Komprimierung von Bildern

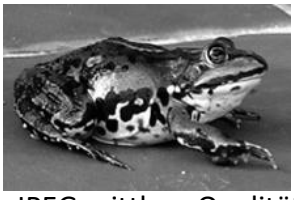
verlustfreie Komprimierung	Komprimierung mit Informations-Verlusten
Der Originalzustand kann wieder hergestellt werden. Arbeitet mit Mustererkennung. Gleichfarbige nebeneinander liegende Pixel werden zusammengefasst. RLE/Huffmann Verfahren (gif, png)	Der Originalzustand kann nicht wieder hergestellt werden. Vergleicht blockweise Pixel und fasst ähnliche Pixel zusammen. Die Ähnlichkeit ist von der eingestellten Qualitätsstufe abhängig. JPEG-Verfahren (jpg)

Kompressionsrate= 100% - komprimiertes Bild/Originalbild · 100%

Gib die Kompressionsraten in % , d. h. wie viel % des Speicherplatzes eingespart wurde, an.

Originalbild TIFF - 97 kB

GIF (64 Graustufen) - 19 kB



JPEG mittlere Qualität - 8 kB

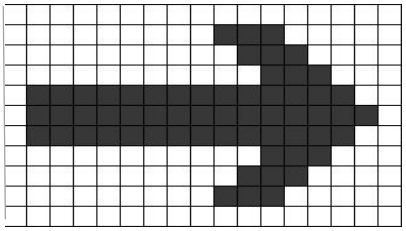
JPEG niedrigste Qualität - 2 kB

Codierung mit Blöcken fester Länge

1. Aufgabe: Gegeben ist ein Schwarz-Weiß-Bild mit der Auflösung 1024 x 768 Pixel.

a) Gegeben ist ein Schwarz-Weiß-Bild mit der Auflösung 1024 x 768 Pixel.
Berechne den Speicherbedarf des Bildes. (B und kB)

b) Gib den Speicherbedarf des unkomprimierten Pfeilbildes (s-w) an. (in bit)



Ist es möglich, die Speichergröße zu verkleinern ohne Bildinformationen zu verlieren?
Ja, z.B. mit Lauflängencodierung (RLE). Man kann schreiben:

26 x weiß, 3 x schwarz, 15 x weiß, 3 x schwarz, 15 x weiß, 3 x schwarz, 4 x weiß, 14 x schwarz, 3 x weiß, 15 x schwarz, 2 x weiß, 14 x schwarz, 13 x weiß, 3 x schwarz, 13 x weiß, 3 x schwarz, 13 x weiß, 3 x schwarz, 22 x weiß.

c) Berechne jetzt den Speicherbedarf (bit) des mit RLE komprimierten Pfeilbildes.

2. Aufgabe:

Skizziere ein kleines Bild, bei dem die RLE-Codierung eine besonders wirksame Komprimierung ermöglicht. Ermittle die Kompressionsrate in Prozent.

3. Aufgabe:

Skizziere ein kleines Bild, bei dem die RLE-Codierung eine besonders schlechte Komprimierung ermöglicht. Ermittle die Kompressionsrate in Prozent.

Es geht noch viel besser als mit RLE, nämlich mit der bitweisen Lauflängen-Kompression

I. Wir bilden Codes aus 4 Bit. (--> Tabellen)
II. Das erste Bit legt die Farbe fest (0 für Weiß, 1 für Schwarz)
III. Die folgenden 3 Bit geben an, wie oft diese Farbe vorkommt (000 für 1 x, 001 für 2 x, 010 für 3 x usw.)

0000	**1 weißes Pixel**
0001	**2 weiße Pixel**
0010	**3 weiße Pixel**
0011	**4 weiße Pixel**
0100	**5 weiße Pixel**
0101	**6 weiße Pixel**
0110	**7 weiße Pixel**
0111	**8 weiße Pixel**

1000	**1 schwarzes Pixel**
1001	**2 schwarze Pixel**
1010	**3 schwarze Pixel**
1011	**4 schwarze Pixel**
1100	**5 schwarze Pixel**
1101	**6 schwarze Pixel**
1110	**7 schwarze Pixel**
1111	**8 schwarze Pixel**

4. Aufgabe:

Codiere den Pfeil noch einmal mit dieser Code-Tabelle. Wie viele Bit benötigst Du dafür? Ermittle die Kompressionsrate in Prozent.

5. Aufgabe:

Zeichne das Bild mit 8 Pixel Breite und 11 Pixel Höhe, für welches der 4 Bit-Code gegeben ist.
0111 0111 0001 1011 0110 1000 0110 1000 0101 1001 0100 1000 0101 1000 0110 1011 0111 0111 0001 *Ermittle die Kompressionsrate in Prozent.*

6. Aufgabe:

Die bisher verwendete Form der Lauflängencodierung bestand aus Codeblöcken von je vier Bit. Ausreichend ist eine Codetabelle für die Blocklänge von drei Bit.
Codiere den "Pfeil" mit dieser Code-Tabelle. Wie viele Bit benötigst Du dafür? Ermittle die Kompressionsrate in Prozent.

Code weiß	Bedeutung
100	**1 weiß**
101	**2 weiß**
110	**4 weiß**
111	**8 weiß**

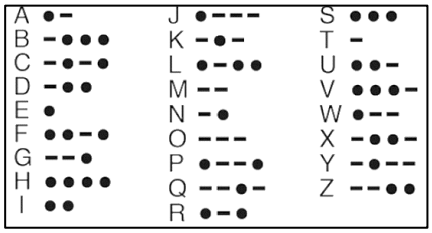
Code schwarz	Bedeutung
000	**1 schwarz**
001	**2 schwarz**
010	**4 schwarz**
011	**8 schwarz**

7. Aufgabe:

Wie viele Bit sind eigendtlich für die Codierung der Farben des S-W- Pfeils ausreichend?
Berechne den dann benötigten Speicherplatz und den Kompressionsfaktor.

Codierung mit Blöcken variabler Länge

Beim Morsen z.B. sind nicht alle Codes gleich lang (Codes veränderlicher Länge) und man benötigt ein drittes Symbol, um die Codewörter voneinander zu trennen.



Beispiel:
Was kann diese Codesequenz« • • – • • • • – » bedeuten?
Vielleicht IDEA? • • – • • • • – •
oder doch USA? • • – • • • • – •
Und geht nicht auch FV? • • – • • • • –

Also keinesfalls eindeutig! Der Nachteil beim Morsecode liegt darin, dass es ohne spezielles Trennzeichen Missverständnisse geben kann, wo das Zeichen beginnt und wo es endet.
Die Funker haben dies mit einer etwas längeren Pause gekennzeichnet. So**: 0000-00-11-01-0100-01-1011-01**
Dieses Problem besteht z.B. bei der nachfolgenden Huffman-Kodierung nicht, weil hier die Eigenschaft erfüllt sein muss, dass kein Codewort der Beginn eines anderen Codewortes sein darf.

Die Huffman-Codierung
Die amerikanischen Mathematiker David A. Huffman und Claude Shannon solche Codes entwickelt. Die Idee ist nicht schwer zu verstehen:
Man muss der Codierung ansehen können, wo Codewörter enden. Also darf kein Codewort so aussehen wie das "Anfangsstück" eines anderen Codewortes. Solche Codes werden "präfixfrei" genannt.

Beispiel (mit vier Codewörtern)

Buchstabe	Code
E	1
T	01
R	000
0	001

8. Aufgabe:
Dekodiere mit der linken Tabelle:
a) 001000011
b) 00000101
c) 01001000011
d) 000101011000

Damit können wir jetzt auch Bilddateien codieren. Dabei müssen wir die Bilddatei aber zunächst untersuchen. Je häufiger eine Kombination vorkommt, desto kürzer muss der der sein.

Verwenden wir wieder unser Pfeilbild. Wir betrachten Paare von zwei aufeinander folgenden Pixeln und erstellen eine Häufigkeitstabelle.

"Pixelpaar"	Häufigkeit	Code
weiß-weiß	58	
weiß-schwarz	2	
schwarz-weiß	9	
schwarz-schwarz	25	

8. Aufgabe: *Erstelle den minimalen Binär- Code. Berechne den komprimierten Speicherplatz. Gib die Kompressionsrate an.*

Kompression mit Informationsverlust

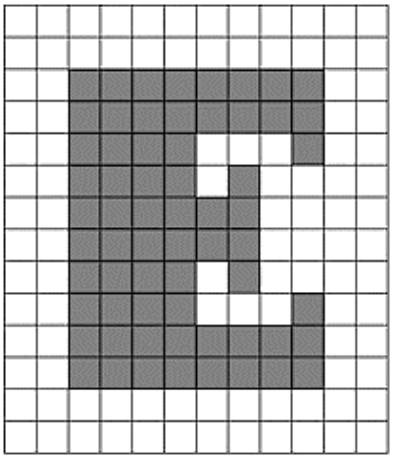
Verfahren, die Bilder verlustbehaftet komprimieren, verwenden im Prinzip zwei Schritte:

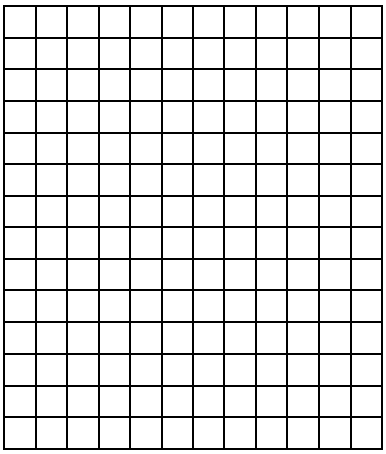
I. Vereinfachung der Bildinformationen
II. Kodierung des vereinfachten Bildes.

9. Aufgabe:
Für ein Beispiel benutzen wir eine einfache Regel zum "Vereinfachen":

- weiße Pixel bleiben unverändert
- schwarze Pixel bleiben schwarz, wenn ihr linker **und** rechter Nachbar auch schwarz war, sonst werden sie weiß.

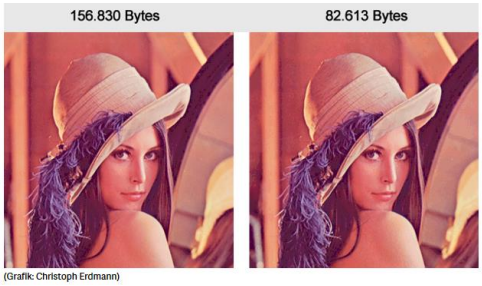
Bearbeite das folgende Bild nach dieser Regel. Wende danach die 4-Bit-Lauflängenkodierung an. Ermittle den Komprimierungsfaktor.





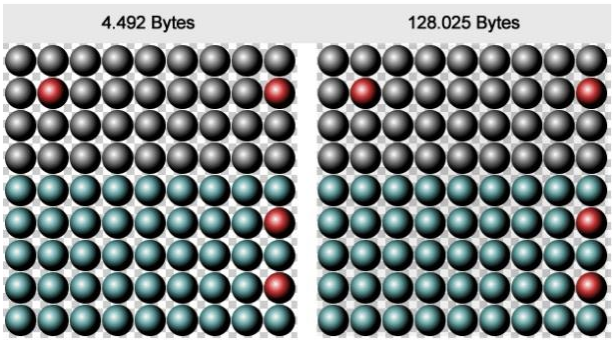
Das PNG-Format

(Quelle: https://t3n.de/news/png-format-guide-1117239 Christoph Erdmann)



(Grafik: Christoph Erdmann)

Hochaufgelöste Bilder mit wenig Speicherplatz – wie gelingt das?



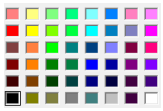
Schaut euch diese beiden Bälle-Bilder an:
Sie sehen gleich aus, haben die gleiche Breite und Höhe, beide sind PNG. Dennoch unterscheidet sich die Dateigröße deutlich. Warum?

Farbtiefe
Im Normalfall werden Bilder im RGB-Farbmodell vorliegen.
Ein Pixel besteht aus den Farbkanälen Rot, Grün und Blau – mit jeweils 256 Abstufungen– jeweils ein Byte für Rot, Grün und Blau. 24 Bit ergeben sich, weil ein Byte aus acht Bits besteht.

32-Bit und 8-Bit-Farbtiefen

Ein **32-Bit-PNG** gleicht einem 24-Bit-PNG, enthält aber noch einen zusätzlichen Acht-Bit-Kanal für Transparenz (Alphakanal genannt). Es kann also transparente Bereiche in 256 Abstufungen beinhalten.

Ein **8-Bit-PNG** funktioniert ein wenig anders:



Hier werden 256 Farben mit ihren Rot-, Grün-, Blau- und Transparenz-Informationen in einer sogenannten „Palette“ festgelegt. Für jeden Pixel wird definiert, an welcher Stelle der Palette die entsprechende Farbe steht. Somit braucht ihr zum Speichern eines Pixels nur noch ein Byte statt drei.

Die Farbtiefe ist übrigens ein Grund, warum sich die beiden Bälle-Beispielbilder in der Dateigröße unterscheiden. Das erste PNG ist ein Acht-Bit-PNG, während das größere PNG ein 32-Bit-PNG ist.

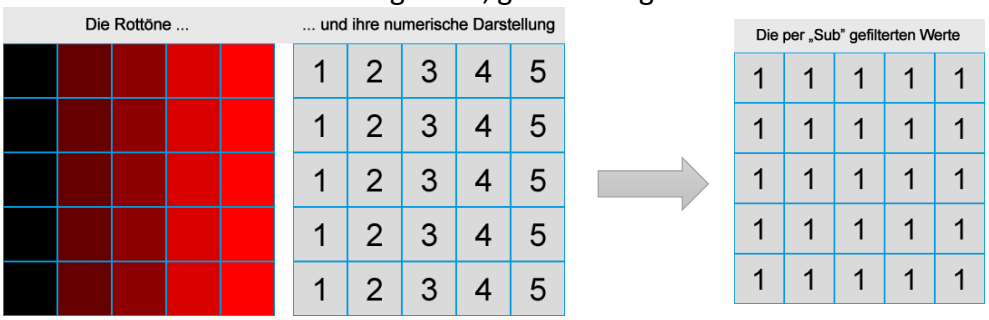
8-Bit-PNG	**24-Bit-PNG**	**32-Bit-PNG**
Paletten-Bild	RGB-Bild	RGBA-Bild
maximal 256 Farben	16,8 Millionen Farben	16,8 Millionen Farben + Transparenz

Die Komprimierung eines PNG-Bildes

Ein PNG-Bild wird in drei Schritten komprimiert:
Vorfiltern, wörterbuch-basierte Codierung per LZ77 und schließlich Häufigkeits-Kodierung nach Huffman.

Schritt 1: Das Vorfiltern

Daten können effizienter komprimiert werden, wenn sie sich möglichst oft wiederholen. Das Vorfiltern versucht, aus unterschiedlichen Daten möglichst gleichartige zu generieren. Schauen wir uns mal die Rotwerte eines 5x5 Pixel großen, gleichmäßigen Farbverlaufs an:



(Grafik: Christoph Erdmann)

Auf jede Zeile wird ein Filter angewendet. In diesem Fall der Subtraktions-Filter.
Wir merken uns einfach die Differenz zum vorangegangenen Wert in der aktuellen Zeile. Dieser Schritt ist umkehrbar, sodass keine Daten verlorengehen. Die Werte sind sehr viel besser zu komprimieren.

Schritt 2: Wörterbuch-basierte Kodierung per LZ77

LZ77 ist ein verlustfreies Verfahren, das sich wiederholende Sequenzen von Daten sucht. Trifft der Algorithmus auf eine Sequenz von Daten, die er schon einmal weiter vorn in der Datei gesehen hat, ersetzt er diese mit einem Hinweis auf die erste Sequenz: "Hier bitte die Sequenz an Daten einsetzen, die 1300 Zeichen vorher auftaucht und 200 Zeichen lang ist".
Kurz: Habt ihr gleiche Abbildungen von Objekten in einer PNG-Datei, belegt nur die erste Darstellung Platz. Die anderen werden euch praktisch geschenkt.

Schritt 3: Entropiekodierung

Dies ist der letzte Schritt der PNG-Kompression. Hier wird versucht, jedes Zeichen mit einer möglichst kleinen Bitfolge darzustellen. (siehe Aufgabe 8)