Universität des Saarlandes
FR 6.2 Informatik

Prof. Dr. Kurt Mehlhorn, Dr. Konstantinos Panagiotou **WiSe 2011/2012**

Übungen zu Computational Thinking

http://www.mpi-inf.mpg.de/departments/d1/teaching/ws11/ct/

Blatt 5 Abgabeschluss: 5. 12. 11 16:00

Regeln: Bis zum Semesterende müssen mindestens 42% der maximal erreichbaren Punkte aller Übungszettel erworben werden.
Programmcode ist elektronisch per E-Mail abzugeben. Zusätzlich müssen die Ausgaben einer exemplarischen Programmausführung mitgeliefert werden.

Bleistiftaufgaben

Aufgabe 1 (*25 Punkte*) Angenommen Sie haben eine spezielle Schutzverpackung für Eier entwickelt und möchten nun feststellen, wie widerstandsfähig sie Eier macht. Das Hauptquartier ihrer Firma ist ein 36 stöckiges Hochhaus. Sie wollen wissen, bis zu welchem Stockwerk ein Ei einen Fall überstehen kann. Vereinfachend machen wir folgende Annahmen:

- Eier sind alle vollkommen identisch, wenn ein Ei bei einem Sturz zerbricht, so tun es auch alle anderen Eier.
- Eier, die nicht zerbrechen, nehmen keinen anderen Schaden, werden also insbesondere nicht zerbrechlicher.
- Wenn ein Ei einen Sturz aus einem Stockwerk übersteht, so übersteht es auch alle tieferen Stockwerke; umgekehrtes gilt für Zerbrechen.

a) Wenn nur ein Ei verfügbar ist, müssen wir vom ersten Stock anfangen und alle Stockwerke durchprobieren. Angenommen wir haben zwei Eier, was ist die minimale Anzahl von Versuchen, die wir brauchen um das höchste sichere Stockwerk zu finden?

b) Gegeben sind zwei aufsteigend sortierte Listen von ganzen Zahlen. Wir wollen die Elemente finden, die beiden Listen gemeinsam sind. Nehmen sie an, dass eine der Listen sehr viel länger ist als die andere, etwa 10^6 Elemente in der ersten Liste und 10^2 Elemente in der zweiten Liste. Sie können Listen nur von vorne nach hinten lesen (Binärsuche in einer Liste ist also nicht möglich). Sie können aber Listen in Blöcke von 1000 Elementen einteilen und das erste Element des nächsten Blocks schnell finden. Was würden Sie tun und was ist der Zusammenhang mit den Eiern?

Aufgabe 2 (*5 Punkte*) Kann es ein Komprimierungsverfahren geben, dass alle möglichen Zeichenketten verkleinern kann? Begründen Sie ihre Antwort kurz.

Python

Aufgabe 3 (*5 Punkte*) Run-Length Encoding ist ein einfaches Komprimierverfahren. Der zu komprierende Text wird nach sich wiederholenden Buchstabensequenzen durchsucht. Eine solche Sequenz wird dann durch die Anzahl der Wiederholungen und einen Buchstaben kodiert. `rle("aaabbbccc")` → `(3,a),(3,b),(3,c)`.
Schreiben Sie eine Funktion, die einen String so kodiert und eine weitere Funktion, die den String wieder dekodiert. Für welche Art Daten ist dieses Verfahren besonders geeignet?

Von den folgenden Aufgaben können Sie sich eine aussuchen.

Aufgabe 4 (*25 Punkte*) Schreiben Sie ein Programm, das die Sprache eines Textes erkennt. Gehen Sie in drei Schritten vor:

a) Schreiben Sie eine Funktion, die eine Textdatei einliest und ein Wörterbuch erstellt, das die Frequenz des Vorkommens von n-Grammen, also die Frequenz von Buchstaben ($n = 1$), Buchstabenpaaren ($n = 2$), -tripeln ($n = 3$), etc, ausrechnet. Die Zahl n soll ein Parameter der Funktion sein. In deutschen Texten sollte 'e' zum Beispiel etwa 17% der Buchstaben ausmachen, 'er' etwa 13% der Buchstabenpaare.

b) Schreiben Sie eine Funktion, die den quadratischen Abstand zweier Frequenzwörterbücher ausrechnet. Also für die Wörterbücher w_1, w_2 mit der Schlüsselmenge X die Summe
$$\sum_{c \in X} (w_1[c] - w_2[c])^2.$$

c) Sammeln Sie sich ein Beispielset von Texten aus bekannten Sprachen (mindestens zwei), zum Beispiel Wikipediaartikel. Für einen unbekannten Text soll ihr Programm die Sprache des Beispieltextes ausgeben, dessen n-Grammfrequenzen des kleinsten quadratischen Abstand zu den Frequenzen des zu klassifizierenden Textes haben.

Probieren Sie Ihr Programm für verschiedene Texte und verschiedene Werte von n aus. Was ist der beste Wert für n auf Ihrem Testset, was ist die durchschnittliche Wortlänge der Texte?

Aufgabe 5 (*25 Punkte*) Implementieren Sie Huffman-Komprimierung.

a) Schreiben Sie geeignete Klassen, um einen Huffman-Baum zu konstruieren.

b) Schreiben Sie eine Funktion, die aus einem Text einen Huffman-Baum erzeugt.

c) Schreiben Sie eine Funktion, die einen Huffman-Baum in ein Wörterbuch von Buchstaben nach Kodierungen übersetzt.

d) Schreiben Sie Funktionen um einen Text zu kodieren und wieder zu dekodieren. Achten Sie darauf, dass eine geeignete Repräsentation der Kodierungstabelle mit gespeichert wird. Dafür können Sie zum Beispiel das Python-Modul `pickle` verwenden[1]. Für diese Aufgabe reicht es, 0 und 1 in die Datei zu schreiben. Eine (tatsächliche) binäre Kodierung ist nicht nötig.

[1] `http://docs.python.org/library/pickle.html`

Sie können Ihr Programm auf den gesammelten Werken von Shakespeare[2] ausprobieren. Wieviele Bits braucht ihre Kodierung (ohne die Kodierungstabelle)?

[2] http://www.gutenberg.org/files/100/100.txt