PR2 Aufgabenblatt für Testate Nr. 3

Allgemeine Hinweise

Ausgabe der Übung:	Di., 05.05.2020
Abgabe:	Mo., 18.05.2020 bis 16 Uhr in Moodle.
Testat am:	Di., 19.05.2020

- Die Abnahme der Übungen gilt als **Prüfungsleistung**. Bei einer Verhinderung durch Krankheit ist eine ärztliche Bescheinigung der Arbeitsunfähigkeit vorzulegen.
- Laden Sie Ihre Lösungen bis zur Deadline in Moodle bei der entsprechenden Unteraufgabe hoch. Quellcodes müssen einheitlich und sinnvoll formatiert sein (vorzugsweise mithilfe Ihrer IDE wie Eclipse oder IntelliJ). **Ausarbeitungen in einem anderen Format werden nicht berücksichtigt**.
- Während der Abnahme sind die Ergebnisse am Rechner live zu demonstrieren.
- Bei der Abnahme der Übung ist der Studentenausweis vorzulegen.

Lernziele

- Datenstruktur *verkettete Liste* verstehen, selbst entwickeln und anwenden.
- Datenstruktur *binärer Baum* verstehen, selbst entwickeln und anwenden.
- Datenstruktur *binärer Suchbaum* verstehen, selbst entwickeln und anwenden.

Hinweise zur Bearbeitung

- Sie finden den in diesen Übungen verwendeten Source-Code im Moodle-Kurs unter *Source-Code Gumbel*. Die Source-Dateien enthalten zahlreiche Beispiele und teilweise noch nicht fertig programmierten Java-Programme. Nur einige davon werden in den Übungen benötigt. Es empfiehlt sich aber, alle Dateien in ihr Projekt zu übernehmen.
- Vermeiden Sie unterschiedliche Lösungen, wenn ein Algorithmus in verschiedenen Darstellungsformen (Struktogramm, Pseudocode, usw.) verlangt ist.
- Formatieren Sie den Sourcecode mithilfe Ihrer IDE (Eclipse, IntelliJ o. ä.) einheitlich. Das Einrücken erfolgt durch 2 oder 4 Leerzeichen oder Tabs entsprechender Länge.

1 Aufgabe: `toListString` in `ListNode`

In der einfachsten Form kann man eine verkettete Liste einfach als eine Verkettung von Listenknoten auffassen. Die Klasse `ListNode` im Package `de.hsmannheim.inf.pr2.ads` ist eine Implementierung davon. Schreiben Sie dafür eine Methode `String toListString()`, die alle Elemente in dieser verketteten als String zurückgibt. Hierbei sollen die Elemente in runde Klammern eingefasst werden und durch Komma und gefolgt von einem Leerzeichen getrennt werden. Erzeugen Sie in einem Hauptprogramm verkettete Listen bestehend aus Elementen von `ListNode` und geben Sie diese mittels `toListString` auf die Konsole aus. Es sollen zwei Listen erzeugt werden, die folgende Ausgaben auf der Konsole haben:

- `(1, 1, 2, 3, 5, 8)`
- `(Hallo, Welt, das, Wetter, ist, schön)`

Die erste Liste enthält ganze Zahlen, die zweite Liste Strings. **Hinweis**: `toString` könnte auch für diese Methode genutzt werden – allerdings gibt es diese überschriebene Methode schon. Sie liefert nur das einzelne Element des Listenknoten zurück.

2 Aufgabe: `containsIter()` in `ListNode`

Nun soll für die Klasse `ListNode` die Methode `boolean containsIter(E e)` geschrieben werden, die für ein übergebenes Element ermittelt, ob dieses in der Liste enthalten ist. Die Methode gibt `true` zurück, wenn das gesuchte Elemente mindestens einmal in der Liste enthalten ist. Andernfalls wird `false` zurückgegeben. Der Suchalgorithmus soll **iterativ** sein, d. h. die Elemente sollen in einer Schleife gesucht werden. Schreiben Sie hierfür mindestens drei Unit-Tests.

3 Aufgabe: `contains()` rekursiv in `ListNode`

Die Datenstruktur `ListNode` ist selbstähnlich, da der Rest (Tail) einer Liste selbst wieder eine Liste ist. Deshalb liegt es nahe, die Methode `boolean contains(E e)` in rekursiver Form zu entwickeln. Implementieren Sie diese Methode `contains` für `ListNode`. `contains` soll die gleichen Eigenschaften haben wie `containsIter`. Schreiben Sie hierfür mindestens drei Unit-Tests. **Wichtig**: Rekursiv bedeutet hier, dass Sie nur lokale Variablen nutzen sollen und Schleifen nur über Rekursion erreichen können. Nutzt Ihre Lösung z. B. normale Schleifen, gibt es Punktabzug.

4 Aufgabe: Gleichheit von Binärbäumen

Ein binärer Baum (kein **Such**baum) kann durch die Klasse `TreeNode` im Package `de.hsmannheim.inf.pr2.ads` realisiert werden, die Verweise auf auf den linken und rechten Nachfolgerknoten hat. Entwerfen Sie eine Methode `boolean equalStructure(TreeNode<E> otherNode)` in der Klasse `TreeNode`, die einen binären, geordneten Baum mit einem anderen auf Gleichheit untersucht und `true` zurückgibt,

wenn beide Bäume identisch sind und `false`, wenn nicht. Binäre Bäume sind gleich, wenn sie übereinandergelegt gleich aussehen – d. h. die Reihenfolge der Kinder spielt eine Rolle. Beachten Sie, dass ein Teilbaum auch leer sein kann. Beispiele für Vergleiche sind in Abb. 1 zu sehen. Es ist nicht erlaubt, dass die Gleichheit durch Vergleich der Textdarstellung (z. B. `toString()`) von Bäumen verglichen wird. Der Algorithmus soll die Bäume knotenweise vergleichen und sofort beendet werden, wenn Unterschiede gefunden wurden.

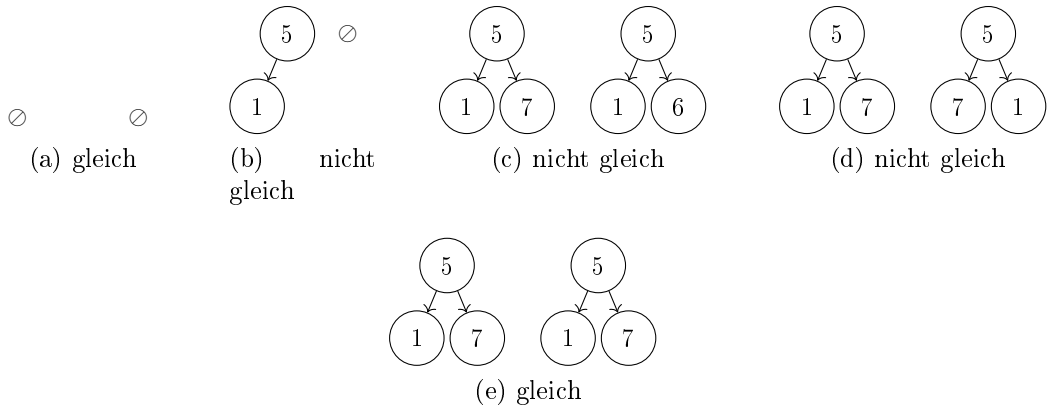

Abbildung 1: Beispiele für Gleichheit von Bäumen. ⊘ ist der leere Baum.

Beschreiben Sie zunächst nur die Idee Ihres Algorithmus in eigenen Worten (ca. 1/3 Seite). Erläutern Sie, wie Sie diese Aufgabe algorithmisch lösen wollen und welche Ansätze Sie wählen. Ein Pseudocode ist nicht verlangt, da die Implementierung in Aufgabe 5 dem Pseudocode sehr ähnlich wäre.

5 Aufgabe: `equalStructure()` in `TreeNode`

Implementieren Sie die Methode `boolean equalStructure(TreeNode<E> otherNode)`, die in Aufgabe 4 eingeführt wurde, und weitere Hilfsmethoden, falls nötig. Erzeugen Sie Testfälle für verschiedene Bäume, so dass damit mindestens fünf repräsentative Vergleichsmöglichkeiten getestet werden können.

6 Aufgabe: `isEmpty()` in `SearchTree`

In der Klasse `de.hsmannheim.inf.pr2.ads.SearchTree` wird ein binärer Suchbaum vorgestellt, der noch nicht zu Ende programmiert ist. Nutzen Sie diese Klasse `SearchTree` (sowie `SearchTreeNode`, wenn sinnvoll) und vervollständigen Sie die Methode `boolean isEmpty()`. Diese Methode gibt `true` zurück, wenn der Suchbaum kein Element enthält oder `false`, wenn er mindestens ein Element enthält. Zeigen Sie die Funktionalität anhand einiger Unit-Tests.

7 Aufgabe: `clear()` in `SearchTree`

Programmieren Sie in der Klasse `SearchTree`, die in Aufgabe 6 eingeführt wurde, die Methode `void clear()` zu Ende und demonstrieren Sie die Funktionen anhand einiger Unit-Tests. `clear()` löscht alle Elemente aus dem Suchbaum.

8 Aufgabe: `height()` in `SearchTree`

Programmieren Sie in `SearchTree` (vgl. Aufgabe 6) die Methode `int height()` zu Ende und demonstrieren Sie die Funktionen anhand einiger Unit-Tests. `height()` liefert die Höhe des Baums zurück.

9 Aufgabe: `contains()` rekursiv in `SearchTree`

In der Klasse `SearchTree` (vgl. Aufgabe 6) gibt es eine Implementierung der Methode `boolean contains(E e)`, die überprüft, ob ein Elemente in dem Suchbaum gespeichert ist. Diese Implementierung ist iterativ. Ändern Sie diese Methode so um, dass die Suche rekursiv erfolgt und demonstrieren Sie die Funktion anhand einiger Unit-Tests. **Wichtig!** Bitte die Hinweise zur Rekursion in Aufgabe 3 beachten.

10 Aufgabe: Live-Testat

Sie bekommen die Aufgabe während des Testats gestellt.