

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。

```

1  template<class T, class Cmp = less<T>> struct RMQ {
2      const Cmp cmp = Cmp();
3      static constexpr unsigned B = 64;
4      using u64 = unsigned long long;
5      int n;
6      vector<vector<T>> a;
7      vector<T> pre, suf, ini;
8      vector<u64> stk;
9      RMQ() {}
10     RMQ(const vector<T> &v) {
11         init(v);
12     }
13     void init(const vector<T> &v) {
14         n = v.size();
15         pre = suf = ini = v;
16         stk.resize(n);
17         if (!n) {
18             return;
19         }
20         const int M = (n - 1) / B + 1;
21         const int lg = __lg(M);
22         a.assign(lg + 1, vector<T>(M));
23         for (int i = 0; i < M; i++) {
24             a[0][i] = v[i * B];
25             for (int j = 1; j < B && i * B + j < n; j++) {
26                 a[0][i] = min(a[0][i], v[i * B + j], cmp);
27             }
28         }
29         for (int i = 1; i < n; i++) {
30             if (i % B) {
31                 pre[i] = min(pre[i], pre[i - 1], cmp);
32             }
33         }
34         for (int i = n - 2; i >= 0; i--) {
35             if (i % B != B - 1) {
36                 suf[i] = min(suf[i], suf[i + 1], cmp);
37             }
38         }
39         for (int j = 0; j < lg; j++) {
40             for (int i = 0; i + (2 << j) <= M; i++) {
41                 a[j + 1][i] = min(a[j][i], a[j][i + (1 << j)], cmp);
42             }
43         }
44         for (int i = 0; i < M; i++) {
45             const int l = i * B;
46             const int r = min(1U * n, l + B);

```

```

47         u64 s = 0;
48         for (int j = 1; j < r; j++) {
49             while (s && cmp(v[j], v[__lg(s) + 1])) {
50                 s ^= 1ULL << __lg(s);
51             }
52             s |= 1ULL << (j - 1);
53             stk[j] = s;
54         }
55     }
56 }
57 T operator()(int l, int r) {
58     if (l / B != (r - 1) / B) {
59         T ans = min(suf[l], pre[r - 1], cmp);
60         l = l / B + 1;
61         r = r / B;
62         if (l < r) {
63             int k = __lg(r - l);
64             ans = min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
65         }
66         return ans;
67     } else {
68         int x = B * (l / B);
69         return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + 1];
70     }
71 }
72 };

```

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： $\mathcal{O}(N \log \log N)$ 。

```

1  struct ODT {
2      struct node {
3          int l, r;
4          mutable LL v;
5          node(int l, int r = -1, LL v = 0) : l(l), r(r), v(v) {}
6          bool operator<(const node &o) const {
7              return l < o.l;
8          }
9      };
10     set<node> s;
11     ODT() {
12         s.clear();
13     }
14     auto split(int pos) {
15         auto it = s.lower_bound(node(pos));
16         if (it != s.end() && it->l == pos) return it;
17         it--;
18         int l = it->l, r = it->r;
19         LL v = it->v;
20         s.erase(it);
21         s.insert(node(l, pos - 1, v));

```

```

22     return s.insert(node(pos, r, v)).first;
23 }
24 void assign(int l, int r, LL x) {
25     auto itr = split(r + 1), itl = split(l);
26     s.erase(itl, itr);
27     s.insert(node(l, r, x));
28 }
29 void add(int l, int r, LL x) {
30     auto itr = split(r + 1), itl = split(l);
31     for (auto it = itl; it != itr; it++) {
32         it->v += x;
33     }
34 }
35 LL kth(int l, int r, int k) {
36     vector<pair<LL, int>> a;
37     auto itr = split(r + 1), itl = split(l);
38     for (auto it = itl; it != itr; it++) {
39         a.push_back(pair<LL, int>(it->v, it->r - it->l + 1));
40     }
41     sort(a.begin(), a.end());
42     for (auto [val, len] : a) {
43         k -= len;
44         if (k <= 0) return val;
45     }
46 }
47 LL power(LL a, int b, int mod) {
48     a %= mod;
49     LL res = 1;
50     for (; b; b /= 2, a = a * a % mod) {
51         if (b % 2) {
52             res = res * a % mod;
53         }
54     }
55     return res;
56 }
57 LL powersum(int l, int r, int x, int mod) {
58     auto itr = split(r + 1), itl = split(l);
59     LL ans = 0;
60     for (auto it = itl; it != itr; it++) {
61         ans = (ans + power(it->v, x, mod) * (it->r - it->l + 1) % mod) % mod;
62     }
63     return ans;
64 }
65 };

```

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：

```

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

```

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 using V = pair<int, int>;
4 tree<V, null_type, less<V>, rb_tree_tag, tree_order_statistics_node_update> ver;
5 map<int, int> dic;
6
7 int n; cin >> n;
8 for (int i = 1, op, x; i <= n; i++) {
9     cin >> op >> x;
10    if (op == 1) { // 插入一个元素x, 允许重复
11        ver.insert({x, ++dic[x]});
12    } else if (op == 2) { // 删除元素x, 若有重复, 则任意删除一个
13        ver.erase({x, dic[x]--});
14    } 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)
15        cout << ver.order_of_key({x, 1}) + 1 << endl;
16    } else if (op == 4) { // 查询排名为x的元素
17        cout << ver.find_by_order(--x)->first << endl;
18    } else if (op == 5) { // 查询元素x的前驱
19        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
20        cout << ver.find_by_order(idx)->first << endl;
21    } else if (op == 6) { // 查询元素x的后继
22        int idx = ver.order_of_key({x, dic[x]}); // 如果x不存在, 那么idx就是x的后继
23        if (ver.find({x, 1}) != ver.end()) idx++; // 如果x存在, 那么idx是x的位置, 需要+1
24        cout << ver.find_by_order(idx)->first << endl;
25    }
26 }

```

vector 模拟实现平衡二叉树

```

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

```

14 | }

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。

```

1  using i64 = long long;
2
3  template<class T> constexpr T mypow(T n, i64 k) {
4      T r = 1;
5      for (; k; k /= 2, n *= n) {
6          if (k % 2) {
7              r *= n;
8          }
9      }
10     return r;
11 }
12
13 template<class T> constexpr T power(int n) {
14     return mypow(T(2), n);
15 }
16
17 template<const int &MOD> struct Zmod {
18     int x;
19     Zmod(signed x = 0) : x(norm(x % MOD)) {}
20     Zmod(i64 x) : x(norm(x % MOD)) {}
21
22     constexpr int norm(int x) const noexcept {
23         if (x < 0) [[unlikely]] {
24             x += MOD;
25         }
26         if (x >= MOD) [[unlikely]] {
27             x -= MOD;
28         }
29         return x;
30     }
31     explicit operator int() const {
32         return x;
33     }
34     constexpr int val() const {
35         return x;
36     }
37     constexpr Zmod operator-() const {
38         Zmod val = norm(MOD - x);
39         return val;
40     }
41     constexpr Zmod inv() const {
42         assert(x != 0);
43         return mypow(*this, MOD - 2);
44     }
45     friend constexpr auto &operator>>(istream &in, Zmod &j) {
46         int v;

```

```

47     in >> v;
48     j = Zmod(v);
49     return in;
50 }
51 friend constexpr auto &operator<<(ostream &o, const Zmod &j) {
52     return o << j.val();
53 }
54 constexpr Zmod &operator++() {
55     x = norm(x + 1);
56     return *this;
57 }
58 constexpr Zmod &operator--() {
59     x = norm(x - 1);
60     return *this;
61 }
62 constexpr Zmod operator++(signed) {
63     Zmod res = *this;
64     ++*this;
65     return res;
66 }
67 constexpr Zmod operator--(signed) {
68     Zmod res = *this;
69     --*this;
70     return res;
71 }
72 constexpr Zmod &operator+=(const Zmod &i) {
73     x = norm(x + i.x);
74     return *this;
75 }
76 constexpr Zmod &operator-=(const Zmod &i) {
77     x = norm(x - i.x);
78     return *this;
79 }
80 constexpr Zmod &operator*=(const Zmod &i) {
81     x = i64(x) * i.x % MOD;
82     return *this;
83 }
84 constexpr Zmod &operator/=(const Zmod &i) {
85     return *this *= i.inv();
86 }
87 constexpr Zmod &operator%=(const int &i) {
88     return x %= i, *this;
89 }
90 friend constexpr Zmod operator+(const Zmod i, const Zmod j) {
91     return Zmod(i) += j;
92 }
93 friend constexpr Zmod operator-(const Zmod i, const Zmod j) {
94     return Zmod(i) -= j;
95 }
96 friend constexpr Zmod operator*(const Zmod i, const Zmod j) {
97     return Zmod(i) *= j;
98 }
99 friend constexpr Zmod operator/(const Zmod i, const Zmod j) {

```

```

100     return Zmod(i) /= j;
101 }
102 friend constexpr Zmod operator%(const Zmod i, const int j) {
103     return Zmod(i) %= j;
104 }
105 friend constexpr bool operator==(const Zmod i, const Zmod j) {
106     return i.val() == j.val();
107 }
108 friend constexpr bool operator!=(const Zmod i, const Zmod j) {
109     return i.val() != j.val();
110 }
111 friend constexpr bool operator<(const Zmod i, const Zmod j) {
112     return i.val() < j.val();
113 }
114 friend constexpr bool operator>(const Zmod i, const Zmod j) {
115     return i.val() > j.val();
116 }
117 };
118
119 int MOD[] = {998244353, 1000000007};
120 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`。

```

1  template<class T> struct Frac {
2      T x, y;
3      Frac() : Frac(0, 1) {}
4      Frac(T x_) : Frac(x_, 1) {}
5      Frac(T x_, T y_) : x(x_), y(y_) {
6          if (y < 0) {
7              y = -y;
8              x = -x;
9          }
10     }
11
12     constexpr double val() const {
13         return 1. * x / y;
14     }
15     constexpr Frac norm() const { // 调整符号、转化为最简形式
16         T p = gcd(x, y);
17         return {x / p, y / p};
18     }
19     friend constexpr auto &operator<<(ostream &o, const Frac &j) {
20         T p = gcd(j.x, j.y);
21         if (j.y == p) {
22             return o << j.x / p;
23         } else {
24             return o << j.x / p << "/" << j.y / p;
25         }
26     }

```

```

27     constexpr Frac &operator/=(const Frac &i) {
28         x *= i.y;
29         y *= i.x;
30         if (y < 0) {
31             x = -x;
32             y = -y;
33         }
34         return *this;
35     }
36     constexpr Frac &operator+=(const Frac &i) { return x = x * i.y + y * i.x, y *= i.y,
*this; }
37     constexpr Frac &operator-=(const Frac &i) { return x = x * i.y - y * i.x, y *= i.y,
*this; }
38     constexpr Frac &operator*=(const Frac &i) { return x *= i.x, y *= i.y, *this; }
39     friend constexpr Frac operator+(const Frac i, const Frac j) { return i += j; }
40     friend constexpr Frac operator-(const Frac i, const Frac j) { return i -= j; }
41     friend constexpr Frac operator*(const Frac i, const Frac j) { return i *= j; }
42     friend constexpr Frac operator/(const Frac i, const Frac j) { return i /= j; }
43     friend constexpr Frac operator~(const Frac i) { return Frac(-i.x, i.y); }
44     friend constexpr bool operator<(const Frac i, const Frac j) { return i.x * j.y <
i.y * j.x; }
45     friend constexpr bool operator>(const Frac i, const Frac j) { return i.x * j.y >
i.y * j.x; }
46     friend constexpr bool operator==(const Frac i, const Frac j) { return i.x * j.y ==
i.y * j.x; }
47     friend constexpr bool operator!=(const Frac i, const Frac j) { return i.x * j.y !=
i.y * j.x; }
48 };

```

大整数类（高精度计算）

```

1  const int base = 1000000000;
2  const int base_digits = 9; // 分解为九个数位一个数字
3  struct bigint {
4      vector<int> a;
5      int sign;
6
7      bigint() : sign(1) {}
8      bigint operator-() const {
9          bigint res = *this;
10         res.sign = -sign;
11         return res;
12     }
13     bigint(long long v) {
14         *this = v;
15     }
16     bigint(const string &s) {
17         read(s);
18     }
19     void operator=(const bigint &v) {
20         sign = v.sign;
21         a = v.a;

```



```

22     }
23     void operator=(long long v) {
24         a.clear();
25         sign = 1;
26         if (v < 0) sign = -1, v = -v;
27         for (; v > 0; v = v / base) {
28             a.push_back(v % base);
29         }
30     }
31
32     // 基础加减乘除
33     bigint operator+(const bigint &v) const {
34         if (sign == v.sign) {
35             bigint res = v;
36             for (int i = 0, carry = 0; i < (int)max(a.size(), v.a.size()) || carry;
37 ++i) {
38                 if (i == (int)res.a.size()) {
39                     res.a.push_back(0);
40                 }
41                 res.a[i] += carry + (i < (int)a.size() ? a[i] : 0);
42                 carry = res.a[i] >= base;
43                 if (carry) {
44                     res.a[i] -= base;
45                 }
46             }
47             return res;
48         }
49         return *this - (-v);
50     }
51     bigint operator-(const bigint &v) const {
52         if (sign == v.sign) {
53             if (abs() >= v.abs()) {
54                 bigint res = *this;
55                 for (int i = 0, carry = 0; i < (int)v.a.size() || carry; ++i) {
56                     res.a[i] -= carry + (i < (int)v.a.size() ? v.a[i] : 0);
57                     carry = res.a[i] < 0;
58                     if (carry) {
59                         res.a[i] += base;
60                     }
61                 }
62                 res.trim();
63                 return res;
64             }
65             return -(v - *this);
66         }
67         return *this + (-v);
68     }
69     void operator*=(int v) {
70         check(v);
71         for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
72             if (i == (int)a.size()) {
73                 a.push_back(0);

```

```

74         long long cur = a[i] * (long long)v + carry;
75         carry = (int)(cur / base);
76         a[i] = (int)(cur % base);
77     }
78     trim();
79 }
80 void operator/=(int v) {
81     check(v);
82     for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
83         long long cur = a[i] + rem * (long long)base;
84         a[i] = (int)(cur / v);
85         rem = (int)(cur % v);
86     }
87     trim();
88 }
89 int operator%(int v) const {
90     if (v < 0) {
91         v = -v;
92     }
93     int m = 0;
94     for (int i = a.size() - 1; i >= 0; --i) {
95         m = (a[i] + m * (long long)base) % v;
96     }
97     return m * sign;
98 }
99
100 void operator+=(const bigint &v) {
101     *this = *this + v;
102 }
103 void operator-=(const bigint &v) {
104     *this = *this - v;
105 }
106 bigint operator*(int v) const {
107     bigint res = *this;
108     res *= v;
109     return res;
110 }
111 bigint operator/(int v) const {
112     bigint res = *this;
113     res /= v;
114     return res;
115 }
116 void operator%=(const int &v) {
117     *this = *this % v;
118 }
119
120 bool operator<(const bigint &v) const {
121     if (sign != v.sign) return sign < v.sign;
122     if (a.size() != v.a.size()) return a.size() * sign < v.a.size() * v.sign;
123     for (int i = a.size() - 1; i >= 0; i--)
124         if (a[i] != v.a[i]) return a[i] * sign < v.a[i] * sign;
125     return false;
126 }

```

```

127     bool operator>(const bigint &v) const {
128         return v < *this;
129     }
130     bool operator<=(const bigint &v) const {
131         return !(v < *this);
132     }
133     bool operator>=(const bigint &v) const {
134         return !(*this < v);
135     }
136     bool operator==(const bigint &v) const {
137         return !(*this < v) && !(v < *this);
138     }
139     bool operator!=(const bigint &v) const {
140         return *this < v || v < *this;
141     }
142
143     bigint abs() const {
144         bigint res = *this;
145         res.sign *= res.sign;
146         return res;
147     }
148     void check(int v) { // 检查输入的是否为负数
149         if (v < 0) {
150             sign = -sign;
151             v = -v;
152         }
153     }
154     void trim() { // 去除前导零
155         while (!a.empty() && !a.back()) a.pop_back();
156         if (a.empty()) sign = 1;
157     }
158     bool isZero() const { // 判断是否等于零
159         return a.empty() || (a.size() == 1 && !a[0]);
160     }
161     friend bigint gcd(const bigint &a, const bigint &b) {
162         return b.isZero() ? a : gcd(b, a % b);
163     }
164     friend bigint lcm(const bigint &a, const bigint &b) {
165         return a / gcd(a, b) * b;
166     }
167     void read(const string &s) {
168         sign = 1;
169         a.clear();
170         int pos = 0;
171         while (pos < (int)s.size() && (s[pos] == '-' || s[pos] == '+')) {
172             if (s[pos] == '-') sign = -sign;
173             ++pos;
174         }
175         for (int i = s.size() - 1; i >= pos; i -= base_digits) {
176             int x = 0;
177             for (int j = max(pos, i - base_digits + 1); j <= i; j++) x = x * 10 + s[j]
178 - '0';
179             a.push_back(x);

```

```

179     }
180     trim();
181 }
182 friend istream &operator>>(istream &stream, bigint &v) {
183     string s;
184     stream >> s;
185     v.read(s);
186     return stream;
187 }
188 friend ostream &operator<<(ostream &stream, const bigint &v) {
189     if (v.sign == -1) stream << '-';
190     stream << (v.a.empty() ? 0 : v.a.back());
191     for (int i = (int)v.a.size() - 2; i >= 0; --i)
192         stream << setw(base_digits) << setfill('0') << v.a[i];
193     return stream;
194 }
195
196 /* 大整数乘除大整数部分 */
197 typedef vector<long long> vll;
198 bigint operator*(const bigint &v) const { // 大整数乘大整数
199     vector<int> a6 = convert_base(this->a, base_digits, 6);
200     vector<int> b6 = convert_base(v.a, base_digits, 6);
201     vll a(a6.begin(), a6.end());
202     vll b(b6.begin(), b6.end());
203     while (a.size() < b.size()) a.push_back(0);
204     while (b.size() < a.size()) b.push_back(0);
205     while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);
206     vll c = karatsubaMultiply(a, b);
207     bigint res;
208     res.sign = sign * v.sign;
209     for (int i = 0, carry = 0; i < (int)c.size(); i++) {
210         long long cur = c[i] + carry;
211         res.a.push_back((int)(cur % 1000000));
212         carry = (int)(cur / 1000000);
213     }
214     res.a = convert_base(res.a, 6, base_digits);
215     res.trim();
216     return res;
217 }
218 friend pair<bigint, bigint> divmod(const bigint &a1,
219                                   const bigint &b1) { // 大整数除大整数, 同时返回答案
与余数
220     int norm = base / (b1.a.back() + 1);
221     bigint a = a1.abs() * norm;
222     bigint b = b1.abs() * norm;
223     bigint q, r;
224     q.a.resize(a.a.size());
225     for (int i = a.a.size() - 1; i >= 0; i--) {
226         r *= base;
227         r += a.a[i];
228         int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
229         int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
230         int d = ((long long)base * s1 + s2) / b.a.back();

```

```

231         r -= b * d;
232         while (r < 0) r += b, --d;
233         q.a[i] = d;
234     }
235     q.sign = a1.sign * b1.sign;
236     r.sign = a1.sign;
237     q.trim();
238     r.trim();
239     return make_pair(q, r / norm);
240 }
241 static vector<int> convert_base(const vector<int> &a, int old_digits, int
new_digits) {
242     vector<long long> p(max(old_digits, new_digits) + 1);
243     p[0] = 1;
244     for (int i = 1; i < (int)p.size(); i++) p[i] = p[i - 1] * 10;
245     vector<int> res;
246     long long cur = 0;
247     int cur_digits = 0;
248     for (int i = 0; i < (int)a.size(); i++) {
249         cur += a[i] * p[cur_digits];
250         cur_digits += old_digits;
251         while (cur_digits >= new_digits) {
252             res.push_back((int)(cur % p[new_digits]));
253             cur /= p[new_digits];
254             cur_digits -= new_digits;
255         }
256     }
257     res.push_back((int)cur);
258     while (!res.empty() && !res.back()) res.pop_back();
259     return res;
260 }
261 static vll karatsubaMultiply(const vll &a, const vll &b) {
262     int n = a.size();
263     vll res(n + n);
264     if (n <= 32) {
265         for (int i = 0; i < n; i++) {
266             for (int j = 0; j < n; j++) {
267                 res[i + j] += a[i] * b[j];
268             }
269         }
270         return res;
271     }
272
273     int k = n >> 1;
274     vll a1(a.begin(), a.begin() + k);
275     vll a2(a.begin() + k, a.end());
276     vll b1(b.begin(), b.begin() + k);
277     vll b2(b.begin() + k, b.end());
278
279     vll a1b1 = karatsubaMultiply(a1, b1);
280     vll a2b2 = karatsubaMultiply(a2, b2);
281
282     for (int i = 0; i < k; i++) a2[i] += a1[i];

```

```

283     for (int i = 0; i < k; i++) b2[i] += b1[i];
284
285     vll r = karatsubaMultiply(a2, b2);
286     for (int i = 0; i < (int)a1b1.size(); i++) r[i] -= a1b1[i];
287     for (int i = 0; i < (int)a2b2.size(); i++) r[i] -= a2b2[i];
288
289     for (int i = 0; i < (int)r.size(); i++) res[i + k] += r[i];
290     for (int i = 0; i < (int)a1b1.size(); i++) res[i] += a1b1[i];
291     for (int i = 0; i < (int)a2b2.size(); i++) res[i + n] += a2b2[i];
292     return res;
293 }
294
295 void operator*=(const bigint &v) {
296     *this = *this * v;
297 }
298 bigint operator/(const bigint &v) const {
299     return divmod(*this, v).first;
300 }
301 void operator/=(const bigint &v) {
302     *this = *this / v;
303 }
304 bigint operator%(const bigint &v) const {
305     return divmod(*this, v).second;
306 }
307 void operator%=(const bigint &v) {
308     *this = *this % v;
309 }
310 };

```

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 $A = (a_1, a_2, \dots, a_n)$ ，右移 x 位后变为 $A = (a_{x+1}, a_{x+2}, \dots, a_n, a_1, a_2, \dots, a_x)$ 。

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。

```

1  cin >> l >> r >> x;
2  l--; // 坐标修改为 0 开始
3  r--;
4  l = (l + shift) % n; // 偏移
5  r = (r + shift) % n;
6  if (l > r) { // 区间分离则分别操作
7      segt.modify(l, n - 1, x);
8      segt.modify(0, r, x);
9  } else {
10     segt.modify(l, r, x);
11 }

```

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 ($1 \leq n, q \leq 133333$) 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 ($1 \leq a_i \leq 10^6$) 代表初始颜色；随后 q 行为具体操作。

```

1  const int N = 1e6 + 7;
2  signed main() {
3      int n, q;
4      cin >> n >> q;
5      vector<int> w(n + 1);
6      for (int i = 1; i <= n; i++) {
7          cin >> w[i];
8      }
9
10     vector<array<int, 4>> query = {}; // {左区间, 右区间, 累计修改次数, 下标}
11     vector<array<int, 2>> modify = {}; // {修改的值, 修改的元素下标}
12     for (int i = 1; i <= q; i++) {
13         char op;
14         cin >> op;
15         if (op == 'Q') {
16             int l, r;
17             cin >> l >> r;
18             query.push_back({l, r, (int)modify.size() - 1, (int)query.size()});
19         } else {
20             int idx, w;
21             cin >> idx >> w;
22             modify.push_back({w, idx});
23         }
24     }
25
26     int Knum = 2154; // 计算块长
27     vector<int> K(n + 1);
28     for (int i = 1; i <= n; i++) { // 固定块长
29         K[i] = (i - 1) / Knum + 1;
30     }
31     sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
32         if (K[x[0]] != K[y[0]]) return x[0] < y[0];
33         if (K[x[1]] != K[y[1]]) return x[1] < y[1];
34         return x[3] < y[3];
35     });
36
37     int l = 1, r = 0, val = 0;
38     int t = 0; // 累计修改次数
39     vector<int> ans(query.size()), cnt(N);
40     for (int i = 1; i < query.size(); i++) {
41         auto [ql, qr, qt, id] = query[i];
42         auto add = [&](int x) -> void {
43             if (cnt[x] == 0) ++ val;

```

```

44         ++ cnt[x];
45     };
46     auto del = [&](int x) -> void {
47         -- cnt[x];
48         if (cnt[x] == 0) -- val;
49     };
50     auto time = [&](int x, int l, int r) -> void {
51         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
52             del(w[modify[x][1]]);
53             add(modify[x][0]);
54         }
55         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
56     };
57     while (l > ql) add(w[--l]);
58     while (r < qr) add(w[++r]);
59     while (l < ql) del(w[l++]);
60     while (r > qr) del(w[r--]);
61     while (t < qt) time(++t, ql, qr);
62     while (t > qt) time(t--, ql, qr);
63     ans[id] = val;
64 }
65 for (int i = 1; i < ans.size(); i++) {
66     cout << ans[i] << endl;
67 }
68 }

```

/END/