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：

```
struct Treap{
    int son[2], size, val, hp;
}tr[N*50];
int cnt, root[N];
inline int newNode(int val = 0);
inline void pushup(int id);
inline void pushdown(int id);
int merge(int a, int b){
    if(!a ||!b) return a + b;
    if(tr[a].hp <= tr[b].hp){
        pushdown(a);
        int cur = newNode();
        tr[cur] = tr[a];
        tr[cur].son[1] = merge(tr[cur].son[1], b);
        pushup(cur);
        return cur;
    }
    else{
        pushdown(b);
        int cur = newNode();
        tr[cur] = tr[b];
        tr[cur].son[0] = merge(a, tr[cur].son[0]);
        pushup(cur);
        return cur;
    }
}
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){
        tr[y = newNode()] = tr[id];
        split(tr[y].son[0], k, x, tr[y].son[0]);
        pushup(y);
    }
    else{
        tr[x = newNode()] = tr[id];
        split(tr[x].son[1], k, tr[x].son[1], y);
        pushup(x);
    }
}
inline void insert(int &cur, int val){ // insert val into treap
    int l = 0, r = 0;
    split(cur, val, l, r);
    int t = newNode(val);
    cur = merge(merge(l, t), r);
}
inline void del(int &cur, int val){ // delete one val from treap
```

```
    int l = 0, t = 0, r = 0;
    split(cur, val-1, l, t);
    split(t, val, t, r);
    t = merge(tr[t].son[0], tr[t].son[1]);
    cur = merge(merge(l, t), r);
}
inline int getRank(int cur, int val){ // get the rank of val x
    int l = 0, r = 0;
    split(cur, val-1, l, r);
    int res = tr[l].size + 1;
    merge(l, r);
    return res;
}
inline int findRank(int cur, int x){ // find the val whose rank is x
    int now = cur;
    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 cur, int val){
    // find the predecessor of val x (the greatest value less than x)
    int now = cur, 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 cur, int val){
    // find the successor of val x (the least value greater than x)
    int now = cur, 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;
}
int main(){
    srand(20010130);
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        scanf("%d%d%d", &qv, &opt, &qx);
        root[i] = root[qv];
        switch(opt){
            //...;
        }
    }
    return 0;
}
```

Code（其他操作）：

带 `rev` 标记的 `pushdown`：

注意：`pushdown` 也需要**新建节点**！

```
inline void pushdown(int id){
    if(tr[id].rev){
        if(tr[id].son[0]){
            Treap tmp = tr[tr[id].son[0]];
            tr[id].son[0] = newNode();
            tr[tr[id].son[0]] = tmp;
            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]){
            Treap tmp = tr[tr[id].son[1]];
            tr[id].son[1] = newNode();
            tr[tr[id].son[1]] = tmp;
            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`：

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