LEHRSTUHL FÜR RECHNERTECHNIK UND RECHNERORGANISATION

# Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Projektaufgabe – Aufgabenbereich Theorie der Informationsverarbeitung (A404)

## 1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung [1] festgesetzt, die Sie auch über die Praktikumshomepage [2] aufrufen können.

Der **Abgabetermin** ist der **1. Februar 2016 (23:59 MEZ)**. Die Abgabe erfolgt per Git in das im Gitlab für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie unbedingt die in der Praktikumsordnung angebene Liste von abzugebenden Dateien!

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. **Der erste Teil Ihrer Projektpräsentation** ist eine kurze Vorstellung Ihrer Aufgabe in einer Tutorübung in der Woche **12.12.2016-16.12.2016**. Wählen Sie bitte eine Übung, in der mindestens ein Team-Mitglied angemeldet ist.
Erscheinen Sie bitte **mit allen Team-Mitgliedern** und bereiten Sie einen Kurzvortrag mit folgenden Inhalten vor:

- 1 Folie: Vorstellung der Team-Mitglieder
- 2 Folien: Zusammenfassung der Aufgabenstellung

In der Übung wird eine Beamer vorhanden sein, an den Sie Ihr eigenes Notebook anschließen können. Alternativ können Sie auch Ihre Präsentation als PDF Datei auf einem USB-Stick mitbringen. Der Kurzvortrag wird von einer Person gehalten.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihren Tutor.

Mit freundlichen Grüßen
Die Übungsleitung

PS: Vergessen Sie nicht, sich rechtzeitig in TUMonline zur Prüfung anzumelden. Dies ist Voraussetzung für eine erfolgreiche Teilnahme am Praktikum im laufenden Semester.

[1] `http://wwwi10.lrr.in.tum.de/~buettned/GEP-ASP/GEP-ASP-Praktikumsordnung.pdf`
[2] `http://www.lrr.in.tum.de/lehre/`

# 2 Huffmankodierung

## 2.1 Überblick

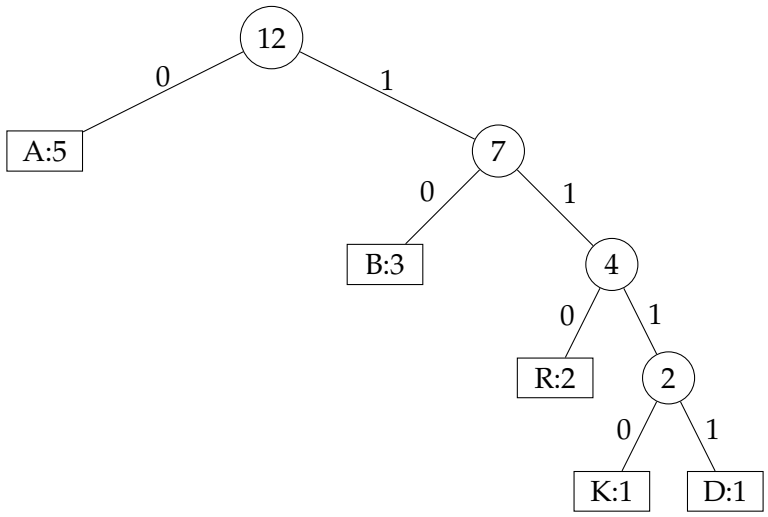
Die Informationstheorie ist ein Feld der Mathematik, in welchem man sich allgemein mit dem Kodieren von Nachrichten aufgrund von statistischen Gegebenheiten beschäftigt. Sie werden in Ihrer Projektaufgabe einen bestimmten Teilbereich näher beleuchten und einen bestimmten Algorithmus auf dem BeagleBoard in Assembler implementieren.

## 2.2 Funktionsweise

Die Huffmankodierung ist ein zur Datenkompression eingesetztes Entropiekodierungsschema. Bei der Kompression wird zunächst eine Häufigkeitsanalyse der zu kodierenden Nachricht durchgeführt. Anschließend wird ein sogenannter Huffmanbaum erstellt, der Symbolen entsprechend ihren Häufigkeiten unterschiedliche Bitsequenzen zuweist. Wir betrachten dazu ein einfaches Beispiel mithilfe des Wortes *ABRAKADABRAB*.
Zunächst führt man die Häufigkeitsanalyse durch:

| Zeichen | A | B | R | K | D |
|---|---|---|---|---|---|
| Häufigkeit | 5 | 3 | 2 | 1 | 1 |

Anschließend konstruiert man den Huffmanbaum als präfix-freien Baum, in welchem die häufigsten Symbole die kürzesten Codewörter zugewiesen bekommen.




Der fertige Baum kann dann verwendet werden, um mit den einzelnen Bits an den Ästen die Eingabe binär zu kodieren. Dazu erstellt man ein sogenanntes *Dictionary*:

| Symbol | Kodierung |
|---|---|
| A | 0 |
| B | 10 |
| R | 110 |
| K | 1110 |
| D | 1111 |

Das gewählte Beispiel wird dann mithilfe der Tabelle buchstabenweise kodiert:

```
ABRAKADABRAB = 0 10 110 0 1110 0 1111 0 10 110 0 10
```

Beachten Sie, dass die Gruppierung der Bits nur zur visuellen Verdeutlichung erfolgt. Das Eingabewort kann dann statt mit $12 \cdot 8 = 96$ Bit mit $1+2+3+1+4+1+4+1+2+3+1+2 = 25$ Bit dargestellt werden. Natürlich muss zu diesen $25$ Bit noch zusätzlich der Baum als Datenstruktur gespeichert werden, dennoch lässt sich aber leicht einsehen, dass die Huffmankodierung für längere Texte eine gute Kompressionsrate erreichen kann.

## 2.3 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Alle Antworten auf konzeptionelle Fragen sollten in Ihrer Ausarbeitung erscheinen. Besprechen Sie nach eigenem Ermessen außerdem im Zuge Ihres Vortrags einige der konzeptionellen Fragen. Die Antworten auf die Implementierungsaufgaben werden durch Ihrem Code reflektiert.

### 2.3.1 Theoretischer Teil

- Erarbeiten Sie anhand geeigneter Sekundärliteratur die genaue Funktionsweise der Huffmankodierung. Berechnen Sie dazu zunächst einige Beispiele auf Papier. Warum müssen die Bits der Ausgabesymbole nicht durch spezielle Trennzeichen markiert werden?

- Erarbeiten Sie ein geeignetes Format zum Speichern des Huffmanbaums im Arbeitsspeicher. Schlagen Sie dazu nach, wie Bäume in C üblicherweise dargestellt werden.

- Erarbeiten Sie ein geeignetes Ausgabeformat, in welchem die komprimierte Nachricht gespeichert werden soll. Das Format muss hierbei nicht die optimale Darstellung verwenden, das heißt, Sie können das Dictionary und die Daten in einem für Menschen lesbaren Format ablegen.

- Untersuchen Sie Ihre fertige Implementierung auf Performanz. Errechnen Sie auch die durchschnittliche Kompressionsrate für Eingabendaten Ihrer Wahl.

### 2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie Ihrem Assemblerprogramm die Inhalte einer eingelesenen Datei als Pointer übergeben können.

  Eine unkomprimierte Nachricht ist als ASCII-Text in der Eingabedatei gespeichert.

- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

  ```
  int huffman_encode(char *data, char *result, unsigned int result_size)
  ```

  welche einen Pointer auf die unkomprimierten Eingabedaten `data` übergeben bekommt und die Huffmankomprimierten Daten in das Array `result` speichert. Denken Sie daran, dass die Huffmankompression sowohl Daten als auch das Dictionary zurückgeben muss. Wählen Sie hierfür ein sinnvolles Format (es ist Ihnen hierfür erlaubt, die Parameter der Funktion gegebenenfalls zu erweitern, sollte es Ihnen sinnvoll erscheinen). Das Argument `result_size` gibt die maximale Größe des für `result` reservierten Speicherbereichs an. Der Rückgabewert soll im Erfolgsfall die Länge der kodierten Daten sein. Wenn die Funktion fehlschlägt (beispielsweise, weil das Ergebnisarray nicht groß genug gewählt wurde) soll $-1$ zurückgegeben werden. Speichern Sie die komprimierte Nachricht im von Ihnen gewählten Format in einer vom Benutzer bestimmbaren Datei im Rahmenprogramm (d.h. in C).

- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

  ```
  int huffman_decode(char *data, char *result, unsigned int result_size)
  ```

  welche einen Pointer auf die Huffmankomprimierten Eingabedaten `data` übergeben bekommt und die unkomprimierten Daten in das Array `result` speichert. Sie können die Funktionssignatur gegebenenfalls erweitern, um das Dictionary komfortabel übergeben zu können. Der Parameter `result_size` gibt die maximale Größe des für `result` reservierten Speicherbereichs an. Der Rückgabewert soll im Erfolgsfall die Länge der unkomprimierten Daten sein. Wenn die Funktion fehlschlägt (beispielsweise, weil das Dictionary nicht vollständig ist) soll $-1$ zurückgegeben werden.

### 2.3.3 Bonusaufgabe

Implementieren Sie im C-Rahmenprogramm eine Funktion, welche den in Assembler generierten Huffmanbaum auf die Konsole ausgibt.

## 2.4 Checkliste

Die folgende Liste soll Ihnen als Gedächtnisstütze beim Bearbeiten der Aufgaben dienen. Sollten Sie eine Reverse-Engineering-Aufgabe erhalten haben, sind diese Punkte für Sie weitestgehend hinfällig.

- Verwenden Sie keinen Inline-Assembler.
- I/O-Operationen dürfen grundsätzlich in C implementiert werden.
- Sie dürfen die Signatur der in Assembler zu implementierenden Funktion nur dann ändern, wenn Sie dies (in Ihrer Ausarbeitung) rechtfertigen können.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz).
- Verwenden Sie NEON/SIMD-Befehle, wenn möglich.
- Fügen Sie Ihrem fertigen Quelltext Anweisungen hinzu, wie das Projekt kompiliert werden kann.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen sollten mit abgegeben werden.
- Bonusaufgaben (sofern vorhanden) müssen nicht implementiert werden.