Übungen zu Objektorientierte Programmierung 2

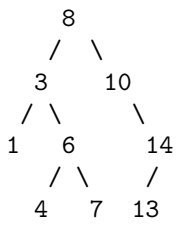
frieder.loch@ost.ch

Woche 10: Trees

Aufgabe 1: Preorder, Inorder und Postorder

Aufgabe 1.1: Preorder, Inorder und Postorder

Gegeben ist folgender Binärer Baum:

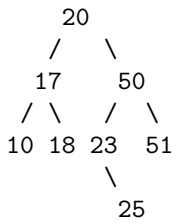


In welcher Reihenfolge werden die Nodes in der

- Preorder-Traversierung
- Inorder-Travesierung
- Postorder-Travesierung besucht?

Aufgabe 1.2: addNode, sibling

Gegeben ist der folgende Binäre Suchbaum:



Fügen Sie den die Werte 1, 5, 19, 24, 26 und 52 in dieser Reihenfolge in den Baum ein. Was ist der sibling von 1, 18, 23 und 26?

Aufgabe 2: Binärer-Such-Baum

In der Vorlesung haben Sie Bäume kennengelernt. Ihre Aufgabe ist es nun einen BinarySearchTree zu implementieren. Nutzen sie dazu das Interface `IBinarySearchTree`.

```
public interface IBinarySearchTree<E> extends Tree<E> {
    // Returns the left Node of the passed Node
    Node<E> left(Node<E> p);
    // Returns the right Node of the passed Node
    Node<E> right(Node<E> p);
    // Returns the Sibling Node. Nodes are siblings if they share the same parent
    Node<E> sibling(Node<E> p);
    // Adds a new Node in the Tree
    Node<E> addNode(E value);
    // Removes a Node from the Tree
    Node<E> removeNode(E value);
    // Returns the Root of the Tree
    Node<E> getRoot();
    // Returns the children of a Node
    public Iterable<Node<E>> children(Node<E> p) throws IllegalArgumentException;
    // Returns the number of children a Node has
    public int numChildren(Node<E> p) throws IllegalArgumentException;
    // Returns if a Node is Internal (Has Children)
    public boolean isInternal(Node<E> p) throws IllegalArgumentException;
    // Returns if a Node is External (Has no Children)
    public boolean isExternal(Node<E> p) throws IllegalArgumentException;
    // Returns if a Node is the root
    public boolean isRoot(Node<E> p) throws IllegalArgumentException;
    // Returns all Nodes inorder
    List<E> inorderTraversal();
    // Numbers of Nodes in the Tree
    int size();
}
```

Hinweise zur Implementierung von `removeNode`:

Es müssen drei Fälle unterschieden werden:

- Fall 1: Der Knoten ist ein Blatt.
- Fall 2: Der Knoten hat genau ein Kind.
- Fall 3: Der Knoten hat zwei Kinder. Tipp: Hier wird der Inorder-Nachfolger (der kleinste Knoten im rechten Teilbaum) oder der Inorder-Vorgänger (der grösste Knoten im linken Teilbaum) benötigt.

Überlegt euch wie der Baum im entsprechenden Fall angepasst werden muss.