Winfried Auzinger
Klaus Felsenstein
Dirk Praetorius

SS 2010
(12. März 2010; korrigierte Fassung vom 16.März)

Übungsaufgaben zur VL Computermathematik

Serie 2

Die Aufgaben mit Stern (*) sind bis zur Übung in der kommenden Woche fix vorzubereiten und werden dort abgeprüft (Minimalerfordernis).

Kopieren Sie Ihre Worksheets auf Ihren Account auf `lva.student.tuwien.ac.at`*. Überprüfen Sie vor der Übung, ob Ihre Codes unter Maple 12 auf* `lva` *einwandfrei funktionieren. Es wird empfohlen, für jedes Beispiel im Verzeichnis* `serie02` *ein eigenes Worksheet mit dem Namen* `aufgabey-y.mw` *anzulegen. Alle zu verfassenden Prozeduren sind zu kommentieren und – je nach Angabe – mit verschiedenen Werten für die Parameter auszutesten.*

Verwenden Sie konsequent die Online-Hilfe – das ist wichtig für den erfolgreichen praktischen Umgang mit Maple. Insbesondere kommt es immer wieder vor, dass etwas in der Vorlesung [noch] nicht im Detail besprochen wurde – dann muss man sich zu helfen wissen. Explizite Verweise auf die Hilfe (z.B. `?set`*) deuten an, dass man sich die entsprechende Hilfe-Seite auf jeden Fall ansehen sollte. Manchen Aufgaben dienen auch der Vertiefung von Themen, die in der Vorlesung behandelt wurden; auch weitere Themen werden angesprochen (mit entsprechenden Hinweisen).*

Manche der Aufgaben widmen sich dem prinzipiellen Umgang mit Prozeduren und Datenstrukturen; andere dienen einem konkreten mathematischen Zweck.

Aufgabe 2.1*. Schreiben Sie eine Prozedur `convert_list_to_table(l)`, die eine Liste `l` in eine Tabelle umwandelt. Als Tabellen-Indizes verwende man aber nicht denjeweiligen Index `k` des `k`-ten Listenelementes, sondern den Wert `f(k)`, wobei `f` irgendeine Funktion ist, die Sie innerhalb der Prozedur `convert_list_to_table` definieren (das geht).

So ist das nicht sehr flexibel. Besser (2. Variante): `convert_list_to_table_f(l,f)`, wobei der Name der betreffenden Funktion `f` als Parameter mit übergeben wird.

Aufgabe 2.2*. Schreiben Sie eine Prozedur `auzprod(A,B,type)`, die das sogenannte Auzinger-Produkt 1. bzw. 2. Typs zweier Mengen zurückliefert: [1]

$$\text{auzprod}(A,B,1) := \{(a,b) \in A \times B \text{ mit der Eigenschaft } a \in B \text{ und } b \in A\},$$
$$\text{auzprod}(A,B,2) := \{(a,b) \in A \times B \text{ mit der Eigenschaft } a \in B \text{ oder } b \in A\}.$$

Die geordneten Paare (a,b) werden als Listen `[a,b]` der Länge 2 dargestellt.

Aufgabe 2.3*. Gegeben seien n reelle Datenpaare (x_i, y_i), $i = 1 \ldots n$, mit paarweise verschiedenen x_i. Zu berechnen sind die Parameter a, b der *Ausgleichsgerade* $g(x) = ax + b$, festgelegt durch die Minimalbedingung

$$\varphi(a,b) := \sum_{i=1}^{n} (g(x_i) - y_i)^2 \quad \rightarrow \quad \text{minimal!}$$

Schreiben Sie eine Prozedur `ausgleichsgerade(x,y::list)`, die die betreffenden optimalen Parameter a und b zurückliefert. (Diese sind für $n \geq 2$ durch die Bedingung $\frac{\partial \varphi}{\partial a} = 0$ und $\frac{\partial \varphi}{\partial b} = 0$ eindeutig festgelegt.)

Aufgabe 2.4*. Schreiben Sie eine Prozedur `merge_lists(l1,l2)`, die zwei als aufsteigend sortiert angenommene Listen `l1` und `l2` zu einer neuen aufsteigend sortierten Liste zusammenfügt, die alle Elemente von `l1` und `l2` enthält.

---

[1] G.G. Auzinger, *Set theory revolutionized*, Journal of odd and useless sets 1 (2003), pp. 1–200.

Aufgabe 2.5. Schreiben Sie zwei Maple-Prozeduren `innerprod3(a,b)` und `outerprod3(a,b)`, die zwei Listen `a` und `b` der Länge 3 als Vektoren im $\mathbb{R}^3$ interpretieren und das innere (Skalarprodukt, $a \cdot b$) bzw. das äußere (Vektorprodukt, $a \times b$, letzteres wieder als Liste) zurückgeben. Falls `a` oder `b` nicht die Länge 3 hat, ist jeweils `NULL` zurückzugeben.

Versuchen Sie damit zu 'beweisen' dass gilt: $a \times b \perp a$, $a \times b \perp b$, sowie die Gültigkeit der *Lagrange-Identität*

$$(a \times b) \cdot (c \times d) = (a \cdot c)(b \cdot d) - (b \cdot c)(a \cdot d)$$

indem Sie beliebige 'generische' Vektoren (ohne konkrete Werte) übergeben.

Aufgabe 2.6. Manche Operationen an Listen sind ein wenig umständlich, z.B. Hinzufügen eines Elementes:

```
> l := [1,zwei,'drei'];
                          l := [1, zwei, drei]
> l[4] := "vier";
                          Error, out of bound assignment to a list
> l:=[op(l),"vier"];
                          l := [1, zwei, drei, "vier"]
```

Schreiben Sie

- eine Prozedur `ladd(l,entries,pos)`, die die Elemente einer Liste `entries` in eine Liste `l` vor der `pos`-ten Position einfügt und die entsprechend verlängerte Liste zurückgibt;
- eine Prozedur `lcut(l,anzahl,pos)`, die `anzahl` Elemente vor der `pos`-ten Position entfernt und die entsprechend verkürzte Liste zurückgibt.

Für `pos <= 0` oder `pos >` Länge von `l` soll `l` unverändert zurückgegeben werden.

Zu beachten: Die Anzahl der Elemente einer Liste erhält man mittels `nops(...)`. (`length(...)` ist etwas anderes.) Siehe auch Aufgabe 2.7.

Aufgabe 2.7. Studieren Sie die Dokumentation des packages `ListTools` und bereiten Sie ein Worksheet vor, das die Funktionalität der darin verfügbaren Tools anhand einiger Beispiele illustriert.

Aufgabe 2.8. Das 'Game of Life' von John Conway ist ein sogenannter zellulärer Automat, ein einfaches Modell für die Entwicklung einer Population. Unsere Population 'lebt' in einem `Array` der Dimension `0..m,0..n`; ein Eintrag `'X'` bezeichne eine lebendige Zelle, `' '` (Leerzeichen) eine tote Zelle. Gegeben ist irgendein Anfangszustand, und dann entwickelt sich die Population in diskreten Zeitschritten nach folgenden Regeln:

The Rules:

For a space that is alive (populated):

- *Each cell with one or no neighbors dies, as if by loneliness.*
- *Each cell with four or more neighbors dies, as if by overpopulation.*
- *Each cell with two or three neighbors survives.*

For a space that is empty (unpopulated):

- *Each cell with three neighbors becomes alive.*

Für das Verhalten am Rand nehmen wir Periodizität in beiden Dimensionen an, d.h. wir identifizieren den Index `m+1` bzw. `n+1` mit dem Index 0. [2] (Topologisch betrachtet entspricht unsere Welt also einem Torus.)

Man verwende zwei Arrays und simuliere die Entwicklung der Population nach diesen Regeln, wobei man in einem Array immer den alten Zustand speichert und davon ausgehend in dem zweiten Array den neuen Zustand generiert. Das Ganze wird in eine Prozedur `GameofLife(P,steps)` verpackt, die den Anfangszustand `P` übernimmt, `steps` Zeitschritte ausführt und das Endergebnis als Resultat zurückliefert.

[2] Überlegen Sie sich etwas für die vier Zellen in den Ecken.