

1.)

Da alle Ansätze von Huffman-, LZW-, BWT-, RLE- und Arithmetischer Codierung sehr schnell an ihre Grenzen stossen, (siehe Aufgabe 3 – 845KB von 2849KB = 29.7%), muessen also noch Vorcodierungen erfolgen, die etwaige Spezialstrukturen der Datei ausnutzen.

1. Die Codierung der Zwischenraeume wird weggelassen. (Leerzeichen, Zeilenumbrueche)
2. Die Zahlen werden Spaltenweise codiert, da hier stärkere Zusammenhänge bestehen.
3. Fuer die Spalte 1, Spalte 5 und Spalte 6 bestehen enge Zusammenhaenge, d.h. fuer viele Werte aus Spalte 1 stehen in Spalte 5 und 6 immer feste Werte.

Der Codierungs-Algorithmus ueberprueft fuer jeden in Spalte 1 auftretenden Wert die Werte in den Spalten 5 und 6. Stehen dort immer die gleichen Werte, so wird dies in einer Tabelle vermerkt, und beim Codieren werden die entsprechenden Elemente einfach nicht mit codiert.

Dafuer wird die Tabelle jedoch an die codierte Datei angefuegt.

Der Decodierer erkennt somit beim Decodieren der Spalte 1, ob er in Spalte 5 oder 6 einen bestimmten Wert schreiben soll.

Die Gesamtzahl der Spalten ist hierbei egal, so lange mindestens 6 Spalten vorliegen.

4. Für jede Spalte wird unter den folgenden 3 Methoden nach der optimalen Codierungsvariante gesucht :
 - **Method 1:** Ausgehend vom Maximalwert einer Spalte, werden die Elemente mit der minimal notwendigen Bit-Anzahl codiert (Bsp : Spalte1 Maximalwert : 190, d.h. 8Bit je Element).
 - **Method 2:** Es wird eine Lookup-Tabelle fuer die verschiedenen auftretenden Werte erstellt, und die Werte werden dann als die Indizes fuer die Lookup-Tabelle gespeichert. (Bsp : Spalte 1 Anzahl der Werte : 180, d.h. 8Bit je Element)
 - **Method 3:** Es wird eine Lookup-Tabelle fuer die verschiedenen auftretenden Differenzen zweier benachbarter Zeilen erstellt, und die Werte werden dann als die Indizes fuer die Lookup-Tabelle gespeichert. (Bsp : Spalte 1 Anzahl der Spruenge(Differenzen) : 375, d.h. 9Bit je Element)

In der folgenden Tabelle sind fuer die Beispieldatei die Werte aller Methoden aufgefuehrt :

Spalte	1	2	3	4	5	6
Wertebereich	[0,190]	[0,511]	[0,7]	[0,511]	[0,20]	[0,102]
Anzahl der Werte	180	512	8	512	18	51
Wertebereich der Sprünge	[-188,+189]	[0,1] (sehr lange Ketten, bei denen die Differenz zum Vorgänger 0 ist)	[-7,+7]	[-511,+21]	[-19,+20] (aber 109.726 Sprünge der Länge 0)	[-99,+78]
Anzahl der Sprünge	375	2	15	31	33	101

Hier sieht man z.B. sehr gut, dass fuer die Spalten folgende Methoden effizient sind :

- Spalte 1 : Methode 1 oder 2
 Spalte 2 : Methode 3
 Spalte 3 : Methode 1 oder 2
 Spalte 4 : Methode 3
 Spalte 5 : Methode 1 oder 2
 Spalte 6 : Methode 2

Der beigefuegte Codierungs-Algorithmus erkennt die Anzahl der Spalten, ermittelt fuer jede Spalte die notwendigen Werte und entscheidet dann anhand der zu erwartenden Kompressionsgrosse, welche Codierungsmethode fuer jede einzelne Spalte am effizientesten ist.

Nachdem alle Spalten codiert wurden, werden diese noch mit dem allgemeinen Codierer aus Aufgabe 3 codiert und schliesslich mit allen benoetigten Zusatzinformationen in die Ausgabedatei geschrieben.

Man wuerde eine weitere Kompressions-Steigerung erzielen , wenn man die Codewoerter , welche sich durch Anwendung einer der oben aufgefuehrte Methoden ergeben , Huffman-Codieren wuerde. Dies wuerde allerdings nur noch eine minimale Verbesserung auf ca. 290KB bringen , dafuer jedoch einen hoeheren Rechen- und Komplexitaetsaufwand mit sich bringen. (Wurde nicht implementiert.)

Der Codierungs-Algorithmus angewandt auf die Beispieldatei „testinput.txt“ liefert :

Groesse der komprimierten Datei : 308KB (=10.8 %)

Codierungszeit auf einem Notebook mit AMD-K6 400MHz Prozessor :

Codierung : 5 sec

Decodierung : 5.5 sec

Dateien des Codecs :

„encode.cpp“ - Programm zum komprimieren

„decode.cpp“ - Programm zum dekomprimieren

„SpecialCodec.h“ - Codec zur Vorcodierung nach oben beschriebener Vorgehensweise

„Codec.h“ - allgemeiner Kompressionsalgorithmus nach Aufgabe 3 bestehend aus :

„BWT.h“ - 1. Schritt BurrowsWheelerTransformation

„MTF.h“ - 2. Schritt MoveToFront Coder

„RLE.h“ - 3. Schritt RunLengthCoder

„ArithmeticCodec.h“ - 4. Schritt ArithmeticCoder

Anforderungen an die Datei , damit der Codierungsalgorithmus arbeiten kann :

1. Die Datei darf nur aus Zahlen bestehen , welche in Spalten , durch ein Leerzeichen getrennt angeordnet sind
2. mindestens 6 Spalten , maximal 255 Spalten
3. Spaltenelemente koennen nur Werte von 0 bis 511 annehmen

Formale Anmerkungen :

Wurden in der Originaldatei die Spaltenelemente mit mehreren Leerzeichen oder Tabulatoren getrennt , oder auch mit einer je weils verschiedenen Anzahl an Leerzeichen oder Tabulatoren , so arbeitet der Algorithmus nur noch im weiteren Sinne korrekt.

Soll heissen , er codiert und decodiert alle Werte korrekt , trennt die Spaltenelemente in der decodierten Datei jedoch konsequent mit einem Leerzeichen.

Ob in der Originaldatei der Abschluss einer Zeile mit LF oder mit CRLF erfolgte ist egal. Dies erkennt der Algorithmus und speichert die Information fuer den Decodierer mit ab.

Die Quellcode-Dateien wurden nicht „aufgeraemt“ , d.h. es stehen noch sehr viele „TODO's“ als Anmerkungen fuer Verbesserungsvorschlaege sowie DEBUG-Kommentare darin , da ich zeitlich nicht alle Ideen umsetzen konnte.

Ich hoffe dies beeintraehtigt die Uebersichtlichkeit nicht allzu sehr.