数据结构B

基于状压的线性 RMQ 算法

严格 $\mathcal{O}(N)$ 预处理，$\mathcal{O}(1)$ 查询。

```cpp
template<class T, class Cmp = less<T>> struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    vector<vector<T>> a;
    vector<T> pre, suf, ini;
    vector<u64> stk;
    RMQ() {}
    RMQ(const vector<T> &v) {
        init(v);
    }
    void init(const vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = __lg(M);
        a.assign(lg + 1, vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[__lg(s) + l])) {
                    s ^= 1ULL << __lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = __lg(r - l);
                ans = min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

珂朵莉树 (OD Tree)

区间赋值的数据结构都可以骗分，在数据随机的情况下，复杂度可以保证，时间复杂度：$\mathcal{O}(N\log\log N)$。

```
struct ODT {
    struct node {
        int l, r;
        mutable LL v;
        node(int l, int r = -1, LL v = 0) : l(l), r(r), v(v) {}
        bool operator<(const node &o) const {
            return l < o.l;
        }
    };
    set<node> s;
    ODT() {
        s.clear();
    }
    auto split(int pos) {
        auto it = s.lower_bound(node(pos));
        if (it != s.end() && it->l == pos) return it;
        it--;
        int l = it->l, r = it->r;
        LL v = it->v;
        s.erase(it);
        s.insert(node(l, pos - 1, v));
        return s.insert(node(pos, r, v)).first;
    }
    void assign(int l, int r, LL x) {
        auto itr = split(r + 1), itl = split(l);
        s.erase(itl, itr);
        s.insert(node(l, r, x));
    }
    void add(int l, int r, LL x) {
        auto itr = split(r + 1), itl = split(l);
        for (auto it = itl; it != itr; it++) {
            it->v += x;
        }
    }
    LL kth(int l, int r, int k) {
        vector<pair<LL, int>> a;
        auto itr = split(r + 1), itl = split(l);
        for (auto it = itl; it != itr; it++) {
            a.push_back(pair<LL, int>(it->v, it->r - it->l + 1));
        }
        sort(a.begin(), a.end());
        for (auto [val, len] : a) {
            k -= len;
            if (k <= 0) return val;
        }
    }
    LL power(LL a, int b, int mod) {
        a %= mod;
        LL res = 1;
        for (; b; b /= 2, a = a * a % mod) {
            if (b % 2) {
                res = res * a % mod;
            }
        }
        return res;
    }
    LL powersum(int l, int r, int x, int mod) {
        auto itr = split(r + 1), itl = split(l);
        LL ans = 0;
        for (auto it = itl; it != itr; it++) {
            ans = (ans + power(it->v, x, mod) * (it->r - it->l + 1) % mod) % mod;
        }
        return ans;
    }
};
```

pbds 扩展库实现平衡二叉树

记得加上相应的头文件，同时需要注意定义时的参数，一般只需要修改第三个参数：即定义的是大根堆还是小根堆。

> 附常见成员函数：

```
empty() / size()
insert(x) // 插入元素x
erase(x) // 删除元素/迭代器x
order_of_key(x) // 返回元素x的排名
find_by_order(x) // 返回排名为x的元素迭代器
lower_bound(x) / upper_bound(x) // 返回迭代器
join(Tree) // 将Tree树的全部元素并入当前的树
split(x, Tree) // 将大于x的元素放入Tree树
```

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using V = pair<int, int>;
tree<V, null_type, less<V>, rb_tree_tag, tree_order_statistics_node_update> ver;
map<int, int> dic;

int n; cin >> n;
for (int i = 1, op, x; i <= n; i++) {
    cin >> op >> x;
    if (op == 1) { // 插入一个元素x，允许重复
        ver.insert({x, ++dic[x]});
    } else if (op == 2) { // 删除元素x，若有重复，则任意删除一个
        ver.erase({x, dic[x]--});
    } else if (op == 3) { // 查询元素x的排名（排名定义为比当前数小的数的个数+1）
        cout << ver.order_of_key({x, 1}) + 1 << endl;
    } else if (op == 4) { // 查询排名为x的元素
        cout << ver.find_by_order(--x)->first << endl;
    } else if (op == 5) { // 查询元素x的前驱
        int idx = ver.order_of_key({x, 1}) - 1; // 无论x存不存在，idx都代表x的位置，需要-1
        cout << ver.find_by_order(idx)->first << endl;
    } else if (op == 6) { // 查询元素x的后继
        int idx = ver.order_of_key( {x, dic[x]}); // 如果x不存在，那么idx就是x的后继
        if (ver.find({x, 1}) != ver.end()) idx++; // 如果x存在，那么idx是x的位置，需要+1
        cout << ver.find_by_order(idx)->first << endl;
    }
}
```

vector 模拟实现平衡二叉树

```
#define ALL(x) x.begin(), x.end()
#define pre lower_bound
#define suf upper_bound
int n; cin >> n;
vector<int> ver;
for (int i = 1, op, x; i <= n; i++) {
    cin >> op >> x;
    if (op == 1) ver.insert(pre(ALL(ver), x), x);
    if (op == 2) ver.erase(pre(ALL(ver), x));
    if (op == 3) cout << pre(ALL(ver), x) - ver.begin() + 1 << endl;
    if (op == 4) cout << ver[x - 1] << endl;
    if (op == 5) cout << ver[pre(ALL(ver), x) - ver.begin() - 1] << endl;
    if (op == 6) cout << ver[suf(ALL(ver), x) - ver.begin()] << endl;
}
```

取模类

集成了常见的取模四则运算，运算速度与手动取模相差无几，效率极高。

```
using i64 = long long;

template<class T> constexpr T mypow(T n, i64 k) {
    T r = 1;
    for (; k; k /= 2, n *= n) {
        if (k % 2) {
            r *= n;
        }
    }
    return r;
}

template<class T> constexpr T power(int n) {
    return mypow(T(2), n);
}

template<const int &MOD> struct Zmod {
    int x;
    Zmod(signed x = 0) : x(norm(x % MOD)) {}
    Zmod(i64 x) : x(norm(x % MOD)) {}

    constexpr int norm(int x) const noexcept {
        if (x < 0) [[unlikely]] {
            x += MOD;
        }
        if (x >= MOD) [[unlikely]] {
            x -= MOD;
        }
        return x;
    }
    explicit operator int() const {
        return x;
    }
    constexpr int val() const {
        return x;
    }
    constexpr Zmod operator-() const {
        Zmod val = norm(MOD - x);
        return val;
    }
    constexpr Zmod inv() const {
        assert(x != 0);
        return mypow(*this, MOD - 2);
    }
    friend constexpr auto &operator>>(istream &in, Zmod &j) {
        int v;
        in >> v;
        j = Zmod(v);
        return in;
    }
    friend constexpr auto &operator<<(ostream &o, const Zmod &j) {
        return o << j.val();
    }
    constexpr Zmod &operator++() {
        x = norm(x + 1);
        return *this;
    }
    constexpr Zmod &operator--() {
        x = norm(x - 1);
        return *this;
    }
    constexpr Zmod operator++(signed) {
        Zmod res = *this;
        ++*this;
        return res;
    }
    constexpr Zmod operator--(signed) {
        Zmod res = *this;
        --*this;
        return res;
    }
    constexpr Zmod &operator+=(const Zmod &i) {
        x = norm(x + i.x);
        return *this;
    }
    constexpr Zmod &operator-=(const Zmod &i) {
        x = norm(x - i.x);
        return *this;
    }
    constexpr Zmod &operator*=(const Zmod &i) {
        x = i64(x) * i.x % MOD;
        return *this;
    }
    constexpr Zmod &operator/=(const Zmod &i) {
        return *this *= i.inv();
    }
    constexpr Zmod &operator%=(const int &i) {
        return x %= i, *this;
    }
    friend constexpr Zmod operator+(const Zmod i, const Zmod j) {
        return Zmod(i) += j;
    }
    friend constexpr Zmod operator-(const Zmod i, const Zmod j) {
        return Zmod(i) -= j;
    }
    friend constexpr Zmod operator*(const Zmod i, const Zmod j) {
        return Zmod(i) *= j;
    }
    friend constexpr Zmod operator/(const Zmod i, const Zmod j) {
```

```cpp
        return Zmod(i) /= j;
    }
    friend constexpr Zmod operator%(const Zmod i, const int j) {
        return Zmod(i) %= j;
    }
    friend constexpr bool operator==(const Zmod i, const Zmod j) {
        return i.val() == j.val();
    }
    friend constexpr bool operator!=(const Zmod i, const Zmod j) {
        return i.val() != j.val();
    }
    friend constexpr bool operator<(const Zmod i, const Zmod j) {
        return i.val() < j.val();
    }
    friend constexpr bool operator>(const Zmod i, const Zmod j) {
        return i.val() > j.val();
    }
};

int MOD[] = {998244353, 1000000007};
using Z = Zmod<MOD[1]>;
```

分数运算类

定义了分数的四则运算，如果需要处理浮点数，那么需要将函数中的 `gcd` 运算替换为 `fgcd` 。

```cpp
template<class T> struct Frac {
    T x, y;
    Frac() : Frac(0, 1) {}
    Frac(T x_) : Frac(x_, 1) {}
    Frac(T x_, T y_) : x(x_), y(y_) {
        if (y < 0) {
            y = -y;
            x = -x;
        }
    }

    constexpr double val() const {
        return 1. * x / y;
    }
    constexpr Frac norm() const { // 调整符号、转化为最简形式
        T p = gcd(x, y);
        return {x / p, y / p};
    }
    friend constexpr auto &operator<<(ostream &o, const Frac &j) {
        T p = gcd(j.x, j.y);
        if (j.y == p) {
            return o << j.x / p;
        } else {
            return o << j.x / p << "/" << j.y / p;
        }
    }
    constexpr Frac &operator/=(const Frac &i) {
        x *= i.y;
        y *= i.x;
        if (y < 0) {
            x = -x;
            y = -y;
        }
        return *this;
    }
    constexpr Frac &operator+=(const Frac &i) { return x = x * i.y + y * i.x, y *= i.y, *this; }
    constexpr Frac &operator-=(const Frac &i) { return x = x * i.y - y * i.x, y *= i.y, *this; }
    constexpr Frac &operator*=(const Frac &i) { return x *= i.x, y *= i.y, *this; }
    friend constexpr Frac operator+(const Frac i, const Frac j) { return i += j; }
    friend constexpr Frac operator-(const Frac i, const Frac j) { return i -= j; }
    friend constexpr Frac operator*(const Frac i, const Frac j) { return i *= j; }
    friend constexpr Frac operator/(const Frac i, const Frac j) { return i /= j; }
    friend constexpr Frac operator-(const Frac i) { return Frac(-i.x, i.y); }
    friend constexpr bool operator<(const Frac i, const Frac j) { return i.x * j.y < i.y * j.x; }
    friend constexpr bool operator>(const Frac i, const Frac j) { return i.x * j.y > i.y * j.x; }
    friend constexpr bool operator==(const Frac i, const Frac j) { return i.x * j.y == i.y * j.x; }
    friend constexpr bool operator!=(const Frac i, const Frac j) { return i.x * j.y != i.y * j.x; }
};
```

大整数类（高精度计算）

```
const int base = 1000000000;
const int base_digits = 9; // 分解为九个数位一个数字
struct bigint {
    vector<int> a;
    int sign;

    bigint() : sign(1) {}
    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }
    bigint(long long v) {
        *this = v;
    }
    bigint(const string &s) {
        read(s);
    }
    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }
    void operator=(long long v) {
        a.clear();
        sign = 1;
        if (v < 0) sign = -1, v = -v;
        for (; v > 0; v = v / base) {
            a.push_back(v % base);
        }
    }

    // 基础加减乘除
    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;
            for (int i = 0, carry = 0; i < (int)max(a.size(), v.a.size()) || carry; ++i) {
                if (i == (int)res.a.size()) {
                    res.a.push_back(0);
                }
                res.a[i] += carry + (i < (int)a.size() ? a[i] : 0);
                carry = res.a[i] >= base;
                if (carry) {
                    res.a[i] -= base;
                }
            }
            return res;
        }
        return *this - (-v);
    }
    bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                bigint res = *this;
                for (int i = 0, carry = 0; i < (int)v.a.size() || carry; ++i) {
                    res.a[i] -= carry + (i < (int)v.a.size() ? v.a[i] : 0);
                    carry = res.a[i] < 0;
                    if (carry) {
                        res.a[i] += base;
                    }
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }
    void operator*=(int v) {
        check(v);
        for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
            if (i == (int)a.size()) {
                a.push_back(0);
            }
            long long cur = a[i] * (long long)v + carry;
            carry = (int)(cur / base);
            a[i] = (int)(cur % base);
        }
        trim();
    }
    void operator/=(int v) {
        check(v);
        for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
            long long cur = a[i] + rem * (long long)base;
            a[i] = (int)(cur / v);
            rem = (int)(cur % v);
        }
        trim();
    }
    int operator%(int v) const {
        if (v < 0) {
            v = -v;
        }
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i) {
            m = (a[i] + m * (long long)base) % v;
        }
        return m * sign;
    }

    void operator+=(const bigint &v) {
        *this = *this + v;
    }
    void operator-=(const bigint &v) {
        *this = *this - v;
    }
    bigint operator*(int v) const {
        bigint res = *this;
        res *= v;
        return res;
    }
    bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
    }
    void operator%=(const int &v) {
        *this = *this % v;
    }

    bool operator<(const bigint &v) const {
        if (sign != v.sign) return sign < v.sign;
        if (a.size() != v.a.size()) return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i]) return a[i] * sign < v.a[i] * sign;
        return false;
    }
```

```cpp
    bool operator>(const bigint &v) const {
        return v < *this;
    }
    bool operator<=(const bigint &v) const {
        return !(v < *this);
    }
    bool operator>=(const bigint &v) const {
        return !(*this < v);
    }
    bool operator==(const bigint &v) const {
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const bigint &v) const {
        return *this < v || v < *this;
    }

    bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
    }
    void check(int v) { // 检查输入的是否为负数
        if (v < 0) {
            sign = -sign;
            v = -v;
        }
    }
    void trim() { // 去除前导零
        while (!a.empty() && !a.back()) a.pop_back();
        if (a.empty()) sign = 1;
    }
    bool isZero() const { // 判断是否等于零
        return a.empty() || (a.size() == 1 && !a[0]);
    }
    friend bigint gcd(const bigint &a, const bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const bigint &b) {
        return a / gcd(a, b) * b;
    }
    void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int)s.size() && (s[pos] == '-' || s[pos] == '+')) {
            if (s[pos] == '-') sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++) x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }
    friend istream &operator>>(istream &stream, bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }
    friend ostream &operator<<(ostream &stream, const bigint &v) {
        if (v.sign == -1) stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int)v.a.size() - 2; i >= 0; --i)
            stream << setw(base_digits) << setfill('0') << v.a[i];
        return stream;
    }

    /* 大整数乘除大整数部分 */
    typedef vector<long long> vll;
    bigint operator*(const bigint &v) const { // 大整数乘大整数
        vector<int> a6 = convert_base(this->a, base_digits, 6);
        vector<int> b6 = convert_base(v.a, base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size()) a.push_back(0);
        while (b.size() < a.size()) b.push_back(0);
        while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int)c.size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int)(cur % 1000000));
            carry = (int)(cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits);
        res.trim();
        return res;
    }
    friend pair<bigint, bigint> divmod(const bigint &a1,
                                       const bigint &b1) { // 大整数除大整数，同时返回答案与余数
        int norm = base / (b1.a.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.a.resize(a.a.size());
        for (int i = a.a.size() - 1; i >= 0; i--) {
            r *= base;
            r += a.a[i];
            int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
            int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
            int d = ((long long)base * s1 + s2) / b.a.back();
            r -= b * d;
            while (r < 0) r += b, --d;
            q.a[i] = d;
        }
        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
    }
    static vector<int> convert_base(const vector<int> &a, int old_digits, int new_digits) {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int)p.size(); i++) p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int)a.size(); i++) {
            cur += a[i] * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back((int)(cur % p[new_digits]));
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int)cur);
        while (!res.empty() && !res.back()) res.pop_back();
        return res;
    }
    static vll karatsubaMultiply(const vll &a, const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    res[i + j] += a[i] * b[j];
                }
            }
            return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++) a2[i] += a1[i];
        for (int i = 0; i < k; i++) b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int)a1b1.size(); i++) r[i] -= a1b1[i];
        for (int i = 0; i < (int)a2b2.size(); i++) r[i] -= a2b2[i];

        for (int i = 0; i < (int)r.size(); i++) res[i + k] += r[i];
        for (int i = 0; i < (int)a1b1.size(); i++) res[i] += a1b1[i];
        for (int i = 0; i < (int)a2b2.size(); i++) res[i + n] += a2b2[i];
        return res;
    }

    void operator*=(const bigint &v) {
        *this = *this * v;
    }
    bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
    }
    void operator/=(const bigint &v) {
        *this = *this / v;
    }
    bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
    }
    void operator%=(const bigint &v) {
        *this = *this % v;
    }
};
```

常见结论

题意：（区间移位问题）要求将整个序列左移/右移若干个位置，例如，原序列为 $A = (a_1, a_2, \ldots, a_n)$ ，右移 x 位后变为 $A = (a_{x+1}, a_{x+2}, \ldots, a_n, a_1, a_2, \ldots, a_x)$ 。

区间的端点只是一个数字，即使被改变了，通过一定的转换也能够还原，所以我们可以 $\mathcal{O}(1)$ 解决这一问题。为了方便计算，我们规定下标从 0 开始，即整个线段的区间为 $[0, n)$ ，随后，使用一个偏移量 `shift` 记录。使用 `shift = (shift + x) % n;` 更新偏移量；此后的区间查询/修改前，再将坐标偏移回去即可，下方代码使用区间修改作为示例。

```
cin >> l >> r >> x;
l--; // 坐标修改为 0 开始
r--;
l = (l + shift) % n; // 偏移
r = (r + shift) % n;
if (l > r) { // 区间分离则分别操作
    segt.modify(l, n - 1, x);
    segt.modify(0, r, x);
} else {
    segt.modify(l, r, x);
}
```

常见例题

题意：（带修莫队 - 维护队列）要求能够处理以下操作：

- `'Q' l r`：询问区间 $[l,r]$ 有几个颜色；
- `'R' idx w`：将下标 **idx** 的颜色修改为 **w**。

输入格式为：第一行 n 和 q $(1 \le n, q \le 133333)$ 分别代表区间长度和操作数量；第二行 n 个整数 $a_1, a_2 \dots, a_n$ $(1 \le a_i \le 10^6)$ 代表初始颜色；随后 q 行为具体操作。

```cpp
const int N = 1e6 + 7;
signed main() {
    int n, q;
    cin >> n >> q;
    vector<int> w(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> w[i];
    }

    vector<array<int, 4>> query = {{}}; // {左区间, 右区间, 累计修改次数, 下标}
    vector<array<int, 2>> modify = {{}}; // {修改的值, 修改的元素下标}
    for (int i = 1; i <= q; i++) {
        char op;
        cin >> op;
        if (op == 'Q') {
            int l, r;
            cin >> l >> r;
            query.push_back({l, r, (int)modify.size() - 1, (int)query.size()});
        } else {
            int idx, w;
            cin >> idx >> w;
            modify.push_back({w, idx});
        }
    }

    int Knum = 2154; // 计算块长
    vector<int> K(n + 1);
    for (int i = 1; i <= n; i++) { // 固定块长
        K[i] = (i - 1) / Knum + 1;
    }
    sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
        if (K[x[0]] != K[y[0]]) return x[0] < y[0];
        if (K[x[1]] != K[y[1]]) return x[1] < y[1];
        return x[3] < y[3];
    });

    int l = 1, r = 0, val = 0;
    int t = 0; // 累计修改次数
    vector<int> ans(query.size()), cnt(N);
    for (int i = 1; i < query.size(); i++) {
        auto [ql, qr, qt, id] = query[i];
        auto add = [&](int x) -> void {
            if (cnt[x] == 0) ++ val;
            ++ cnt[x];
        };
        auto del = [&](int x) -> void {
            -- cnt[x];
            if (cnt[x] == 0) -- val;
        };
        auto time = [&](int x, int l, int r) -> void {
            if (l <= modify[x][1] && modify[x][1] <= r) { //当修改的位置在询问期间内部时才会改变num的值
                del(w[modify[x][1]]);
                add(modify[x][0]);
            }
            swap(w[modify[x][1]], modify[x][0]); //直接交换修改数组的值与原始值，减少额外的数组开销，且方便复原
        };
        while (l > ql) add(w[--l]);
        while (r < qr) add(w[++r]);
        while (l < ql) del(w[l++]);
        while (r > qr) del(w[r--]);
        while (t < qt) time(++t, ql, qr);
        while (t > qt) time(t--, ql, qr);
        ans[id] = val;
    }
    for (int i = 1; i < ans.size(); i++) {
        cout << ans[i] << endl;
    }
}
```

/END/