[P3372]普通平衡树（模板）

Algorithm	平衡树 模板
Created	@Jun 21, 2020 8:13 AM
Difficulty	提高+/省选-
Related to 近期更新 (Property)	
URL	https://www.luogu.com.cn/problem/P3369

题目链接：

【模板】普通平衡树

您需要写一种数据结构（可参考题目标题），来维护一些数，其中需要提供以下操作： 1. 插入 x 数 2. 删除 x 数(若有多个相同的数，因只删除一个) 3. 查询 x 数的排名(排名定义为比当前数小的数的个数 $+1$) 4. 查询排名为 x 的数 5. 求 x 的前驱(前驱定义为小于 x，且最大的数) 6.

https://www.luogu.com.cn/problem/P3369

题解：

平衡树后面学习笔记也许会讲到，看心情吧，有点难。

模板题都不难，细心就行了（是不是好像有点矛盾）

附上AC代码：

```
//
// Created by admin on 2020/6/20.
//
#include <bits/stdc++.h>
using namespace std;
const int inf=1 << 30;
struct treap
{
    int l,r;
    int val,dat;
    int cnt,size;//重复个数，以此节点为根的字数大小
}t[1000000];
int tot,root,n;
int New(int val)
{
    t[++tot].val=val;
    t[tot].size=t[tot].cnt=1;
    t[tot].dat=rand();
    return tot;
}
void update(int p)
{
    t[p].size=t[t[p].l].size+t[t[p].r].size+t[p].cnt;
}
void build()
{
    root=New(-inf);
    t[root].r=New(inf);
    update(root);
}
int GetRankbyVal(int p,int val)
{
    if(p==0)
        return 0;
    if(t[p].val==val)
        return t[t[p].l].size+1;
    if(val<t[p].val)
        return GetRankbyVal(t[p].l,val);//左子树中的排名
    else
        return GetRankbyVal(t[p].r,val)+t[t[p].l].size+t[p].cnt;//右子树中排名+左子树
大小+本节点重复个数
}
int GetValbyRank(int p,int rank)
{
    if(p==0)
        return inf;
    if(t[t[p].l].size>=rank)
        return GetValbyRank(t[p].l,rank);//在左子树中搜索
    if(t[t[p].l].size+t[p].cnt>=rank)
        return t[p].val; //就是本次的值
    return GetValbyRank(t[p].r,rank-t[t[p].l].size-t[p].cnt);//在右子树中搜索，排名减去
左子树大小和本节点重复数
```

```
}
void zig(int &p) //向右旋转（以p为根）
{
    int q=t[p].l;
    t[p].l=t[q].r;
    t[q].r=p;
    p=q; //将根替换为左孩子
    update(t[p].r);update(p); //全部更新
}
void zag(int &p) //向左旋转（以p为根）
{
    int q=t[p].r;
    t[p].r=t[q].l;
    t[q].l=p;
    p=q;  //将根替换成左孩子
    update(t[p].l); update(p); //全部更新
}
void insert(int &p,int val)
{
    if(p==0)
    {
        p=New(val);
        return;
    }
    if(t[p].val==val)
    {
        t[p].cnt++;  //直接添加重复个数
        update(p);
        return ;
    }
    if(val<t[p].val)  //无重复项
    {
        insert(t[p].l,val);  //递归插入
        if(t[p].dat<t[t[p].l].dat)
            zig(p);  //不满足堆性质，右旋
    } else{
        insert(t[p].r,val);
        if(t[p].dat<t[t[p].r].dat)
            zag(p);
    }
    update(p);
}
int pre(int val) // 求前驱
{
    int ans=1;  //t[1].val==-inf
    int p=root;
    while(p)
    {
        if(t[p].val==val) //找到了
        {
            if(t[p].l>0) //节点左子树中的最大值
            {
                p=t[p].l;
                while(t[p].r>0)
                    p=t[p].r;
                ans=p;
            }
            break;
```

```
        }
        if(t[p].val<val && t[p].val>t[ans].val)
            ans=p;  //迭代
        p= val<t[p].val ? t[p].l : t[p].r;
    }
    return t[ans].val;
}
int next(int val) //求后继，与前驱相似
{
    int ans=2;  //t[2].val==inf
    int p=root;
    while(p)
    {
        if(t[p].val==val)
        {
            if(t[p].r>0)
            {
                p=t[p].r;
                while(t[p].l>0)
                    p=t[p].l;
                ans=p;
            }
            break;
        }
        if(t[p].val>val && t[p].val<t[ans].val)
            ans=p;
        p= val < t[p].val ? t[p].l : t[p].r;
    }
    return t[ans].val;
}
void remove(int &p,int val)//将待删节点旋转到叶子上
{
    if(p==0)
        return;
    if(t[p].val==val)
    {
        if(t[p].cnt>1)
        {
            t[p].cnt--;
            update(p);
            return;
        }
        if(t[p].l||t[p].r) //非叶子节点
        {
            if(t[p].r==0 || t[t[p].l].dat>t[t[p].r].dat)
                zig(p),remove(t[p].r,val);// 右旋
            else
                zag(p),remove(t[p].l,val);
            update(p);
        } else
            p=0; //是叶子节点，直接删除
        return;
    }
    val<t[p].val ? remove(t[p].l,val) : remove(t[p].r,val);
    update(p);
}
int main()
{
```

```
    build();
    cin>>n;
    while(n--)
    {
        int opt,x;
        cin>>opt>>x;
        switch (opt) {
            case 1:
                insert(root,x);
                break;
            case 2:
                remove(root,x);
                break;
            case 3:
                cout<<GetRankbyVal(root,x)-1<<endl;
                break;
            case 4:
                cout<<GetValbyRank(root,x+1)<<endl;
                break;
            case 5:
                cout<<pre(x)<<endl;
                break;
            case 6:
                cout<<next(x)<<endl;
                break;
        }
    }
    return 0;
}
```