Algorithmen und Datenstrukturen

Teil 15: Kompression

DHBW Stuttgart Campus Horb
Fakultät Technik
Studiengang Informatik
Dozent: Olaf Herden
Stand: 06/2020

Gliederung

- Einleitung
- Arithmetische Codierung
- Lauflängencodierung
- Differenzcodierung
- LZW-Algorithmus

Grundbegriffe (I)

- Blockcode vs. Codes variabler Länge
- Blockcode := Alle Codes haben feste Wortlänge
- Haben in der Regel hohe Redundanz

- Messung der Kompression:
  - Kompressionsfaktor KF:

$$KF = \frac{Platz\ unkomprimiert}{Platz\ komprimiert}$$

  - Eingesparter Speicherplatz:

$$ES = 1 - \frac{1}{KF}$$

- Beispiel:
  - Daten unkomprimiert: 1000
  - Daten komprimiert: 200
  - Dann: $KF = \frac{1000}{200} = 5\ und\ ES = 1 - \frac{1}{5} = 0{,}8 = 80\%$

Grundbegriffe (II)

- Statistisches Kompressionsverfahren:
  - Arbeitet auf Basis der Häufigkeiten der Daten in zu komprimierendem Objekt

- Verlustfreie vs. verlustbehaftete Kompression:
  - Verlustfrei: Informationsgehalt der Daten bleibt vollständig erhalten
  - Beispiele: Texte, Programmdateien, PNG
  - Verlustbehaftet: Informationen bleiben „im Wesentlichen“ erhalten, „gewisser“ Informationsverlust wird in Kauf genommen
  - Bsp.: JPEG, MP3
  - Vorteile:
    - Erhebliche höhere Kompressionsfaktoren
    - Anwender/in kann Kompressionsfaktor steuern (zu Lasten der Qualität)

Übersicht

- Einleitung
- Arithmetische Codierung
- Lauflängencodierung
- Differenzcodierung
- LZW-Algorithmus

Arithmetische Codierung

- Grundidee:
  - Wie beim Huffman-Verfahren:
    - Verlustfrei
    - Ausnutzung unterschiedlicher Häufigkeitsverteilungen in Ausgangsdatei
    - Codierung von Einzelzeichen, d.h. keine Berücksichtigung von Korrelationen benachbarter Zeichen
  - Jedem Ausgangstext wird Gleitkommazahl $0 \leq x < 1$ zugeordnet
- Ablauf:
  - Vor eigentlicher Codierung Ermittlung der Häufigkeiten einzelner Zeichen
  - Aufteilung Intervall $[0; 1[$ in n Intervalle, wobei jedes Intervall einem Zeichen entspricht und Intervallbreite Auftretenshäufigkeit entspricht
  - Tabelle ist für Codierung wie Decodierung erforderlich

Beispiel (I)

- Codierung Wort `ESSEN`
- Tabelle mit Häufigkeiten und Intervallen

Zeichen c_i	Auftrittswahrscheinlichkeit p_i	Intervall [u;o[
E	2/5	[0,0;0,4[
S	2/5	[0,4;0,8[
N	1/5	[0,8;1,0[

Kompressionsvorgang

```
( 1) u = 0
( 2) o = 1
( 3) REPEAT
( 4)   Lese nächstes Eingabezeichen c_i
( 5)   d = o - u          // Aktuelle Intervalllänge
( 6)   o = u + d*o(c_i)   // Neue Obergrenze
( 7)   u = u + d*u(c_i)   // Neue Untergrenze
( 8) UNTIL Textende ist erreicht
( 9) Gebe u als Ergebnis aus
```

- Erläuterungen:
  - Intervallgrenzen u und o werden pro Schritt vergrößert bzw. verkleinert (abhängig von Intervall des Zeichens)
  - Weil stets $d < 1 \Rightarrow$ u und o können nie Grenzen des durch erstes Zeichen gegebenen Intervalls unter- bzw. überschreiten
  - Durch nächstes Zeichen hinzukommende Zuwachs kann nie größer sein als Intervalllänge dieses Zeichens
  - Dadurch ist eindeutige Umkehrbarkeit garantiert
  - Ergebnis abspeichern als Hexadezimalzahl statt Dezimalzahl

Beispiel (II)

Zeichen c_i	Intervalllänge d	Untergrenze u	Obergrenze o
	---	0,0	1,0
	0 … 1		
E	1,0	0,0	0,4
	0 … 0,4		
S	0,4	0,16	0,32
	0,16 … 0,32		
S	0,16	0,224	0,288
	0,224 … 0,288		
E	0,064	0,224	0,2496
	0,224 … 0,2496		
N	0,0256	0,24448	0,2496
	0,24448 … 0,2496		

- Ergebnis x = 0,24448
- Nachkommastellen speichern

Dekompression

- Kompressionsalgorithmus umkehren:
  - D.h. aktuelles Intervall auf [0;1[strecken
  - Zu decodierendes Zeichen kann aus Tabelle abgelesen werden
  - Ende der Dekompression muss explizit geregelt werden:
    - Abspeichern Anzahl der Zeichen oder
    - Einführen spezielles Endezeichen

```
( 1) WHILE (Nicht alle Zeichen decodiert)
( 2)   Lese Code x
( 3)   Suche Zeichen c_i, in dessen Intervall x liegt
( 4)   Gebe Zeichen c_i aus
( 5)   d = o(c_i) - u(c_i)        // Neue Intervalllänge
( 6)   x = (x-u(c_i)) / d         // Neuer Code
( 7) END WHILE
```

Beispiel

- Rückcodierung von 0,24448:

Code x	Intervall-länge d	Unter-grenze u	Ober-grenze o	Ausgabe-zeichen c_i
0,24448	0,4	0,0	0,4	E
0,6112	0,4	0,4	0,8	S
0,528	0,4	0,4	0,8	S
0,32	0,4	0,0	0,4	E
0,8	0,2	0,8	1,0	N
0,0	---	---	---	Ende

- Zur Erinnerung:

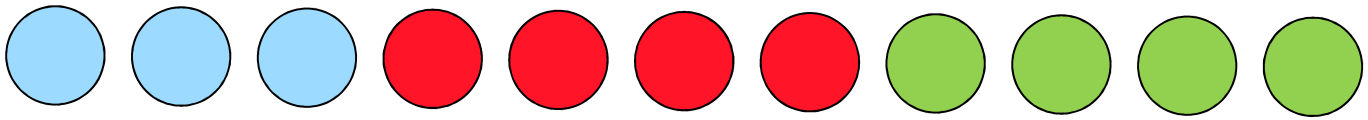
Zeichen c_i	Auftrittswahrscheinlichkeit p_i	Intervall [u;o[
E	2/5	[0,0;0,4[
S	2/5	[0,4;0,8[
N	1/5	[0,8;1,0[

Übersicht

- Einleitung
- Arithmetische Codierung
- Lauflängencodierung
- Differenzcodierung
- LZW-Algorithmus

Lauflängen

- Idee: Stelle Folge gleicher Zeichen (Run) durch Angabe des Zeichens und Anzahl an



- Lauflängencodierung = RLE (Run Length Encoding)
- Anwendungsgebiete:
  - Allgemein: Datensequenzen mit wenigen verschiedenen Werten
  - Speziell: Bilder mit wenigen verschiedenen Farben
- Erreichbare Kompressionsrate abhängig von:
  - Anzahl unterschiedlicher Ausgangswerte
  - Länge der Runs

Vorgehen

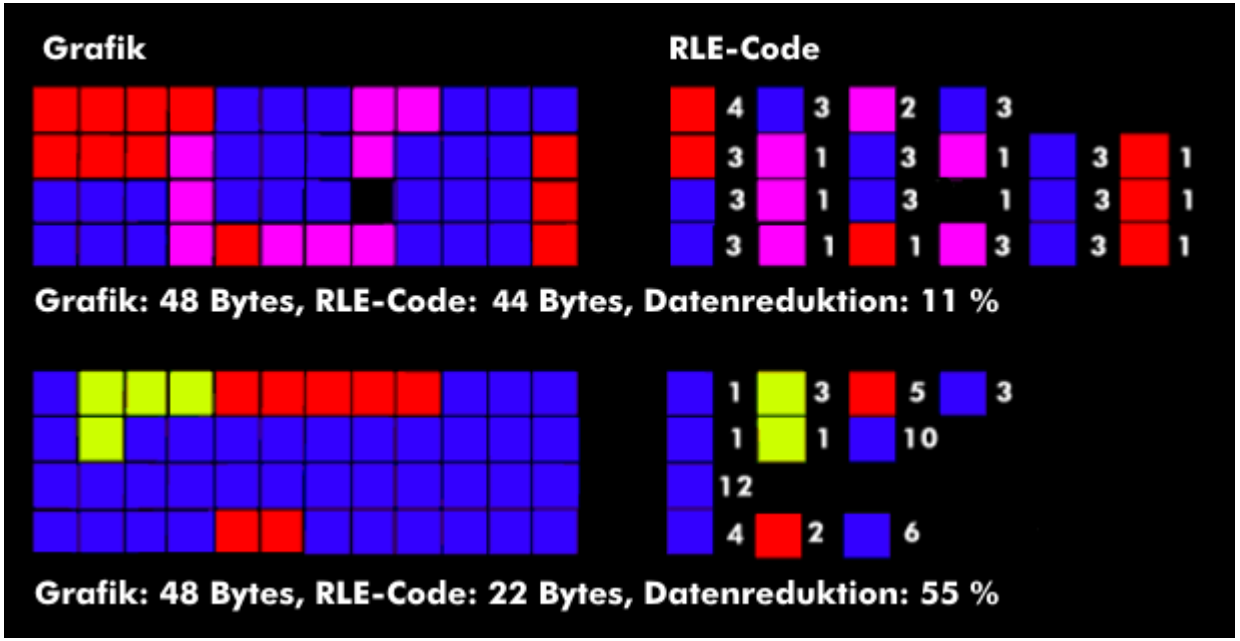
- Sonderzeichen festlegen, z.B. #
- Codierung:
  - Run Länge n von x als x#<n> darstellen
  - n ∈ [4, 259] mit einem Byte darstellbar
  - # als ## darstellen
- Beispiele:
  - Eingabe: XXXXXYYZZZZZZZZZU
  - RLE: X#5YYZ#9U

  - Eingabe: X#Y##
  - RLE: X##Y####

Beispiel binäre Werte

- Originalwerte: 0000000000000011111110000000111111111111
- Idee zur Speicherung:
  - Ermittle längsten Run
  - Wähle Wortbreite, so dass längster Run repräsentierbar
  - Speichere nacheinander alternierende Anzahl 0en und 1en ab
- Im Beispiel:
  - Längster Run = 15
  - D.h. 4 Bit-Wort zum Speichern einer Run-Länge
  - Also: 1111 0111 0111 1011
    15 7 7 11
- Kompressionsfaktor:
  - Unkomprimierte Speicherung: 40 Bit
  - Komprimierte Speicherung: 16 Bit
  - Kompressionsfaktor: 2,5

Beispiel Grafiken



[http://www.itwissen.info]

- Anwendung: TIFF-, Bitmap-, TGA-Format

Quadtrees (I)

- Erweiterung der Lauflängencodierung auf zwei Dimensionen
- Idee:
  - Baumstruktur von rechteckigen Bereichen abnehmender Größe mit jeweils einheitlichem Wert
- Aufbau Quadtree:
  - Wurzel als unterste Ebene entspricht Gesamtbereich (bzw. ausgewähltem rechteckigen Bereich)
  - Wurzel hat vier Nachfolger, jeder repräsentiert ein Viertel der Wurzel
  - Dieses wird weitergeführt, bis alle Werte eines Knoten gleichen Wert haben
- Zwei Extremfälle:
  - Alle Daten in Wurzel haben gleichen Wert $\Rightarrow$ Baum besteht nur aus Wurzel
  - Verfahren muss weitergeführt werden, bis Knoten nur noch Einzelwerte besitzen

Quadtrees (II)

- Speicherung Quadtree:
  - Markiere Knoten mit 1, falls er weiter zerlegt werden muss
  - Markiere Knoten mit 0 sonst
  - Durch schichtweise Bitfolge ist Baum eindeutig identifiziert
  - Zusätzlich müssen Wert der Blattknoten gemerkt werden

Beispiel

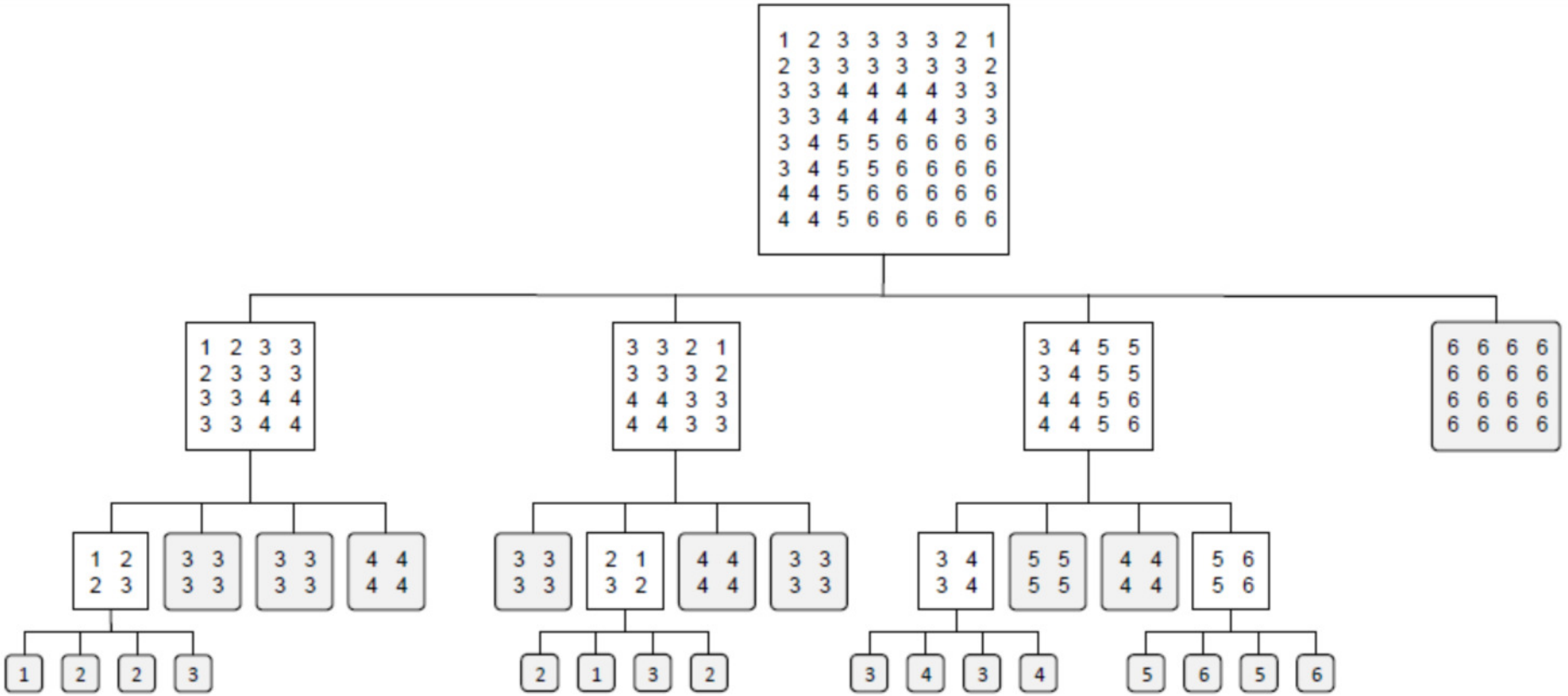


Abb. 3.14 Beispiel für einen Quadtree für ein einfaches Bild. Gespeichert werden die Knotenadressen als Folge von 1en (innere Knoten) und 0en (Blätter mit gerundeten Ecken) und danach die den Blättern zugeordneten Grauwerte. Hier ergeben sich 17 Bit für die Baumstruktur gefolgt von 25 Grauwerten: 1 1110 1000 0100 1001 63343435412232132343456 56. Bei einer Codierung der Grauwerte mit 8 Bit ergibt sich eine Kompression von 512 Bit auf 217 Bit

Übersicht

- Einleitung
- Arithmetische Codierung
- Lauflängencodierung
- Differenzcodierung
- LZW-Algorithmus

Idee

- Speichere nicht einzelne Werte, sondern Differenz aufeinanderfolgender Werte
- Anwendungsgebiet: Messwerte
- Aufeinanderfolgende Werte hier meist nah beieinander
- Lässt gute Kompressionsraten erwarten

Verlustfreie Differenz-Codierung

- Erster Wert sei f_0
- Speichere diesen ab
- Speichere danach jeweils Differenzen $d_i = f_i - f_{i-1}$
- Typische Vorgehensweise:
  - Führe dies nur bis bestimmter Differenz durch
  - Nehme dann tatsächlichen Wert als neuen Referenzpunkt

  - Verwendung variabler Codelängen
  - Dadurch bekommen häufige Differenzen kurze Codes

  - Spezielles Codewort für Fall, dass Differenz größer als Maximalwert, d.h. tatsächlicher Wert wird gespeichert

Beispiel

- D_1: Differenzcode variabler Wortlänge
- D_2: Differenzcode konstante Wortlänge (4 Bit, Werte 0 bis 7)

Differenz d	Code D_1	Code D_2
0	1	0000
1	0100	0001
-1	0101	1001
2	0110	0010
-2	0111	1010
3	00100	0011
-3	00101	1011
4	00110	0100
-4	00111	1100
5	000100	0101
-5	000101	1101

Differenz d	Code D_1	Code D_2
6	000110	0110
-6	000111	1110
7	0000100	0111
-7	0000101	1111
8	0000110	|d|>7: 1000 und danach 8 Bit für den Datenwert
-8	0000111	
|d|>8	00000 und danach Daten-wert	

Ungünstiges Szenario

- Treten häufig den Maximalwert überschreitende Differenzen auf $\Rightarrow$ Keine/schlechte Kompressionsrate

Verlustbehaftete Differenzcodierung

- Erweiterung des verlustfreien Verfahrens
- Kann größere Differenzen verarbeiten
- Codewort wird nicht einzelnem Zahlwert, sondern Intervall zugeordnet

Beispiel

Tabelle 3.22 Zwei Differenz-Codes (variable und konstante Wortlänge) für verlustbehaftete Differenz-Codierung. Hier werden mehrere Differenzen dem gleichen Codewort zugeordnet

Differenz d	Code D_1	Code D_2
−1, 0, 1	1	0000
2, 3, 4	0100	0001
−2, −3, −4	0101	0010
5, 6, 7	0110	0011
−5, −6, −7	0111	0100
8, 9, 10	00100	0101
−8, −9, −10	00101	0110
11, 12, 13	00110	0111
−11, −12, −13	00111	1000
14, 15, 16	000100	1001
−14, −15, −16	000101	1010
17, 18, 19	000110	1011
−17, −18, −19	000111	1100
20, 21, 22	0000100	1101
−20, −21, −22	0000101	1110
23, 24, 25	0000110	$\|d\| > 22$ 1111 und
−23, −24, −25	0000111	danach 8 Bit für den Datenwert
$\|d\| > 25$	00000	und danach 8 Bit für den Datenwert

Übersicht

- Einleitung
- Arithmetische Codierung
- Lauflängencodierung
- Differenzcodierung
- LZW-Algorithmus

Prinzip

- 1977: LZW Abraham Lempel, Jakob Ziv und Terry A. Welch
- Dynamisches Wörterbuchverfahren

[http://www.geeksforgeeks.org/]

- Unterschied zu Huffman- und Arithmetischer Codierung: Berücksichtigung aufeinanderfolgender Zeichengruppen
- Prinzip:
  - Codetabelle:
    - Jeder Eintrag String aus Quellalphabet und zugehöriger Code
    - Wird anfangs mit Einzelzeichen vorbelegt
    - Wird während der Kompression nach und nach erweitert und an Eingabe angepasst
    - Daher: Im Vorfeld keine statistischen Informationen notwendig
    - Codetabelle für Decodierung nicht notwendig

Kompression (I): Algorithmus

- Variablen:
  - P: Präfix
  - c: Aktuelles gelesenes Zeichen
  - Z: Eingabestring

```
( 1) Initialisiere Codetabelle mit Einzelzeichen
( 2) P = ""
( 3) WHILE (Eingabezeichen vorhanden)
( 4)   Lies nächstes Zeichen c aus Z
( 5)   IF Pc in Codetabelle
( 6)      P = Pc
( 7)   ELSE
( 8)     IF Codetabelle nicht voll
( 9)       Trage Pc in nächste freie Position ein
(10)     END IF
(11)     Gib Code für P aus
(12)     P = c
(13)   END IF
(14) END WHILE
(15) Gib Code für P aus
```

Kompression (II): Beispiel (I)

- Komprimiere ABABCBABAB
- Annahme: Codetabelle habe 8 Einträge, Code 3 Bit breit
- Initiale Codetabelle:

Zeichenkette	Ausgabe-Code
A	0 = 000
B	1 = 001
C	2 = 010
	3 = 011
	4 = 100
	5 = 101
	6 = 110
	7 = 111

Kompression (III): Beispiel (II): Ablauf

Schritt	Zeichen c	Präfix P	Eintrag in Codetabelle	Ausgabe
0	---	---	Vorbelegung	---
1	A	A	---	---
2	B	B	AB=3	0
3	A	A	BA=4	1
4	B	AB	---	---
5	C	C	ABC=5	3
6	B	B	CB=6	2
7	A	BA	---	---
8	B	B	BAB=7	4
9	A	BA	---	---
10	B	BAB	---	---
11	---	---	---	7

Resultierender Code: 013247 = `000 001 011 010 100 111`

Kompression (IV): Beispiel (III)

- Finale Codetabelle:

Zeichenkette	Ausgabe-Code
A	0 = 000
B	1 = 001
C	2 = 010
AB	3 = 011
BA	4 = 100
ABC	5 = 101
CB	6 = 110
BAB	7 = 111

Dekompression (I): Idee

- Prinzipielle Vorgehensweise:
  - Lege Code-Tabelle an, belege mit Eingabezeichen vor
  - Algorithmus liest Codewort, sucht in Codetabelle und gibt Zeichen aus
  - Zusätzlich: Hänge an zuletzt decodierten String erstes Zeichen des aktuell decodierten String an und trage Ergebnis in Codetabelle ein
- Sonderfall:
  - Wird String in Codetabelle eingetragen und im nächsten Schritt bereits wieder verwendet, so kann er bei Dekompression noch nicht vorhanden sein
  - Fehlender Code: Verlängerung des Präfix um erstes Zeichen des zuvor ausgegebenen String
  - In Codetabelle einzutragender String ist gleich auszugebendem String

Dekompression (II): Algorithmus

- Variablen:
  - P: Präfix, c: Aktuelles gelesenes Codewort

```
( 1) Initialisiere Codetabelle mit Einzelzeichen
( 2) P = ""
( 3) WHILE (Codeworte vorhanden)
( 4)   Lese nächstes Codewort c
( 5)   IF c in Codetabelle
( 6)     Gib zu c gehörenden String s aus
( 7)     k = Erstes Zeichen von s
( 8)     Trage Pk in Codetabelle ein, falls nicht vorhanden
( 9)     Setze P auf zu Code c gehörenden String
(10)   ELSE //Sonderfall
(11)     k = Erstes Zeichen von P
(12)     Gebe Pk aus
(13)     Trage Pk in Codetabelle ein
(14)     P = Pk
(15)   END IF
(16) END WHILE
```

Dekompression (III): Beispiel (I)

- Zeichen A,B,E,N,U Vorbelegung mit 0,1,2,3,4
- Code 4 Bit breit
- Codierte Eingabe: `0001 0000 0011 0110 0010 0111 0011 0101 0100`

Schritt	Code c	Zeichen k	Ausgabe	P	Eintrag in Codetabelle
0	---	---	---	---	Vorbelegung
1	0001	B	B	B	---
2	0000	A	A	A	BA(5)
3	0011	N	N	N	AN(6)
4	0110	A	AN	AN	NA(7)
5	0010	E	E	E	ANE(8)
6	0111	N	NA	NA	EN(9)
7	0011	N	N	N	NAN(10)
8	0101	B	BA	B	NB(11)
9	0100	U	U	U	BU(12)

Dekompression (IV): Beispiel (II)

- Finale Codetabelle:

Code	Zeichenkette
A	0000
B	0001
E	0010
N	0011
U	0100
BA	0101
AN	0110

Code	Zeichenkette
NA	0110
ANE	1000
EN	1001
NAN	1010
NB	1011
BU	1100

Zusammenfassung (I)

- Einleitung:
  - Kompressionsfaktor
  - Kompressionsrate
  - Statistisches Kompressionsverfahren
  - Verlustfrei vs. verlustbehaftete Kompression

- Arithmetische Codierung:
  - Idee
  - (De-)Kompression

- Lauflängencodierung:
  - Idee
  - Binärwerte
  - Bilder
  - Quadtrees

Zusammenfassung (II)

- Differenzcodierung:
  - Verlustfreie Variante
  - Verlustbehaftete Variante

- LZW-Algorithmus:
  - Prinzip
  - Kompression
  - Dekompression

Übungen (I)

- **Aufgabe 1**

  Finden Sie im folgenden Rätsel fünf Begriffe aus diesem Vorlesungsabschnitt!

```
O Q K W B Z G Z B J B T K L V R C K E F
J Z S X R D D C V Q H L C V V I O J E B
F D M S W U B M B P E I T L F M L R R Z
Y N V O M P X P S P A T E X P X R Z T F
E V Z A K D I A M E K F V R V N S C D C
G N I D O C N E H T G N E L N U R R A R
D L J M K V L Y R T O S P I S O C S U Q
G B P M A V K G R F S E P Y T X A F Q V
K O A L M Q W W S I T W V I O T E P F U
D M K X Y Y O W O Z H S F F M N O H D D
L R C B L Q L N U J Y M U X S D O U R Q
G N H K I E S T W O D F T L H X L T M A
E U D F I F Z O B W U I I K R F G I O L
N V J R A L P R D A D I N C Q E C M L D
G M M K Z Y X Y O J I Y G T C B V P T H
E T T E N X F V P X V O W W W V D G P F
Q O Z W J P J F B O N X T J T O B D L P
R T S Z N V D Z X Y O Y U A D X R D R O
D V Y K F D N G W O C E Q W M I P E L G
O H K N M O P I I O G P W H K G W R M S
```

Übungen (II)

- **Aufgabe 2**

  Führen Sie eine Lauflängencodierung durch für

  a) AAAAAAAAADDDDDAABBBBBBCCCCCBABABDDDDDDDEEEEEE

  b) 000000111111000000000000000000001111111111111111100000

  c) 0101001010010110010100100100101010101010000101010101010101

  Geben Sie auch jeweils den Kompressionsfaktor an!

  Bitte geben Sie auch alle Annahmen und Rahmenbedingungen an!

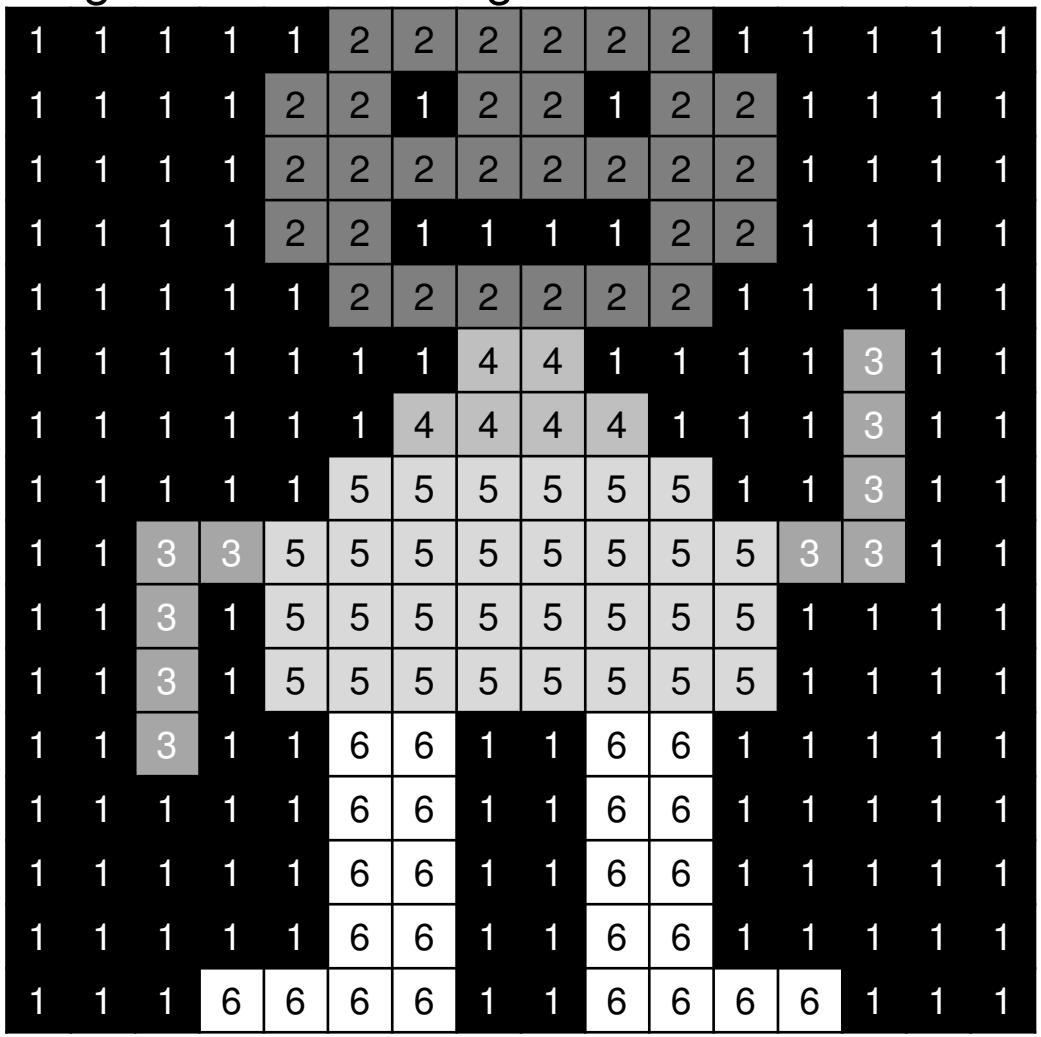
- **Aufgabe 3**

  Lässt sich die Eingabe `ANTANANARIVO` (Hauptstadt von Madagaskar) oder `PAPAYAPALMEN` durch den LZW-Algorithmus effizienter komprimieren?

Übungen (III)

- **Aufgabe 4**

  Gegeben sei das folgende Bild mit Grauwerten 1 bis 6:



Übungen (IV)

a) Bestimmen Sie die Auftrittswahrscheinlichkeiten der Grauwerte!

b) Geben Sie für die Grauwerte einen Binärcode mit minimaler konstanter Wortlänge an und berechnen Sie die Größe des so codierten Bildes.

c) Konstruieren Sie unter Verwendung des Huffman-Algorithmus einen optimalen Code variabler Wortlänge. Wie viele Bit benötigt man hiermit zur Speicherung des Bildes? Wie groß ist der Kompressionsfaktor?

d) Konstruieren Sie einen Lauflängencode. Wie viele Bit benötigt man hiermit zur Speicherung des Bildes? Wie groß ist der Kompressionsfaktor? (Platzbedarf für Zeilenumbrüche kann vernachlässigt werden)

e) Geben Sie eine Quadtree-Speicherung des Bildes an!

Übungen (V)

- **Aufgabe 5**

  Gegeben sei das Wort „AAAAHHABBLLABBLAAA“

  a) Führen Sie eine arithmetische Codierung durch!

  b) Führen Sie eine Huffman-Codierung durch!