# SBT节点大小平衡树总结

**SBT是二叉查找树的优化。**

**与二叉平衡树AVL类似，我们定义一棵树：**

1. **这棵树的size为这棵树所有节点的个数**
2. **每棵子树的大小不小于其兄弟的子树大小**

**即s[right[t]]≥s[left[left[t]]], s[right[left[t]]] 且 s[left[t]]≥s[right[right[t]]], s[left[right[t]]]**

**如果插入结点后不满足SBT性质，那么递归修复这棵树的性质，通过旋转达到目的。**

**void Right\_Rotate(int &t){**

**int k=left[t];**

**left[t]=right[k];**

**right[k]=t;**

**s[k]=s[t];**

**s[t]=s[left[t]]+s[right[k]]+1;**

**t=k;**

**return;**

**}**

**void Left\_Rotate(int &t){**

**int k=right[t];**

**right[t]=left[k];**

**left[k]=t;**

**s[k]=s[t];**

**s[t]=s[left[k]]+s[right[t]]+1;**

**t=k;**

**return;**

**}**

**这里可以运用小技巧，在传入变量的时候传入指针，修改变量的同时也修改了原来根节点的值**

**怎么判断旋转的方式？用Maintain函数，形参为根节点与变量flag，false表示当前考虑的是左边的结点 true是右边的结点。其中我们应该判断四个子节点和根节点是否需要修复，但是“左右”，“右左”的情况被证明已经被根节点的修复操作所判断，又少了两个操作**

**void Maintain(int &t,bool flag){**

**if(flag==false){**

**if(s[left[left[t]]]>s[right[t]])**

**Right\_Rotate(t);**

**else if(s[right[left[t]]]>s[right[t]]){**

**Left\_Rotate(left[t]);**

**Right\_Rotate(t);**

**}else return;//一定记住返回**

**}else{**

**if(s[right[right[t]]]>s[left[t]])**

**Left\_Rotate(t);**

**else if(s[left[right[t]]]>s[left[t]]){**

**Right\_Rotate(right[t]);**

**Left\_Rotate(t);**

**}else return;//一定记住返回**

**}**

**Maintain(left[t],false);**

**Maintain(right[t],true);**

**Maintain(t,false);**

**Maintain(t,true);**

**}**

**插入操作：插入元素后修复整棵树**

**void Insert(int &t,int k){**

**if(t==0){//新结点**

**key[++cnt]=k;**

**s[cnt]=1;**

**t=cnt;**

**left[cnt]=right[cnt]=0;**

**}**

**else{**

**s[t]++;**

**if(k<key[t]) Insert(left[t],k);**

**else Insert(right[t],k);**

**Maintain(t,k>=key[t]);**

**}**

**}**

**找前驱：**

**int Pred(int rt,int k){//找前驱**

**if(rt==0) return k;**

**if(k<key[rt]) return Pred(left[rt],k);//如果仍然不是前驱，就继续往前找**

**else return Pred(right[rt],k);**

**}**

**找后继：**

**int Succ(int rt,int k){//找后继**

**if(rt==0) return k;**

**if(k>key[rt]) return Succ(right[rt],k);**

**else return Succ(left[rt],k)**

**}**

**寻找第K小的数**

**int select(int &x,int k)//求第k小数**

**{**

**int r = tree[tree[x].left].size + 1;**

**if(r == k) return tree[x].key;**

**else if(r < k) return select(tree[x].right,k - r);**

**else return select(tree[x].left,k);**

**}**

**询问某元素在树中是第几大**

**int rank(int &x,int key)//求key排第几**

**{**

**if(key < tree[x].key)**

**return rank(tree[x].left,key);**

**else if(key > tree[x].key)**

**return rank(tree[x].right,key) + tree[tree[x].left].size + 1;**

**return tree[tree[x].left].size + 1;**

**}**