Kompressionsprogramm

Entwicklerdokumentation

Huffman-Kodierung ist eine Entropiekodierungsform, damit man die Größe von Files ohne Datenverlust effizient reduzieren kann. Die Grundidee ist, das File auf Teilen mit denselben Längen aufzuteilen, und für jede mögliche Variation einen Kode zu generieren so, dass die Kodes nicht Präfixen voneinander sind, und die öfter vorkommenden Variationen kürzere Kodes haben. Die Kraft-Ungleichung sagt, dass eine solche Kodeliste eindeutig generierbar ist, wenn man die Schlüssellängen weiß.

Um eine Liste von Schlüssellängen zu haben lesen wir zuerst das File Byte zu Byte durch, und zahlen, wie viel es von den verschiedenen Variationen gibt. Mithilfe dieser Häufigkeitsliste können wir einen binären Baum bauen, in dem je häufiger ein Element vorkommt, desto näher es zu der Wurzel ist. Dann, um die Kodelänge für ein Element zu bekommen, zahlen wir viele Tritte weg es von der Wurzel liegt. Wenn wir die Kodelängen haben, generieren wir den Kode jeder Variation mit einem Algorithmus, das in diesem Dokument später bespricht wird. Wenn wir die Kodes haben, können wir das File einfach kompressieren. Zuerst schreiben wir Header-Information, wie die benutzte Aufteilung, und die Länge des Files, verfolgt von den Schlüssellängen aus. Danach lesen wir die Daten Byte zu Byte (oder Wort zu Wort in 16-Bit Mode) ein, und schreiben die entsprechenden Kodes aus. Weil wir einzige Bits nicht ausschreiben können, müssen wir ein Buffer realisieren.

Wenn wir ein File dekompressieren möchten, lesen wir zuerst die Header-Information ein. Mit dem eingelesenen Schlüssellängen können wir dann die bei der Kompression benutzte Kodes herstellen, und ein binäres Suchbaum bauen. Mit einem Buffer können wir das File Bit zu Bit bearbeiten. Wir fangen an der Wurzel des Baumes an, und nach jedem eingelesenen Bit treten nach links oder nach rechts. Wenn wir das Ende des Baumes erreichen, schreiben wir die entsprechenden Byte/Wort aus, und treten wir zu der Wurzel. Das machen wir bis Ende des Files.

Aufbau des Programmes

Das Programm besteht aus 3 C-Files, und 2, zu denjenigen gehörigen Header-Files.

kompression.c

Dieses File enthielt die Main-Funktion, und handelt Benutzerinteraktion. Das Programm fängt hier an, und endet hier. Kommandozeilenparameter werden hier bearbeitet, und die entsprechenden Funktionen der anderen C-Files gerufen.

files.c

Files.c enthielt zum File-Handling gehörigen Funktionen.

create_filename:

Zufügt oder schneidet “.komp“ zum Ende des Files.

read_num_of_occurences:

Liest das File Byte zu Byte (oder Wort zu Wort) durch, und zählt, wie viel es von den verschiedenen Variationen gibt. Gibt ein Array mit den Häufigkeiten zurück.

write_codelengths:
Schreibt Header-Information zu dem File.

read_codelengths:
Liest Header-Information ein.

compress_file:
Führt die Kompression des Files durch. Liest das File Byte zu Byte ein, und schreibt die entsprechenden Kodes aus.

decompress_file:
Liest das File Bit zu Bit ein, und schreibt die Daten, die es mit dem Suchbaum findet.

get_file_size:
Gibt die Größe des Files zurück.

maxcodelength:
Gibt die größte Kodelänge zurück.

create_header_byte:
Generiert die Header-Byte, die Information über die Kompression enthielt. Die Header-Byte sieht so aus: 00000CBA, wo A die benutzte Aufteilung (0:8-Bit 1:16-Bit), B die maximale Kodelänge (256/65536), und C die Existenz der Leftover-Byte zeichnet.

huffman.c
Dieses File enthielt zum Baumbauen und Kodegeneration gehörenden Funktionen.

free_tree:
Befreit die Speicher, die für den Baum allokiert war.

add_list_node:
Addiert ein neues Element zu der Liste.

find_least_frequent:
Findet das Element mit der kleinsten Häufigkeit, gibt ein Pointer zu ihm zurück.

remove_from_list:
Entfernt ein Element von der Liste.

newnode:
Gibt ein Pointer zu einem Node mit dem gegebenen Wert und Häufigkeit zurück

build_nodeptr_list:
Auffüllt die Liste mit den Häufigkeiten.

build_node_tree:
Baut den Baum, damit die Kodelängen kalkuliert werden.

build_codes:

Generiert die Kodes. Zuerst ordnet es die Elemente aufgrund der Kodelängen. Der Kode des ersten Elements wird nur aus Nullen bestehen. Dann, um den Kode eines Elements zu bekommen, addiert es 1 zu dem Kode des vorigen Elements, und schiebt es mit der Kodelängendifferenz nach links.

Ein Beispiel:

Seien die Kodelängen die kodierenden Daten die folgenden: **A**,**B**,**C**,**D**,**E**, und die zu ihnen gehörigen Kodelängen **4**,**3**,**5**,**5**,**3**. Zuerst ordnen wir die daten, dann generieren die Kodes, und bekommen die folgende:

Die Daten	**Die Kodelängen**	**Die Kodes**
B	3	000
E	3	001
A	4	0100
C	5	01010
D	5	01011

Wir können hier auch sehen, dass keine von den Kodes Präfixen voneinander sind.

build_dictionary:

Generiert ein String-Array mit den Kodes.

build_codelength_array:

Findet die Kodelänge für jedem Wert, und zurückgibt ein Array, die die enthielt.

populate_codelength_array:

Eine rekursive Funktion, die den Array auffüllt.

build_tree_from_codes:

Baut den Suchbaum, der für Dekompression benutzt wird.

search_in_tree:

Tritt nach links oder nach rechts in dem Baum, und zurückgibt die Werte die es findet.

print_state.s

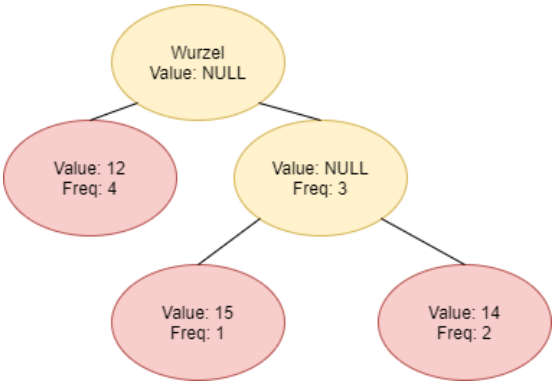
Enthielt ein Funktion heißt “print_num“, die ein Nummer und ein String auf dem Bildschirm schreibt.

Datenstruktur

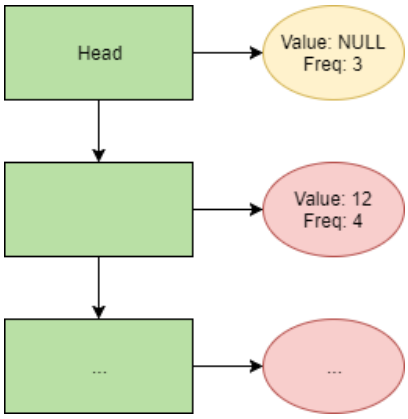
Bei Kompression

Die Liste funktioniert als eine Warteliste, die Nodes, die keine Eltern haben, sind da lagert. Wir finden immer die zwei Nodes, die die kleinsten Häufigkeiten haben, und addieren ein neues Node zu der Liste, die Kinder von denen die zwei auswählten Nodes sind, und dessen Häufigkeit die Summe der Häufigkeiten seiner Kinder ist.

Der Baum:

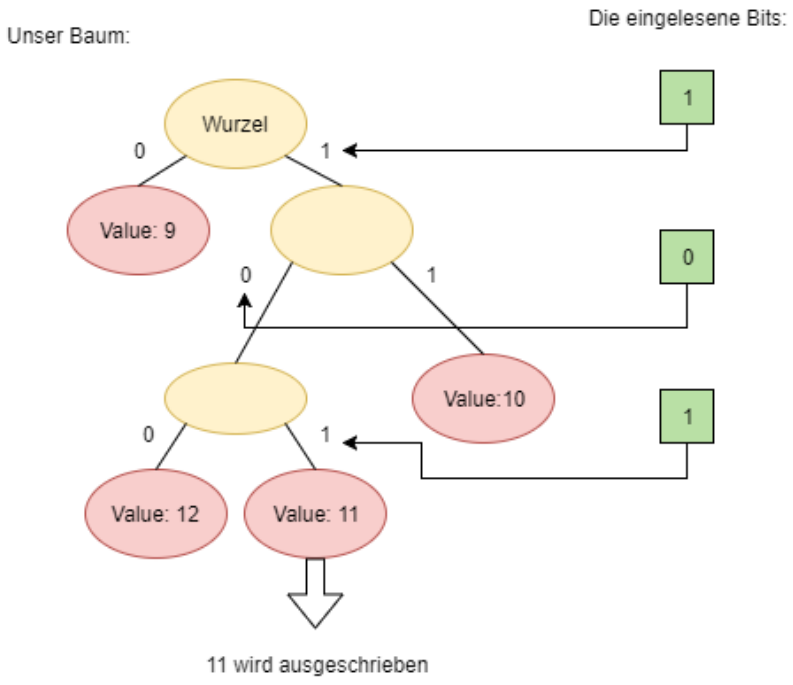


Die Liste:



Bei Dekompression

Wir bauen mit den Kodes einen Baum. Bei Dekompression beschäftigen wir uns mit Häufigkeiten nicht, unser Baum enthielt nur die Werte. Wir lesen das File Bit zu Bit ein, und treten immer nach links oder nach rechts.



Übersetzungsinstruktionen

Das Programm kann man mit folgenden Befehlen herunterladen, übersetzen, und installieren:

```
git clone --recursive https://github.com/mondokm/kompressionsprogramm.git
cd kompressionsprogramm
sudo make install
```

Anforderungen

- Linux oder BSD basiertes System
- GMP
- AMD64 Architektur (empfohlen)