无旋 Treap

Idea：treap = tree + heap，既满足二叉搜索树的性质，又满足堆的性质。

Complexity：单次操作 $O(\lg n)$

Code（基础操作）：

```cpp
struct Treap{
    int son[2], size, val, hp;
}tr[N];
int cnt, root;
inline int newNode(int val = 0){
    cnt++;
    tr[cnt].son[0] = tr[cnt].son[1] = 0;
    tr[cnt].size = 1;
    tr[cnt].val = val;
    tr[cnt].hp = rand();
    return cnt;
}
inline void pushup(int id){
    if(!id) return;
    tr[id].size = 1;
    if(tr[id].son[0])   tr[id].size += tr[tr[id].son[0]].size;
    if(tr[id].son[1])   tr[id].size += tr[tr[id].son[1]].size;
}
inline void pushdown(int id){
    return;
}
int merge(int a, int b){
    if(a == 0)  return b;
    if(b == 0)  return a;
    if(tr[a].hp <= tr[b].hp){
        pushdown(a);
        tr[a].son[1] = merge(tr[a].son[1], b);
        pushup(a);
        return a;
    }
    else{
        pushdown(b);
        tr[b].son[0] = merge(a, tr[b].son[0]);
        pushup(b);
        return b;
    }
}
void split(int id, int k, int &x, int &y){ // split treap into 2 parts according to values: <= k and > k, and
store them in x and y
    if(!id){
        x = 0; y = 0;
        return;
    }
    pushdown(id);
    if(k < tr[id].val){
        y = id;
        split(tr[id].son[0], k, x, tr[id].son[0]);
    }
    else{
        x = id;
        split(tr[id].son[1], k, tr[id].son[1], y);
    }
    pushup(id);
}
inline void insert(int val){ // insert val into treap
    int l = 0, r = 0;
    split(root, val, l, r);
    int t = newNode(val);
    root = merge(merge(l, t), r);
}
inline void del(int val){ // delete one val from treap
    int l = 0, t = 0, r = 0;
    split(root, val-1, l, t);
    split(t, val, t, r);
    t = merge(tr[t].son[0], tr[t].son[1]);
```

```
65        root = merge(merge(l, t), r);
66  }
67  inline int getRank(int val){ // get the rank of val x
68      int l = 0, r = 0;
69      split(root, val-1, l, r);
70      int res = tr[l].size + 1;
71      merge(l, r);
72      return res;
73  }
74  inline int findRank(int x){ // find the val whose rank is x
75      int now = root;
76      while(now){
77          if(tr[tr[now].son[0]].size + 1 == x)    return tr[now].val;
78          else if(tr[tr[now].son[0]].size >= x)   now = tr[now].son[0];
79          else{
80              x -= tr[tr[now].son[0]].size + 1;
81              now = tr[now].son[1];
82          }
83      }
84      return -INF;
85  }
86  inline int getPre(int val){ // find the predecessor of val x (the greatest value less than x)
87      int now = root, res = -INF;
88      while(now){
89          if(tr[now].val < val){
90              res = max(res, tr[now].val);
91              now = tr[now].son[1];
92          }
93          else    now = tr[now].son[0];
94      }
95      return res;
96  }
97  inline int getSuc(int val){ // find the successor of val x (the least value greater than x)
98      int now = root, res = INF;
99      while(now){
100         if(tr[now].val > val){
101             res = min(res, tr[now].val);
102             now = tr[now].son[0];
103         }
104         else    now = tr[now].son[1];
105     }
106     return res;
107 }
```

Code（其他操作）：

带 rev 标记的 pushdown：

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

按大小分裂的 split（与之前按值分裂进行区分）：

```
1  void splitSize(int id, int k, int &x, int &y){ // split treap into 2 parts according to ranking: <= k and > k,
   and store them in x and y
2      if(!id){
3          x = 0; y = 0;
4          return;
5      }
6      pushdown(id);
7      if(k <= tr[tr[id].son[0]].size){
8          y = id;
9          splitSize(tr[id].son[0], k, x, tr[id].son[0]);
10     }
```

```
11        else{
12            x = id;
13            splitSize(tr[id].son[1], k - tr[tr[id].son[0]].size - 1, tr[id].son[1], y);
14        }
15        pushup(id);
16    }
```

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        printf("%d ", tr[x].val);
5        if(tr[x].son[1])    print(tr[x].son[1]);
6    }
```