

1 Data Structures

1.1 Segment Tree

1.1.1 Lazy

```

1
2 struct RSQ // Range sum query
3 {
4     static ll const neutro = 0;
5     static ll op(ll x, ll y)
6     {
7         return x + y;
8     }
9     static ll
10    lazy_op(int i, int j, ll x)
11    {
12        return (j - i + 1) * x;
13    }
14 };
15
16 struct RMinQ // Range minimum query
17 {
18     static ll const neutro = 1e18;
19     static ll op(ll x, ll y)
20     {
21         return min(x, y);
22     }
23     static ll
24    lazy_op(int i, int j, ll x)
25    {
26        return x;
27    }
28 };
29
30 template <class t>
31 class SegTreeLazy
32 {
33     vector<ll> arr, st, lazy;
34     int n;
35
36     void build(int u, int i, int j)
37     {
38         if (i == j)
39         {
40             st[u] = arr[i];
41             return;
42         }
43         int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
44         build(l, i, m);
45         build(r, m + 1, j);
46         st[u] = t::op(st[l], st[r]);
47     }
48
49     void propagate(int u, int i, int j, ll x)

```

```

50     {
51         // nota, las operaciones pueden ser un and, or, ..., etc.
52         st[u] += t::lazy_op(i, j, x); // incrementar el valor (+)
53         st[u] = t::lazy_op(i, j, x); // setear el valor
54         if (i != j)
55         {
56             // incrementar el valor
57             lazy[u * 2 + 1] += x;
58             lazy[u * 2 + 2] += x;
59             // setear el valor
60             //lazy[u * 2 + 1] = x;
61             //lazy[u * 2 + 2] = x;
62         }
63         lazy[u] = 0;
64     }
65
66     ll query(int a, int b, int u, int i, int j)
67     {
68         if (j < a or b < i)
69             return t::neutro;
70         int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
71         if (lazy[u])
72             propagate(u, i, j, lazy[u]);
73         if (a <= i and j <= b)
74             return st[u];
75         ll x = query(a, b, l, i, m);
76         ll y = query(a, b, r, m + 1, j);
77         return t::op(x, y);
78     }
79
80     void update(int a, int b, ll value,
81                int u, int i, int j)
82     {
83         int m = (i + j) / 2, l = u * 2 + 1, r = u * 2 + 2;
84         if (lazy[u])
85             propagate(u, i, j, lazy[u]);
86         if (a <= i and j <= b)
87             propagate(u, i, j, value);
88         else if (j < a or b < i)
89             return;
90         else
91         {
92             update(a, b, value, l, i, m);
93             update(a, b, value, r, m + 1, j);
94             st[u] = t::op(st[l], st[r]);
95         }
96     }
97
98     public:
99     SegTreeLazy(vector<ll> &v)
100     {
101         arr = v;
102         n = v.size();
103         st.resize(n * 4 + 5);
104         lazy.assign(n * 4 + 5, 0);

```

```

105     build(0, 0, n - 1);
106 }
107
108 ll query(int a, int b)
109 {
110     return query(a, b, 0, 0, n - 1);
111 }
112
113 void update(int a, int b, ll value)
114 {
115     update(a, b, value, 0, 0, n - 1);
116 }
117 };

```

1.1.2 Iterative

```

1 // It requires a struct for a node (e.g. prodsgn)
2 // A node must have three constructors
3 // Arity 0: Constructs the identity of the operation (e.g. 1 for
4 //   prodsgn)
5 // Arity 1: Constructs a leaf node from the input
6 // Arity 2: Constructs a node from its children
7 //
8 // Building the Segment Tree:
9 //   Create a vector of nodes (use constructor of arity 1).
10 //   ST<miStructNode> mySegmentTree(vectorOfNodes);
11 // Update:
12 //   mySegmentTree.set_points(index, myStructNode(input));
13 // Query:
14 //   mySegmentTree.query(l, r); (It searches on the range [l,r), and
15 //   returns a node.)
16
17 // Logic And Query
18 struct ANDQ
19 {
20     ll value;
21     ANDQ() { value = -1ll; }
22     ANDQ(ll x) { value = x; }
23     ANDQ(const ANDQ &a,
24           const ANDQ &b)
25     {
26         value = a.value & b.value;
27     }
28 };
29
30 // Interval Product (LiveArchive)
31 struct prodsgn
32 {
33     int sgn;
34     prodsgn() { sgn = 1; }
35     prodsgn(int x)
36     {
37         sgn = (x > 0) - (x < 0);
38     }
39 };

```

```

38 prodsgn(const prodsgn &a,
39         const prodsgn &b)
40 {
41     sgn = a.sgn * b.sgn;
42 }
43 };
44
45 // Maximum Sum (SPOJ)
46 struct maxsum
47 {
48     int first, second;
49     maxsum() { first = second = -1; }
50     maxsum(int x)
51     {
52         first = x;
53         second = -1;
54     }
55     maxsum(const maxsum &a,
56           const maxsum &b)
57     {
58         if (a.first > b.first)
59         {
60             first = a.first;
61             second = max(a.second,
62                         b.first);
63         }
64         else
65         {
66             first = b.first;
67             second = max(a.first,
68                         b.second);
69         }
70     }
71     int answer()
72     {
73         return first + second;
74     }
75 };
76
77 // Range Minimum Query
78 struct rminq
79 {
80     int value;
81     rminq() { value = INT_MAX; }
82     rminq(int x) { value = x; }
83     rminq(const rminq &a,
84           const rminq &b)
85     {
86         value = min(a.value,
87                     b.value);
88     }
89 };
90
91 template <class node>
92 class ST

```

```

93 {
94     vector<node> t;
95     int n;
96
97 public:
98     ST(vector<node> &arr)
99     {
100         n = arr.size();
101         t.resize(n * 2);
102         copy(arr.begin(), arr.end(), t.begin() + n);
103         for (int i = n - 1; i > 0; --i)
104             t[i] = node(t[i << 1], t[i << 1 | 1]);
105     }
106
107     // 0-indexed
108     void set_point(int p, const node &value)
109     {
110         for (t[p += n] = value; p > 1; p >>= 1)
111             t[p >> 1] = node(t[p], t[p ^ 1]);
112     }
113
114     // inclusive exclusive, 0-indexed
115     node query(int l, int r)
116     {
117         node ans1, ansr;
118         for (l += n, r += n; l < r; l >>= 1, r >>= 1)
119         {
120             if (l & 1)
121                 ans1 = node(ans1, t[l++]);
122             if (r & 1)
123                 ansr = node(t[--r], ansr);
124         }
125         return node(ans1, ansr);
126     }
127 };

```

1.2 Wavelet Tree

```

1  class WaveTree
2  {
3
4      typedef vector<int>::iterator iter;
5      vector<vector<int>>> r0;
6      int n, s;
7      vector<int> arrCopy;
8
9      void build(iter b, iter e, int l, int r, int u)
10     {
11         if (l == r)
12             return;
13         int m = (l + r) / 2;
14         r0[u].reserve(e - b + 1);
15         r0[u].push_back(0);
16         for (iter it = b; it != e; ++it)
17             r0[u].push_back(r0[u].back() + (*it <= m));

```

```

18         iter p = stable_partition(b, e, [=](int i) { return i <= m; });
19         build(b, p, l, m, u * 2);
20         build(p, e, m + 1, r, u * 2 + 1);
21     }
22
23     int q, w;
24     int range(int a, int b, int l, int r, int u)
25     {
26         if (r < q or w < l)
27             return 0;
28         if (q <= l and r <= w)
29             return b - a;
30         int m = (l + r) / 2, za = r0[u][a], zb = r0[u][b];
31         return range(za, zb, l, m, u * 2) +
32             range(a - za, b - zb, m + 1, r, u * 2 + 1);
33     }
34
35 public:
36     //arr[i] in [0,sigma)
37     WaveTree(vector<int> arr, int sigma)
38     {
39         n = arr.size();
40         s = sigma;
41         r0.resize(s * 2);
42         arrCopy = arr;
43         build(arr.begin(), arr.end(), 0, s - 1, 1);
44     }
45
46     //k in [1,n], [a,b) is 0-indexed, -1 if error
47     int quantile(int k, int a, int b)
48     {
49         //extra conditions disabled
50         if (/*a < 0 or b > n or*/ k < 1 or k > b - a)
51             return -1;
52         int l = 0, r = s - 1, u = 1, m, za, zb;
53         while (l != r)
54         {
55             m = (l + r) / 2;
56             za = r0[u][a];
57             zb = r0[u][b];
58             u *= 2;
59             if (k <= zb - za)
60                 a = za, b = zb, r = m;
61             else
62                 k -= zb - za, a -= za, b -= zb,
63                 l = m + 1, ++u;
64         }
65         return r;
66     }
67
68     //counts numbers in [x,y] in positions [a,b)
69     int range(int x, int y, int a, int b)
70     {
71         if (y < x or b <= a)
72             return 0;

```

```

73     q = x;
74     w = y;
75     return range(a, b, 0, s - 1, 1);
76 }
77
78 //count occurrences of x in positions [0,k)
79 int rank(int x, int k)
80 {
81     int l = 0, r = s - 1, u = 1, m, z;
82     while (l != r)
83     {
84         m = (l + r) / 2;
85         z = r0[u][k];
86         u *= 2;
87         if (x <= m)
88             k = z, r = m;
89         else
90             k -= z, l = m + 1, ++u;
91     }
92     return k;
93 }
94
95 //x in [0,sigma)
96 void push_back(int x)
97 {
98     int l = 0, r = s - 1, u = 1, m, p;
99     ++n;
100    while (l != r)
101    {
102        m = (l + r) / 2;
103        p = (x <= m);
104        r0[u].push_back(r0[u].back() + p);
105        u *= 2;
106        if (p)
107            r = m;
108        else
109            l = m + 1, ++u;
110    }
111 }
112
113 //doesn't check if empty
114 void pop_back()
115 {
116     int l = 0, r = s - 1, u = 1, m, p, k;
117     --n;
118     while (l != r)
119     {
120         m = (l + r) / 2;
121         k = r0[u].size();
122         p = r0[u][k - 1] - r0[u][k - 2];
123         r0[u].pop_back();
124         u *= 2;
125         if (p)
126             r = m;
127         else

```

```

128         l = m + 1, ++u;
129     }
130 }
131
132 //swap arr[i] with arr[i+1], i in [0,n-1)
133 void swap_adj(int i)
134 {
135     int &x = arrCopy[i], &y = arrCopy[i + 1];
136     int l = 0, r = s - 1, u = 1;
137     while (l != r)
138     {
139         int m = (l + r) / 2, p = (x <= m), q = (y <= m);
140         if (p != q)
141         {
142             r0[u][i + 1] ^= r0[u][i] ^ r0[u][i + 2];
143             break;
144         }
145         u *= 2;
146         if (p)
147             r = m;
148         else
149             l = m + 1, ++u;
150     }
151     swap(x, y);
152 }
153 };

```