

# 可持久化平衡树

## Persistent Treap

**Idea:** 将无旋 Treap 可持久化，主要是修改 `split` 和 `merge` 的代码，把原来覆盖的代码改为新建新节点。

**ATT:** 在进行需要改变平衡树的操作时，应传引用（代码中的 `&cur`），如此才能改变新建的节点。

**Code:**

```
1 struct Treap{
2     int son[2], size, val, hp;
3 }tr[N*50];
4 int cnt, root[N];
5 inline int newNode(int val = 0);
6 inline void pushup(int id);
7 inline void pushdown(int id);
8 int merge(int a, int b){
9     if(!a || !b) return a + b;
10    if(tr[a].hp <= tr[b].hp){
11        pushdown(a);
12        int cur = newNode();
13        tr[cur] = tr[a];
14        tr[cur].son[1] = merge(tr[cur].son[1], b);
15        pushup(cur);
16        return cur;
17    }
18    else{
19        pushdown(b);
20        int cur = newNode();
21        tr[cur] = tr[b];
22        tr[cur].son[0] = merge(a, tr[cur].son[0]);
23        pushup(cur);
24        return cur;
25    }
26 }
27 void split(int id, int k, int &x, int &y){
28     // split treap into 2 parts according to values: <= k and > k, and store them in x and y
29     if(!id){
30         x = 0; y = 0;
31         return;
32     }
33     pushdown(id);
34     if(k < tr[id].val){
35         tr[y = newNode()] = tr[id];
36         split(tr[y].son[0], k, x, tr[y].son[0]);
37         pushup(y);
38     }
39     else{
40         tr[x = newNode()] = tr[id];
41         split(tr[x].son[1], k, tr[x].son[1], y);
42         pushup(x);
43     }
44 }
45 inline void insert(int &cur, int val){ // insert val into treap
46     int l = 0, r = 0;
47     split(cur, val, l, r);
48     int t = newNode(val);
49     cur = merge(merge(l, t), r);
50 }
51 inline void del(int &cur, int val){ // delete one val from treap
```

```

52     int l = 0, t = 0, r = 0;
53     split(cur, val-1, l, t);
54     split(t, val, t, r);
55     t = merge(tr[t].son[0], tr[t].son[1]);
56     cur = merge(merge(l, t), r);
57 }
58 inline int getRank(int cur, int val){ // get the rank of val x
59     int l = 0, r = 0;
60     split(cur, val-1, l, r);
61     int res = tr[l].size + 1;
62     merge(l, r);
63     return res;
64 }
65 inline int findRank(int cur, int x){ // find the val whose rank is x
66     int now = cur;
67     while(now){
68         if(tr[tr[now].son[0]].size + 1 == x) return tr[now].val;
69         else if(tr[tr[now].son[0]].size >= x) now = tr[now].son[0];
70         else{
71             x -= tr[tr[now].son[0]].size + 1;
72             now = tr[now].son[1];
73         }
74     }
75     return -INF;
76 }
77 inline int getPre(int cur, int val){
78     // find the predecessor of val x (the greatest value less than x)
79     int now = cur, res = -INF;
80     while(now){
81         if(tr[now].val < val){
82             res = max(res, tr[now].val);
83             now = tr[now].son[1];
84         }
85         else now = tr[now].son[0];
86     }
87     return res;
88 }
89 inline int getSuc(int cur, int val){
90     // find the successor of val x (the least value greater than x)
91     int now = cur, res = INF;
92     while(now){
93         if(tr[now].val > val){
94             res = min(res, tr[now].val);
95             now = tr[now].son[0];
96         }
97         else now = tr[now].son[1];
98     }
99     return res;
100 }
101 int main(){
102     srand(20010130);
103     scanf("%d", &n);
104     for(int i = 1; i <= n; i++){
105         scanf("%d%d%d", &qv, &opt, &qx);
106         root[i] = root[qv];
107         switch(opt){
108             //...;
109         }
110     }
111     return 0;
112 }

```

**Code (其他操作) :**

带 `rev` 标记的 `pushdown` :

注意: `pushdown` 也需要新建节点!

```
1  inline void pushdown(int id){
2      if(tr[id].rev){
3          if(tr[id].son[0]){
4              Treap tmp = tr[tr[id].son[0]];
5              tr[id].son[0] = newNode();
6              tr[tr[id].son[0]] = tmp;
7              tr[tr[id].son[0]].rev ^= 1;
8              swap(tr[tr[id].son[0]].son[0], tr[tr[id].son[0]].son[1]);
9          }
10         if(tr[id].son[1]){
11             Treap tmp = tr[tr[id].son[1]];
12             tr[id].son[1] = newNode();
13             tr[tr[id].son[1]] = tmp;
14             tr[tr[id].son[1]].rev ^= 1;
15             swap(tr[tr[id].son[1]].son[0], tr[tr[id].son[1]].son[1]);
16         }
17         tr[id].rev ^= 1;
18     }
19 }
```

按大小分裂的 `split`:

```
1  inline void splitSize(int id, int k, int &x, int &y){
2      if(!id){
3          x = y = 0;
4          return;
5      }
6      pushdown(id);
7      if(tr[tr[id].son[0]].size >= k){
8          tr[y = newNode()] = tr[id];
9          splitSize(tr[y].son[0], k, x, tr[y].son[0]);
10         pushup(y);
11     }
12     else{
13         tr[x = newNode()] = tr[id];
14         splitSize(tr[x].son[1], k - tr[tr[id].son[0]].size - 1, tr[x].son[1], y);
15         pushup(x);
16     }
17 }
```