**Team**: < 04 >, < Schäfer & Ahmad >

**Aufgabenaufteilung**:

1. <Aufgaben, für die Teammitglied 1 verantwortlich ist>,   
   <Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden>
2. <Aufgaben, für die Teammitglied 2 verantwortlich ist>,   
   <Dateien, die komplett/zum Teil von Teammitglied 2 implementiert/bearbeitet wurden>

**Quellenangaben**: <

>

**Begründung für Codeübernahme**: < Es wurde kein Code übernommen. >

**Bearbeitungszeitraum**: <

Für den Entwurf: 2 Stunden 15.11.2014

1 Stunde 21.11.2014

1 Stunde 25.11.2014

http://users.informatik.haw-hamburg.de/~klauck/AlguDat/AD.pdf

>

**Aktueller Stand**: < Entwurf fertig >

**Änderungen im Entwurf**: < KEINE >

***Implementieren sie die ADT AVL-Baum reskursiv:***

Dieser soll als höhenbalancierter binärer Suchbaum realisiert werden, gemäß der Definition aus der Vorlesung.   
(Siehe Quelle)

***Was ist ein höhenbalancierter binärer Suchbaum?***

Definition AVL-Baum: Ein binärer Suchbaum heißt **AVL-Baum** oder höhenbalancierter Baum, wenn für jeden Knoten *v* gilt, dass sich die Höhe des rechten Teilbaumes ***h*(*Tr*)** von *v* und die Höhe des linken Teilbaumes ***h*(*Tl*)** von *v* um maximal 1 unterscheiden.

*bal(v) = h(Tr) – h(Tl)* ∈ {-1,0,1}

***Es sind folgende Funktionen für die ADT AVL-Baum zu implementieren, die nach Außen geliefert werden!***

1. AVL-Baum Initialisieren
2. Einfügen
3. *Löschen:*  
   „Kopiere den größten Knoten im linken Teilbaum bzw. kleinstem Knoten im rechten Teilbaum auf den zu löschenden Knoten. Lösche nun (rekursiv) diesen Knoten.“  
   Für welche Variante Sie auswählen ist Ihnen überlassen!
4. Linksrotation
5. Rechtsrotation
6. Doppellinksrotation
7. Doppelrechtsrotation

Alle diese Funktionen sollen den modifizierten AVL-Baum zurück liefern!

***Die Signaturen der zu implementierenden Funktionen:***

*%% Erstellt Die Datenstruktur*

*init() ->*

*Funktion haengt genau ein Knoten an einen Baum an.*

*@param avlBaum Tree - Die Datenstruktur des AVL Baumes.*

*@param String Key - Das neu hin zugefuegte Element.*

*@param String File – Pfad zum automatisch generierten Bild*

einfuegen(Tree, Key, File) ->

*Diese Funktion entfernt einen Knoten aus einem Baum.*

*@param avlBaum Tree - Der AVL-Baum auf dem ein Schluessel entfernt werden soll*

*@param Integer Key - Der zu entfernende Schluessel.*

*@param String File – Pfad zum automatisch generierten Bild*

*@return avlBaum - Der modifizierte Baum.*

loeschen(Tree, Key, File) ->

*Diese Funktion implementiert die einfache Linksrotation.*

*@param avlBaum Tree - Der Baum in den rotiert werden soll.*

*@param Integer Key - Der Vertex um den rotiert werden soll.*

*@param String File – Pfad zum automatisch generierten Bild*

*@return Modifizierter AVL-Baum*

linksrotation(Tree, Key, File) ->

*Diese Funktion implementiert die einfache Rechtsrotation.*

*@param avlBaum Tree - Der Baum in den rotiert werden soll.*

*@param Integer Key - Der Vertex um den rotiert werden soll.*

*@param String File – Pfad zum automatisch generierten Bild*

*@return Modifizierter AVL-Baum*

rechtsrotation(Tree, Key, File) ->

*Diese Funktion implementiert die doppelte Linksrotation.*

*@param avlBaum Tree - Der Baum in den rotiert werden soll.*

*@param Integer Key - Der Vertex um den rotiert werden soll.*

*@param String File – Pfad zum automatisch generierten Bild*

*@return Modifizierter AVL-Baum*

doppelLinksrotation(Tree, Key, File) ->

*Diese Funktion implementiert die doppelte Rechtsrotation.*

*@param avlBaum Tree - Der Baum in den rotiert werden soll.*

*@param Integer Key - Der Vertex um den rotiert werden soll.*

*@param String File – Pfad zum automatisch generierten Bild*

*@return Modifizierter AVL-Baum*

doppelRechtsrotation(Tree, Key, File) ->

***ADT AVL-Baum-Struktur:***

[ ExecuteCounter, [RotationLeftCounter, RotationRightCounter], [node, Key, [ChildLeftFromKey, ChildRightFromKey], Height, PredecessorKey] ] , ... ]

***Beschreibung der ADT AVL-Baum***

Pos 1: Counter wie oft die Funktion aufgerufen wurden sind, die nach außen sichtbar sind.  
Warum das ganze? Mit diesen Zusatzpunkt können wir für jeden Schritt eigenes Bild generieren mit der Hilfe von Graphviz.

Pos 2: Counter für die Rotationen links und rechts Example: [2, 1]  
Warum speichern wir das in der ADT?   
Aus Bequemlichkeit, wenn die benötigten Information im Baum liegen, muss man sie nicht außerhalb irgendwie verwalten, ist aber jeden selber überlassen, wie er die Rotationen zählen möchte.

Pos 3..n: Ab dieser Positionen kommen die Nodes, einzelner Node wird unten erläutert.  
Wir haben uns entschieden jeden einzelnen Node separat im Baum zu halten und die Nodes über ID/Key Referenzen mit einander zu verbinden. Dies hätte man auch z.B. verschachteln können, jedoch würde das zu einer enormen Unübersichtlichkeit führen, deshalb haben wir uns dagegen entschieden!

***So ist ein Node aufgebaut:***

Pos 1: Prefix – node/atom  
Dies dient nur der Übersichtlichkeit, sonst hat es keinen Nutzen.

Pos 2: Der Schluessel/Key.  
An dieser Position wird der Key abgelegt

Pos 3: [Linkes Kind, Rechtes Kind].  
Diese Liste hät die Key’s seiner direkten Nachbars.   
Warum das ganze? Damit wir z.B. beim einfügen von Elementen wissen, welchen Node wir als nächstes und anschauen müssen.

Pos 4: Höhe des Baumes.  
Anhand diesen Parameter können wir kontrollieren wie hoch der Knoten liegt, erster Knoten sprich die Wurzel hat den Wert 1, wenn man den Baum von oben nach unten betrachtet, hätte das Kind von der Wurzel die Höhe 2.

Pos 5: Vorgänger Schlüssel/Key; if predecessor == nil -> this.Node = RootNode .  
Diesen Parameter brauchen wir für die interne Implementierung, hier können wir prüfen wer der Elternknoten eines Knotens/Nodes ist.

**Tipps:**

- Sie können bei aufrufen der 7 Funktionen die nach außen   
geliefert werden, jedes mal ein Bild generieren mit Hilfe von Graphviz, dies dient der besseren Kontrolle, ob der zu implementierende Algorithmus richtig funktioniert.