10. Aufgabenblatt zu Funktionale Programmierung vom 14.12.2010.
Fällig: Di, 21.12.2010 / Di, 11.01.2011 (jeweils 15:00 Uhr)

Themen: *Teile und Herrsche, Knobeleien und backtracking*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe10.hs` ablegen. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

1. In dieser Aufgabe greifen wir noch einmal das Prinzip "Teile und Herrsche" in der Form von Aufgabenblatt 9 auf.

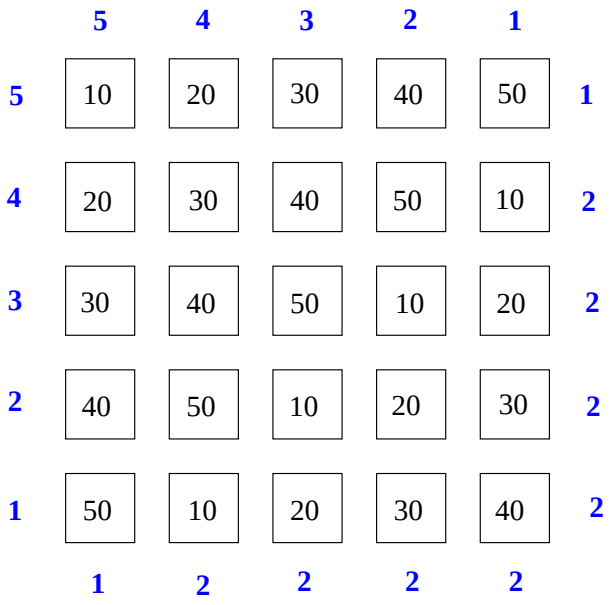
   Zeigen Sie, dass der von Tony Hoare erfundene `quickSort`-Algorithmus und die in der Vorlesung besprochene darauf beruhende Funktion `quickSort` dem "Teile und Herrsche"-Prinzip genügt und sich als Spezialisierung des Funktionals `divAndConquer` darstellen lässt. Geben Sie dazu geeignete Implementierungen der Funktionen `indQ`, `solveQ`, `divideQ` und `combineQ` an, so dass `quickSort` in Ihrer Abgabedatei insgesamt wie folgt implementiert ist:

   ```
   quickSort :: Ord a => [a] -> [a]
   quickSort = divAndConquer indQ solveQ divideQ combineQ
   ```

   Die Funktionen `indQ`, `solveQ`, `divideQ`, `combineQ` sollen dabei global definiert sein, also nicht lokal in einer `let`- oder `where`-Klausel.

2. In dieser Aufgabe betrachten wir die vollständige Variante der Skyline-Knobelei. In jeder Zeile und in jeder Spalte einer $n \times n$-Matrix stehen n-Hochhäuser mit 10, 20, 30,...,$n * 10$ Etagen. Die Zahlen am Rand geben an, wieviele Hochhäuser man sieht, wenn man von diesem Punkt in die Zeile bzw. Spalte schaut. Dabei gilt: Höhere Hochhäuser verdecken niedrigere.

   Die folgende Abbildung zeigt ein Beispiel für eine 5×5-Matrix (bzw. $(5 + 2) \times (5 + 2)$-Matrix einschließlich Sichtbarkeitsinformation).



   Wir modellieren Skyline durch folgenden Datentyp:

   ```
   type Row     = [Integer]
   type Skyline = [Row]
   ```

In einer $(n+2) \times (n+2)$-Matrix beschreiben die 1-te und $(n+2)$-te Reihe der Matrix die Anzahl der sichtbaren Hochhäuser in den Spalten, die Werte in der 1-ten und $(n+2)$-ten Spalte die Anzahl der sichtbaren Hochhäuser in den Zeilen. Die Zahlen an den Eckpositionen $(1,1)$, $(1,n+2)$, $(n+2,1)$ und $(n+2,n+2)$ spielen keine Rolle. Sie werden nicht betrachtet oder ausgewertet. Als *Sichtbarkeitsfelder* bezeichnen wir die Randfelder einer solchen Matrix, die die Sichtbarkeitsinformation tragen. Als *Hochhausstadt* bezeichnen wir die innere Matrix bestehend aus denjenigen Feldern, auf denen Hochhäuser stehen. Eine Hochhausstadt heißt gültig, wenn in jeder Zeile und in jeder Spalte je ein Hochhaus mit 10, 20, 30,...,$n * 10$ Etagen steht. Sichtbarkeitsinformation heißt *gültig*, wenn es eine gültige Hochhausstadt dazu gibt.

(a) Schreiben Sie eine Haskell-Funktion `isValid` mit der Signatur `isValid :: Skyline -> Bool`. Angewendet auf einen Wert vom Typ `Skyline` überprüft `isValid`, ob das Argument eine *gültige* Skyline beschreibt, d.h., das Argument eine $(n+2) \times (n+2)$-Matrix ist, in jeder Zeile und Spalte je ein Hochhaus der erwarteten Etagenzahl auftritt, und die Sichtbarkeitsangaben korrekt sind. In diesem Fall ist das Resultat des Aufrufs `True`, sonst `False`.

(b) Schreiben Sie eine Haskell-Funktion `compVisibility` mit der Signatur `compVisibility :: Skyline -> Skyline`, die angewendet auf ein Argument mit gültiger Hochhausstadt zugehörige Sichtbarkeitsinformation korrekt ergänzt. Die Werte in den Eckpositionen können im Resultat beliebig gewählt sein. Beschreibt das Argument keine gültige Hochhausstadt, wird das Argument unverändert zurückgegeben.

(c) Schreiben Sie eine Haskell-Funktion `buildSkyscrapers` mit der Signatur `buildSkyscrapers :: Skyline -> Maybe Skyline`, die angewendet auf ein Argument mit Sichtbarkeitsinformation und leerer oder teilbebauter Hochhausstadt eine zugehörige Hochhausstadt korrekt aufbaut; in diesem Fall ist das Resultat ein Wert vom Typ `Just Skyline`. Anderenfalls, wenn dies nicht möglich ist, z.B. wenn die teilbebaute Hochhausstadt Hochhäuser unzulässiger Etagenzahl enthält oder es zur gegebenen Sichtbarkeitsinformation und Teilbebauung keine gültige Hochhausstadt gibt, ist das Resultat `Nothing`. Sind mehrere korrekte Hochhausstadtbebauungen möglich, reicht es, wenn Ihre Funktion eine gültige Hochhausstadt berechnet. Unbebaute Grundstücke tragen den Wert 0; in der *leeren* Hochhausstadt sind alle Grundstücke unbebaut.

Sie können davon ausgehen, dass die obigen Funktionen nur auf Hochhausstädte der Größe (5×5) (zuzüglich Sichtbarkeitsinformation) oder kleiner angewendet werden.

Hinweis:

- Verwenden Sie *keine* Module, um frühere eigene Funktionen wiederzuwenden. Wenn Sie eigene Funktionen wiederverwenden möchten, kopieren Sie diese in die Abgabedatei `Aufgabe10.hs`. Andere Dateien als diese werden vom Abgabeskript ignoriert.

Frohe Weihnachten

und

einen guten Rutsch ins neue Jahr!

Haskell Live

Die beiden abschließenden *Haskell Live*-Termine finden am Freitag, den 17.12.2010, und am Freitag, den 14.01.2011, statt.