

数据结构A

笛卡尔树

小根笛卡尔树

```
1 cin >> n;
2 for (int i = 0; i < n; ++i) cin >> nums[i];
3 for (int i = 0; i < n; ++i) rs[i] = -1;
4 for (int i = 0; i < n; ++i) ls[i] = -1;
5 top = 0;
6 for (int i = 0; i < n; i++) {
7     int k = top;
8     while (k > 0 && nums[stk[k - 1]] > nums[i]) k--;
9     if (k) rs[stk[k - 1]] = i; // rs代表笛卡尔树每个节点的右儿子
10    if (k < top) ls[i] = stk[k]; // ls代表笛卡尔树每个节点的左儿子
11    stk[k++] = i;
12    top = k;
13 }
```

dsu并查集

路径优化(普遍)

```
1 struct dsu {
2     std::vector<int> d;
3     dsu(int n) { d.resize(n + 1); iota(d.begin(), d.end(), 0); }
4     int get_root(int x) { return d[x] = (x == d[x] ? x : get_root(d[x])); }
5 };
6 bool merge(int u, int v) {
7     if (get_root(u) != get_root(v)) {
8         d[get_root(u)] = get_root(v);
9         return true;
10    }
11    else return false;
12 }
```

根据集合的大小优化

```
1 //左移位数根据节点个数定
2 #define UFLIMIT (2<<17)
3 int unicnt[UFLIMIT];
4 void ufini(int n) {
5     for (int i = 0; i < n; i++) unicnt[i] = 1;
6 }
7 int ufroot(int x) { return unicnt[x] <= 0 ? -(unicnt[x] = -ufroot(-unicnt[x])) : x; }
8 int ufsame(int x, int y) { return ufroot(x) == ufroot(y); }
9 void uni(int x, int y) {
10    if ((x = ufroot(x)) == (y = ufroot(y))) return;
11    if (unicnt[x] < unicnt[y]) std::swap(x, y);
```

```

12     unicnt[x] += unicnt[y];
13     unicnt[y] = -x;
14 }

```

按秩合并优化

```

1  class UnionFind {
2  private:
3      std::vector<int> parent;
4      std::vector<int> rank;
5  public:
6      UnionFind(int n) {
7          parent.resize(n, 0);
8          rank.resize(n, 0);
9          iota(parent.begin(), parent.end(), 0);
10     }
11     int find(int x) {
12         if (parent[x] == x)
13             return x;
14         return parent[x] = find(parent[x]);
15     }
16     void merge(int x, int y) {
17         int rootX = find(x);
18         int rootY = find(y);
19         if (rootX == rootY) return;
20         if (rank[rootX] > rank[rootY])
21             std::swap(rootX, rootY);
22         parent[rootX] = rootY;
23         if (rank[rootX] == rank[rootY]) {
24             rank[rootY]++;
25         }
26     }
27     bool isConnect(int x, int y) {
28         return find(x) == find(y);
29     }
30 };

```

常用操作

```

1  struct DSU {
2      vector<int> fa, p, e, f;
3
4      DSU(int n) {
5          fa.resize(n + 1);
6          iota(fa.begin(), fa.end(), 0);
7          p.resize(n + 1, 1);
8          e.resize(n + 1);
9          f.resize(n + 1);
10     }
11     int get(int x) {
12         while (x != fa[x]) {
13             x = fa[x] = fa[fa[x]];
14         }
15         return x;

```

```

16     }
17     bool merge(int x, int y) { // 设x是y的祖先
18         if (x == y) f[get(x)] = 1;
19         x = get(x), y = get(y);
20         e[x]++;
21         if (x == y) return false;
22         if (x < y) swap(x, y); // 将编号小的合并到大的上
23         fa[y] = x;
24         f[x] |= f[y], p[x] += p[y], e[x] += e[y];
25         return true;
26     }
27     bool same(int x, int y) {
28         return get(x) == get(y);
29     }
30     bool F(int x) { // 判断连通块内是否存在自环
31         return f[get(x)];
32     }
33     int size(int x) { // 输出连通块中点的数量
34         return p[get(x)];
35     }
36     int E(int x) { // 输出连通块中边的数量
37         return e[get(x)];
38     }
39 };

```

ST 表

用于解决区间可重复贡献问题，需要满足 x 运算符 $x = x$ （如区间最大值： $\max(x, x) = x$ 、区间 gcd： $\gcd(x, x) = x$ 等），但是不支持修改操作。 $\mathcal{O}(N \log N)$ 预处理， $\mathcal{O}(1)$ 查询。

```

1  struct ST {
2      const int n, k;
3      vector<int> in1, in2;
4      vector<vector<int>> Max, Min;
5      ST(int n) : n(n), in1(n + 1), in2(n + 1), k(__lg(n)) {
6          Max.resize(k + 1, vector<int>(n + 1));
7          Min.resize(k + 1, vector<int>(n + 1));
8      }
9      void init() {
10         for (int i = 1; i <= n; i++) {
11             Max[0][i] = in1[i];
12             Min[0][i] = in2[i];
13         }
14         for (int i = 0, t = 1; i < k; i++, t <= 1) {
15             const int T = n - (t < 1) + 1;
16             for (int j = 1; j <= T; j++) {
17                 Max[i + 1][j] = max(Max[i][j], Max[i][j + t]);
18                 Min[i + 1][j] = min(Min[i][j], Min[i][j + t]);
19             }
20         }
21     }
22     int getMax(int l, int r) {
23         if (l > r) {
24             swap(l, r);
25         }

```

```

26     int k = __lg(r - l + 1);
27     return max(Max[k][l], Max[k][r - (1 << k) + 1]);
28 }
29 int getMin(int l, int r) {
30     if (l > r) {
31         swap(l, r);
32     }
33     int k = __lg(r - l + 1);
34     return min(Min[k][l], Min[k][r - (1 << k) + 1]);
35 }
36 };

```

Fenwick Tree 树状数组

```

1  template<class T> struct BIT {
2      int n;
3      vector<T> w;
4      BIT(int n, auto &in) : n(n), w(n + 1) { // 预处理填值
5          for (int i = 1; i <= n; i++) {
6              add(i, in[i]);
7          }
8      }
9      void add(int x, T v) {
10         for (; x <= n; x += x & -x) {
11             w[x] += v;
12         }
13     }
14     T ask(int x) { // 前缀和查询
15         T ans = 0;
16         for (; x; x -= x & -x) {
17             ans += w[x];
18         }
19         return ans;
20     }
21     T ask(int l, int r) { // 差分实现区间和查询
22         return ask(r) - ask(l - 1);
23     }
24 };

```

逆序对扩展

```

1  struct BIT {
2      int n;
3      vector<int> w, chk; // chk 为传入的待处理数组
4      BIT(int n, auto &in) : n(n), w(n + 1), chk(in) {}
5      /* 需要全部常规封装 */
6      int get() {
7          vector<array<int, 2>> alls;
8          for (int i = 1; i <= n; i++) {
9              alls.push_back({chk[i], i});
10         }
11         sort(alls.begin(), alls.end());
12         int ans = 0;
13         for (auto [val, idx] : alls) {

```

```

14         ans += ask(idx + 1, n);
15         add(idx, 1);
16     }
17     return ans;
18 }
19 };

```

前驱后继扩展 (常规+第 k 小值查询+元素排名查询+元素前驱后继查询)

注意, 被查询的值都应该小于等于 N , 否则会越界; 如果离散化不可使用, 则需要使用平衡树替代。

```

1  struct BIT {
2      int n;
3      vector<int> w;
4      BIT(int n) : n(n), w(n + 1) {}
5      void add(int x, int v) {
6          for (; x <= n; x += x & -x) {
7              w[x] += v;
8          }
9      }
10     int kth(int x) { // 查找第 k 小的值
11         int ans = 0;
12         for (int i = __lg(n); i >= 0; i--) {
13             int val = ans + (1 << i);
14             if (val < n && w[val] < x) {
15                 x -= w[val];
16                 ans = val;
17             }
18         }
19         return ans + 1;
20     }
21     int get(int x) { // 查找 x 的排名
22         int ans = 1;
23         for (x--; x; x -= x & -x) {
24             ans += w[x];
25         }
26         return ans;
27     }
28     int pre(int x) { return kth(get(x) - 1); } // 查找 x 的前驱
29     int suf(int x) { return kth(get(x + 1)); } // 查找 x 的后继
30 };
31 const int N = 10000000; // 可以用于在线处理平衡二叉树的全部要求
32 signed main() {
33     BIT bit(N + 1); // 在线处理不能够离散化, 一定要开到比最大值更大
34     int n;
35     cin >> n;
36     for (int i = 1; i <= n; i++) {
37         int op, x;
38         cin >> op >> x;
39         if (op == 1) bit.add(x, 1); // 插入 x
40         else if (op == 2) bit.add(x, -1); // 删除任意一个 x
41         else if (op == 3) cout << bit.get(x) << "\n"; // 查询 x 的排名
42         else if (op == 4) cout << bit.kth(x) << "\n"; // 查询排名为 x 的数
43         else if (op == 5) cout << bit.pre(x) << "\n"; // 求小于 x 的最大值 (前

```

驱)

```

44         else if (op == 6) cout << bit.suf(x) << "\n"; // 求大于 x 的最小值（后
    继）
45     }
46 }

```

最值查询扩展（常规+区间最值查询+单点赋值）

以 $\mathcal{O}(\log \log N)$ 的复杂度运行，但是即便如此依然略优于线段树（后者常数较大）。

```

1  template<class T> struct BIT {
2      int n;
3      vector<T> w, base;
4      #define low(x) (x & -x)
5      BIT(int n, auto &in) : n(n), w(n + 1), base(n + 1) {
6          for (int i = 1; i <= n; i++) {
7              update(i, in[i]);
8          }
9      } /* 可以增加并使用常规封装中的几个函数 */
10     void update(int x, int v) { // 单点赋值
11         base[x] = max(base[x], v);
12         for (; x <= n; x += low(x)) {
13             w[x] = max(w[x], v);
14         }
15     }
16     T getMax(int l, int r) { // 最值查询
17         T ans = T();
18         while (r >= 1) {
19             ans = max(base[r], ans);
20             for (r--; r - low(r) >= 1; r -= low(r)) {
21                 ans = max(w[r], ans);
22             }
23         }
24         return ans;
25     }
26 };

```

二维树状数组

封装一：该版本不能同时进行区间修改+区间查询。无离散化版本的空间占用为 $\mathcal{O}(NM)$ 、建树复杂度为 $\mathcal{O}(NM)$ 、单次查询复杂度为 $\mathcal{O}(\log N \cdot \log M)$ 。

```

1  struct BIT_2D {
2      int n, m;
3      vector<vector<int>> w;
4
5      BIT_2D(int n, int m) : n(n), m(m) {
6          w.resize(n + 1, vector<int>(m + 1));
7      }
8      void add(int x, int y, int k) {
9          for (int i = x; i <= n; i += i & -i) {
10             for (int j = y; j <= m; j += j & -j) {
11                 w[i][j] += k;
12             }
13         }
14     }
15 };

```

```

14     }
15     void add(int x, int y, int X, int Y, int k) { // 区块修改：二维差分
16         X++, Y++;
17         add(x, y, k), add(X, y, -k);
18         add(X, Y, k), add(x, Y, -k);
19     }
20     int ask(int x, int y) { // 单点查询
21         int ans = 0;
22         for (int i = x; i; i -= i & -i) {
23             for (int j = y; j; j -= j & -j) {
24                 ans += w[i][j];
25             }
26         }
27         return ans;
28     }
29     int ask(int x, int y, int X, int Y) { // 区块查询：二维前缀和
30         x--, y--;
31         return ask(X, Y) - ask(x, Y) - ask(X, y) + ask(x, y);
32     }
33 };

```

封装二：该版本支持全部操作。但是时空复杂度均比上一个版本多 4 倍。

```

1 struct BIT_2D {
2     int n, m;
3     vector<vector<int>> b1, b2, b3, b4;
4
5     BIT_2D(int n, int m) : n(n), m(m) {
6         b1.resize(n + 1, vector<int>(m + 1));
7         b2.resize(n + 1, vector<int>(m + 1));
8         b3.resize(n + 1, vector<int>(m + 1));
9         b4.resize(n + 1, vector<int>(m + 1));
10    }
11    void add(auto &w, int x, int y, int k) { // 单点修改
12        for (int i = x; i <= n; i += i & -i) {
13            for (int j = y; j <= m; j += j & -j) {
14                w[i][j] += k;
15            }
16        }
17    }
18    void add(int x, int y, int k) { // 多了一步计算
19        add(b1, x, y, k);
20        add(b2, x, y, k * (x - 1));
21        add(b3, x, y, k * (y - 1));
22        add(b4, x, y, k * (x - 1) * (y - 1));
23    }
24    void add(int x, int y, int X, int Y, int k) { // 区块修改：二维差分
25        X++, Y++;
26        add(x, y, k), add(X, y, -k);
27        add(X, Y, k), add(x, Y, -k);
28    }
29    int ask(auto &w, int x, int y) { // 单点查询
30        int ans = 0;
31        for (int i = x; i; i -= i & -i) {
32            for (int j = y; j; j -= j & -j) {

```

```

33         ans += w[i][j];
34     }
35 }
36 return ans;
37 }
38 int ask(int x, int y) { // 多了一步计算
39     int ans = 0;
40     ans += x * y * ask(b1, x, y);
41     ans -= y * ask(b2, x, y);
42     ans -= x * ask(b3, x, y);
43     ans += ask(b4, x, y);
44     return ans;
45 }
46 int ask(int x, int y, int X, int Y) { // 区块查询: 二维前缀和
47     x--, y--;
48     return ask(X, Y) - ask(x, Y) - ask(X, y) + ask(x, y);
49 }
50 };

```

线段树

快速线段树 (单点修改+区间最值)

```

1  struct Segt {
2      vector<int> w;
3      int n;
4      Segt(int n) : w(2 * n, (int)-2E9), n(n) {}
5
6      void modify(int pos, int val) {
7          for (w[pos += n] = val; pos > 1; pos /= 2) {
8              w[pos / 2] = max(w[pos], w[pos ^ 1]);
9          }
10     }
11
12     int ask(int l, int r) {
13         int res = -2E9;
14         for (l += n, r += n; l < r; l /= 2, r /= 2) {
15             if (l % 2) res = max(res, w[l++]);
16             if (r % 2) res = max(res, w[--r]);
17         }
18         return res;
19     }
20 };

```

区间加法修改、区间最小值查询 已修正 2025.7.19

```

1  template<class T> struct Segt {
2      struct node {
3          int l, r;
4          T w, rmq, lazy;
5      };
6      std::vector<T> w;
7      std::vector<node> t;
8

```



```

9 Segt() {}
10 Segt(int n) { init(n); }
11 Segt(std::vector<int> in) {
12     int n = in.size() - 1;
13     w.resize(n + 1);
14     for (int i = 1; i <= n; i++) {
15         w[i] = in[i];
16     }
17     init(in.size() - 1);
18 }
19
20 #define GL (k << 1)
21 #define GR (k << 1 | 1)
22
23 void init(int n) {
24     w.resize(n + 1);
25     t.resize(n * 4 + 1);
26     auto build = [&](auto self, int l, int r, int k = 1) {
27         if (l == r) {
28             t[k] = { l, r, w[l], w[l], 0 }; // 如果有赋值为 0 的操作，则懒标
记必须要 -1
29             return;
30         }
31         t[k] = { l, r, 0, 0, 0 };
32         int mid = (l + r) / 2;
33         self(self, l, mid, GL);
34         self(self, mid + 1, r, GR);
35         pushup(k);
36     };
37     build(build, 1, n);
38 }
39 void pushdown(node& p, T lazy) { /* 【在此更新下递函数】 */
40     p.w += (p.r - p.l + 1) * lazy;
41     p.rmqs += lazy;
42     p.lazy += lazy;
43 }
44 void pushdown(int k) {
45     if (t[k].lazy == 0) return;
46     pushdown(t[GL], t[k].lazy);
47     pushdown(t[GR], t[k].lazy);
48     t[k].lazy = 0;
49 }
50 void pushup(int k) {
51     auto pushup = [&](node& p, node& l, node& r) { /* 【在此更新上传函数】
*/
52         p.w = l.w + r.w;
53         p.rmqs = std::min(l.rmqs, r.rmqs); // RMQ -> min/max
54     };
55     pushup(t[k], t[GL], t[GR]);
56 }
57 void modify(int l, int r, T val, int k = 1) { // 区间修改
58     if (l <= t[k].l && t[k].r <= r) {
59         pushdown(t[k], val);
60         return;
61     }
62     pushdown(k);

```

```

63     int mid = (t[k].l + t[k].r) / 2;
64     if (l <= mid) modify(l, r, val, GL);
65     if (mid < r) modify(l, r, val, GR);
66     pushup(k);
67 }
68 T rmq(int l, int r, int k = 1) { // 区间询问最小值
69     if (l <= t[k].l && t[k].r <= r) {
70         return t[k].rmq;
71     }
72     pushdown(k);
73     int mid = (t[k].l + t[k].r) / 2;
74     T ans = std::numeric_limits<T>::max(); // RMQ -> 为 max 时需要修改为
    ::lowest()
75     if (l <= mid) ans = std::min(ans, rmq(l, r, GL)); // RMQ -> min/max
76     if (mid < r) ans = std::min(ans, rmq(l, r, GR)); // RMQ -> min/max
77     return ans;
78 }
79 T ask(int l, int r, int k = 1) { // 区间询问
80     if (l <= t[k].l && t[k].r <= r) {
81         return t[k].w;
82     }
83     pushdown(k);
84     int mid = (t[k].l + t[k].r) / 2;
85     T ans = 0;
86     if (l <= mid) ans += ask(l, r, GL);
87     if (mid < r) ans += ask(l, r, GR);
88     return ans;
89 }
90 void debug(int k = 1) {
91     std::cout << "[" << t[k].l << ", " << t[k].r << "]: ";
92     std::cout << "w = " << t[k].w << ", ";
93     std::cout << "Min = " << t[k].rmq << ", ";
94     std::cout << "lazy = " << t[k].lazy << ", ";
95     std::cout << std::endl;
96     if (t[k].l == t[k].r) return;
97     debug(GL), debug(GR);
98 }
99 };

```

同时需要处理区间加法与乘法修改

```

1  template <class T> struct Segt_ {
2      struct node {
3          int l, r;
4          T w, add, mul = 1; // 注意初始赋值
5      };
6      vector<T> w;
7      vector<node> t;
8
9      Segt_(int n) {
10         w.resize(n + 1);
11         t.resize((n << 2) + 1);
12         build(1, n);
13     }
14     Segt_(vector<int> in) {

```

```

15     int n = in.size() - 1;
16     w.resize(n + 1);
17     for (int i = 1; i <= n; i++) {
18         w[i] = in[i];
19     }
20     t.resize((n << 2) + 1);
21     build(1, n);
22 }
23 void pushdown(node &p, T add, T mul) { // 在此更新下递函数
24     p.w = p.w * mul + (p.r - p.l + 1) * add;
25     p.add = p.add * mul + add;
26     p.mul *= mul;
27 }
28 void pushup(node &p, node &l, node &r) { // 在此更新上传函数
29     p.w = l.w + r.w;
30 }
31 #define GL (k << 1)
32 #define GR (k << 1 | 1)
33 void pushdown(int k) { // 不需要动
34     pushdown(t[GL], t[k].add, t[k].mul);
35     pushdown(t[GR], t[k].add, t[k].mul);
36     t[k].add = 0, t[k].mul = 1;
37 }
38 void pushup(int k) { // 不需要动
39     pushup(t[k], t[GL], t[GR]);
40 }
41 void build(int l, int r, int k = 1) {
42     if (l == r) {
43         t[k] = {l, r, w[l]};
44         return;
45     }
46     t[k] = {l, r};
47     int mid = (l + r) / 2;
48     build(l, mid, GL);
49     build(mid + 1, r, GR);
50     pushup(k);
51 }
52 void modify(int l, int r, T val, int k = 1) { // 区间修改
53     if (l <= t[k].l && t[k].r <= r) {
54         t[k].w += (t[k].r - t[k].l + 1) * val;
55         t[k].add += val;
56         return;
57     }
58     pushdown(k);
59     int mid = (t[k].l + t[k].r) / 2;
60     if (l <= mid) modify(l, r, val, GL);
61     if (mid < r) modify(l, r, val, GR);
62     pushup(k);
63 }
64 void modify2(int l, int r, T val, int k = 1) { // 区间修改
65     if (l <= t[k].l && t[k].r <= r) {
66         t[k].w *= val;
67         t[k].add *= val;
68         t[k].mul *= val;
69         return;
70     }

```

```

71     pushdown(k);
72     int mid = (t[k].l + t[k].r) / 2;
73     if (l <= mid) modify2(l, r, val, GL);
74     if (mid < r) modify2(l, r, val, GR);
75     pushup(k);
76 }
77 T ask(int l, int r, int k = 1) { // 区间询问, 不合并
78     if (l <= t[k].l && t[k].r <= r) {
79         return t[k].w;
80     }
81     pushdown(k);
82     int mid = (t[k].l + t[k].r) / 2;
83     T ans = 0;
84     if (l <= mid) ans += ask(l, r, GL);
85     if (mid < r) ans += ask(l, r, GR);
86     return ans;
87 }
88 void debug(int k = 1) {
89     cout << "[" << t[k].l << ", " << t[k].r << "]: ";
90     cout << "w = " << t[k].w << ", ";
91     cout << "add = " << t[k].add << ", ";
92     cout << "mul = " << t[k].mul << ", ";
93     cout << endl;
94     if (t[k].l == t[k].r) return;
95     debug(GL), debug(GR);
96 }
97 };

```

区间赋值/推平

如果存在推平为 0 的操作，那么需要将 lazy 初始赋值为 -1 。

```

1 void pushdown(node &p, T lazy) { /* 【在此更新下递函数】 */
2     p.w = (p.r - p.l + 1) * lazy;
3     p.lazy = lazy;
4 }
5 void modify(int l, int r, T val, int k = 1) {
6     if (l <= t[k].l && t[k].r <= r) {
7         t[k].w = val;
8         t[k].lazy = val;
9         return;
10    }
11    // 剩余部分不变
12 }

```

区间取模

原题需要进行“单点赋值+区间取模+区间求和”[See](#)。该操作不需要懒标记。

需要额外维护一个区间最大值，当模数大于区间最大值时剪枝，否则进行单点取模。由于单点 $\text{MOD} < x$ 时 $x \bmod \text{MOD} < \frac{x}{2}$ ，故单点取模至 0 最劣只需要 $\log x$ 次。

```

1 void modifyMod(int l, int r, T val, int k = 1) {
2     if (l <= t[k].l && t[k].r <= r) {
3         if (t[k].rmq < val) return; // 重要剪枝

```

```

4     }
5     if (t[k].l == t[k].r) {
6         t[k].w %= val;
7         t[k].rmq %= val;
8         return;
9     }
10    int mid = (t[k].l + t[k].r) / 2;
11    if (l <= mid) modifyMod(l, r, val, GL);
12    if (mid < r) modifyMod(l, r, val, GR);
13    pushup(k);
14 }

```

区间异或修改

原题需要维护“区间异或修改+区间求和”[See](#)。

```

1  struct Segt { // #define GL (k << 1) // #define GR (k << 1 | 1)
2      struct node {
3          int l, r;
4          int w[N], lazy; // 注意这里为了方便计算, w 只需要存位
5      };
6      vector<int> base;
7      vector<node> t;
8
9      Segt(vector<int> in) : base(in) {
10         int n = in.size() - 1;
11         t.resize(n * 4 + 1);
12         auto build = [&](auto self, int l, int r, int k = 1) {
13             t[k] = {l, r}; // 前置赋值
14             if (l == r) {
15                 for (int i = 0; i < N; i++) {
16                     t[k].w[i] = base[l] >> i & 1;
17                 }
18                 return;
19             }
20             int mid = (l + r) / 2;
21             self(self, l, mid, GL);
22             self(self, mid + 1, r, GR);
23             pushup(k);
24         };
25         build(build, 1, n);
26     }
27     void pushdown(node &p, int lazy) { /* 【在此更新下递函数】 */
28         int len = p.r - p.l + 1;
29         for (int i = 0; i < N; i++) {
30             if (lazy >> i & 1) { // 即 p.w = (p.r - p.l + 1) - p.w;
31                 p.w[i] = len - p.w[i];
32             }
33         }
34         p.lazy ^= lazy;
35     }
36     void pushdown(int k) { // 【不需要动】
37         if (t[k].lazy == 0) return;
38         pushdown(t[GL], t[k].lazy);
39         pushdown(t[GR], t[k].lazy);

```

```

40     t[k].lazy = 0;
41 }
42 void pushup(int k) {
43     auto pushup = [&](node &p, node &l, node &r) { /* 【在此更新上传函数】
44 */
45         for (int i = 0; i < N; i++) {
46             p.w[i] = l.w[i] + r.w[i]; // 即 p.w = l.w + r.w;
47         }
48     };
49     pushup(t[k], t[GL], t[GR]);
50 }
51 void modify(int l, int r, int val, int k = 1) { // 区间修改
52     if (l <= t[k].l && t[k].r <= r) {
53         pushdown(t[k], val);
54         return;
55     }
56     pushdown(k);
57     int mid = (t[k].l + t[k].r) / 2;
58     if (l <= mid) modify(l, r, val, GL);
59     if (mid < r) modify(l, r, val, GR);
60     pushup(k);
61 }
62 i64 ask(int l, int r, int k = 1) { // 区间求和
63     if (l <= t[k].l && t[k].r <= r) {
64         i64 ans = 0;
65         for (int i = 0; i < N; i++) {
66             ans += t[k].w[i] * (1LL << i);
67         }
68         return ans;
69     }
70     pushdown(k);
71     int mid = (t[k].l + t[k].r) / 2;
72     i64 ans = 0;
73     if (l <= mid) ans += ask(l, r, GL);
74     if (mid < r) ans += ask(l, r, GR);
75     return ans;
76 }
};

```

拆位运算

原题同上。使用若干棵线段树维护每一位的值，区间异或转变为区间翻转。

```

1  template<class T> struct Segt_ { // GL 为 (k << 1), GR 为 (k << 1 | 1)
2      struct node {
3          int l, r;
4          T w;
5          bool lazy; // 注意懒标记用布尔型足以
6      };
7      vector<T> w;
8      vector<node> t;
9
10     Segt_() {}
11     void init(vector<int> in) {
12         int n = in.size() - 1;

```

```

13     w.resize(n * 4 + 1);
14     for (int i = 0; i <= n; i++) { w[i] = in[i]; }
15     t.resize(n * 4 + 1);
16     build(1, n);
17 }
18 void pushdown(node &p, bool lazy = 1) { // 【在此更新下递函数】
19     p.w = (p.r - p.l + 1) - p.w;
20     p.lazy ^= lazy;
21 }
22 void pushup(node &p, node &l, node &r) { // 【在此更新上传函数】
23     p.w = l.w + r.w;
24 }
25 void pushdown(int k) { // 【不需要动】
26     if (t[k].lazy == 0) return;
27     pushdown(t[GL]), pushdown(t[GR]); // 注意这里不再需要传入第二个参数
28     t[k].lazy = 0;
29 }
30 void pushup(int k) { pushup(t[k], t[GL], t[GR]); } // 【不需要动】
31 void build(int l, int r, int k = 1) {
32     if (l == r) {
33         t[k] = {l, r, w[l], 0}; // 注意懒标记初始为 0
34         return;
35     }
36     t[k] = {l, r};
37     int mid = (l + r) / 2;
38     build(l, mid, GL);
39     build(mid + 1, r, GR);
40     pushup(k);
41 }
42 void reverse(int l, int r, int k = 1) { // 区间翻转
43     if (l <= t[k].l && t[k].r <= r) {
44         pushdown(t[k], 1);
45         return;
46     }
47     pushdown(k);
48     int mid = (t[k].l + t[k].r) / 2;
49     if (l <= mid) reverse(l, r, GL);
50     if (mid < r) reverse(l, r, GR);
51     pushup(k);
52 }
53 int ask(int l, int r, int k = 1) { // 区间求和
54     if (l <= t[k].l && t[k].r <= r) {
55         return t[k].w;
56     }
57     pushdown(k);
58     int mid = (t[k].l + t[k].r) / 2;
59     int ans = 0;
60     if (l <= mid) ans += ask(l, r, GL);
61     if (mid < r) ans += ask(l, r, GR);
62     return ans;
63 }
64 };
65 signed main() {
66     int n; cin >> n;
67     vector in(20, vector<int>(n + 1));
68     Segt<i64> segt[20]; // 拆位建线段树

```

```

69     for (int i = 1, x; i <= n; i++) { cin >> x;
70         for (int bit = 0; bit < 20; bit++) {
71             in[bit][i] = x >> bit & 1;
72         }
73     }
74     for (int i = 0; i < 20; i++) {
75         segt[i].init(in[i]);
76     }
77
78     int m, op;
79     for (cin >> m; m; m--) { cin >> op;
80         if (op == 1) {
81             int l, r; i64 ans = 0; cin >> l >> r;
82             for (int i = 0; i < 20; i++) {
83                 ans += segt[i].ask(l, r) * (1LL << i);
84             }
85             cout << ans << "\n";
86         } else {
87             int l, r, val; cin >> l >> r >> val;
88             for (int i = 0; i < 20; i++) {
89                 if (val >> i & 1) { segt[i].reverse(l, r); }
90             }
91         }
92     }
93 }

```

树套树

线段树套平衡树

```

1  #include <bits/stdc++.h>
2  #include <ext/pb_ds/assoc_container.hpp>
3  using namespace __gnu_pbds;
4  using pii = std::pair<int, int>;
5  const int inf = 2147483647;
6
7  tree<pii, null_type, std::less<pii>, rb_tree_tag,
8  tree_order_statistics_node_update> ver;
9
10 template<class Info>
11 struct SegmentTree {
12     int n;
13     std::vector<Info> info;
14
15     SegmentTree() : n(0) {}
16
17     void init(int n_) {
18         n = n_;
19         info.assign(4 << std::__lg(n), Info());
20     }
21 public:
22     // --- 初始化与构建 ---
23     void build(const std::vector<int>& a) {
24         _build(1, 0, n, a);
25     }
26
27     void _build(int l, int r, int n, const std::vector<int>& a) {
28         if (l == r) {
29             info[l].x = a[n];
30             return;
31         }
32         int m = (l + r) / 2;
33         _build(l, m, n, a);
34         _build(m + 1, r, n, a);
35         info[m].x = info[l].x;
36         info[m].y = info[r].y;
37     }
38
39     int query(int l, int r, int k) {
40         if (l == r) return info[l].x;
41         int m = (l + r) / 2;
42         if (k <= info[m].y) return query(l, m, k);
43         else return query(m + 1, r, k - info[m].y);
44     }
45
46     void update(int l, int r, int k, int v) {
47         if (l == r) {
48             info[l].x = v;
49             return;
50         }
51         int m = (l + r) / 2;
52         update(l, m, k, v);
53         update(m + 1, r, k, v);
54         info[m].x = info[l].x;
55         info[m].y = info[r].y;
56     }
57
58     int kth(int l, int r, int k) {
59         if (l == r) return info[l].x;
60         int m = (l + r) / 2;
61         if (k <= info[m].y) return kth(l, m, k);
62         else return kth(m + 1, r, k - info[m].y);
63     }
64
65     void erase(int l, int r, int k) {
66         if (l == r) {
67             info[l].x = -1;
68             return;
69         }
70         int m = (l + r) / 2;
71         erase(l, m, k);
72         erase(m + 1, r, k);
73         info[m].x = info[l].x;
74         info[m].y = info[r].y;
75     }
76
77     int getRank(int l, int r, int x) {
78         if (l == r) return 1;
79         int m = (l + r) / 2;
80         if (x < info[m].x) return getRank(l, m, x);
81         else return getRank(m + 1, r, x) + 1;
82     }
83
84     int getPre(int l, int r, int x) {
85         if (l == r) return -1;
86         int m = (l + r) / 2;
87         if (x < info[m].x) return getPre(l, m, x);
88         else return info[m].x;
89     }
90
91     int getSucc(int l, int r, int x) {
92         if (l == r) return -1;
93         int m = (l + r) / 2;
94         if (x < info[m].x) return info[m].x;
95         else return getSucc(m + 1, r, x);
96     }
97
98     void erase(int l, int r, int x) {
99         if (l == r) {
100             info[l].x = -1;
101             return;
102         }
103         int m = (l + r) / 2;
104         erase(l, m, x);
105         erase(m + 1, r, x);
106         info[m].x = info[l].x;
107         info[m].y = info[r].y;
108     }
109
110     int kth(int l, int r, int k) {
111         if (l == r) return info[l].x;
112         int m = (l + r) / 2;
113         if (k <= info[m].y) return kth(l, m, k);
114         else return kth(m + 1, r, k - info[m].y);
115     }
116
117     void update(int l, int r, int k, int v) {
118         if (l == r) {
119             info[l].x = v;
120             return;
121         }
122         int m = (l + r) / 2;
123         update(l, m, k, v);
124         update(m + 1, r, k, v);
125         info[m].x = info[l].x;
126         info[m].y = info[r].y;
127     }
128
129     int getRank(int l, int r, int x) {
130         if (l == r) return 1;
131         int m = (l + r) / 2;
132         if (x < info[m].x) return getRank(l, m, x);
133         else return getRank(m + 1, r, x) + 1;
134     }
135
136     int getPre(int l, int r, int x) {
137         if (l == r) return -1;
138         int m = (l + r) / 2;
139         if (x < info[m].x) return getPre(l, m, x);
140         else return info[m].x;
141     }
142
143     int getSucc(int l, int r, int x) {
144         if (l == r) return -1;
145         int m = (l + r) / 2;
146         if (x < info[m].x) return info[m].x;
147         else return getSucc(m + 1, r, x);
148     }
149
150     void erase(int l, int r, int x) {
151         if (l == r) {
152             info[l].x = -1;
153             return;
154         }
155         int m = (l + r) / 2;
156         erase(l, m, x);
157         erase(m + 1, r, x);
158         info[m].x = info[l].x;
159         info[m].y = info[r].y;
160     }
161
162     int kth(int l, int r, int k) {
163         if (l == r) return info[l].x;
164         int m = (l + r) / 2;
165         if (k <= info[m].y) return kth(l, m, k);
166         else return kth(m + 1, r, k - info[m].y);
167     }
168
169     void update(int l, int r, int k, int v) {
170         if (l == r) {
171             info[l].x = v;
172             return;
173         }
174         int m = (l + r) / 2;
175         update(l, m, k, v);
176         update(m + 1, r, k, v);
177         info[m].x = info[l].x;
178         info[m].y = info[r].y;
179     }
180
181     int getRank(int l, int r, int x) {
182         if (l == r) return 1;
183         int m = (l + r) / 2;
184         if (x < info[m].x) return getRank(l, m, x);
185         else return getRank(m + 1, r, x) + 1;
186     }
187
188     int getPre(int l, int r, int x) {
189         if (l == r) return -1;
190         int m = (l + r) / 2;
191         if (x < info[m].x) return getPre(l, m, x);
192         else return info[m].x;
193     }
194
195     int getSucc(int l, int r, int x) {
196         if (l == r) return -1;
197         int m = (l + r) / 2;
198         if (x < info[m].x) return info[m].x;
199         else return getSucc(m + 1, r, x);
200     }
201
202     void erase(int l, int r, int x) {
203         if (l == r) {
204             info[l].x = -1;
205             return;
206         }
207         int m = (l + r) / 2;
208         erase(l, m, x);
209         erase(m + 1, r, x);
210         info[m].x = info[l].x;
211         info[m].y = info[r].y;
212     }
213
214     int kth(int l, int r, int k) {
215         if (l == r) return info[l].x;
216         int m = (l + r) / 2;
217         if (k <= info[m].y) return kth(l, m, k);
218         else return kth(m + 1, r, k - info[m].y);
219     }
220
221     void update(int l, int r, int k, int v) {
222         if (l == r) {
223             info[l].x = v;
224             return;
225         }
226         int m = (l + r) / 2;
227         update(l, m, k, v);
228         update(m + 1, r, k, v);
229         info[m].x = info[l].x;
230         info[m].y = info[r].y;
231     }
232
233     int getRank(int l, int r, int x) {
234         if (l == r) return 1;
235         int m = (l + r) / 2;
236         if (x < info[m].x) return getRank(l, m, x);
237         else return getRank(m + 1, r, x) + 1;
238     }
239
240     int getPre(int l, int r, int x) {
241         if (l == r) return -1;
242         int m = (l + r) / 2;
243         if (x < info[m].x) return getPre(l, m, x);
244         else return info[m].x;
245     }
246
247     int getSucc(int l, int r, int x) {
248         if (l == r) return -1;
249         int m = (l + r) / 2;
250         if (x < info[m].x) return info[m].x;
251         else return getSucc(m + 1, r, x);
252     }
253
254     void erase(int l, int r, int x) {
255         if (l == r) {
256             info[l].x = -1;
257             return;
258         }
259         int m = (l + r) / 2;
260         erase(l, m, x);
261         erase(m + 1, r, x);
262         info[m].x = info[l].x;
263         info[m].y = info[r].y;
264     }
265
266     int kth(int l, int r, int k) {
267         if (l == r) return info[l].x;
268         int m = (l + r) / 2;
269         if (k <= info[m].y) return kth(l, m, k);
270         else return kth(m + 1, r, k - info[m].y);
271     }
272
273     void update(int l, int r, int k, int v) {
274         if (l == r) {
275             info[l].x = v;
276             return;
277         }
278         int m = (l + r) / 2;
279         update(l, m, k, v);
280         update(m + 1, r, k, v);
281         info[m].x = info[l].x;
282         info[m].y = info[r].y;
283     }
284
285     int getRank(int l, int r, int x) {
286         if (l == r) return 1;
287         int m = (l + r) / 2;
288         if (x < info[m].x) return getRank(l, m, x);
289         else return getRank(m + 1, r, x) + 1;
290     }
291
292     int getPre(int l, int r, int x) {
293         if (l == r) return -1;
294         int m = (l + r) / 2;
295         if (x < info[m].x) return getPre(l, m, x);
296         else return info[m].x;
297     }
298
299     int getSucc(int l, int r, int x) {
300         if (l == r) return -1;
301         int m = (l + r) / 2;
302         if (x < info[m].x) return info[m].x;
303         else return getSucc(m + 1, r, x);
304     }
305
306     void erase(int l, int r, int x) {
307         if (l == r) {
308             info[l].x = -1;
309             return;
310         }
311         int m = (l + r) / 2;
312         erase(l, m, x);
313         erase(m + 1, r, x);
314         info[m].x = info[l].x;
315         info[m].y = info[r].y;
316     }
317
318     int kth(int l, int r, int k) {
319         if (l == r) return info[l].x;
320         int m = (l + r) / 2;
321         if (k <= info[m].y) return kth(l, m, k);
322         else return kth(m + 1, r, k - info[m].y);
323     }
324
325     void update(int l, int r, int k, int v) {
326         if (l == r) {
327             info[l].x = v;
328             return;
329         }
330         int m = (l + r) / 2;
331         update(l, m, k, v);
332         update(m + 1, r, k, v);
333         info[m].x = info[l].x;
334         info[m].y = info[r].y;
335     }
336
337     int getRank(int l, int r, int x) {
338         if (l == r) return 1;
339         int m = (l + r) / 2;
340         if (x < info[m].x) return getRank(l, m, x);
341         else return getRank(m + 1, r, x) + 1;
342     }
343
344     int getPre(int l, int r, int x) {
345         if (l == r) return -1;
346         int m = (l + r) / 2;
347         if (x < info[m].x) return getPre(l, m, x);
348         else return info[m].x;
349     }
350
351     int getSucc(int l, int r, int x) {
352         if (l == r) return -1;
353         int m = (l + r) / 2;
354         if (x < info[m].x) return info[m].x;
355         else return getSucc(m + 1, r, x);
356     }
357
358     void erase(int l, int r, int x) {
359         if (l == r) {
360             info[l].x = -1;
361             return;
362         }
363         int m = (l + r) / 2;
364         erase(l, m, x);
365         erase(m + 1, r, x);
366         info[m].x = info[l].x;
367         info[m].y = info[r].y;
368     }
369
370     int kth(int l, int r, int k) {
371         if (l == r) return info[l].x;
372         int m = (l + r) / 2;
373         if (k <= info[m].y) return kth(l, m, k);
374         else return kth(m + 1, r, k - info[m].y);
375     }
376
377     void update(int l, int r, int k, int v) {
378         if (l == r) {
379             info[l].x = v;
380             return;
381         }
382         int m = (l + r) / 2;
383         update(l, m, k, v);
384         update(m + 1, r, k, v);
385         info[m].x = info[l].x;
386         info[m].y = info[r].y;
387     }
388
389     int getRank(int l, int r, int x) {
390         if (l == r) return 1;
391         int m = (l + r) / 2;
392         if (x < info[m].x) return getRank(l, m, x);
393         else return getRank(m + 1, r, x) + 1;
394     }
395
396     int getPre(int l, int r, int x) {
397         if (l == r) return -1;
398         int m = (l + r) / 2;
399         if (x < info[m].x) return getPre(l, m, x);
400         else return info[m].x;
401     }
402
403     int getSucc(int l, int r, int x) {
404         if (l == r) return -1;
405         int m = (l + r) / 2;
406         if (x < info[m].x) return info[m].x;
407         else return getSucc(m + 1, r, x);
408     }
409
410     void erase(int l, int r, int x) {
411         if (l == r) {
412             info[l].x = -1;
413             return;
414         }
415         int m = (l + r) / 2;
416         erase(l, m, x);
417         erase(m + 1, r, x);
418         info[m].x = info[l].x;
419         info[m].y = info[r].y;
420     }
421
422     int kth(int l, int r, int k) {
423         if (l == r) return info[l].x;
424         int m = (l + r) / 2;
425         if (k <= info[m].y) return kth(l, m, k);
426         else return kth(m + 1, r, k - info[m].y);
427     }
428
429     void update(int l, int r, int k, int v) {
430         if (l == r) {
431             info[l].x = v;
432             return;
433         }
434         int m = (l + r) / 2;
435         update(l, m, k, v);
436         update(m + 1, r, k, v);
437         info[m].x = info[l].x;
438         info[m].y = info[r].y;
439     }
440
441     int getRank(int l, int r, int x) {
442         if (l == r) return 1;
443         int m = (l + r) / 2;
444         if (x < info[m].x) return getRank(l, m, x);
445         else return getRank(m + 1, r, x) + 1;
446     }
447
448     int getPre(int l, int r, int x) {
449         if (l == r) return -1;
450         int m = (l + r) / 2;
451         if (x < info[m].x) return getPre(l, m, x);
452         else return info[m].x;
453     }
454
455     int getSucc(int l, int r, int x) {
456         if (l == r) return -1;
457         int m = (l + r) / 2;
458         if (x < info[m].x) return info[m].x;
459         else return getSucc(m + 1, r, x);
460     }
461
462     void erase(int l, int r, int x) {
463         if (l == r) {
464             info[l].x = -1;
465             return;
466         }
467         int m = (l + r) / 2;
468         erase(l, m, x);
469         erase(m + 1, r, x);
470         info[m].x = info[l].x;
471         info[m].y = info[r].y;
472     }
473
474     int kth(int l, int r, int k) {
475         if (l == r) return info[l].x;
476         int m = (l + r) / 2;
477         if (k <= info[m].y) return kth(l, m, k);
478         else return kth(m + 1, r, k - info[m].y);
479     }
480
481     void update(int l, int r, int k, int v) {
482         if (l == r) {
483             info[l].x = v;
484             return;
485         }
486         int m = (l + r) / 2;
487         update(l, m, k, v);
488         update(m + 1, r, k, v);
489         info[m].x = info[l].x;
490         info[m].y = info[r].y;
491     }
492
493     int getRank(int l, int r, int x) {
494         if (l == r) return 1;
495         int m = (l + r) / 2;
496         if (x < info[m].x) return getRank(l, m, x);
497         else return getRank(m + 1, r, x) + 1;
498     }
499
500     int getPre(int l, int r, int x) {
501         if (l == r) return -1;
502         int m = (l + r) / 2;
503         if (x < info[m].x) return getPre(l, m, x);
504         else return info[m].x;
505     }
506
507     int getSucc(int l, int r, int x) {
508         if (l == r) return -1;
509         int m = (l + r) / 2;
510         if (x < info[m].x) return info[m].x;
511         else return getSucc(m + 1, r, x);
512     }
513
514     void erase(int l, int r, int x) {
515         if (l == r) {
516             info[l].x = -1;
517             return;
518         }
519         int m = (l + r) / 2;
520         erase(l, m, x);
521         erase(m + 1, r, x);
522         info[m].x = info[l].x;
523         info[m].y = info[r].y;
524     }
525
526     int kth(int l, int r, int k) {
527         if (l == r) return info[l].x;
528         int m = (l + r) / 2;
529         if (k <= info[m].y) return kth(l, m, k);
530         else return kth(m + 1, r, k - info[m].y);
531     }
532
533     void update(int l, int r, int k, int v) {
534         if (l == r) {
535             info[l].x = v;
536             return;
537         }
538         int m = (l + r) / 2;
539         update(l, m, k, v);
540         update(m + 1, r, k, v);
541         info[m].x = info[l].x;
542         info[m].y = info[r].y;
543     }
544
545     int getRank(int l, int r, int x) {
546         if (l == r) return 1;
547         int m = (l + r) / 2;
548         if (x < info[m].x) return getRank(l, m, x);
549         else return getRank(m + 1, r, x) + 1;
550     }
551
552     int getPre(int l, int r, int x) {
553         if (l == r) return -1;
554         int m = (l + r) / 2;
555         if (x < info[m].x) return getPre(l, m, x);
556         else return info[m].x;
557     }
558
559     int getSucc(int l, int r, int x) {
560         if (l == r) return -1;
561         int m = (l + r) / 2;
562         if (x < info[m].x) return info[m].x;
563         else return getSucc(m + 1, r, x);
564     }
565
566     void erase(int l, int r, int x) {
567         if (l == r) {
568             info[l].x = -1;
569             return;
570         }
571         int m = (l + r) / 2;
572         erase(l, m, x);
573         erase(m + 1, r, x);
574         info[m].x = info[l].x;
575         info[m].y = info[r].y;
576     }
577
578     int kth(int l, int r, int k) {
579         if (l == r) return info[l].x;
580         int m = (l + r) / 2;
581         if (k <= info[m].y) return kth(l, m, k);
582         else return kth(m + 1, r, k - info[m].y);
583     }
584
585     void update(int l, int r, int k, int v) {
586         if (l == r) {
587             info[l].x = v;
588             return;
589         }
590         int m = (l + r) / 2;
591         update(l, m, k, v);
592         update(m + 1, r, k, v);
593         info[m].x = info[l].x;
594         info[m].y = info[r].y;
595     }
596
597     int getRank(int l, int r, int x) {
598         if (l == r) return 1;
599         int m = (l + r) / 2;
600         if (x < info[m].x) return getRank(l, m, x);
601         else return getRank(m + 1, r, x) + 1;
602     }
603
604     int getPre(int l, int r, int x) {
605         if (l == r) return -1;
606         int m = (l + r) / 2;
607         if (x < info[m].x) return getPre(l, m, x);
608         else return info[m].x;
609     }
610
611     int getSucc(int l, int r, int x) {
612         if (l == r) return -1;
613         int m = (l + r) / 2;
614         if (x < info[m].x) return info[m].x;
615         else return getSucc(m + 1, r, x);
616     }
617
618     void erase(int l, int r, int x) {
619         if (l == r) {
620             info[l].x = -1;
621             return;
622         }
623         int m = (l + r) / 2;
624         erase(l, m, x);
625         erase(m + 1, r, x);
626         info[m].x = info[l].x;
627         info[m].y = info[r].y;
628     }
629
630     int kth(int l, int r, int k) {
631         if (l == r) return info[l].x;
632         int m = (l + r) / 2;
633         if (k <= info[m].y) return kth(l, m, k);
634         else return kth(m + 1, r, k - info[m].y);
635     }
636
637     void update(int l, int r, int k, int v) {
638         if (l == r) {
639             info[l].x = v;
640             return;
641         }
642         int m = (l + r) / 2;
643         update(l, m, k, v);
644         update(m + 1, r, k, v);
645         info[m].x = info[l].x;
646         info[m].y = info[r].y;
647     }
648
649     int getRank(int l, int r, int x) {
650         if (l == r) return 1;
651         int m = (l + r) / 2;
652         if (x < info[m].x) return getRank(l, m, x);
653         else return getRank(m + 1, r, x) + 1;
654     }
655
656     int getPre(int l, int r, int x) {
657         if (l == r) return -1;
658         int m = (l + r) / 2;
659         if (x < info[m].x) return getPre(l, m, x);
660         else return info[m].x;
661     }
662
663     int getSucc(int l, int r, int x) {
664         if (l == r) return -1;
665         int m = (l + r) / 2;
666         if (x < info[m].x) return info[m].x;
667         else return getSucc(m + 1, r, x);
668     }
669
670     void erase(int l, int r, int x) {
671         if (l == r) {
672             info[l].x = -1;
673             return;
674         }
675         int m = (l + r) / 2;
676         erase(l, m, x);
677         erase(m + 1, r, x);
678         info[m].x = info[l].x;
679         info[m].y = info[r].y;
680     }
681
682     int kth(int l, int r, int k) {
683         if (l == r) return info[l].x;
684         int m = (l + r) / 2;
685         if (k <= info[m].y) return kth(l, m, k);
686         else return kth(m + 1, r, k - info[m].y);
687     }
688
689     void update(int l, int r, int k, int v) {
690         if (l == r) {
691             info[l].x = v;
692             return;
693         }
694         int m = (l + r) / 2;
695         update(l, m, k, v);
696         update(m + 1, r, k, v);
697         info[m].x = info[l].x;
698         info[m].y = info[r].y;
699     }
700
701     int getRank(int l, int r, int x) {
702         if (l == r) return 1;
703         int m = (l + r) / 2;
704         if (x < info[m].x) return getRank(l, m, x);
705         else return getRank(m + 1, r, x) + 1;
706     }
707
708     int getPre(int l, int r, int x) {
709         if (l == r) return -1;
710         int m = (l + r) / 2;
711         if (x < info[m].x) return getPre(l, m, x);
712         else return info[m].x;
713     }
714
715     int getSucc(int l, int r, int x) {
716         if (l == r) return -1;
717         int m = (l + r) / 2;
718         if (x < info[m].x) return info[m].x;
719         else return getSucc(m + 1, r, x);
720     }
721
722     void erase(int l, int r, int x) {
723         if (l == r) {
724             info[l].x = -1;
725             return;
726         }
727         int m = (l + r) / 2;
728         erase(l, m, x);
729         erase(m + 1, r, x);
730         info[m].x = info[l].x;
731         info[m].y = info[r].y;
732     }
733
734     int kth(int l, int r, int k) {
735         if (l == r) return info[l].x;
736         int m = (l + r) / 2;
737         if (k <= info[m].y) return kth(l, m, k);
738         else return kth(m + 1, r, k - info[m].y);
739     }
740
741     void update(int l, int r, int k, int v) {
742         if (l == r) {
743             info[l].x = v;
744             return;
745         }
746         int m = (l + r) / 2;
747         update(l, m, k, v);
748         update(m + 1, r, k, v);
749         info[m].x = info[l].x;
750         info[m].y = info[r].y;
751     }
752
753     int getRank(int l, int r, int x) {
754         if (l == r) return 1;
755         int m = (l + r) / 2;
756         if (x < info[m].x) return getRank(l, m, x);
757         else return getRank(m + 1, r, x) + 1;
758     }
759
760     int getPre(int l, int r, int x) {
761         if (l == r) return -1;
762         int m = (l + r) / 2;
763         if (x < info[m].x) return getPre(l, m, x);
764         else return info[m].x;
765     }
766
767     int getSucc(int l, int r, int x) {
768         if (l == r) return -1;
769         int m = (l + r) / 2;
770         if (x < info[m].x) return info[m].x;
771         else return getSucc(m + 1, r, x);
772     }
773
774     void erase(int l, int r, int x) {
775         if (l == r) {
776             info[l].x = -1;
777             return;
778         }
779         int m = (l + r) / 2;
780         erase(l, m, x);
781         erase(m + 1, r, x);
782         info[m].x = info[l].x;
783         info[m].y = info[r].y;
784     }
785
786     int kth(int l, int r, int k) {
787         if (l == r) return info[l].x;
788         int m = (l + r) / 2;
789         if (k <= info[m].y) return kth(l, m, k);
790         else return kth(m + 1, r, k - info[m].y);
791     }
792
793     void update(int l, int r, int k, int v) {
794         if (l == r) {
795             info[l].x = v;
796             return;
797         }
798         int m = (l + r) / 2;
799         update(l, m, k, v);
800         update(m + 1, r, k, v);
801         info[m].x = info[l].x;
802         info[m].y = info[r].y;
803     }
804
805     int getRank(int l, int r, int x) {
806         if (l == r) return 1;
807         int m = (l + r) / 2;
808         if (x < info[m].x) return getRank(l, m, x);
809         else return getRank(m + 1, r, x) + 1;
810     }
811
812     int getPre(int l, int r, int x) {
813         if (l == r) return -1;
814         int m = (l + r) / 2;
815         if (x < info[m].x) return getPre(l, m, x);
816         else return info[m].x;
817     }
818
819     int getSucc(int l, int r, int x) {
820         if (l == r) return -1;
821         int m = (l + r) / 2;
822         if (x < info[m].x) return info[m].x;
823         else return getSucc(m + 1, r, x);
824     }
825
826     void erase(int l, int r, int x) {
827         if (l == r) {
828             info[l].x = -1;
829             return;
830         }
831         int m = (l + r) / 2;
832         erase(l, m, x);
833         erase(m + 1, r, x);
834         info[m].x = info[l].x;
835         info[m].y = info[r].y;
836     }
837
838     int kth(int l, int r, int k) {
839         if (l == r) return info[l].x;
840         int m = (l + r) / 2;
841         if (k <= info[m].y) return kth(l, m, k);
842         else return kth(m + 1, r, k - info[m].y);
843     }
844
845     void update(int l, int r, int k, int v) {
846         if (l == r) {
847             info[l].x = v;
848             return;
849         }
850         int m = (l + r) / 2;
851         update(l, m, k, v);
852         update(m + 1, r, k, v);
853         info[m].x = info[l].x;
854         info[m].y = info[r].y;
855     }
856
857     int getRank(int l, int r, int x) {
858         if (l == r) return 1;
859         int m = (l + r) / 2;
860         if (x < info[m].x) return getRank(l, m, x);
861         else return getRank(m + 1, r, x) + 1;
862     }
863
864     int getPre(int l, int r, int x) {
865         if (l == r) return -1;
866         int m = (l + r) / 2;
867         if (x < info[m].x) return getPre(l, m, x);
868         else return info[m].x;
869     }
870
871     int getSucc(int l, int r, int x) {
872         if (l == r) return -1;
873         int m = (l + r) / 2;
874         if (x < info[m].x) return info[m].x;
875         else return getSucc(m + 1, r, x);
876     }
877
878     void erase(int l, int r, int x) {
879         if (l == r) {
880             info[l].x = -1;
881             return;
882         }
883         int m = (l + r) / 2;
884         erase(l, m, x);
885         erase(m + 1, r, x);
886         info[m].x = info[l].x;
887         info[m].y = info[r].y;
888     }
889
890     int kth(int l, int r, int k) {
891         if (
```



```

25     }
26
27 private:
28     void _build(int p, int l, int r, const std::vector<int>& a) {
29         // 从叶子节点开始，向上启发式合并来构建整棵树
30         if (r - l == 1) {
31             if (l < a.size()) info[p].ver.insert({ a[l], 1 });
32             return;
33         }
34         int m = (l + r) / 2;
35         _build(2 * p, l, m, a);
36         _build(2 * p + 1, m, r, a);
37
38         // 启发式合并（小树合并到大树）
39         if (info[2 * p].ver.size() > info[2 * p + 1].ver.size()) {
40             info[p].ver = info[2 * p].ver;
41             for (const auto& item : info[2 * p + 1].ver)
42                 info[p].ver.insert(item);
43         }
44         else {
45             info[p].ver = info[2 * p + 1].ver;
46             for (const auto& item : info[2 * p].ver)
47                 info[p].ver.insert(item);
48         }
49     }
50
51 public:
52     // 单点修改
53     void update(int pos, int old_val, int new_val) {
54         _update(1, 0, n, pos, old_val, new_val);
55     }
56
57     // 查询排名（返回比 k 小的数的个数）
58     int query_rank_count(int l, int r, int k) {
59         return _query_rank_count(1, 0, n, l, r, k);
60     }
61
62     // 查询前驱
63     int query_pred(int l, int r, int k) {
64         return _query_pred(1, 0, n, l, r, k);
65     }
66
67     // 查询后继
68     int query_succ(int l, int r, int k) {
69         return _query_succ(1, 0, n, l, r, k);
70     }
71
72 private:
73     void _update(int p, int l, int r, int pos, int old_val, int new_val) {
74         info[p].ver.erase({ old_val, pos });
75         info[p].ver.insert({ new_val, pos });
76         if (r - l == 1) return;
77         int m = (l + r) / 2;
78         if (pos < m) _update(2 * p, l, m, pos, old_val, new_val);
79         else _update(2 * p + 1, m, r, pos, old_val, new_val);
80     }

```

```

79
80     int _query_rank_count(int p, int l, int r, int x, int y, int k) {
81         if (l >= y || r <= x) return 0;
82         if (l >= x && r <= y) {
83             return info[p].ver.order_of_key({ k, -1 });
84         }
85         int m = (l + r) / 2;
86         return _query_rank_count(2 * p, l, m, x, y, k) +
87             _query_rank_count(2 * p + 1, m, r, x, y, k);
88     }
89
90     int _query_pred(int p, int l, int r, int x, int y, int k) {
91         if (l >= y || r <= x) return -inf;
92         if (l >= x && r <= y) {
93             auto it = info[p].ver.lower_bound({ k, -1 });
94             if (it == info[p].ver.begin()) return -inf;
95             return (--it)->first;
96         }
97         int m = (l + r) / 2;
98         return std::max(_query_pred(2 * p, l, m, x, y, k), _query_pred(2 *
99             p + 1, m, r, x, y, k));
100     }
101
102     int _query_succ(int p, int l, int r, int x, int y, int k) {
103         if (l >= y || r <= x) return inf;
104         if (l >= x && r <= y) {
105             auto it = info[p].ver.upper_bound({ k, inf });
106             if (it == info[p].ver.end()) return inf;
107             return it->first;
108         }
109         int m = (l + r) / 2;
110         return std::min(_query_succ(2 * p, l, m, x, y, k), _query_succ(2 *
111             p + 1, m, r, x, y, k));
112     }
113 };
114
115 struct Info {
116     tree<pii, null_type, std::less<pii>, rb_tree_tag,
117         tree_order_statistics_node_update> ver;
118 };

```

坐标压缩与离散化

简单版本

```

1 | sort(alls.begin(), alls.end());
2 | alls.erase(unique(alls.begin(), alls.end()), alls.end());
3 | auto get = [&](int x) {
4 |     return lower_bound(alls.begin(), alls.end(), x) - alls.begin();
5 | };

```

```

1  template <typename T> struct Compress_ {
2      int n, shift = 0; // shift 用于标记下标偏移量
3      vector<T> alls;
4
5      Compress_() {}
6      Compress_(auto in) : alls(in) {
7          init();
8      }
9      void add(T x) {
10         alls.emplace_back(x);
11     }
12     template <typename... Args> void add(T x, Args... args) {
13         add(x), add(args...);
14     }
15     void init() {
16         alls.emplace_back(numeric_limits<T>::max());
17         sort(alls.begin(), alls.end());
18         alls.erase(unique(alls.begin(), alls.end()), alls.end());
19         this->n = alls.size();
20     }
21     int size() {
22         return n;
23     }
24     int operator[](T x) { // 返回 x 元素的新下标
25         return upper_bound(alls.begin(), alls.end(), x) - alls.begin() +
shift;
26     }
27     T Get(int x) { // 根据新下标返回原来元素
28         assert(x - shift < n);
29         return x - shift < n ? alls[x - shift] : -1;
30     }
31     bool count(T x) { // 查找元素 x 是否存在
32         return binary_search(alls.begin(), alls.end(), x);
33     }
34     friend auto &operator<< (ostream &o, const auto &j) {
35         cout << "{";
36         for (auto it : j.alls) {
37             o << it << " ";
38         }
39         return o << "}";
40     }
41 };
42 using Compress = Compress_<int>;

```

轻重链剖分/树链剖分

将线段树处理的部分分离，方便修改。支持链上查询/修改、子树查询/修改，建树时间复杂度 $\mathcal{O}(N \log N)$ ，单次查询时间复杂度 $\mathcal{O}(\log^2 N)$ 。

```

1  struct Segt {
2      struct node {
3          int l, r, w, lazy;

```

```

4     };
5     vector<int> w;
6     vector<node> t;
7
8     Segt() {}
9     #define GL (k << 1)
10    #define GR (k << 1 | 1)
11
12    void init(vector<int> in) {
13        int n = in.size() - 1;
14        w.resize(n + 1);
15        for (int i = 1; i <= n; i++) {
16            w[i] = in[i];
17        }
18        t.resize(n * 4 + 1);
19        auto build = [&](auto self, int l, int r, int k = 1) {
20            if (l == r) {
21                t[k] = {l, r, w[l], 0}; // 如果有赋值为 0 的操作，则懒标记必须
要 -1
22                return;
23            }
24            t[k] = {l, r};
25            int mid = (l + r) / 2;
26            self(self, l, mid, GL);
27            self(self, mid + 1, r, GR);
28            pushup(k);
29        };
30        build(build, 1, n);
31    }
32    void pushdown(node &p, int lazy) { /* 【在此更新下递函数】 */
33        p.w += (p.r - p.l + 1) * lazy;
34        p.lazy += lazy;
35    }
36    void pushdown(int k) { // 不需要动
37        if (t[k].lazy == 0) return;
38        pushdown(t[GL], t[k].lazy);
39        pushdown(t[GR], t[k].lazy);
40        t[k].lazy = 0;
41    }
42    void pushup(int k) { // 不需要动
43        auto pushup = [&](node &p, node &l, node &r) { /* 【在此更新上传函
数】 */
44            p.w = l.w + r.w;
45        };
46        pushup(t[k], t[GL], t[GR]);
47    }
48    void modify(int l, int r, int val, int k = 1) {
49        if (l <= t[k].l && t[k].r <= r) {
50            pushdown(t[k], val);
51            return;
52        }
53        pushdown(k);
54        int mid = (t[k].l + t[k].r) / 2;
55        if (l <= mid) modify(l, r, val, GL);
56        if (mid < r) modify(l, r, val, GR);
57        pushup(k);

```

```

58     }
59     int ask(int l, int r, int k = 1) {
60         if (l <= t[k].l && t[k].r <= r) {
61             return t[k].w;
62         }
63         pushdown(k);
64         int mid = (t[k].l + t[k].r) / 2;
65         int ans = 0;
66         if (l <= mid) ans += ask(l, r, GL);
67         if (mid < r) ans += ask(l, r, GR);
68         return ans;
69     }
70 };
71
72 struct HLD {
73     int n, idx;
74     vector<vector<int>> ver;
75     vector<int> siz, dep;
76     vector<int> top, son, parent;
77     vector<int> in, id, val;
78     Segt segt;
79
80     HLD(int n) {
81         this->n = n;
82         ver.resize(n + 1);
83         siz.resize(n + 1);
84         dep.resize(n + 1);
85
86         top.resize(n + 1);
87         son.resize(n + 1);
88         parent.resize(n + 1);
89
90         idx = 0;
91         in.resize(n + 1);
92         id.resize(n + 1);
93         val.resize(n + 1);
94     }
95     void add(int x, int y) { // 建立双向边
96         ver[x].push_back(y);
97         ver[y].push_back(x);
98     }
99     void dfs1(int x) {
100         siz[x] = 1;
101         dep[x] = dep[parent[x]] + 1;
102         for (auto y : ver[x]) {
103             if (y == parent[x]) continue;
104             parent[y] = x;
105             dfs1(y);
106             siz[x] += siz[y];
107             if (siz[y] > siz[son[x]]) {
108                 son[x] = y;
109             }
110         }
111     }
112     void dfs2(int x, int up) {
113         id[x] = ++idx;

```

```

114     val[idx] = in[x]; // 建立编号
115     top[x] = up;
116     if (son[x]) dfs2(son[x], up);
117     for (auto y : ver[x]) {
118         if (y == parent[x] || y == son[x]) continue;
119         dfs2(y, y);
120     }
121 }
122 void modify(int l, int r, int val) { // 链上修改
123     while (top[l] != top[r]) {
124         if (dep[top[l]] < dep[top[r]]) {
125             swap(l, r);
126         }
127         segt.modify(id[top[l]], id[l], val);
128         l = parent[top[l]];
129     }
130     if (dep[l] > dep[r]) {
131         swap(l, r);
132     }
133     segt.modify(id[l], id[r], val);
134 }
135 void modify(int root, int val) { // 子树修改
136     segt.modify(id[root], id[root] + siz[root] - 1, val);
137 }
138 int ask(int l, int r) { // 链上查询
139     int ans = 0;
140     while (top[l] != top[r]) {
141         if (dep[top[l]] < dep[top[r]]) {
142             swap(l, r);
143         }
144         ans += segt.ask(id[top[l]], id[l]);
145         l = parent[top[l]];
146     }
147     if (dep[l] > dep[r]) {
148         swap(l, r);
149     }
150     return ans + segt.ask(id[l], id[r]);
151 }
152 int ask(int root) { // 子树查询
153     return segt.ask(id[root], id[root] + siz[root] - 1);
154 }
155 void work(auto in, int root = 1) { // 在此初始化
156     assert(in.size() == n + 1);
157     this->in = in;
158     dfs1(root);
159     dfs2(root, root);
160     segt.init(val); // 建立线段树
161 }
162 void work(int root = 1) { // 在此初始化
163     dfs1(root);
164     dfs2(root, root);
165     segt.init(val); // 建立线段树
166 }
167 };

```

小波矩阵树：高效静态区间第 K 大查询

手写 `bitset` 压位, 以 $\mathcal{O}(N \log N)$ 的时间复杂度和 $\mathcal{O}(N + \frac{N \log N}{64})$ 的空间建树后, 实现单次 $\mathcal{O}(\log N)$ 复杂度的区间第 k 大值询问。建议使用 0-idx 计数法, 但是经测试 1-idx 也有效, 但需要更多的检验。

```
1  #define __count(x) __builtin_popcountll(x)
2  struct wavelet {
3      vector<int> val, sum;
4      vector<u64> bit;
5      int t, n;
6
7      int getSum(int i) {
8          return sum[i >> 6] + __count(bit[i >> 6] & ((1ULL << (i & 63)) -
9  1));
10     }
11
12     Wavelet(vector<int> v) : val(v), n(v.size()) {
13         sort(val.begin(), val.end());
14         val.erase(unique(val.begin(), val.end()), val.end());
15
16         int n_ = val.size();
17         t = __lg(2 * n_ - 1);
18         bit.resize((t * n + 64) >> 6);
19         sum.resize(bit.size());
20         vector<int> cnt(n_ + 1);
21
22         for (int &x : v) {
23             x = lower_bound(val.begin(), val.end(), x) - val.begin();
24             cnt[x + 1]++;
25         }
26         for (int i = 1; i < n_; ++i) {
27             cnt[i] += cnt[i - 1];
28         }
29         for (int j = 0; j < t; ++j) {
30             for (int i : v) {
31                 int tmp = i >> (t - 1 - j);
32                 int pos = (tmp >> 1) << (t - j);
33                 auto setBit = [&](int i, u64 v) {
34                     bit[i >> 6] |= (v << (i & 63));
35                 };
36                 setBit(j * n + cnt[pos], tmp & 1);
37                 cnt[pos]++;
38             }
39             for (int i : v) {
40                 cnt[(i >> (t - j)) << (t - j)]--;
41             }
42         }
43         for (int i = 1; i < sum.size(); ++i) {
44             sum[i] = sum[i - 1] + __count(bit[i - 1]);
45         }
46
47         int small(int l, int r, int k) {
48             r++;
```

```

49     for (int j = 0, x = 0, y = n, res = 0;; ++j) {
50         if (j == t) return val[res];
51         int A = getSum(n * j + x), B = getSum(n * j + 1);
52         int C = getSum(n * j + r), D = getSum(n * j + y);
53         int ab_zeros = r - 1 - C + B;
54         if (ab_zeros > k) {
55             res = res << 1;
56             y -= D - A;
57             l -= B - A;
58             r -= C - A;
59         } else {
60             res = (res << 1) | 1;
61             k -= ab_zeros;
62             x += y - x - D + A;
63             l += y - l - D + B;
64             r += y - r - D + C;
65         }
66     }
67 }
68 int large(int l, int r, int k) {
69     return small(l, r, r - l - k);
70 }
71 };

```

普通莫队

以 $\mathcal{O}(N\sqrt{N})$ 的复杂度完成 Q 次询问的离线查询，其中每个分块的大小取 $\sqrt{N} = \sqrt{10^5} = 317$ ，也可以使用 `n / min<int>(n, sqrt(q))`、`ceil((double)n / (int)sqrt(n))` 或者 `sqrt(n)` 划分。

```

1  signed main() {
2      int n;
3      cin >> n;
4      vector<int> w(n + 1);
5      for (int i = 1; i <= n; i++) {
6          cin >> w[i];
7      }
8
9      int q;
10     cin >> q;
11     vector<array<int, 3>> query(q + 1);
12     for (int i = 1; i <= q; i++) {
13         int l, r;
14         cin >> l >> r;
15         query[i] = {l, r, i};
16     }
17
18     int knum = n / min<int>(n, sqrt(q)); // 计算块长
19     vector<int> k(n + 1);
20     for (int i = 1; i <= n; i++) { // 固定块长
21         k[i] = (i - 1) / knum + 1;
22     }
23     sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
24         if (k[x[0]] != k[y[0]]) return x[0] < y[0];

```



```

25         if (k[x[0]] & 1) return x[1] < y[1];
26         return x[1] > y[1];
27     });
28
29     int l = 1, r = 0, val = 0;
30     vector<int> ans(q + 1);
31     for (int i = 1; i <= q; i++) {
32         auto [ql, qr, id] = query[i];
33         auto add = [&](int x) -> void {};
34         auto del = [&](int x) -> void {};
35         while (l > ql) add(w[--l]);
36         while (r < qr) add(w[++r]);
37         while (l < ql) del(w[l++]);
38         while (r > qr) del(w[r--]);
39         ans[id] = val;
40     }
41     for (int i = 1; i <= q; i++) {
42         cout << ans[i] << endl;
43     }
44 }

```

需要注意的是，在普通莫队中，`k` 数组的作用是根据左边界的值进行排序，当询问次数很少时 ($q \ll n$)，可以直接合并到 `query` 数组中。

```

1 vector<array<int, 4>> query(q);
2 for (int i = 1; i <= q; i++) {
3     int l, r;
4     cin >> l >> r;
5     query[i] = {l, r, i, (l - 1) / knum + 1}; // 合并
6 }
7 sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
8     if (x[3] != y[3]) return x[3] < y[3];
9     if (x[3] & 1) return x[1] < y[1];
10    return x[1] > y[1];
11 });

```

带修改的莫队（带时间维度的莫队）

以 $\mathcal{O}(N^{\frac{5}{3}})$ 的复杂度完成 Q 次询问的离线查询，其中每个分块的大小取 $N^{\frac{2}{3}} = \sqrt[3]{100000^2} = 2154$ （直接取会略快），也可以使用 `pow(n, 0.6666)` 划分。

```

1 signed main() {
2     int n, q;
3     cin >> n >> q;
4     vector<int> w(n + 1);
5     for (int i = 1; i <= n; i++) {
6         cin >> w[i];
7     }
8
9     vector<array<int, 4>> query = {}; // {左区间, 右区间, 累计修改次数, 下标}
10    vector<array<int, 2>> modify = {}; // {修改的值, 修改的元素下标}
11    for (int i = 1; i <= q; i++) {
12        char op;
13        cin >> op;

```

```

14         if (op == 'Q') {
15             int l, r;
16             cin >> l >> r;
17             query.push_back({l, r, (int)modify.size() - 1,
(int)query.size()});
18         } else {
19             int idx, w;
20             cin >> idx >> w;
21             modify.push_back({w, idx});
22         }
23     }
24
25     int Knum = 2154; // 计算块长
26     vector<int> K(n + 1);
27     for (int i = 1; i <= n; i++) { // 固定块长
28         K[i] = (i - 1) / Knum + 1;
29     }
30     sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
31         if (K[x[0]] != K[y[0]]) return x[0] < y[0];
32         if (K[x[1]] != K[y[1]]) return x[1] < y[1];
33         return x[3] < y[3];
34     });
35
36     int l = 1, r = 0, val = 0;
37     int t = 0; // 累计修改次数
38     vector<int> ans(query.size());
39     for (int i = 1; i < query.size(); i++) {
40         auto [ql, qr, qt, id] = query[i];
41         auto add = [&](int x) -> void {};
42         auto del = [&](int x) -> void {};
43         auto time = [&](int x, int l, int r) -> void {};
44         while (l > ql) add(w[--l]);
45         while (r < qr) add(w[++r]);
46         while (l < ql) del(w[l++]);
47         while (r > qr) del(w[r--]);
48         while (t < qt) time(++t, ql, qr);
49         while (t > qt) time(t--, ql, qr);
50         ans[id] = val;
51     }
52     for (int i = 1; i < ans.size(); i++) {
53         cout << ans[i] << endl;
54     }
55 }

```

对顶堆

```

1 namespace Set {
2     const int kInf = 1e9 + 2077;
3     std::multiset<int> less, greater;
4     void init() {
5         less.clear(), greater.clear();
6         less.insert(-kInf), greater.insert(kInf);
7     }
8     void adjust() {
9         while (less.size() > greater.size() + 1) {

```

```

10         std::multiset<int>::iterator it = (--less.end());
11         greater.insert(*it);
12         less.erase(it);
13     }
14     while (greater.size() > less.size()) {
15         std::multiset<int>::iterator it = greater.begin();
16         less.insert(*it);
17         greater.erase(it);
18     }
19 }
20 void add(int val_) {
21     if (val_ <= *greater.begin()) less.insert(val_);
22     else greater.insert(val_);
23     adjust();
24 }
25 void del(int val_) {
26     std::multiset<int>::iterator it = less.lower_bound(val_);
27     if (it != less.end()) {
28         less.erase(it);
29     }
30     else {
31         it = greater.lower_bound(val_);
32         greater.erase(it);
33     }
34     adjust();
35 }
36 int get_middle() {
37     return *less.rbegin();
38 }
39 }

```

主席树（可持久化线段树）

以 $\mathcal{O}(N \log N)$ 的时间复杂度建树、查询、修改。

```

1  struct PresidentTree {
2      static constexpr int N = 2e5 + 10;
3      int cntNodes, root[N];
4      struct node {
5          int l, r;
6          int cnt;
7      } tr[4 * N + 17 * N];
8      void modify(int &u, int v, int l, int r, int x) {
9          u = ++cntNodes;
10         tr[u] = tr[v];
11         tr[u].cnt++;
12         if (l == r) return;
13         int mid = (l + r) / 2;
14         if (x <= mid)
15             modify(tr[u].l, tr[v].l, l, mid, x);
16         else
17             modify(tr[u].r, tr[v].r, mid + 1, r, x);
18     }
19     int kth(int u, int v, int l, int r, int k) {
20         if (l == r) return l;

```

```

21     int res = tr[tr[v].l].cnt - tr[tr[u].l].cnt;
22     int mid = (l + r) / 2;
23     if (k <= res)
24         return kth(tr[u].l, tr[v].l, l, mid, k);
25     else
26         return kth(tr[u].r, tr[v].r, mid + 1, r, k - res);
27 }
28 };

```

KD Tree

在第 k 维上的单次查询复杂度最坏为 $\mathcal{O}(n^{1-k^{-1}})$ 。

```

1  struct KDT {
2      constexpr static int N = 1e5 + 10, K = 2;
3      double alpha = 0.725;
4      struct node {
5          int info[K];
6          int mn[K], mx[K];
7      } tr[N];
8      int ls[N], rs[N], siz[N], id[N], d[N];
9      int idx, rt, cur;
10     int ans;
11     KDT() {
12         rt = 0;
13         cur = 0;
14         memset(ls, 0, sizeof ls);
15         memset(rs, 0, sizeof rs);
16         memset(d, 0, sizeof d);
17     }
18     void apply(int p, int son) {
19         if (son) {
20             for (int i = 0; i < K; i++) {
21                 tr[p].mn[i] = min(tr[p].mn[i], tr[son].mn[i]);
22                 tr[p].mx[i] = max(tr[p].mx[i], tr[son].mx[i]);
23             }
24             siz[p] += siz[son];
25         }
26     }
27     void maintain(int p) {
28         for (int i = 0; i < K; i++) {
29             tr[p].mn[i] = tr[p].info[i];
30             tr[p].mx[i] = tr[p].info[i];
31         }
32         siz[p] = 1;
33         apply(p, ls[p]);
34         apply(p, rs[p]);
35     }
36     int build(int l, int r) {
37         if (l > r) return 0;
38         vector<double> avg(K);
39         for (int i = 0; i < K; i++) {
40             for (int j = l; j <= r; j++) {
41                 avg[i] += tr[id[j]].info[i];
42             }

```

```

43         avg[i] /= (r - l + 1);
44     }
45     vector<double> var(K);
46     for (int i = 0; i < K; i++) {
47         for (int j = l; j <= r; j++) {
48             var[i] += (tr[id[j]].info[i] - avg[i]) *
(tr[id[j]].info[i] - avg[i]);
49         }
50     }
51     int mid = (l + r) / 2;
52     int x = max_element(var.begin(), var.end()) - var.begin();
53     nth_element(id + l, id + mid, id + r + 1, [&](int a, int b) {
54         return tr[a].info[x] < tr[b].info[x];
55     });
56     d[id[mid]] = x;
57     ls[id[mid]] = build(l, mid - 1);
58     rs[id[mid]] = build(mid + 1, r);
59     maintain(id[mid]);
60     return id[mid];
61 }
62 void print(int p) {
63     if (!p) return;
64     print(ls[p]);
65     id[++idx] = p;
66     print(rs[p]);
67 }
68 void rebuild(int &p) {
69     idx = 0;
70     print(p);
71     p = build(l, idx);
72 }
73 bool bad(int p) {
74     return alpha * siz[p] <= max(siz[ls[p]], siz[rs[p]]);
75 }
76 void insert(int &p, int cur) {
77     if (!p) {
78         p = cur;
79         maintain(p);
80         return;
81     }
82     if (tr[p].info[d[p]] > tr[cur].info[d[p]]) insert(ls[p], cur);
83     else insert(rs[p], cur);
84     maintain(p);
85     if (bad(p)) rebuild(p);
86 }
87 void insert(vector<int> &a) {
88     cur++;
89     for (int i = 0; i < K; i++) {
90         tr[cur].info[i] = a[i];
91     }
92     insert(rt, cur);
93 }
94 bool out(int p, vector<int> &a) {
95     for (int i = 0; i < K; i++) {
96         if (a[i] < tr[p].mn[i]) {
97             return true;

```

```

98         }
99     }
100     return false;
101 }
102 bool in(int p, vector<int> &a) {
103     for (int i = 0; i < K; i++) {
104         if (a[i] < tr[p].info[i]) {
105             return false;
106         }
107     }
108     return true;
109 }
110 bool all(int p, vector<int> &a) {
111     for (int i = 0; i < K; i++) {
112         if (a[i] < tr[p].mx[i]) {
113             return false;
114         }
115     }
116     return true;
117 }
118 void query(int p, vector<int> &a) {
119     if (!p) return;
120     if (out(p, a)) return;
121     if (all(p, a)) {
122         ans += siz[p];
123         return;
124     }
125     if (in(p, a)) ans++;
126     query(ls[p], a);
127     query(rs[p], a);
128 }
129 int query(vector<int> &a) {
130     ans = 0;
131     query(rt, a);
132     return ans;
133 }
134 };

```

/END/

