

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 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

split 操作分两种，分别用于维护集合和序列。第一种 split 操作根据权值 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 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);
    }
}
```

另一种 split 操作根据节点数 sz 将 fhq treap 分成两部分。

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

merge 操作根据优先级 r 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 k_1 或 k_2 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 $O(\log n)$

```
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);
    }
}
```

有了 split 和 merge 操作，不难得到其他操作。

代码模板

集合维护版本

[洛谷p3369](#)

```
const int MAXN=1e5+5;
namespace fhq_treap{
    int root;
    struct Node{
        int val,r,sz,ch[2];
    }node[MAXN];
    int node_cnt;
    int New(int v){
        int k=++node_cnt;
        node[k].val=v;
        node[k].r=rand(),node[k].sz=1,node[k].ch[0]=node[k].ch[1]=0;
        return k;
    }
    #define lch(k) node[node[k].ch[0]]
    #define rch(k) node[node[k].ch[1]]
    void push_up(int k){
        node[k].sz=lch(k).sz+rch(k).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);
            push_up(k1);
        }else{
            k2=k;
            split(node[k].ch[0],k1,node[k2].ch[0],v);
            push_up(k2);
        }
    }
}
```

```

        split(node[k].ch[0], k1, node[k2].ch[0], v);
        push_up(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);
        push_up(k);
    }else{
        k=k2;
        merge(node[k].ch[0], k1, node[k2].ch[0]);
        push_up(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);
    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);
}
#undef lch
#undef rch
};
int main()
{
    int n=read_int(),opt,x;
    while(n--){
        opt=read_int(),x=read_int();
        switch(opt){
            case 1:
                fhq_treap::insert(x);
                break;
            case 2:
                fhq_treap::erase(x);
                break;
            case 3:
                enter(fhq_treap::rank(x));
                break;
            case 4:
                enter(fhq_treap::kth(x));
                break;
            case 5:
                enter(fhq_treap::pre(x));
                break;
            case 6:
                enter(fhq_treap::suf(x));
                break;
        }
    }
    return 0;
}
```

序列维护版本

[洛谷p3391](#)

```
const int MAXN=1e5+5;
namespace Treap{
    int root;
    struct Node{
```

```

    int val;
    int r,sz,ch[2];
    int flip;
}node[MAXN];
int node_cnt;
int new_node(int v){
    int k=++node_cnt;
    node[k].val=v;
    node[k].r=rand(),node[k].sz=1,node[k].ch[0]=node[k].ch[1]=0;
    node[k].flip=0;
    return k;
}
#define lch(k) node[node[k].ch[0]]
#define rch(k) node[node[k].ch[1]]
void push_up(int k){
    node[k].sz=lch(k).sz+rch(k).sz+1;
}
void push_flip(int k){
    swap(node[k].ch[0],node[k].ch[1]);
    node[k].flip^=1;
}
void push_down(int k){
    if(node[k].flip){
        push_flip(node[k].ch[0]);
        push_flip(node[k].ch[1]);
        node[k].flip=0;
    }
}
void split(int k,int &k1,int &k2,int rk){
    if(!k)return k1=k2=0,void();
    push_down(k);
    if(lch(k).sz+1<=rk){
        k1=k;
        split(node[k].ch[1],node[k1].ch[1],k2,rk-lch(k).sz-1);
        push_up(k1);
    }
    else{
        k2=k;
        split(node[k].ch[0],k1,node[k2].ch[0],rk);
        push_up(k2);
    }
}
void merge(int &k,int k1,int k2){
    if(!k1||!k2)return k=k1|k2,void();
    if(node[k1].r>node[k2].r){
        push_down(k1);
        k=k1;
        merge(node[k].ch[1],node[k1].ch[1],k2);
        push_up(k);
    }
    else{

```

```
        push_down(k2);
        k=k2;
        merge(node[k].ch[0],k1,node[k2].ch[0]);
        push_up(k);
    }
}
int Stack[MAXN];
void build(int k){
    if(!k)return;
    build(node[k].ch[0]);
    build(node[k].ch[1]);
    push_up(k);
}
void build(int *a,int n){
    int top=0;
    Stack[top]=0;
    node[0].r=0x7fffffff;
    _for(i,0,n){
        int k=new_node(a[i]);
        while((node[k].r>node[Stack[top]].r)top--;
        node[k].ch[0]=node[Stack[top]].ch[1];
        node[Stack[top]].ch[1]=k;
        Stack[++top]=k;
    }
    node[0].ch[0]=node[0].ch[1]=0;
    root=Stack[1];
    build(root);
}
void flip(int L,int R){
    int lef,mid,rig;
    split(root,lef,rig,R);
    split(lef,lef,mid,L-1);
    push_flip(mid);
    merge(lef,lef,mid);
    merge(root,lef,rig);
}
void dfs(int k,int *a){
    if(!k)return;
    push_down(k);
    dfs(node[k].ch[0],a);
    a[lch(k).sz]=node[k].val;
    a+=lch(k).sz+1;
    dfs(node[k].ch[1],a);
}
};
int a[MAXN];
int main()
{
    int n=read_int(),m=read_int();
    _for(i,0,n)a[i]=i+1;
```

```
Treap::build(a,n);
while(m--){
    int lef=read_int(),rig=read_int();
    Treap::flip(lef,rig);
}
Treap::dfs(Treap::root,a);
_for(i,0,n)space(a[i]);
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

Last update: **2021/08/11 10:37**

