HUMBOLDT-UNIVERSITÄT ZU BERLIN
INSTITUT FÜR INFORMATIK

Sommersemester 2010

Prof. Dr. Holger Schlingloff
Dr. Anatolij Zubow

Übungen zu Grundlagen der Programmierung

Übungsblatt Nr. 4 (30 Punkte)

Ausgabedatum: 31. 5. 2010, 17:00 – Abgabedatum: 14. 6. 2010, 8:00

Die Lösung der Aufgaben soll in Gruppen zu je zwei Studierenden erfolgen. Die Aufgaben sind elektronisch abzugeben. Auf jedem Lösungsblatt sind Name und Matrikelnummer der beiden Gruppenmitglieder anzugeben! Von jedem Blatt werden mindestens 20% der Punkte (also 6) verlangt.

Aufgabe 1 (8 Punkte)

Im Folgenden sollen die folgenden Matrix-Operationen in Groovy/Java implementiert werden. Addition, Subtraktion, Multiplikation sowie die Berechnung der transponierten Matrix:

a.) `double[][] add(double[][] M1, double[][] M2)`
b.) `double[][] sub(double[][] M1, double[][] M2)`
c.) `double[][] mul(double[][] M1, double[][] M2)`
d.) `double[][] transpose(double[][] M)`

Hinweise

Die oben genannten Matrix-Operationen gelten natürlich auch für Skalare (1x1-Matrizen). Wenn die Größen der Matrizen für die Matrix-Operation nicht kompatibel sind, soll eine Fehlermeldung an den Nutzer ausgegeben werden. Eine Ausnahme bildet hier der Fall von Skalar-Matrix-Operationen (für Addition, Subtraktion, Multiplikation sowie Division), wo jeder Eintrag der Matrix mit dem Skalar verrechnet wird.
Ferner beschränken wir uns auf reelle Matrizen.

Aufgabe 2 (6 Punkte)

Schreiben Sie ein Programm `Permutations.{groovy|java}` welches ein Kommandozeilenargument N übergeben bekommt und alle N!-viele Permutationen der ersten N Buchstaben des Alphabets ausgibt (Sie dürfen annehmen, dass N nicht größer als 26 ist). Eine Permutation von N Elementen ist eine von N! möglichen Anordnungen der N Elemente.

Ein Beispiel: für N = 3 sollte Ihr Programm die folgenden 3! = 6 Permutationen als Ausgabe erzeugen (die Reihenfolge, in der die 6 Permutationen erzeugt werden, spielt dabei keine Rolle):

```
% groovy Permutations.groovy 3
bca cba cab acb bac abc
```

Aufgabe 3 (8 Punkte)

Im λ-Kalkül sind die arithmetischen Funktionen *Plus* und *Mult* wie folgt definiert:

$$\textbf{plus} \equiv \lambda m.\lambda n.\lambda f.\lambda x.\ m\ f\ (n\ f\ x)$$

bzw.

$$\textbf{mult} \equiv \lambda m.\lambda n.\lambda f.\ n\ (m\ f)$$

Hierbei sind m und n die zwei Zahlen und f und x die Parameter der entstehenden Zahl.
Zeigen Sie, dass $z_2 + z_3 = z_5$ bzw. $z_2 * z_3 = z_6$ ist. Geben Sie alle Ableitungsschritte an!

Aufgabe 4 (8 Punkte)

Das Rucksackproblem (*Knapsack*) ist ein Optimierungsproblem der Kombinatorik. Aus einer Menge von Objekten, die jeweils ein Gewicht und einen Nutzenwert haben, soll eine Teilmenge ausgewählt werden, deren Gesamtgewicht eine vorgegebene Gewichtsschranke nicht überschreitet. Unter dieser Bedingung soll der Nutzenwert der ausgewählten Objekte maximiert werden.

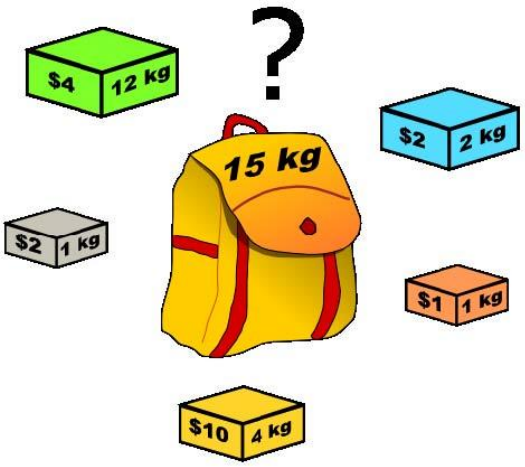

Abbildung 1. Welche der Gewichte können in den Rucksack mit Maximallast von 15 kg gepackt werden, so dass der Geldwert maximal wird? (Lösung in diesem Fall: Alle Gewichte außer dem schwersten einpacken.) Quelle: Wikipedia.org

Problemformulierung

Gegeben sind eine endliche Menge von Objekten U. Durch eine Gewichtsfunktion w und der Nutzenfunktion v wird den Objekten ein Gewicht und ein Nutzenwert zugeordnet: $w\colon U \to R$, $v\colon U \to R$
Des Weiteren gibt es eine vorgegebene Gewichtsschranke: $B \in R$.
Gesucht ist eine Teilmenge $K \subseteq R$, die die Bedingung $\sum_{u \in K} w(u) \leq B$ einhält und die Zielfunktion maximiert: $\sum_{u \in K} v(u)$.

Aufgabe

a.) (8 Punkte) Implementieren Sie das Rucksackproblem mit Hilfe eines Groovy/Java-Programms. Verwenden Sie Rekursion und geben Sie die Komplexität Ihrer Lösung an.
b.) (**Zusatz**) Können Sie auch eine iterative Lösung finden?

Hinweise

Die Beschreibung findet sich unter de.wikipedia.org/wiki/Rucksackproblem. Die Kapazität C des Rucksacks ist ein frei wählbarer Parameter. Sowohl die Gewichte w der Gegenstände als auch deren Wertigkeit v sind konfigurierbar. Beide sollen in einem Feld verwaltet werden.