Bild II

Kompression

Grafik-Formate

1. Grundlagen

- Da **Rastergrafiken** oftmals viel Speicherplatz benötigen, spielt für diese Formate die Datenkompression eine besonders große Rolle.

- Neben der **Einsparung von Speicherplatz** ist die **Verringerung der notwendigen Übertragungsbandbreite** ein wichtiges Ziel, besonders für die im Internet gebräuchlichen Formate **JPEG**, **GIF**, aber auch für **Audio-** oder **Videoformate**.

- Man unterscheidet:
  - unkomprimierte Formate,
  - Formate mit ***verlustloser Kompression*** (engl. lossless),
  - Formate mit ***verlustbehafteter Kompression*** (engl. lossy).

- Bei verlustbehafteten Formaten ist die Bildqualität durch Parameter steuerbar.

1. Grundlegende Kompressionsverfahren

Man unterscheidet drei verschiedene Ansätze zur Datenkompression:

1. **RLE - Run-length Encoding** (*Lauflängenkodierung*)

   Eine Gruppe von Kompressionsverfahren, die Daten komprimieren, indem sie eine Folge von gleichen Zeichen (run-length) durch einen Code oder ein Token ersetzen, der das Zeichen und die Länge der Zeichenfolge enthält.

   RLE wird häufig als Bestandteil mehrschrittiger **wörterbuchbasierter** oder **statistischer Kompressionsverfahren** eingesetzt.

2. **Statistische Kompressionsmethoden**

   Statistische Kompressionsmethoden beziehen **statistische Eigenschaften der Daten**, die komprimiert werden sollen, bei der Kompression mit ein, um ihnen **Codes von variabler Länge** zuzuweisen. Dies geschieht in Abhängigkeit von der Häufigkeit des Auftretens.
   Das Morsealphabet ist z.B. so aufgebaut, dass im Englischen häufig auftretende Zeichen kurze Codes, und selten auftretende Zeichen lange Codes haben.

   – **Huffman-Kodierung**

3. **Wörterbuchbasierte Kompressionsmethoden**

   Kompressionsverfahren, die Teile (Token) der Daten in einem „**Wörterbuch**“ speichern. Wenn ein eingelesener String mit einem Eintrag im Wörterbuch übereinstimmt, wird nur ein Pointer auf diesen Eintrag gespeichert. Auf diese Weise speichert man das betreffende Wort 1x kann aber evtl. mehrfach in der Datei darauf verweisen (→ Kompression!).

   – **LZW-Algorithmus**

1.2 RLE (Run Length Encoding)

1.2.1 Charakteristik

- **RLE** ist ein verlustfreies Kompressionsverfahren.

- **RLE** (dt. Lauflängenkodierung) nutzt lange Folgen sich wiederholender. Zeichen oder Zeichenketten, sog. Läufe (runs) , aus (→ Redundanz!).

  Bsp.:
  Statt **AAAABBBCCCCDEEEEE**
  könnte man auch **4A3B4CD5E**
  schreiben.

- Funktioniert am besten bei Daten, in denen **lange Folgen gleicher Zeichen** enthalten sind.
  (→ z.B. Monochrome Grafiken = viel weiß, wenig schwarz.)

- Bei Daten, mit hoher Entropie („große Unordnung“) kann RLE sogar zu negativer Kompression führen, da die Codierung unter Umständen mehr Platz in Anspruch nimmt, als die Originaldaten.

  Bsp.:
  Enthält eine Datei viele verschiedene **Zeichenketten der Länge 1** (einzelne Zeichen) dann wird für jedes Zeichen **x** folgendes gespeichert:

  - **1x** statt einfach nur **x**

    → Verdopplung des Speicherbedarfs

1.2.2 Einfache RLE-Methoden zur Kompression von **Binärdateien**

- Gegeben sei folgende binäre Zeichenfolge: 0001111000000111111
  Um in Binärdateien Daten zu komprimieren, kann man verschiedene Methoden anwenden, z.B. diese beiden:

- *Methode 1:*
  - Man speichert jeweils eine Kombination aus dem Zeichen und seiner Lauflänge nach dem obigen Muster.
  - Aus der Zeichenfolge oben wird so (3,0)(4,1)(5,0)(6,1).
  - Wenn man zusätzlich festlegt, dass der erste Wert immer eine Null betreffen muss und sich danach 0 und 1 immer abwechseln, kann man die Kodierung abkürzen zu 3 4 5 6.
  - In der Regel steht eine feste Anzahl Bits für die Lauflängenangaben zur Verfügung. Falls die Blocklänge für diese Bitzahl zu groß sein sollte, muss der Block geteilt werden.

- *Methode 2:*
  - Man speichert die Position und Länge der 1-Blöcke eines Bildes.
    → Für das obige Beispiel erhält man die Ausgabe: (3,4)(12,6).

1.2.3 Kodierung mit variabler Länge

- Diese Art der Lauflängenkodierung ist z.B. für Textdateien oder für detailreiche Bilder nicht besonders effizient, da hier keine langen Folgen gleicher Zeichen auftreten.

- Besonders bei Textdateien ist es also geschickter, nicht jedes Zeichen gleichmäßig mit 7, 8 oder gar 16 Bit darzustellen, sondern für Zeichen, die häufig vorkommen, wenige Bits zu benutzen, für Zeichen, die selten vorkommen mehr Bits.

1.3 Statistische Verfahren: **Huffman-Kodierung**

1.3.1 Motivation

- In natürlichen Sprachen werden bestimmte Buchstaben häufiger benutzt als andere. Im Englischen z.B. sind **E** und **T** die Buchstaben mit der höchsten Frequenz. Dieser Umstand wurde beim Entwurf des Morsealphabets berücksichtigt. Hier haben die Buchstaben mit der höchsten Frequenz **kurze** Codes, während die Buchstaben mit der niedrigsten Frequenz **lange** Codes haben.

- In der Computerwelt werden Alphabete meistens mit einer **festen Länge** kodiert (z.B. ASCII). Jedes Zeichen verwendet eine feste Anzahl von Bits, ganz egal wie häufig es verwendet wird.
  Wenn es um eine hohe **Ausführungsgeschwindigkeit** geht, ist dies sehr **effizient**. Bei Anwendungen wie der Bilddatenkompression, wo **Speicherplatz** das wichtigste Kriterium ist, sind Codes mit **variabler Länge** jedoch oft sinnvoller.

1.3.2 Charakteristik

- Das allgemeine Verfahren zur Bestimmung des variablen Codes ist 1952 von D. Huffman entwickelt worden und wird deshalb Huffman-Kodierung genannt.

- Huffman-Kodierung ist **verlustfrei**.

- Jeder beliebige Baum mit ***M*** äußeren Knoten kann benutzt werden, um jede beliebige Zeichenfolge mit ***M*** verschiedenen Zeichen zu kodieren.

- Die beste Lösung, um eine solche Zeichenfolge zu kodieren ist ein **Binärbaum**

Wie wird ein solcher Binärbaum aufgebaut?

1. Im ersten Schritt wird die Häufigkeit jedes Zeichens in der zu kodierenden Zeichenfolge ermittelt. Zum Beispiel:

„A MAN A PLAN A CANAL PANAMA“

Wert	**Frequenz**
A	10
C	1
L	2
M	2
N	4
P	2
LEER	6
. (Punkt)	1

1. Als nächstes wird der Baum entsprechend der Häufigkeiten der Zeichen aufgebaut.
2. Wir erzeugen einen binären Baum, mit Knoten, in denen die Häufigkeiten gespeichert werden.
3. Die beiden Knoten mit den kleinsten Häufigkeiten werden ausgewählt und es wird ein neuer Knoten **darüber** erzeugt. Die Häufigkeit des so entstandenen Knotens entspricht der Summe der Häufigkeiten seiner beiden Kinder.
4. Bei jedem Schritt verringert sich so die Zahl der Knoten um 1 (da zwei Knoten durch einen ersetzt werden), solange bis alle Knoten durch einen einzigen Baum verbunden sind.
5. Den Code jedes Knotens erhält man, wenn man den jeweiligen Pfad von der Wurzel bis zum Knoten durchläuft. Für jedes Mal „**links abbiegen**“ notiert man eine **0**, für jedes Mal „**rechts abbiegen**“ eine **1.**

Beispiel

- Zur Beschreibung des Huffman-Verfahrens verwenden wir folgendes Palindrom:

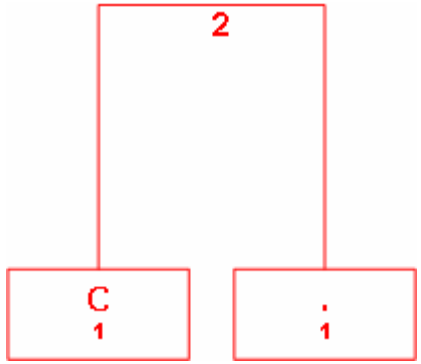
A MAN A PLAN A CANAL PANAMA

- Es besteht aus acht verschiedenen Zeichen mit den folgenden Häufigkeiten:

Wert	**Frequenz**
A	10
C	1
L	2
M	2
N	4
P	2
LEER	6
. (Punkt)	1

Beispiel

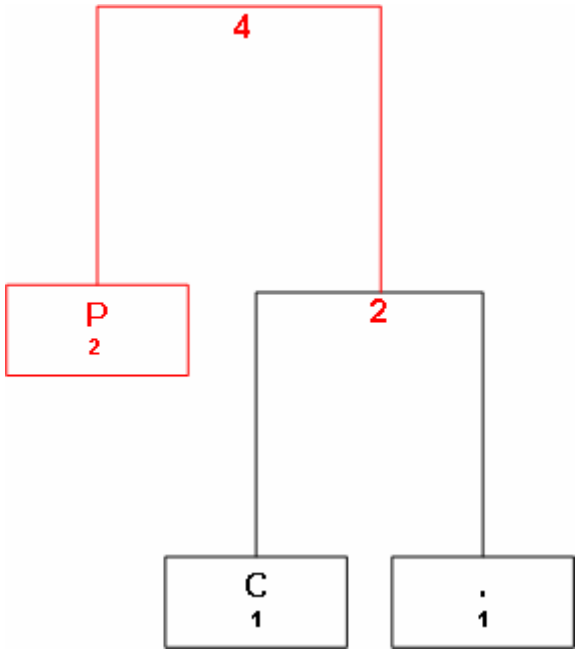
- Die Zeichen „**C“** und „**.“** (Punkt) haben die niedrigste Frequenz. Daher werden sie als erstes verwendet, um einen neuen Knoten zu erzeugen. Als Wert erhält dieser neue Knoten die Summe der Häufigkeiten seiner Kinder „**C“** und „**.“**:



Wert	Frequenz
A	10
C	**1**
L	2
M	2
N	4
P	2
LEER	6
. (Punkt)	**1**

Beispiel

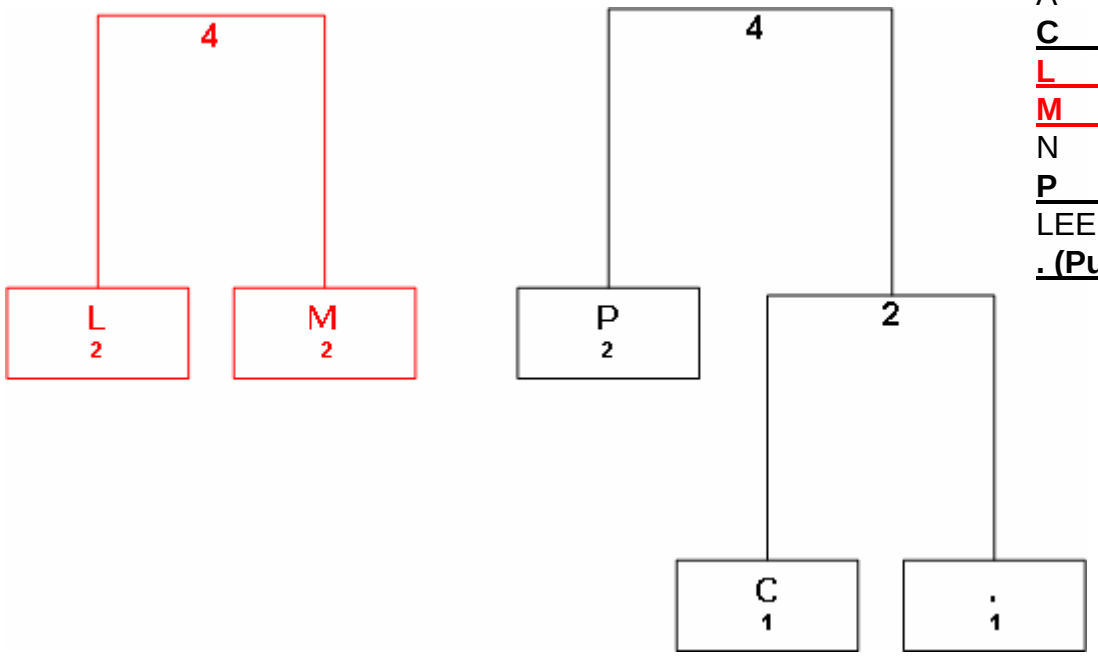
- Unter den übrigen Zeichen befinden sich 4 Zeichen mit der nun niedrigsten Frequenz von 2. Wir nehmen **P** und den im vorherigen Schritt erzeugten Knoten und verbinden beide. Wir erhalten einen neuen Knoten mit einer Frequenz von 4:



Wert	Frequenz
A	10
C	**1**
L	2
M	2
N	4
P	**2**
LEER	6
. (Punkt)	**1**

Beispiel

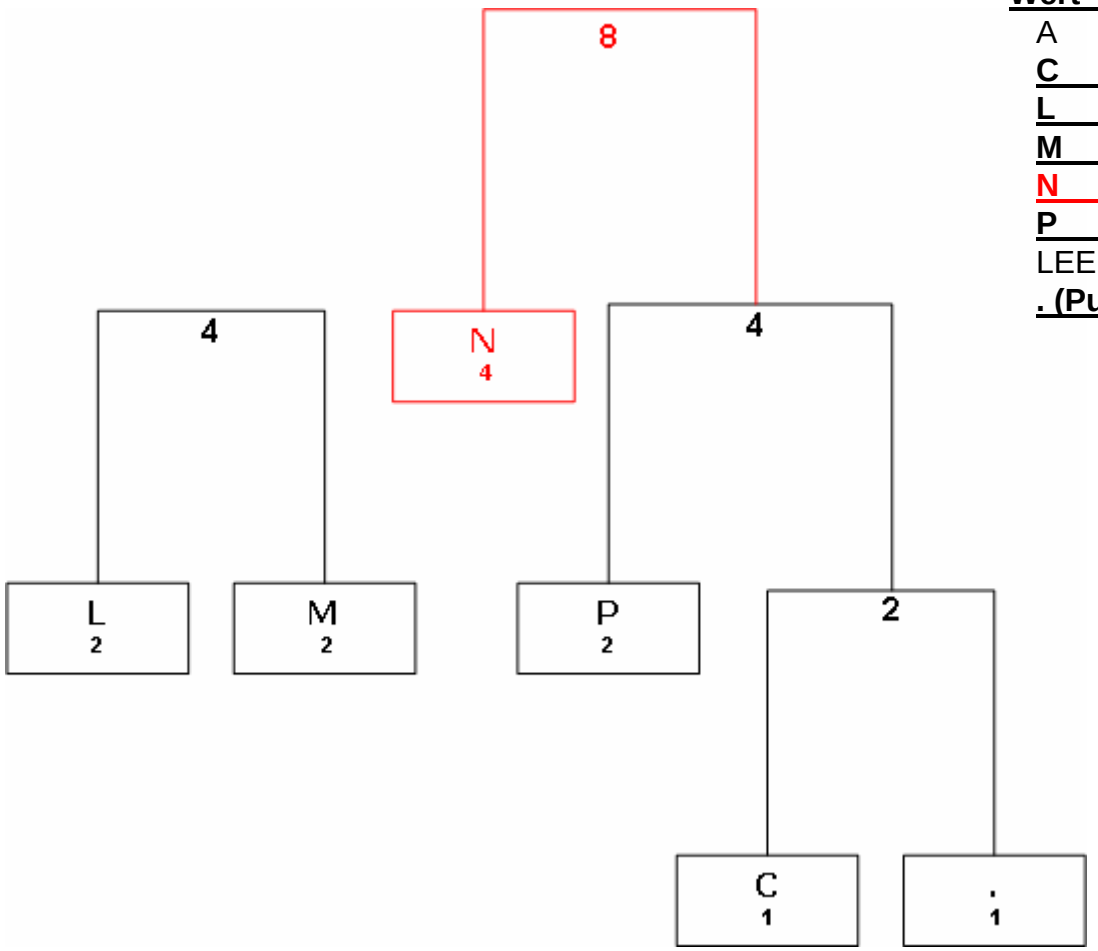
- Nun haben **L** und **M** die niedrigste Frequenz. Da alle vorhandenen Knoten in dem Baum eine höhere Frequenz haben, müssen wir einen neuen Knoten erzeugen:



Wert	Frequenz
A	10
C	**1**
L	**2**
M	**2**
N	4
P	**2**
LEER	6
. (Punkt)	**1**

Beispiel

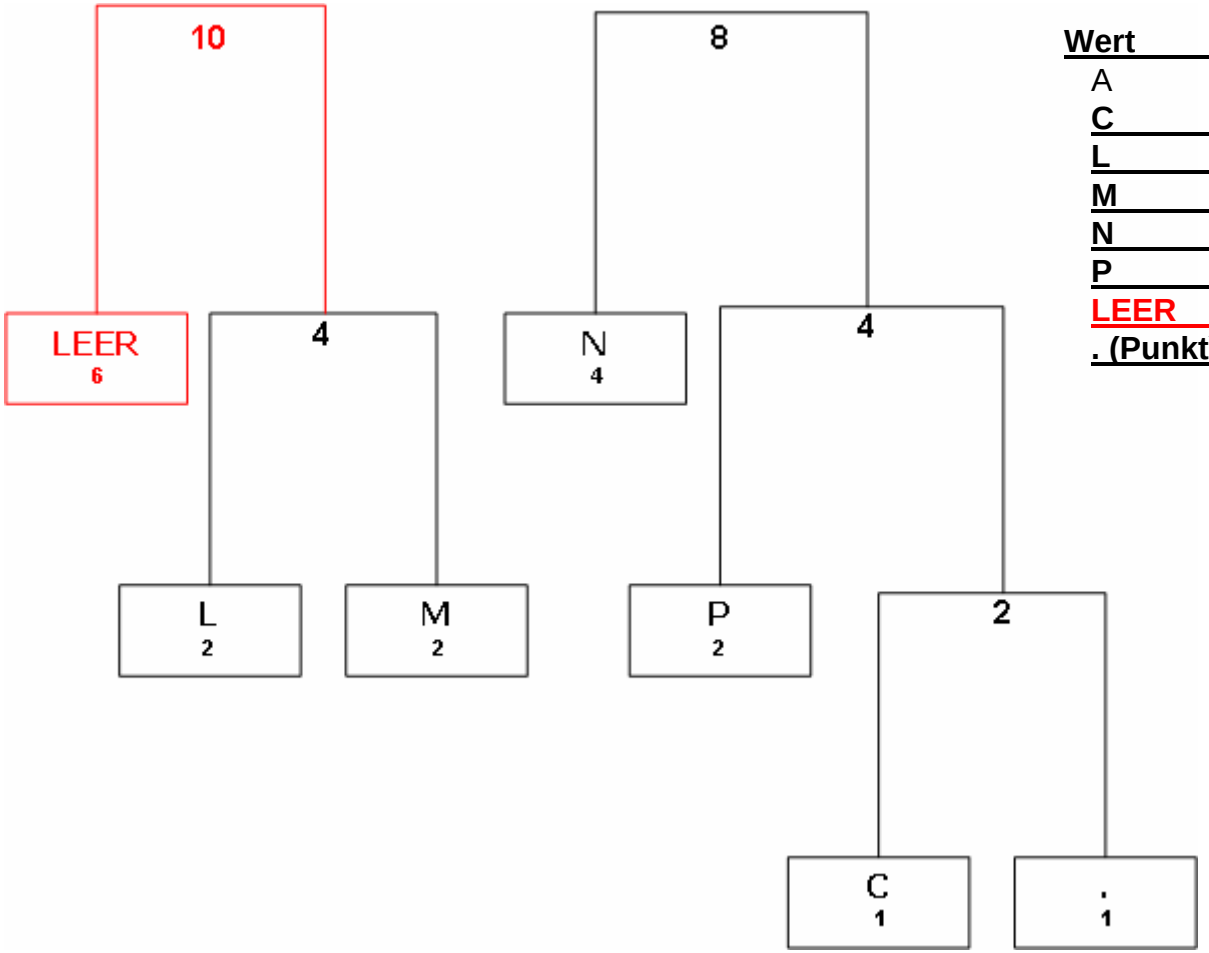
- Nun haben **N** und die beiden zuletzt erzeugten Knoten die niedrigste Frequenz. Wir fügen **N** dem Baum hinzu:



Wert	Frequenz
A	10
C	**1**
L	**2**
M	**2**
N	**4**
P	**2**
LEER	6
. (Punkt)	**1**

Beispiel

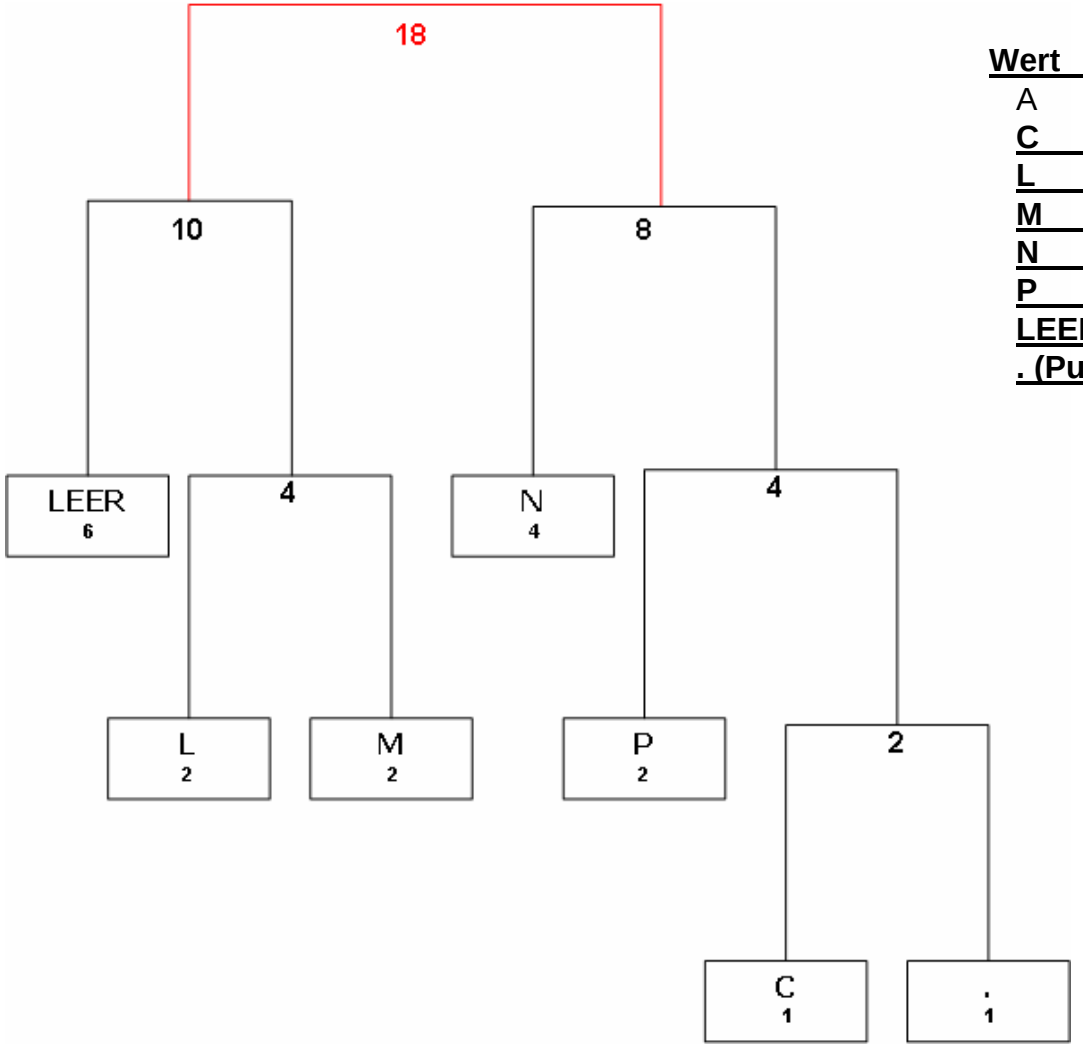
- Das Leerzeichen und der **L-M**-Teilbaum haben nun die niedrigste Frequenz. Daher fügen wir sie zusammen:



Wert	Frequenz
A	10
C	1
L	2
M	2
N	4
P	2
LEER	6
. (Punkt)	1

Beispiel

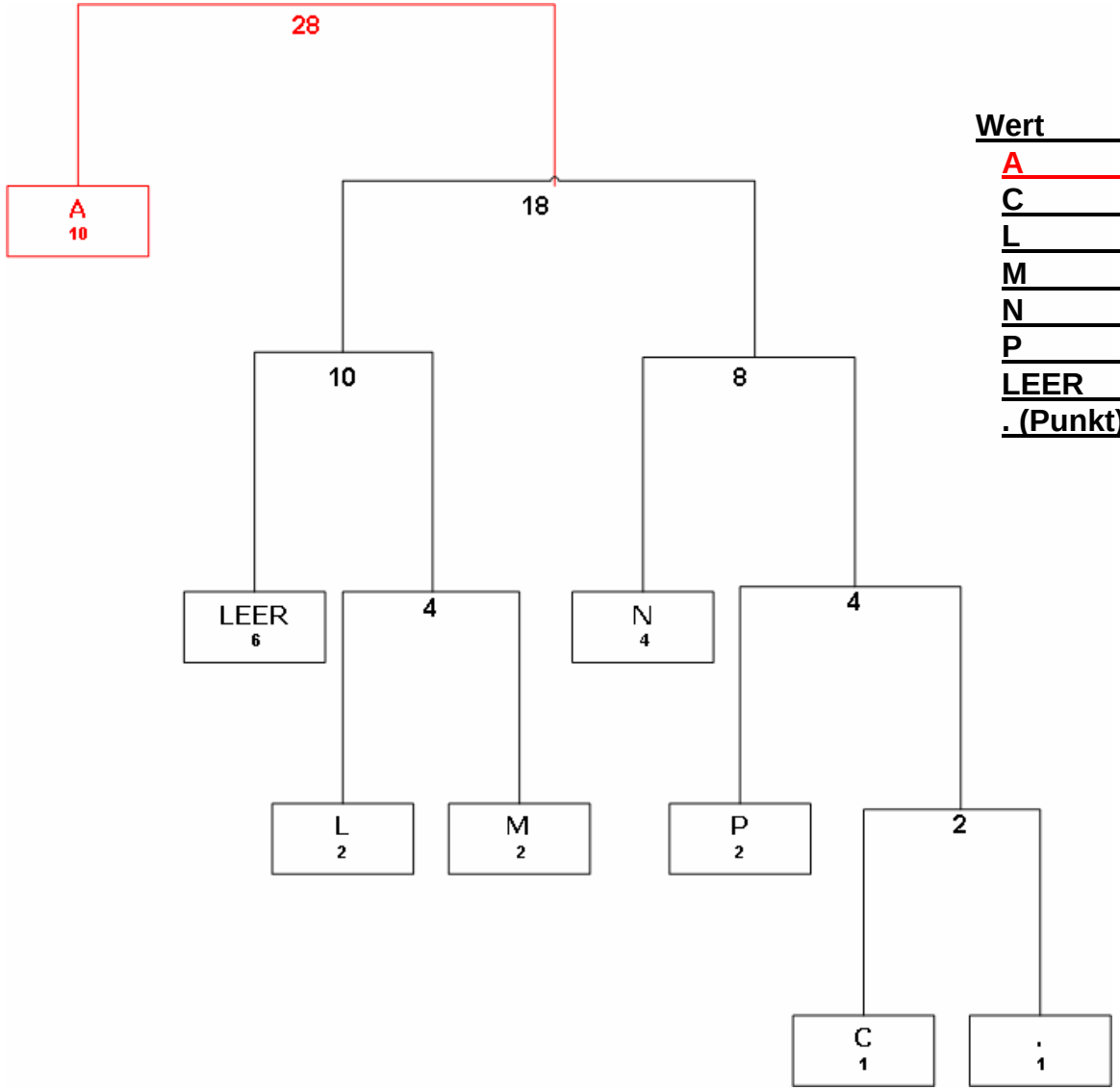
- Nun haben wir zwei Knoten (Teilbäume) und es fehlt noch der Buchstabe **A**. Wir beschließen zunächst **willkürlich** beide Teilbäume zu verbinden:



Wert	Frequenz
A	10
C	**1**
L	**2**
M	**2**
N	**4**
P	**2**
LEER	**6**
. (Punkt)	**1**

Beispiel

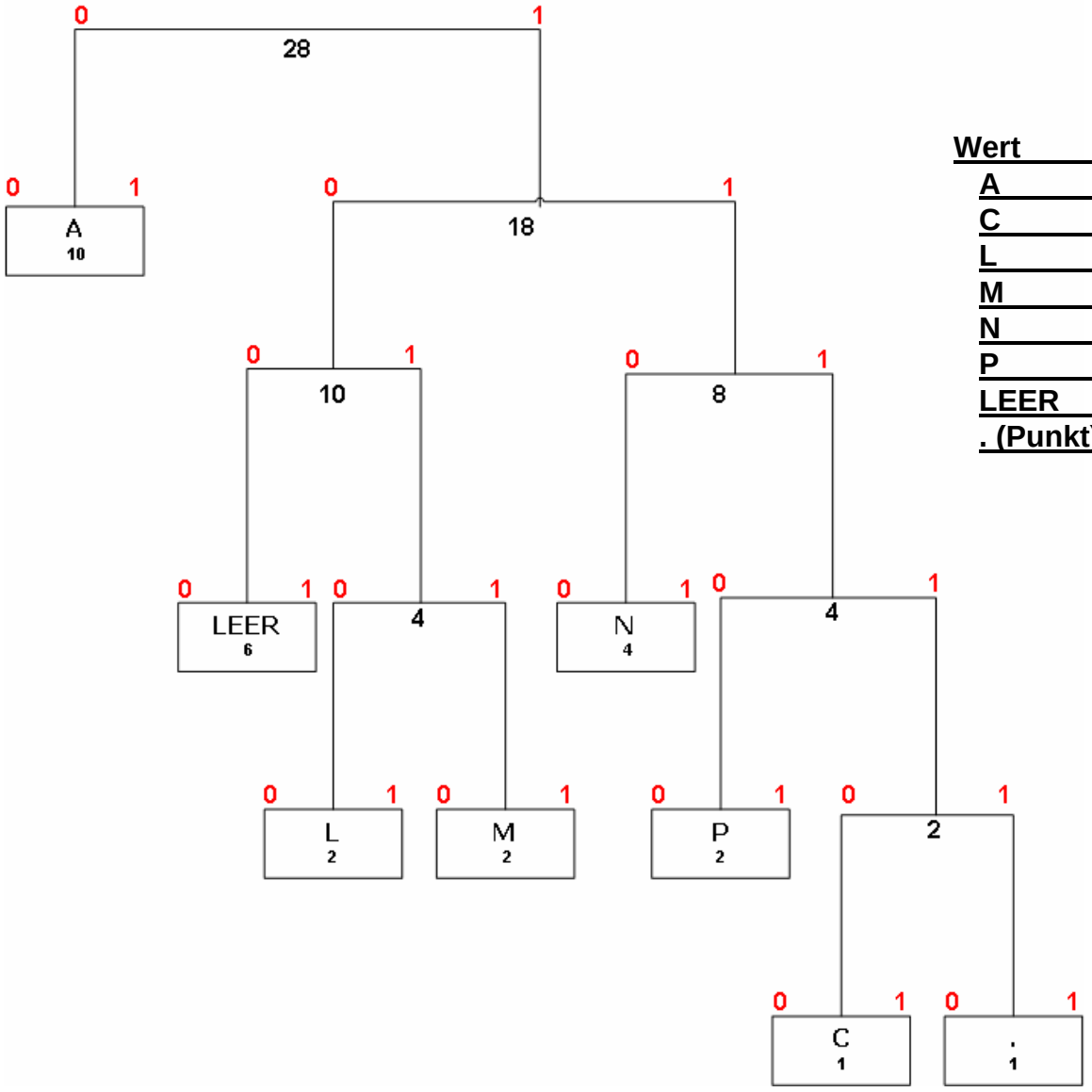
- Schließlich fügen wir den Buchstaben **A** hinzu…



Wert	Frequenz
A	10
C	1
L	2
M	2
N	4
P	2
LEER	6
. (Punkt)	1

Beispiel

- …und markieren jeden **linken Ast mit 0** und jeden **rechten Ast mit 1**:



Wert	Frequenz
A	10
C	1
L	2
M	2
N	4
P	2
LEER	6
. (Punkt)	1

Beispiel

- Wir generieren den Huffman-Kode für jedes Zeichen, indem wir den Code 0 oder 1 auf dem Pfad zwischen der **Wurzel** und dem **Knoten** für das entsprechende Zeichen notieren.
  Die folgende Tabelle zeigt den Huffman-Kode für jeden Wert:

Wert	**Huffman-Code**	**Länge**	**Frequenz**	**Verwendete Bits**
A	0	1	10	10
C	11110	5	1	5
L	1010	4	2	8
M	1011	4	2	8
N	110	3	4	12
P	1110	4	2	8
LEER	100	3	6	18
.(Punkt)	11111	5	1	5
				GESAMT: 74 Bits

- Mit der Huffman-Kodierung kann unser Beispielpalindrom mit **74 Bits** kodiert werden. Würden wir ASCII-Codes zu jeweils 7 Bits verwenden, würden wir **196 (7 x 28) Bits** benötigen. Wir erhalten damit eine **Kompression von ca. 60%.**

1.4 Wörterbuchbasierte Verfahren: **LZW (Lempel-Ziv-Welch)**

1.4.1 Charakteristik

- LZW gehört zur Familie der LZ-Algorithmen
- Der LZW (Lempel-Ziv-Welch) -Algorithmus ist ein *Wörterbuch-basierter, substituierender*, ***verlustfreier*** Kompressionsalgorithmus.
- Der Algorithmus versucht, den zu komprimierenden Zeichenstrom in Teilketten zu zerlegen und diese in einem Wörterbuch zu speichern.
- Anschließend werden nur die Indizes in der betreffende Tabelle als Ausgangscode gespeichert.
- Anhand dieser Ausgangscodes läßt sich dann der ursprüngliche Zeichenstrom wieder generieren.

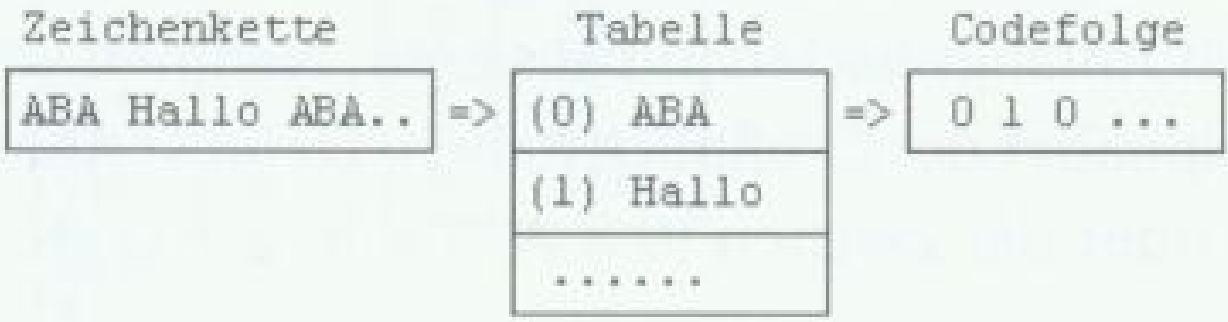


Abbildung 22.6 LZW-Encoding

1.4.2 LZW-Kodierer

- Der Kodierer beginnt mit der Initialisierung des Wörterbuches durch z.B. alle Symbole des Alphabets. Bei herkömmlichen 8 Bit pro Zeichen ist das Wörterbuch z.B. zunächst 256 (2^8) Einträge lang.

- Anschließend liest der Kodierer den Zeichenstrom der zu komprimierenden Datei Zeichen für Zeichen ein und akkumuliert die eingelesenen Zeichen im String **I**.

- Der so entstandene String **I** wird im Wörterbuch gesucht. Das nächste (erste) eingelesene Zeichen x wird im Wörterbuch gefunden.

- Solange **I** im Wörterbuch gefunden wird, wird das nächste Zeichen eingelesen und an den String angehängt. **I** = **I**x

- Wird der String **I**x im Wörterbuch nicht gefunden,
  - gibt der Kodierer den Zeiger von **I** auf das Wörterbuch aus,
  - schreibt den String **I**x an die nächste freie Stelle im Wörterbuch und
  - initialisiert den String **I** auf das zuletzt eingelesene Zeichen x.

1.4.3 LZW-Dekodierer

- Der Dekodierer beginnt mit der Initialisierung des Wörterbuches durch z.B. alle Symbole des Alphabets.

- Anschließend liest der Dekodierer den Zeichenstrom der zu dekomprimierenden Datei Zeichen für Zeichen ein. Die Zeichen sind die Wörterbuch-Zeiger.

- Die eingelesenen Zeiger werden benutzt, um aus dem Wörterbuch die unkomprimierten Zeichen bzw. Strings auszulesen.

- Der Dekodierer bildet gleichzeitig das Wörterbuch in derselben Weise wie der Kodierer. Man bezeichnet Kodierer und Dekodierer als *synchronisiert.*

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258

Wörterbuch:

…
a= 97
b=98
c=99
d=100
r=114
…

Beispiel

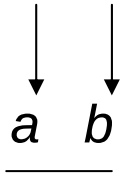
abracadabra = 97,98,114,97,99,97,100,256,258

a

Wörterbuch: unverändert

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258



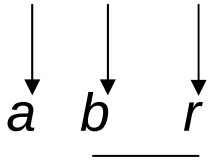
Wörterbuch:

…

neuer Eintrag: ab= 256

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258



Wörterbuch:

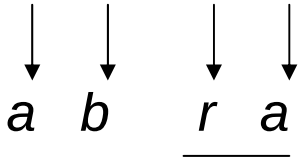
…

ab = 256

neuer Eintrag: br = 257

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258



Wörterbuch:

…

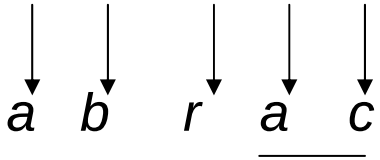
ab = 256

br = 257

neuer Eintrag: ra= 258

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258

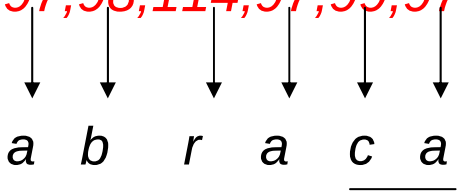


Wörterbuch:

…
ab = 256
br = 257
ra = 258
neuer Eintrag: ac = 259

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258



Wörterbuch:

…

ab = 256

br = 257

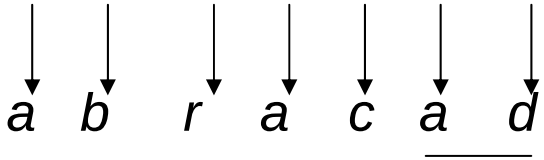
ra = 258

ac = 259

neuer Eintrag: ca= 260

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258

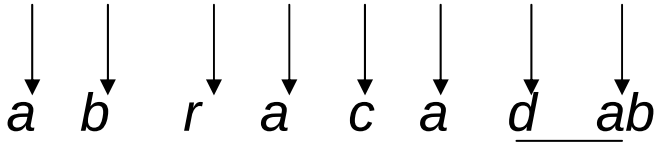


Wörterbuch:

…
ab = 256
br = 257
ra = 258
ac = 259
ca= 260
neuer Eintrag: ad = 261

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258

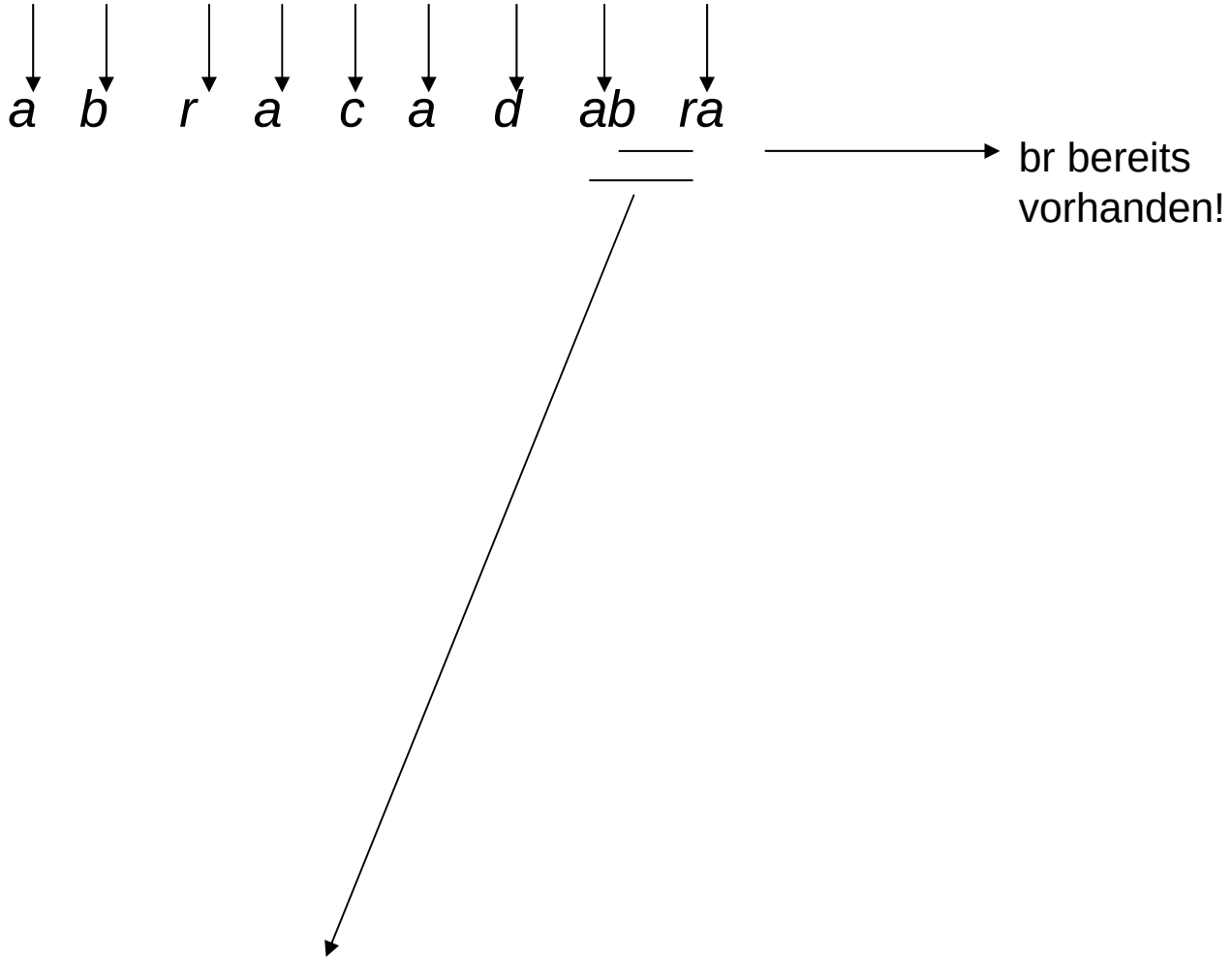


Wörterbuch:

…
ab = 256
br = 257
ra = 258
ac = 259
ca = 260
ad = 261
neuer Eintrag: da = 262

Beispiel

abracadabra = 97,98,114,97,99,97,100,256,258



Wörterbuch:

…
ab = 256
br = 257
ra = 258
ac = 259
ca = 260
ad = 261
da = 262

neuer Eintrag: abr = 263

1.4.4 Problemstellung bei der LZW-Kompression

- **Problem**:
  Die Codetabelle wird sich nur in den seltensten Fällen im Vorfeld bestimmen lassen.

- **Lösung**:
  Der Algorithmus muss die Wörterbuch-Tabelle bei der Komprimierung und Dekomprimierung selber aufbauen, denn sie wird nicht mit dem Code gespeichert
  (*um den Kompressionserfolg nicht durch Abspeichern des Wörterbuchs zu zerstören*).

2. Dateiformate für Bildinformationen

2.1 TIFF (Tagged Image File Format)

2.1.1 Charakteristik/ Basisstruktur

- Das TIFF (Tag[ged] Image File Format) geht auf eine gemeinsame Definition verschiedener Firmen (Aldus, HP, Microsoft etc.) zurück.
- Das TIFF-Format unterstützt u.a. LZW und Huffman-Kodierung (modifiziert) zur Komprimierung.
- Version: 6.0 .
- Eine TIFF-Datei besteht aus einem Header und einer variablen Zahl von Datenblöcken mit unterschiedlicher Länge, die über Zeiger adressiert werden.
- Die Struktur der Datei wird im wesentlichen durch die als *IFD (Image File Directory)* bezeichneten Blöcke geprägt.
- Diese IFDs bilden eine verkettete Liste innerhalb der Datei und enthalten (Meta-)Informationen bezüglich der gespeicherten Datentypen, der Bilddaten, des Grafikmodus, etc.
- Aus den IFDs verweisen Zeiger auf die eigentlichen Datenblöcke.

- Die Bilddaten werden in freien Bereichen innerhalb der Datei gespeichert.
- Der Aufbau der TIFF-Datei ist somit sehr flexibel und es können z.B. mehrere Bilder oder verschiedene Varianten eines Bildes innerhalb einer Datei gespeichert werden.

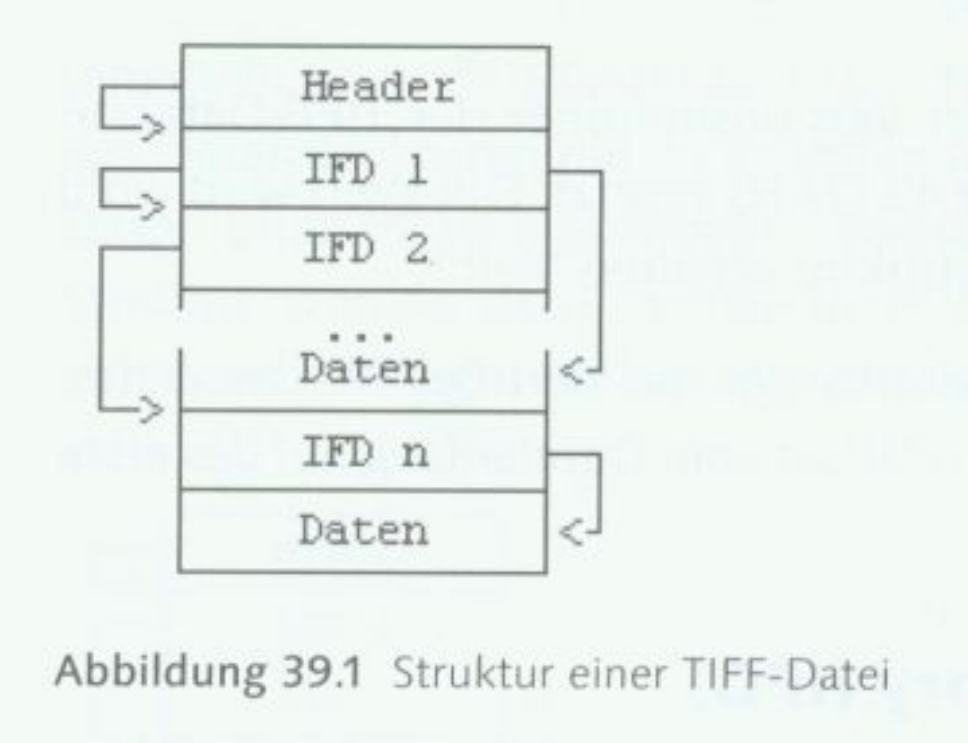


Abbildung 39.1 Struktur einer TIFF-Datei

2.1.2 TIFF-Header

- Der Header besitzt ein festes Format und belegt immer die ersten 8 Bytes der Datei.

Offset	Bytes	Bemerkungen
00H	2	Byte Order ("II" = Intel, "MM" = Motorola)
02H	2	Version Number (immer 2AH)
04H	4	Pointer to first IFD

2.1.3 Image File Directory

- Innerhalb der TIFF-Datei können die Daten beliebig angeordnet sein. Der Bezug darauf wird durch die IFDs vorgenommen.
- Das IFD funktioniert dabei als Inhaltsverzeichnis und Header für die eigentlichen Datenbereiche.
- Beginnend mit dem Header sind alle IFDs durch Zeiger verkettet.
- Die Länge eines IFDs ist variabel und wird durch die Anzahl der *Tag-Einträge* bestimmt.

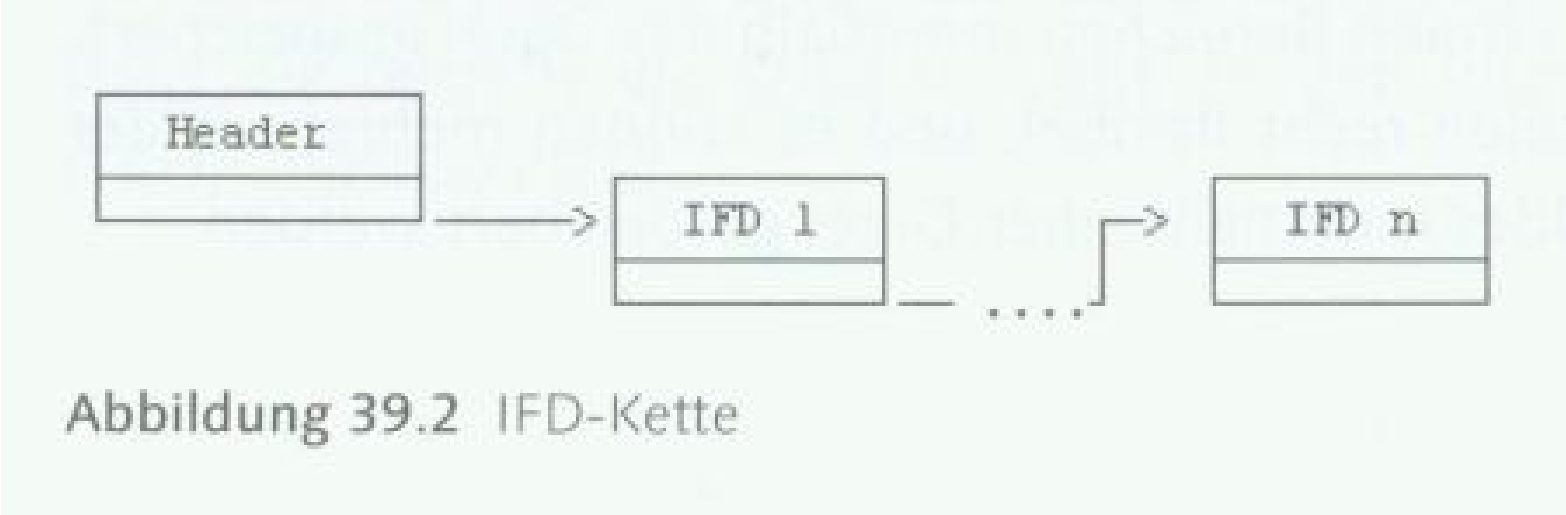


Abbildung 39.2 IFD-Kette

2.1.4 Struktur der Tags

- Das Tag enthält Informationen über Bilddaten wie z.B. Bildabmessungen, Auflösung etc.
- Passen nicht alle Angaben in das Tag, werden die Reste in freie Bereiche innerhalb der Datei ausgelagert. Ein Zeiger weist auf diesen Datenbereich.

Offset	Bytes	Bemerkungen
00H	2	Tag-Typ
02H	2	Datentyp (1 = Byte, 2 = ASCII-String (mit 0-Byte abzuschliessen), 3 = Short, 4 = Long, etc.)
04H	4	Länge des Datenbereichs
08H	4	Zeiger auf Datenbereiche oder Werte

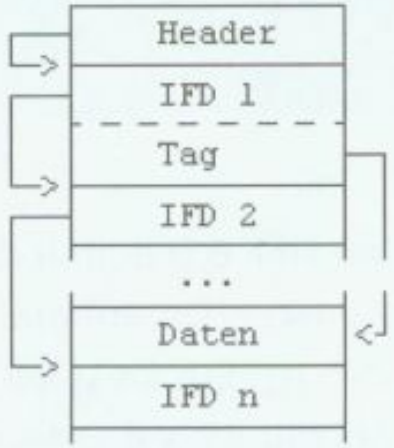


Abbildung 39.3 Zeiger von Tags in den Datenbereich

2.2 GIF (Graphics Interchange Format)

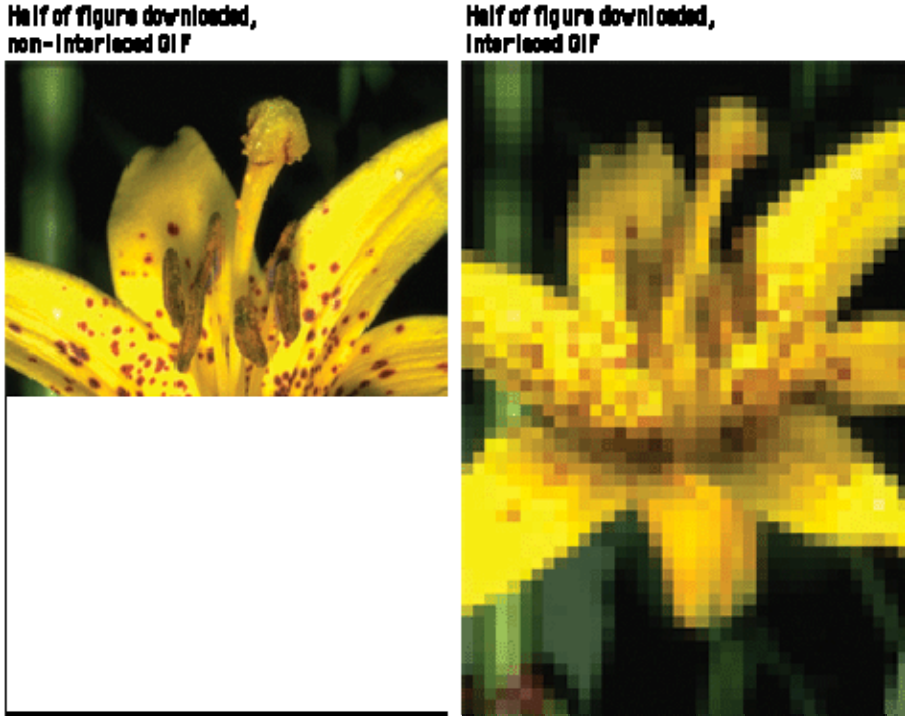
2.2.1 Charakteristik

- GIF ist ein Rastergrafik-Format, das eine Farbtiefe von bis zu 8 Bit haben kann (also max. 256 Farben).
- Das GIF-Format liegt in zwei Spezifikationen vor:
  - GIF-87a ist die 1987 entwickelte Standardversion.
  - GIF-89a erlaubt zusätzlich zu GIF-87a *Transparenz*.
- Auch GIF-Dateien werden mit dem LZW-Verfahren komprimiert.
- GIF-Dateien können im *Interlacing*-Verfahren gespeichert werden.
- GIF ist potentiell ein „Auslaufformat“:

  Das GIF-Format wurde von der Firma Compuserve entwickelt und frei zugänglich gemacht, das LZW-Verfahren wurde allerdings von Unisys patentiert.

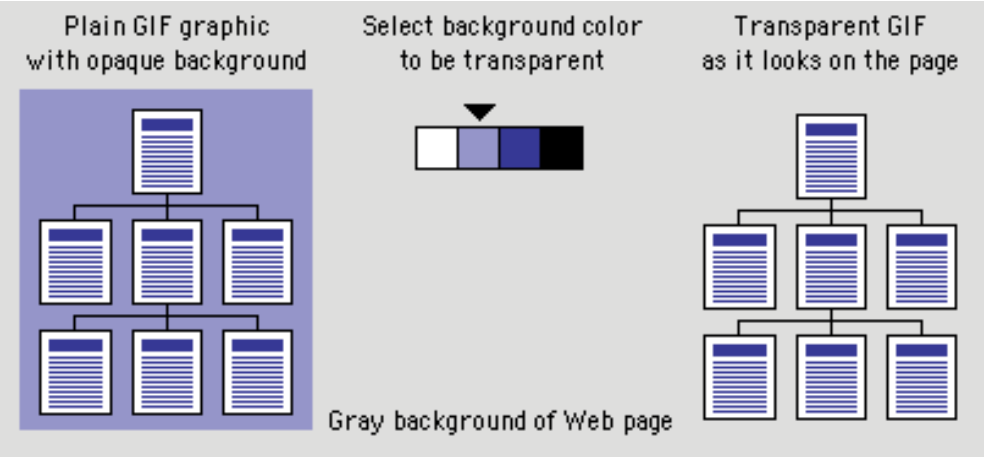
2.2.2 Interlacing-Verfahren

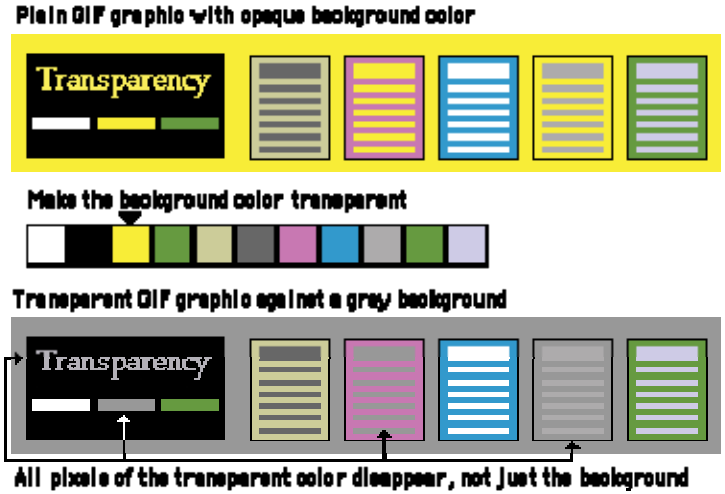
- die Formate GIF, JPEG und PNG kennen das Interlacing-Verfahren als Alternative zum konventionellen zeilenweisen Aufbau der Grafik (non-interlaced) durch einen Browser.
- Das Bild wird dabei unmittelbar in voller Größe, allerdings nur zunächst schemenhaft, in geringer Auflösung, aufgebaut. Fortlaufend werden immer mehr Daten nachgeladen (*interlace* = *„einflechten“*), wodurch die Grafik sich sukzessive der Originalqualität nähert.
- Technik: In mehreren Schritten werden die Zeilen einer Grafik übermittelt (zunächst jede achte, …); die fehlenden Zeile werden jeweils kopiert.
  →höherer Speicherbedarf



2.2.3 Transparente GIFs

- Mit GIF können Daten auch transparent übermittelt werden; D.h.: Bestimmte Farben der Grafik können ausgespart werden.
- Technik: diverse Anwendungsprogramme (z.B. Photoshop) erlauben die Selektion von Farbwerten aus der GIF-Farbtabelle der Grafik; diese werden dann nicht angezeigt.





2.3 PNG (Portable Network Graphics)

2.3.1 Charakteristik

- Das PNG-Format ist entwickelt worden, da es rund um das GIF-Format immer wieder Copyright-Probleme gegeben hat.

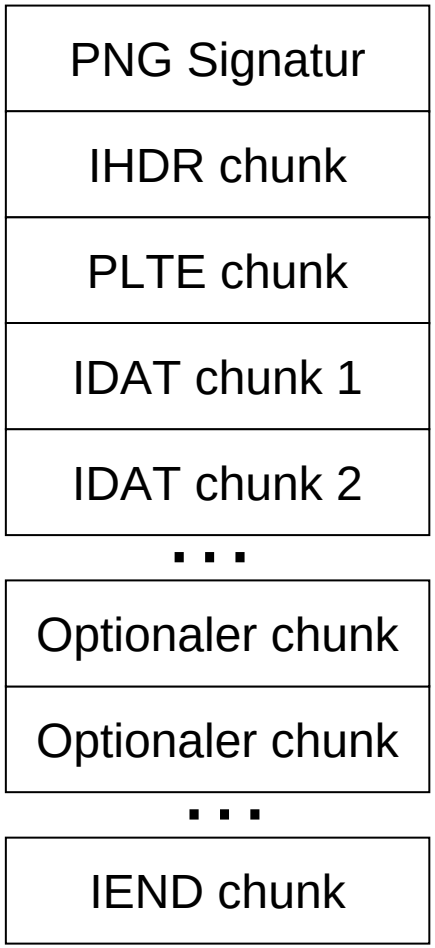
- PNG ist ein Rastergrafik-Format, das eine Farbtiefe von 48 Bit bei RGB-Bildern und 16 Bit bei Graustufenbildern haben kann.

- PNG-Dateien lassen sich ebenfalls interlaced (*Adam 7*, nach Adam M. Costello) speichern: Das Verfahren teilt die Daten in 8x8 Pixel große Blöcke; in 7 Durchläufen werden daraus nach einem Schema bestimmte Bildpunkte übertragen; Grundprinzip: Abwechselnde Verdopplung der horizontalen und vertikalen Auflösung.

- PNG unterstützt u.a. Huffman-Kodierung.

2.3.2 Aufbau einer PNG-File

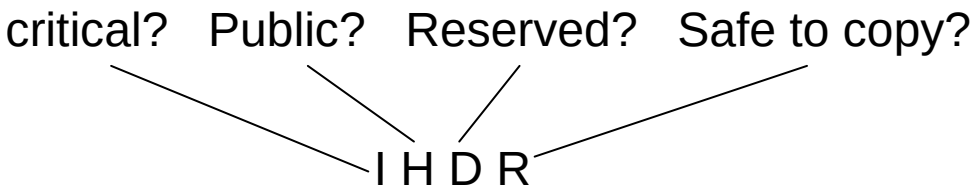
- Eine PNG-Datei hat einen blockweisen Aufbau, der aus sogenannten *chunks (Blöcke)* besteht.
- Jeder chunk enthält Informationen über einen bestimmten Aspekt der Grafik, also z.B. Farbpalette, zusätzliche textuelle Informationen, Zeitangaben,…



- Chunk werden über verschiedene Quellen definiert:
  - PNG Standard
  - Liste der registrierten chunks (PNG Development Group)
  - Anwendungsprogramme

2.3.3 Chunk-Nomenklatur

- Jeder chunk-Name besteht aus 4 ASCII-kodierten Buchstaben, wobei der erste, zweite und letzte in Großschreibung vorliegen kann (der dritte muß).
- Der Decoder kann anhand der Buchstaben Informationen über den chunk ermitteln.



P L T E = critical, public, unsafe to copy

t E X t = non-critical, public, safe to copy

A 1 PX = ungültig

Critical: der Dekoder muß den chunk verarbeiten

Public: alle chunks des PNG Standards und alle registrierten chunks

Reserved: reservierte Bits (z.Z. immer große Buchstaben (=ja))

Safe to copy: garantiert das Kopieren des chunks fehlerfreie Prozessierbarkeit? (groß=nein)

2.3.4 Struktur von chunks

Feld	Größe	Beschreibung
Länge	4 Byte	Länge des Datenfelds
Typ	4 Byte	Chunk-Name
Daten	n Byte	Datenbereich
CRC	4 Byte	Cyclic redundancy check (Kontrollwert)

2.3.5 Essentielle Chunks: Header-chunk (IHDR)

- Der Header-chunk muss neben dem/den Daten-chunk(s) (IDAT) und dem End-Chunk (IEND) in jeder PNG-Datei vorhanden sein;

  wird zur Farbdarstellung eine Farbpalette verwendet, so muss der PLTE-chunk ebenfalls lesbar vorhanden sein.
- Der Header-chunk enthält Informationen über die Daten, die in der PNG-Datei gespeichert werden.
- Der Header-chunk muss sofort nach den 8 Byte der Signatur auftreten.

Header-Chunk (IHDR)

Offset	Bytes	Bemerkungen
00H	4	Breite des Bildes in Pixel
04H	4	Höhe des Bildes in Pixel
08H	1	Bit pro Pixel (oder pro Sample) Hier: 8 Bit Zulässig für Graustufen 1,2,4,8,16, für Farbpalette 1,2,4,8, für TrueColor 8,16 Bit.
09H	1	Color Typ Hier: 6 = Farbbild mit Alphakanal Bitweise Kodierung: Bit 0 = Palette, Bit 1 = Farbbild, Bit 2 = Alphakanal benutzt.
0AH	1	Compression-Typ
0BH	1	Filter Typ
0CH	1	Interlace-Flag

2.3.6 Optionale chunks: Textual Data-chunk (tEXt)

- Der Textual Data-chunk erlaubt es, lesbare Texte in einer PNG-Datei mit abzulegen.
- Zur Zeit sind folgende Begriffe definiert: Title, Author, Description, Copyright, Creation Time, Software, Disclaimer, Warning, Source, Comment.
- Als Signatur wird der Name tEXt benutzt.

Offset	Bytes	Bemerkungen
00H	n	String mit dem Schlüsselwort
...H	1	00H als Seperator
...H	n	Text

2.4 JPEG (Joint Photographics Expert Group)

2.4.1 Charakteristik

- JPEG ist eher eine Kompressionsmethode denn ein richtiges Bildformat.

- JPEG ist ein **verlustbehafteter** Algorithmus.

- Um Kompressionsraten von bis zu 90% bei akzeptabler Bildqualität zu erreichen, werden verschiedene Methoden kombiniert eingesetzt, darunter auch Huffman, RLE oder DCT (**lossy**).

- JPEG-Dateien beschreiben Bilder als Rastergrafik, die eine Farbtiefe von 24 Bit haben kann.

- Dem *Interlacing*-Verfahren beim GIF entspricht das progressive JPEG.

- JPEG ist in ISO DIS 10918-1 definiert.

- Die Definition von JPEG erlaubt allerdings viele Freiheiten, so dass der Austausch von JPEG-Bilddaten zwischen verschiedenen Anwendungen und Plattformen relativ problematisch ist.

- Als minimaler Standard für den Austausch wurde das JFIF (JPEG File Interchange-Format) -Format definiert.

2.4.2 JPEG-Farbmodell

- JPEG-Grafiken werden im *YCbCr*-Farbmodell gespeichert.

- **Y** ist ein Luminaz-Wert und gibt die Helligkeit eines Punktes an, **Cb** (Blau-Information) und **Cr** (Rot-Information) sind Crominanz-Werte, welche die Farben charakterisieren.

- RGB- und YCbCr-Farbmodelle lassen sich linear ineinander überführen: die Berechnung der YCbCr-Farbe geschieht gemäß der folgenden Werte:

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- Dieses Farbmodell empfiehlt sich aufgrund der Tatsache, dass der Mensch Helligkeitsunterschiede stärker wahrnimmt als Farbunterschiede. (*siehe Bsp. nächste Seite*)

Y-Kanal

Cb-Kanal

Cr-Kanal

2.4.4 Downsampling

- Mit *Downsampling* bezeichnet den Vorgang, aus den originalen Pixeln neue Pixel zu berechnen, die eine geringere Auflösung haben. Hierbei geht Bildinformation verloren.

- Graustufenbilder werden nicht auf diese Weise reduziert.

Aufgaben

1) **Wiederholen** Sie den Stoff dieser Sitzung **bis zur nächsten Sitzung** (siehe dazu den Link zur Sitzung auf der HKI-Homepage).
   Informieren Sie sich zusätzlich durch eigene Literaturrecherche!

2) Beantworten Sie die Fragen aus der Sammlung **„beispielhafte Klausurfragen“** zum Bereich Bild (soweit in dieser Sitzung behandelt).