Prof. Dr. Stefan Göller
Florian Bruse
Dr. Norbert Hundeshagen

Einführung in die Informatik

WS 2018/2019

Übungsblatt 4
15.11.2018-22.11.2018

Abgabe: Bis zum 22.11. 18:00 Uhr über moodle. Reichen Sie pro Aufgabe, die Sie bearbeitet haben, genau eine Textdatei mit dem Namen `aufgabe_i.py`, (wobei i die Aufgabennummer ist) ein, welche die Lösung ihrer Gruppe enthält.

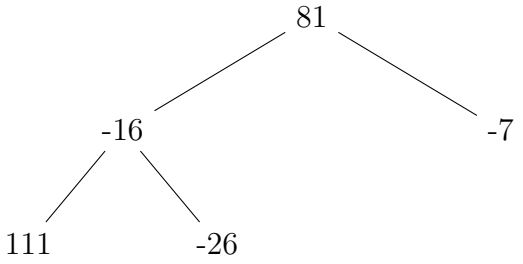
Aufgabe 1 (Rekursion) (25 Punkte):

Ein Baum ist ein in der Informatik häufig genutztes Konzept um strukturierte Daten zu repräsentieren. In dieser Aufgabe behandeln wir sogenannte Integer-Bäume, die wie folgt induktiv mit Hilfe von Tupeln definiert sind:

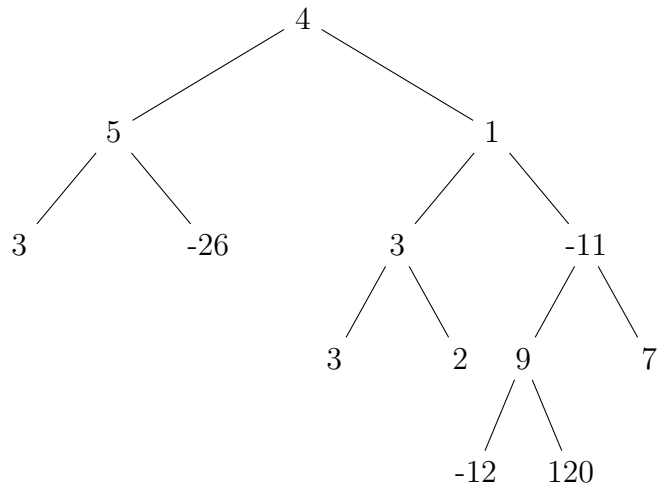
1. Jeder Integer ist ein Integer-Baum.
2. Wenn b_1 und b_2 Integer-Bäume sind und `n` ein Integer, dann ist auch das Tupel `(`b_1`,n,`b_2`)` ein Integer-Baum.

Beispiel:

- `((111,-16,-26),81,-7)` ist ein Integer-Baum. Grafisch kann man sich diesen wie folgt vorstellen:



- `((3,5,-26),4,((3,3,2),1,((-12,9,120),-11,7)))` ist ein Integer-Baum. Grafisch kann man sich diesen wie folgt vorstellen:



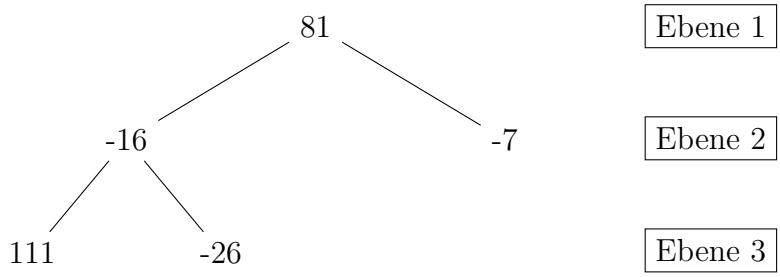
a) Schreiben Sie eine Funktion `summiere`, die die Zahlen in einem gegebenen Integerbaum, multipliziert mit der jeweiligen Ebenennummer, aufsummiert. Sie sollen dieses Problem mittels Rekursion lösen.

Beispiel:

- Für den Integer-Baum

  `((111,-16,-26),81,-7))`

  bildlich



  soll die Funktion also die Summe

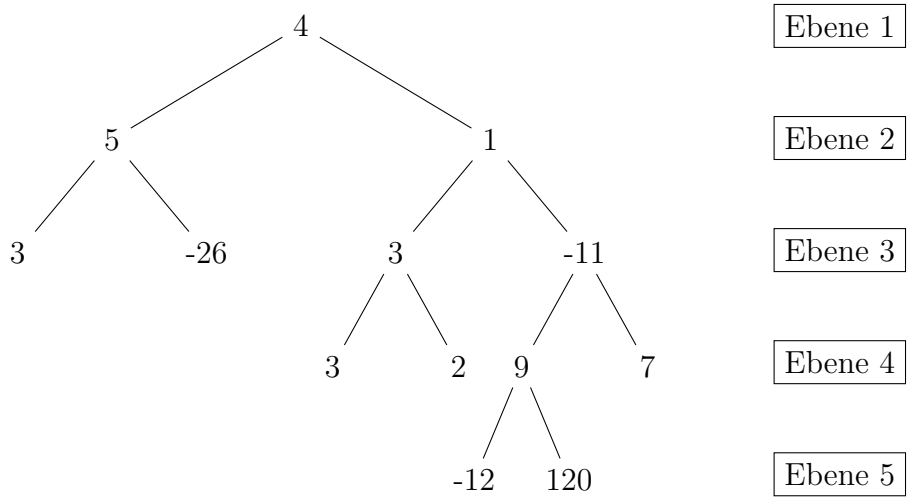
$$1 \cdot 81 + 2 \cdot (-16 - 7) + 3 \cdot (111 - 26) = 290$$

  berechnen.

- Für den Integer-Baum

  `((3,5,-26),4,((3,3,2),1,((-12,9,120),-11,7)))`

bildlich



soll die Funktion also die Summe

$$1 \cdot 4 + 2 \cdot (5 + 1) + 3 \cdot (3 - 26 + 3 - 11) + 4 \cdot (3 + 2 + 9 + 7) + 5 \cdot (-12 + 120) = 547$$

berechnen.

Hinweis: Es ist sinnvoll, eine weitere Hilfsfunktion zu schreiben, der sowohl der Baum als auch die Ebene übergeben werden. Diese Funktion soll rekursiv vorgehen. Um Ihre Funktion zu testen, legen Sie sich selber in einer Variablen Integer-Bäume, wie oben definiert, an und übergeben Sie diese Ihrer Funktion.

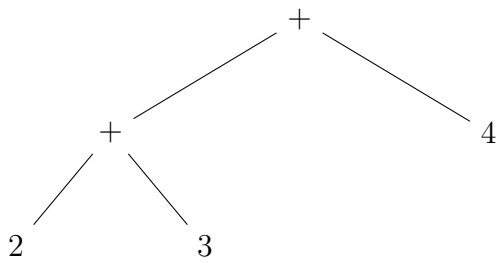
b) **(Sternchenaufgabe)** Schreiben Sie eine Funktion welche eine Zeichenkette bestehend aus `"("` , `")"` , `","` , `"-"` und Zahlen einliest. Falls es sich bei der eingelesenen Zeichenkette um die Kodierung eines Integer-Baums handelt, soll der entsprechende Integer-Baum zurückgegeben werden und falls nicht, soll NaN zurückgegeben werden.

Aufgabe 2 (Rekursive Baumtraversierungen) (20 Punkte):

Angelehnt an die Integer-Bäume aus der vorigen Aufgabe betrachten wir nun die Syntaxbäume arithmetischer Ausdrücke. Ein arithmetischer Baum ist folgendermaßen definiert:

1. Ein Integer ist ein arithmetischer Baum
2. Wenn b_1 und b_2 Integer-Bäume sind und `o` entweder `"+"`, `"-"`, `"*"` oder `"/"`, dann ist auch (`$b_1$,o,`b_2`)` ein arithmetischer Baum.

Beispielsweise ist der Baum `((2, "+", 3), "+ ", 4)` ein arithmetischer Baum, den man sich wie folgt vorstellen kann:



Wir wollen nun den arithmetischen Ausdruck, den ein solcher Baum beschreibt, als Zeichenkette ausgeben, und zwar jeweils in Präfix-, Infix- und Postfixnotation. Die Infixnotation ist dabei die Notation, die sie aus der Schule kennen, also mit dem Operator eines Ausdrucks (der Beschriftung eines inneren Knotens) zwischen den Ausdrücken, die sich aus dem linken und rechten Unterbaum ergeben. Beispielsweise ergibt sich für den obigen Baum die Zeichenkette `((2 + 3) + 4)`. Bei der Präfixnotation (auch normale polnische Notation genannt) kommt der Operator zuerst, und dann die Ausdrücke, die sich aus dem linken und dem rechten Unterbaum ergeben (Beispiel: `+ (+ 2 3) 4`). Bei der Postfixnotation (auch umgekehrt polnische Notation genannt) kommen erst die Ausdrücke, die sich aus dem linken und dem rechten Unterbaum ergeben, dann der Operator (Beispiel: `(2 3 +) 4 +`). Tatsächlich werden bei Prä- und Postfixnotation auch keine Klammern benötigt. Wir werden diese daher im Folgenden weglassen.

Ihr Aufgabe ist es nun, drei (rekursive) Funktionen zu schreiben, welche jeweils einen arithmetischen Baum (wie oben beschrieben) entgegen nehmen, und dann den dazugehörigen Ausdruck als Zeichenkette ausgeben. Eine Funktion soll diese Zeichenkette in Infixnotation ausgeben, eine in Präfixnotation, eine in Postfixnotation. Klammern sollen nur bei der Ausgabe als Infixnotation ausgegeben werden.

Bei Eingabe des arithmetischen Baums 5 sollen beispielsweise alle Funktionen die Ausgabe `"5"` liefern. Bei Eingabe des arithmetischen Baums `((2, "+", 3), "+", 4)` sollen die Funkionen die Ausgaben `"+ + 2 3 4 "` (Präfixnotation), `"((2 + 3) + 4)"` (Infixnotation), bzw. `"2 3 + 4 + "` (Postfixnotation) ausgeben.

Aufgabe 3 (Rekursion, Backtracking) (5 + 5 + 5 Punkte):

Es soll ein Programm geschrieben werden, welches einen vorhandenen Portmonee-Inhalt einließt und zurückgibt, ob und wie (in welcher Stückelung) ein gegebener Geldbetrag bezahlt werden kann.

a) Der Einfachheit halber nehmen wir an, dass der Portmonee-Inhalt nur aus 6€-Münzen, 11€-Münzen, 17€-Münzen und 25€-Münzen besteht. Schreiben Sie zunächst den Programmrahmen, in welchem eingelesen wird, wieviele der Münzen der oben angegebenen Beträge im Portmonee vorhanden sind. Speichern Sie diese Informationen in einer Liste. Ferner soll ein zu zahlender Geldbetrag eingegeben werden können und geprüft werden, ob dieser überhaupt durch den Portmonee-Inhalt bezahlt werden kann.

b) Schreiben Sie nun eine Funktion, welche mittels Backtracking bestimmt ob der Geldbetrag passend bezahlt werden kann.

c) Erweitern Sie Ihre obige Funktion zu einer neuen Funktion, welche im Fall, dass der Betrag passend bezahlt werden kann, 2 verschiedene Möglichkeiten ausgibt, wie der Betrag aus dem Portmonee-Inhalt bezahlt werden kann, falls es mindestens zwei solcher Möglichkeiten gibt.

   Beispiel: Portmonee-Inhalt: 7 x 6€, 2 x 11€, 1 x 17€, 2 x 25€.

   - Der Betrag 42€ kann also mit 7 x 6€ oder 1 x 25€, 1 x 11€, 1 x 6€ bezahlt werden.
   - Der Betrag 26€ lässt sich mit diesem Portmonee-Inhalt nicht passend bezahlen.