

RED-BLACK TREE

I. Java Implementation

wellConnected

```
public boolean wellConnected(Entry<V> expectedParent) {
    boolean ok = true;
    if (expectedParent != parent) {
        return false;
    }
    if (right != null) {
        ok = ok && right.wellConnected(this);
    }
    if (left != null) {
        ok = ok && left.wellConnected(this);
    }
    if(right==left && right!=null && left!=null) {
        return false;
    }
    return ok;
}
```

redConsistency

```
protected boolean redConsistency() {
    boolean ret = true;
    if (color == RED && ((left != null && left.color == RED) || (right != null && right.color == RED)))
        return false;
    if (left != null) {
        ret = ret && left.redConsistency();
    }
    if (right != null) {
        ret = ret && right.redConsistency();
    }
    return ret;
}
```

blackConsistency

```
protected int blackHeight() {
    int ret = 0;
    if (color == BLACK) {
```

```

        ret = 1;
    }
    if (left != null) {
        ret += left.blackHeight();
    }
    return ret;
}

protected boolean blackConsistency() {
    if (color != BLACK){// root must be black
        return false;
    }
    // the number of black nodes on way to any leaf must be same
    if (!consistentlyBlackHeight(blackHeight())) {
        return false;
    }
    return true;
}

protected boolean consistentlyBlackHeight(int height) {
    boolean ret = true;
    if (color == BLACK)
        height--;
    if (left == null) {
        ret = ret && (height == 0);
    }else{
        ret = ret && (left.consistentlyBlackHeight(height));
    }
    if (right == null) {
        ret = ret && (height == 0);
    } else {
        ret = ret && (right.consistentlyBlackHeight(height));
    }
    return ret;
}

```

ordered

```

boolean ordered() {
    return ordered(this,new Range());
}

```

```

boolean ordered(Entry<V> t, Range range) {
    if(t == null) {
        return true;
    }
    if(!range.inRange(t.key)) {
        return false;
    }
    boolean ret=true;
    ret = ret && ordered(t.left,range.setUpper(t.key));
    ret = ret && ordered(t.right,range.setLower(t.key));
    return ret;
}

```

remove

```

@Assertion(value = { @Case(
pre = "(root == null || root.consistency())&& size==realSize()",
post = "(root == null || root.consistency())&& size==realSize()") })
public V remove(int key) {
    Entry<V> p = getEntry(key);
    if (p == null)
        return null;
    V oldValue = p.value;
    deleteEntry(p);
    return oldValue;
}

private int realSize() {
    if(root==null) {
        return 0;
    }
    return root.size();
}

```