LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Huffmankodierung (A404)

Projektaufgabe – Aufgabenbereich Theorie der Informationsverarbeitung

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 festgesetzt, die Sie über die Praktikumshomepage[1] aufrufen können.

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. Die Teile der Aufgabe, in denen C-Code anzufertigen ist, sind in C nach dem C17-Standard zu schreiben.

Der **Abgabetermin** ist **Sonntag 05. Februar 2023, 23:59 Uhr (CEST)**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der `README.md` angegebene Liste von abzugebenden Dateien.

Die **Abschlusspräsentationen** finden in der Zeit vom **14.03.2023 – 24.03.2023** statt. Weitere Informationen werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind und keine nachträglichen Änderungen akzeptiert werden können.

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

Wir wünschen Ihnen viel Erfolg und Freude bei der Bearbeitung Ihrer Aufgabe!

Mit freundlichen Grüßen
Die Praktikumsleitung

[1] `https://gra.caps.in.tum.de`

2 Huffmankodierung

2.1 Überblick

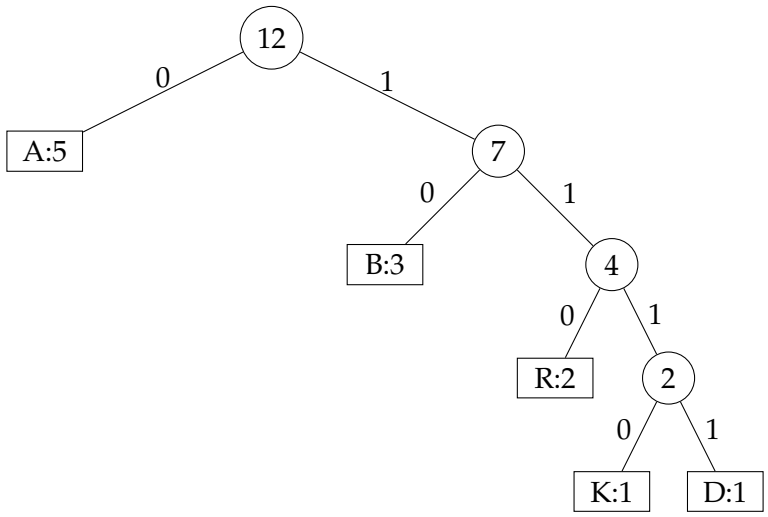
Die Informationstheorie ist ein Feld der Mathematik, in welchem man sich allgemein mit dem Kodieren von Nachrichten aufgrund von statistischen Gegebenheiten beschäftigt. Sie werden in Ihrer Projektaufgabe einen bestimmten Teilbereich näher beleuchten und einen bestimmten Algorithmus in C implementieren.

2.2 Funktionsweise

Die Huffmankodierung ist ein zur Datenkompression eingesetztes Entropiekodierungschema. 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. Antworten auf konzeptionelle Fragen sollten an den passenden Stellen in Ihrer Ausarbeitung in angemessenem Umfang erscheinen. Entscheiden Sie nach eigenem Ermessen, ob Sie im Rahmen Ihres Abschlussvortrags auch auf konzeptionelle Fragen eingehen. Die Antworten auf die Implementierungsaufgaben werden durch Ihren 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 könnten beispielsweise 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 Ihrer C-Funktion die Inhalte einer eingelesenen Datei als Pointer übergeben können und das Ergebnis in einer vom Benutzer gewählten Datei speichern.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

  ```
  void huffman_encode(size_t len, const char data[len])
  ```

  Die Funktion bekommt einen Pointer auf die unkomprimierten Eingabedaten `data` mit Länge `len` übergeben und gibt die Huffmankomprimierten Daten sowie deren Länge in dem von Ihnen entwickelten Format zurück. Passen Sie hierzu die Parameter und den Rückgabetyp der Funktion geeignet an. Denken Sie dabei auch an mögliche Fehlerfälle.

  Hinweis: Das Ausgabeformat dieser Funktion muss nicht optimal sein.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

  ```
  void huffman_decode(size_t len, const char data[len])
  ```

  Die Funktion bekommt einen Pointer auf die Huffmankomprimierten Eingabedaten `data` mit Länge `len` übergeben und gibt die unkomprimierten Daten sowie deren Länge zurück. Passen Sie hierzu die Parameter und den Rückgabetyp der Funktion geeignet an. Denken Sie dabei auch an mögliche Fehlerfälle.

2.3.3 Rahmenprogramm

Ihr Rahmenprogramm muss bei einem Aufruf die folgenden Optionen entgegennehmen und verarbeiten können. Wenn möglich soll das Programm sinnvolle Standardwerte definieren, sodass nicht immer alle Optionen gesetzt werden müssen.

- `-V <Zahl>` — Die Implementierung, die verwendet werden soll. Hierbei soll mit `-V 0` Ihre Hauptimplementierung verwendet werden. Wenn diese Option nicht gesetzt wird, soll ebenfalls die Hauptimplementierung ausgeführt werden.
- `-B <Zahl>` — Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das *optionale* Argument dieser Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs an.
- `<Dateiname>` — Positional Argument: Eingabedatei
- `-d` — Falls gesetzt, werden die Bilddaten dekodiert, ansonsten encodiert
- `-o <Dateiname>` — Ausgabedatei
- `-h` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.

- `--help` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.

Sie dürfen weitere Optionen implementieren, beispielsweise um vordefinierte Testfälle zu verwenden. Ihr Programm muss jedoch nur unter Verwendung der oben genannten Optionen verwendbar sein. Beachten Sie ebenfalls, dass Ihr Rahmenprogramm etwaige Randfälle korrekt abfangen muss und im Falle eines Fehlers mit einer aussagekräftigen Fehlermeldung auf `stderr` und einer kurzen Erläuterung zur Benutzung terminieren sollte.

2.4 Allgemeine Bewertungshinweise

Beachten Sie grundsätzlich alle in der Praktikumsordnung angegebenen Hinweise. Die folgende Liste konkretisiert einige der Bewertungspunkte:

- Stellen Sie unbedingt sicher, dass *sowohl* Ihre Implementierung *als auch* Ihre Ausarbeitung auf der Referenzplattform des Praktikums (`lxhalle`) kompilieren und vollständig korrekt bzw. funktionsfähig sind.
- Die Implementierung soll mit GCC/GNU as kompilieren. Verwenden Sie keinen Inline-Assembler. Achten Sie darauf, dass Ihr Programm keine x87-FPU- oder MMX-Instruktionen und SSE-Erweiterungen nur bis SSE4.2 verwendet. Andere ISA-Erweiterungen (z.B. AVX, BMI1) dürfen Sie nur benutzen, sofern Ihre Implementierung auch auf Prozessoren ohne derartige Erweiterungen lauffähig ist.
- Sie dürfen die angegebenen Funktionssignaturen (nur dann) ändern, wenn Sie dies (in Ihrer Ausarbeitung) begründen.
- Verwenden Sie die angegebenen Funktionsnamen für Ihre Hauptimplementierung. Falls Sie mehrere Implementierungen schreiben, legen wir Ihnen nahe, für die Benennung der alternativen Implementierungen mit dem Suffix „`_V1`“, „`_V2`“ etc. zu arbeiten.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz) und behandeln Sie *alle möglichen Eingaben*, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden; größere Eingaben sollten stattdessen stark komprimiert oder (bevorzugt) über ein abgegebenes Skript generierbar sein.
- Stellen Sie Performanz-Ergebnisse nach Möglichkeit grafisch dar.
- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Geben Sie die Folien für Ihre Abschlusspräsentation im PDF-Format ab. Achten Sie auf hinreichenden Kontrast (schwarzer Text auf weißem Grund!) und eine angemessene Schriftgröße. Verwenden Sie 16:9 als Folien-Format.