Levelorder Traversierung, Iterative Tiefensuche

Bei den drei rekursiv implementierbaren Traversierungen wird der Baum zuerst in die Tiefe durchwandert bis hin zu seinen Blättern („Tiefensuche“) - hier noch einmal am Beispiel bei der Preorder-Traversierung:

Bei der Levelorder Traversierung werden auf jedem Niveau des Baums erst alle Knoten besucht, bevor auf das nächste Niveau gewechselt wird, in unserem Beispielbaum ergibt sich damit die Traversierungsreihenfolge: A→B→F→C→D→G→E. Der Algorithmus zur Levelorder Traversierung ist nicht rekursiv.

Iterative Traversierung

Die rekursiven Implementationen der Traversierungen versagen ihren Dienst, wenn die Bäume zu tief werden, da der Call-Stack für die Rekursion nicht beliebig wachsen kann. Bei Java ist seine Größe auf ca. 256kB beschränkt, wenn diese Größe überschritten wird, erhältst du einen *Stack Overflow Error*:

Die einfachste Lösung für dieses Problem ist es, den Stack, in dem man darüber Buch führt, welche Knoten des Baums als nächstes zu bearbeiten sind, selbst zu verwalten.

Rekursiv	**Iterativ**

`anzahl(b: Binaerbaum):`
`  falls b == null:`
`  return 0`
`  sonst:`
`  t1 = anzahl(b.links)`
`  t2 = anzahl(b.rechts)`
`  return 1 + t1 + t2`
	`anzahl(b: Binaerbaum):`
	`  todo = new Stack`
	`  todo.push(b)`
	`  zaehler = 0`
	`  solange todo nicht leer:`
	`  tmp = todo.pop()`
	`  zaehler++`
	`  falls tmp.rechts != null:`
	`  todo.push(tmp.rechts)`
	`  falls tmp.links != null:`
	`  todo.push(tmp.links)`
	`  return zaehler`

[n/a: Keine Treffer]