Dr. Harald Sack
Institut für Informatik
Friedrich-Schiller-Universität Jena

Sommersemester 2008

http://www.informatik.uni-jena.de/~sack/SS08/

Rechnernetze und Internettechnologien

1 | **21.04.2008 – Vorlesung Nr. 2** | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13

2. Grundlagen der Digitalisierung - Datenrepräsentation im Computer

Rechnernetze und Internettechnologien

2. Grundlagen der Digitalisierung – Datenrepräsentation im Computer

Rechnernetze und Internettechnologien

2.1 Kodierung und Zeichencodes

- **Kodierungen** (1/1)
  - Textdateien werden üblicherweise im Computer im ASCII-Code oder Unicode gespeichert.
  - Im ASCII-Code wird jedem Zeichen einer Nachricht ein 8 Bit Folge zugewiesen

- Ein **Code** ist eine Abbildung von Zeichen auf Folgen von Bits.
- Diese Bitfolgen werden als **Codewörter** bezeichnet.
- Die Menge der vom Code abgebildeten Zeichen heißt **Alphabet**.

*Ein **Code** f über den Alphabeten A und B ist eine (eindeutige) Abbildung der Form $f: A \rightarrow B$. Der Code ordnet Wörtern aus Symbolen des Alphabets A Wörter aus dem Alphabet B zu. Ein Code ist entzifferbar, wenn es eine Umkehrabbildung f^{-1} gibt, die jedem Nachrichtenwort aus B wieder das ursprüngliche Wort aus A zuordnet.*

Rechnernetze und Internettechnologien

2.1 Kodierung und Zeichencodes

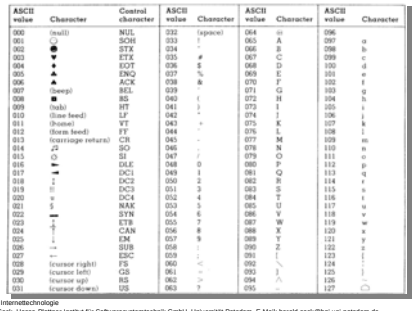
- **ASCII-Code** (1/5)
  - **(A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
  - in **1950er** Jahre gab es keinen einheitlichen Kodierungsstandard von Zeichen/Ziffern für **Computersysteme**
  - **1961** (Robert Bemer, IBM) entwickelt **7-Bit** Kodierung
    - basiert auf 7-Bit FIELDATA Code
    - 99 Zeichen (Ziffern, Großbuchstaben, Steuersymbole)
    - ECMA belegt Rest mit Kleinbuchstaben
  - **1963** erstmals von ANSI standardisiert als ANSI X3.4-1968
  - **1974** ISO I-646 Standard
  - *...von IBM aber erst **1981** eingesetzt (IBM PC)*


Robert Bemer
(1920 -2004)

Rechnernetze und Internettechnologien

2.1 Kodierung und Zeichencodes

- **ASCII-Code** (2/5)



- **ASCII-Code** (3/5)
  - **Problem:**
    - 7 Bit ausreichend für 128 Zeichen
    - International existieren aber viele Umlaute und Sonderzeichen
  - **Lösung:**
    - **ISO 8859-x** Standard,
      - 8-Bit ASCII-Kodierung mit nationalen Erweiterungen (Umlaute)
      - 0-127 identisch mit Standard-ASCII
      - 128-159 seltene Steuerzeichen
      - **160-255 nationale Erweiterungen**

- **ASCII-Code** (4/5)
  - nationale Erweiterungen
    - ISO-8859-1 Westeuropa, Amerika, Australien, Afrika (ISO-8859-15)
    - ISO-8859-2 Osteuropa (ISO-8859-16)
    - ISO-8859-3 Esperanto und Maltesisch
    - ISO-8859-4 Baltisch, Grönland, Lappland
    - ISO-8859-5 Bulgarien, Mazedonien, Russisch, Serbien, Ukraine
    - ISO-8859-6 Arabisch (ohne Persisch/Urdu)
    - ISO-8869-7 Griechenland
    - ISO-8859-8 Hebräisch
    - ISO-8859-9 Island, Türkei
    - ÎSO-8859-10 Grönland, Lappland
    - ISO-8859-11 Thai
    - ISO-8859-12 Indien
    - ISO-8859-13 Baltikum
    - ISO-8859-14 Gälisch, Walisisch

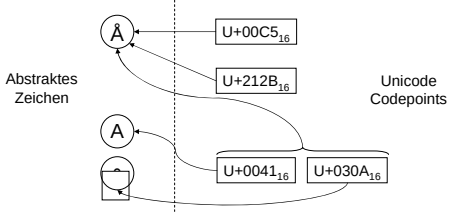
- **ASCII-Code** (5/5)
  - **Problem:**
    - 8 Bit sind ausreichend für 256 Zeichen
    - Chinesische, japanische, koreanische oder indische Schriftzeichen lassen sich damit nur schwer repräsentieren

  - Bsp.: chinesische Schriftzeichen in Japan
    - Gakashu Kanji:
      1006 Zeichen (Grundschule)
    - Joyo Kanji
      1945 Zeichen (offizielle Dokumente, Zeitung)
    - Jinmei-yo Kanji
      285 Zeichen (Namen)

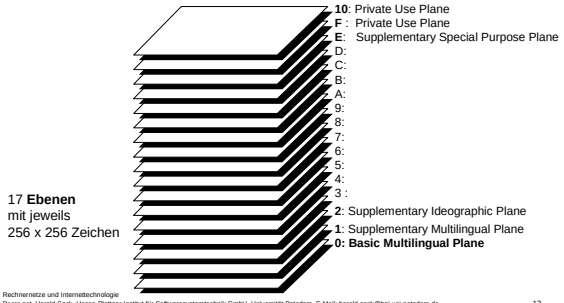
- **Unicode** (1/7)
  - **1984** begann die Entwicklung an einer universellen Zeichenkodierung (Unicode)
  - **1992** ISO/IEC 10646 Universal Character Set (UCS) Standard
  - ursprünglich 16-Bit, dann **21** (32)**-Bit Kodierung**
  - ermöglicht **multilinguale Textverarbeitung**
  - potenziell sind $2.147.483.648 = 2^{21}$ Zeichen möglich
  - genutzt werden nur 17 Ebenen (planes) mit je 65.536 Zeichen

- Grundidee:
  Jedem potenziellen Zeichen wird ein so genannter **Codepoint** zugeordnet anstelle einer Glyphe
  - Zeichen (character) = abstrakte Idee eines Buchstabens
  - Glyphe = konkrete grafische Darstellung eines Zeichens

- **Unicode** (2/7)
  - Codepoints
    - Identische Zeichen kommen in unterschiedlichen Alphabeten vor
    - Daher können in Unicode einem Zeichen verschiedene Codepoints zugeordnet werden

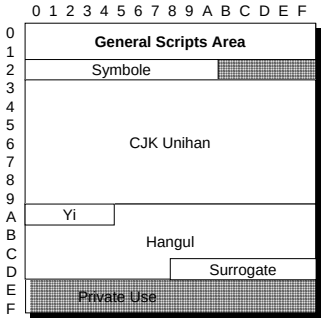


- **Unicode** (3/7)



- **Unicode** (4/7)
  - **Basic Multilingual Plane** (BMP, Unicode 3.2.0, 2002)
  - 49194 ausgewiesene Zeichen
  - unterstützt fast alle gebräuchlichen modernen Schreibsysteme
  - kann in 16-Bit als UTF-16 kodiert werden
  - Schreibweise: **U+xxxx$_{16}$**
  - BMP umfasst
    - 10236 Buchstaben
    - 27786 **CJK**-Unihan-Zeichen (vereinheitlichte **c**hinesische, **j**apanische und **k**oreanische Schrift)
    - 11172 koreanische Hangul-Zeichen
    - 8515 Kontrollsymbole

- **Unicode** (5/7)
  - **BMP**



- **Unicode** (6/7)
  - Unicode Transformation Formate (UTF)
    - Allgemein werden Unicode Codepoints in der folgenden Form dargestellt: **U+xxxxxxxx$_{16}$**
    - Da aber meist nur Codepoints aus dem BMP benutzt werden, wurden effizientere Kodierungen entwickelt, z.B **UTF-8**
    - UTF-8 kodiert Codepoints mit 1 - 4 Bytes Länge

1 Byte	0xxxxxxx				(7 Bit)
2 Byte	110xxxxx	10xxxxxx			(11 Bit)
3 Byte	1110xxxx	10xxxxxx	10xxxxxx		(16 Bit)
4 Byte	1111xxxx	10xxxxxx	10xxxxxx	10xxxxxx	(21 Bit)

    - Wähle für Codepoint stets die kürzeste Kodierungsvariante
    - 1 Byte UTF-8 ist kompatibel mit 7-Bit ASCII

- **Unicode** (7/7)
  - Unicode Transformation Formate (UTF)
    - Beispiele **UTF-8 Kodierung**:

Zeichen	Codepoint	Unicode binär		UTF-8		
y	U+0079$_{16}$	00000000	01111001	**0**1111001		
ä	U+00E4$_{16}$	00000000	11100100	**110**00011	**10**100100	
€	U+20AC$_{16}$	00100000	10101100	**1110**0010	**10**000000	**10**101100

Vgl. The Unicode Consortium: The Unicode Standard, Version 5, Addison-Wesley Professional, 5th ed., 2006

Rechnernetze und Internettechnologien

2. Grundlagen der Digitalisierung – Datenrepräsentation im Computer

- **Was ist Information?** (1/2)

  - **Maßgröße** für die Ungewissheit des Eintretens von **Ereignissen** im Sinne der Wahrscheinlichkeitsrechnung
  - = beseitigte Ungewissheit (z.B. durch Auskunft, Aufklärung, Mitteilung, Benachrichtigung über Gegenstände)

  - Ereignisse = Zeichen (Nachrichtenelemente)
    - Werden durch Auswahlvorgang aus einem Zeichenvorrat von einer Nachrichtenquelle erzeugt
  - Durch diese Festlegung wird Information zu einem **berechenbaren Maß für die Wahrscheinlichkeit zukünftiger Ereignisse** in einem technischen System

- **Was ist Information?** (2/2)
  - **Zeichenkette** = Folge von Elementen eines Alphabets

    Wirsing - *Alphabet = {a,b,c,d,….,A,B,C,D,…}*
    1001001 - *Alphabet = {0,1}*
  - **Nachricht** = übermittelte Zeichenkette, die meist nach bestimmten, vorgegebenen Regeln (**Syntax**) aufgebaut ist.
  - durch Verarbeitung erhält die Nachricht Bedeutung (**Semantik**)
  - durch die Verarbeitung der Nachricht ändert sich der Zustand des Empfängers der Nachricht (**Pragmatik**)

- **Wie messe ich Information?** (1/5)
  - z.B. **kürzeste Beschreibung**, die eine Nachricht benötigt, welche dieselbe Bedeutung für den Empfänger besitzt, wie die ursprüngliche vorgegebene Information (**Beschreibungskomplexität**)
  - **Wie viele Bits benötige ich** mindestens, um eine Nachricht mit einem bestimmten Informationsgehalt zu kodieren?

Alphabet = {a,n,s, <leerzeichen> }
Kodierung: Blockcode mit 2 Bit
Nachricht: *anna an ananas*

a	**00**
n	**01**
s	**10**
<leerzeichen>	**11**

- **Wie messe ich Information?** (2/5)
  - Alphabet = {a,n,s,<leerzeichen>}
  - Nachricht: *anna an ananas*
  - Kodierte Nachricht:

a	**00**
n	**01**
s	**10**
<leerzeichen>	**11**

00 01 01 00 11 00 01 11 00 01 00 01 00 10
a n n a a n a n a n a s

  - Gesamtinformation: **14 x 2 Bit = 28 Bit**
  - Mittlerer Informationsgehalt eines Zeichens: **2 Bit**
  - Tatsächlicher Informationsgehalt einer **kompletten Nachricht**?

- **Wie messe ich Information?** (3/5)
  - Betrachte folgende Kodierung:
    - Alphabet = {a,n,s,<leerzeichen>}
    - Nachricht: *anna an ananas*
    - Kodierte Nachricht:

a	**0**
n	**1**
s	**01**
<leerzeichen>	**10**

0 1 1 0 10 0 1 10 0 1 0 1 0 01
a n n a a n a n a n a s

    - Gesamtinformation: **17 Bit**
    - Aber: Dekodierung ist NICHT eindeutig möglich!
  - Jede Folge von Bits muss eindeutig dekodierbar sein

- **Wie messe ich Information?** (4/5)
  - Betrachte folgende Kodierung:
    - Alphabet = {a,n,s,<leerzeichen>}
    - Nachricht: *anna an ananas*
    - Kodierte Nachricht:

a	**1**
n	**01**
s	**001**
<leerzeichen>	**000**

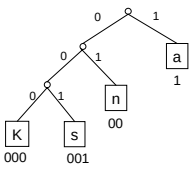
1 01 01 1 000 1 01 000 1 01 1 01 1 001
a n n a a n a n a n a s

    - Gesamtinformation: **25 Bit**
    - Code kann auch als Binärbaum dargestellt werden

- **Wie messe ich Information?** (5/5)
  - Betrachte folgende Kodierung:
    - Alphabet = {a,n,s,<leerzeichen>}
    - Nachricht: *anna an ananas*

a	**1**
n	**01**
s	**001**
<leerzeichen>	**000**

Binärbaumkodierung



Eindeutige Dekodierung:
- Starte mit 1. Bit der Folge an der Wurzel des Baums
- 0 --> links, 1 --> rechts
- Gelangt man an ein Blatt, hat man das Zeichen dekodiert und startet mit dem nächsten Bit wieder an der Wurzel
- Gelangt man an einen inneren Knoten, fährt man mit dem nächst Bit an diesem Knoten fort

- **Entropie** (1/3)
  - Wie messe ich tatsächlich Information?
    - Informationsgehalt ist abhängig von Kodierung einer Nachricht
    - Nach Claude E. Shannon: **Entropie H**


Claude E. Shannon
(1916-2001)

$$H(I) = \sum_{i}^{n} p_i \log_2(\frac{1}{p_i})$$

  - **Nachricht I**,
    besteht aus unterschiedlichen **Symbolen {c_1, c_2, ..., c_n}**
  - jedes Symbol c_i ($1 \le i \le n$) kommt in Nachricht I mit einer bestimmter Häufigkeit (**Wahrscheinlichkeit**) **p_i** vor
  - Die Entropie H(I) ist der gewichtete Mittelwert der Informationsgehalte aller Zeichen c_i

- **Entropie** (2/3)
  - Nachricht: *anna an ananas (14 Zeichen)*

Zeichen c_i	a	n	s	<leerzeichen>
Häufigkeit	6	4	1	2
Relative Häufigkeit $p(c_i)$	6/14	4/14	1/14	2/14
Informationsgehalt $\log_2 1/p_i$	1.222	1,485	3,807	2,807

  - Entropie

$$\sum_{i=1}^{4} p_i \log_2(\frac{1}{p_i}) = \frac{6}{14} \cdot 1{,}222 + \frac{5}{14} \cdot 1{,}485 + \frac{1}{14} \cdot 3{,}807 + \frac{2}{14} \cdot 2{,}807 = 1{,}727 \ \ bit$$

- **Entropie** (3/3)
  - Nachricht: *anna an ananas (14 Zeichen)*
  - Informationsgehalt der gesamten Nachricht:
    Länge x Entropie = 14 x 1,727 bit = 24,183 bit
  - Unsere ursprüngliche Kodierung benötigte 25 Bit

a	1
n	01
s	001
<leerzeichen>	000

  - Da $\lceil 24{,}183 \rceil$ bit = 25 bit
    ⇨ unsere Kodierung ist eine optimale Kodierung

Rechnernetze und Internettechnologien

2. Grundlagen der Digitalisierung – Datenrepräsentation im Computer

- **Redundanz –
  Mehrwert oder Verschwendung?** (1/3)
  - [lat.]*redundare* = überlaufen
  - Anteile einer Nachricht, die **keine** zur Nachricht **beitragende Information** enthalten, also aus dieser entfernt werden können, ohne den eigentlichen Informationsgehalt zu verringern
  - Bsp.:
    - **Whnachtsman**

= Weihnachtsmann

⇨ unsere Sprache enthält bereits Redundanz

- **Redundanz –
  Mehrwert oder Verschwendung?** (2/3)
  - Wozu ist Redundanz dann eigentlich gut?
    - **Bsp.:**
      - **„Frxxdrxch Sxhxllxr xnxversxtät Jxna“**
    - Information kann selbst bei
      - **unvollständiger Übermittlung** oder
      - **Übertragungsfehlern** rekonstruiert werden
    - Information ist **leichter zu lesen/interpretieren**

↑ **Fehlertoleranz und Vereinfachung**
↓ **größere Informationsmenge**

- **Redundanz – Mehrwert oder Verschwendung?** (3/3)
  - Kann man Daten beliebig komprimieren?
  - **Claude E. Shannon** zeigte 1948 die Existenz einer maximalen Grenze, wie weit sich Information ohne Verlust komprimieren läst
  - Er definiert den **Informationsgehalt** einer Nachricht, die **Entropie H**
    - abhängig von **statistischer Natur** der Nachrichtenquelle
    - **keine weitere verlustfreie Komprimierung (kleiner als H) möglich!**

Claude E. Shannon (1916-2001)

vgl. *C.E. Shannon: A Mathematical Theory of Communication, Bell Technical Journal, vol.27, 1948, pp. 379-423.*

- **Komprimierungsvarianten** (1/5)
  - Unter **Komprimierung** versteht man die Beseitigung oder Verringerung der Redundanz einer Nachricht.
  - Ziel der Komprimierung ist es, einen möglichst **redundanzfreien Code** zu erzeugen, aus dem die ursprüngliche Information **eindeutig** und möglichst **ohne Informationsverlust** wieder rekonstruiert werden kann
  - Man kann verschiedene **Varianten** der Komprimierung unterscheiden:
    - logische / physikalische Komprimierung
    - symmetrische / asymmetrische Komprimierung
    - adaptive / semiadaptive / nichtadaptive Komprimierung
    - verlustfreie / verlustbehaftete Komprimierung

- **Komprimierungsvarianten** (2/5)
  - **Logische Komprimierung**
    - fortlaufende Substitution von Symbolen durch andere Symbole
    - **Nutzung der inhärenten Information** der Daten
    - z.B.: **„USA“** statt **„United States of America“**
  - **Physikalische Komprimierung**
    - **ohne Nutzung inhärenter Information**
    - Austausch einer Kodierung durch eine kompaktere
    - kann leicht **automatisiert** werden

- **Komprimierungsvarianten** (3/5)
  - **Symmetrische Komprimierung**
    - Verfahren zur Kodierung und Dekodierung besitzen **dieselbe Berechnungskomplexität** (d.h. sind gleich schwierig)
  - **Asymmetrische Komprimierung**
    - Kodierungs- und Dekodierungsverfahren besitzen **unterschiedliche Berechnungskomplexität**
    - In der Regel ist **Kodierung** komplexer
      - ist dann sinnvoll, wenn nur selten auszuführen

- **Komprimierungsvarianten** (4/5)
  - **Nicht-adaptive Komprimierung**
    - Verwendet **statisches Wörterbuch** mit vorgegebenen Datenmustern (schnell, aufwändiges Wörterbuch)
  - **Adaptive Komprimierung**
    - Für den zu komprimierenden Text wird ein **eigenes Wörterbuch** erstellt (enthält nur Worte aus dem zu komprimierenden Text)
  - **Semi-adaptive Komprimierung**
    - **Mischform** aus adaptiver und nicht-adaptiver Komprimierung

- **Komprimierungsvarianten** (5/5)
  - **Verlustfreie Komprimierung**
    - Nach Kodierung und Dekodierung können die ursprünglichen Daten **unverändert** und **ohne Verlust** rekonstruiert werden
  - **Verlustbehaftete Komprimierung**
    - Beim Komprimieren **gehen** (unwichtige) Teile der ursprünglichen Information **verloren**, so dass diese nach dem Dekodieren nicht exakt mit den ursprünglichen Daten übereinstimmt

Rechnernetze und Internettechnologien

2. Grundlagen der Digitalisierung – Datenrepräsentation im Computer

- **Lauflängenkodierung** (1/1)
  - Einfachste Variante der Komprimierung, Bsp.

AAAADEBBHHHHHCAAABCCCC

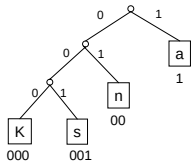
  - **Folgen von sich wiederholenden Zeichen** lassen sich kompakter kodieren, indem man die Folge **nur einmal** angibt und **dazu die Anzahl** der jeweiligen Wiederholungen

⇨ **4ADE2B5HC3AB4C**

- **Huffman-Kodierung** (1/9)
  - Achte darauf, dass **kein Code zugleich der Beginn eines anderen Codes** ist
  - → **Fano-Bedingung**


Robert M. Fano
(*1917)

a	**1**
n	**01**
s	**001**
<leerzeichen>	**000**



  - **Präfixfreie Kodierung:**
    - kein Code ist zugleich Anfang eines anderen Codes
    - lässt sich am leichtesten durch **Binärbaumkodierung** erzeugen

- **Huffman-Kodierung** (2/9)
  - Wie gewinnt man einen möglichst effizienten präfixfreien Code?
  - 1952 Huffman-Kodierung
    - Optimale Kodierung einer Textdatei lässt sich stets als Binärbaum darstellen (jeder innere Knoten besitzt 2 Nachfolger)
    - Tiefe eines Blattknotens gibt die Länge des zugeordneten Codes an
    - Basiert auf folgender Beobachtung:
      „...in einem optimalen Code ist das Codewort für ein Zeichen c_a, das häufiger vorkommt als das Zeichen c_b, höchstens so lang wie das Codewort für c_b."


David Huffman
1925-1999

Dieser Absatz wurde korrigiert...

- **Huffman-Kodierung** (3/9)
  - Wie gewinnt man einen möglichst effizienten präfixfreien Code?

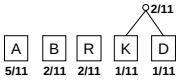
1. Ermittle die **relative Häufigkeit** der zu kodierenden Zeichen

ABRAKADABRA ⇨

Buchstabe	Anzahl	Häufigkeit
A	5	**5/11**
B	2	**2/11**
R	2	**2/11**
K	1	**1/11**
D	1	**1/11**

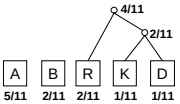
- **Huffman-Kodierung** (4/9)

2. Fasse die beiden Zeichen c_i und c_j mit der **geringsten Häufigkeit** $f(c_i)$ und $f(c_j)$ zusammen zu einem **neuen Knoten** mit der Häufigkeit $f(c_i)+f(c_j)$



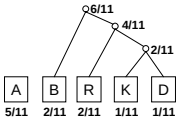
- **Huffman-Kodierung** (5/9)

2. Fasse die beiden Zeichen c_i und c_j mit der **geringsten Häufigkeit** $f(c_i)$ und $f(c_j)$ zusammen zu einem **neuen Knoten** mit der Häufigkeit $f(c_i)+f(c_j)$
3. Fahre fort, bis alle Blattknoten in einem gemeinsamen Baum verbunden sind



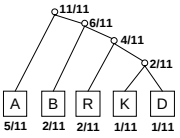
- **Huffman-Kodierung** (6/9)

2. Fasse die beiden Zeichen c_i und c_j mit der **geringsten Häufigkeit** $f(c_i)$ und $f(c_j)$ zusammen zu einem **neuen Knoten** mit der Häufigkeit $f(c_i)+f(c_j)$
3. Fahre fort, bis alle Blattknoten in einem gemeinsamen Baum verbunden sind



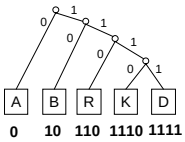
- **Huffman-Kodierung** (7/9)

2. Fasse die beiden Zeichen c_i und c_j mit der **geringsten Häufigkeit** $f(c_i)$ und $f(c_j)$ zusammen zu einem **neuen Knoten** mit der Häufigkeit $f(c_i)+f(c_j)$
3. Fahre fort, bis alle Blattknoten in einem gemeinsamen Baum verbunden sind

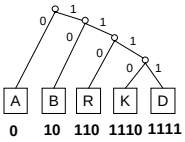


- **Huffman-Kodierung** (8/9)

2. Fasse die beiden Zeichen c_i und c_j mit der **geringsten Häufigkeit** $f(c_i)$ und $f(c_j)$ zusammen zu einem **neuen Knoten** mit der Häufigkeit $f(c_i)+f(c_j)$
3. Fahre fort, bis alle Blattknoten in einem gemeinsamen Baum verbunden sind
4. Interpretiere Baum als **Binärbaumkodierung**



- **Huffman-Kodierung** (9/9)



ABRAKADABRA ⇨ **0 10 110 0 1110 0 1111 0 10 110 0**
A B R A K A D A B R A

• Gesamtlänge: **23 Bit**

• **Achtung**: Es kann mehrere, unterschiedliche optimale Codes geben
  • *Teilbäume mit identischer Tiefe besitzen auch identische Häufigkeit und können beliebig vertauscht werden*

- **LZW-Komprimierung** (1/6)
  - Wörterbuchbasierte Komprimierung
  - **L**empel, **Z**iv, **W**elch
    - 1977 / 1978 LZ77 / LZ78
    - 1984 LZW


  - Adaptives Verfahren (zip-Komprimierung)
  - patentiert von Unisys/IBM,
  - wird für Grafikformate GIF/TIFF genutzt
  - Prinzipieller Ablauf:
    (1) erzeuge aus zu komprimierenden Zeichenketten ein Wörterbuch
    (2) Daten werden mit Wörterbuch kodiert (komprimiert)
    (3) Wörterbuch muss (implizit) mit übertragen (gespeichert) werden

- **LZW-Komprimierung** (2/6)
  - **LZW-Algorithmus** - Ablauf

1. Lese Zeichen aus zu komprimierenden Daten und akkumuliere diese zu Zeichenkette S, solange sich S als Wörterbucheintrag findet.
2. Sobald Zeichen x gelesen wird, für da sich Sx nicht im Wörterbuch findet, fahre folgendermaßen fort:
   - nehme Sx in das Wörterbuch auf
   - Starte eine neue Zeichenkette S mit dem Zeichen x
3. Wiederhole (1,2) bis das Ende der zu komprimierenden Daten erreicht ist.

- **LZW-Komprimierung** (3/6)
  - Beispiel:
    - Algorithmus startet mit Wörterbuch, in dem die ersten 256 Einträge aus den 8-Bit ASCII-Zeichen besteht
    - Die Wörterbucheinträge bestehen typischerweise aus 12 Bit langen Codewörtern (= 4096 Einträge)
    - Zu komprimieren ist die folgende Zeichenfolge:

      ABRAKADABRAABRAKADABRA
    - Als 8-Bit ASCII Kodierung beträgt die Länge der Zeichenkette 22 x 8 Bit = 172 Bit

- **LZW-Komprimierung** (4/6) - Kodierung

Restzeichenkette	Gefundener Eintrag	Ausgabe	Neuer Eintrag
ABRAKADABRAABRAKADABRA	A	A	AB <256>
BRAKADABRAABRAKADABRA	B	B	BR <257>
RAKADABRAABRAKADABRA	R	R	RA <258>
AKADABRAABRAKADABRA	A	A	AK <259>
KADABRAABRAKADABRA	K	K	KA <260>
ADABRAABRAKADABRA	A	A	AD <261>
DABRAABRAKADABRA	D	D	DA <262>
ABRAABRAKADABRA	AB	<256>	ABR <263>
RAABRAKADABRA	RA	<258>	RAA <264>
ABRAKADABRA	ABR	<263>	ABRA <265>
AKADABRA	AK	<259>	AKA <266>
ADABRA	AD	<261>	ADA <267>
ABRA	ABRA	<265>	

- **LZW-Komprimierung** (5/6)
  - Beispiel:
    - Ausgabe:

      ABRAKAD<256><258><263><259><261><265>

      Länge der komprimierten Zeichenkette:
      13 x 12 Bit = 156 Bit
  - Bei der **Dekodierung** wird das Wörterbuch schrittweise rekonstruiert
  - *Dies ist möglich, da die Ausgabe des LZW-Algorithmus stets nur Codewörter enthält, die zu diesem Zeitpunkt bereits im Wörterbuch standen*

- **LZW-Komprimierung** (6/6) - Dekodierung

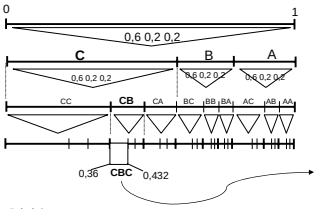
Erstes Zeichen	Ausgabe	Neuer Eintrag
A	A	
B	B	AB <256>
R	R	BR <257>
A	A	RA <258>
K	K	AK <259>
A	A	KA <260>
D	D	AD <261>
<256>	AB	DA <262>
<258>	RA	ABR <263>
<263>	ABR	RAA <264>
<259>	AK	ABRA <265>
<261>	AD	AKA <266>
<265>	ABRA	ADA <267>

- **Arithmetische Kodierung** (1/5)
  - **Zeichen** werden durch **Häufigkeitsintervalle** kodiert
  - **Zeichenfolgen** werden durch bedingte („geschachtelte“) Häufigkeitsintervalle kodiert
  - Verfahren arbeitet theoretisch mit unendlich genauen reellen Zahlen, auch wenn in den eigentlichen Implementierungen wieder auf endlich genaue Integer- oder Fließkommazahlen zurückgegriffen werden muss.
  - Notwendiges Auf-/Abrunden führt zu **Nicht-Optimalität**
  - patentrechtlich geschütztes Verfahren (Q-Coder, IBM)
  - Nähert sich bei sehr langen zu komprimierenden Nachrichten einer optimalen Kodierung an

- **Arithmetische Kodierung** (2/5)
  - **Enkodierung - prinzipieller Ablauf**
    1. Initialisiere **Intervall [0..1]**
    2. Zerlege aktuelles Intervall in **Subintervalle** und weise jedem Subintervall ein Zeichen zu. Die Subintervallgröße entspricht der relativen Häufigkeit der zugehörigen Zeichen.
    3. Das Subintervall, das dem nächsten Zeichen der Eingabe entspricht, wird zum aktuellen Intervall.
    4. Wiederhole (2/3) solange noch Zeichen zu kodieren sind.
    5. Wähle eine Zahl **x** aus dem aktuellen Intervall und zusätzlich die Anzahl der kodierten Zeichen aus.
       x wird so gewählt, dass x möglichst wenig signifikante Nachkommastellen besitzt (d.h. "rund" ist) und sich mit möglichst wenigen Bits darstellen lässt.

- **Arithmetische Kodierung** (3/5)
  - Einfaches Bsp.:
    - kodiere **CBC**

Zeichen	Relative Häufigkeit
A	0,2
B	0,2
C	0,6

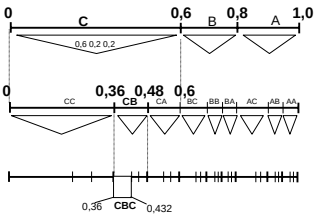


0,375 = 0,011
Zeichenkettenlänge: 3

- **Arithmetische Kodierung** (4/5)
  - **Dekodierung - prinzipieller Ablauf**
    1. Initialisiere das aktuelle **Intervall [0..1]**
    2. Zerlege aktuelles Intervall in **Subintervalle** und weise jedem Subintervall ein Zeichen zu. Die Subintervallgröße entspricht der relativen Häufigkeit der zugehörigen Zeichen.
    3. Finde heraus, in welchem dieser Subintervalle die zu dekodierende Zahl **x** liegt und gebe das **Zeichen** aus, das diesem Subintervall zugeordnet ist. Das Subintervall wird nun zum aktuellen Intervall.
    4. Sind noch weitere Zeichen zu dekodieren, fahre mit (2/3) fort.

- **Arithmetische Kodierung** (5/5)
  - Dekodierung **0,375**

Zeichen	Relative Häufigkeit
A	0,2
B	0,2
C	0,6



0,375 = CBC

Rechnernetze und Internettechnologien

2. Grundlagen der Digitalisierung – Datenrepräsentation im Computer

Rechnernetze und Internettechnologien

2. Grundlagen der Digitalisierung – Datenrepräsentation im Computer

- **Literatur**

  - Ch. Meinel, H. Sack:
    WWW– Kommunikation, Internetworking, Web-Technologien,
    Springer, 2004.
  - P.A. Henning:
    Taschenbuch Multimedia,
    3. Aufl., Fachbuchverlag Leipzig, 2003.
  - The Unicode Consortium:
    The Unicode Standard, Version 5,
    Addison-Wesley Professional, 5th ed., 2006
  - The Unicode Consortium: http://www.unicode.org
  - Thilo Strutz:
    Bilddatenkompression - Grundlagen, Codierung, JPEG, MPEG, Wavelets,
    Vieweg, 2. Aufl. 2002.