WELCOME TO *ZERO'S* BLOG.

2015-12-28 | Splay, 算法

Splay学习总结

基本思路

Splay，即伸展树，以其鬼魅的伸展操作而得名。核心操作是通过旋转来把某个节点转到根节点，并通过旋转操作改变树的结构来保证复杂度均摊O(log n)。

旋转操作有单旋和双旋两种，而单旋是可以被卡成O(n)的复杂度的。下面主要介绍双旋：

若目标节点为x，则Splay旋转的情况有三种：

1. 如果x的父节点是根，那么将x旋上去即可
2. 如果x的父节点非根，而爷节点是根，且x和x的父节点处于同一方向上（即能连成一条直线），那么先将x的父节点转上去，再将x转上去。

3. 如果x的父节点非根，而爷节点是根，且x和x的父节点不在同一方向上（即为之字形？），那么先将x旋两次旋至根节点。

实现步骤

本人模板风格大多来源于白书+自己修改

~~（然而机房很多童鞋的代码都是%H，找不到同好啊，so sad）~~

基本操作

- Update(更新)

```
1 inline void Update(int k){
2     tr[k].size=tr[tr[k].ch[0]].size+tr[tr[k].ch[1]].sz+tr[k].s;
3 }
```

- Rotate(旋转)

```
void Rotate(int &k,int d){ //d=0为右旋，d=1为左旋
        int t=tr[k].ch[d^1];
        tr[k].ch[d^1]=tr[t].ch[d]; tr[t].ch[d]=k;
        Update(k); Update(t); k=t;
}
```

维护离散元素

用于维护一些离散的数据

- Splay(伸展)

```
void Splay(int &k,int x){//用于在树中找到元素x并提到根节点
        if(x!=tr[k].key){
                int d1=(x<tr[k].key?0:1),t=tr[k].ch[d1];
                int d2=(x<tr[t].key?0:1);
                if(t && x!=tr[t].key){ //双旋操作
                    Splay(tr[t].ch[d2],x);
                    if(d1==d2) Rotate(k,d1^1); //
                    else Rotate(tr[k].ch[d1],d1);
                }
                Rotate(k,d1^1);
        }
}
```

- Insert(插入)

```
void Insert(int &k,int x){
    if(!k){
                k=++cnt; tr[k].key=x;
                tr[k].ch[0]=tr[k].ch[1]=0; tr[k].size=tr[k].s=1;
                Splay(root,x);
                return;
    }
    tr[k].sz++;
    if(tr[k].key==x){
                tr[k].s++; Splay(root,x);
                return;
    }
    if(tr[k].key>x) Insert(tr[k].ch[0],x);
    else Insert(tr[k].ch[1],x);
}
```

- Delete(删除)

  删除操作需要充分利用到Splay的伸展操作。先找到要删除的元素的前驱和后继，之后把前驱转到根节点，把后继转到根结点的右子节点，根据二叉树的性质，要删除的节点一定在其后继的左子树上且这棵树上就这一个节点，直接删除即可。

```
void Delete(int x){
    Splay(root,x);
    if(tr[root].s>1){
                tr[root].s--; tr[root].sz--;
                return;
    }
    if(!(tr[root].ch[0]*tr[root].ch[1])){
                root=tr[root].ch[0]^tr[root].ch[1];
                return;
    }
    int pre=0,suf=0;
    find_pre(root,x,pre); find_suf(root,x,suf);
    Splay(root,pre); Splay(tr[root].ch[1],suf);
    int pos=tr[tr[root].ch[1]].ch[0];
    tr[pos].sz=tr[pos].s=tr[pos].ch[0]=tr[pos].ch[1]=tr[pos].key=0;
    tr[tr[root].ch[1]].ch[0]=0;
    Update(tr[root].ch[1]); Update(root);
}
```

维护序列

对于维护序列，我们需要在序列的开头和结尾引进两个虚点，来方便我们在序列开头和结尾进行插入和删除等操作。

- Splay(伸展)

```
void Splay(int &k,int x){//用于找到树中第x大的元素并提到树根，一般适用于维护序列的i
//      Pushdown(k);
//      if(tr[k].ch[0]) Pushdown(tr[k].ch[0]);
//      if(tr[k].ch[1]) Pushdown(tr[k].ch[1]);
    int d1=(tr[tr[k].ch[0]].sz<x?1:0),t=tr[k].ch[d1];
    if(d1==1) x-=tr[tr[k].ch[0]].sz+1;
    if(x){
                int d2=(tr[tr[t].ch[0]].sz<x?1:0);
                if(d2==1) x-=tr[tr[t].ch[0]].sz+1;
                if(x){
                    Splay(tr[t].ch[d2],x);
                    if(d1==d2) Rotate(k,d1^1);
                    else Rotate(tr[k].ch[d1],d1);
                }
                Rotate(k,d1^1);
    }
}
```

- Insert(插入)

  当要在序列的pos位置插入元素（新的序列）时，可以将序列第x位的节点转至根，第x+1位的节点转至根的右子节点，然后直接插入到x+1位的节点的左子节点即可。

  （由于一开始引进了虚点，所以我们需要将x+1位的节点转至根，第x+2位的节点转至根

的右子节点）

```
void Insert(int x,char val){
    Splay(root,x+1); Splay(tr[root].ch[1],x+1-tr[tr[root].ch[0]].sz);
    //本应为Splay(tr[root].r,x+2-(tr[tr[root].ch[0]].sz+1));
    int k=++cnt;
    tr[k].key=val; tr[k].ch[0]=tr[k].ch[1]=0;
    tr[tr[root].ch[1]].ch[0]=k;
    Update(k); Update(tr[root].ch[1]); Update(root);
}
```

- Delete(删除)

  类似插入，当删除序列[l,r]段的元素时，将序列第l-1位的元素转至根，第r+1位的元素转至根的右子节点，然后删除其左子节点。为了节省空间，一般我们需要将这些被删除的元素记录到一个循环队列中，重复使用这些被删除的编号。

  （由于之前引进的虚点，我们需要操作的是第l位和第r+2位的节点，下同）

```
void Delete(int l,int r){
    Splay(root,l); Splay(tr[root].ch[1],r+1-tr[tr[root].ch[0]].sz);
    tr[tr[root].ch[1]].ch[0]=0;
    Update(tr[root].ch[1]); Update(root);
}
```

- Reverse(翻转)

  要实现对[l,r]的翻转操作，可以将[l,r]对应的子树中所有节点的左右子节点交换位置，这样即可使树的中序遍历翻转。类似于线段树，我们可以使用对节点的标记和下传来完成对这棵子树的翻转。

```
void Reverse(int l,int r){
    Splay(root,l); Splay(tr[root].ch[1],r+1-tr[tr[root].ch[0]].sz);
    tr[tr[tr[root].ch[1]].ch[0]].flip^=1;
    Pushdown(tr[tr[root].ch[1]].ch[0]);
}
```

- Pushdown(下传)

```
void Pushdown(int k){
    if(tr[k].flip){ //序列翻转标记下传
                tr[k].flip=0;
                swap(tr[k].ch[0],tr[k].ch[1]);
                tr[tr[k].ch[0]].flip^=1;
                tr[tr[k].ch[1]].flip^=1;
    }
}
```

用途&优缺点

用途：实现一系列的序列操作，维护数据什么的……

优点：快

缺点：代码长容易打挂

推荐题目

BZOJ 1014 3223 3224 1251 1500 1503 1507 3506