Some practice with Red Black Trees

Consider a red-black tree implementation in Java, where each node of the tree is represented like so:

```java
public class RBNode< E extends Comparable<E> > {
    public E key;
    public RBNode< E > parent;
    public RBNode< E > left;
    public RBNode< E > right;
    public boolean red;

    public boolean isNullLeaf () {
        return key == null;
    }
}
```

Notice first that the key can be null, marking the node as being a null leaf. Note also that a leaf can be red (when red is true) or black (when red is false). Finally, don’t worry about the detail of E extending the Comparable contain interface; that detail guarantees that the keys are ordered, which is necessary to form a binary search tree, but not particularly germane to how red-black tree operations.

Write a method that takes a pointer to the root node of a tree, and then returns whether that tree is a valid red-black tree. That is, the tree must fulfill the standard red-black tree properties:

1. Each node is colored red or black.
2. The root note is black.
3. Each null leaf is black.
4. A red node can have only black children.
5. At each node, the path to each null leaf must traverse an equal number of black nodes.

Your method certainly may (and probably should) call on a number of helper methods.

---

1This property requires no evaluation; the RBNode objects are all red or black.
1 How to submit your work

Don’t. This is practice for mid-term 2. I will post solutions soon, but work on these questions yourself, or in groups, or both to see how far you can get.