technische universität dortmund

Fakultät für Informatik
Lehrstuhl 11

Mart Hagedoorn
Guangping Li

Sommersemester 2024
24. Mai

DAP2 Praktikum – Blatt 8

Abgabe: KW 23

Studienleistung

- Zum Bestehen des Praktikums muss jeder Teilnehmer*innen die folgenden Leistungen erbringen:
  - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
  - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
- Im Krankheitsfall kann ein Testat bei Vorlage eines Attests in der folgenden Woche nachgeholt werden.
- Wenn ein Praktikumstermin auf einen Feiertag fällt, müssen Sie sich an einem beliebigen anderen Praktikumstermin in der gleichen Woche testieren lassen.
- **Hinweis:** Notieren Sie sich Ihre Punkte nach jedem Testat! Dies dient der eigenen Kontrolle. (Ihr Punktestand kann Ihnen während des Semesters nicht genannt werden.)

Wichtige Information (im Moodle verfügbar)

- Beachten Sie die Erklärung des **Ablaufs (Blatt A)**.
- Beachten Sie die **Regeln und Hinweise (Blatt R)** in der aktuellsten Version!
- Beachten Sie die **Hilfestellungen (Blatt H)** in der aktuellsten Version!

Langaufgabe 8.1: Traversierung von Binärbäumen (4 Punkte)

Ein *Suchbaum* ist ein binärer Baum mit der Eigenschaft, dass der Wert an jedem Knoten v größer oder gleich jedem Wert im linken Teilbaum von v und kleiner oder gleich jedem Wert im rechten Teilbaum von v ist. Um in einen bestehenden Suchbaum einen neuen Wert w einzufügen, soll folgendermaßen vorgegangen werden: Zunächst wird die Wurzel des Baums betrachtet. Ist der Wert hier kleiner als w, so wird w in den rechten Teilbaum eingefügt. Ist der Wert hier größer als w so wird w in den linken Teilbaum eingefügt. Das Vorgehen wird wiederholt, bis ein leerer Teilbaum (zugewiesener Wert == null) erreicht wurde. Dort kann der Wert w gespeichert werden.

(a) Implementieren Sie einen Suchbaum von Ganzzahlen in der Klasse `B8A1` mit:

- Jeweils einem Attribut für den linken und rechten Teilbaum/Nachfolger
- dem Wert des Knotens
- Einem Attribut, welches die Höhe des Teilbaums angibt.
- Einem Konstruktor `public B8A1(int[] array)`, der ein Array von Integern erwartet, um aus diesen Zahlen einen Suchbaum aufzubauen. Die Zahlen sollen dabei exakt nach der Reihenfolge im übergebenen Array mit Hilfe der Methode `public void add(int value)` eingefügt werden.
- Eine Methode `public void add(int value)`, welche den übergebenen Integer an die korrekte Position im Baum einsortiert.
- `public void inOrder()`, die den Suchbaum in in-order traversiert und ausgibt.
- `public void preOrder()`, die den Suchbaum in pre-order traversiert und ausgibt.
- `public void postOrder()`, die den Suchbaum in post-order traversiert und ausgibt.

Weitere Informationen zu die Preorder-Traversierung und Postorder-Traversierung im Internet, z.B.:
`https://geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/`

(b) Abschließend sollen Sie eine geeignete `main`-Methode schreiben, sodass Ihr Porgramm eine Ganzzahl n und eine Liste von Ganzzahlen $a_0, a_1, \ldots, a_{n-1}$ via Standard-In erhält, Die Werte sollen jeweils in ein Integer-Array geladen und in einen binären Suchbaum eingefügt werden. Geben Sie die Höhe des jeweiligen Baums aus, nachdem alle Elemente eingefügt wurden. Es soll der Baum in in-, pre- und post-order ausgegeben werden.

Beispielausgaben des Programms:

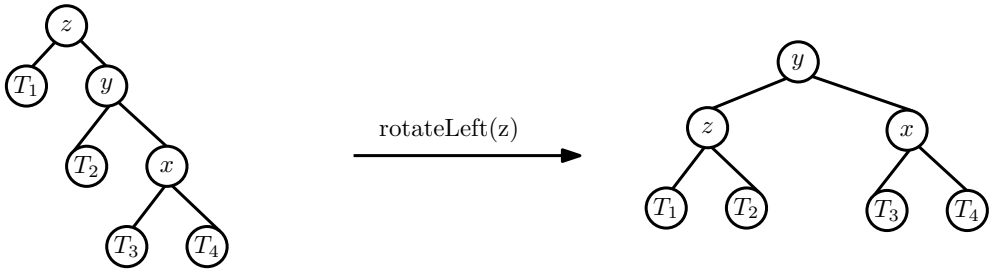
```
java B8A1.java <Enter>
11 12 3 11 40 -10 17 -5 4 5 13 100 <Enter>
Input Array: [12, 3, 11, 40, -10, 17, -5, 4, 5, 13, 100]
Hoehe des Baums: 5
In-Order Traversierung:
-10 -5 3 4 5 11 12 13 17 40 100
Pre-Order Traversierung:
12 3 -10 -5 11 4 5 40 17 13 100
Post-Order Traversierung:
-5 -10 5 4 11 3 13 17 100 40 12
```

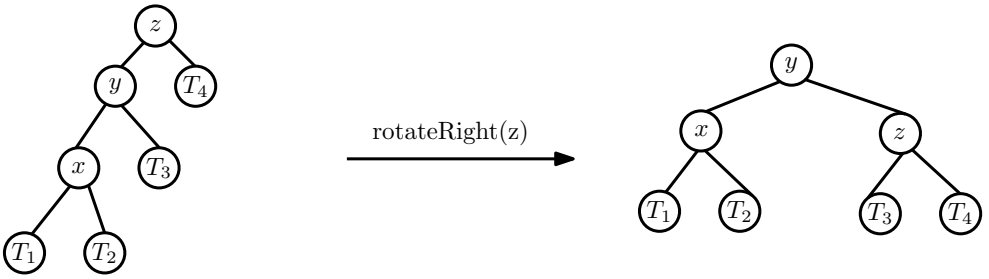
Vergleichen Sie die Höhe der Bäume für verschiedene Eingabebeispiele. Bei welcher Eingabe ist der Unterschied am größten?

Langaufgabe 8.2: Rot-Schwarz-Bäume (4 Punkte)

Da Laufzeiten für Suche, Einfügen und Löschen in binären Suchbäumen im wesentlichen durch Baumtiefe bestimmt werden, sollte der erstellte Baum für eine effiziente Nutzung der Datenstruktur eine möglichst geringe Tiefe ausweisen. In der Vorlesung haben Sie Rot-Schwarz-Bäume kennengelernt. Rot-Schwarz-Bäume verwenden eine besondere Art der Balancierung , um sicherzustellen, dass die Höhe des Baums logarithmisch zur Anzahl der Knoten bleibt.

- Ergänzen Sie die Klasse `B8A2` für Rot-Schwarz-Baum von Ganzzahlen mit folgenden Methoden:
  - Eine Methode `public void add(int value)`, welche den übergebenen Integer an die korrekte Position im Baum einsortiert und den Baum nach der Einfüge-Operation "balanciert" um die Rot-Schwarz-Baum Eigenschaft sicherzustellen. Verwenden Sie dazu die in der Vorlesung vorgestellten Algorithmen `RBInsertFixup`. Geben Sie Das eingefügte Element aus. Falls ein Teilbaum balanciert werden muss, geben Sie die durchgeführten Rotationsoperationen aus (Siehe Ausgabebeispiel). Implementieren Sie die folgende Hilfsmethode:
    (a) Die Hilfsmethode `private void insert(Node root, Node newNode)`, welche die Einfügeoperation für den Baum macht.
    (b) Die Hilfsmethode `private void fixInsert(Node node)`, welche den Baum balanciert nach dem Einfügen eines neuen Knotens.
  - Die Hilfsmethoden `private void rotateLeft()` und `private void rotateRight()` zur Durchführung von Links-Rotation und Rechts-Rotation. Rotationen in einem Rot-Schwarz-Baum werden verwendet, um den Baum nach Einfügeoperationen balanciert zu halten. Es gibt zwei grundlegende Arten von Rotationen: Links-Rotation und Rechts-Rotation.





  - Die Methode `public void inOrderTraversal(Node node)`, die einen in-order Traversal des Baums mit Ausgabe der Farben durchführt.

  - Die Methode `public int height(Node node)` die die Höhe des Baums zurück gibt.

- Abschließend sollen Sie eine geeignete `main`-Methode schreiben. Diese soll dieselben Parameter akzeptieren wie in Aufgabe 8.1 und einen Rot-Schwarz-Baum erstellen. Es sollen Einfüge- und Rotationsoperationen, die Höhe des Baums und die Werte gemäß jeweiligen spezifizierten Traversierung ausgegeben werden.

Beispielausgaben des Programms:

```
Input:  java B8A2.java <Enter>
11 -10 -5 3 4 5 11 12 13 17 40 100 <Enter>
Input Array: [-10, -5, 3, 4, 5, 11, 12, 13, 17, 40, 100]
Fuege -10 in den Rot-Schwarz-Baum ein.
Fuege -5 in den Rot-Schwarz-Baum ein.
Fuehre Links-Rotation durch bei Knoten: -10
Fuege 3 in den Rot-Schwarz-Baum ein.
Fuege 4 in den Rot-Schwarz-Baum ein.
Fuehre Links-Rotation durch bei Knoten: 3
Fuege 5 in den Rot-Schwarz-Baum ein.
Fuege 11 in den Rot-Schwarz-Baum ein.
Fuehre Links-Rotation durch bei Knoten: 5
Fuege 12 in den Rot-Schwarz-Baum ein.
Fuehre Links-Rotation durch bei Knoten: -5
Fuege 13 in den Rot-Schwarz-Baum ein.
Fuehre Links-Rotation durch bei Knoten: 12
Fuege 17 in den Rot-Schwarz-Baum ein.
Fuege 40 in den Rot-Schwarz-Baum ein.
Fuehre Links-Rotation durch bei Knoten: 17
Fuege 100 in den Rot-Schwarz-Baum ein.
Hoehe: 5
In-Order Traversierung: (-10, schwarz) (-5, schwarz) (3, schwarz) (4, schwarz)
(5, schwarz) (11, schwarz) (12, schwarz) (13, rot) (17, rot) (40, schwarz) (100, rot)
```