8. Übungsaufgabe zu
Fortgeschrittene funktionale Programmierung
Thema: Teile und Herrsche, Spezifikationsbasiertes Testen
ausgegeben: Mi, 16.05.2012, fällig: Mi, 30.05.2012

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `AufgabeFFP8.hs` in Ihrem Gruppenverzeichnis ablegen, wie gewohnt auf oberstem Niveau. Kommentieren Sie Ihre Programme aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

In Kapitel 4.1 der Vorlesung sind verschiedene Algorithmen zur Lösung des "Smallest Free Number (SFN)"-Problems angegeben, darunter:

(i) Mithilfe der Funktion `minfree`, die sich in ihrer Grundversion zur Lösung des SFN-Problems auf die Differenz des Stroms natürlicher Zahlen und der anfänglich gegebenen Zahlenmenge abstützt, siehe Folie 247.

(ii) Zwei Array-basierte Algorithmen:

- Mithilfe der Funktion `checklist`, siehe Folie 251.
- Mithilfe der Funktion `countlist`, siehe Folie 252.

(iii) Drei "teile-und-herrsche"-basierte Algorithmen:

- Der "Basic Divide-and-Conquer-Algorithm", siehe Folie 256.
- Der "Refined Divide-and-Conquer-Algorithm", siehe Folie 259.
- Der "Optimized Divide-and-Conquer-Algorithm", siehe Folie 262.

- Implementieren Sie die obigen 6 Algorithmen zur Lösung des SFN-Problems wie in der Vorlesung angegeben und vergleichen Sie (ohne Abgabe!) die relative Performanz der verschiedenen Implementierungen miteinander. Untersuchen Sie insbesondere, ob Sie die Aussagen zu den relativen Geschwindigkeiten der Algorithmen von Folie 263 bestätigt finden.
- Die obigen 3 nach dem "Teile-und-Herrsche"-Prinzip vorgehenden Implementierungen folgen nicht dem Schema des "Teile-und-Herrsche"-Funktionals aus Kapitel 3.1 der Vorlesung.

  Geben Sie für jeden dieser 3 Algorithmen eine entsprechende Implementierung mit Hilfe des "Teile-und-Herrsche"-Funktionals aus Kapitel 3.1 der Vorlesung an. Geben Sie dazu jeweils die Implementierungen der Funktionen `indiv`, `solve`, `divide` und `combine` an und verwenden Sie für die 3 Implementierungen für jeden Namen jeweils den Präfix `b_`, `r_` und `o_` für die "basic", "refined" und "optimized" Algorithmusvariante. Die Funktionen heißen also `b_indiv`, `r_indiv`, `o_indiv` usw.

- Validieren Sie mithilfe von `QuickCheck`, dass die obigen 9 Implementierungen zur Lösung des SFN-Problems
  - für (zulässige) duplikatfreie gleiche Argumente die gleiche Funktion festlegen, also gleiche Resultate liefern
  - für (zulässige) nichtduplikatfreie gleiche Argumente sich möglicherweise Unterschiede beobachten lassen.

  Definieren Sie dafür zwei entsprechende Eigenschaften

  ```
  prop_allImplsEq_a :: [Int] -> Bool
  prop_allImplsEq_b :: [Int] -> Property
  ```

  in Ihrem Programm. Die obigen Eigenschaften sollen also "wahr" sein, wenn alle 9 Implementierungen gleichzeitig überprüft werden und alle dabei zum selben Ergebnis kommen.

  Für die Eigenschaft `prop_allImplsEq_b` soll durch eine geeignete Vorbedingung sichergestellt werden, dass negative Listenelemente enthaltende automatisch generierte Testfälle verworfen und nicht als gültiger Testfall behandelt werden.

Verwenden Sie für die 9 Varianten von `minfree` die Namen

- `minfree_bv`: "basic version"
- `minfree_chl`: "checklist"
- `minfree_col`: "countlist"
- `minfree_b`: "basic divide-and-conquer"
- `minfree_r`: "refined divide-and-conquer"
- `minfree_o`: "optimized divide-and-conquer"
- `minfree_bhof`: "basic divide-and-conquer mittels higher-order function"
- `minfree_rhof`: "refined divide-and-conquer mittels higher-order function"
- `minfree_ohof`: "optimized divide-and-conquer mittels higher-order function"