7. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Spezifikationsbasiertes Testen
ausgegeben: Mi, 06.06.2018, fällig: Mi, 13.06.2018 (15:00 Uhr)

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP7.hs` in Ihrem Gruppenverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Wir betrachten folgende Aufgabe: Gegeben sind ein Text und ein Suchwort. Gesucht ist die Menge aller Positionen, in denen das Suchwort im Text vorkommt. Zur Modellierung benutzen wir folgende Typen:

```
type Text  = String
type Word  = String
type First = Int
type Last  = Int
```

Gesucht ist dann eine Funktion

```
occ :: Text -> Word -> [(First,Last)]
```

wobei die Ergebnisliste für jedes gefundene Vorkommen des Suchwortes im Text jeweils die erste und letzte Indexposition des Suchwortvorkommens im Text angibt. Dabei gilt, dass sich Vorkommen von Suchwörtern im Text nicht überlappen. Endet ein Suchwortvorkommen an Indexposition j des Textes, kann das nächste Suchwortvorkommen frühestens an Indexposition $j + 1$ des Textes beginnen. Die Suchwortvorkommen in der Resultatliste sollen aufsteigend angeordnet sein, d.h. das linkeste Vorkommen des Suchworts im Text ist das Kopfelement der Resultatliste.

Wir wollen eine naive und eine besser performante Suchfunktion nach dem Verfahren von Boyer-Moore-Horspool im Sinn der Idee funktionaler Perlen schreiben, die wir auch als Spezifikation (S) bzw. Implementierung (I) für dieses Problem sehen wollen:

1. Schreiben Sie eine Funktion

   ```
   occS :: Text -> Word -> [(First,Last)]
   ```

   für das Suchproblem, die nach folgender Idee vorgeht. Angewendet auf einen Text t und ein Suchwort w mit Länge n überprüft `occS`, ob die Zeichen w_{n-1} und t_{n-1} übereinstimmen, d.h. die Zeichen von w bzw. t an den jeweiligen Indexpositionen $n - 1$ (der erste Index hat jeweils Wert 0!) Falls ja, überprüft `occS`, ob die Zeichen w_{n-2} und t_{n-2} übereinstimmen und so weiter. Gilt eine Übereinstimmung schließlich auch für w_0 und t_0, so ist ein Vorkommen gefunden und das Element $(0, n - 1)$ zur Ergebnisliste hinzuzufügen. Tritt für einen der Indizes ein Unterschied auf, so kann ausgeschlossen werden, dass ein Suchwortvorkommen an Indexposition $n - 1$ im Text endet und die Suche wird in

gleicher Weise fortgesetzt an der Indexposition n im Text, d.h. ob das Suchwort an dieser Indexposition im Text endet.

2. Schreiben Sie eine Funktion

   ```
   occI :: Text -> Word -> [(First,Last)]
   ```

   für das Suchproblem, die nach der Idee von Boyer-Moore-Horspool vorgeht. Angewendet auf einen Text t und ein Suchwort w mit Länge n geht `occI` zunächst wie `occS` vor, d.h. es wird überprüft, ob das Suchwort an der Indexposition $n-1$ im Text endet. Ist dies der Fall unterscheidet sich `occI` nicht von `occS`. Ist dies jedoch nicht der Fall, so setzt `occI` nicht wie `occS` die Suche genau eine Position weiter rechts im Text fort, sondern so viele Positionen rechts wie möglich. Dieser "Rechtsvorschub" kann für jedes Zeichen von w vorweg berechnet werden! In Abhängigkeit der Folge der Zeichen von w und dem Auftreten der Nichtübereinstimmungsstelle kann die Suche im günstigsten Fall um die gesamte Länge von w nach rechts verschoben fortgesetzt werden, im schlechtesten Fall auch nur um ein Zeichen wie bei `occS`. Insgesamt ergibt sich so aber ein im allgemeinen deutlich besseres Laufzeitverhalten für `occI` als für `occS`.

3. Validieren Sie mithilfe von `QuickCheck`, dass `occS` und `occI` stets dieselben Resultate liefern. Definieren Sie dafür eine Eigenschaft

   ```
   prop_coincide :: Text -> Word -> Bool
   ```

   geeignet.