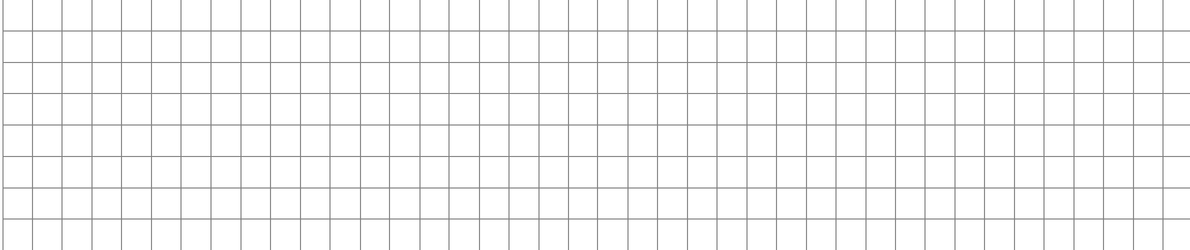
# Modul 114: Verlustfreie Kompressionsverfahren

## Zweck

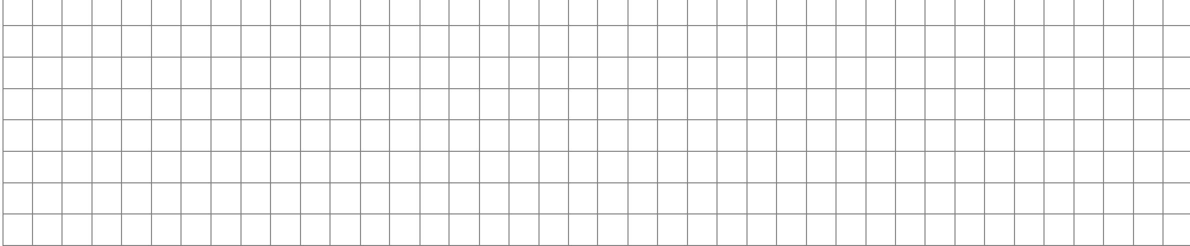
Es gibt viele Gründe Daten zu komprimieren:
- Sie machen von ihrem Server jeden Tag eine Datensicherung von 200 GB. Die Sicherungen müssen bis zu 12 Monate aufbewahrt werden.
- Sie möchten alle ihre Bücher digital auf einem "Tablet" speichern um sie auf Reisen immer dabei zu haben, auch offline.
- Ihre Firma hat 3 Standorte. Alle Daten eines Standorts müssen möglichst schnell an den anderen Standorten zur Verfügung gestellt werden, am besten in Sekundenschnelle.
- Sie möchten gerne Videos übers Handy anschauen ohne das er ruckelt, auch wenn Sie nur schwachen Empfang haben.

Die Datenspeicherung und die Datenübertragung können durch die Kompression optimiert werden Je nach Art der Daten kommen **verlustfreie** oder **verlustbehaftete** Verfahren zum Einsatz.

Nennen Sie Beispiele für die jeweilige Kategorie!
**Verlustfrei**
(beim Wiederherstellen der Daten erhalten Sie ein exaktes Abbild der unkomprimierten Daten)



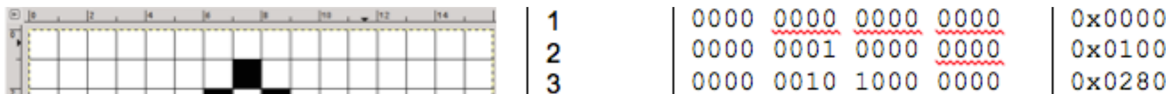
**Verlustbehaftet**
(beim Wiederherstellen der Daten erhalten fehlen Informationen bzw. die Qualität ist schlechter)



## Lauflängencodierung S. 184

### Lauflängencodierung mit binären Daten

Wir haben in diesem Modul gelernt, dass alle Informationen als Zahlen gespeichert werden, diese Zahlen wiederum werden als Bits gespeichert. Ein Kompressionsverfahren sollte also darauf abzielen diese Bits auf eine "sparsame" Weise darstellen zu können.
Schauen wir und die Darstellung der Bitmapgrafik auf AB114-05 nochmals an:




Pro Zeile werden 2 Bytes verwendet, kann das optimiert werden?

Wir könnten doch folgendes notieren:
Zeile1: 16x Wert 0 Zeile2: 7x0 1x1 8x0 Zeile3: 6x0 1x1 1x0 1x1 7x0

Wird dadurch Speicher eingespart?

| | | |
|---|---|---|
| Codierung Zeile1: | 010000 | neue Länge -10 Bit |
| Codierung Zeile2: | 000111 100001 001000 | neue Länge +2 Bit |
| Codierung Zeile3: | 000110 100001 000001 100001 000111 | neue Länge +14 Bit |
| **Total** | | **neue Länge +6 Bit** |

Total werden leider nach dieser "Kompression" 6 Bit mehr benötigt. Die Ursache liegt in der maximalen Anzahl von 16 Bit pro Zeile, welche nur einmal erreicht wird. Wenn wir nun stattdessen eine maximale Länge von 7 aufeinanderfolgenden Bit definieren erhalten wir folgendes Resultat:

| | | |
|---|---|---|
| Codierung Zeile1: | 0111 0111 0010 | neue Länge -4 Bit |
| Codierung Zeile2: | 0111 1001 0111 0001 | neue Länge -0 Bit |
| Codierung Zeile3: | 0110 1001 0001 1001 0111 | neue Länge +4 Bit |
| **Total** | | **neue Länge -0 Bit** |

Was lernen wir daraus?

- Da es im binären System kein Trennzeichen gibt, müssen wir eine fixe Länge für die Mengenangabe definieren. 1 Bit Daten, 5 Bit Mengenangabe bzw. 1 Bit Daten 3 Bit Mengenangabe in den obigen Beispielen
- Je nach Daten, welche wir komprimieren möchten schwankt das Resultat enorm! Zeile 1 und 2 sind eher komprimierbar, Zeile 3 kaum. -> Es gibt nicht komprimierbare Daten
  Wenn das Bild grösser wäre, würde sich das Komprimieren auf jeden Fall lohnen!
- Eine vorgängige Analyse der zu komprimierenden Daten kann bei der Auswahl des Verfahrens nützlich sein. (In diesem Fall um die Länge der Mengenangabe zu optimieren)
- Ein Komprimierungsverfahren hat immer einen Anteil Daten und einen Anteil Zusatzinformation, welche aus dem Verfahren heraus entsteht. Das Ziel ist es nun, die Nutzdaten dermassen zu verkleinern, dass zusammen mit den Zusatzinformationen am Ende ein kürzerer Code entsteht.

### Lauflängencodierung mit Textdaten

Bei Textdaten (erweiterter ASCII, ANSI usw.) kann die Lauflängencodierung ebenfalls eingesetzt werden. Um Speicherplatz zu sparen wird die Mengenangabe binär codiert. Selten kommen mehr als 2 identische Zeichen hintereinander vor im normalen Sprachgebrauch.
Idee: 1 Zeichen + binäre Mengenangabe (0 -> kommt 1x vor, 1 --> kommt 2x vor)
Beispiel:

| | |
|---|---|
| 100 leere Brunnen | $10_2 01_2$ $0_2 l0_2 e1_2 r0_2 e0_2$ $0_2 B0_2 r0_2 u0_2 n1_2 e0_2 n0_2$ |
| 17x8 Bit = 136 Bit | (14x8) + (14) Bit = 112 + 14 Bit = 126 Bit |

Je nach Text kann die Lauflängencodierung eine Ersparnis bringen. Als Beispiele finden man deshalb oft solche "Texte": XXXXX YYYYY ZZZZZZZZ

Zum Thema Lauflängencodierung gibt es natürlich Übungen ➔*1*

## Wörterbuchverfahren zur Datenkompression S. 184

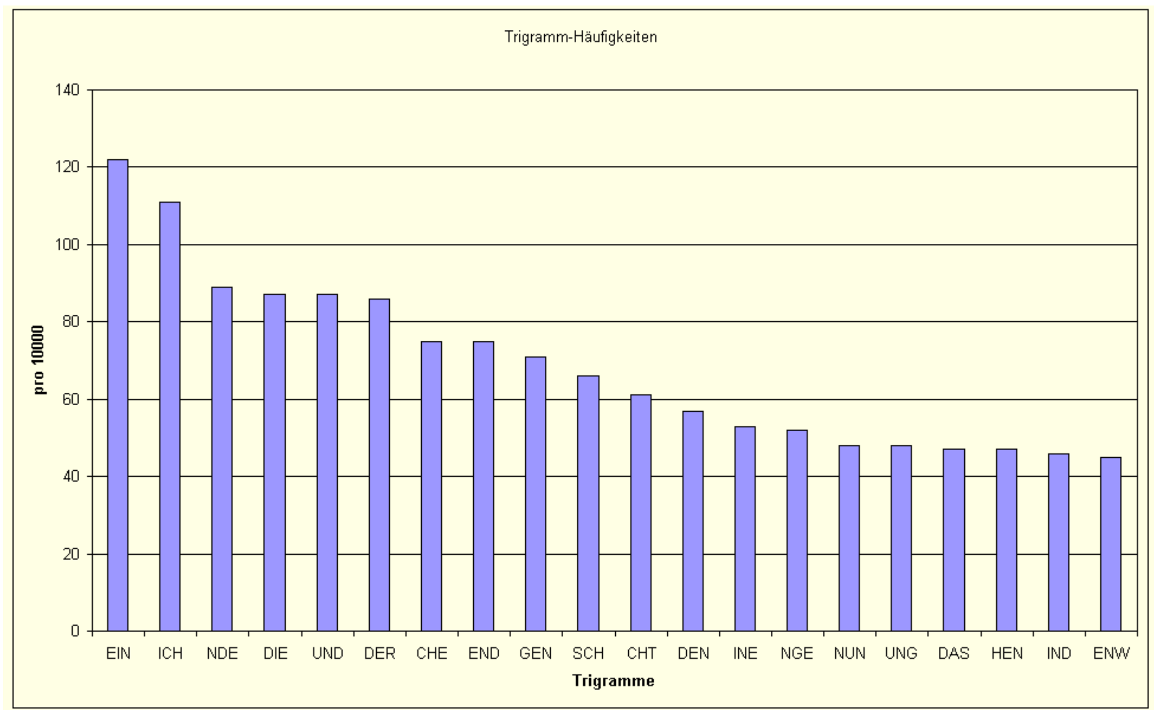
Bei der Wörterbuchmethode (engl. token based compression) werden wiederkehrende, häufig auftretende Muster zusammengefasst und durch einen kürzeren Code **ersetzt**. Dazu wird eine Liste (Wörterbuch) benötigt, in welcher die Codes und ihre ursprüngliche Bedeutung abgelegt sind. Übertragen wird dann Das Wörterbuch und die codierten Daten.

Die minimalste Variante eines Wörterbuchs für Text könnte so aussehen:

<Leer>: 00000    A: 00001    B: 00010    C: 00011 ...........    Z: 11010

Mit dieser Methode kann ein 8 Bit ASCII Zeichen auf lediglich 5 Bits dargestellt werden, zudem müsste das Wörterbuch nicht einmal mitgeliefert werden, da es bei der Dekompression als bekannt angenommen werden könnte.

Ein Wörterbuch kann statt einzelner Buchstaben auch ganze Wörter aufnehmen. So kann beispielsweise für die deutsche Sprache ein intelligentes Wörterbuch erstellt werden anhand der statistischen Vorkommen von Buchstabenkombinationen:




(Quelle: http://www.staff.uni-mainz.de/pommeren/Kryptologie/Klassisch/1_Monoalph/deutsch.html)

Anhand dieser Tabelle könnte das Wörterbuch von oben um die Einträge aus dem Diagramm erweitert werden.

Bsp: EIN: 11011 (27) ICH: 11100 (28) NDE: 11101 DIE: 11110 UND: 11111 (31)

Der Satz "ICH UND MEIN HUND " würde dann so dargestellt: (28)(0)(31)(0)(13)(27)(0)(8)(31)
Dadurch würden lediglich 9x5 = 45 Bit benötigt, im Original wären es mit ASCII 17x8 = 136 Bit.

Was passiert aber mit englischen Texten oder mit neutralen Daten? (Bilder, Programme usw.)
Dazu gibt es spezialisierte Algorithmen (Rechenmodelle), welche vor der Kompression die Daten analysieren und dann selber eine Häufigkeit von wiederholenden Mustern erstellen. (siehe weitere Themen für Interessierte)

## Huffman Code (David A. Huffman 1952) **S. 184**

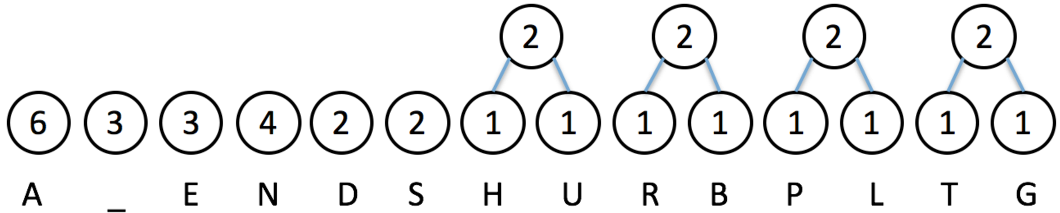
Der Huffman Code funktioniert auch nach dem System der Zeichenersetzung. Dabei werden die zu komprimierenden Daten analysiert, um herauszufinden welches Zeichen wie häufig vorkommt.
Im Gegensatz zu den bisher betrachteten Verfahren ist beim Huffman Code keine fixe Länge der codierten Zeichen notwendig. Anhand des Aufbaus des Codes kann bei **sequentiellem** Durchlaufen entschieden werden wann ein neues Zeichen kommt.

Häufige Zeichen erhalten einen kurzen Code, seltene Zeichen einen längeren Code.
Als Hilfsmittel zur Erstellung der Codes wird ein Baum erstellt (Metadaten). Am einfachsten ist das Verfahren an einem Beispiel zu verstehen:

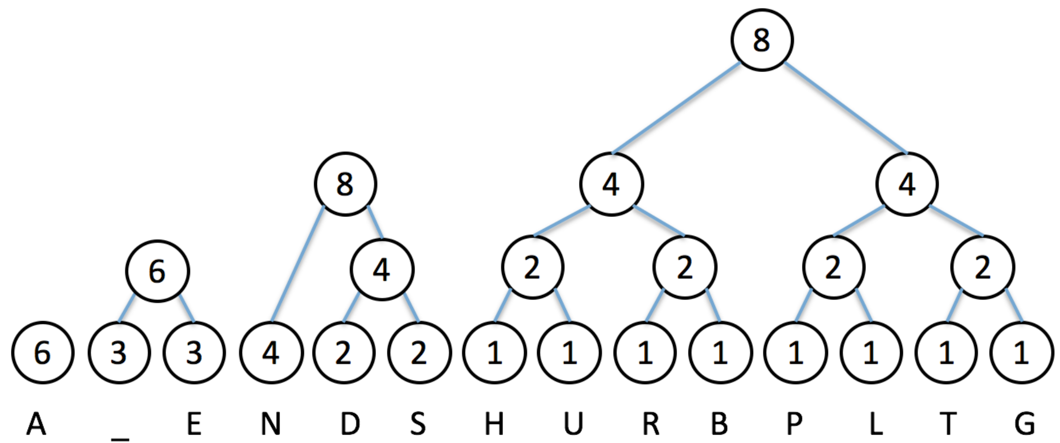
Zu komprimierende Daten: *DAS HAUS DER BANANENPLANTAGE*

Zählen der Vorkommen: (Das Leerzeichen nicht vergessen, wir schreiben es als "_")
D: 2 A: 6 S: 2 _: 3 H: 1 U: 1 E: 3 R: 1 B: 1 N: 4 P: 1 L: 1 T: 1
G: 1
Tipp: Am besten addieren wir alle Vorkommen und vergleichen Sie mit den Anzahl Zeichen der Daten!

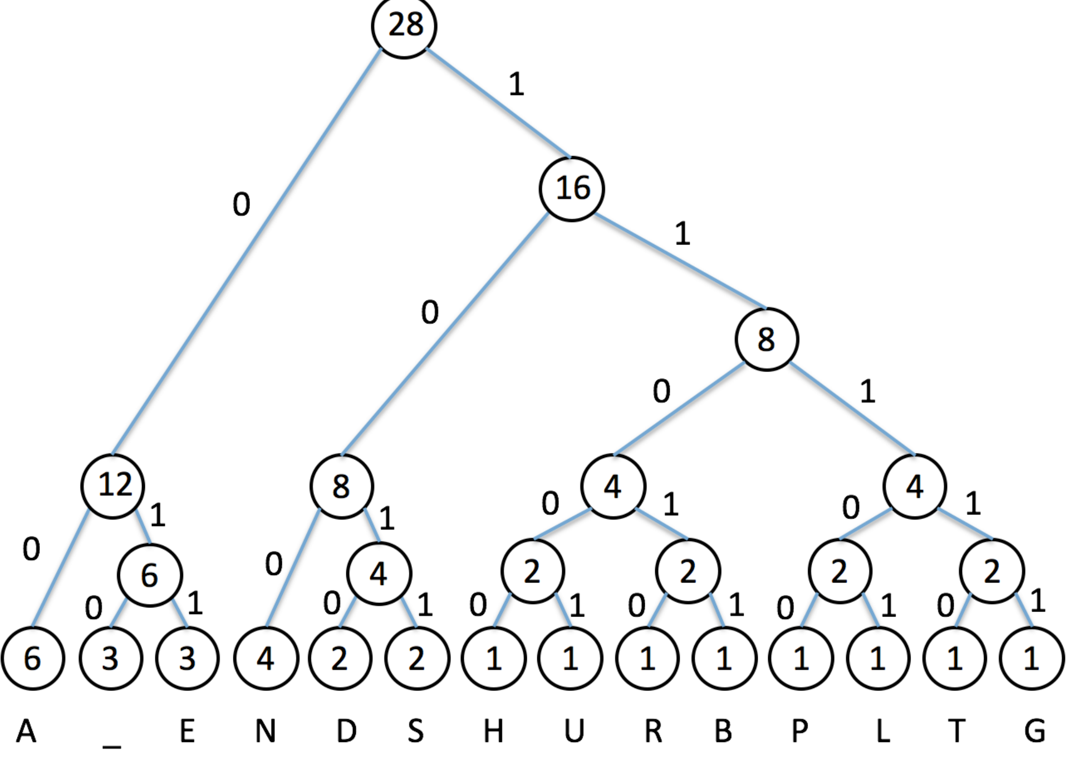
Nun bauen wir Schritt für Schritt den Baum auf, indem wir immer die tiefsten Vorkommen zusammenziehen zu einem neuen Knoten im Baum. Der neue Knoten erhält die Summe seiner beiden Äste als Inhalt:




Die Äste werden immer weiter verknüpft. Wichtig: immer die tiefsten Nummern miteinander verbinden!




Wenn der Baum bis oben zur Wurzel fertiggestellt ist, werden die Äste links und rechts jeweils mit 0 oder 1 beschriftet:




An dieser Stelle wäre wieder einmal eine Kontrolle des Totals (28) fällig.

Der Baum dient nun dazu die ursprüngliche Nachricht zu komprimieren:

*DAS HAUS DER BANANENPLANTAGE*

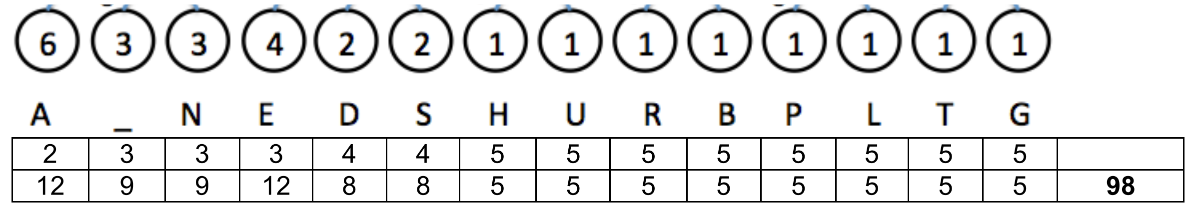
| D | 1010 |
|---|---|
| A | 00 |
| S | 1011 |
| _ | 010 |
| H | 11000 |
| ... | |

Die einzelnen Zeichen können nun vom Baum abgelesen werden und bilden zusammengesetzt den komprimierten Code.

101000101101011000...

Das Ziel, häufig auftretenden Zeichen einen kurzen Code zu vergeben wurde erreicht. Das "A", welches am häufigsten vorkommt hat den kürzesten Code "00" erhalten.

Wie viel Speicher wird nun benötigt? Dazu können wir nochmals den Baum zur Hilfe nehmen. Wir notieren unter jedes Zeichen die Länge des Codes (Astes) und multiplizieren diese mit der Anzahl Vorkommen. (diese haben wir ja schon)




| 2 | 3 | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 9 | 9 | 12 | 8 | 8 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | **98** |

Es werden also 98 Bits verwendet, diese zählen zu den Nutzdaten. Der Baum (Metadaten bzw. das Wörterbuch) müssten zusätzlich noch übertragen werden. Wenn wir nun nur die Nutzdaten anschauen, sparen wir mit 98 Bit gegenüber dem erweiterten ASCII (28 * 8 Bit = 224 Bit) enorm Speicherplatz ein.

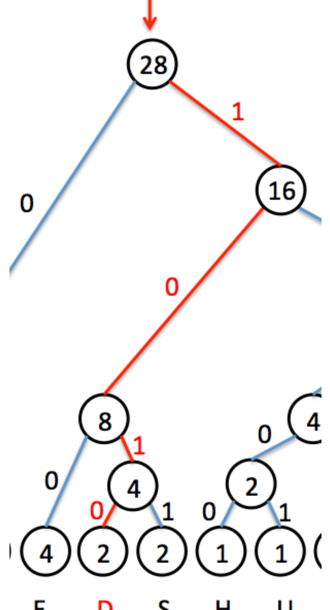
### Dekompression des Huffman Codes

Bei Dekomprimieren von Inhalten, welche nach dem Huffman Verfahren verkleinert wurden benötigt man 2 Angaben: Die Nutzdaten (Bitfolge) Den Baum

Nutzdaten:
10100010110101100000110011011010101010011010010110 11 …

Um die Originaldaten herzustellen nehmen wir die Bits aus den Nutzdaten und folgen dem Pfad, welcher uns die Nutzdaten vorgeben. Es wird immer oben im Baum begonnen:

Rechts ist sichtbar wie die Wiederherstellung funktioniert. Der Code der Nutzdaten "1010" wird durchlaufen bis ein Blatt (Buchstabe) erreicht ist. In diesem Fall der Buchstabe "D". Nach diesen 4 Bit wird wieder oben im Baum angefangen usw. Dies geschieht solange bis alle Nutzdaten "ausgepackt" worden sind. Wir erkennen, dass die Dekodierung nur möglich ist, wenn am Anfang der Nutzdaten begonnen wird! ➔**2**

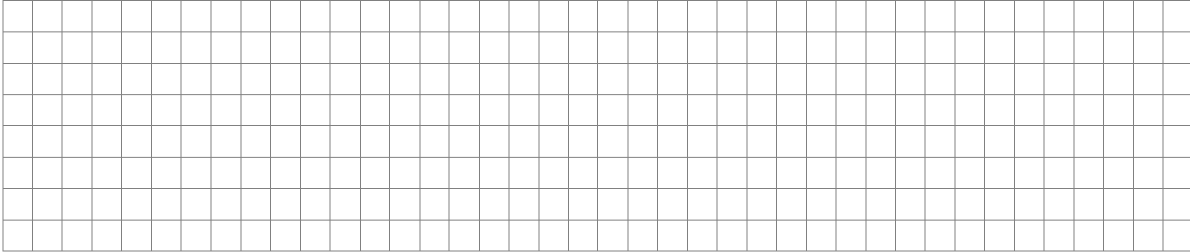


## Kompressionsrate und Kompressionsfaktor S. 183

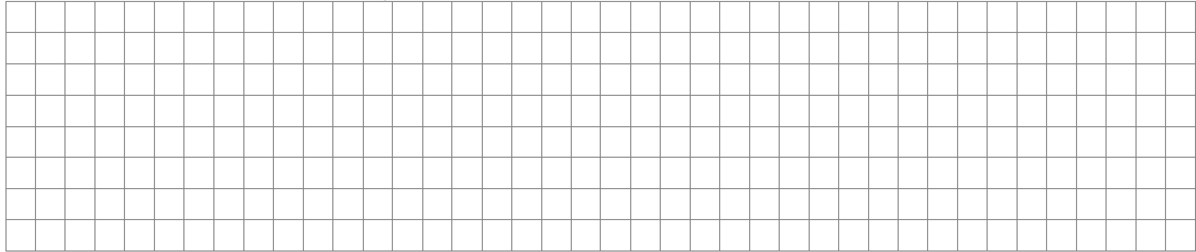
Um die eingesparte Datenmenge zu beziffern gibt es verschiedene Methoden. Wir betrachten diese anhand eines Beispiels:

Originaldatei: 800 kB Komprimierte Datei: 200 kB

Die **Kompressionsrate** sagt aus um, wieviel mal grösser die Originaldaten waren. Auf der rechten Seite des Vergleichs steht immer eine 1. In diesem Beispiel beträgt sie 4:1
Wie funktioniert die Berechnung?



Der **Kompressionsfaktor** sagt aus, um wie viel mal kleiner die komprimierten Daten sind. Auf der linken Seite des Vergleichs steht immer eine 1. In diesem Beispiel beträgt sie 1:4
Wie funktioniert die Berechnung?



Eine andere Variante der Grössenangabe bilden die Prozentzahlen. Dabei muss genau auf die Aussage geachtet werden!

Die Dateigrösse hat um 30% abgenommen.
Die komprimierte Datei ist nur noch 30% so gross wie das Original.

Die beiden Aussagen klingen zwar ähnlich, drücken aber andere Tatsachen aus. ➔*3*

## Praxis: ZIP und TAR.GZ

Wir alle haben sie schon 100x erhalten, entpackt und die enthaltenen Dateien verwendet. Bevorzugterweise handelt es sich dabei um einen Vorgang, ausgelöst durch die rechte Maustaste oder Doppelklick, welcher innerhalb kürzester Zeit funktioniert, keine Probleme macht und niemandem erklärt werden muss. Was kann so .zip Archiv alles?

1. Reduzieren des Speicherplatzes durch Kompression
2. Zusammenfassen mehrerer Dateien und Verzeichnisse in eine Datei (Archiv)

Für das Zusammenfassen wird innerhalb der .zip Datei eine eigene Ordnerstruktur angelegt, welche dann die Dateien innerhalb von .zip referenziert.
Die Reduktion des Speicherplatzes wird unter anderem mit dem Huffman Code erreicht, das genaue Verfahren heisst Deflate. Im .zip Format werden die Dateien einzeln komprimiert, dadurch können leichter neue Dateien hinzugefügt oder Dateien gelöscht werden innerhalb der .zip Datei. Was das in der Praxis bedeutet, können wir in den Übungen noch klären.
Je nach Speichermedium kann das Archiv in mehrere kleine Archive aufgeteilt werden. So können grosse Datenbestände trotzdem auf CD gebrannt oder per USB Stick transportiert werden. Dies ist ebenfalls praktisch bei der Verwendung des Dateisystems FAT32, welches eine Dateiobergrösse von 4 GB hat.

Auf linuxbasierten Systemen wird anders komprimiert, da finden wir .tar.gz Dateien. Auch dieses (zusammengesetzte) Format übernimmt die beiden Funktionen:

1. tar: Zusammenfassen mehrerer Dateien und Verzeichnisse in eine Datei (Archiv)
2. gz: Reduzieren des Speicherplatzes durch Kompression

tar ist die Abkürzung für "Tape Archive", welches für die alten Magnetbänder steht. Anders als beim .zip Format werden im .tar die einzelnen Dateien direkt hintereinandergereiht, ohne eine zusätzliche Ordnerstruktur innerhalb des Archivs abzubilden. Selbstverständlich können aber auch Verzeichnisse und Dateien in einem Archiv enthalten sein, lediglich die interne Speicherung ist unterschiedlich.
Das gz Format zum Komprimieren funktioniert praktisch gleich wie bei .zip, nämlich mit dem Deflate Algorithmus.

Auch unter Linux kann in der grafischen Oberfläche einfach mit .tar.gz Dateien umgegangen werden. Da es aber sehr peinlich werden kann, wenn man in der Shell nicht mit diesen Dateien umgehen kann, üben wir dies. ***➔4***
Hilfreich ist diese Tabelle:

| Optionen von tar | |
|---|---|
| **Option** | **Beschreibung** |
| `--help` | Zeigt eine vollständige Übersicht über alle Optionen. |
| `--version` | Gibt die installierte Version von tar aus. |
| `-c` | Ein neues Archiv erzeugen. |
| `-d` | Dateien im Archiv und im Dateisystem miteinander vergleichen. |
| `-f` | Archiv in angegebene Datei schreiben. / Daten aus angegebener Datei lesen. |
| `-j` | Archiv zusätzlich mit bzip2 (de)komprimieren. |
| `-J` | Archiv zusätzlich mit xz (de)komprimieren. |
| `-k` | Das Überschreiben existierender Dateien beim Extrahieren aus einem Archiv verhindern. |
| `-p` | Zugriffsrechte beim Extrahieren erhalten. |
| `-r` | Dateien an ein bestehendes Archiv anhängen. |
| `-t` | Inhalt eines Archivs anzeigen. |
| `-u` | Nur Dateien anhängen, die jünger sind als ihre Archiv-Version. |
| `-v` | Ausführliche Ausgabe aktivieren. Hierbei ist zu beachten, dass man dies möglichst an den Anfang des Befehls anhängt, wenn mehrere Optionen kombiniert werden. Z.B. würde `-cfv` zu einer Fehlermeldung führen. Korrekt wäre `-vcf`. |
| `-w` | Jede Aktion bestätigen. |
| `-x` | Dateien aus einem Archiv extrahieren. |
| `-z` | Archiv zusätzlich mit gzip (de)komprimieren. |
| `-Z` | Archiv zusätzlich mit `compress` (de)komprimieren. |
| `-A` | Inhalt eines bestehenden Archivs in ein anderes Archiv kopieren. |
| `-M` | Mehrteiliges Archiv anlegen/anzeigen/extrahieren. |
| `-L` | Medium wechseln, wenn ZAHL KBytes geschrieben sind. |
| `-W` | Archiv nach dem Schreiben prüfen. |

(Quelle: http://wiki.ubuntuusers.de/tar)

## Weitere Themen für Interessierte

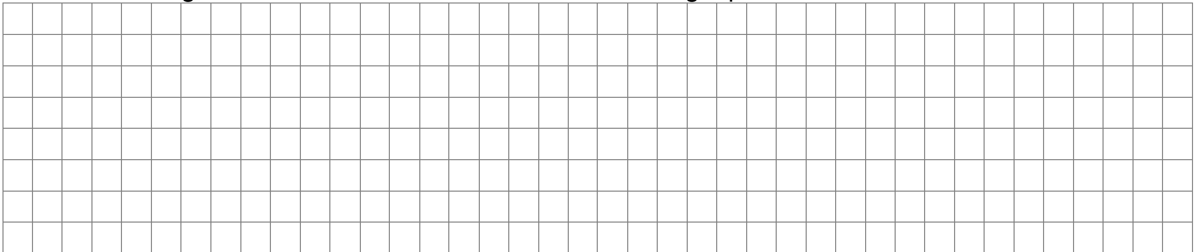
Falls Ihnen das Thema zusagt, können Sie eigene Recherchen zu den folgenden Themen durchführen:

- LZW Algorithmus (siehe Video im Ordner)
- 4 Komprimierungs- und Archivlösungen im Vergleich (http://www.tomshardware.de/archivierung-zip-kompression,testberichte-240497.html)

# Aufgabenteil

## 1) Lauflängencodierung Binär

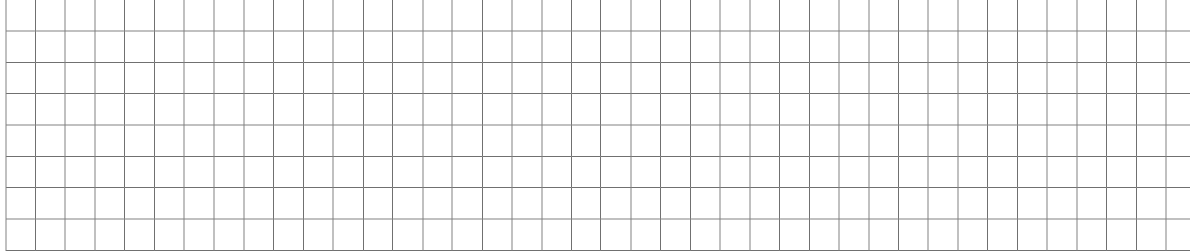
Codieren Sie die 3 Zeilen der Bitmapgrafik neu und verwenden Sie die maximale Länge von 15 aufeinanderfolgenden Bits. Wird in der Summe etwas eingespart?



Im Internet habe ich folgende Erklärung zur Lauflängencodierung gefunden:

Originaldaten Binärfolge: 0000000000111100000111111111
Wird komprimiert als: 10 4 5 8

Analysieren Sie diese "Methode" und besprechen Sie die Machbarkeit und Problematiken dieses Beispiels mit Ihrem Pultnachbarn!



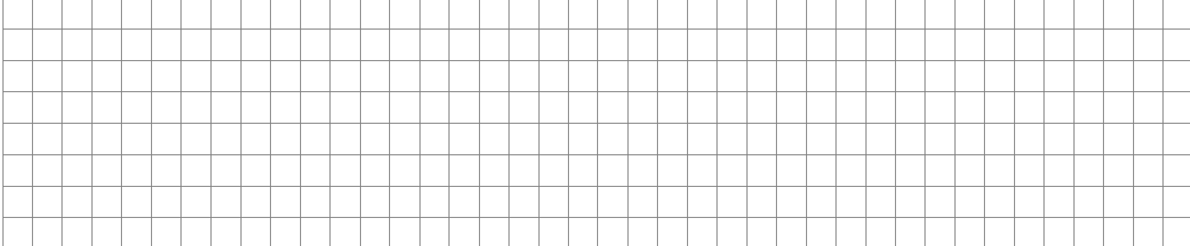
## 2) Huffman Code

Sie denken sich eine Nachricht aus und erstellen auf einem Blatt Papier den Baum zur Huffman Codierung. Sie notieren den Baum sowie die codierte Nachricht auf einem Blatt und deponieren dieses in einer von der Lehrperson bereitgestellten Schachtel. Nehmen Sie ein anderes Blatt aus der Schachtel und decodieren Sie diese! Sie sollten mindestens eine Nachricht codiert und 2 andere decodiert haben (dauert weniger lang).
Nehmen Sie sich Zeit für die Codierung und verzweifeln Sie nicht, wenn der Baum ein wenig seltsam aussieht. Sie können ihren Baum vor dem Deponieren selber kontrollieren indem Sie das JCrypTool auf die bmLP1 kopieren und verwenden. Das Tool finden Sie bei den Modulunterlagen. "Visualisierungen" -> "Huffman Kodierungen". Zeit: 20 Minuten

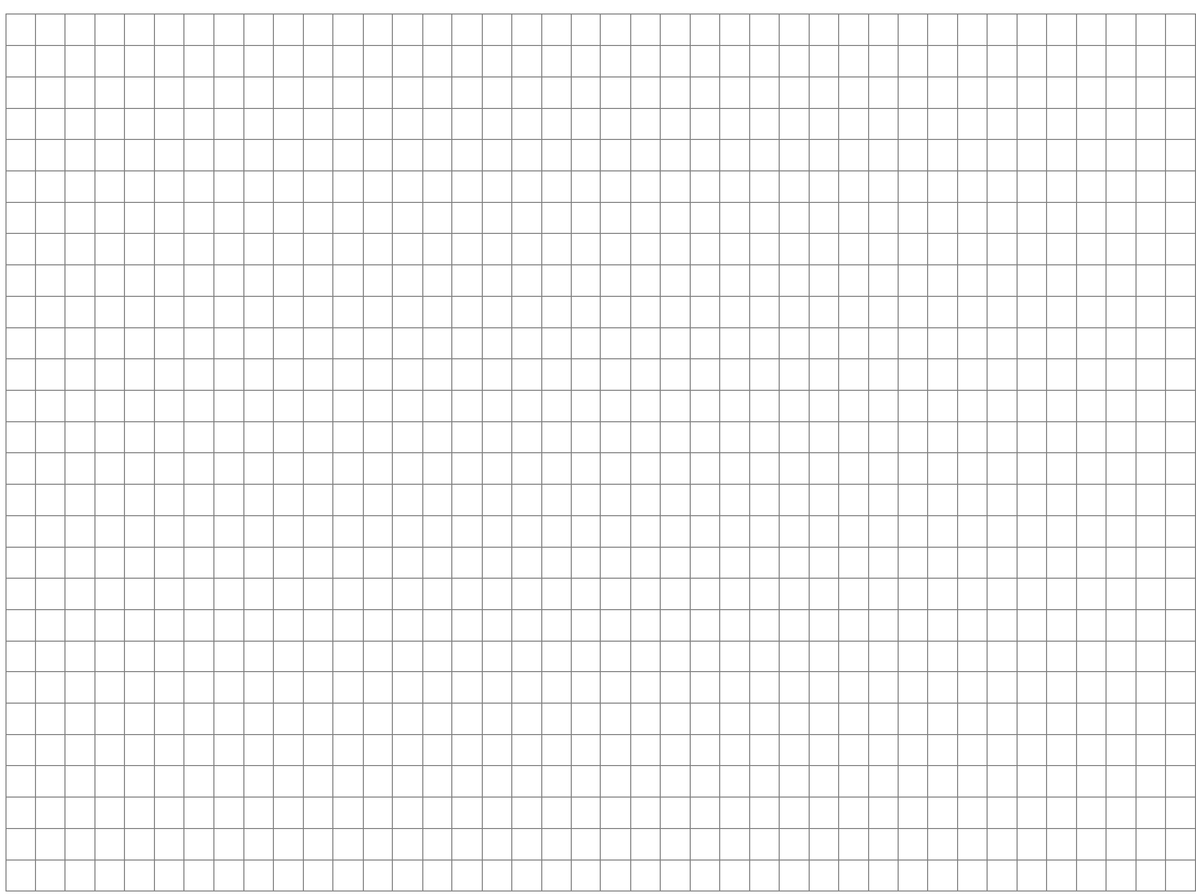
## 3) Kompressionsrate und Kompressionsfaktor

Berechnen Sie die Kompressionsrate und den Kompressionsfaktor der beiden Aussagen:

a) Die Dateigrösse hat um 30% abgenommen.
b) Die komprimierte Datei ist nur noch 30% so gross wie das Original.



Berechnen Sie Rate und Faktor aller Beispiele aus dem Theorieteil!



## 4) .zip und .tar.gz

Starten Sie auf der virtuellen Maschine bmLP1 ein Shellfenster.
a) Komprimieren Sie einen Ordner des Moduls 114 in eine neue Datei. Bsp: Ordner "AB114-05" als Datei "M114-Bildcodierung.tar.gz". Wie lautet der korrekte tar Befehl?

b) Nun möchten Sie noch einen weiteren Ordner "AB114-01" ins gleiche Archiv hinzufügen, wie lautet der Befehl?

c) Sie wollen kontrollieren welche Dateien ins Archiv aufgenommen wurden, wie lautet der Befehl um den Inhalt des Archivs anzuzeigen?

d) Wie gross ist der Kompressionsfaktor dieser Datei?

e) Sie wollen das erstellte Archiv in einem testordner wiederherstellen, wie lautet der Befehl?

f) Kopieren Sie eine bereits komprimierte Datei /var/log/syslog.2.gz in Ihre Homeverzeichnis und versuchen Sie diese zu entpacken!

Windows Übungen auf der bmWP2 mit .zip

g) Kopieren Sie aus dem Ordner AB114-06 Dateien das Foto "AB114-06 A5.JPG" in ein eigenes Verzeichnis und duplizieren Sie dieses, so dass Sie 10x die gleiche Datei haben. Zippen Sie nun alle 10 Dateien in ein neues Archiv. Wie gross wird dieses? Kompressionsfaktor?

h) Zippen Sie diese Datei nochmals, wie ist der Kompressionsfaktor jetzt?