Beschreibung der Rekursiven Funktionen

INSERT

Die „Insert“ Funktion ist einer der einfachen Funktionen dieser Anwendung. Ein Baum wird immer mit einer Wurzel initialisiert welche den wert „null“ zugewiesen bekommt. Der erste Schritt beim „Insert“ ist es zu prüfen ob der erste Knoten also die Wurzel den Wert „null“ hat somit noch kein Wert in den Baum eingefügt wurde. Sollte dies der Fall sein wird der einzufügende Wert als neuer Knoten in die Wurzel geschrieben. Sollte der Wert des aktuell betrachtenden Knotens nicht „null“ sein und der einzufügende Wert kleiner dem Wert des aktuellen Knotens sein, so wandern wir zu dessen nächsten **linken** Knoten (soweit vorhanden). Sollte der einzufügende Wert größer dem aktuell betrachtenden Knoten sein so wandern wir zu dessen nächsten **rechten** Knoten (soweit vorhanden). Dies wird solange fortgesetzt bis wir entweder an der rechten oder linken Seite eines Knoten (je nach dem einzufügenden Wert) keinen Knoten vorfinden, nun können wir an dieser Stelle einen neuen Knoten mit dem neuen Wert einfügen.

MINKEY, MAXKEY

Diese Funktion ist dafür verantwortlich den kleinsten Wert des Baumes zu finden. Da wir wissen, dass ein Binärer-Baum den kleinsten Wert im am weitest linken Knoten speichert müssen wir nur solange nach links den Baum entlang folgen bis wir an den letzten linken Knoten landen um an den kleinsten Wert zu finden. Dies tun wir wie folgt. Wir starten an der Wurzel des Baumes und prüfen ob sich links von diesem ein weitere Knoten befindet, sollte dies der Fall sein wird dieser Vorgang mit diesem Konten wieder holt bis wir auf einen Konten treffen welcher links von sich keinen Konten mehr hat, dann wissen wir, wir haben den kleinsten Wert gefunden. Dieses Prozedere können wir natürlich auch mit der rechten Seite des Baumes machen um den größten Wert des Baumes zu bekommen.

TOTALKEYNUM

Diese Funktion liefert uns die Summe aller Werte eines Baumes. Auch hier starten wir wieder an der Wurzel des Baumes und nehmen dessen Wert als ersten zu unserer Variablen „sum“ hinzu. Nun überprüfen wir ob es an der linken oder rechten Seite des Knotens einen weiteren Knoten gibt und führen das Prozedere an diesen Knoten fort und addieren deren rückgabewert zu unsere Summe hinzu. Dies tun wir bis alle Knoten besucht wurden und deren Werte zur gesamten Summe hinzuaddiert wurden.

TOTOALNODENUM

Diese Funktion liefert uns die Anzahl aller Knoten zurück. Diese Funktion bedient sich exakt dem gleichen Schema wie die Funktion „TOTALKEYNUM“ mit dem einzigen unterschied, dass nun die Knoten der Funktion TOTALNODENUM übergeben werden. Bei jedem so besuchten Knoten wird die Zahl 1 zu unseren Gesamtanzahl an Knoten hinzuaddiert.

CHECKAVL

Diese Funktion ist dafür verantwortlich zu prüfen, ob es sich bei einem Binären-Baum um einen ausbalancierten AVL-Baum handelt. Ein AVL-Baum ist ein Baum bei welchem jeder Knoten eine maximale Differenz zwischen der Höhe des linke Teilbaumes und der höhe des rechten Teilbaumes von 1 aufweist. Dieser Funktion wird jener Knoten übergeben welcher auf die oben genannte AVL Bedienung geprüft werden soll. Zu Beginn der Funktion werden zwei Variablen „rightMax“ und "leftMax“ jeweils mit 0 initialisiert. Diese Variablen spiegeln zu Ende der Funktion die maximale Höhe des rechten und des linken Teilbaums wieder, wobei nur das Maximum beider Variablen +1 (als Standardhöhe für den aktuellen Knoten) zurückgegeben wird. Sollte der übergebene Knoten einen weiteren Konten an seiner rechten Seite haben so wird dieser Knoten der Funktion CHECKVAL übergeben und dessen Resultat „rightMax“ zugewiesen. Ebenso wird (wenn vorhanden) der linke Teilbaum des Knotens auf diese Weise durchwandert und das Ergebnis der Variable „leftMax“ zugewiesen. Nach dem dies geschehen ist wird die Differenz dieser beiden Variablen eruiert und geprüft ob dieser größer 1 ist und somit eine „AVL-Violation“ hervorruft. Wie schon oben bestrochen handelt es sich bei dem Rückgabewert dieser Funktion um das Maximum von „rightMax“ und „leftMax“ +1 also die Maximale Tiefe des aktuellen Knotens.

AUFWANDSABSCHÄTZUNG

Function Signature	Worst Case	Best Case	AVL Case
public Node insert(**int** key, Node node)	O(n)	O(1)	$O(\log_2(n))$
private int minKey(Node node)	O(n)	O(1)	$O(\log_2(n))$
private int maxKey(Node node)	O(n)	O(1)	$O(\log_2(n))$
private int totalKeyNum(Node node)	O(n)	O(n)	O(n)
private int totalNodeNum(Node node)	O(n)	O(n)	O(n)
public int checkAVL(Node node)	O(n)	O(n)	O(n)

n … Number of Nodes

public Node insert(int key, Node node);

private int minKey(Node node);

private int maxKey(Node node):

Im besten Fall ist ein einseitiger Baum vorhanden und das Einfügen findet auf der anderen Seite statt. D.h. nur 1 Funktionsaufruf und Nodezugriff (root). → O(1)

Im schlechtesten Fall muss der Knoten bei einem einseitigen Baum auf der längeren Seite nach allen anderen Knoten eingefügt werden. → O(n)

Im Fall eines AVL-Baumes sind beide Seiten gleich tief mit einer Verdopplung der Knoten per Ebene. → $O(\log_2(n))$

private int totalKeyNum(Node node);

private int totalNodeNum(Node node;

public int checkAVL(Node node):

Sowohl im besten, im schlechtesten, als auch im AVL Fall wird der ganze Baum traversiert. Dies geschieht rekursiv, sodass die die Funktionen per Knoten 1-Mal aufgerufen werden und die Rückgabewerte von den Blattknoten auf bis zur Wurzel übergeben werden. → O(n)