```
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 isBST()
{
    return isBST(root);
}

private boolean isBST(BSTNode <T> t)
{
    boolean bst = true;
    if (t != null)
    {
      if (t.left != null)
      {
           if (t.key < t.left.key)
                bst = false;

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

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

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

    return bst;
}
```

```
private boolean isLeaf(BSTNode <T> t)
{
    if (t.left == null && t.right == null)
      return true;
    else
      return false;
}

public int countNodes()
{
 return countNodes(root);
}

private int countNodes(BSTNode <T> t)
{
    if (t == null)
      return 0;

    return 1 + countNodes(t.left) + countNodes(t.right);
}

public int totalNodes()
{
 return totalNodes(root);
}

private int totalNodes(BSTNode <T> t)
{
    if (t == null)
      return 0;

    return t.key + countNodes(t.left) + countNodes(t.right);
}

public int avg()
{
 if (root == null)
      return 0;
 else
      return totalNodes() / countNodes();
}
```

```
public int countParents()
{
 return countParents(root);
}

private int countParents(BSTNode <T> t)
{
    if (t == null || (t.left == null && t.right == null))
      return 0;

    return 1 + countParents(t.left) + countParents(t.right);
}

public int countLeaf()
{
 return countLeaf(root);
}

private int countLeaf(BSTNode <T> t)
{
    if (t == null)
      return 0;
    else if (t.left == null && t.right == null)
      return 1;

    return countLeaf(t.left) + countLeaf(t.right);
}

public int countOneChild()
{
 return countOneChild(root);
}

private int countOneChild(BSTNode <T> t)
{
    if (t == null)
      return 0;
    else if ((t.left == null && t.right != null)
           || (t.left != null && t.right == null))
      return 1 + countOneChild(t.left) + countOneChild(t.right);

    return countOneChild(t.left) + countOneChild(t.right);
}
```

```
    public int height()
    {
     return height(root);
    }

    private int height(BSTNode <T> t)
    {
        if (t == null)
          return 0;

        return 1 + Math.max(height(t.left),height(t.right));
    }

    public int countLevel(int level)
    {
        return countLevel(root,0,level);
    }

    private int countLevel(BSTNode <T> t,int l,int level)
    {
        if (t == null)
          return 0;

        l++;

        if(l == level)
          return 1 + countLevel(t.left,l,level) +
countLevel(t.right,l,level);
        else
          return countLevel(t.left,l,level) +
countLevel(t.right,l,level);
    }

    public BST copyBST()
    {
        if (root == null)
          return null;

        BST t = new BST();
        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;
    }
}

public T findMax()
{
    return findMax(root);
}

private T findMax(BSTNode <T> t)
{
    while(t.right != null)
      t = t.right;

    return t.data;
}

public T findMinRec()
{
    return findMinRec(root);
}

private T findMinRec(BSTNode <T> t)
{
    if (t == null)
      return null;
    else if (t.left != null)
      return findMinRec(t.left);
    else
      return t.data;
}
```

```
public T findSuccessor(int tkey)
{
    findkey(tkey);

    return findSuccessor(current);
}

private T findSuccessor(BSTNode <T> t)
{
    return findMin(t.right);
}

public T findPredessor(int tkey)
{
    findkey(tkey);

    return findPredessor(current);
}

private T findPredessor(BSTNode <T> t)
{
    return findMax(t.left);
}

public BSTNode <T> findSmallestKth(int k)
{
    return findKthSmallest(root,k);
}

private BSTNode <T> findKthSmallest(BSTNode <T> root, int k)
{
     if (root == null)
       return null;  // can't find anything, empty

     int numLeft = countNodes(root.left);

     if (numLeft + 1 == k) // current node
        return root;
     else if (numLeft >= k) // in left subtree
        return findKthSmallest(root.left, k);
     else
        return findKthSmallest(root.right, k - (numLeft + 1));
 }
```

```
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);

           if (t.right != null)
                q.enqueue(root.right);
      }
    }
}

public void printLevel(int level)
{
    printLevel(root,0,level);
}

private void printLevel(BSTNode <T> t, int l, int level)
{
    if (t !=  null)
    {
      l++;

      if(l == level)
           System.out.print(t.data + " ");
      else if (l < level)
      {
           printLevel(t.left,l,level);
           printLevel(t.right,l,level);
      }
    }
}
```

```
    private void printDescendents(BSTNode <T> t)
    {
        if (root != null)
        {
                printDescendents(t.left);

                System.out.println("Number of descendebts of node "
                                        + t.key + " : " +
(countNodes(t) - 1));

                printDescendents(t.right);
        }
    }

    public String pathBST(int k)
    {
        boolean found = findkey(k);
        String path = null;

        if(found)
        {
          path = "";

          BSTNode <T> p = current;

          while(p != root)
          {
               path += p.data + " ";
               p = findparent(p);
          }

          path += p.data + " ";
        }

        return path + "\n";
    }
```