LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Gruppe 134 – Abgabe zu Aufgabe A404
Wintersemester 2022/23

Arman Habibi Larissa Manalil Andrei Stoica

1 Einleitung

Die von David A. Huffman, im jahr 1952, entwickelte Huffmankodierung dient zur verlustfreien Datenkompression. Um Redundanz zu minimieren, werden, nach einer Häufigkeitsanalyse der Symbole, diese in einem Binärbaum angeordnet, sodass der entstehende Binärkode, mit welchem das Symbol im Baum zu erreichen ist, präfixfrei ist. Dies bedeutet, dass kein Binärkode Teil eines anderen sein darf und ermöglicht es die Daten ohne Trennzeichen zu speichern. Die Symbole, die häufiger in der Quelldatei auftauchen, befinden sich weiter oben im Baum und brauchen daher auch weniger Bits zum abspeichern. Das Verfahren ist soweit Optimal, da es keinen anderen symbolbasierten Weg gibt, der einen kürzeren Code erzeugen könnte, insofern die Häufigkeiten der Symbole bekannt sind. [1] Im folgenden gilt es diesen Algorithmus in der Programmiersprache C zu implementieren und ein Rahmenprogramm zu erstellen, welches es ermöglicht Dateien zu komprimieren und zu dekodieren.

2 Lösungsansatz

2.1 Encode

Zuerst wird die Häufigkeitsanalyse der Symbole durchgeführt. Dafür wird ein Array mit Pointern auf Nodes für jedes ASCII Zeichen angelegt, welche die Frequenz des Zeichens speichern, sowie das Zeichen für die spätere Verwendung. Das Zählen erfolgt hierbei mit einer Effizienz von $\mathcal{O}(1)$, da der Array-Index dem ASCII-Code entspricht. Zusätzlich wird Speicher gespart, indem die Nodes erst erstellt werden sobald das zugehörige ASCII Zeichen das erste mal gelesen wird.

Als Beispiel wird der String $ABRACADABRAB$ komprimiert, welcher die Folgenden Häufigkeiten enthält. Nicht vorhandene Zeichen wurden vernachlässigt.

A	B	C	D	R
5	3	1	1	2

Anschließend wird ein Min-Heap erstellt und die erstellten Nodes darin eingefügt. Hier fängt der von Huffman entwickelte Algorithmus an. [2] Solange mehr als ein Node im Min-Heap ist, werden die beiden Nodes mit der geringsten Frequenz aus dem Min-Heap entfernt und an einem neuen Node angehängt (mit einem Null Zeichen), welcher die Summe der beiden Frequenzen als Wert erhält. Dieser wird wieder in den Min-Heap

eingefügt. Am Ende bleibt nur noch ein Node im Min-Heap übrig, welcher den fertigen Huffman-Baum enthält.

2.2 Decode

3 Korrektheit

Es wird auf die Korrektheit des Huffman Algorithmus eingegangen, da er verlustfreie Komprimierung ermöglicht und somit Genauigkeit als Thema unangemessen wäre. Um das Problem des optimalen und präfixfreien Codes zu lösen sind zunächst einige Definitionen notwendig.

3.1 Kodierungstheorie

Das *Quellalphabet* der Länge n entspricht der Anzahl an verschiedenen Zeichen die in der Quelldatei vorkommen. In diesem Fall entspricht das maximale Quellalphabet der Menge an ASCII Zeichen. Jedem Zeichen wird ein *Gewicht* w zugeordnet, welches hier der absoluten Häufigkeit des Zeichens in der Quelldatei entspricht. Die Menge der Gewichte ist somit definiert als $W = \langle w_i > 0 \mid 0 \leq i < n \rangle$. Des Weiteren wird auch ein *Codealphabet* definiert, welches die Binäre Menge $\{0, 1\}$ ist, mit dem die Quelldatei komprimiert wird. Der *Code* ordnet jedem Zeichen i des Quelllphabets ein Codewort aus dem Codealphabet der Länge l_i zu und ist definiert als die Menge $T = \langle l_i > 0 \mid 0 \leq i < n \rangle$.

Die Kraft-McMillan-Ungleichung gilt als notwendige Bedingung für präfixfreie Codes.

$$\mathcal{K}(T) = \sum_{i=0}^{n-1} 2^{-l_i} \leq 1$$

Codes die diese Bedingung erfüllen werden *zulässig* genannt, dies bedeutet dass eine Belegung an Codewörten existiert, die präfixfrei sind. Eine Menge an Codewörten ist für einen gegebenen Code *gültig*, insofern die Längen $\langle l_i \rangle$ übereinstimmen und kein Codewort präfix eines Anderen ist. Es ist somit möglich zu sagen, dass eine präfixfreier Code determiniert wurde, sobald ein zulässiger Code gefunden wurde, ohne die spezifischen Codewörter zu betrachten, da man weiß, dass eine gültige Belegung existiert.

Zuletzt sind noch die *Kosten* C eines Codes zu definieren, welches der Länge der kodierten Nachricht entspricht.

$$C(W, T) = C(\langle w_i \rangle, \langle l_i \rangle) = \sum_{i=0}^{n-1} w_i \cdot l_i$$

Ein Code $T = \langle l_i > 0 \mid 0 \leq i < n \rangle$ gilt als optimal, falls für jeden anderen zulässigen Code T' der Länge n gilt: $C(W, T) \leq C(W, T')$. Dies bedeutet auch, dass mehrere optimale Codes für eine gegebene Quelldatei existieren können. Für einen optimalen Code muss die Kraft-McMillan-Summe 1 ergeben, da bei kleineren Werten immer ein Codewort gekürzt werden kann, während größere Werte von Anfang an nicht zulässig sind.

4 Performanzanalyse

5 Zusammenfassung und Ausblick

Literatur

[1] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[2] Alistair Moffat. Huffman coding. *ACM Comput. Surv.*, 52(4), aug 2019.