Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Tim Schulte, Christoph-Simon Senjak
Wintersemester 2018/2019

Universität Freiburg
Institut für Informatik

Übungsblatt 7

Abgabe: Dienstag, 4.12.2018, 20:00 Uhr

Aufgabe 7.1 (Primzahlen; Datei: `homeprimes.py`; Punkte: 4+4)

In der Zahlentheorie ist die Hausprimzahl HP(n) einer ganzen Zahl $n > 1$ diejenige Primzahl, welche durch wiederholtes Faktorisiern der zunehmenden Verkettung von Primfaktoren erhalten wird (siehe `https://en.wikipedia.org/wiki/Home_prime`). Beispielsweise ist HP(10) = 773 und berechnet sich wie folgt: Zunächst wird die Primfaktorzerlegung für $n = 10$ bestimmt. Die einzelnen Primfaktoren 2 und 5 ($10 = 2 \cdot 5$) werden in aufsteigender Reihenfolge zu 25 verkettet. Entsteht eine Primzahl aus der Verkettung, so ist dies die gesuchte Hausprimzahl. Andernfalls wird der Prozess wiederholt, wobei mit der zuletzt verketteten Zahl, in diesem Fall 25, fortgefahren wird:

Schritt	n	Primfaktoren	Verkettung	
1	10	2, 5	25	(nicht prim)
2	25	5, 5	55	(nicht prim)
3	55	5, 11	511	(nicht prim)
4	511	7, 73	**773**	(prim)

(a) Implementieren Sie eine Funktion `primefactors(n: int) -> list`, welche die Primfaktoren (mit Wiederholungen) einer ganzen Zahl $n > 1$ berechnet und diese in aufsteigender Reihenfolge als Liste zurückgibt.

```
>>> primefactors(12)
[2, 2, 3]             # Wiederholung
>>> primefactors(30)
[2, 3, 5]
>>> primefactors(37)  # Primzahl
[37]
>>> primefactors(1001)
[7, 11, 13]
```

(b) Implementieren Sie eine Funktion `homeprime(n: int) -> int`, welche die Hausprimzahl HP(n) einer ganzen Zahl $n > 1$ berechnet und zurückgibt.

```
>>> homeprime(9) == 311
True
>>> homeprime(10) == 773
True
>>> homeprime(5) == 5
True
```

Aufgabe 7.2 (Wort-Baum; Datei: `wordtree.py`; Punkte: 4+3+3)

In dieser Aufgabe geht es darum, eine Zeichenkette einzulesen und dabei eine Datenstruktur anzulegen, die es später erlaubt für ein gegebenes Wort zu entscheiden, ob

und wie oft dieses Wort in der Zeichenkette vorkommt. Unter einem *Wort* verstehen wir im Folgenden jede endliche Folge von Buchstaben des deutschen Alphabets (also den Zeichen a, b, c, . . . , z, A, B,. . . , Z, ä, Ä, ö, Ö, ü, Ü, ß) der Länge ≥ 1. Je zwei Wörter in der Zeichenfolge werden durch eine nicht-leere, endliche Folge von Zeichen, die nicht zu diesen Buchstaben gehören (z.B. Leerzeichen, Satzzeichen, Zeilenumbrüche), getrennt. Es soll nun ein Suchbaum erzeugt werden, sodass jeder Knoten ein in einem String `s` vorkommendes Wort und dessen Häufigkeit repräsentiert. Laden Sie das Template `wordtree.py` von der Vorlesungswebsite herunter. Dieses enthält eine Funktion `next_word(s)` und eine Klasse `Node`:

- `next_word(s)` gibt, angewendet auf einen String `s`, ein Tupel `(word, rest)` zurück, wobei `word` das erste Wort (im Sinne der Spezifikation) in `s` ist und `rest` die Zeichenfolge ist, die in `s` auf `word` folgt.
- `Node` repräsentiert einen Knoten im Suchbaum und verfügt über folgende Attribute: (1) eine Markierung `mark`, (2) die Worthäufigkeit `frequency`, (3) einen linken Teilbaum `left` und (4) einen rechten Teilbaum `right`.

(a) Definieren Sie eine Funktion `word_tree(s)`, die aus dem übergebenen String `s` diesen Suchbaum erzeugt und zurückgibt. Natürlich kann Ihre Funktion eine selbst-definierte Hilfsfunktion verwenden. Beispiel:

```
>>> s = "spam eggs spam eggs ham spam hamham Spam"
>>> t = Node('spam', 3, Node('eggs', 2, Node('Spam', 1, None,
             None), Node('ham', 1, None, Node('hamham', 1,
             None, None))), None)
>>> str(word_tree(s)) == str(t)
True
```

(b) Definieren Sie eine Funktion `word_freq(tree, word)`, die für einen solchen Suchbaum `tree` und ein Wort `word`, die in `tree` hinterlegte Anzahl der Wortvorkommnisse von `word` zurückgibt. Falls das Wort in dem Baum nicht vorkommt, soll die Funktion den Wert `0` zurückgeben.

```
>>> t = word_tree("spam eggs spam")
>>> word_freq(t, 'spam'), word_freq(t, 'ham')
(2, 0)
```

(c) Definieren Sie eine Funktion `print_tree(tree)`, die alle in `tree` abgelegten Wörter und die in `tree` jeweils hinterlegte Anzahl der jeweiligen Wortvorkommnisse zeilenweise (pro Zeile ein Wort und dessen Anzahl) ausgibt. Dabei soll der Baum in symmetrischer Reihenfolge (*In-Order*) traversiert werden. Die Ausgabe könnte in etwa wie folgt aussehen:

```
>>> print_tree(word_tree("spam eggs spam alma"))
alma: 1
eggs: 1
spam: 2
```

Aufgabe 7.3 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet07` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, Bezug zur Vorlesung, Interessantes, benötigter Zeitaufwand, etc.).