```
public class BST<T> {
	private BSTNode<T> root, current;

	public BST() {
		root = current = null;
	}

	public boolean empty() {
		return root == null ? true : false;
	}

	public void traverse(Order ord) {
		switch (ord) {
		case preOrder:
			preOrder(root);
			break;
		case inOrder:
			inOrder(root);
			break;
		case postOrder:
			postOrder(root);
			break;
		}
	}

	private void preOrder(BSTNode<T> p) {
		if (p != null) {
			System.out.println(p.key);
			preOrder(p.left);
			preOrder(p.right);
		}
	}

	private void inOrder(BSTNode<T> p) {
		if (p != null) {
			inOrder(p.left);
			System.out.println(p.key);
			inOrder(p.right);
		}
	}

	private void postOrder(BSTNode<T> p) {
		if (p != null) {
			postOrder(p.left);
			postOrder(p.right);
			System.out.println(p.key);
		}
	}
```

```
private BSTNode<T> findparent(BSTNode<T> p) {
        LinkStack<BSTNode<T>> stack = new LinkStack<BSTNode<T>>();
        BSTNode<T> q = root;

        while (q.right != p && q.left != p) {
                if (q.right != null)
                        stack.push(q.right);

                if (q.left != null)
                        q = q.left;
                else
                        q = stack.pop();
        }

        return q;
}

public T retrieve() {
        return current.data;
}

public boolean findkey(int tkey) {
        BSTNode<T> p, q;
        p = root;
        q = root;
        if (empty())
                return false;
        while (p != null) {
                q = p;
                if (p.key == tkey) {
                        current = p;
                        return true;
                } else if (tkey < p.key)
                        p = p.left;
                else
                        p = p.right;
        }
        current = q;
        return false;
}
```

```
public boolean insert(int k, T val) {
        BSTNode<T> p, q = current;
        if (findkey(k)) {
                current = q;
                return false;
        }

        p = new BSTNode<T>(k, val);

        if (empty()) {
                root = current = p;
                return true;
        } else {
                if (k < current.key)
                        current.left = p;
                else
                        current.right = p;
                current = p;
                return true;
        }
}

public boolean remove_key(int tkey) {
        Flag removed = new Flag(false);
        BSTNode<T> p;
        p = remove_aux(tkey, root, removed);
        current = root = p;
        return removed.get_value();
}
```

```
	private BSTNode<T> remove_aux(int key, BSTNode<T> p, Flag flag) {
		BSTNode<T> q, child = null;
		if (p == null)
			return null;
		if (key < p.key)
			p.left = remove_aux(key, p.left, flag);
		else if (key > p.key)
			p.right = remove_aux(key, p.right, flag);
		else {
			flag.set_value(true);
			if (p.left != null && p.right != null) {
				q = find_min(p.right);
				p.key = q.key;
				p.data = q.data;
				p.right = remove_aux(q.key, p.right, flag);
			} else {
				if (p.right == null)
					child = p.left;
				else if (p.left == null)
					child = p.right;
				return child;
			}
		}
		return p;
	}

	private BSTNode<T> find_min(BSTNode<T> p) {
		if (p == null)
			return null;
		while (p.left != null)
			p = p.left;
		return p;
	}

	public boolean update(int key, T data) {
		remove_key(current.key);
		return insert(key, data);
	}
}
```