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. [3] 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. [4] 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.

3.2 Optimalität

Um zu zeigen, dass der Huffman Algorithmus tatsächlich optimal ist, muss zunächst die *geschwister Eigenschaft* betrachtet werden. Diese besagt, dass ein optimaler Code existiert, bei dem die zwei Zeichen mit dem geringsten Gewicht geschwister sind, also einen Elternknoten im zugehörigen Binärbaum teilen.

Angenommen es gibt einen optimalen Code $T = \langle l_i \rangle$ mit einer Häufigkeitsverteilung $W = \langle w_i \rangle$ auf n Zeichen. Da jeder nicht-Blatt Knoten zwei Kinder haben muss, existieren zwei Blatt Knoten mit der höchsten Tiefe $L = max_i l_i$. Ohne Beschränkung der Allgemeinheit sind diese Knoten w_{n-1} und w_{n-2} diejenigen mit den geringsten Gewicht. Falls sie die selbe Tiefe haben, kann der Baum nun umstrukturiert werden, um die beiden Knoten zu Geschwistern zu machen, ohne die Kosten des Codes zu verändern.

Der Beweis der Optimalität erfolgt durch Induktion über der Anzahl an Zeichen n. [1] Als Induktionsbasis gilt n = 2, da es hier nur den optimalen Code $\langle l_i \rangle = \langle 1, 1 \rangle$ geben kann, nähmlich die beiden Codewörter 0 und 1. Es wird angenommen, dass die Aussage für $(n-1)$ Zeichen valide ist um sie für n Zeichen zu beweisen. Man nimmt an es gibt die Gewichte $W = \langle w_0, ..., w_{n-2}, w_{n-1} \rangle$, sodass für die Gewichte ohne Beschränkung der Allgemeinheit gilt $w_o \geq w_1 \geq ... \geq w_{n-1} > 0$.

Ausgehend von der geschwister Eigenschaft kann man nun sagen, dass es einen Optimalen Code $T = \langle l_0, ..., l_{n-2}, l_{n-1} \rangle$ gibt, wo die Gewichte w_{n-1} und w_{n-2} geschwister sind. Es gilt nun zu Beweisen, dass der durch Huffman hergeleitete Code H dieselben Kosten hat.

Wenn man nun die beiden Knoten w_{n-2} und w_{n-1} entfernt und dem Elternknoten die Summe der Gewichte zuordnet, erhält man den Code T' mit $(n-1)$ Zeichen und den Gewichten $W' = \langle w_0, ..., w_{n-3}, w_{n-2} + w_{n-1} \rangle$. Das Selbe geschieht nun für den Huffman Code H' mit den gleichen Gewichten $W'_H = \langle w_0, ..., w_{n-3}, w_{n-2} + w_{n-1} \rangle$ per Definition vom Huffman Algorithmus. Es gilt per Induktionsannahme:

$$C(W'_H, H') = C(W', T')$$

Ebenfalls gilt,

$$C(W', T') = C(W, T) - (w_{n-2} + w_{n-1})$$

da die Summe der Knoten w_{n-2} und w_{n-1} nun auf der Tiefe $L - 1$ liegt. Das selbe gilt wiederum für den Huffman Code:

$$C(W'_H, H') = C(W_H, H) - (w_{n-2} + w_{n-1})$$

Diese drei Gleichungen resultieren in folgendem Beweis.

$$C(W_H, H) = C(W'_H, H') + w_{n-2} + w_{n-1} = C(W', T') + w_{n-2} + w_{n-1} = C(W, T)$$

3.3 Komprimierungseffektivität

Claude Shannon definierte, 1948, die *Entropie* einer Menge an Gewichten folgendermaßen. [6]

$$\mathcal{H}(W) = -\sum_{i=0}^{n-1} w_i \cdot log_2 \frac{w_i}{m}$$

Sie lässt sich aufteilen auf die Summe der Gewichte $m = \sum_{i=0}^{n-1} w_i$ und den entropischen Kosten (in Bits) $-log_2(w_i/m) = log_2(m/w_i)$ einer Instanz eines Zeichens, welches mit einer Wahrscheinlichkeit von (w_i/m) auftritt. Die Entropie $\mathcal{H}(W)$ gibt die geringste mögliche Länge in Bits an, die der komprimierte Code haben kann, wenn man die Häufigkeitsverteilung mit einbezieht. Der *relative Effektivitätsverlust* $\mathcal{E}$ eines zulässigen Codes T und der zugehörigen Gewichte W ist durch folgende Gleichung zu berechnen.

$$\mathcal{E}(W,T) = \frac{C(W,T) - \mathcal{H}(W)}{\mathcal{H}(W)}$$

Ein relativer Effektivitätsverlust von 0 bedeutet, dass die Zeichen mit ihren entropischen Kosten komprimiert werden, wohingegend bei größeren Werten, diese nicht erreicht werden.

Die Huffmankodierung kommt, in den meisten Situationen, den entropischen Kosten sehr nahe und resultiert daher in Effektivitätsverlusten, die wenige Prozente über dem Optimalfall liegen. Jedoch gibt es auch einige Fälle, wo der Huffman Code einen großen relativen Effektivitätsverlust hat, insbesonderen, wenn w_0 im Vergleich zur Summe der anderen Zeichen sehr groß wird und w_0/m nahe an die 1 kommt. Zum Beispiel hat die Gewichtmenge $W = \langle 96, 1, 1, 1, 1\rangle$ eine Entropie von $\mathcal{H}(W) = 32.2$ Bits , ein zugehöriger optimaler Code $T = \langle 1, 3, 3, 3, 3\rangle$ mit Kosten $C(W,T) = 108$ Bits, und somit einen relativen Effektivitätsverlust von $\mathcal{E}(W,T) = 235\%$. Ein optimaler Code bedeutet somit nicht unbedingt optimale Komprimierung.

Robert Gallager [2] zeigte folgende Beziehung, zwischen einem optimalen Code $T = \langle l_i\rangle$ und der Gewichtsmenge $W = \langle w_i\rangle$ mit der Summe $\sum_{i=0}^{n-1} w_i = m$, mit der sich eine obere Schranke für den relativen Effektivitätsverlust berechnen lässt.

$$C(W,T) - \mathcal{H}(W) \leq \begin{cases} w_0 + 0.086 \cdot m, \text{wenn } w_0 < \mathrm{m}/2 \\ w_0, \text{wenn } w_0 \geq \mathrm{m}/2 \end{cases}$$

4 Performanzanalyse

5 Zusammenfassung und Ausblick

Insgesamt ist der Huffman Algorithmus, aufgrund seiner Optimalität bei symbolbasierter Komprimierung, immer noch sehr relevant, da er in den meisten Fällen sehr knapp an die, durch die Entropie gegebene, perfekte Komprimierung kommt. Dennoch ist es möglich die Effizienz noch weiter zu erhöhen, indem man sich nicht nur auf symbolbasierte Algorithmen beschränkt.

Es wäre somit möglich, zum Beispiel bei von Menschen geschriebenen Fließtext, einen Huffman Baum für die einzelnen, durch Leerzeichen gentrennte, Wörter zu erstellen. Dies könnte bei sehr großen Quelldateien, im Vergleich zum symbolbasierten Ansatz, effizienter sein.

Eine spezielle Anpassung auf das gespeicherte Format sollte ebenfalls zu Optimierungen führen. Man denke sich, zum Beispiel, PNG für Bild- oder FLAC für Audiodateien, da hier mehr Daten, als nur Buchstaben, abgespeichert werden müssen.

Zu der Implementierung kann man sagen, dass diese noch deutlich optimierbar ist. Zu einem ist das Verfahren mit den Min-Heap sehr anschaulich, aber dafür ineffizient, da es auch Lösungswege gibt, die dies in $\mathcal{O}(n)$ erreichen [7], unter anderem auch in-place [5]. Das Speicherformat könnte ebenfalls weiter optimiert werden, indem man, anstatt diese Menschenleslich zu strukturieren, direkt die Binärdaten reinschreibt. Zudem wäre es auch denkbar die Laufzeit, durch Parallelisierung und der Verwendung von mehreren Threads, zu reduzieren.

Zusammenfassend können wir sagen, das wir, durch das Projekt, zum optimierten Programmieren angeregt und somit ein hoher Lerneffekt erreicht wurde.

Literatur

[1] UC Berkeley. Proof of optimality of huffman coding. `https://inst.eecs.berkeley.edu/~cs170/fa20/assets/notes/huffman.pdf`. Accessed: 2023-02-04.

[2] R. Gallager. Variations on a theme by huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.

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

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

[5] Alistair Moffat and Jyrki Katajainen. In-place calculation of minimum-redundancy codes. In Selim G. Akl, Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, pages 393–402, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[6] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[7] Jan van Leeuwen. On the construction of huffman trees. In *International Colloquium on Automata, Languages and Programming*, 1976.