14.12.15

Team:

5, Marc Kaepke & Constantin Wahl

Aufgabenteilung:

M. Kaepke

Ausarbeitung der Skizze

C. Wahl

Messungen und Auswertung

Gemeinsam

Implementierung des ADTs

Quellenangabe:

- Vorlesungsscript
- Folien
- https://courses.cs.washington.edu/courses/cse373/06sp/handouts/lecture12.pdf (Rechtsrotationsgrafik)

Bearbeitungszeitraum:

Gemeinsam

M. Kaepke

5,5 Stunden: Skizze

0,5 Stunden: Count-Klasse und #sortNum() erweitert

C. Wahl

1 Stunde: Skizze

Aktueller Stand:

Skizze wurde erstellt. Mit der Implementierung wurde bereits begonnen.

Aktualisierung an der Skizze:

Die ADT ist detaillierter beschrieben. Dazu gehört der funktionale und technische Bereich, ebenso die Fehlerbehandlung, Objektmengen und die Operationen an der ADT.

Es wird weiter auf die Wurzel und Knoten eingegangen, das Einfügen bzw. Entfernen, sowie die ggf. damit verbundenen Rotationen um einen balancierten Baum zu erhalten.

14.12.15

Skizze

Package-Struktur:

src.adt.implementations.
- AdtArrayImpl
- AdtAVLBaumImpl
- AdtListImpl

src.adt.interfaces.
- AdtArray
- AdtAVLBaum
- AdtList

src.general.
- NumGenerator
- Count

src.tests.
- AdtAVLBaumTests

NumGenerator:

Die Methoden #sortNum(String, int) und #sortNum(String, int, boolean) werden jeweils um einen weiteren Parameter erweitert, der angibt, ob Duplikate beim Generieren erlaubt sind oder nicht.
#sortNum(String, int) → #sortNum(String, int, boolean)
#sortNum(String, int, boolean) → #sortNum(String, int, boolean, boolean)

ADT-AVL-Baum:

Ein AVL Baum ist ein balancierter binärer Suchbaum. Er ist genau dann balanciert, wenn sich für jeden Knoten die Höhe der zugehörigen Teilbäume um höchstens 1 unterscheidet.

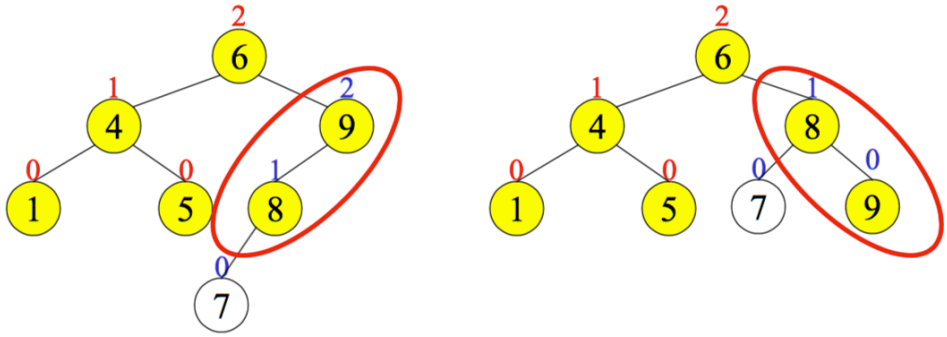
- Beim Erzeugen eines AVLBaums wird der Wurzelknoten mit null initialisiert
- Jedes Element wird beim Einfügen in den AVLBaum in ein AVLKnoten umgewandelt und rekursiv an die richtige Stelle gehängt (referenziert)
    - Überprüfe von diesem Knoten ausgehend bottom-up die AVL-Bedingung (Balance)
    - Führe gegebenenfalls an dem Knoten, an dem die Bedingung verletzt wurde, eine Rebalancierung durch (mittels entsprechender Rotation, siehe unten)
- Zum Löschen eines Knotens;
    - Kopiere (referenziere) den größten Knoten im linken Teilbaum bzw. kleinsten Knoten im rechten Teilbaum auf den zu löschenden Knoten. Lösche nun (rekursiv) diesen Knoten
    - Ist der zu löschende Knoten ein Blatt (= keine Kinder), dann wird der Knoten aus dem AVLBaum entfernt -> Rekursionsabbruch

- Überprüfe von dem Vorgänger (höher hängenden Knoten) des entfernten Blatts ausgehend bottom-up die AVL-Bedingung (Balance)
- Führe gegebenenfalls an dem Knoten, an dem die Bedingung verletzt wurde, eine Rebalancierung durch (mittels entsprechender Rotation, siehe unten)

Balance:

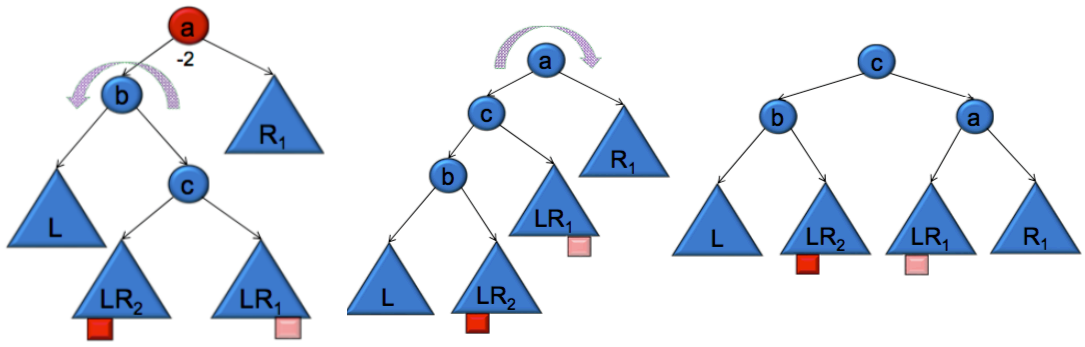
Wenn die Höhendifferenz vom linken und rechten Teilbaum nicht in der Menge {-1, 0, +1} enthalten ist, dann muss rotiert werden. Hierbei gibt es 4 unterschiedliche Situationen:

Links bzw. Rechtsrotation:



In dieser Situation (Rechtsrotation) wurde die 7 als letztes hinzugefügt. Die AVL-Bedingung ist verletzt, da der Teilbaum unterhalb der 2 eine Differenz größer (+/-)1 hat.
Die 9 wird zum rechten Kind von der 8 und rückt selbst rechts unterhalb der 6.
Durch diese Rotation ist die AVL-Bedingung nicht mehr verletzt.
Die Linksrotation funktioniert genauso, nur dass der rechte Teilbaum übergewichtig gegenüber dem linken ist.

Links-Rechts-Rotation bzw. Rechts-Links-Rotation:



In dieser Situation (Links-Rechts-Rotation) bedarf es zwei hintereinander folgende Rotationen. Zunächst eine einfache Linksrotation um b und anschließend eine Rechtsrotation um a.
Die Rechts-Links-Rotation funktioniert nach selbigen Prinzip, nur folgt zunächst eine Rechtsrotation und dann eine Linksrotation.

Funktional (nach außen):

- Die Elemente sind vom Typ „ganze Zahlen“
- Beim Einfügen wird das neue Element gemäß der Sortierung als Knoten in den AVLBaum eingegliedert
    - Alle Knoten im linken Teilbaum < Wurzel
    - Alle Knoten im rechten Teilbaum >= Wurzel
- Nach jeder Einfüge- oder Löschoperation ist der AVLBaum balanciert

Technisch (nach innen):

- Beim Export wird als Zwischendatei eine *.dot Datei erzeugt
- Beim Einfügen oder Löschen eines Knotens muss geprüft werden, ob der Baum nach der Aktion noch balanciert ist, andernfalls muss eine der Rotationsfunktionen aufgerufen werden
- Die ADTAVLBaum ist intern mittels einer Datenstruktur „AVLKnoten“ zu realisieren
    - Ein AVLKnoten hält eine Referenzierung seiner Kinder (links und rechts) und den Integer-Wert, ebenso die Höhe seines Teilbaums

Objektmengen:

- elem (int), adt (AVLBaum), png

Operationen:

- public static AdtAVLBaum create()
    - Erzeugt eine neue Adt AVLBaum
- public boolean isEmpty()
    - Gibt zurück ob der AVLBaum leer ist
    - true => der Baum ist leer (hat keinen Knoten)
    - false => der Baum hat mindestens einen Knoten
- public int high()
    - Gibt die Gesamthöhe des Baums zurück
    - Ein leerer Baum hat die Höhe = 0
    - Ein Baum, nur mit der Wurzel ohne Kinder, hat die Höhe = 1
- public AdtAVLBaum insert(int elem)
    - Fügt das Element in den Baum ein und wird einsortiert
- public long insertRunTime(int elem)
    - Fügt das Element in den Baum ein
    - Misst die Laufzeit in Nanosekunden
- public Count insertCount(int elem)
    - Fügt das Element in den Baum ein
    - Zählt die Links- & Rechtsrotationen und die Lese- & Schreibzugriffe
    - Die Informationen sind im Return-Type Count gespeichert
- public AdtAVLBaum delete(int elem)
    - Entfernt das Element bzw. den Knoten aus dem Baum
- public Count deleteCount(int elem)
    - Entfernt das Element bzw. den Knoten aus dem Baum
    - Zählt die Links- & Rechtsrotationen und die Lese- & Schreibzugriffe
    - Die Informationen sind im Return-Type Count gespeichert
- public boolean print()
    - Exportiert den Baum als *.png Datei

Fehlerbehandlung:

- Beim Einfügen eines ungültigen Knotens (z. B. keine ganze Zahl) wird das Einfügen des Knotens ignoriert.
- Soll ein nicht vorhandenes Element gelöscht werden, so wird die ADT unverändert zurückgegeben.

JUnit:

- Es muss überprüft werden, das Elemente nicht doppelt eingefügt werden können
- Es muss überprüft werden, ob das Entfernen des letzten Knotens dafür sorgt, das die #isEmpty() Methode mit true evaluiert
- Es muss überprüft werden, das jede Einfüge- und Löschoperation die Höhe anpasst
- Es muss überprüft werden, ob die ADT mit großen Datenmengen (bspw. 300 Knoten) einwandfrei funktioniert

Count:

Hält die Zähldaten zu den Links- und Rechtsrotationen und den Lese- und Schreibzugriffen.
Konstruktor:

public Count(int, int, int, int)

Methoden:

Getter-Methoden für Instanzvariablen