Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
Übungsblatt 2
19. Mai 2006

Übungen zu Einführung in die Informatik II

Aufgabe 5 Zählen von Zeichen-Häufigkeiten

Ziel der Aufgabe ist es eine Funktion *generate_charcntlist* zu entwerfen, welche die Häufigkeiten der Zeichen in einem gegebenen Eingabe-String zählt und diese als `charcntlist` über Paare der Form: [(Zeichen, Häufigkeit), (Zeichen, Häufigkeit), ...], also z.B. [('h',55);('a',45);('e',68)], als Ergebnis zurückgibt.

a) Schreiben Sie hierzu zunächst eine OCaml-Funktion *add_char_to_charcntlist*, welche ein einzelnes Zeichen in eine vorhandene `charcntlist` einträgt, und die derart modifizierte Liste zurückgibt.

b) Entwerfen Sie nun die Funktion *generate_charcntlist*, welche den auszuwertenden String als Eingabe erwartet, diesen mittels der *list_of_string*-Funktion (vgl. Übungsblatt 3) in eine Liste von Zeichen überführt, und diese Zeichen für Zeichen mittels *add_char_to_charcntlist* auswertet. Verifizieren Sie Ihre Methode an folgender einfacher Zeichenkette: `abbaccabba`

Aufgabe 6 Generierung von Huffmanbäumen

Ein Huffman-Baum sei durch folgende Definition gegeben:

```
type hufftree = Atom of (char * int) | Node of (hufftree * hufftree);;
```

a) Entwickeln Sie eine Funktion *treesum*, welche, ausgehend von einem gegebenen `hufftree`, die Summe der Zeichen-Häufigkeiten in den Atomen berechnet.

b) Entwickeln Sie eine Funktion *hufftreelist_of_charcntlist*, welche die Elemente einer gem. der vorhergehenden Aufgabe generierten `charcntlist` zu Atomen aus `hufftree` konvertiert, und diese zu einer neuen Liste zusammenfügt.

c) Entwickeln Sie eine Funktion *combine_min*, welche die beiden Elemente mit der geringsten *treesum* einer gem. b) aufgestellten Liste zu einem neuen `hufftree` zusammenfaßt, und diesen statt derer in die Liste einträgt. Rückgabe der Funktion sei folglich die entsprechend verkürzte Eingangs-Liste.
Gehen Sie hierbei davon aus, dass Ihnen eine Funktion *sort_hufftreelist* zur Verfügung steht, welche eine `hufftreelist` als Eingabe erwartet, und diese der *treesum* nach (kleinstes Element zuerst) zurückgibt.

d) **(H)** Entwickeln Sie nun eine Funktion *generate_hufftree*, welche aus den gem. b) definierten Atom-Listen einen zugehörigen Huffmanbaum generiert.

Aufgabe 7 Optimale Berechnung von Kontonummern (P)

Die Aufgabe bezieht sich auf die in Blatt 1 → Aufgabe 2 vermittelte Theorie zum Hammingabstand. Es soll nun ein Programm geschrieben werden, das eine Menge von 15 Kontonummern berechnet, deren Hammingabstand ≥ 3 ist. Außerdem wird eine Startzahl (1000) vorgegeben, die in der Menge enthalten ist, aber nicht notwendigerweise die kleinste Zahl in der Menge sein muss. Die größte Zahl in der Menge soll jedoch so klein wie möglich sein.

a) Schreiben Sie zunächst eine Funktion *int hamDist(int a, int b)* in *Java* die den Hammingabstand zweier *Integer*-Zahlen bestimmt und das Ergebnis als *Integer*-Wert zurückgibt. Beachten Sie dabei, dass die beiden Zahlen evtl. unterschiedlich lang sind und Sie die kleinere mit führenden Nullen auffüllen müssen. Testen Sie die Funktion mit einer kurzen *main*-Routine, indem Sie dort einfach den Hammingabstand von einigen Zahlen berechnen.

b) Implementieren Sie nun ein *Java*-Programm, das unter Verwendung der *hamDist*-Funktion die Kontonummern nach obiger Vorgabe erzeugt und am Bildschirm ausgibt.

c) Erweitern Sie das Programm so, dass der Benutzer die Anzahl der Kontonummern und die Startzahl vorgeben kann. Außerdem sollen die ausgegebenen Kontonummern so formatiert sein, dass alle dieselbe Länge haben; d.h. zu kurze Zahlen müssen Sie mit führenden Nullen auffüllen!