Fakultät IV: Elektrotechnik und Informatik
Informatik 1 WS 2001/02,
Jähnichen / Flik / Lechtchinsky / Neunast / Tutoren

Aufgabenblatt 4

letzte Aktualisierung: 15. November, 18:45

Ausgabe: *16.11.2001*
Abgabe: *26./27.11.2001* *Prozent:* *100*

Themen: *Rekursion, Sequenzen von Sequenzen, einfache Algorithmen.*

Der erste von drei Tests findet in den Tutorien am 26./27.11.01 statt.

Hinweise zu den Tests:

Der Tutor entscheidet, ob der Test zu Beginn oder am Ende des Tutoriums geschrieben wird.

Die Bearbeitungszeit beträgt 15 Minuten. Es sind keine Unterlagen zugelassen.
Die Bearbeitung erfolgt einzeln, **nicht** in Gruppen.
Täuschungsversuche werden mit sofortiger Bewertung des Tests als „nicht bestanden“ geahndet.

Grundsätzlich sind für den Erhalt eines Übungsscheins zwei der drei Tests zu bestehen.

Die Themen des ersten Tests orientieren sich am Stoffumfang der Aufgabenblätter 1 bis 4.

Viel Erfolg!

1. Aufgabe (40 Prozent): Sequenzen

1.1. Primfaktorzerlegung (20 Prozent) Deklariert und definiert eine Funktion `primeFactorDecomposition`, die eine natürliche Zahl größer 1 in ihre Primfaktoren zerlegt und diese als Sequenz zurückgibt.

Beispiel: `primeFactorDecomposition`$(314159265) = \langle 3, 3, 5, 7, 127, 7853\rangle$

Hinweis:
Für die Primfaktoren $p_1, \ldots, p_m$ ($m \in \mathbb{N}$ geeignet) von $n \in \mathbb{N}$, $n \geq 2$, gilt:

$$n = \prod_{i=1}^{m} p_i \quad \text{und} \quad p_i = \begin{cases} \min\{k \in \mathbb{N} \mid k \geq 2, k \text{ teilt } n\} & \text{falls } i = 1 \\ \min\{k \in \mathbb{N} \mid k \geq p_{i-1}, k \text{ teilt } \left(n / \prod_{j=1}^{i-1} p_j\right)\} & \text{falls } 1 < i \leq m \end{cases}$$

Ist es erforderlich, in jeder Rekursionsinstanz das ganze Produkt zu bilden? Muß auf der Suche nach dem Minimum jede Zahl geprüft werden oder kann man trivial optimieren?

1.2. Aufteilen einer Sequenz (20 Prozent) Deklariert und definiert eine Funktion `ascendingPrefix`, die eine gegebene Sequenz in zwei Teilsequenzen aufteilt, wobei die erste Teilsequenz alle vom Sequenzanfang an aufsteigenden Elemente (monotones Wachstum) und die zweite Teilsequenz den Rest der Sequenz enthält.

Beispiel: `ascendingPrefix`$(\langle 1, 9, 1, 2, 3, 5, 7, 2, 3, 7\rangle) = (\langle 1, 9\rangle, \langle 1, 2, 3, 5, 7, 2, 3, 7\rangle)$

Hinweis: Diese Aufgabe stammt aus der letzten Klausur (Bearbeitungszeit: ca. 12 Minuten).

2. Aufgabe (20 Prozent): Sequenzen von Sequenzen

2.1. Zerlegen einer Sequenz (Tut) Deklariert und definiert eine Funktion `ascendingParts`, die eine gegebene Sequenz in eine Sequenz von Teilsequenzen aufteilt, so daß ihre Verkettung gleich der Ausgangssequenz ist. Dabei sollen die Teilsequenzen jeweils monoton steigend sein.

Beispiel: `ascendingParts`$(\langle 1, 9, 1, 2, 3, 5, 7, 2, 3, 7\rangle) = \langle\langle 1, 9\rangle, \langle 1, 2, 3, 5, 7\rangle, \langle 2, 3, 7\rangle\rangle$

Hinweis: Die Funktion `ascendingPrefix` aus Aufgabe 1 könnte hilfreich sein.

2.2. Potenzmenge (Tut) Definiert eine Funktion `FUN powerSet: seq[nat] -> seq[seq[nat]]`, die eine Sequenz auf naive Weise als Repräsentation einer Menge auffaßt und ihre Potenzmenge berechnet (warum ist diese Darstellung des Mengenbegriffs im allgemeinen eine schlechte Wahl?).

Beispiel: $\mathcal{P}(\{1, 2, 3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$

Hinweis:

$$\mathcal{P}(M) = \begin{cases} \{\emptyset\} & \text{für } M = \emptyset \\ \mathcal{P}(M \setminus \{x\}) \cup \left(\bigcup_{P \in \mathcal{P}(M \setminus \{x\})} \{P \cup \{x\}\} \right) & \text{für } M \neq \emptyset \text{ mit beliebigem } x \in M \end{cases}$$

2.3. Permutationen einer Sequenz (20 Prozent) Deklariert und definiert eine Funktion `perm`, die alle möglichen Permutationen einer übergebenen Sequenz natürlicher Zahlen generiert. Verwendet **nicht** die Funktion `permutations'SeqOfSeq`.

Beispiel: $\mathbb{P}(\langle 1, 2, 3\rangle) = \{\langle 1, 2, 3\rangle, \langle 2, 1, 3\rangle, \langle 2, 3, 1\rangle, \langle 1, 3, 2\rangle, \langle 3, 1, 2\rangle, \langle 3, 2, 1\rangle\}$

Hinweis:

$$\mathbb{P}(S) = \begin{cases} \{\langle\rangle\} & \text{für } S = \langle\rangle \\ \bigcup\limits_{\substack{P_1, P_2 \\ P \in \mathbb{P}(S') \\ P = P_1.P_2}} \{P_1.x.P_2\} & \text{für } S = x.S' \text{ mit geeignetem } x \text{ und } S' \end{cases}$$

3. Aufgabe (40 Prozent): Matrizen als Sequenzen von Sequenzen

Eine Möglichkeit, Matrizen zu repräsentieren, ist es, sie als Sequenzen von Sequenzen darzustellen, z.B. `seq[seq[nat]]`. Wir treffen die Vereinbarung, daß die „inneren“ Sequenzen den Zeilen entsprechen und daher gleiche Länge haben müssen. Außerdem machen nur Matrizen mit Dimensionen größer Null einen Sinn.

Beispiel: $\langle\langle 1, 2, 3, 4\rangle, \langle 5, 6, 7, 8\rangle, \langle 9, 10, 11, 12\rangle\rangle$ entspricht $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$

und besitzt in Opal die Darstellung

$(1 :: 2 :: 3 :: 4 :: (\langle\rangle)) :: (5 :: 6 :: 7 :: 8 :: (\langle\rangle)) :: (9 :: 10 :: 11 :: 12 :: (\langle\rangle)) :: (\langle\rangle)$

3.1. Matrix-Vektor-Produkt (Tut) Deklariert und definiert eine Funktion `matVecProd`, die eine Matrix mit einem Vektor multipliziert.

Beispiel: $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 30 \\ 70 \\ 110 \end{pmatrix}$

Hinweis: `skalarProd` von Blatt 3 steht zur Verfügung.

3.2. Matrix-Transposition (20 Prozent) Deklariert und definiert eine Funktion `transp`, die eine Matrix transponiert. Verwendet **nicht** die Funktion `transpose'SeqOfSeq`.

Beispiel: $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix}$

3.3. Matrix-Matrix-Produkt (20 Prozent) Deklariert und definiert eine Funktion `matMatProd`, die zwei Matrizen multipliziert.

Beispiel: $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} = \begin{pmatrix} 70 & 80 & 90 \\ 158 & 184 & 210 \\ 246 & 288 & 330 \end{pmatrix}$

Hinweis:
Die mathematische Definition der Ergebnismatrix lautet:

$$C_{m\times n} = A_{m\times l} \cdot B_{l\times n} \quad \text{mit} \quad c_{ij} = \sum_{k=1}^{l} a_{ik} \cdot b_{kj}$$

Nützlicher ist allerdings die Erkenntnis, daß jede Spalte von C durch Bilden des Matrix-Vektor-Produkts von A und der korrespondierenden Spalte von B gebildet wird. Beim Versuch, das umzusetzen, wird man vermutlich auf die Idee kommen, zweimalige Transposition zu Hilfe zu nehmen (warum?). Mit ein bißchen Überlegung genügt aber auch die einmalige Transposition (wie geht das?).

Falls ihr Aufgabe 3.2 nicht lösen konntet, verwendet `transpose'SeqOfSeq`, um diese Aufgabe zu lösen, ansonsten natürlich das eigene `transp`.