Praktische Informatik 3 WS 06/07

5. Übungsblatt

Ausgabe: 09.01.07
Abgabe: 23.01.07

Christoph Lüth <cxl>
Matthias Berger <tokio>
Klaus Hartke <hartke>
Cui Jian <ken>
Friederike Jolk <rikej>
Christian Maeder <maeder>
Diedrich Wolter <dwolter>

13 *Permutationen* — *5 Punkte*

In der folgenden Aufgabe werden Permutationen von Listen benötigt. Implementieren Sie zunächst die folgenden Funktionen

```
allPerms :: Ord a=> [a]-> [[a]]
randomPerm :: Ord a=> [a]-> IO [a]
```

welche alle Permutationen einer Liste sowie eine zufällige Permutation erzeugen. Der Aufwand von `randomPerm` sollte quadratisch sein. Darüber hinaus soll eine Funktion

```
nextPerm :: Ord a=> [a]-> [a]
```

implementiert werden, welche die in der lexikographischen Ordnung nächste Permutation berechnet. Der Algorithmus hierzu geht auf den holländischen Informatiker Edsger Dijkstra zurück. Sei `a` die Eingabeliste, dann wird die nächste Permutation wie folgt berechnet:

- Zuerst finden wir den größten Index i, so dass das i-te Element der Liste kleiner ist als das folgende.
- Gibt es kein solches Element, ist die Liste absteigend sortiert — dies ist die in der lexikographischen Ordnung größte Liste. Ihr Nachfolger ist die in der lexikographischen Ordnung kleinste Liste, die aufsteigend sortiert ist (und die wir einfach durch Umkehrung erhalten).
- Danach finden wir dasjenige k aus $i+1, \ldots, n$, so dass das k-te Element der Liste das kleinste Element ist, welches größer als das i-te Element ist.
- Dann vertauschen wir das i-te und k-te Element und sortieren den Rest der Liste ab dem $i+1$-ten Elemen in aufsteigender Reihenfolge.

Die Funktionalität von `nextPerm` können Sie mit `allPerms` testen.

14 *Sudoku!* — *10 Punkte*

Unser nächster Schritt in das lukrative Entertainment-Business sind Zahlenrätsel, genauer gesagt Sudokus. Sudokus sind eine Art Kreuzworträtsel für Zahlenfreunde. Kurz gesagt geht es darum, in einem 9x9-Feld, in dem einige Zahlen eingetragen sind, die Zahlen von 1

bis 9 so einzutragen, dass in keiner Zeile, in keiner Spalte und in keinem der 3x3-Unterblöcke des 9x9-Feldes eine Zahl doppelt auftritt. [1]

Es gibt eine Menge hilfreicher Programme, die ein Sudoku lösen; hier soll eines implementiert werden, welches ein Sudoku entwirft. Die Größe des Feldes ist ein Programmparameter. Ein Sudoku-Feld wird wie folgt modelliert:

```
boxSize :: Int
boxSize = 3
squareSize :: Int
squareSize = boxSize^2

type Box = [Int]      -- Eine Box (Größe boxSize x boxSize)
type Col = [Int]      -- Eine Spalte
type Row = [Int]      -- Eine Zeile
type Square = [[Int]] -- Das gesamte Feld
```

Hauptsächlich besteht das Programm aus folgenden Funktionen:

- Die Funktion

  ```
  makeSudoku :: IO Square
  ```

  berechnet zufällig ein neues Sudoku nach folgendem, recht einfachen Algorithmus. Davon ausgehend, dass jede Zeile sowieso eine Permutation der Ziffern von `1` bis `squareSize` ist, erstellen wir einen zufälligen ersten Kandidaten einer Lösung als `squareSize` zufällige Permutationen (also im 'normalen' Fall 9 zufällige Permutationen der Liste `[1..9]`).

  Aus diesem Kandidaten wird eine Lösung erstellt, indem wir sukzessive jede Zeile in die Lösung übertragen. Hierbei wird jedesmal geprüft, ob die Lösung die Sudoku-Bedingung erfüllt, d.h. Zeilen, Spalten und Boxen keine doppelten Elemente enthalten. Ist dies nicht der Fall, wird von der zu übertragenen Zeile die Folgepermutation versucht. Ist die Folgepermutation wieder die ursprüngliche, gibt es keine weiteren Permutation mehr, und es muss von der vorherigen Zeile die nächste genommen werden (backtracking).

  Es wird also sukzessiv eine Liste von Zeilen aufgebaut, welche als Invariante die Sudoku-Bedingung erfüllt; hat diese Lösung die erforderliche Länge, bildet sie ein vollständiges Feld und wird zurückgegeben.

- Die Funktion

  ```
  printSudoku :: Int-> Square-> IO ()
  ```

[1] Für weitere Informationen siehe zum Beispiel `http://www.zeit.de/sudoku`.

gibt eine Lösung als Rätsel in graphisch ansprechender Form aus, d.h. es wird das Feld ausgegeben, wobei einige Zahlen bedeckt sind (hier wird der Wert nicht ausgegeben). Die Zahl ist der 'Schwierigkeitsgrad' und gibt an, wieviel Prozent des Feldes bedeckt werden sollen.

Fügen Sie das Ganze zu einem aus der Kommandozeile ausführbaren Programm zusammen, welches mit einem Schwierigkeitsgrad aufgerufen wird und ein zufälliges Sudoku dieses Schwierigkeitsgrades ausgibt.

15 *Findet Java!* *5 Punkte*

Wir schreiben das Jahr 2067. Nach einem unglücklichen Zwischenfall im Jahr 2058 mit dem in Java implementierten europäischen Klimakontrollsystem, einer `Null`-Referenz und einem Typcast, der zur Überflutung weiter Teile der Niederlande und Norddeutschlands führte, wurde die Programmiersprache Java vor sechs Jahren endlich von der Europäischen Kommission verboten und ihre Benutzung unter Strafe gestellt.

Leider horten einige Unverbesserliche immer noch Java-Programme (obwohl der Besitz nicht mehr erlaubt ist), oft unter falschem Namen: in der Datei `Fac.hs` verbirgt sich dann beispielsweise ein heimtückisches Test-Applet, welches Apfelmännchen zeichnet.

Deshalb sollen Sie in dieser Aufgabe ein Programm implementieren, welches alle Dateien in einem gegebenen Dateibaum untersucht, ob es sich dabei um Java-Programme handeln könnte. Das Programm hat zwei Teile:

- Zum einen eine Funktion, welche den Dateibaum ab einer gegebenen Stelle traversiert und die gegebene Aktion mit jeder Datei ausführt:

  ```
  traverseFiles :: FilePath -> (FilePath-> IO ())-> IO ()
  ```

- Zum anderen die eigentliche Prüffunktion. Diese soll einfach den Inhalt der Datei lesen, in Wörter zerlegen, und prüfen, ob der Anteil der Java-Schlüsselwörter dabei einen bestimmten absoluten Schwellenwert N_J oder einen prozentualen Anteil n_J übersteigt. Wenn ja, wird der Dateiname mit einer Warnung ausgegeben (bzw. in der Produktivversion ohne Warnung gelöscht)

Bestimmen Sie anhand einiger geeigneter Testdateien Werte für N_J und n_J und implementieren Sie ein aus der Kommandozeile aufrufbares Programm `findJava`, welches als Argument einen Pfad erhält und den Dateibaum ab diesem Pfad traversiert und nach Java-Programmen durchsucht.

Hinweis: Die Funktionen aus dem Modul `System.Directory`, insbesondere `getDirectoryContents`, können hierfür hilfreich sein.

Dies ist Version 1.0 vom 2007/01/09 13:02:12.