Übungen zur Datenkompression **Sommersemester 2011**
WSI für Informatik **Universität Tübingen**
Krebs/Deininger **14.4.2011**

Blatt 1

(Abgabe am 5.5.2011)

Beachten Sie bitte die Hinweise zu den Übungen!

Aufgabe 1 (Entropie)
Eine Informationsquelle sende Buchstaben aus dem Alphabet $\Sigma = \{a, b, c, d, e\}$ mit den Wahrscheinlichkeiten $P(a) = 0,15$, $P(b) = 0,04$, $P(c) = 0,26$, $P(d) = 0,05$ und $P(e) = 0,5$.

1. Berechnen Sie die Entropie der Quelle!
2. Bestimmen Sie einen Huffmann-Code für diese Quelle per Hand!

Aufgabe 2 (Programmieraufgaben)
Bitte beachten Sie die Hinweise zu den Programmieraufgaben.

Schreiben Sie Routinen, die folgende Aufgaben erfüllen:

1. Einfache Statistik: Gibt an, welche Bytes in einer Datei vorkommen und wie oft jedes Byte vorkommt.
2. Entropie: Eingabe eine Liste von Zeichenhäufigkeiten einer Quelle, Ausgabe die Entropie der Quelle.
3. Entropie: Berechnet die Entropie einer Datei, wobei als Zeichen Bytes verwendet werden.
4. (*) Erweitern Sie obige Routinen, so dass sie auch mit Alphabeten von n Bytes arbeiteten. Hat die Eingabe nicht eine Länge, die ein vielfaches von n ist, so padden sie sie mit 0-Bytes.

Aufgabe 3 (Empirische Tests)

1. (*) Generieren Sie eine Datei, bei der der Kompressionsquotient größer als 1 ist!
2. Laden sie die Beispieldateien zum Übungsblatt herunter und nutzen Sie Ihr Programm um die Entropien der Dateien byteweise zu ermitteln!
3. Berechnen Sie mit Hilfe der Entropie die erwarteten minimalen Codelängen der `*.tex`-Dateien. Wenden sie ein Programm wie `gzip` auf diese Dateien an und Vergleichen sie deren Größe mit der erwarteten minimalen Codelänge.

   Interpretieren Sie das Ergebnis!
4. (*) Ermitteln Sie die Entropien der Dateien aus Teil 2 für Alphabetgrößen à 3 und 5 Byte!

Aufgabe 4 (Arithmetisches Kodieren)
In dieser Aufgabe greifen wir dem Stoff ein wenig vor. Ihre Aufgabe ist es, Funktionen zum arithmetschen Kodieren eines Strings zu entwerfen.

Bei der arithmetischen Kodierung wird einer Folge von Zeichen eine Dezimalzahl im Intervall $[0, 1)$ bzw. ein Bruch zugeordnet. Dazu wird jedem Zeichen Anhand der Häufigkeit ein Bereich in dem Intervall zugeordnet, dann wird beim Einlesen eines Zeichens das Startintervall auf den Bereich zusammengeschrumpft. Das neue Intervall wird nun wieder in Bereiche unterteilt, und der Algorithmus wird

rekursiv angewandt. Ist das letzte Zeichen eingelesen, wird eine beliebige Zahl im übriggebliebenen Intervall als Ausgabe gewählt, zusammen mit der Anzahl der Zeichen die eingelesen wurden.

Decodierung erfolgt analog: Mit der Ausgabe und der Anzahl der Zeichen wird überprüft in welchem Teilintervall von $[0, 1)$ die Zahl liegt. Dann wird das Zeichen dieses Teilintervalls an die Ausgabe gehängt, das Teilintervall wird nun betrachtet, wieder in Bereiche aufgeteilt und der Algorithmus wird rekursiv angewendet, bis die Anzahl der gelesenen Zeichen der Länge der Ausgabe entspricht.

Beispiel: Zeichensatz a, b, Eingabe zu Codieren: 'abba'. Das Intervall wird aufgeteilt in zwei Hälften, die untere Hälfte für a, die obere für b. Wir fangen an mit Intervall $[0, 1)$, lesen Zeichen 'a'. Das neue Intervall wird daher $[0, 0.5)$. Das nächste Zeichen ist 'b', daher wird das neue Intervall $[0.25, 0.5)$. Wieder 'b', führt zu $[0.375, 0.5)$. Das letzte Zeichen, 'a' führt zum Intervall $[0.375, 0.4375)$. Aus diesem Intervall wählen wir jetzt eine beliebige Zahl als Rückgabewert, z.B. 0.375, die Länge der Codierten Zeichen ist damit 4.

Rückrichtung: gleicher Zeichensatz und Verteilung, Zahl ist 0.375, Länge ist 4. Wir fangen an mit dem Intervall $[0, 1)$. 0.375 ist in der unteren Hälfte, d.h. zur Ausgabe kommt 'a' hinzu und das neue Intervall ist $[0, 0.5)$. 0.375 ist hier in der oberen Hälfte, d.h. ein 'b' kommt zur Ausgabe und wir erhalten $[0.25, 0.5)$. Wir wenden das weiter an, bis wir die Länge der Ausgabe erhalten haben, da sind wir wieder im Intervall $[0.375, 0.4375)$, die Ausgabe ist 'abba'.

Für Ihre Implementation dürfen Sie den Zeichensatz einschränken, soweit Sie dies dokumentieren. Alle lateinischen Buchstaben in Standard-ASCII sollten allerdings vertreten sein. Verwenden Sie eine gleichmässige Verteilung der Zeichen auf das Intervall zur Vereinfachung der Implementation.

Verwenden Sie für die Rückgabewerte entsprechend Ihrer Programmiersprache typische Konstrukte: In Java bietet es sich an, für den Rückgabewert eine Klasse zu definieren. In C oder Pascal währen Structs bzw. Records sinnvoll, in LISP-Dialekten stattdessen Listen.

1. (*) Schreiben Sie Routinen zur Codierung des Strings: Gegeben ein String `s` soll die Funktion `EncodeArithmetic` entweder eine Dezimalzahl im Intervall $[0, 1)$ oder einen Bruch zurückgeben, der wie beschrieben codiert ist. Weiterhin soll die Funktion die Länge der Eingabe zurückgeben.
2. (*) Schreiben Sie Routinen zur Decodierung: Gegeben dem Rückgabewert der `EncodeArithmetic` Funktion soll die `DecodeArithmetic` Funktion den String rekonstruieren.
3. (*) Überprüfen Sie Ihre Implementation mit kurzen und langen Strings.
4. (*) Überlegen Sie sich, welche Probleme bei der Implementierung im Computer auftreten können.
5. (*) Überlegen Sie sich, wie Sie das Verfahren von dieser vereinfachten arithmetischen Codierung verbessern können. Überlegen Sie sich dabei getrennt Verbesserungen in Ausgabegrösse und Codierungs-/Decodierungsaufwand.

Hinweise zu den Programmieraufgaben

Die für die Programmieraufgaben vorgesehene Programmiersprache ist Java. Wenn Sie die Übungen in einer anderen Programmiersprache abgeben wollen, so wenden Sie sich an den Betreuer der Übungsgruppen.

Allgemein: Die Abgaben erfolgen elektronisch in Form von gut dokumentiertem und compilierfähigen Quellcode.

Sofern nicht anders angegeben, sollen die geforderten Programme die zu bearbeitende Datei als ersten Kommandozeilenparameter erhalten.

Die auf diesem Blatt erstellten Routinen werden in den weiteren Übungen gebraucht werden. Erstellen Sie die in Aufgabe 2 geforderten Aufgaben in Bibliotheksform und erstellen Sie mit deren Hilfe die für Aufgabe 3 geforderten Programme.

Nutzen Sie für das Einlesen der Datei in Java die Klasse `FileInputStream`, um die Datei in Form von Bytes zu einzulesen.

Unter Windows können die Ergebnisse wegen der unterschiedlichen Darstellung des Zeilenendsymbols abweichen.