= 1)
        {
            d[pos] = op(d[2*pos], d[2*pos+1]);
            pos >>= 1;
        }
    }

    S get(int p)
    {
        assert(0 <= p && p < _n);
        return d[size + p];
    }

    // prod on [l, r)
    S prod(int l, int r)
    {
        assert(0 <= l && l <= r && r <= _n);
        if (l == r) return e();
        l += size; r += size;
        S sml = e(), smr = e();
        while (l < r)
        {
            if (l & 1) sml = op(sml, d[l++]);
            if (r & 1) smr = op(d[--r], smr);
            l >>= 1; r >>= 1;
        }
        return op(sml, smr);
    }

    // apply range [l, r) with mapping(f, s) done as point-wise updates (O(range * (log n)
       )!)
    void apply(int l, int r, F f)
    {
        assert(0 <= l && l <= r && r <= _n);
        if (l == r) return;
        for (int i = l; i < r; ++i)
        {
            S cur = get(i);
            S nxt = mapping(f, cur);
            set(i, nxt);
        }
    }

    template <bool (*g)(S)>
    int max_right(int l)
    {
        return max_right(l, [](S x){ return g(x); });
    }

    template <class G>
    int max_right(int l, G g)
    {
        assert(0 <= l && l <= _n);
        assert(g(e()));
```

```
        if (l == _n) return _n;
        int idx = l + size;
        S sm = e();
        do
        {
            while ((idx & 1) == 0) idx >>= 1;
            if (!g(op(sm, d[idx])))
            {
                while (idx < size)
                {
                    idx <<= 1;
                    if (g(op(sm, d[idx])))
                    {
                        sm = op(sm, d[idx]);
                        ++idx;
                    }
                }
                return idx - size;
            }
            sm = op(sm, d[idx]);
            ++idx;
        } while ((idx & -idx) != idx);
        return _n;
    }

    template <bool (*g)(S)>
    int min_left(int r)
    {
        return min_left(r, [](S x){ return g(x); });
    }

    template <class G>
    int min_left(int r, G g)
    {
        assert(0 <= r && r <= _n);
        assert(g(e()));
        if (r == 0) return 0;
        int idx = r + size;
        S sm = e();
        do
        {
            --idx;
            while (idx > 1 && (idx & 1))
                idx >>= 1;
            if (!g(op(d[idx], sm)))
            {
                while (idx < size)
                {
                    idx = idx * 2 + 1;
                    if (g(op(d[idx], sm)))
                    {
                        sm = op(d[idx], sm);
                        --idx;
                    }
                }
                return idx + 1 - size;
            }
            sm = op(d[idx], sm);
        } while ((idx & -idx) != idx);
        return 0;
    }
};

//* 0 based indexing
//* [l,r)
```

```
//* int max_right(int l, G g) - max r s.t g(l...r-1) is true and g(l...r) is false; true for e()
//* int min_left(int r, G g) - min l s.t g(l...r-1) is true and g(l-1....r-1) is false; true for e()
//* monoid - op is associative and e() is identity



====================  sparse_table.cpp  ====================
class sparse_table
{
    vector<vector<int>> st;
    vector<int> log;
    int func(int a, int b)
    {
        return min(a, b);
    }

public:
    sparse_table(vector<int> &arr)
    {
        int n = arr.size();
        log.resize(n + 1);
        for (int i = 2; i <= n; ++i)
        {
            log[i] = log[i / 2] + 1;
        }
        int k = log[n] + 1;
        st.assign(n, vector<int>(k));
        for (int i = 0; i < n; ++i)
        {
            st[i][0] = arr[i];
        }
        for (int j = 1; (1 << j) <= n; ++j)
        {
            for (int i = 0; i + (1 << j) - 1 < n; ++i)
            {
                st[i][j] = func(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    int query(int L, int R) // 0 based indexing [l,r]
    {
        int j = log[R - L + 1];
        return func(st[L][j], st[R - (1 << j) + 1][j]);
    }
};



====================  string_hash.cpp  ====================
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define INF (int)1e18

mt19937_64 RNG(chrono::steady_clock::now().time_since_epoch().count());

struct Hash{
    int b, n; // b = number of hashes
    const int mod = 1e9 + 7;
    vector<vector<int>> fw, bc, pb, ib;
    vector<int> bases;

```

```
    inline int power(int x, int y){
        if (y == 0){
            return 1;
        }

        int v = power(x, y / 2);
        v = 1LL * v * v % mod;
        if (y & 1) return 1LL * v * x % mod;
        else return v;
    }

    inline void init(int nn, int bb, string str)//nn=size,bb=number of mods, str=string
    {
        n = nn;
        b = bb;
        fw = vector<vector<int>>(b, vector<int>(n + 2, 0));
        bc = vector<vector<int>>(b, vector<int>(n + 2, 0));
        pb = vector<vector<int>>(b, vector<int>(n + 2, 1));
        ib = vector<vector<int>>(b, vector<int>(n + 2, 1));
        bases = vector<int>(b);
        str = "0" + str;

        for (auto &x : bases) x = RNG() % (mod / 10);

        for (int i = 0; i < b; i++){
            for (int j = 1; j <= n + 1; j++){
                pb[i][j] = 1LL * pb[i][j - 1] * bases[i] % mod;
            }
            ib[i][n + 1] = power(pb[i][n + 1], mod - 2);
            for (int j = n; j >= 1; j--){
                ib[i][j] = 1LL * ib[i][j + 1] * bases[i] % mod;
            }

            for (int j = 1; j <= n; j++){
                fw[i][j] = (fw[i][j - 1] + 1LL * (str[j] - 'a' + 1) * pb[i][j]) % mod;
            }
            for (int j = n; j >= 1; j--){
                bc[i][j] = (bc[i][j + 1] + 1LL * (str[j] - 'a' + 1) * pb[i][n + 1 - j]) %
                    mod;
            }
        }
    }

    inline int getfwhash(int l, int r, int i){ //[l,r] in 1 based indexing
        int ans = fw[i][r] - fw[i][l - 1];
        ans = 1LL * ans * ib[i][l - 1] % mod;

        if (ans < 0) ans += mod;

        return ans;
    }

    inline int getbchash(int l, int r, int i){ //[l,r] in 1 based indexing
        int ans = bc[i][l] - bc[i][r + 1];
        ans = 1LL * ans * ib[i][n - r] % mod;

        if (ans < 0) ans += mod;

        return ans;
    }

    inline bool equal(int l1, int r1, int l2, int r2){ //checks if s[l1,r1]==s[l2,r2] in
        one based indexing
        for (int i = 0; i < b; i++){
            int v1, v2;
```

```
            if (l1 <= r1) v1 = getfwhash(l1, r1, i);
            else v1 = getbchash(r1, l1, i);

            if (l2 <= r2) v2 = getfwhash(l2, r2, i);
            else v2 = getbchash(r2, l2, i);

            if (v1 != v2) return false;
        }
        return true;
    }

    inline bool pal(int l, int r){ //checks if a substring [l,r] is pallindrome
        return equal(l, r, r, l);
    }
};

int32_t main()
{
    string s;
    cin>>s;
    int n=s.length();
    Hash h;
    h.init(n,2,s);
    cout<<h.getfwhash(1,2,1)<<endl;
    cout<<h.getbchash(3,4,1)<<endl;
}


====================  trie.cpp  ====================
class Trie
{
public:
    static constexpr int ALPHA = 26;
    vector<array<int,ALPHA>> nxt;
    vector<int> cnt, pref;
    int nodes;

    Trie(int mx = 1e6+1)
    {
        nxt.reserve(mx);
        cnt.reserve(mx);
        pref.reserve(mx);
        newNode(); // create root
    }

    int newNode()
    {
        nxt.push_back({});
        cnt.push_back(0);
        pref.push_back(0);
        return nodes++;
    }

    // insert s with frequency f (use f=-1 to remove)
    void insert(const string &s, int f = 1)
    {
        int u = 0;
        for (char ch : s) {
            int c = ch - 'a';
            if (!nxt[u][c]) {
                nxt[u][c] = newNode();
            }
            u = nxt[u][c];
            pref[u] += f;
        }
```

```
        cnt[u] += f;
    }

    // exact string count
    int count(const string &s) const
    {
        int u = 0;
        for (char ch : s)
        {
            int c = ch - 'a';
            if (!nxt[u][c] || pref[nxt[u][c]] <= 0)
                return 0;
            u = nxt[u][c];
        }
        return cnt[u];
    }

    // count of strings with prefix s
    int prefCount(const string &s) const
    {
        int u = 0;
        for (char ch : s) {
            int c = ch - 'a';
            if (!nxt[u][c] || pref[nxt[u][c]] <= 0)
                return 0;
            u = nxt[u][c];
        }
        return pref[u];
    }

    // sum of prefCounts along path matching s
    int query(const string &s) const
    {
        // depends on use case

        // int u = 0;
        // int sum = 0;
        // for (char ch : s) {
        //     int c = ch - 'a';
        //     if (!nxt[u][c]) break;
        //     u = nxt[u][c];
        //     sum += pref[u];
        // }
        // return sum;
    }

    // remove s (decrement counts)
    void remove(const string &s)
    {
        insert(s, -1);
    }
};


====================  debug.hpp  ====================
#include<bits/stdc++.h>
using namespace std;
void __print(long long x) {cerr << x;}
void __print(int64_t x) {cerr << x;}
void __print(int32_t x) {cerr << x;}
void __print(unsigned x) {cerr << x;}
void __print(unsigned long x) {cerr << x;}
void __print(unsigned long long x) {cerr << x;}
void __print(float x) {cerr << x;}
void __print(double x) {cerr << x;}
```

```
void __print(long double x) {cerr << x;}
void __print(char x) {cerr << '\'' << x << '\'';}
void __print(const char *x) {cerr << '\"' << x << '\"';}
void __print(const string &x) {cerr << '\"' << x << '\"';}
void __print(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void __print(const pair<T, V> &x) {cerr << '{'; __print(x.first); cerr << ','; __print(x.second); cerr << '}';}
template<typename T>
void __print(const T &x) {int f = 0; cerr << '{'; for (auto &i: x) cerr << (f++ ? "," : ""), __print(i); cerr << "}";}
void _print() {cerr << "]\n";}
template <typename T, typename... V>
void _print(T t, V... v) {__print(t); if (sizeof...(v)) cerr << ", "; _print(v...);}


====================  segtree_beats.cpp  ====================

#include <algorithm>
#include <cassert>
#include <cstdint>
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstdint>

#define REP(i, n) for (int i = 0; (i) < (int)(n); ++ (i))
#define REP3(i, m, n) for (int i = (m); (i) < (int)(n); ++ (i))
#define REP_R(i, n) for (int i = (int)(n) - 1; (i) >= 0; -- (i))
#define REP3R(i, m, n) for (int i = (int)(n) - 1; (i) >= (int)(m); -- (i))
#define ALL(x) std::begin(x), std::end(x)

class segment_tree_beats {
    // MEMO: values for queries (max, min, lazy_add, and lazy_update) already apply to the current node; but not for children
    typedef struct {
        int64_t max;
        int64_t max_second;
        int max_count;
        int64_t min;
        int64_t min_second;
        int min_count;
        int64_t lazy_add;
        int64_t lazy_update;
        int64_t sum;
    } value_type;

    int n;
    std::vector<value_type> a;

public:
    segment_tree_beats() = default;
    template <class InputIterator>
    segment_tree_beats(InputIterator first, InputIterator last) {
        int n_ = std::distance(first, last);
        n = 1; while (n < n_) n *= 2;
        a.resize(2 * n - 1);
        REP (i, n_) {
            tag<UPDATE>(n - 1 + i, *(first + i));
        }
        REP3 (i, n_, n) {
            tag<UPDATE>(n - 1 + i, 0);
        }
        REP_R (i, n - 1) {
```

```
            update(i);
        }
    }

    void range_chmin(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<CHMIN>(0, 0, n, l, r, value);
    }
    void range_chmax(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<CHMAX>(0, 0, n, l, r, value);
    }
    void range_add(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<ADD>(0, 0, n, l, r, value);
    }
    void range_update(int l, int r, int64_t value) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        range_apply<UPDATE>(0, 0, n, l, r, value);
    }

    int64_t range_min(int l, int r) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        return range_get<MIN>(0, 0, n, l, r);
    }
    int64_t range_max(int l, int r) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        return range_get<MAX>(0, 0, n, l, r);
    }
    int64_t range_sum(int l, int r) {  // 0-based, [l, r)
        assert (0 <= l and l <= r and r <= n);
        return range_get<SUM>(0, 0, n, l, r);
    }

private:
    static constexpr char CHMIN = 0;
    static constexpr char CHMAX = 1;
    static constexpr char ADD = 2;
    static constexpr char UPDATE = 3;
    static constexpr char MIN = 10;
    static constexpr char MAX = 11;
    static constexpr char SUM = 12;

    template <char TYPE>
    void range_apply(int i, int il, int ir, int l, int r, int64_t g) {
        if (ir <= l or r <= il or break_condition<TYPE>(i, g)) {
            // break
        } else if (l <= il and ir <= r and tag_condition<TYPE>(i, g)) {
            tag<TYPE>(i, g);
        } else {
            pushdown(i);
            range_apply<TYPE>(2 * i + 1, il, (il + ir) / 2, l, r, g);
            range_apply<TYPE>(2 * i + 2, (il + ir) / 2, ir, l, r, g);
            update(i);
        }
    }
    template <char TYPE>
    inline bool break_condition(int i, int64_t g) {
        switch (TYPE) {
            case CHMIN: return a[i].max <= g;
            case CHMAX: return g <= a[i].min;
            case ADD: return false;
            case UPDATE: return false;
            default: assert (false);
        }
```

```
    }
    template <char TYPE>
    inline bool tag_condition(int i, int64_t g) {
        switch (TYPE) {
            case CHMIN: return a[i].max_second < g and g < a[i].max;
            case CHMAX: return a[i].min < g and g < a[i].min_second;
            case ADD: return true;
            case UPDATE: return true;
            default: assert (false);
        }
    }
    template <char TYPE>
    inline void tag(int i, int64_t g) {
        int length = n >> (32 - __builtin_clz(i + 1) - 1);
        if (TYPE == CHMIN) {
            if (a[i].max == a[i].min or g <= a[i].min) {
                tag<UPDATE>(i, g);
                return;
            }
            if (a[i].max != INT64_MIN) {
                a[i].sum -= a[i].max * a[i].max_count;
            }
            a[i].max = g;
            a[i].min_second = std::min(a[i].min_second, g);
            if (a[i].lazy_update != INT64_MAX) {
                a[i].lazy_update = std::min(a[i].lazy_update, g);
            }
            a[i].sum += g * a[i].max_count;
        } else if (TYPE == CHMAX) {
            if (a[i].max == a[i].min or a[i].max <= g) {
                tag<UPDATE>(i, g);
                return;
            }
            if (a[i].min != INT64_MAX) {
                a[i].sum -= a[i].min * a[i].min_count;
            }
            a[i].min = g;
            a[i].max_second = std::max(a[i].max_second, g);
            if (a[i].lazy_update != INT64_MAX) {
                a[i].lazy_update = std::max(a[i].lazy_update, g);
            }
            a[i].sum += g * a[i].min_count;
        } else if (TYPE == ADD) {
            if (a[i].max != INT64_MAX) {
                a[i].max += g;
            }
            if (a[i].max_second != INT64_MIN) {
                a[i].max_second += g;
            }
            if (a[i].min != INT64_MIN) {
                a[i].min += g;
            }
            if (a[i].min_second != INT64_MAX) {
                a[i].min_second += g;
            }
            a[i].lazy_add += g;
            if (a[i].lazy_update != INT64_MAX) {
                a[i].lazy_update += g;
            }
            a[i].sum += g * length;
        } else if (TYPE == UPDATE) {
            a[i].max = g;
            a[i].max_second = INT64_MIN;
            a[i].max_count = length;
            a[i].min = g;
```

```
            a[i].min_second = INT64_MAX;
            a[i].min_count = length;
            a[i].lazy_add = 0;
            a[i].lazy_update = INT64_MAX;
            a[i].sum = g * length;
        } else {
            assert (false);
        }
    }
    void pushdown(int i) {
        int l = 2 * i + 1;
        int r = 2 * i + 2;
        // update
        if (a[i].lazy_update != INT64_MAX) {
            tag<UPDATE>(l, a[i].lazy_update);
            tag<UPDATE>(r, a[i].lazy_update);
            a[i].lazy_update = INT64_MAX;
            return;
        }
        // add
        if (a[i].lazy_add != 0) {
            tag<ADD>(l, a[i].lazy_add);
            tag<ADD>(r, a[i].lazy_add);
            a[i].lazy_add = 0;
        }
        // chmin
        if (a[i].max < a[l].max) {
            tag<CHMIN>(l, a[i].max);
        }
        if (a[i].max < a[r].max) {
            tag<CHMIN>(r, a[i].max);
        }
        // chmax
        if (a[l].min < a[i].min) {
            tag<CHMAX>(l, a[i].min);
        }
        if (a[r].min < a[i].min) {
            tag<CHMAX>(r, a[i].min);
        }
    }
    void update(int i) {
        int l = 2 * i + 1;
        int r = 2 * i + 2;
        // chmin
        std::vector<int64_t> b { a[l].max, a[l].max_second, a[r].max, a[r].max_second };
        std::sort(b.rbegin(), b.rend());
        b.erase(std::unique(ALL(b)), b.end());
        a[i].max = b[0];
        a[i].max_second = b[1];
        a[i].max_count = (b[0] == a[l].max ? a[l].max_count : 0) + (b[0] == a[r].max ? a[r
            ].max_count : 0);
        // chmax
        std::vector<int64_t> c { a[l].min, a[l].min_second, a[r].min, a[r].min_second };
        std::sort(ALL(c));
        c.erase(std::unique(ALL(c)), c.end());
        a[i].min = c[0];
        a[i].min_second = c[1];
        a[i].min_count = (c[0] == a[l].min ? a[l].min_count : 0) + (c[0] == a[r].min ? a[r
            ].min_count : 0);
        // add
        a[i].lazy_add = 0;
        // update
        a[i].lazy_update = INT64_MAX;
        // sum
        a[i].sum = a[l].sum + a[r].sum;
```

```
    }

    template <char TYPE>
    int64_t range_get(int i, int il, int ir, int l, int r) {
        if (ir <= l or r <= il) {
            return 0;
        } else if (l <= il and ir <= r) {
            // base
            switch (TYPE) {
                case MIN: return a[i].min;
                case MAX: return a[i].max;
                case SUM: return a[i].sum;
                default: assert (false);
            }
        } else {
            pushdown(i);
            int64_t value_l = range_get<TYPE>(2 * i + 1, il, (il + ir) / 2, l, r);
            int64_t value_r = range_get<TYPE>(2 * i + 2, (il + ir) / 2, ir, l, r);
            // mult
            switch (TYPE) {
                case MIN: return std::min(value_l, value_r);
                case MAX: return std::max(value_l, value_r);
                case SUM: return value_l + value_r;
                default: assert (false);
            }
        }
    }
};
// 0-based, [l, r) in logn
int main() {
    int n, q; scanf("%d%d", &n, &q);

    std::vector<long long> a(n);
    for (int i = 0; i < n; i++) {
        scanf("%lld", &a[i]);
    }
    segment_tree_beats beats(ALL(a));

    for (int ph = 0; ph < q; ph++) {
        int ty, l, r; scanf("%d%d%d", &ty, &l, &r);
        if (ty == 0) {
            long long b; scanf("%lld", &b);
            beats.range_chmin(l, r, b);
        } else if (ty == 1) {
            long long b; scanf("%lld", &b);
            beats.range_chmax(l, r, b);
        } else if (ty == 2) {
            long long b; scanf("%lld", &b);
            beats.range_add(l, r, b);
        } else {
            long long sum = beats.range_sum(l, r);
            printf("%lld\n", sum);
        }
    }
    return 0;
}
```

Listing 1: C++ Templates