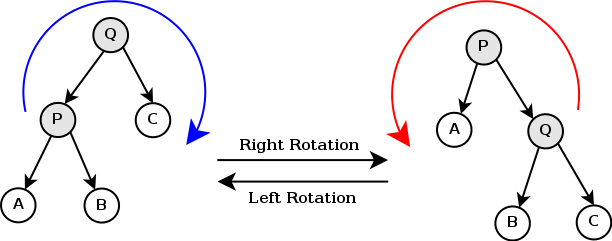
**Tree Rotation**



树的旋转是splay的基础，对于二叉查找树来说，树的旋转不破坏查找树的结构。

**Splaying**

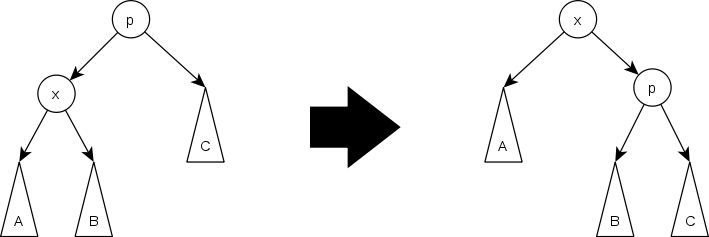
Splaying是Splay Tree中的基本操作，为了让被查询的条目更接近树根，Splay Tree使用了树的旋转操作，同时保证二叉排序树的性质不变。

Splaying的操作受以下三种因素影响：

* 节点x是父节点p的左孩子还是右孩子
* 节点p是不是根节点，如果不是
* 节点p是父节点g的左孩子还是右孩子

同时有三种基本操作：

**Zig Step**

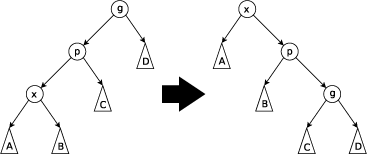
****

当p为根节点时，进行zip step操作。

当x是p的左孩子时，对x右旋；

当x是p的右孩子时，对x左旋。

**Zig-Zig Step**

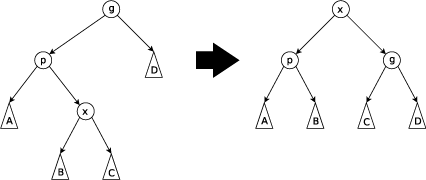


当p不是根节点，且x和p同为左孩子或右孩子时进行Zig-Zig操作。

当x和p同为左孩子时，依次将p和x右旋；

当x和p同为右孩子时，依次将p和x左旋。

**Zig-Zag Step**



当p不是根节点，且x和p不同为左孩子或右孩子时，进行Zig-Zag操作。

当p为左孩子，x为右孩子时，将x左旋后再右旋。

当p为右孩子，x为左孩子时，将x右旋后再左旋。

具体操作包括：

1、access(i,t):如果i在树t中，则返回指向它的[指针](http://baike.baidu.com/view/159417.htm)，否则返回空指针。为了实现access(i,t)，可以从树t的根部向下查找i。如果 查找操作遇到了一个含有i的节点x，就在x处进行splay操作，并返回指向x的[指针](http://baike.baidu.com/view/159417.htm)，访问结束。如果遇到了空[指针](http://baike.baidu.com/view/159417.htm)，表示i不在树中，此时就在最后一个非 空[节点](http://baike.baidu.com/view/47398.htm)处进行splay操作，然后返回空指针。如果树是空的，将忽略掉splay操作。

2、insert(i,t):将条目i插入树t中（假设其尚不存在）。为了实现insert(i,t)，首先执行split(i,t)，然后把t换成一个由新的包含有i的根节点组成的树，这个根节点的左右子树分别是split返回的树t1和t2。

3、delete(i,t):从树t中删除条目i（假设其已经存在）。为了实现delete(i,t)，首先执行access(i,t)，然后把t换成其左子树和右子树join之后的新树。

4、join(t1,t2):将树t1和t2合并成一棵树，其中包含之前两棵树的所有条目，并返回合并之后的树。这个操作假设t1中的所有条目都小于t2 中的条目，操作完成之后会销毁t1和t2。为了实现join(t1,t2)，首先访问t1中最大的条目i。访问结束之后，t1的根节点中包含的就是i，它 的右孩子显然为空。于是把t2作为这个根节点的右子树并返回完成之后的新树即可实现join操作。

5、split(i,t):构建并返回两棵树t1和t2，其中t1包含t中所有小于等于i的条目，t2包含t中所有大于i的条目。操作完成之后销毁t。为 了实现split(i,t)，首先执行access(i,t)，然后根据新根节点中的值是大于还是小于等于i来切断这个根节点的左链接或右链接，并返回形 成的两棵树。

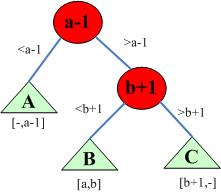
另外insert和delete方法有更好的实现，[时间复杂度](http://baike.baidu.com/view/104946.htm)更小：

1、insert(i, t):查找i，把遇到的空指针替换成一个含有i的新节点，然后再在新节点处对树进行splay操作。

2、delete(i, t):查找含有i的节点，设此节点为x，其父节点为y。把x的左右子树合并之后替换掉x，然后再从y处进行splay操作。

**3、伸展树区间操作**

在实际应用中，伸展树的中序遍历即为我们维护的数列，这就引出一个问题，怎么在伸展树中表示某个区间？比如我们要提取区间[a,b]，那么我们将a前面一个数对应的结点转到树根，将b 后面一个结点对应的结点转到树根的右边，那么根右边的左子树就对应了区间[a,b]。原因很简单，将a 前面一个数对应的结点转到树根后， a 及a 后面的数就在根的右子树上，然后又将b后面一个结点对应的结点转到树根的右边，那么[a,b]这个区间就是下图中B所示的子树。



利用区间操作我们可以实现线段树的一些功能，比如回答对区间的询问（最大值，最小值等）。具体可以这样实现，在每个结点记录关于以这个结点为根的子树的信息，然后询问时先提取区间，再直接读取子树的相关信息。还可以对区间进行整体修改，这也要用到与线段树类似的延迟标记技术，即对于每个结点，额外记录一个或多个标记，表示以这个结点为根的子树是否被进行了某种操作，并且这种操作影响其子结点的信息值，当进行旋转和其他一些操作时相应地将标记向下传递。

与线段树相比，伸展树功能更强大，它能解决以下两个线段树不能解决的问题：

（1） 在a后面插入一些数。方法是：首先利用要插入的数构造一棵伸展树，接着，将a 转到根，并将a 后面一个数对应的结点转到根结点的右边，最后将这棵新的子树挂到根右子结点的左子结点上。

（2）  删除区间[a,b]内的数。首先提取[a,b]区间，直接删除即可。

const int SPLAYmaxn=200005;

const int SPLAYinf=100000000;

struct Splay\_Node

{

    int l,r,fa,v,sum;

};

struct Splay

{

    Splay\_Nodet[SPLAYmaxn];

    int root,tot;

};

void create()

{

    root=1,tot=2;

    t[1].v=-SPLAYinf;

    t[2].v=SPLAYinf;

    t[1].r=t[1].sum=2;

    t[2].fa=t[2].sum=1;

}

void update(int now)

{

    t[now].sum=t[t[now].l].sum+t[t[now].r].sum+1;

}

void left(int now)

{

    int fa=t[now].fa;

    t[now].fa=t[fa].fa;

    if(t[t[fa].fa].l==fa)

        t[t[fa].fa].l=now;

    if(t[t[fa].fa].r==fa)

        t[t[fa].fa].r=now;

    t[fa].fa=now;

    t[fa].r=t[now].l;

    t[t[now].l].fa=fa;

    t[now].l=fa;

    update(fa);

}

void right(int now)

{

    int fa=t[now].fa;

    t[now].fa=t[fa].fa;

    if(t[t[fa].fa].l==fa)

        t[t[fa].fa].l=now;

    if(t[t[fa].fa].r==fa)

        t[t[fa].fa].r=now;

    t[fa].fa=now;

    t[fa].l=t[now].r;

    t[t[now].r].fa=fa;

    t[now].r=fa;

    update(fa);

}

void splay(int now,int FA=0)

{

    while(t[now].fa!=FA)

    {

        int fa=t[now].fa;

        if(t[fa].fa==FA)

            if(t[fa].l==now)

                right(now);

            else

                left(now);

        else

            if(t[t[fa].fa].l==fa)

                if(t[fa].l==now)

                    right(fa),right(now);

                else

                    left(now),right(now);

            else

                if(t[fa].l==now)

                    right(now),left(now);

                else

                    left(fa),left(now);

    }

    update(now);

    if(!FA)

        root=now;

}

int lower\_bound(int v)

{

    int ans=0,la=0;

    for(int now=root;now;)

    {

        la=now;

        if(t[now].v>=v)

            ans=now,now=t[now].l;

        else

            now=t[now].r;

    }

    splay(la);

    return ans;

}

void insert(int v)

{

    for(int now=root;;)

    {

        ++t[now].sum;

        if(t[now].v>=v)

            if(t[now].l)

                now=t[now].l;

            else

            {

                t[now].l=++tot;

                t[tot].sum=1;

                t[tot].fa=now;

                t[tot].v=v;

                splay(tot);

                return;

            }

        else

            if(t[now].r)

                now=t[now].r;

            else

            {

                t[now].r=++tot;

                t[tot].sum=1;

                t[tot].fa=now;

                t[tot].v=v;

                splay(tot);

                return;

            }

    }

}

int get\_lower(int a)

{

    splay(a);

    for(a=t[a].l;t[a].r;a=t[a].r);

    return a;

}

int get\_upper(int a)

{

    splay(a);

    for(a=t[a].r;t[a].l;a=t[a].l);

    return a;

}

int get\_rank(int a)

{

    splay(a);

    return t[t[a].l].sum;

}

void del(int l,int r)

{

    l=get\_lower(l);

    r=get\_upper(r);

    splay(l);

    splay(r,l);

    t[r].l=0;

    update(r);

    update(l);

}

int get\_kth(int k)

{

    ++k;

    for(int now=root;;)

    {

        if(t[t[now].l].sum==k-1)

            return now;

        if(t[t[now].l].sum>=k)

            now=t[now].l;

        else

            k-=t[t[now].l].sum+1,now=t[now].r;

    }

}