HSLU Hochschule Luzern

Algorithmen & Datenstrukturen (AD)

Übung SW03: Bäume, Binärbäume und Suchbäume

Themen:	Bäume allgemein, binäre Bäume, Operationen auf binären Bäumen, Modellierung und Implementation, Generics, Schnittstellen.
Zeitbedarf:	ca. 180 Minuten
Wichtig:	Ziel ist es, exemplarische Datenstrukturen selbst zu implementieren, um deren Aufbau zu verstehen und das algorithmische Denken zu üben. In der Praxis vermeidet man eigene Implementationen und verwendet stattdessen möglichst bereits vorhandene und erprobte Implementationen!

Roland Gisler / Ingmar Baetge, Version 1.6.0 (HS25)

1 Bäume (allgemein): Begriffe und Merkmale (ca. 15')

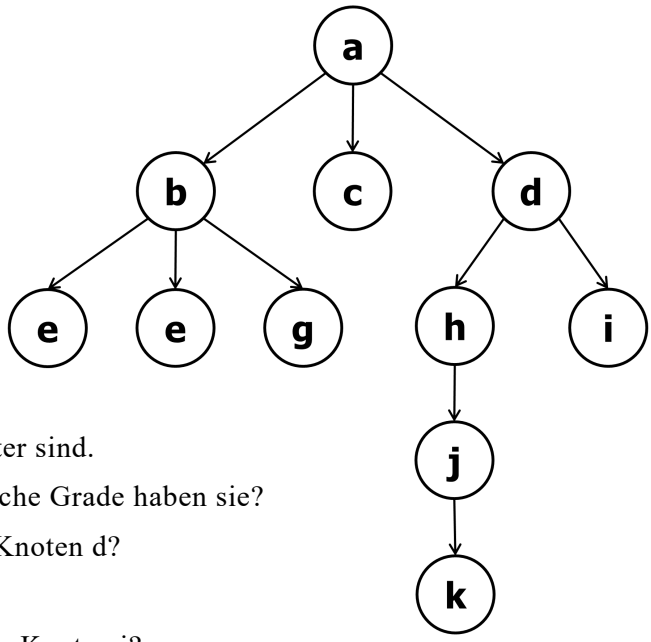
1.1 Lernziele

- Die verschiedenen Bezeichnungen und Begriffe zu Bäumen kennen.

1.2 Grundlagen

Diese Aufgabe basiert auf dem Input **D21**. In dieser Aufgabe schreiben Sie keinen Code.

Gegeben ist der folgende Baum:



1.3 Aufgaben

a.) Welcher Knoten ist die Wurzel?

b.) Listen Sie alle Knoten auf, welche Blätter sind.

c.) Listen Sie alle inneren Knoten auf. Welche Grade haben sie?

d.) Auf welchem Niveau befindet sich der Knoten d?

e.) Wie viele Konten gibt es auf Niveau 2?

f.) Welche Tiefe (Länge des Pfades) hat der Knoten j?

g.) Wie hoch ist dieser Baum?

h.) Können Sie eine Aussage zur Ordnung dieses Baumes machen?

i.) Ist dieser Baum ausgeglichen? Begründen Sie ihre Antwort.

2 Binäre Bäume: Begriffe und Merkmale (ca. 15')

2.1 Lernziele

- Spezifische Grenzen und Regeln von binären (Such-)Bäumen kennen.

2.2 Grundlagen

Diese Aufgabe basiert auf dem Input **D22**. In dieser Aufgabe schreiben Sie keinen Code.

2.3 Aufgaben

a.) Wie viele Wurzeln hat ein binärer Baum?

b.) Was hat ein binärer Baum für eine Ordnung?

c.) Wie viele Niveaus benötigt ein binärer Baum im besten Fall für 31 Knoten?
Wie berechnet sich dieser Wert?

d.) Wie viele Niveaus ergeben sich im schlechtesten Fall?
Wie berechnet sich dieser Wert?

e.) Wie viele Knoten kann ein Binärbaum maximal je für die Niveaus 0 bis 5 enthalten?

f.) Können Sie einen binären Baum mit insgesamt fünf Knoten entwerfen, welcher voll ist?

g.) Wie gross ist der Aufwand für die Suche nach einem Element in einem binären Baum im jeweils Besten und jeweils schlechtesten Fall? Welche Szenarien sind das?

3 Binäre Suchbäume: Suchen, Einfügen und Traversieren (ca. 30')

3.1 Lernziele

- Verstehen, wie ein binärer Suchbaum aufgebaut ist.
- Einfügen von Elementen in einen binären Suchbaum.
- Traversieren von Bäumen.

3.2 Grundlagen

Diese Aufgabe basiert auf dem Input **D22**. In dieser Aufgabe schreiben Sie keinen Code.

3.3 Aufgaben

a.) Wir wollen 15 Datenelemente in einen binären Baum einfügen. Wie viele Niveaus benötigen wir dafür im **besten** Fall? Welchen Füllgrad hat dieser Baum somit?

b.) Fügen Sie die folgenden 15 Schlüsselelemente schrittweise und von Hand in einen leeren binären Baum ein:

```
H  D  L  B  F  J  N  A  C  E  G  I  K  M  O
```

c.) Suchen Sie im Baum von b) nach den folgenden Elementen: `N`, `K` und `O`.

d.) Fügen Sie die folgenden 8 Schlüsselelemente ein einen neuen, leeren binären Baum ein:

```
G  H  B  F  E  A  D  C
```

e.) Wie sieht ein binärer Baum aus, in welchen die folgenden Elemente eingefügt werden, und welcher Datenstruktur entspricht er somit?

```
A  B  C  D  E  F  G  H
```

f.) Sie haben verschiedene Arten wie ein Baum traversiert werden kann kennen gelernt. In welcher Art müssen Sie binäre Bäume traversieren, damit sie der Sortierung folgen?

g.) Überprüfen Sie ihre Antwort von f), indem Sie die Bäume der Teilaufgaben b), d) und e) entsprechend traversieren. Formulieren Sie dazu (in Pseudocode) eine Funktion mit einem **rekursiven** Algorithmus.
Tipp: Die Funktion nimmt einen Knoten als formalen Parameter entgegen, der initiale Aufruf bekommt die Wurzel des Baumes als aktuellen Parameter!

4 Modellierung: Datenstruktur für binären Suchbaum (ca. 30')

4.1 Lernziele

- Objektorientiertes Modellieren eines binären Suchbaumes.
- Struktur eines binären Buchbaumes verstehen.
- Visualisierung mit UML.

4.2 Grundlagen

Diese Aufgabe basiert auf dem Input **D22**. In dieser Aufgabe schreiben Sie (noch) keinen Code, sondern entwerfen ein objektorientiertes Modell.

Wir wollen einen binären Baum modellieren, in welchen wir Datenelemente ablegen können. Je nach Vorkenntnissen können Sie einen einfachen elementaren Datentyp (z.B. `int`) verwenden, oder einen Klassentyp (z.B. `String` oder `Object` etc.)

Der Baum soll **keine** doppelten Werte enthalten dürfen. Zur Darstellung verwenden Sie ein UML-Klassendiagramm (Handskizze reicht völlig aus).

4.3 Aufgaben

a.) Entwerfen Sie als erstes die Schnittstelle für den Baum, dann die Implementationsklasse. Welche Attribute und Methoden könnten zusätzlich zur Schnittstelle in der Implementation noch nützlich sein?

b.) Ergänzen Sie ihren Entwurf nun mit einer Klasse, welche die Knoten repräsentiert. Welche Attribute und Methoden sind auf dieser Klasse sinnvoll bzw. notwendig?

c.) Für verschiedene Operationen auf der Baumstruktur müssen wir für die Datenelemente entscheiden können, ob diese im Vergleich mit einem anderen Datenelement kleiner, gleich oder grösser sind. Überlegen Sie sich, welche Klassen dazu welche weiteren Methoden überschreiben und/oder welche Interfaces implementieren sollten.

d.) Überprüfen Sie kritisch ihre Beziehungen. Achten Sie auf die Richtung und geben Sie wenn immer möglich eine Kardinalität (typisch: 0, 1, 0..1, 0..n, 1..n etc.) an.

5 Implementation eines binären Suchbaumes (ca. 90')

5.1 Lernziele

- Exemplarische Implementation eines binären Baumes.
- Binäre Suche und Traversierung.

5.2 Grundlagen

Diese Aufgabe basiert auf dem Input **D22** und der Aufgabe **3** und **4**.

5.3 Aufgaben

a.) Erstellen Sie gemäss ihrem Entwurf von Aufgabe **4** die grundlegenden Klassen und implementieren Sie vorerst noch **keine** der Methoden (leer lassen).

b.) Entwerfen Sie (auf «Papier»!) ein Beispiel für einem mit mindestens 10 eindeutigen, geordneten Elementen befüllten binären Suchbaum. Sie dürfen die Datentypen bzw. Inhalte selbst wählen, aber achten Sie darauf, dass ihr Baum **alle möglichen Szenarien** enthält: (Eltern-)Knoten ohne, mit linkem, mit rechtem oder mit beiden Kindern. Erstellen Sie eine Skizze ihres Baumes

c.) Konstruieren Sie den in b) geplanten Baum **nun direkt programmiert**, also z.B. im Konstruktor des Baumes selbst. Das bedeutet Sie erstellen alle Knoten und verknüpfen diese Schritt für Schritt miteinander, in dem Sie die jeweiligen Referenzen auf die Kinder setzen. Dadurch haben wir vorerst einen statischen, aber fixfertigen Baum, der unseren Vorstellungen entspricht.

d.) Implementieren Sie auf dem Baum nun als erstes die `search()`-Methode, welche im Baum ein bestimmtes Element sucht. Überlegen Sie sich dazu nochmal den Algorithmus und notieren Sie sich ihn ggf. vorgängig als Pseudocode.

Hinweis: Verwenden Sie Logging, um möglichst gut nachvollziehen zu können, wie einzelne Knoten besucht, überprüft (verglichen) und dann über den weiteren Weg entschieden wird. Das Überschreiben der `toString()`-Methode in den Knoten (und in den Datenelementen) leistet Ihnen dabei sicher gute Dienste.

e.) Überlegen Sie sich nun, wie Sie neue Elemente einfügen können. Dazu müssen Sie zuerst die geeignete Position finden, und dann einen neuen Knoten an der richtigen Stelle einfügen. Auch hier kann es eine Hilfe sein, vorgängig den Ablauf als Pseudocode zu formulieren.

f.) In der Aufgabe 3.f.) haben Sie bereits einen rekursiven Algorithmus zur Traversierung des Baumes in der «Inorder»-Semantik entworfen. Implementieren Sie nun den Algorithmus und testen Sie ihn aus! Zur Vereinfachung geben Sie die Datenelemente einfach auf der Konsole aus.

5.4 Optionale Aufgaben

g.) Versuchen Sie auch die Preorder -und die Postorder-Traversierung zu implementieren.

h.) Funktionieren ihre Implementationen von d), e), f) und g) auch auf einem leeren Baum oder auf einem Baum mit nur einem Element? Korrigieren Sie wenn nötig!

i.) Versuchen Sie nun auch Elemente aus dem Suchbaum zu entfernen. Zumindest der erste, einfachste Fall (Knoten hat keine Kinder) sollte Ihnen eigentlich leicht gelingen.

j.) Schaffen Sie auch das Entfernen von Elementen mit einem (zweiter Fall) und vielleicht sogar mit zwei (dritter Fall) Kindern? Respekt! Das ist aufwändig!

6 Mathematische Ausdrücke als Bäume (ca. 60')

6.1 Lernziele

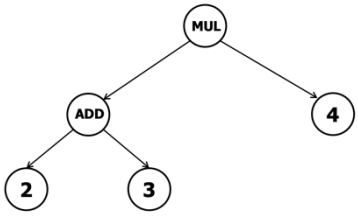
- Weitere Anwendung von Bäumen und Baumstrukturen

6.2 Grundlagen

Diese Aufgabe basiert auf dem Input **D22** und der Aufgabe **5**.

6.3 Aufgaben

Im Folgenden wollen wir Bäume nutzen, um einfache mathematische Ausdrücke zu speichern. Der Ausdruck (2 + 3) ∗ 4 kann z.B. wie folgt als Baum dargestellt werden:



Dazu benötigen wir verschiedene Knotentypen, die am besten als Unterklassen einer allgemeinen **`Node`**-Klasse (oder Interface) implementiert werden:

- ein **`Mul`**-Knoten repräsentiert eine Multiplikation des linken und rechten Teilbaums
- ein **`Add`**-Knoten repräsentiert eine Addition des linken und rechten Teilbaums
- ein weiterer Knotentyp (z.B. **`Number`**) repräsentiert Integer-Werte, dies sind Blattknoten
- Optional: weitere Knotentypen für Subtraktion oder Division

a.) Zeichnen Sie einen Baum, der den Ausdruck (4 + 5) ∗ (2 + 3) repräsentiert

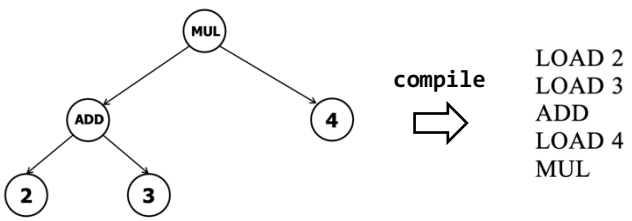
b.) Implementieren Sie Klassen und Konstruktoren, so dass Sie mit folgendem Code einen Ausdrucksbaum erzeugen können:

```
Node n = new Mul(
    new Add(new Number(2), new Number(3)),
    new Number(4)
);
```

c.) Implementieren Sie eine rekursive Funktion **`String toString(Node node)`**, welche einen Ausdrucksbaum in einer Zeichenkette umwandelt. Tipp: Wandeln Sie dazu zunächst den linken und rechten Teilbaum in eine Zeichenkette um (sofern vorhanden), und erzeugen Sie dann das Ergebnis für den aktuellen Knoten. Das Ergebnis für das Beispiel könnte **`((2+3)*4)`** lauten.

d.) Implementieren Sie eine rekursive Funktion **`int eval(Node node)`**, welche den Wert eines Ausdrucksbaums berechnet (Tipp: Post-Order) und testen Sie die Funktion mit den beiden Beispielbäumen.

e.) Schreiben Sie eine rekursive Funktion **`List<String> compile(Node node)`**, welche Ihren Baum in ein Programm für eine Stackmaschine aus der letzten Übung übersetzt (Tipp: Post-Order) und testen Sie die Funktion mit den beiden Beispielbäumen.



7 Optional: Binärer Suchbaum mit Hashcode als Schlüssel (ca. 30')

7.1 Lernziele

- Praktische Anwendung von Hashcodes.
- Implementation eines binären Suchbaumes mit Schlüssel.

7.2 Grundlagen

Diese Aufgabe basiert auf dem Input **D22** und der Aufgabe **5**.

7.3 Aufgaben

Wichtiger Hinweis: Lösen Sie diese Aufgabe am besten auf einer **Kopie** Ihrer Lösung von Aufgabe 5, so dass Sie deren Lösung danach auch noch haben!

a.) Erweitern bzw. refaktorieren Sie ihre Implementation des binären Suchbaumes von Aufgabe **5** so, dass neben dem Datenelement auch ein `int`-Hashwert (wie ihn die Methode `hashCode()` liefert) abgespeichert werden kann.

b.) Beim Einfügen von Elementen berechnen Sie den Hashwert des Elementes und prüfen (suchen) im Binärbaum, ob dieser Hash schon enthalten ist. Wenn nein, fügen Sie das Element in den Baum ein.

c.) Die bisherige Lösung dieser Aufgabe setzt perfekte Hashwerte voraus. Das können wir aber nicht immer erfüllen. Wie sähe eine Lösung aus, welche auch mit Hashwertkollisionen umgehen kann? Es reicht vorerst eine konzeptionelle Lösung, Sie dürfen es aber auch ausprobieren.

d.) Refaktorieren Sie Ihren Baum als generisch typisierbare Datenstruktur.