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）：

```
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]);
    root = merge(merge(l, t), r);
}
inline int getRank(int val){ // get the rank of val x
    int l = 0, r = 0;
    split(root, val-1, l, r);
    int res = tr[l].size + 1;
    merge(l, r);
    return res;
}
inline int findRank(int x){ // find the val whose rank is x
    int now = root;
    while(now){
        if(tr[tr[now].son[0]].size + 1 == x)    return tr[now].val;
        else if(tr[tr[now].son[0]].size >= x)   now = tr[now].son[0];
        else{
```

```
            x -= tr[tr[now].son[0]].size + 1;
            now = tr[now].son[1];
        }
    }
    return -INF;
}
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;
}
```

Code（其他操作）：

带 `rev` 标记的 `pushdown`：

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

按**大小**分裂的 `split`（与之前按**值**分裂进行区分）：

```
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
    if(!id){
        x = 0; y = 0;
        return;
    }
    pushdown(id);
    if(k <= tr[tr[id].son[0]].size){
        y = id;
        splitSize(tr[id].son[0], k, x, tr[id].son[0]);
    }
    else{
        x = id;
        splitSize(tr[id].son[1], k - tr[tr[id].son[0]].size - 1, tr[id].son[1], y);
    }
    pushup(id);
}
```

按中序遍历输出：

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