

1 Data Structures

1.1 Segment Tree

1.1.1 Lazy

```

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

```

```

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

```

```

105     lazy.assign(n * 4 + 5, 0);
106     build(0, 0, n - 1);
107 }
108
109 ll query(int a, int b)
110 {
111     return query(a, b, 0, 0, n - 1);
112 }
113
114 void update(int a, int b, ll value)
115 {
116     update(a, b, value, 0, 0, n - 1);
117 }
118 };

```

1.1.2 Iterative

```

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

```

```

37         sgn = (x > 0) - (x < 0);
38     }
39     prodsgn(const prodsgn &a,
40             const prodsgn &b)
41     {
42         sgn = a.sgn * b.sgn;
43     }
44 };
45
46 // Maximum Sum (SPOJ)
47 struct maxsum
48 {
49     int first, second;
50     maxsum() { first = second = -1; }
51     maxsum(int x)
52     {
53         first = x;
54         second = -1;
55     }
56     maxsum(const maxsum &a,
57           const maxsum &b)
58     {
59         if (a.first > b.first)
60         {
61             first = a.first;
62             second = max(a.second,
63                         b.first);
64         }
65         else
66         {
67             first = b.first;
68             second = max(a.first,
69                         b.second);
70         }
71     }
72     int answer()
73     {
74         return first + second;
75     }
76 };
77
78 // Range Minimum Query
79 struct rminq
80 {
81     int value;
82     rminq() { value = INT_MAX; }
83     rminq(int x) { value = x; }
84     rminq(const rminq &a,
85           const rminq &b)
86     {
87         value = min(a.value,
88                     b.value);
89     }
90 };
91

```

```
92 template <class node>
93 class ST
94 {
95     vector<node> t;
96     int n;
97
98 public:
99     ST(vector<node> &arr)
100     {
101         n = arr.size();
102         t.resize(n * 2);
103         copy(arr.begin(), arr.end(), t.begin() + n);
104         for (int i = n - 1; i > 0; --i)
105             t[i] = node(t[i << 1], t[i << 1 | 1]);
106     }
107
108     // 0-indexed
109     void set_point(int p, const node &value)
110     {
111         for (t[p += n] = value; p > 1; p >>= 1)
112             t[p >> 1] = node(t[p], t[p ^ 1]);
113     }
114
115     // inclusive exclusive, 0-indexed
116     node query(int l, int r)
117     {
118         node ans1, ansr;
119         for (l += n, r += n; l < r; l >>= 1, r >>= 1)
120         {
121             if (l & 1)
122                 ans1 = node(ans1, t[l++]);
123             if (r & 1)
124                 ansr = node(t[--r], ansr);
125         }
126         return node(ans1, ansr);
127     }
128 };
```