TU Kaiserslautern – FB Informatik
Dr. Patrick Michel

14.01.20

Algorithmen und Datenstrukturen (WS 2019)

Aufgabenblatt 9

zu bearbeiten bis: 20.01.20 - 22.01.20

Aufgabe 9.1 (Bäume - Theorie)
Wir beschäftigen uns mit Binärbäumen im Sinne der `BinTree` Definition aus der Vorlesung. Als Werte sollen die `Node` Objekte dabei `Integer` enthalten.

- Zeichnen Sie einen *vollständigen* Binärbaum der Höhe 4; Wieviele Elemente enthält der Baum? Wieviele Element kämen auf Ebene 5 hinzu? Wieviele auf Ebene 6?
- Markieren Sie Knoten des Baums *fortlaufend* in **level-order**, *left-to-right*, angefangen bei Markierung 1 für die Wurzel. Sie markieren also Ebene für Ebene, von links nach rechts.
- Geben Sie jeweils die Sequenz von Knotenmarkierungen an, die entsteht, wenn Sie den Baum auf folgende Arten durchlaufen: (a) **pre-order**, (b) **in-order**, (c) **post-order**, (d) **level-order**, *left-to-right*, (e) **level-order**, *right-to-left*.

Aufgabe 9.2 (Iteratoren - Praxis)
Den abstrakten Datentypen wurde ein neuer Typ `Collection` hinzugefügt, den alle anderen abstrakten Datentypen erweitern. Dieser enthält lediglich drei Anforderungen: Eine `Collection` hat eine Größe (`size`), man kann testen ob sie leer ist (`isEmpty`) und man kann sie durchlaufen (`iterator`). Die letzte Anforderung ensteht dadurch, dass `Collection` den Typ `Iterable` erweitert.
Dies bedeutet, dass alle Ihre Implementierungen die Funktionen nun implementieren müssen! Die Methoden `size` und `isEmpty` waren in den bisherigen ADTs schon vorhanden, dafür müssen Sie also nichts tun.

- Bringen Sie Ihre Interface Definitionen im Package `exercise.adt` auf den neusten Stand.
- Ergänzen Sie die fehlende Methode `iterator()` des `Iterable<T>` Interface für die Klasse `ArrayList`. Erstellen Sie dazu eine lokale, innere Klasse (oder alternativ eine anonyme Implementierung) des `Iterator<T>` Interface. Implementieren Sie neben den beiden Methoden `hasNext()` und `next()` auch die Methode `remove()`.
- Welche Komplexität haben die drei Methoden jeweils?
- Ergänzen Sie nun die `iterator()` Methode für die Klasse `DoubleLinkedList` und implementieren sie die gleichen drei Methoden im passenden `Iterator`.
- Welche Komplexität haben die drei Methoden jeweils?

Hinweis: Damit Ihr Projekt wieder kompiliert, können Sie in allen Typen die *iterator()* Methode trivial implementieren, z.B. durch `return null`.

Aufgabe 9.3 (Bäume - Praxis)
Als Ausgangspunkt ist eine `BinTree` Implementierung gegeben, deren `add` Methode die Elemente in **level-order**, *left-to-right* in den Baum einfügt.

- Vervollständigen Sie die Implementierung des `Collection` Interface für `BinTree`:
  - Implementieren Sie die Methoden `size` und dann `isEmpty`, achten Sie bei letzterer darauf, dass die Komplexität besser als die von `size` ist! Welche Komplexität haben die beiden Methoden?
  - Schauen Sie sich die Implementierung der `iterator` Methode und die `Iterator` Implementierung an, welche einen **in-order** Durchlauf macht. Um die Implementierung zu vervollständigen müssen Sie nur noch eine passende Implementierung für den `Stack` wählen.
- Implementieren Sie die Methode `flip`, die einen Baum horizontal spiegelt, indem sie alle `left` und `right` Referenzen aller Knoten vertauscht.
- Implementieren Sie einen **pre-order** `Iterator` für `BinTree`.
- Implementieren Sie einen **level-order**, *left-to-right* `Iterator` für `BinTree`.
- Implementieren Sie einen **post-order** `Iterator` für `BinTree`.

Hinweis: Die Implementierungen der drei Iteratoren sind in ihrer Schwierigkeit von einfach nach schwer sortiert! **pre-order** und **level-order** sind einfacher als **in-order**, **post-order** etwas komplizierter. Überlegen Sie sich jeweils welche abstrakte Datenstruktur die passende für den aktuellen Iterator ist! Sie müssen jeweils nur `hasNext` und `next` implementieren.

Hinweis: Zur Abgabe im System müssen Sie neben `BinTree` selbst natürlich auch die Implementierungen von abstrakten Datentypen hochladen, die sie verwenden!

- Welche Komplexität haben bei den Iteratoren jeweils die `hasNext` und `next` Methoden. Unterscheiden Sie dabei, wenn sinnvoll, auch die *average* und *worst*-case Komplexität.
- Welche Speicherkomplexität haben die Iteratoren?