fhq treap

算法简介

Treap 的无旋版本，支持快速分裂、合并以及**可持久化**。

算法思想

Treap 保证深度的方法为给每个结点赋随机的优先级 r，权值方面 Treap 为二叉搜索树，优先级方面 Treap 为堆。

而当每个点的权值 val，r 确定后树的形态唯一，这与操作顺序无关。而由于 r 的随机性，保证了 Treap 的深度。

fhq treap 利用了 Treap 的这条性质，实现了 $O(\log n)$ 的 split 和 merge。

```
void split(int k,int &k1,int &k2,int v){
    if(!k) return k1=k2=0,void();
    if(node[k].val<=v){
        k1=k;split(node[k].ch[1],node[k1].ch[1],k2,v);pushup(k1);
    }else{
        k2=k;split(node[k].ch[0],k1,node[k2].ch[0],v);pushup(k2);
    }
}
```

split 操作为根据权值 v 将 fhq treap 分成两部分。

每层 split 都会到达结点 k 的子节点，所以时间复杂度为原树的深度，根据 Treap 的性质，深度为 $O(\log n)$。

```
void merge(int &k,int k1,int k2){
    if(!k1||!k2)return k=k1|k2,void();
    if(node[k1].r>node[k2].r){
        k=k1;merge(node[k].ch[1],node[k1].ch[1],k2);pushup(k);
    }else{
        k=k2;merge(node[k].ch[0],k1,node[k2].ch[0]);pushup(k);
    }
}
```

merge 操作为合并两棵树，要求保证前一棵树的结点权值均小于后一棵树的结点权值。

每层 merge 都会到达结点 $k1$ 或 $k2$ 的子节点，所以时间复杂度为两树的深度和，根据 Treap 的性质，深度为 $O(\log n)$。

有了 split 和 merge 操作，不难得到其他操作。

代码模板

洛谷p6136

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
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 LL read_LL(){
    LL 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 char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
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=2e6+5;
struct fhq_treap{
    int root;
    struct Node{
        int val,r,sz,ch[2];
    }node[MAXN];
    int pool[MAXN],top,tot;
    int New(int v){
        int x=top?pool[top--]:++tot;
node[x].val=v,node[x].r=rand(),node[x].sz=1,node[x].ch[0]=node[x].ch[1]=0;
        return x;
    }
    void Del(int x){if(x)pool[++top]=x;}
    void pushup(int
k){node[k].sz=node[node[k].ch[0]].sz+node[node[k].ch[1]].sz+1;}
    void split(int k,int &k1,int &k2,int v){
        if(!k) return k1=k2=0,void();
        if(node[k].val<=v){
            k1=k;split(node[k].ch[1],node[k1].ch[1],k2,v);pushup(k1);
        }else{
            k2=k;split(node[k].ch[0],k1,node[k2].ch[0],v);pushup(k2);
        }
    }
    void merge(int &k,int k1,int k2){
        if(!k1||!k2)return k=k1|k2,void();
        if(node[k1].r>node[k2].r){
            k=k1;merge(node[k].ch[1],node[k1].ch[1],k2);pushup(k);
        }else{
            k=k2;merge(node[k].ch[0],k1,node[k2].ch[0]);pushup(k);
        }
    }
    void insert(int v){
        int lef,rig;
        split(root,lef,rig,v);
        merge(lef,lef,New(v));
        merge(root,lef,rig);
    }
    void erase(int v){
        int lef,mid,rig;
        split(root,lef,rig,v);
        split(lef,lef,mid,v-1);
        Del(mid);
        merge(mid,node[mid].ch[0],node[mid].ch[1]);
        merge(lef,lef,mid);
        merge(root,lef,rig);
    }
    int rank(int v){
        int lef,rig,ans;
```

```
        split(root,lef,rig,v-1);
        ans=node[lef].sz+1;
        merge(root,lef,rig);
        return ans;
    }
    int kth(int rk){
        int pos=root;
        while(true){
            if(rk==node[node[pos].ch[0]].sz+1)return node[pos].val;
            else if(rk<node[node[pos].ch[0]].sz+1)pos=node[pos].ch[0];
            else rk-=node[node[pos].ch[0]].sz+1,pos=node[pos].ch[1];
        }
    }
    int pre(int v){
        int lef,rig,rk;
        split(root,lef,rig,v-1);
        rk=node[lef].sz;
        merge(root,lef,rig);
        return kth(rk);
    }
    int suf(int v){
        int lef,rig,rk;
        split(root,lef,rig,v);
        rk=node[lef].sz+1;
        merge(root,lef,rig);
        return kth(rk);
    }
}tree;
int main()
{
    int n=read_int(),m=read_int(),opt,x,last=0,ans=0;
    while(n--){
        x=read_int();
        tree.insert(x);
    }
    while(m--){
        opt=read_int(),x=read_int()^last;
        switch(opt){
            case 1:
                tree.insert(x);
                break;
            case 2:
                tree.erase(x);
                break;
            case 3:
                ans^=(last=tree.rank(x));
                break;
            case 4:
                ans^=(last=tree.kth(x));
                break;
```

```
            case 5:
                ans^=(last=tree.pre(x));
                break;
            case 6:
                ans^=(last=tree.suf(x));
                break;
        }
    }
    enter(ans);
    return 0;
}
```

From:
https://wiki.cvbbacm.com/ - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E6%97%A0%E6%97%8Btreap&rev=1594182894

Last update: **2020/07/08 12:34**