Einführung in die Informatik

Fehlererkennung und Kompression

Meik Teßmer

Inhalt der Veranstaltung

Lernziele:

- Fehlererkennung/-korrektur
- Kompression

Motivation

Fehlererkennung

Wo und warum?

Motivation

- Datenübertragung
  - Netzwerke
  - CPU, Festplatte, RAM, ...
  - defekte Medien (CD, DVD, ...)
  - Telefon
  - Funk
- Rauschen
  - stört das Übertragungssignal
  - einzelne Bitfehler vs. gebündelte Fehler

Was ist das Problem?

Ausgangssituation: Daten werden als Serie von Paketen übertragen

Unklar

Kommen die Daten so an, wie sie verschickt wurden?

Was ist das Problem?

Risiko: *Integrität* der Verbindung ist gestört

Integrität bezeichnet „Korrektheit (Unversehrtheit) von Daten und der korrekten Funktionsweise von Systemen“

$\rightarrow$ Integrität muss sichergestellt werden

erster Schritt: Erkennung von Verfälschungen/Fehlern

Fehlererkennung

- Idee: Pakete einzeln sichern mit einer sog. *Prüfsumme*
- allg. Konzept von Prüfsummen:
  - Bestandteile des Pakets werden mit einem Faktor multipliziert
  - Ergebnisse werden aufsummiert
  - Ergebnis (Prüfsumme) wird mit Daten übertragen
- Empfänger kann die Prüfsumme selbst erneut berechnen und dann mit der übertragenen vergleichen

Beispiel: Quersumme als Prüfsumme

einfaches Beispiel: Quersumme

- zu übertragendes Datenpaket: Ziffernfolge 125
- Quersumme von 125: $1 + 2 + 5 = 8$
- Quersumme an Datenpaket anhängen: 1258

$\rightarrow$ übertragen wird 1258

Empfänger bildet ebenfalls die Quersumme und vergleicht

Welche Fehler fallen bei diesem Verfahren aber nicht auf?

Beispiel: ISBN als Prüfsumme

- ISBN: International Standard Book Number
- dient zur Verarbeitung in Warenwirtschaftssystemen und Organisation
- früher 10-, jetzt 13-stellig (inkl. Prüfziffer)

  besteht aus Gruppennummer (Länder), Verlagsnummer, Titel-/Bandnummer und Prüfziffer (letzte Stelle)

Berechnung der Prüfziffer:

$$z_{10} = \left(\sum_{i=1}^{9} i \cdot z_i \right) \ mod\ 11$$

Beispiel: ISBN als Prüfsumme

- Beispiel-ISBN: 3-86680-192-X
- Summe ausrechnen:

$$1 \cdot 3 + 2 \cdot 8 + 3 \cdot 6 + 4 \cdot 6 + 5 \cdot 8 + 6 \cdot 0 + 7 \cdot 1 + 8 \cdot 9 + 9 \cdot 2$$
$$= 3 + 16 + 18 + 24 + 40 + 7 + 72 + 18$$
$$= 198$$

- Modulo rechnen:

  $198 \; mod \; 11 = 0$

  da $198 : 11 = 18$ Rest 0 ist

  $\rightarrow$ ISBN lautet dann 3-86680-192-0

Beispiel: ISBN als Prüfsumme

Prüfung der ISBN 3-680-08783-7

$$\left(\sum_{i=1}^{10} i \cdot z_i \right) mod\ 11 = 0$$

- Summe berechnen:

$$1 \cdot 3 + 2 \cdot 6 + 3 \cdot 8 + 6 \cdot 8 + 7 \cdot 7 + 8 \cdot 8 + 9 \cdot 3 + 10 \cdot 7$$
$$= 297$$

- Module rechnen:

  $297\ mod\ 11 = 0$

  da $297 : 11 = 27$ Rest 0

Zyklische Redundanzprüfung (CRC)

- Idee: zu jedem Datenblock *Redundanz* hinzufügen
- Beispiel von Redundanz: mehrere Netzteile im Computer
  - Aufgabe: Stromversorgung
  - es sind 4 Netzteile vorhanden
  - fällt eines aus, kann eines der noch vorhandenen übernehmen
  - bis zu 3 können ausfallen

  $\rightarrow$ dreifach redundant ausgelegte Stromversorgung
- übertragen auf Informationstheorie:

  Information ist mehrfach in einem Datenpaket vorhanden

Zyklische Redundanzprüfung (CRC)

- gut geeignet, um zufällige Fehler zu erkennen
- Einsatzbereiche: Ethernet, Festplatten
- nicht geeignet, um beabsichtigte Manipulationen zu erkennen
- jeder Block bekommt eine Prüfsumme angehängt
- Verfahren beruht auf Polynomdivision
- Empfänger rechnet Datenpaket modulo CRC-Polynom

Beispiel: CRC

- benötigte Bausteine: CRC-Polynom, Daten, Berechnungsverfahren
- CRC-Polynom: 110101 (muss vorher vereinbart werden)

  Polynom 5. Grades:

$$1x^5 + 1x^4 + 0x^3 + 1x^2 + 0x^1 + 1x^0$$

- Daten: 11011
- aufgefüllt: 11011 00000 (5 Nullen, da Polynom 5 Stellen hat)
- Division von links mit XOR durch das Generatorpolynom

Beispiel: CRC

Division (XOR: 1 XOR 1 = 0):

```
1101100000
110101
------
0000110000
    110101
    ------
       101 (Rest)
```

Rest: Prüfsumme

auf 5 Stellen (Stellenzahl des CRC-Polynoms) mit Nullen auffüllen

$\rightarrow$ zu übertragender Datenblock: 11011 00101

Beispiel: CRC

Überprüfung des empfangenen Datenblocks: Division mit XOR durch das CRC-Polynom

```
1101100101
110101
------
    110101
    110101
    -------
    000000
```

$\rightarrow$ kein Rest: kein Übertragungsfehler

erkannte Fehler (je nach gewähltem Generatorpolynom):

alle 1-Bit-Fehler, ungerade Anzahl verfälschter Bits, Fehler, deren Grad der Polynomdarstellung $<$ Grad des CRC-Polynoms ist

Beispiele für CRC-Polynome

- CRC-CCITT (CRC-4): 10011

$$x^4 + x + 1$$

- USB (CRC-5): 100101

$$x^5 + x^2 + 1$$

- Bluetooth: 110101

$$x^5 + x^4 + x^2 + 1$$

- SD/MMC-Card (CRC-7): 10001001

Hamming-Abstand

- Maß für die Unterschiedlichkeit von Zeichenketten
- Beispiele:
  - 00110 und $00100 \rightarrow$ Hamming-Abstand=1
  - 12345 und $13344 \rightarrow$ Hamming-Abstand=2
- wichtig für die Entwicklung von Codes zur Fehlererkennung (EDC) und -korrektur (ECC)
- Codes mit Hamming-Abstand h: erkennen alle (h-1)-Bit-Fehler
  - $h = 2$: erkennt alle 1-Bit-Fehler
  - $h = 3$: erkennt und korrigiert 1-Bit-Fehler; 2-Bit-Fehler können *falsch* korrigiert werden
  - $h = 4$: erkennt/korrigiert 1-Bit-Fehler, erkennt alle 2-Bit-Fehler, kann sie aber nicht korrigieren

Anwendungsbeispiel

- mögliche Datenblöcke: 00, 01, 10, 11
- Hamming-Distanz h ist jeweils 1
- Sender+Empfänger vereinbaren bestimmte längere Datenblöcke: 001, 010, 100, 111

  $\rightarrow h = 2$
- 011 empfangen: Fehler (kein gültiger Datenblock)

  $\rightarrow$ Fehler erkannt, nicht korrigierbar

  kann aus 111 oder 010 entstanden sein

Anwendungsbeispiel

- Erweiterung auf Datenblöcke mit $h >= 3$: 01011, 01100, 10010, 10101
- 01111 empfangen: Fehler
- Annahme: 1-Bit-Fehler

  $\rightarrow$ 01111 kann nur aus 01011 entstanden sein, also wird 01011 als *übertragen* angesehen
- 2-Bit-Fehler werden erkannt, können aber nicht korrigiert werden

üblich sind Hamming-Distanzen von 4 bis 5

Fehlerkorrekturverfahren

- nicht nur Erkennung, sondern auch Korrektur von Fehlern
- heißen BCH-Codes (Bose-Chaudhuri-Hocquenhem-Codes)
- eine Unterklasse: Reed-Solomon-Codes

  Fehlerkorrektur auf CDs

Kompression

- Idee: Datenblöcke bzw. Informationen werden kompakter dargestellt
- verlustfreie und verlustbehaftete Kompression:
  - verlustfrei: nach der Dekompression identischer Datenblock
  - verlustbehaftet: nach Dekompression *ähnlicher* Datenblock
- Beispiel: MP3
  - verlustbehaftete Audiodatenkompression
  - Konzept: psychoakustisches Modell des menschlichen Hörvorgangs

  $\rightarrow$ Informationen können weggelassen werden

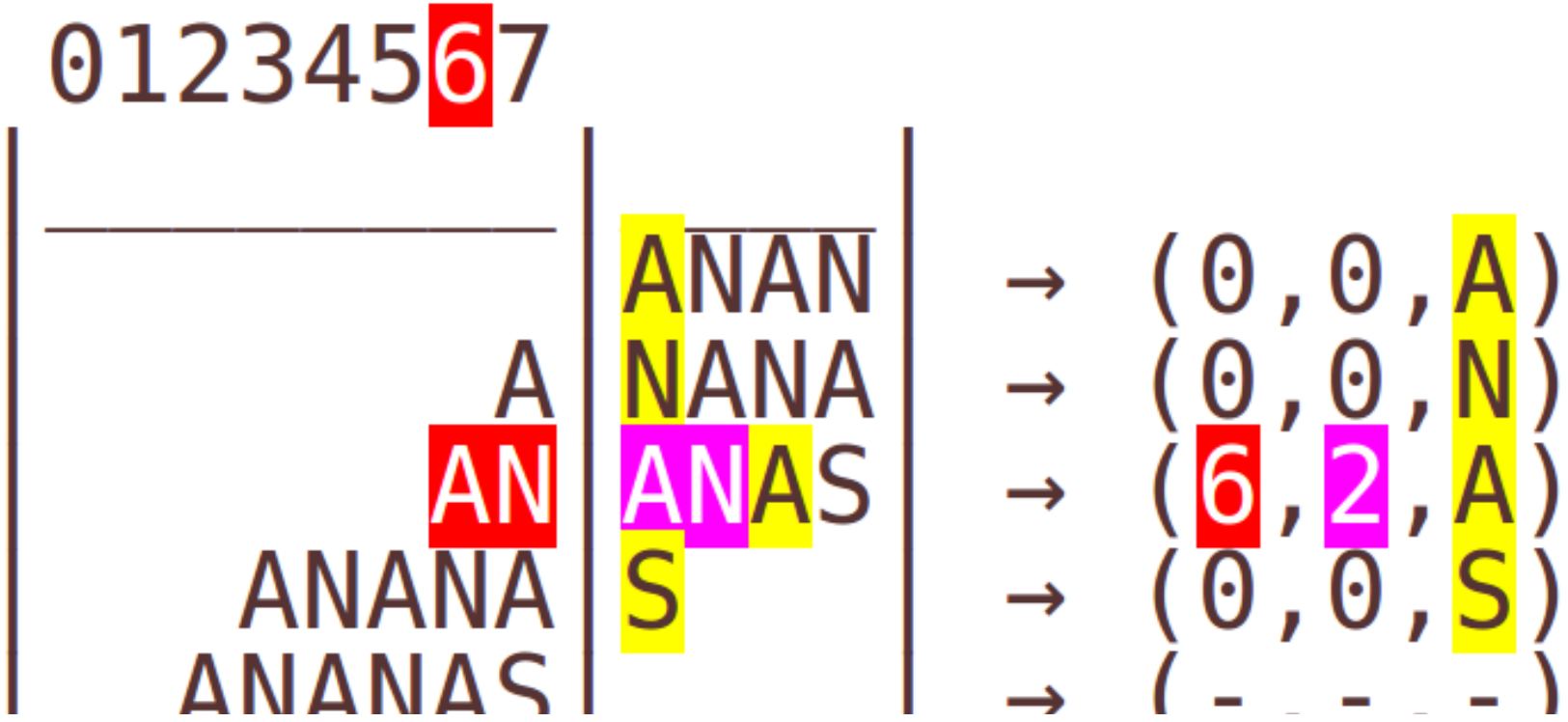
Kompression

- Beispiel: LZ77 (Lempel-Ziv-Algorithmus)
  - verlustfreie Datenkompression
  - nutzt Wörterbuch für Kompression
- Umsetzung:
  - Lexikonfenster
  - *gleitendes Fenster* für Vorschau
  - Fenster wandert über Eingabe
  - Ergebnis: Serie von Tripeln (Lexikoneintrag, Länge, Folgezeichen des Eintrags)

Beispiel: Lempel-Ziv-Algorithmus

- zu komprimierendes Wort: ANANAS
- linke Spalte: Lexikon (8 Bit)
- rechte Spalte: gleitendes Vorschaufenster (4 Bit)

Ergebnis: `[(0,0,A),(0,0,N),(6,2,A),(0,0,S)]`



Beispiel: Dekompression

`[(0,0,A),(0,0,N),(6,2,A),(0,0,S)]`:

```
01234567
          kein Eintrag, Zeichen A
        A kein Eintrag, Zeichen N
       AN ab Stelle 6 zwei Zeichen, dann A
    ANANA kein Eintrag, Zeichen S
   ANANAS fertig
```

weiteres Beispiel: `ABABCDAB`

Diskussion

- Kompressionsgüte: direkt abhängig vom Lexikon

  $\rightarrow$ Vorschaufenster muss eine Mindestgröße haben

- aber je größer, desto mehr Rechenaufwand (Anzahl Vergleiche steigt stark an)
- kombiniert mit Huffman-Kodierung: *Deflate*-Algorithmus (bspw. PNG-Bilder)
- weiterer Kompressionsansatz: Entropiekodierung

Huffman-Kodierung

- Idee: zu jedem Zeichen wird die relative Häufigkeit bestimmt
- baue daraus einen Baum (Zeichen: Blätter)
- entlang der Blätter bis zur Wurzel (oberer Zweig: 1, unter: 0)

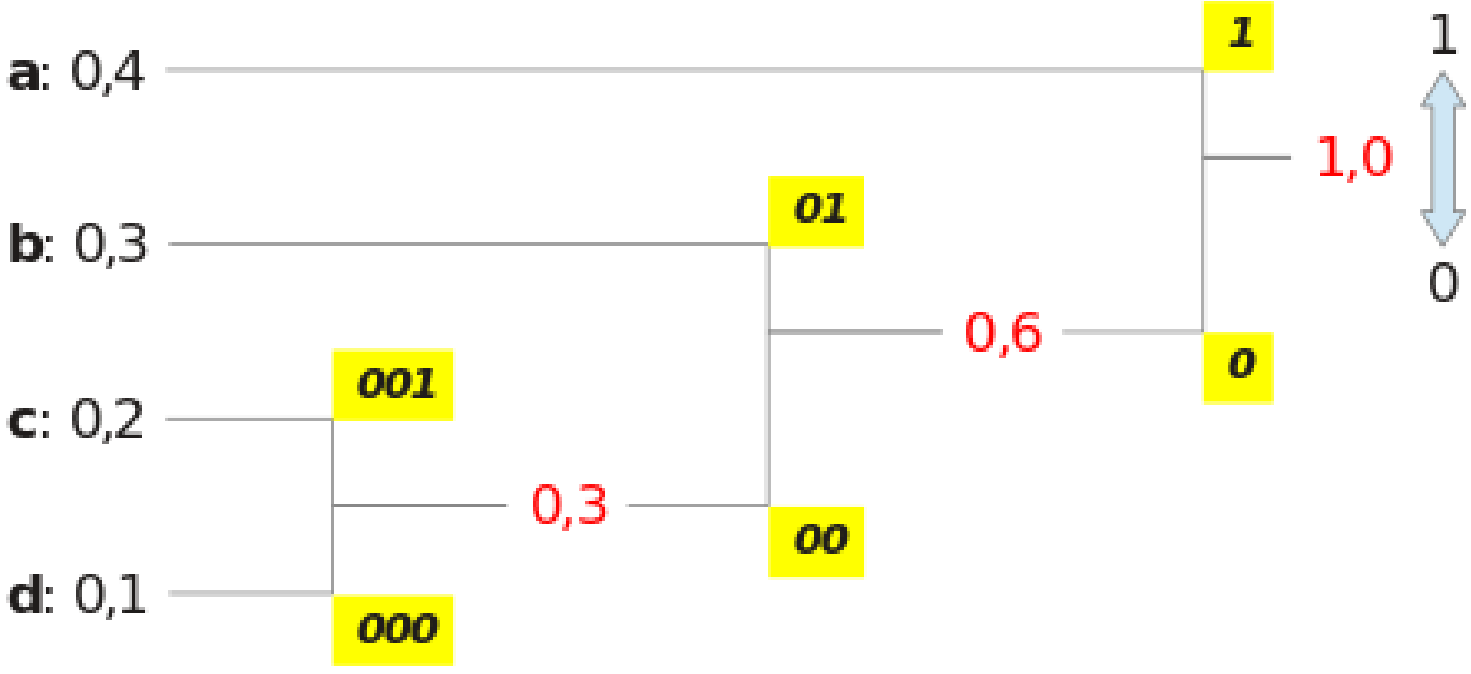


Abbildung 2: Huffman-Baum

Beispiel: Huffman-Kodierung

- verwendete Zeichen: a, b, c, d
- Zeichen für Kodierung: 0, 1 (zwei Zweige)
- Text: a ab abc abcd (ohne Leerzeichen)
- relative Häufigkeiten:

  $p_a = 0.4$, $p_b = 0.3$, $p_c = 0.2$, $p_d = 0.1$
- Teilbäume mit geringster Wahrscheinlichkeit zusammenfassen
  - c und d (0.3)
  - dazu dann b (0.6)
  - dann a (1.0)

Beispiel: Huffman-Kodierung

- Wege ergeben das Code-Wörterbuch:

a	`1`
b	`01`
c	`001`
d	`000`

- Kodierung: Ersetzen der Eingangszeichen durch Codes aus dem Wörterbuch:

a	a	b	a	b	c	a	b	c	d
`1`	`1`	`01`	`1`	`01`	`001`	`1`	`01`	`001`	`000`

- Dekompression: Code-Wörterbuch muss bekannt sein

  dann einfach Code durch Zeichen ersetzen, fertig

Deflate-Algorithmus

- Kombination LZSS (Variante von LZ77) und Huffman-Kodierung
  - zuerst wird mit LZSS komprimiert
  - per Huffman-Kodierung dann kodieren
- einstellbar:
  - hohe Kompression (große Fenster, dynamische Huffman-Tabellen)
  - schnelle Kompression (kleinere Fenster, vorgegebene Tabellen)
- Implementierungen: pkzip, gzip

Einsatzbereiche, Alternativen

- Datenübertragung
- Datenhaltung (Dateien/Ordner komprimieren)
- alternative Algorithmen:
  - bzip2 (Burrows-Wheeler-Transformation + Move-to-Front-Transformation + Huffman-Kodierung), frei
  - 7z (Lempel-Ziv-Markow-Algorithmus (LZMA)), quasi Weiterentwicklung von Deflate, frei
- 7z komprimiert im Durchschnitt besser und schneller als gzip und bzip2 (30% besser als gzip, 15% besser als bzip2)
- Dekompression etwa 10-20 Mal schneller als Kompression

Zusammenfassung

- Fehlererkennung
- Eignung von Zeichenkettengruppen
- mögliche Korrektur von Fehlern
- Kompressionsverfahren

Ende