

可持久化平衡树

算法简介

一种可以维护历史版本的平衡树，时间复杂度和空间复杂度均为 $O(m \log n)$

算法思想

使用类似可持久化线段树的方法继承历史版本，但大部分平衡树都自带旋转操作，节点的父子关系会发生，不利于可持久化。

于是考虑使用 `fhq treap` 进行可持久化。

`fhq treap` 的核心操作为 `split` 和 `merge` 所以考虑 `split` 和 `merge` 时动态开点即可。

关于 `merge` 操作是否需要动态开点存在争议。事实上 `fhq treap` 的 `split` 操作会提前为 `merge` 操作开点，但考虑下面代码

```
void erase(int &root,int p,int v){
    int lef,mid,rig;
    split(p,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);
}
```

我们会发现 `merge(mid,node[mid].ch[0],node[mid].ch[1])` 语句并没有 `split` 操作为其开点，这将导致错误。

解决方案为所有相同关键字的节点共用一个位置，或者为 `merge` 操作开点。考虑到前者编程较为复杂，这里选择后者。

需要注意的是，在保证关键字唯一或维护序列时 `merge` 操作不需要开点。

算法模板

集合维护版本

[洛谷p3835](#)

```
const int MAXN=5e5+5,MAXM=40,Inf=0x7fffffff;
int root[MAXN];
struct fhq_treap{
    int tot;
```

```
struct Node{
    int val,r,sz,ch[2];
}node[MAXN*MAXM];
int Copy(int k){
    node[++tot]=node[k];
    return tot;
}
int New(int v){
    int x=++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 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=Copy(k);split(node[k].ch[1],node[k1].ch[1],k2,v);pushup(k1);
    }else{
        k2=Copy(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=Copy(k1);merge(node[k].ch[1],node[k1].ch[1],k2);pushup(k);
    }else{
        k=Copy(k2);merge(node[k].ch[0],k1,node[k2].ch[0]);pushup(k);
    }
}
void insert(int &root,int p,int v){
    int lef,rig;
    split(p,lef,rig,v);
    merge(lef,lef,New(v));
    merge(root,lef,rig);
}
void erase(int &root,int p,int v){
    int lef,mid,rig;
    split(p,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 root,int v){
    int pos=root,ans=0;
    while(true){
        if(!pos)return ans+1;
if(node[pos].val<v)ans+=node[node[pos].ch[0]].sz+1,pos=node[pos].ch[1];
        else pos=node[pos].ch[0];
    }
}
```

```

    }
}
int kth(int root,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 root,int v){
    int pos=root,ans=-Inf;
    while(pos){
        if(node[pos].val>=v)pos=node[pos].ch[0];
        else ans=max(ans,node[pos].val),pos=node[pos].ch[1];
    }
    return ans;
}
int suf(int root,int v){
    int pos=root,ans=Inf;
    while(pos){
        if(node[pos].val<=v)pos=node[pos].ch[1];
        else ans=min(ans,node[pos].val),pos=node[pos].ch[0];
    }
    return ans;
}
}tree;
int main()
{
    int n=read_int(),v,opt,x;
    _rep(i,1,n){
        v=read_int(),opt=read_int(),x=read_int();
        switch(opt){
            case 1:
                tree.insert(root[i],root[v],x);
                break;
            case 2:
                tree.erase(root[i],root[v],x);
                break;
            case 3:
                enter(tree.rank(root[i]=root[v],x));
                break;
            case 4:
                enter(tree.kth(root[i]=root[v],x));
                break;
            case 5:
                enter(tree.pre(root[i]=root[v],x));
                break;
            case 6:
                enter(tree.suf(root[i]=root[v],x));
                break;
        }
    }
}

```

```
    }  
}  
return 0;  
}
```

序列维护版本

洛谷p5055

需要注意的是，可持久化数据结构的标记下放操作需要复制子节点再下方。

原因是可持久化数据结构多个父节点共用一个子节点，若直接对标记进行下放，将影响多个父节点，即影响多个历史版本。

```
const int MAXN=2e5+5,MAXM=80;  
int root[MAXN];  
struct fhq_treap{  
    int tot;  
    struct Node{  
        int r,val,sz,flip,ch[2];  
        LL sum;  
    }node[MAXN*MAXM];  
    int New(int v){  
        int x=++tot;  
        node[x].sum=node[x].val=v,node[x].r=rand(),node[x].sz=1;  
        return x;  
    }  
    int Copy(int k){  
        node[++tot]=node[k];  
        return tot;  
    }  
    void pushup(int k){  
        node[k].sz=node[node[k].ch[0]].sz+node[node[k].ch[1]].sz+1;  
        node[k].sum=node[node[k].ch[0]].sum+node[node[k].ch[1]].sum+node[k].val;  
    }  
    void pushdown(int k){  
        if(node[k].flip){  
            if(node[k].ch[0])node[k].ch[0]=Copy(node[k].ch[0]);  
            if(node[k].ch[1])node[k].ch[1]=Copy(node[k].ch[1]);  
            swap(node[k].ch[0],node[k].ch[1]);  
            node[node[k].ch[0]].flip^=1;  
            node[node[k].ch[1]].flip^=1;  
            node[k].flip=0;  
        }  
    }  
    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=Copy(k);split(node[k].ch[1],node[k1].ch[1],k2,rk-
node[node[k].ch[0]].sz-1);pushup(k1);
        }else{
k2=Copy(k);split(node[k].ch[0],k1,node[k2].ch[0],rk);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){
pushdown(k1);k=k1;merge(node[k].ch[1],node[k1].ch[1],k2);pushup(k);
        }else{
pushdown(k2);k=k2;merge(node[k].ch[0],k1,node[k2].ch[0]);pushup(k);
        }
    }
    void Flip(int &root,int p,int L,int R){
        int lef,mid,rig;
        split(p,lef,rig,R);
        split(lef,lef,mid,L-1);
        node[mid].flip^=1;
        merge(lef,lef,mid);
        merge(root,lef,rig);
    }
    void Insert(int &root,int p,int pos,int v){
        int lef,rig;
        split(p,lef,rig,pos);
        merge(lef,lef,New(v));
        merge(root,lef,rig);
    }
    void erase(int &root,int p,int pos){
        int lef,mid,rig;
        split(p,lef,rig,pos);
        split(lef,lef,mid,pos-1);
        merge(root,lef,rig);
    }
    LL query(int root,int L,int R){
        int lef,mid,rig;
        split(root,lef,rig,R);
        split(lef,lef,mid,L-1);
        return node[mid].sum;
    }
}tree;
int main()
{
    int n,v,opt;
    LL p,x,l,r,last=0;
    n=read_int();
    _rep(i,1,n){
        v=read_int(),opt=read_int();
        switch(opt){
            case 1:
                p=read_LL()^last,x=read_LL()^last;

```

```
tree.Insert(root[i],root[v],p,x);
break;
case 2:
p=read_LL()^last;
tree.erase(root[i],root[v],p);
break;
case 3:
l=read_LL()^last,r=read_LL()^last;
tree.Flip(root[i],root[v],l,r);
break;
case 4:
l=read_LL()^last,r=read_LL()^last;
enter(last=tree.query(root[i]=root[v],l,r));
break;
}
}
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:%E5%8F%AF%E6%8C%81%E4%B9%85%E5%8C%96%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84_2&rev=1595738712
Last update: 2020/07/26 12:45