```
public boolean equal(BST <T> t2)
{
    return equal(root,t2.root);
}

private boolean equal(BSTNode <T> t1,BSTNode <T> t2)
{
    if (t1 == null && t2 == null)
     return true;
    else if (t1 == null || t2 == null)
     return false;
    else if(t1.key != t2.key)
     return false;
    return equal(t1.left,t2.left) && equal(t1.right,t2.right);
}

public boolean isFull()
{
    return isFull(root);
}

private boolean isFull(BSTNode <T> t)
{
    return countNodes(t) == Math.pow(2,height(t))  1;
}

public boolean isBSTNoStack()
{
        return isBSTNoStack(root);
}
```

```
private boolean isBSTNoStack(BSTNode<T> t)
{
            boolean bst = true;

            if (t != null)
            {
                  if (t.left != null)
                  {
                        if (t.key < t.left.key)
                              bst = false;

                        bst = bst && isBSTNoStack(t.left);
                  }

                  if (t.right != null)
                  {
                        if (t.key > t.right.key)
                              bst = false;

                        bst = bst && isBSTNoStack(t.right);
                  }
            }

            return bst;
      }

 public BST<T> copyBST()
 {
   if (root == null)
    return null;
   BST<T> t = new BST<T>();
   copy(root,t);
   return t;
 }

 private void copy(BSTNode <T> t1,BST<T> t2)
 {
   if (t1 != null)
   {
    t2.insert(t1.key,t1.data);
    copy(t1.left,t2);
    copy(t1.right,t2);
   }
 }
 public BST<T> reverseBST()
 {
    if (root == null)
     return null;
    BST<T> t = new BST<T>();
    reverse(root,t);
    return t;
 }

private void reverse(BSTNode <T> t1,BST<T> t2)
 {
    if (t1 != null)
    {
     t2.root = t2.insertReverse(t2.root,t1.key,t1.data);
     reverse(t1.left,t2);
     reverse(t1.right,t2);
    }
 }


        private BSTNode <T> insertReverse(BSTNode <T> t,int key,T data)
 {
    if (t == null)
    {
     t = new BSTNode<T>(key,null,null);
     t.data = data;
    }
    else if (key > t.key)
            t.left = insertReverse(t.left,key,data);
    else if (key < t.key)
            t.right = insertReverse(t.right,key,data);
    else
            System.out.println("Duplicates not allowed");
    return t;
 }
```

```
public void mirror()
{
   mirror(root);
}
private void mirror(BSTNode <T> t)
{
   if (t != null)
   {
    mirror(t.left);
    mirror(t.right);
    BSTNode <T> temp = t.left;
    t.left = t.right;
    t.right = temp;
   }
}

private void printByLevel(BSTNode<T> t)
{
                if (t != null)
                {
                        LinkQueue<BSTNode<T>> q = new LinkQueue<BSTNode<T>>();
                        q.enqueue(t);

                        while (q.length() != 0)
                        {
                                t = (BSTNode<T>) q.serve();
                                System.out.println(t.data);

                                if (t.left != null)
                                        q.enqueue(t.left);

                        }
                }
}
```