替罪羊树

算法简介

一种平衡树，思维简单，码量少，速度还行，而且没有旋转操作。

算法思想

首先替罪羊树的插入类似普通的二叉搜索树，删除操作就是打上删除标记。

但只有这样显然不能保证替罪羊树的平衡度。

替罪羊树的核心操作是重构操作，当树的不平衡度大于一个范围时就考虑对一棵子树进行重构。

重构方法为中序遍历子树，将没有打上删除标记的结点加入序列。得到一个有序序列，然后用类似线段树建树的方法重新建树。

重构操作虽然单次复杂度为 $O(n)$ 但可以证明均摊复杂度为 $O(\log n)$ 证明方法自行百度。

问题在于何时考虑重构。

考虑维护每个结点所在子树的未被删除的结点个数 cnt 和结点总数 tot

引入一个平衡因子 α 值，当 $\alpha\ast\text{cnt} \lt \max(\text{cnt}_{lson},\text{cnt}_{rson})$ 时考虑重构。

α 过大将导致树的平衡度较差，查询效率低 α 过小将导致树的重构次数过多，插入、删除效率低。

因此 α 值通常会设置成 $0.7 \sim 0.8$ 可以根据题目要求自行调整。

同时，如果一棵树上被删除的无效结点过多，也会影响查找效率，所以也需要重构。

这里设置为当 $\alpha\ast\text{tot} \gt \text{cnt}$ 时考虑重构。

代码模板

洛谷p3369

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <cctype>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;
typedef long long LL;
```

```
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5;
const double alpha=0.75;
struct Node{
    int ch[2],v,cnt,tot;
    bool exist;
    void build(int v){
        this->v=v;
        ch[0]=ch[1]=0;
        cnt=tot=1;
        exist=true;
    }
}node[MAXN];
int pool[MAXN],pos1,temp[MAXN],pos2,root;
void Init(int n){
    for(int i=n;i>=1;i--)
    pool[++pos1]=i;
}
bool isbad(int pos){return
alpha*node[pos].cnt<max(node[node[pos].ch[0]].cnt,node[node[pos].ch[1]].cnt
)?true:false;}
bool isbad_2(int pos){return alpha*node[pos].tot>node[pos].cnt?true:false;}
void dfs(int pos){
    if(!pos)return;
    dfs(node[pos].ch[0]);
    if(node[pos].exist)temp[++pos2]=pos;
    else pool[++pos1]=pos;
    dfs(node[pos].ch[1]);
}
void build(int lef,int rig,int &pos){
    if(lef>rig) return pos=0,void();
    int mid=lef+rig>>1;
    pos=temp[mid];
    if(lef==rig){
        node[pos].ch[0]=node[pos].ch[1]=0;
        node[pos].cnt=node[pos].tot=1;
```

```
        return;
    }
    build(lef,mid-1,node[pos].ch[0]);
    build(mid+1,rig,node[pos].ch[1]);
node[pos].tot=node[pos].cnt=node[node[pos].ch[0]].cnt+node[node[pos].ch[1]]
.cnt+1;
}
void rebuild(int &pos){
    pos2=0;
    dfs(pos);
    build(1,pos2,pos);
}
void check(int &pos,int x){
    if(pos){
        if(isbad(pos)||isbad_2(pos))
        rebuild(pos);
        else if(node[pos].v<x)
        check(node[pos].ch[1],x);
        else
        check(node[pos].ch[0],x);
    }
}
int rank(int x){
    int pos=root,rk=1;
    while(pos){
        if(node[pos].v<x){
            rk+=node[node[pos].ch[0]].cnt+node[pos].exist;
            pos=node[pos].ch[1];
        }
        else
        pos=node[pos].ch[0];
    }
    return rk;
}
int kth(int rk){
    int pos=root;
    while(pos){
        if(node[node[pos].ch[0]].cnt+1==rk&&node[pos].exist)return
node[pos].v;
        if(node[node[pos].ch[0]].cnt+node[pos].exist<rk){
            rk-=node[node[pos].ch[0]].cnt+node[pos].exist;
            pos=node[pos].ch[1];
        }
        else
        pos=node[pos].ch[0];
    }
}
void Insert(int &pos,int x){
    if(!pos){
        pos=pool[pos1--];
        node[pos].build(x);
```

```
        return;
    }
    node[pos].cnt++;node[pos].tot++;
    if(node[pos].v<x)Insert(node[pos].ch[1],x);
    else Insert(node[pos].ch[0],x);
}
void Insert(int x){
    Insert(root,x);
    check(root,x);
}
void Delate(int pos,int rk){
    node[pos].cnt--;
    if(node[node[pos].ch[0]].cnt+1==rk&&node[pos].exist){
        node[pos].exist=false;
        return;
    }
if(node[node[pos].ch[0]].cnt+node[pos].exist<rk)Delate(node[pos].ch[1],rk-
node[node[pos].ch[0]].cnt-node[pos].exist);
    else Delate(node[pos].ch[0],rk);
}
void Delate(int x){
    Delate(root,rank(x));
    check(root,x);
}
int main()
{
    int n=read_int(),opt,x,line=1;
    Init(MAXN-1);
    while(n--){
        opt=read_int(),x=read_int();
        switch(opt){
            case 1:
                Insert(x);
                break;
            case 2:
                Delate(x);
                break;
            case 3:
                enter(rank(x));
                break;
            case 4:
                enter(kth(x));
                break;
            case 5:
                enter(kth(rank(x)-1));
                break;
            case 6:
                enter(kth(rank(x+1)));
                break;
        }
```

```
    }
    return 0;
}
```