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：

```cpp
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：

```
inline int select(int val){
    // return tot of node whose val == val
    int now = root;
    while(now){
        if(tr[now].val == val)  return now;
        else if(tr[now].val > val)  now = tr[now].son[0];
        else if(tr[now].val < val)  now = tr[now].son[1];
    }
    if(!now)    return -1;
    return now;
}
```

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）：

```
inline int getPre(int val){
    // find the predecessor of val x (the greatest value less than x)
    int now = root, res = -INF;
    while(now){
        if(tr[now].val < val){
            res = max(res, tr[now].val);
            now = tr[now].son[1];
        }
        else    now = tr[now].son[0];
    }
    return res;
}
inline int getSuc(int val){
    // find the successor of val x (the least value greater than x)
    int now = root, res = INF;
    while(now){
        if(tr[now].val > val){
            res = min(res, tr[now].val);
            now = tr[now].son[0];
        }
        else    now = tr[now].son[1];
    }
    return res;
}
inline int getPreNode(int val){ // return nodeID
    int now = root, res = 0, preval = -INF;
    while(now){
        pushdown(now);
        if(tr[now].val < val){
            preval = max(preval, tr[now].val);
            res = now;
            now = tr[now].son[1];
        }
        else    now = tr[now].son[0];
    }
    return res;
}
inline int getSucNode(int val){ // return nodeID
    int now = root, res = 0, sucval = INF;
    while(now){
        pushdown(now);
        if(tr[now].val > val){
            sucval = min(sucval, tr[now].val);
            res = now;
            now = tr[now].son[0];
        }
        else    now = tr[now].son[1];
    }
    return res;
}
```

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）：

```
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：

```
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：

```
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：

```
inline int findRank(int x){
    // find the val of x'th node
    int now = root;
    while(now){
        if(tr[tr[now].son[0]].size + 1 <= x && x <= tr[now].size - tr[tr[now].son[1]].size)
            break;
        else if(tr[tr[now].son[0]].size + 1 > x)
            now = tr[now].son[0];
        else if(x > tr[now].size - tr[tr[now].son[1]].size){
            x -= tr[now].size - tr[tr[now].son[1]].size;
            now = tr[now].son[1];
        }
    }
    splay(now, 0);
    return tr[root].val;
}
```

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

```
inline void pushdown(int x){
    if(tr[x].rev){
        if(tr[x].son[0]){
            tr[tr[x].son[0]].rev ^= 1;
            swap(tr[tr[x].son[0]].son[0], tr[tr[x].son[0]].son[1]);
        }
        if(tr[x].son[1]){
            tr[tr[x].son[1]].rev ^= 1;
            swap(tr[tr[x].son[1]].son[0], tr[tr[x].son[1]].son[1]);
        }
        tr[x].rev ^= 1;
    }
}
```

按中序遍历输出：

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

主函数：

```cpp
int main(){
    // ...
    root = newNode(-INF, 0);
    tr[root].son[1] = newNode(INF, root);
    pushup(root);
    // ...
}
```

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：

```cpp
struct Splay{
    int fa, son[2], size;
    LL val, sum, mn;
    void init(){
        fa = son[0] = son[1] = size = 0;
        val = sum = mn = 0;
    }
}tr[N];
#define which(x,fa) (tr[fa].son[1] == x)
int tot = 0, root = 0;
inline void pushup(int x){
    if(x){
        tr[x].size = 1, tr[x].sum = tr[x].mn = tr[x].val;
        if(tr[x].son[0]){
            tr[x].size += tr[tr[x].son[0]].size;
            tr[x].sum += tr[tr[x].son[0]].sum;
            tr[x].mn = min(tr[tr[x].son[0]].mn, tr[x].mn);
        }
        if(tr[x].son[1]){
            tr[x].size += tr[tr[x].son[1]].size;
            tr[x].sum += tr[tr[x].son[1]].sum;
            tr[x].mn = min(tr[tr[x].son[1]].mn, tr[x].mn);
        }
    }
}
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;
        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;
}
```

获取平衡树中第 x 个节点的编号：

```
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：

```
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：

```cpp
int build(int l, int r, int fa){
    if(l > r)   return 0;
    int id = ++tot;
    tr[id].fa = fa, tr[id].size = 1;
    int mid = (l + r) >> 1;
    tr[id].val = tr[id].sum = tr[id].mn = b[mid];
    tr[id].son[0] = build(l, mid - 1, id);
    tr[id].son[1] = build(mid + 1, r, id);
    pushup(id);
    return id;
}
```

按中序遍历输出：

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

主函数：

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