{

    "": {

        "prefix": "graph\_segment\_tree\_template",

        "body": [

          "const int MX = 1e9;",

          " ",

          "// === Segment Tree ===",

          "struct Node;",

          "struct Update;",

          " ",

          "template <typename T, typename Node = Node, typename Update = Update>",

          "class SegmentTree {",

          "private: ",

          " int size = 0;",

          " vector<Node> seg;",

          " ",

          "    void build (int start, int end, int ind, vector<T> &arr) {",

          "     if(start == end) {",

          "         seg[ind] = Node(arr[start]);",

          "         return;",

          "     }",

          "     int mid = (start + end) / 2;",

          "     int leftInd = 2 \* ind + 1, rightInd = 2 \* ind + 2;",

          "     build(start, mid, leftInd, arr);",

          "     build(mid + 1, end, rightInd, arr);",

          "     seg[ind].merge(seg[leftInd], seg[rightInd]);",

          "    }",

          " ",

          " Node query (int start, int end, int ind, int left, int right) {",

          "     if(start > right || end < left) {",

          "         return Node();",

          "     }",

          "     if(start >= left && end <= right) {",

          "         return seg[ind];",

          "     }",

          "     int mid = (start + end) / 2;",

          "     int leftInd = 2 \* ind + 1, rightInd = 2 \* ind + 2;",

          "     Node res;",

          "     Node leftItem = query(start, mid, leftInd, left, right);",

          "     Node rightItem = query(mid + 1, end, rightInd, left, right);",

          "     res.merge(leftItem, rightItem);",

          "     return res;",

          " }",

          " ",

          " void update (int start, int end, int ind, int index, Update &u) {",

          "     if(start == end) {",

          "         u.apply(seg[ind]);",

          "         return;",

          "     }",

          "     int mid = (start + end) / 2;",

          "     int leftInd = 2 \* ind + 1, rightInd = 2 \* ind + 2;",

          "     if(index <= mid) update(start, mid, leftInd, index, u);",

          "     else update(mid + 1, end, rightInd, index, u);",

          "     seg[ind].merge(seg[leftInd], seg[rightInd]);",

          " }",

          " ",

          "public:",

          " SegmentTree () {",

          "     ",

          " }",

          "    ",

          "    SegmentTree (int n) {",

          "     size = n;",

          "     seg.resize(4 \* size + 1);",

          "    }",

          "    ",

          "    SegmentTree (vector<T> &arr) {",

          "     size = arr.size();",

          "     seg.resize(4 \* size + 1);",

          "     build(arr);",

          "    }",

          "    ",

          "    void build (vector<T> &arr) {",

          "     build(0, size - 1, 0, arr);",

          "    }",

          "    ",

          "    Node query (int left, int right) {",

          "     return query(0, size - 1, 0, left, right);",

          "    }",

          "    ",

          "    void update (int index, int value) {",

          "     Update u = Update(value);",

          "     return update(0, size - 1, 0, index, u);",

          "    }",

          "};",

          " ",

          "struct Node {",

          "    long long sum;",

          "    int mx, mn;",

          "    long long andd;",

          "    ",

          "    Node () : sum(0), mx(-MX), mn(MX), andd((1LL << 32) - 1) {}",

          "    Node (int val) : sum(val), mx(val), mn(val), andd(val) {}",

          "    ",

          "    void merge (Node &left, Node &right) {",

          "     sum = left.sum + right.sum;",

          "     mx = max(left.mx, right.mx);",

          "     mn = min(left.mn, right.mn);",

          "     andd = left.andd & right.andd;",

          "    }",

          "};",

          " ",

          "struct Update {",

          " int val;",

          " ",

          " Update () : val(0) {}",

          " Update (int v) : val(v) {}",

          " ",

          " void apply (Node &node) {",

          "     node.sum = val;",

          "     node.mn = val;",

          "     node.mx = val;",

          "     node.andd = val;",

          " }",

          "};"

        ],

        "description": ""

      }

}