Grundlagen der Programmkonstruktion
Übungsblatt 6 (zu lösen bis 24./25./26. Juni 2013)

1 Baum-Datenstruktur (0.5 Punkt)

Erstellen Sie eine generische Klasse `MyTree`. In dieser soll man binäre Bäume abspeichern können, wobei jeder Knoten eine Instanz des generischen Typs enthält. Zu diesem Zweck, erstellen Sie eine innere Node-Klasse und einen Konstruktur in Tree, der einen leeren Baum erstellt. Der generische Typ selbst soll das `Comparable`-Interface implementieren.

Erstellen Sie eine `String toString()`-Methode, die die Struktur des Baums ausgibt.

2 TreeSet

Übernehmen Sie die Baumklasse aus der vorigen Aufgabe und erstellen sie daraus die Klasse `MyTreeSet`. Diese TreeSet ist ein binärer Baum mit folgenden Eigenschaften (nach dem Aufruf jeder Methode müssen die folgenden Vorgaben erfüllt sein):

- Der aktuelle Knoten enthält ein Element `value`.
- Alle Knoten im linken Teilbaum sind *kleiner* als `value`.
- Alle Knoten im rechten Teilbaum sind *größer* als `value`.
- Der Baum enthält jedes Element nur einmal.
- Es gibt keine weiteren Vorgaben (sie müssen nicht darauf achten, dass der Baum balanciert ist).

Hinweise: Nutzen Sie in den folgenden Methoden die Eigenschaften des Baumes und vermeiden Sie unnötiges Kopieren. Achten Sie bei der Implementierung auf Sonderfälle wie z.B. leere Bäume. Sie sollen bei der Präsentation der Lösung in der Lage sein, eine Instanz von MyTreeSet mit einigen Elementen zu zeichnen und zu erklären, wie Ihre Methoden funktionieren. Benutzen Sie in Ihren Lösungen zu `add` und `remove` nicht die Methoden `contains()` oder `size()`. Besonders wenn Sie iterative Lösungsansätze wählen, achten Sie darauf, dass diese für beliebig große Bäume auch wirklich funktionieren. Beim Bestimmen der Laufzeit genügt es, wenn Sie eine asymptotische Abschätzung der Laufzeit in Abhängigkeit zur Anzahl der Knoten im Baum n geben. Als Abschätzung für die durchschnittliche Tiefe der Endknoten verwenden Sie $log(n)$. Geben Sie jeweils die Laufzeit für den durchschnittlichen Fall und auch den worst-case an. Sie sollten in der Lage sein, ein Beispiel mit dem worst-case zu bestimmen.

2.1 Methode `boolean add(A a)` (1 Punkt)

Erstellen Sie eine Methode `boolean add(A a)`, die ein Element der Baum-Menge hinzufügt. Sollte das Element bereits in der `MyTreeSet` vorhanden sein, soll `false` zurückgeliefert werden (in diesem Fall soll die `MyTreeSet` nicht verändert werden), ansonsten `true`.

Bestimmen Sie die Laufzeit ihrer Methode.

2.2 Methode `boolean remove(A a)` (1 Punkt)

Erstellen Sie die Methode `boolean remove(A a)`, die ein Element aus der Baum-Menge entfernt. Sollte das Element in der `MyTreeSet` vorhanden sein, soll `true` zurückgeliefert werden, ansonsten `false` (in diesem Fall soll die `MyTreeSet` nicht verändert werden).

Beachten Sie bei dieser Aufgabe, dass auch Zwischenknoten gelöscht werden können!

Bestimmen Sie die Laufzeit ihrer Methode.

2.3 Methode `boolean contains(A a)` (0.5 Punkte)

Erstellen Sie die Methode `boolean contains(A a)`. Die Methode soll `true` zurückliefern, wenn die Menge `a` enthält, ansonsten `false`.

Bestimmen Sie die Laufzeit ihrer Methode.

2.4 Methode `int size()` (0.5 Punkte)

Erstellen Sie die Methode `int size()`. Diese soll die Anzahl der Elemente in der aktuellen TreeSet zurückliefern.

Bestimmen Sie die Laufzeit ihrer Methode.

3 Iterator ohne remove (1.5 Punkte)

Implementieren Sie das `Iterable<A>`-Interface in `MyTreeSet`. Die Methode `iterator()` soll dabei einen Iterator zurückliefern, der alle Elemente der TreeSet in *absteigender* Reihenfolge zurückliefert. Sie müssen dabei nicht die Methode `remove()` implementieren.

Bestimmen Sie die Laufzeit des Konstruktors, der `hasNext()`-Methode und der `next()`-Methode und begründen Sie Ihre Lösung. Begründen Sie auch, ob Ihre Lösung "optimal" im Sinne der Laufzeit ist bzw. wie man eine bessere Lösung erstellen könnte.