Übung 5 Abgabe 27.5.2014

Aufgabe 1 – Binäre Suchbäume 18 Punkte

a) Gegeben sei eine Python-Klasse `Node` mit folgender Definition *4 Punkte*

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
```

Alle Knoten eines Baumes sind vom Typ `Node`, ein (Teil-)Baum ist durch seine Wurzel repräsentiert (`None` bezeichnet einen leeren Teilbaum). Implementieren Sie die in der Vorlesung behandelten Funktionen und entsprechende Unit Tests im File `searchtree.py`:

`treeInsert(rootnode,key)`: neuen Schlüssel in den Baum einfügen (falls der Schlüssel bereits enthalten war, soll der Baum nicht verändert werden),

`treeRemove(rootnode,key)`: angegebenen Schlüssel entfernen (falls der Schlüssel nicht vorhanden ist, soll eine Exception ausgelöst werden) ,

`treeSearch(rootnode,key)`: gibt `None` zurück, wenn der Schlüssel nicht im Baum vorhanden ist, oder den `Node`, der diesen Schlüssel enthält.

b) Entwickeln Sie einen Algorithmus, der die Tiefe des Baumes (den maximalen Abstand von der Wurzel zu einem Blatt) bestimmt und implementieren Sie ihn als Funktion `depth=treeDepth(rootnode)`. Konstruieren Sie Bäume mit unterschiedlichem Grad der Ausgeglichenheit (vollständiger Baum, ausgeglichener Baum, Kette und etwas dazwischen) und prüfen Sie als Bestandteil der Unit Tests, dass die Tiefe korrekt bestimmt wird. *4 Punkte*

c) Angenommen, Sie können die Schlüssel in einer selbstgewählten Reihenfolge einfügen. Wie gehen Sie vor, damit der Baum nach dem Einfügen eine möglichst geringe Tiefe hat? *5 Punkte*

d) Beweisen oder widerlegen Sie folgende Aussage: Wenn aus einem Binärbaum erst der Schlüssel X und danach der Schlüssel Y entfernt wird, entsteht der gleiche Baum wie bei umgekehrter Reihenfolge. *5 Punkte*

Aufgabe 2 – Taschenrechner 28 Punkte

Binärbäume (hier: *Syntaxbäume*) eignen sich auch, um mathematische Ausdrücke auszuwerten, die als Zeichenketten der Form "2+5*3" oder "2*4*(3+(4-7)*8)-(1-6)" gegeben sind.
Es gelten die üblichen Rechenregeln (Klammern haben die höchste Priorität, Punktrechnung geht vor Strichrechnung), Zahlen sollen der Einfachheit halber immer einstellig sein. Operationen mit gleicher Priorität werden von links nach rechts ausgewertet (sogenannte *Links-Assoziativität*), damit wie gewohnt 5-2+3 = (5-2)+3 = 6 gilt (und nicht 5-(2+3) = 0, was bei Auswertung von rechts herauskäme). Steht jedoch rechts von einer Zahl oder einem Klammerausdruck eine Operation mit höherer Priorität als links, wird der rechte Operator zuerst ausgewertet. Trifft man auf eine öffnende Klammer, muss man den Substring bis zur zugehörigen schließenden Klammer suchen und die Auswertung rekursiv auf diesen Substring anwenden. Dadurch ergibt sich ein Binärbaum.

a) Entwickeln Sie einen Algorithmus, der den zu einem Ausdruck korrespondierenden Binärbaum aufbaut, wobei jeder innere Knoten einen Operator ('+', '-', '*', '/') repräsentiert, jeder Unterbaum einen linken bzw. rechten Operanden, und jedes Blatt eine Zahl. Begründen und implementieren Sie diesen Algorithmus im File `calculator.py`. Die Verwendung der Python-Funktionen `eval()` bzw. `exec` ist dabei nicht erlaubt. *15 Punkte*

b) Skizzieren Sie die Bäume, die sich für obige Beispiele ergeben. *3 Punkte*

c) Implementieren Sie ein Verfahren, um einen Ausdruck mit Hilfe des in a) erstellten Baums auszurechnen und geben Sie die Implementation im File `calculator.py` ab. *5 Punkte*

d) Schreiben Sie Unit Tests für Ihr Verfahren (ebenfalls in `calculator.py`). *4 Punkte*