伸展树

Splay

基于值域

Idea：平衡二叉树，通过旋转操作保持平衡。

ATT：每次操作都保证至少 `splay` 一次，防止被特定数据卡掉；建树时先插入 INF 和 -INF 简化代码。

Complexity：单次操作 $O(\lg n)$

Code：

基础操作：

```
struct Splay{
    int fa, son[2], size, cnt, val;
}tr[N];
#define which(x,fa) (tr[fa].son[1] == x)
int tot = 0, root = 0;
queue<int> rec; // recycle
inline void pushup(int x){
    if(x){
        tr[x].size = tr[x].cnt;
        if(tr[x].son[0])    tr[x].size += tr[tr[x].son[0]].size;
        if(tr[x].son[1])    tr[x].size += tr[tr[x].son[1]].size;
    }
}
inline void rotate(int x, int dir){
    // dir == 0: left, dir == 1: right
    int y = tr[x].fa, z = tr[y].fa, B = tr[x].son[dir];
    tr[z].son[which(y,z)] = x; tr[x].fa = z;
    tr[x].son[dir] = y; tr[y].fa = x;
    tr[y].son[dir^1] = B; tr[B].fa = y;
    pushup(y); pushup(x);
}
inline void splay(int x, int goal){
    // rotate x to the son of goal
    if(x == goal)    return;
    while(tr[x].fa != goal){
        int y = tr[x].fa, z = tr[y].fa, dir1 = which(x,y)^1, dir2 = which(y,z)^1;
        // pushdown(z), pushdown(y), pushdown(x);
        if(z == goal)    rotate(x, dir1);
        else{
            if(dir1 == dir2)    rotate(y, dir2);
            else    rotate(x, dir1);
            rotate(x, dir2);
        }
    }
    if(goal == 0)    root = x;
}
```

获取值为 val 的节点编号：

```
 1   inline int select(int val){
 2       // return tot of node whose val == val
 3       int now = root;
 4       while(now){
 5           if(tr[now].val == val)  return now;
 6           else if(tr[now].val > val)  now = tr[now].son[0];
 7           else if(tr[now].val < val)  now = tr[now].son[1];
 8       }
 9       if(!now)   return -1;
10       return now;
11   }
```

获取前驱/后继的值/节点编号（前驱：最大的严格小于 val 的值；后继：最小的严格大于 val 的值）：

```
 1   inline int getPre(int val){
 2       // find the predecessor of val x (the greatest value less than x)
 3       int now = root, res = -INF;
 4       while(now){
 5           if(tr[now].val < val){
 6               res = max(res, tr[now].val);
 7               now = tr[now].son[1];
 8           }
 9           else    now = tr[now].son[0];
10       }
11       return res;
12   }
13   inline int getSuc(int val){
14       // find the successor of val x (the least value greater than x)
15       int now = root, res = INF;
16       while(now){
17           if(tr[now].val > val){
18               res = min(res, tr[now].val);
19               now = tr[now].son[0];
20           }
21           else    now = tr[now].son[1];
22       }
23       return res;
24   }
25   inline int getPreNode(int val){ // return nodeID
26       int now = root, res = 0, preval = -INF;
27       while(now){
28           pushdown(now);
29           if(tr[now].val < val){
30               preval = max(preval, tr[now].val);
31               res = now;
32               now = tr[now].son[1];
33           }
34           else    now = tr[now].son[0];
35       }
36       return res;
37   }
38   inline int getSucNode(int val){ // return nodeID
39       int now = root, res = 0, sucval = INF;
40       while(now){
41           pushdown(now);
42           if(tr[now].val > val){
43               sucval = min(sucval, tr[now].val);
44               res = now;
45               now = tr[now].son[0];
46           }
47           else    now = tr[now].son[1];
48       }
49       return res;
50   }
```

获取 val 的排名（即小于 val 的数的个数加一，val 可以不存在于平衡树中）：

```cpp
inline int getRank(int val){
    // get the rank of val
    // i.e. the number of those < val plus 1
    int now = root, rank = 0, t = 0;
    while(now){
        t = now;
        if(tr[now].val == val){
            rank += tr[tr[now].son[0]].size;
            break;
        }
        else if(tr[now].val < val){
            rank += tr[now].size - tr[tr[now].son[1]].size;
            now = tr[now].son[1];
        }
        else    now = tr[now].son[0];
    }
    splay(t, 0);
    return rank;
}
```

新建节点：

```cpp
inline int newNode(int val, int fa){
    int id;
    if(!rec.empty())    id = rec.front(), rec.pop();
    else    id = ++tot;
    tr[id].fa = fa;
    tr[id].son[0] = tr[id].son[1] = 0;
    tr[id].size = tr[id].cnt = 1;
    tr[id].val = val;
    return id;
}
```

插入/删除值：

```cpp
inline void insert(int val){
    // insert val into splay tree
    splay(select(getPre(val)), 0);
    splay(select(getSuc(val)), root);
    int &x = tr[tr[root].son[1]].son[0];
    if(x){ tr[x].cnt++; tr[x].size++; }
    else    x = newNode(val, tr[root].son[1]);
    pushup(tr[root].son[1]);
    pushup(root);
}
inline void del(int val){
    // delete one val from splay tree
    splay(select(getPre(val)), 0);
    splay(select(getSuc(val)), root);
    int &x = tr[tr[root].son[1]].son[0];
    if(!x || !tr[x].cnt)    return ;
    tr[x].cnt--; tr[x].size--;
    if(tr[x].cnt == 0)  rec.push(x), x = 0;
    pushup(tr[root].son[1]);
    pushup(root);
}
```

获取平衡树中第 x 个节点的值：

```
1   inline int findRank(int x){
2       // find the val of x'th node
3       int now = root;
4       while(now){
5           if(tr[tr[now].son[0]].size + 1 <= x && x <= tr[now].size - tr[tr[now].son[1]].size)
6               break;
7           else if(tr[tr[now].son[0]].size + 1 > x)
8               now = tr[now].son[0];
9           else if(x > tr[now].size - tr[tr[now].son[1]].size){
10              x -= tr[now].size - tr[tr[now].son[1]].size;
11              now = tr[now].son[1];
12          }
13      }
14      splay(now, 0);
15      return tr[root].val;
16  }
```

`rev` 标记的 `pushdown`：（前述众多操作里，每访问一个节点都要 `pushdown`）

```
1   inline void pushdown(int x){
2       if(tr[x].rev){
3           if(tr[x].son[0]){
4               tr[tr[x].son[0]].rev ^= 1;
5               swap(tr[tr[x].son[0]].son[0], tr[tr[x].son[0]].son[1]);
6           }
7           if(tr[x].son[1]){
8               tr[tr[x].son[1]].rev ^= 1;
9               swap(tr[tr[x].son[1]].son[0], tr[tr[x].son[1]].son[1]);
10          }
11          tr[x].rev ^= 1;
12      }
13  }
```

按中序遍历输出：

```
1   void print(int x){
2       pushdown(x);
3       if(tr[x].son[0])    print(tr[x].son[0]);
4       if(tr[x].val != INF && tr[x].val != -INF)
5           printf("%d ", tr[x].val);
6       if(tr[x].son[1])    print(tr[x].son[1]);
7   }
```

主函数：

```
1   int main(){
2       // ...
3       root = newNode(-INF, 0);
4       tr[root].son[1] = newNode(INF, root);
5       pushup(root);
6       // ...
7   }
```

基于序列

按照原序列顺序建立 Splay.

ATT：建树时先插入 INF 和 -INF 简化代码。

Complexity：单次操作 $O(\lg n)$

Code：

基础操作：

```
1   struct Splay{
2       int fa, son[2], size;
3       LL val, sum, mn;
4       void init(){
5           fa = son[0] = son[1] = size = 0;
6           val = sum = mn = 0;
7       }
8   }tr[N];
9   #define which(x,fa) (tr[fa].son[1] == x)
10  int tot = 0, root = 0;
11  inline void pushup(int x){
12      if(x){
13          tr[x].size = 1, tr[x].sum = tr[x].mn = tr[x].val;
14          if(tr[x].son[0]){
15              tr[x].size += tr[tr[x].son[0]].size;
16              tr[x].sum += tr[tr[x].son[0]].sum;
17              tr[x].mn = min(tr[tr[x].son[0]].mn, tr[x].mn);
18          }
19          if(tr[x].son[1]){
20              tr[x].size += tr[tr[x].son[1]].size;
21              tr[x].sum += tr[tr[x].son[1]].sum;
22              tr[x].mn = min(tr[tr[x].son[1]].mn, tr[x].mn);
23          }
24      }
25  }
26  inline void rotate(int x, int dir){
27      // dir == 0: left, dir == 1: right
28      int y = tr[x].fa, z = tr[y].fa, B = tr[x].son[dir];
29      tr[z].son[which(y,z)] = x; tr[x].fa = z;
30      tr[x].son[dir] = y; tr[y].fa = x;
31      tr[y].son[dir^1] = B; tr[B].fa = y;
32      pushup(y); pushup(x);
33  }
34  inline void splay(int x, int goal){
35      // rotate x to the son of goal
36      if(x == goal)    return;
37      while(tr[x].fa != goal){
38          int y = tr[x].fa, z = tr[y].fa, dir1 = which(x,y)^1, dir2 = which(y,z)^1;
39          if(z == goal)   rotate(x, dir1);
40          else{
41              if(dir1 == dir2)    rotate(y, dir2);
42              else    rotate(x, dir1);
43              rotate(x, dir2);
44          }
45      }
46      if(goal == 0)   root = x;
47  }
```

获取平衡树中第 x 个节点的编号：

```cpp
inline int selectNode(int x){
    // return id of x'th node on the tree
    int now = root;
    while(tr[tr[now].son[0]].size + 1 != x){
        if(tr[tr[now].son[0]].size + 1 > x)
            now = tr[now].son[0];
        else{
            x -= tr[tr[now].son[0]].size + 1;
            now = tr[now].son[1];
        }
    }
    return now;
}
```

插入/删除平衡树中第 x 个节点:

```cpp
inline int del(int x){
    // delete the x'th node on the tree
    splay(selectNode(x-1), 0);
    splay(selectNode(x+1), root);
    int now = tr[tr[root].son[1]].son[0];
    tr[tr[root].son[1]].son[0] = 0;
    tr[now].fa = tr[now].size = 0;
    tr[now].son[0] = tr[now].son[1] = 0;
    tr[now].val = tr[now].sum = tr[now].mn = 0;
    pushup(tr[root].son[1]), pushup(root);
    return now;
}
inline void insert(int x, LL val, int id){
    // insert val as the x'th node on the tree, using id as its id
    splay(selectNode(x-1), 0);
    splay(selectNode(x), root);
    tr[tr[root].son[1]].son[0] = id;
    tr[id].fa = tr[root].son[1];
    tr[id].son[0] = tr[id].son[1] = 0;
    tr[id].size = 1;
    tr[id].val = tr[id].sum = tr[id].mn = val;
    pushup(tr[root].son[1]), pushup(root);
}
```

从第 l 个节点到第 r 个节点求和:

```cpp
inline LL getSum(int l, int r){
    // return the sum of nodes from l'th to r'th node on the tree
    splay(selectNode(l-1), 0);
    splay(selectNode(r+1), root);
    int now = tr[tr[root].son[1]].son[0];
    return tr[now].sum;
}
```

根据初始序列建树:

```
1    int build(int l, int r, int fa){
2        if(l > r)    return 0;
3        int id = ++tot;
4        tr[id].fa = fa, tr[id].size = 1;
5        int mid = (l + r) >> 1;
6        tr[id].val = tr[id].sum = tr[id].mn = b[mid];
7        tr[id].son[0] = build(l, mid - 1, id);
8        tr[id].son[1] = build(mid + 1, r, id);
9        pushup(id);
10       return id;
11   }
```

按中序遍历输出：

```
1    void print(int x){
2        if(tr[x].son[0])    print(tr[x].son[0]);
3        if(tr[x].val != -INF && tr[x].val != INF)
4            printf("%lld ", tr[x].val);
5        if(tr[x].son[1])    print(tr[x].son[1]);
6    }
```

主函数：

```
1    int main(){
2        // ...
3        b[0] = -INF, b[n+1] = INF;
4        for(int i = 1; i <= n; i++) scanf("%lld", &b[i]);
5        root = build(0, n + 1, 0);
6        // ...
7    }
```