

# 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：

```
1 struct Splay{
2     int fa, son[2], size, cnt, val;
3 }tr[N];
4 #define which(x,fa) (tr[fa].son[1] == x)
5 int tot = 0, root = 0;
6 queue<int> rec; // recycle
7 inline void pushup(int x){
8     if(x){
9         tr[x].size = tr[x].cnt;
10        if(tr[x].son[0]) tr[x].size += tr[tr[x].son[0]].size;
11        if(tr[x].son[1]) tr[x].size += tr[tr[x].son[1]].size;
12    }
13 }
14 inline void rotate(int x, int dir){
15     // dir == 0: left, dir == 1: right
16     int y = tr[x].fa, z = tr[y].fa, B = tr[x].son[dir];
17     tr[z].son[which(y,z)] = x; tr[x].fa = z;
18     tr[x].son[dir] = y; tr[y].fa = x;
19     tr[y].son[dir^1] = B; tr[B].fa = y;
20     pushup(y); pushup(x);
21 }
22 inline void splay(int x, int goal){
23     // rotate x to the son of goal
24     if(x == goal) return;
```

```

25     while(tr[x].fa != goal){
26         int y = tr[x].fa, z = tr[y].fa, dir1 = which(x,y)^1, dir2 =
which(y,z)^1;
27         // pushdown(z), pushdown(y), pushdown(x);
28         if(z == goal) rotate(x, dir1);
29         else{
30             if(dir1 == dir2) rotate(y, dir2);
31             else rotate(x, dir1);
32             rotate(x, dir2);
33         }
34     }
35     if(goal == 0) root = x;
36 }

```

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）：

```

1  inline int getRank(int val){
2      // get the rank of val
3      // i.e. the number of those < val plus 1
4      int now = root, rank = 0, t = 0;
5      while(now){
6          t = now;
7          if(tr[now].val == val){
8              rank += tr[t].son[0].size;

```

```

9         break;
10    }
11    else if(tr[now].val < val){
12        rank += tr[now].size - tr[tr[now].son[1]].size;
13        now = tr[now].son[1];
14    }
15    else    now = tr[now].son[0];
16 }
17 splay(t, 0);
18 return rank;
19 }

```

新建节点：

```

1 inline int newNode(int val, int fa){
2     int id;
3     if(!rec.empty())    id = rec.front(), rec.pop();
4     else    id = ++tot;
5     tr[id].fa = fa;
6     tr[id].son[0] = tr[id].son[1] = 0;
7     tr[id].size = tr[id].cnt = 1;
8     tr[id].val = val;
9     return id;
10 }

```

插入/删除值：

```

1 inline void insert(int val){
2     // insert val into splay tree
3     splay(select(getPre(val)), 0);
4     splay(select(getSuc(val)), root);
5     int &x = tr[tr[root].son[1]].son[0];
6     if(x){ tr[x].cnt++; tr[x].size++; }
7     else    x = newNode(val, tr[root].son[1]);
8     pushup(tr[root].son[1]);
9     pushup(root);
10 }
11 inline void del(int val){
12     // delete one val from splay tree
13     splay(select(getPre(val)), 0);
14     splay(select(getSuc(val)), root);
15     int &x = tr[tr[root].son[1]].son[0];
16     if(!x || !tr[x].cnt)    return ;
17     tr[x].cnt--; tr[x].size--;
18     if(tr[x].cnt == 0)    rec.push(x), x = 0;
19     pushup(tr[root].son[1]);

```

```

20     pushup(root);
21 }

```

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：

```

1  inline int selectNode(int x){
2      // return id of x'th node on the tree
3      int now = root;
4      while(tr[tr[now].son[0]].size + 1 != x){
5          if(tr[tr[now].son[0]].size + 1 > x)
6              now = tr[now].son[0];
7          else{
8              x -= tr[tr[now].son[0]].size + 1;
9              now = tr[now].son[1];
10         }
11     }
12     return now;
13 }

```

插入/删除平衡树中第  $x$  个节点:

```

1  inline int del(int x){
2      // delete the x'th node on the tree
3      splay(selectNode(x-1), 0);
4      splay(selectNode(x+1), root);
5      int now = tr[tr[root].son[1]].son[0];
6      tr[tr[root].son[1]].son[0] = 0;
7      tr[now].fa = tr[now].size = 0;
8      tr[now].son[0] = tr[now].son[1] = 0;
9      tr[now].val = tr[now].sum = tr[now].mn = 0;
10     pushup(tr[root].son[1]), pushup(root);
11     return now;
12 }
13 inline void insert(int x, LL val, int id){
14     // insert val as the x'th node on the tree, using id as its id
15     splay(selectNode(x-1), 0);
16     splay(selectNode(x), root);
17     tr[tr[root].son[1]].son[0] = id;
18     tr[id].fa = tr[root].son[1];
19     tr[id].son[0] = tr[id].son[1] = 0;
20     tr[id].size = 1;
21     tr[id].val = tr[id].sum = tr[id].mn = val;
22     pushup(tr[root].son[1]), pushup(root);
23 }

```

从第  $l$  个节点到第  $r$  个节点求和:



```

1 inline LL getSum(int l, int r){
2     // return the sum of nodes from l'th to r'th node on the tree
3     splay(selectNode(l-1), 0);
4     splay(selectNode(r+1), root);
5     int now = tr[tr[root].son[1]].son[0];
6     return tr[now].sum;
7 }

```

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 }

```

