Algorithmen & Datenstrukturen (AD)

# Übung: Bäume, Binärbäume und Suchbäume (D2)

Themen: Bäume allgemein, binäre Bäume, Operationen auf binären Bäumen, Modellierung und Implementation, Generics, Schnittstellen.

Zeitbedarf: ca. 180min.

Wichtig: Ziel dieser Aufgaben ist es, ausgewählte Datenstrukturen mindestens teilweise selber zu implementieren, um deren Funktion und Aufbau besser zu verstehen sowie das algorithmische Denken und auch das Programmieren zu üben.
In der Praxis vermeidet man eigene Implementationen und verwendet stattdessen möglichst die bereits vorhandenen, erprobten und optimierten Implementationen.

Roland Gisler, Version 1.0 (FS 2017)

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

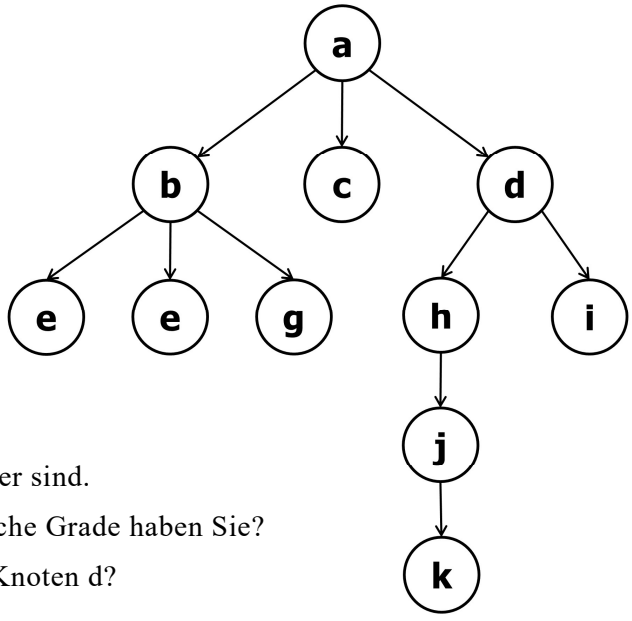
### 1.1 Lernziele

- 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.

Als Grundlage dient 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 3?

f.) Welche Tiefe 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 (Such-)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 1 bis 6 enthalten?

f.) Skizzieren Sie einen binären Baum mit insgesamt vier Knoten, welcher auf dem zweiten Niveau voll ist.

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

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

## 3.1 Lernziele

- Verstehen, wie ein binärer Baum aufgebaut ist.
- Einfügen von Elementen in einen binären Baum.
- 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 Datenelemente schrittweise 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 Datenelemente 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 Baum (ca. 30')

## 4.1 Lernziele

- Objektorientiertes Modellieren eines binären Baumes.
- Struktur eines binären Baumes 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 beliebige Datenelemente(-typen) ablegen können. Der Baum soll keine doppelten Werte enthalten. 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 4.

## 5.3 Aufgaben

a.) Erstellen Sie die grundlegenden Klassen und implementieren sie vorerst nur die einfachen Methoden gemäss Ihrem Entwurf von Aufgabe 4. Die schwierigeren Methoden für das Einfügen und Suchen lassen Sie im Moment noch leer.

b.) Entwerfen Sie ein eigenes Beispiel für einem mit mindestens 10 eindeutigen, geordneten Elementen befüllten binären Suchbaum.
Sie dürfen die Datentypen bzw. Inhalte selber wählen. Achten Sie darauf, dass Ihr Baum alle möglichen Szenarien enthält: (Vater-)Knoten ohne, mit linkem, mit rechtem oder mit beiden Kindern. Am Besen machen Sie sich dazu eine Skizze Ihres Baumes.

c.) Erstellen Sie den in b entworfenen Baum nun zuerst direkt programmiert, also z.B. im Konstruktor des Baumes selber. 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 Baum.

d.) Implementieren Sie auf dem Baum nun 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 einmal mehr grosse 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 3f 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 aus.

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

h.) Funktionieren Ihre Implementationen von 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 gelingen.
Optional: Schaffen Sie auch den zweiten und vielleicht sogar den dritten Fall?

# 6 Binärer Suchbaum mit Hashcode als Schlüssel (ca. 30')

## 6.1 Lernziele

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

## 6.2 Grundlagen

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

## 6.3 Aufgaben

a.) Erweitern Sie Ihre Implementation des binären Suchbaumes von Aufgabe 5 so, dass neben dem Datenelement auch ein **`int`**-Hashwert (wie ihn die Methode **`hashCode()`** liefert) abspeichern 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.) Refaktorieren Sie Ihren Baum so, dass Sie **`Allokations`**-Objekte (aus der Übung E0) in den Baum einfügen können. Funktioniert das?

d.) 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.

# 7 Optional: Generische implementierte binäre (Such-)Bäume

## 7.1 Lernziele

- Implementation von generischen Datenstrukturen.

## 7.2 Grundlagen

Diese Aufgabe basiert auf den Aufgabe 6 und 7.

## 7.3 Aufgaben

a.) Implementieren Sie den Suchbaum der Aufgaben 6 bzw. 7 als generisch typisierbare Datenstruktur und testen Sie Ihre Implementation.

b.) Testen Sie Ihre Implementation, indem Sie Allokations-Objekte in den Baumeinfügen. Als Schlüssel (und Hashwert) kann direkt die Startadresse des Blockes dienen.

c.) Überlegen Sie: Wäre es auch möglich/sinnvoll (freie) Allokationsblöcke nach deren Grösse geordnet einzufügen?