Peter Padawitz / Hubert Wagner Dortmund, den 08.05.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 1

Keine Abgabe - nur Präsenzaufgaben

Aufgabe 1.1
Geben Sie eine `Haskell`-Funktion an, die zu einer ganzen Zahl n die Anzahl der verschiedenen nichtnegativen Teiler von n ermittelt.

Aufgabe 1.2
Gegeben seien zwei Populationen A und B mit m bzw. n Elementen, die sich in ihrem Wachstumverhalten in der folgenden Weise beeinflussen:

- Ist die Population von A oder von B ausgestorben, so sind im darauf folgenden Jahr beide Populationen ausgestorben.
- Ist in einem Jahr die Population von B mehr als viermal so groß wie die von A, so ist im darauf folgenden Jahr die Population von A $\frac{3}{2}$ mal so groß wie im Jahr zuvor und die Population von B doppelt so groß.
- Ist in einem Jahr die Population von B höchstens viermal so groß wie die von A, aber in jedem Fall größer als die von A, so ist im darauffolgenden Jahr die Population von A unverändert, während die Population von B um ein $\frac{1}{4}$ ihrer vorherigen Größe kleiner geworden ist.
- Ist die Population von B kleiner als die von A (aber noch nicht ausgestorben), so ist im darauf folgenden Jahr die Population von B nur noch $\frac{1}{4}$ so groß, während die Population von A um ein $\frac{1}{4}$ ihrer vorherigen Größe kleiner geworden ist.

Implementieren Sie eine Funktionen f, die die Größe von A und von B nach x Jahren angibt!

Aufgabe 1.3
Schreiben Sie eine `Haskell`-Funktion, die aus einer Liste von ganzen Zahlen alle Zahlen entfernt, die sich darstellen lassen als $7 \cdot n + 1$ oder $7 \cdot n + 3$ für ein $n \in \mathbb{Z}$.

Aufgabe 1.4
In `Haskell98` gibt es den (Daten-) Typ `Ratio Integer` für die Menge der rationalen Zahlen.

1. Schreiben Sie eine `Haskell`-Funktion, die bei Eingabe der positiven ganzen Zahl n alle positiven rationalen Zahlen, deren Nenner $\leq n$ und größer als der Zähler sind, ohne Wiederholungen ausgibt. (Z.B. dürfte also nicht 1%2 und 2%4 ausgegeben werden, da beide Darstellungen dieselbe rationale Zahl darstellen.)
2. Schreiben Sie eine `Haskell`-Funktion, die bei Eingabe der positiven ganzen Zahl n die Summe der in der 1. Teilaufgabe ausgegeben rationalen Zahlen berechnet.

Peter Padawitz / Hubert Wagner Dortmund, den 15.05.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 2

Abgabe: bis 22.05.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 2.1
Das folgende Problem „Türme von Brahma" (bekannt auch als „Türme von Hanoi") geht auf eine hinterindische Sage zurück: In einem im Dschungel verborgenen hinterindischen Tempel sind Mönche seit Beginn der Zeitrechnung damit beschäftigt, einen Stapel von 50 goldenen Scheiben mit nach oben hin abnehmendem Durchmesser, die durch einen goldenen Pfeiler in der Mitte zusammengehalten werden, durch sukzessive Bewegungen jeweils einer einzigen Scheibe auf einen anderen goldenen Pfeiler umzuschichten. Dabei dürfen sie einen dritten Pfeiler als Hilfspfeiler benutzen, müssen aber darauf achten, dass niemals eine Scheibe mit einem größerem Durchmesser auf eine mit kleinerem Durchmesser zu liegen kommt.
Die Sage berichtet, dass das Ende der Welt gekommen ist, wenn die Mönche ihre Aufgabe beendet haben.
Schreiben Sie eine Haskell-Funktion, die bei n Scheiben ausgibt, wie die Scheiben umzuschichten sind. Die einzelnen Schritte sollen dabei durchnummeriert sein.

Aufgabe 2.2
Das Pascalsche Dreieck ist ein Zahlendreieck, das sich in der folgenden Weise ergibt:

```
          1
        1 1
      1 2 1
    1 3 3 1
  1 4 6 4 1
1 5 10 10 5 1
  ...
```

Hierbei steht in der ersten Reihe eine 1. Die jeweils nachfolgende Reihe wird gebildet, indem sukzessive die Summe zweier benachbarter Zahlen der darüberliegenden Reihe bestimmt wird und an den Anfang sowie das Ende der entstandenen Reihe eine 1 angefügt wird.
Schreiben Sie ein Haskell-Funktion `pascal`, die bei Eingabe n die n-te Reihe im Pascalschen Dreieck berechnet.
(Die Zahlen im Pascalschen Dreieck stellen somit Binomialkoeffizienten dar.)

Aufgabe 2.3
Schreiben Sie eine `Haskell`-Funktion, die bei Eingabe einer Liste von Strings in jedem String dieser Liste das erste Symbol, wenn es ein Buchstabe ist, in den entsprechenden Großbuchstaben umwandelt.

Aufgabe 2.4
Geben Sie eine Haskell Funktion an, die bei Eingabe zweier Strings feststellt, ob der erste String als Teilstring im zweiten enthalten ist. (Es wäre schön, wenn Sie auch eine effiziente Lösung anbieten können.)

Peter Padawitz / Hubert Wagner Dortmund, den 22.05.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 3

Abgabe: bis 29.05.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 3.1
Das Sieb des Eratosthenes beschreibt ein Verfahren, um alle Primzahlen unterhalb einer Schranke n zu bestimmen:

1. Man schreibe alle natürlichen Zahlen von 2 bis n auf.
2. Man markiere die Zahl 2 und streiche dann jede zweite Zahl.
3. Ist k die erste nicht-gestrichene und nicht-markierte Zahl, so markiere man k und streiche dann jede nachfolgende k-te Zahl (gleichwertig: jede Zahl, die durch k teilbar ist).

Übrig bleiben dann genau die Primzahlen $\leq n$. (In dem Verfahren, das beschrieben wurde, kann natürlich k beschränkt werden durch $k \leq \sqrt{(}n)$.)

Schreiben Sie eine Haskell-Funktion, die mit Hilfe des Siebes von Eratosthenes die unendliche Liste der Primzahlen erzeugt. Verwenden Sie diese Funktion, um damit zu testen, ob eine Zahl eine Primzahl ist.

Aufgabe 3.2
Die Goldbachsche Vermutung besagt, dass jede gerade Zahl ≥ 6 als Summe von 2 Primzahlen darstellbar ist.

Schreiben Sie eine Haskell-Funktion, die eine unendliche Liste von Tripeln (n, p, q) mit $n = p+q$, $n \geq 6$, n gerade und p und q Primzahlen generiert. Für jede gerade Zahl $n \geq 6$, die eine solche Darstellung hat, sollte dabei nur ein Tripel in dieser Liste enthalten sein.

Aufgabe 3.3
Geben Sie eine Haskell-Funktion an, die bei Eingabe eines endlichen Binärbaumes die Anzahl der inneren Knoten des Baumes sowie die Anzahl der Blätter bestimmt.

Aufgabe 3.4
Implementieren Sie in Haskell Addition, Subtraktion, Multiplikation und Division von komplexwertigen Brüchen, d.h. von Zahlen $a + i \cdot b$, wobei a und b rationale Zahlen (Typ: Ratio Integer) sind.

Peter Padawitz / Hubert Wagner Dortmund, den 02.06.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 4

Keine Abgabe - nur Präsenzaufgaben

Aufgabe 4.1

1. Definieren Sie eine Haskell-Funktion

   `paarsumtest:: [Integer]` $\rightarrow$ `Integer` $\rightarrow$ `(Integer,Integer)`,

   die zu einer Liste L von ganzen Zahlen und zu einer ganzen Zahl s ein Paar (x, y) von in L vorkommenden verschiedenen Listenelementen x und y mit $x + y = s$ findet, falls es ein solches Paar mit dieser Eigenschaft gibt. (x und y können durchaus den gleichen Wert haben. Z.B. soll `paarsumtest [2,3,3,5] 6` den Wert (3,3) ausgeben, `paarsumtest [2,3,5] 6` aber nicht definiert sein.)

   Bemerkung: Eine geschickte Implementierung sortiert zunächst die Liste und ermittelt dann in Länge(L) vielen Rekursionsschritten ein solches Paar.

2. Erweitere die Haskell-Funktion `paarsumtest` zu einer Haskell-Funktion `paarsumall`, die die Menge aller Paare (x, y) von in L vorkommenden verschiedenen Listenelementen x und y mit $x + y = s$ und $x \leq y$ in Form einer Liste ausgibt.

Aufgabe 4.2

Schreiben Sie eine Haskell-Funktion, die folgendes leistet:

Bei Eingabe einer natürlichen Zahl n und einer Menge V von Knoten in Form einer Liste soll die Funktion mittels eines durch n gesteuerten Zufallsgenerators einen gerichteten Graphen mit V als Knotenmenge erzeugen. Der erzeugte Graph soll dabei in Form einer Abbildung vorliegen, die jedem Knoten die Liste der anliegenden Knoten, zu denen eine Kante hinführt, zuordnet.

Für die Verwendung eines Zufallszahlengenerators greifen Sie auf das Module Random.hs zurück. Z.B. wird mit `{l :: [Int]; l = randomRs (a, b) (mkStdGen n)}` eine Liste von ganzzahligen Zufallszahlen im Intervall von a bis b erzeugt. n beinflusst dabei das Erzeugungsverfahren der Zufallszahlen. Für unterschiedliche n erhalten Sie unterschiedliche Listen von Zufallszahlen.

Aufgabe 4.3

Geben Sie eine Haskell-Funktion an, die einen gerichteten Graphen topologisch sortiert. Die Eingabe des Graphen erfolgt durch die Angabe der Liste der Knoten und der Abbildung, die jedem Knoten die Liste der anliegenden Knoten, zu denen eine Kante hinführt, zuordnet (siehe Aufgabe 4.1).

Aufgabe 4.4

Implementieren Sie in Haskell den Algorithmus von Prim zur Erzeugung eines minimalen Spannbaums.

Peter Padawitz / Hubert Wagner Dortmund, den 06.06.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 5

Abgabe: bis 12.06.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 5.1
In Aufgabe 4.2 war eine Haskell-Funktion verlangt, die „zufallsgesteuert" einen gerichteten Graphen ausgibt. Modifizieren Sie diese Haskell-Funktion so, dass Sie damit ebenfalls zufallsgesteuert einen ungerichteten Graphen erzeugen.

Aufgabe 5.2
Ein Labyrinth kann als ein ungerichteter Graph aufgefasst werden, in dem 2 Knoten ausgezeichnet sind: der Ausgang aus dem Labyrinth und der Knoten (Punkt), von dem aus der Ausweg aus dem Labyrinth gefunden werden soll.

Schreiben Sie eine Haskell-Funktion, die zu einem Labyrinth einen Ausweg bestimmt.

Aufgabe 5.3
Geben Sie eine Haskell-Funktion an, die zu einer endlichen Folge (Liste) von reellen Zahlen den Mittelwert und die Varianz speichereffizient berechnet.

Aufgabe 5.4
Betrachten Sie das folgende Haskell-Programm:

```
prodsum x = prod x + sum x

prod 0 = 1
prod n = n* prod (n-1)

sum 0 = 0
sum n = n + sum (n-1)
```

1. Ändern Sie die Definition von `prod` und `sum` zu restrekursiven Definitionen.
2. Schreiben Sie für die Funktion `prodsum` ebenfalls ein restrekursives Haskell-Programm.

Peter Padawitz / Hubert Wagner Dortmund, den 12.06.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 6

Abgabe: bis 21.06.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 6.1
Gegeben sei ein $n \times n$-Schachbrett ($n \geq 3$).
Schreiben Sie ein Haskell-Programm, das für die Springerfigur eine Zugfolge bestimmt (sofern es eine gibt), bei dem der Springer jedes Schachfeld genau einmal besucht.

Zur Erinnerung: ein Springer bewegt sich entweder um 2 Felder in der vertikalen Richtung und um ein Feld in der horizontalen Richtung oder aber um 2 Felder in der Horizontalen und um ein Feld in der Vertikalen.

Aufgabe 6.2
Schreiben Sie ein Haskell-Programm, das folgendes leistet:

Nach Aufruf des Programms wird eine Zeile eingelesen. Anschließend wird getestet, ob der Zeileninhalt ein Palindrom darstellt und es wird das Ergebnis mitgeteilt. Danach wird wieder eine Zeile eingelesen, der Test auf die Palindromeigenschaft ausgeführt usw. . Das Programm endet, falls eine Leerzeile eingegeben wird.

Aufgabe 6.3
Schreiben Sie ein Haskell-Programm, das den Inhalt einer Textdatei einliest und ihn im Blocksatz formatiert auf dem Bildschirm ausgibt. (Die Anzahl der Zeichen je Zeile können Sie selbst festlegen. Es ist dabei unterstellt, dass die Wortlänge die Zeilenlänge nicht überschreitet.)

Peter Padawitz / Hubert Wagner Dortmund, den 26.06.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 7

Abgabe: bis 03.07.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 7.1
Definieren Sie den Datentyp `Set a` so, dass er die üblichen mengentheoretischen Operationen aufweist. Erweitern Sie ihn außerdem um die Operation setMap (analog zur map-Funktion für Listen) unter Verwendung der Klasse Functor.

Aufgabe 7.2
In dieser Aufgabe verwenden wir das Grafik-Modul SOE, das dem Buch „The Haskell School of Expression" von Paul Hudak zugrundeliegt.
Schreiben Sie ein Haskell-Programm, das das Schneeflocken-Fraktal als Grafik ausgibt. (Siehe Übungausfgabe 3.2 in dem Buch von Hudak)

Aufgabe 7.3
Ein Polynom p des Polynomringes $R[x]$ stellen wir als eine formale Summe $a_0 + a_1x^1 + a_2x^2 + \ldots + a_nx^n$, $n \geq 0$, dar. Die a_i sind dabei Elemente eines Ringes R, in dem die Operationen $+$ für die Additon und $\cdot$ für die Multiplikation zur Verfügung stehen und in dem bezogen auf Addition $+$ und Multiplikation $\cdot$ je ein neutrales Element n bzw. e vorhanden ist.

Definieren Sie eine polymorphe Klasse für Polynomringe und implementieren Sie eine Instanzierung für den Polynomring über den ganzen Zahlen.

Peter Padawitz / Hubert Wagner Dortmund, den 03.07.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 8

Abgabe: bis 10.07.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 8.1
Ein *2-3 Baum* ist ein Baum, bei dem jeder innere Knoten 2 oder 3 Söhne hat und in dem jedes Blatt die gleiche Tiefe aufweist. Eine linear geordnete Menge kann durch einen solchen 2-3 Baum dadurch repräsentiert werden, dass die Elemente dieser Menge den Blättern des 2-3 Baumes zugeordnet werden.

SchreibeN Sie ein Haskell-Programm, das mit Hilfe von 2-3 Bäumen ein Wörterbuch (dictionary) realisiert, das menügesteuert oder per Abfrage die folgenden Funktionen unterstützt:

- ein neues Wörterbuch anlegen
- ein vorhandenes Wörterbuch öffnen
- ein Element in ein geöffnetes Wörterbuch einfügen
- ein Element in einem geöffneten Wörterbuch löschen
- Testen, ob ein Element in einem geöffneten Wörterbuch vorkommt
- Ausgabe des Wörterbuchs

Erproben Sie bei der Entwicklung des Haskell-Programms schon einmal einen Debugger, z.B. Buddha, siehe Webseite:

http://www.cs.mu.oz.au/ bjpop/buddha/

Peter Padawitz / Hubert Wagner Dortmund, den 10.07.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 9

Abgabe: bis 17.07.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 9.1
Eine Registermaschine (kurz: RM) besteht aus einer zentralen Recheneinheit, einem Speicher und aus einem Programm.
Die zentrale Recheneinheit ihrerseits besteht aus zwei Registern: dem Befehlszähler und dem Akkumulator.
Der Speicher enthält unendlich viele Register: $R_1, R_2, \ldots$ mit den Adressen $1, 2, \ldots$. Der Akkumulator hat die Adresse 0.
Ein RM-Programm besteht aus einer endlichen Folge von RM-Befehlen, die mit 1 beginnend aufsteigend durchnummeriert sind. Wir legen die folgenden RM-Befehle zugrunde:

1. Ein- und Ausgabebefehle:
   `LOAD` i
   `STORE` i
2. Arithmetische Befehle:
   `ADD` i
   `SUB` i
   `MULT` i
   `DIV` i
3. Arithmetische Befehle mit Konstanten:
   `CLOAD` i
   `CADD` i
   `CSUB` i
   `CMULT` i
   `CDIV` i
4. Sprungbefehle:
   `GOTO` j
   `JZERO` j
   `END`

Es gibt in diesem Fall also keine indirekte Adressierung.

Schreiben Sie ein Haskell-Programm, mit dem ein RM-Programm, das als Textdatei gespeichert ist, eingelesen und dann anschließend simuliert wird. Das Programm soll insbesondere auch eine Simulation in Einzelschritten ermöglichen, so dass die Arbeitsweise der RM Schritt für Schritt nachvollzogen werden kann. Wünschenswert wäre eine Darstellung der Simulation unter einer grafischen Oberfläche.

Peter Padawitz / Hubert Wagner Dortmund, den 17.07.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 10

Abgabe: bis 24.07.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 10.1
Sei S eine nichtleere Menge. 2^S bezeichne die Potenzmenge von S. Ein Mengenoperator $f : 2^S \to 2^S$ heißt monoton, falls für alle Mengen $X, Y \subseteq S$ gilt: $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$. Eine Menge $X \subseteq 2^S$ heißt Fixpunkt von f, falls $f(X) = X$. X heißt kleinster Fixpunkt von f, falls es keine echte Teilmenge Y von X gibt, die Fixpunkt von f ist.

Sei nun $G = (V, E)$ ein gerichteter Graph. Für die nachfolgende Aufgabenstellung sei vorausgesetzt, dass der Mengenoperator $f : 2^V \to 2^V$ über eine binäre Relation $R \subseteq V \times V$ in der folgenden Weise definiert ist:

$$f(S) = S \cup \{v \in V \mid (\exists u \in S)\,(u, v) \in E \cap R\}$$

Für den hierdurch definierten Mengenoperator schreiben wir auch f_R.

Schreiben Sie nun eine Haskell-Funktion, die folgendes leistet:
Zu einem gerichteten Graphen, der aus einer Textdatei eingelesen werden soll (oder - alternativ - zufällig erzeugt wird) soll interaktiv eine binäre Relation R auf der Knotenmenge des Graphen sowie eine Teilmenge S der Knotenmenge eingegeben werden. Für den durch R festgelegten Mengenoperator f_R soll dann der kleinste Fixpunkt bestimmt werden, der diese Teilmenge S der Knotenmenge umfasst. Die Berechnung des Fixpunktes soll wahlweise auch in Einzelschritten nachvollziehbar sein.

Peter Padawitz / Hubert Wagner Dortmund, den 24.07.2003

Programmierkurs Haskell

Sommersemester 2003

Aufgabenblatt 11

Abgabe: bis 29.07.03 10.00 Uhr in Raum 218(OH16) oder per email (Hubert.Wagner@udo.edu)

Aufgabe 11.1
Schreiben Sie ein Haskell-Programm, das die k-te Potenz einer $n \times n$-Matrix mittels einer „divide-and-conquer" Methode mit $O(\log k)$ Matrizenmultiplikationen berechnet.

Benutzen Sie dann dieses Programm, um die Fibonacci-Funktion in Zeit $O(\log n)$ zu berechnen.
Hinweis:
Für die Fibonacci-Funktion Fib gilt

$$\begin{pmatrix} Fib\,(n+2) \\ Fib\,(n+1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} Fib\,(n+1) \\ Fib\,(n) \end{pmatrix}$$

Aufgabe 11.2
Schreiben Sie ein Haskell-Programm, mit dem das Produkt zweier $n \times n$-Matrizen mit dem Verfahren von Strassen berechnet wird.