Grundlagen der Informatik

Prof. Dr. J. Schmidt
Fakultät für Informatik

GDI – WS 2018/19

Codesicherung und Kanalcodierung
Hamming, CRC, Reed-Solomon

Hochschule Rosenheim
University of Applied Sciences

Leitfragen 4.2/3/4

- Wie können nicht-binäre Codes gesichert werden?
- Was ist der Hamming-Code und wie wird dieser gebildet?
- Was sind CRC-Codes?
- Wie funktionieren QR-Codes?

Sicherung nicht-binärer Codes (1)

- Code-Sicherung ist nicht nur auf binäre Codes beschränkt
- Manche Fehler in der Übertragung von z.B. natürlicher Sprache lassen sich aufgrund der Redundanz erkennen und korrigieren
  - d.h. die Korrektur ist aus dem Zusammenhang des Textes ersichtlich
- Manche Fehler führen zu gültigen Worten
  - d.h. sind nicht als Fehler erkennbar
  - (Zweideutigkeit)

Sicherung nicht-binärer Codes (2)

- Beispiele
  - Textverarbeitungsprogramme
    - Fehlerkorrektur nach gültigen Rechtschreibregeln
  - Natürliche Sprache
    - Erkennbare und ggf. korrigierbare Fehler in der natürlichen Sprache

Empfangener Text	**Korrigierter Text**	
Vorlesunk	Vorlesung	→ eindeutig korrigierbar
Vorlosung	Verlosung / Vorlesung ?	→ zweideutig
Der Memsch denkt	Der Mensch denkt	→ eindeutig korrigierbar
Der Mensch lenkt	Der Mensch denkt / lenkt ?	→ nicht erkennbar

Sicherung nicht-binärer Codes (3)

- Gegen Fehleingabe gesicherte Ziffern-Codes
  - Bei Eingabe von Dezimalziffern (z.B. Bestellnummern) ist mit Fehlern zu rechnen
  - Die meisten Methoden zur Aufdeckung von Fehleingaben arbeiten mit Prüfziffern
  - Einfachste Methode
    - Bildung der Quersumme
    - Zur Reduktion auf eine Dezimalstelle ➔ Quersumme Modulo 10
    - Divisionsrest stellt Prüfziffer dar, die am Ende der Zeichenreihe angefügt wird

Sicherung nicht-binärer Codes (4)

- Fehlererkennung
  - Eingabe einer falschen Ziffer wird durch Vergleich der
    - resultierende Prüfzimmer
    - mit der erwarteten Prüfziffer

    erkannt

- Vertauschungsfehler
  - Vertauschen zweier Ziffern wird nicht erkannt
  - Lösungsansatz
    - Gewichtung der Ziffern bei der Quersummenbildung

- Beispiel gesicherter Ziffern-Code

  - 10-stellige ISBN-Buchnummer

$$\text{ISBN} = a_1\text{-}a_2a_3a_4\text{-}a_5a_6a_7a_8a_9\text{-}p$$

Ländergruppen-nummer | Verlagsnummer | Titelnummer | Prüfziffer

  - Berechnung von p
    - $10a_1 + 9a_2 + 8a_3 + 7a_4 + 6a_5 + 5a_6 + 4a_7 + 3a_8 + 2a_9$
    - Bestimmung von p so, dass die Summe durch die Addition mit p zu einer ohne Rest durch 11 teilbaren Zahl ergänzt wird

- Beispiel gesicherter Ziffern-Code
  - Daraus folgt: $0 \leq p \leq 10$
  - Zweistellige Prüfziffer 10 wird durch Einzelzeichen X ersetzt
  - Für eine korrekte ISBN-Nummer gilt:
    $(10a_1 + 9a_2 + 8a_3 + 7a_4 + 6a_5 + 5a_6 + 4a_7 + 3a_8 + 2a_9 + p) \bmod 11 = 0$
  - Sowohl falsch eingegebenen Ziffern als auch vertauschte Ziffern können erkannt werden

- Aufgabe: Überprüfen Sie folgende ISBN-Nummer 3-528-25717-2
  - Ist sie korrekt oder ist bei der Eingabe ein Fehler aufgetreten?

$$10 \cdot 3 + 9 \cdot 5 + 8 \cdot 2 + 7 \cdot 8 + 6 \cdot 2 + 5 \cdot 5 + 4 \cdot 7 + 3 \cdot 1 + 2 \cdot 7 + 2$$

$$= 231 \bmod 11 = 0$$

Internationale Bankkontonummer (IBAN)

- bis zu 34 Zeichen, normalerweise kürzer
- in Deutschland: 22 Stellen:

  DE p_1 p_2 $b_1b_2b_3b_4b_5b_6b_7b_8$ $k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}$

  - $k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}$: ehemalige Kontonummer
  - $b_1b_2b_3b_4b_5b_6b_7b_8$: ehemalige Bankleitzahl
  - p_1 p_2 : Prüfziffern

IBAN – **Berechnung der Prüfziffer**

- Initialisiere $p_1\ p_2$ = 00
- stelle Länderkürzel und $p_1\ p_2$ ganz nach rechts:

  $b_1b_2b_3b_4b_5b_6b_7b_8\ k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}$ **DE 00**

- ersetze Länderkennung durch
  Position des Buchstabens im Alphabet + 9 (A=10, B=11, …):

  $b_1b_2b_3b_4b_5b_6b_7b_8\ k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}$ **13 14** 00

- Berechne Rest bei Division durch 97
- Lege $p_1\ p_2$ so fest, dass sich für den Rest 1 ergibt
- Beachte
  - ganzzahlige Arithmetik mit bis zu 36-stelligen Zahlen nötig
  - lässt sich mit Standarddatentypen in gängigen Programmiersprachen von bis zu 64 Bit nicht durchführen

IBAN – Beispiel: **Generierung**

BLZ: 711 500 00, Kontonummer: 215 632

- Kombination, Länderkürzel/Prüfziffern rechts:

  711500000000215632**DE00**

- Ersetze Kürzel DE durch Positionen im Alphabet + 9:

  711500000000215632**1314**00

- Rest bei Division durch 97 liefert: 711500000000215632131400 mod 97 = 49

- Prüfziffern lauten also: 98 – 49 = 49

- IBAN: DE 49 7115 0000 0000 2156 32

IBAN – Beispiel: **Validierung**

IBAN: DE 49 7115 0000 0000 2156 32

- Länderkürzel/Prüfziffern rechts:

  711500000000215632**DE49**

- Ersetze Kürzel DE durch Positionen im Alphabet + 9:

  711500000000215632**1314**49

- Rest bei Division durch 97 liefert:

  711500000000215632131449 mod 97 = 1
  → IBAN korrekt

Hamming-Code (1)

- Von R. Hamming entwickelt
  - 1-korrigierender Code
  - mit einer Hamming-Distanz von 3

- Idee
  - Einführung von Prüf-Bits
  - Deren binäre Kodierung gibt an
    - >0 ➔ die Position des fehlerhaften Bits
    - =0 ➔ fehlerfreie Übertragung
  - Beispiel
    - 3 zusätzliche Prüf-Bits erlauben $2^3 = 8$ Zustände
    - ➔ 7 Fehlerpositionen kodierbar

Hamming-Code (2)

- Einfachster Hamming-Code
  - (7,4)-Code
    - Block-Code der Länge 7, wobei
    - 4 Bits Nutzinformationen
    - 3 Bits zur Fehlerkorrektur

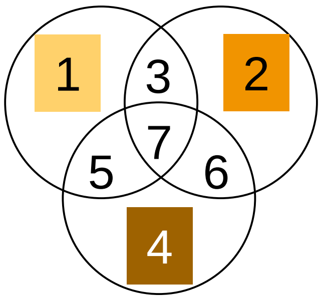
- Allgemein
  - Hamming-Codes der Länge $2^r - 1$, wobei $r \geq 3$ sein muss
    - r Paritätsbits (daraus ergeben sich dann Prüfbits)
    - $2^r - 1 - r$ Informations-Bits

- Grundsätzlicher Aufbau

  - Paritätsbit-Positionen sind Zweierpotenzen ($1 = 2^0$, $2 = 2^1$, $4 = 2^2$, …)
    - gerade Parität, Einsen

  - Beispiel (7,4)-Code

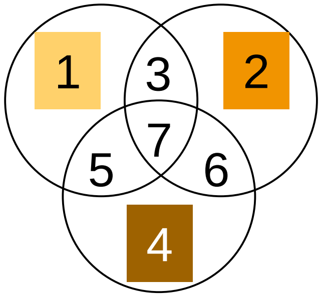
7 D	6 D	5 D	4 P	3 D	2 P	1 P	Paritätsbit an Position
D	-	D	-	D	-	P	2^0
D	D	-	-	D	P	-	2^1
D	D	D	P	-	-	-	2^2



D … Datenbit
P … Paritäts-Bit

Hamming-Code (4)

Dezimal	D	D	D	P	D	P	P
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0
10	1	0	1	0	0	1	0
11	1	0	1	0	1	0	1
12	1	1	0	0	0	0	1
13	1	1	0	0	1	1	0
14	1	1	1	1	0	0	0
15	1	1	1	1	1	1	1



- Hamming-Codes sind optimal
  - Jedes mögliche Wort ist entweder tatsächlich ein Codewort
  - oder hat eine Stellendistanz von 1 zu einem tatsächlichem Codewort

Hamming-Code (5)

- Beispiel: Information 1101 ist zu übertragen
  ➔ Bitfolge 1100110 wird übertragen

7	6	5	4	3	2	1
1	1	0	0	1	1	0
1	-	0	-	1	-	0
1	1	-	-	1	1	-
1	1	0	0	-	-	-

- Bei Änderung eines der Bits 1 bis 7
  ➔ eines oder mehrere der drei Paritäts-Bits sind betroffen

Hamming-Code (6)

7	6	5	4	3	2	1
1	1	0	0	1	1	0
1	-	0	-	1	-	0
1	1	-	-	1	1	-
1	1	0	0	-	-	-

- Bei Änderung eines der Bits 1 bis 7
  ➔ eines oder mehrere der drei Paritäts-Bits sind betroffen
  - Ändert man 7. Bit ➔ Auswirkungen auf alle drei Paritäts-Bits
  - Fehler beim 6. Bit ➔ Auswirkung nur auf Paritäts-Bits 2 und 4
  - Kippen eines Paritäts-Bits ➔ Auswirkung nur auf gekipptes Bit

- Beispiel
  - Bitfolge 1100110 ist zu übertragen, wobei das 6. Bit kippt (Empfänger: 1000110)

7 1	6 0	5 0	**4** **0**	3 1	**2** **1**	**1** **0**	Paritätsbit
1	-	0	-	1	-	**0**	richtig
1	0	-	-	1	**1**	-	falsch
1	0	0	**0**	-	-	-	falsch

  - Bitkombination aus letzter Spalte „von unten nach oben“
    ➔ Dualzahl 110 (Dezimal 6)
    ➔ im 6. Bit ist Fehler aufgetreten

  - Die Prüfbits – dual kodiert – geben die Position des fehlerhaften Bits an

Aufbau des Hamming-Codes

- mit 4 Prüf-Bits?

15 D	14 D	13 D	12 D	11 D	10 D	9 D	2^3 8 **P**	7 D	6 D	5 D	2^2 4 **P**	3 D	2^1 2 **P**	2^0 1 **P**
D	-	D	-	D	-	D	-	D	-	D	-	D	-	P
D	D	-	-	D	D	-	-	D	D	-	-	D	P	-
D	D	D	D	-	-	-	-	D	D	D	P	-	-	-
D	D	D	D	D	D	D	P	-	-	-	-	-	-	-

D … Datenbit
P … Paritäts-Bit

- Sie empfangen folgendes (7,4)-Hamming-Codewort:

  1000111

7 6 5 4 3 2 1
1 0 0 0 1 1 1

1 X 0 X 1 X 1 falsch
1 0 X X 1 1 X falsch
1 0 0 0 X X X falsch

$(7)_{10} = (111)_2$

- War die Übertragung fehlerfrei?
- Falls nein:
  - korrigieren Sie das Codewort!
  - welche Zahl wurde übertragen?

0 0 0 0 1 1 1

$0001_2 = 1_{10}$

Cyclic Redundancy Check (CRC)

- Ziele:
  - Fehlererkennung durch Hinzunahme von möglichst wenig Redundanz
  - Erkennung von
    - Einzel- und Doppelfehlern
    - Burstfehlern (mehrere fehlerhafte Bit am Stück)
  - einfache Implementierung
    (vor allem auch in Hardware)

- Verwendung z.B.
  - Ethernet, USB, Bluetooth, SCSI, Serial ATA, ISDN, DECT (schnurlose Telefone), CAN, FlexRay (Automotive)
  - …

- hänge an eine n Bit lange Nachricht k Bit CRC-Code an
- fasse Nachricht als *Koeffizienten eines dyadischen Polynoms* auf
  - dyadisch = rechne modulo 2
  - Koeffizienten können also nur Werte 0 und 1 annehmen

- Beispiel
  - Nachricht: 10011010
  - Polynom N(x)
    $1 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 0 =$
    $x^7 + x^4 + x^3 + x$

CRC – Idee (2)

- wähle ein Polynom C(x) vom Grad k
  (k = Länge CRC-Code)
- übertrage ein Polynom S(x), das ohne Rest durch C(x) teilbar ist

- Beispiel k = 3
  - $C(x) = x^3 + x^2 + 1$
  - übertrage: S(x) = N(x) + k Bit

CRC – Sender

- Schritte
  - $T(x) = N(x) \cdot x^k$ (→ hänge k Nullen an Nachricht an)
  - berechne Rest R(x) bei Division T(x) / C(x) → T(x) mod C(x)
  - sende S(x) = T(x) – R(x)
    - bei mod 2 → T(x) – R(x) = T(x) + R(x)
    - also: hänge R(x) an N(x) an

- Beispiel
  - N(x) = 10011010 = $x^7 + x^4 + x^3 + x$
  - C(x) = 1101 = $x^3 + x^2 + 1$ → k = 3
  - T(x) = 10011010000 = $x^{10} + x^7 + x^6 + x^4$
  - R(x) = 101 = $x^2 + 1$
  - S(x) = 10011010101 = $x^{10} + x^7 + x^6 + x^4 + x^2 + 1$

CRC – Polynomdivision

- alle Rechnungen mod 2
- daher gilt 1 + 1 = 1 – 1 = 0
- Subtraktion kann durch **stellenweises XOR** erfolgen
- beginne immer beim ersten 1. Koeffizienten der Nachricht N(x) (bzw. der Erweiterung T(x))

CRC – Polynomdivision – Beispiel (Sender)

$C(x) =$	$x^3 + x^2 + 1$	$= 1101$	Generator
$N(x) =$	$x^7 + x^4 + x^3 + x$	$= 10011010$	Nachricht

```
10011010000      ← T(x) = Nachricht mit k Nullen
1101             ← Generator
----   XOR
 1001
 1101
 ----
  1000
  1101
  ----
   1011
   1101
   ----
    1100
    1101
    ----
       1000
       1101
       ----
        101      ← Rest T(x) mod C(x)
```

Übertrage Nachricht und angehängten Rest:

$S(x) = 10011010101$

CRC – Empfänger

- Schritte
  - Empfange Polynom S‘(x)
  - berechne Rest R‘(x) bei Division
    S‘(x) / C(x) → S‘(x) mod C(x)
    - Rest = 0
      - fehlerfreie Übertragung
      - oder nicht detektierbarer Fehler
    - Rest ≠ 0
      - mindestens 1 Bit in Nachricht ist falsch
      - Nachricht muss nochmal gesendet werden

CRC – Beispiel (Empfänger, fehlerfrei)

C(x) =	$x^3 + x^2 + 1$	= 1101	Generator
S‘(x) =	$x^{10} + x^7 + x^6 + x^4 + x^2 + 1$	= 10011010101	empfangene Nachricht

S‘(x) = empfangene Nachricht inkl. CRC

Generator

```
10011010101
1101          XOR
-----
 1001
 1101
 -----
  1000
  1101
  -----
   1011
   1101
   -----
    1100
    1101
    -----
      1101
      1101
      -----
         0
```

Rest S‘(x) mod C(x)

Ergebnis:

- Rest = 0
- CRC-Prüfung ok

CRC – Beispiel (Empfänger, mit Fehler)

$C(x) =$	$x^3 + x^2 + 1$	= 1101	Generator
$S'(x) =$	$x^{10} + x^7 + x^5 + x^4 + x^2 + 1$	= 10010110101	empfangene Nachricht

```
10010110101      S'(x) = empfangene Nachricht
1101                     inkl. CRC
 1000
 1101
  1011
  1101
   1101
   1101
      00101
```

Generator

Doppelfehler

Rest S'(x) mod C(x)

Ergebnis:

- Rest = 101
- CRC-Prüfung nicht ok

CRC – detektierbare Fehler

- empfangen wird Polynom S‘(x) = S(x) + F(x)
  - F(x) ist ein Polynom, das die fehlerhaften Bit repräsentiert
  - F(x) = 0 → keine Fehler
- es können alle Fehler erkannt werden, bei denen F(x) kein Vielfaches von C(x) ist → Anforderungen an C(x)

- Welche Fehler können erkannt werden?
  - alle Einzelfehler, wenn x^k und x^0 von C(x) nicht Null sind
  - alle Doppelfehler, wenn C(x) mindestens drei Terme hat
  - alle r-Bit Fehler für ungerade r, wenn C(x) den Faktor (x + 1) enthält
  - alle Burstfehler der Länge kleiner k, wenn C(x) den konstanten Term enthält
  - die meisten Burstfehler der Länge ≥ k

CRC – verbreitete Generatorpolynome

Name	Verwendung	Polynom
CRC-1	Paritätsbit	$x + 1$
CRC-4-CCITT	Telekommunikation = (15,11)-Hamming	$x^4 + x + 1$
CRC-5-USB	USB	$x^5 + x^2 + 1$
CRC-5-Bluetooth	Bluetooth	$x^5 + x^4 + x^2 + 1 =$ $(x^4 + x + 1)(x + 1)$
CRC-8-ITU-T	ISDN	$x^8 + x^2 + x + 1 =$ $(x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1)(x + 1)$
CRC-15-CAN	CAN-BUS	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1=$ $(x^7 + x^3 + x^2 + x + 1) (x^7 + x^3 + 1)(x + 1)$
CRC-32	Ethernet, Serial ATA, …	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} +$ $x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Aufgabe

Zur Absicherung während der Übertragung sollen Daten mit einem CRC-Code versehen werden.

Die (binäre) zu sendende **Nachricht** lautet:

1100 0110

Als **Generatorpolynom** wird verwendet:

$$x^6 + x + 1$$

Wie lautet die zu sendende Nachricht inklusive des angehängten CRC-Codes?

2D-Barcodes

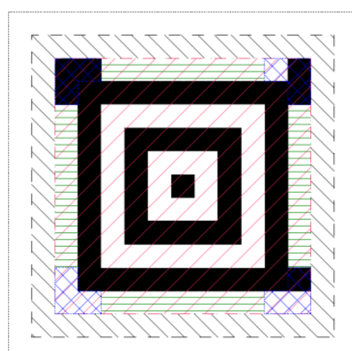
- viele verschiedene Varianten
- typisch:
  - unterschiedlich breite Punkte/Striche
  - dazwischen Lücken → hoher Kontrast zum Auslesen (z.B. mit Laserscanner oder Kamera)

Aztec-Code DataMatrix-Code MaxiCode QR-Code

(Bilder aus Wikipedia)

Aztec-Code

- entwickelt 1995, normiert in ISO/IEC 24778
- Verwendung: Online-Tickets
  - Deutsche/Schweizer/Österreichische Bahn
  - viele Fluggesellschaften
- kodiert 12 – 3000 Zeichen
- Reed-Solomon-Code zur Fehlerkorrektur
  - noch dekodierbar bei Zerstörung von bis zu 25%
- Zentrum: Markierung mit Orientierungspunkten



DataMatrix-Code

- entwickelt 1980er, normiert in ISO/IEC 16022
- Verwendung:
  - Beschriftung von Produkten mit Laser (dauerhaft)
  - Deutsche/Schweizer Post
    (Freimachung ohne Briefmarke)
- kodiert bis ca. 3000 Zeichen
- früher CRC-Code
- jetzt Reed-Solomon-Code
- rechteckiger Rand zum Finden des Codes und für Timing des Lesegeräts

STAMPIT
A00000CEE1
0,55 EUR
01.01.08

Maxicode

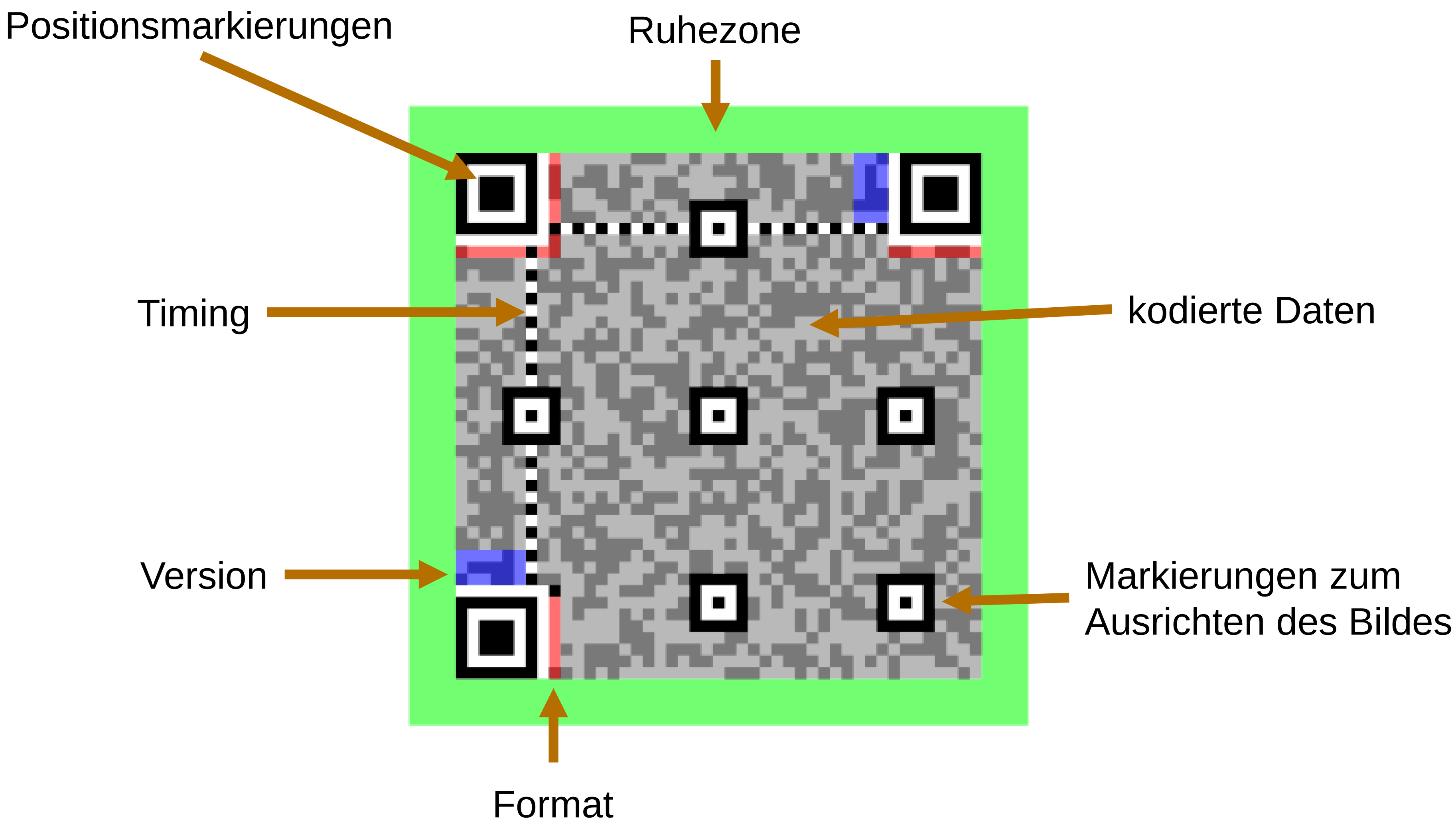
- 1989, normiert in ISO/IEC 16023
- Verwendung: UPS für Paketdaten
- kodiert 93 Zeichen
  - bis zu 8 Codes können kombiniert werden (→ 744 Zeichen)
- Reed-Solomon-Code zur Fehlerkorrektur
- Markierung in der Mitte
- hexagonale Punkte

QR-Code

- QR = Quick Response
- 1994, entwickelt für Automotive-Bereich
- normiert in ISO/IEC 18004
- Verwendung:
  - ursprünglich industrielle Anwendungen
  - mittlerweile verbreitet bei Smartphones

- kodiert ca.1800 – 7000 Zeichen
  - abhängig vom Modus
    (nur Ziffern, lateinische Buchstaben, ganze Bytes,…)
  - und der gewünschten Robustheit gegen Fehler
  - bei mehr Daten: aufteilbar auf bis zu 16 Einzelcodes

- Reed-Solomon-Code zur Fehlerkorrektur
  - je nach Aufwand 7% – 30% der Daten rekonstruierbar
  - je robuster desto weniger Nutzdaten sind speicherbar



(Bild aus Wikipedia)

QR-Codes

so etwas

geht nur wegen guter Fehlerkorrekturmechanismen
→ Reed-Solomon Codes

Reed-Solomon Codes (RS)

- Irving S. Reed und Gustave Solomon, 1960
- Eigenschaften:
  - Erkennung und Korrektur von
    - zufälligen Mehrfachfehlern
    - Burstfehlern
    - Auslöschungen (= fehlenden Daten)
  - nicht-binärer Code
    - also z.B. auf ASCII-Zeichen
    - wird zur eigentlichen Übertragung natürlich nach binär gewandelt
- Verwendung z.B.
  - QR-Codes, Audio-CD, DVD, Blu-Ray, Satelliten-Kommunikation, …

RS – Idee

- fasse Nachricht als *Koeffizienten eines Polynoms* über einem endlichen Körper auf
  - bei q Elementen = rechne modulo q (nur wenn q prim)
  - Koeffizienten können also nur Werte 0, 1, …, q – 1 annehmen
- Codierung durch Auswertung des Polynoms an n verschiedenen Stellen
- Decodierung durch Interpolation

RS – Codierung

- Vorgehen zur Konstruktion von RS(q, m, n)
  - wähle endlichen Körper $\mathbb{F}_q$ mit $q = p^l$ Elementen als Alphabet, p prim, $l \in \{1, 2, 3, ...\}$
  - Nachricht (Block aus m Symbolen) $\mathbf{a} = (a_0, \ldots, a_{m-1})$ aufgefasst als Polynom über $\mathbb{F}_q$:
    $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{m-1}x^{m-1}$
  - wähle n paarweise verschiedene Elemente ($n \geq m$) $u_0, \ldots, u_{n-1} \in \mathbb{F}_q$

- Codierung
  - Auswertung von P(x) an den n Stellen u_i
    - Hornerschema oder Diskrete Fourier-Transformation (DFT)
  - Codewort $\mathbf{c} = (P(u_0), P(u_1), \ldots, P(u_{n-1}))$

- RS(q, m, n) mit q = 5, m = 3, n = 5
- Nachricht **a** = (1, 2, 1)
  - Polynom: $P(x) = 1 + 2x + x^2$
- Auswertung von P(x) an n = 5 Stellen
  - mehr geht nicht, da Körper nur 5 Elemente hat

    $P(0) = 1 + 0 + 0 \quad = 1$

    $P(1) = 1 + 2 + 1 \quad = 4$

    $P(2) = 1 + 4 + 4 = 9 \quad = 4 \mod 5$

    $P(3) = 1 + 6 + 9 = 16 \quad = 1 \mod 5$

    $P(4) = 1 + 8 + 16 = 25 \quad = 0 \mod 5$

- Codewort **c** = (1, 4, 4, 1, 0)

RS – Decodierung – Ausfälle

- RS(q, m, n) toleriert bis zu n – m Ausfälle
  - Ausfall:
    - ein Teil des Codes wurde nicht empfangen
    - Positionen der Ausfälle sind bekannt
  - es wurden also mindestens m Datenpunkte empfangen

- Polynom P(x) hat Grad m – 1
  - aus m Datenpunkten lässt sich P(x) rekonstruieren
  - und damit die Nachricht (= Koeffizienten von P(X))
  - → Lagrange Interpolation

RS – Decodierung – Lagrange Interpolation

- gegeben: mindestens m Datenpunkte (u_i, $P(u_i)$)
  - zur Vereinfachung der Notation:
    Annahme, dass die ersten m empfangen wurden
- setze $g_i(x) = \prod_{j=0, j\neq i}^{m-1} (x - u_j), i = 0, \ldots, m-1$
- es gilt $g_i(u_j) = 0, j \neq i$
- P(x) erhält man aus

$$P(x) = \sum_{i=0}^{m-1} \frac{P(u_i)}{g_i(u_i)} g_i(x)$$

- RS(q, m, n) mit q = 5, m = 3, n = 5 wie vorher
- Auswertung von P(x) erfolgte an den Stellen 0, 1, 2, 3, 4
- gesendetes Codewort **c** = (1, 4, 4, 1, 0)
  - letzte zwei Werte ausgefallen → empfangen (1, 4, 4, ε, ε)
- Berechne Polynome $g_i(x)$:

$(x-0)(x-1)(x-2)$

mod 5 !

$$g_0(x) = (x-1)(x-2) = x^2 - 3x + 2 = x^2 + 2x + 2$$
$$g_1(x) = x(x-2) = x^2 - 2x = x^2 + 3x$$
$$g_2(x) = x(x-1) = x^2 - x = x^2 + 4x$$

RS – Decodierung – Beispiel

- Auswertung der $g_i(u_i)$ an den Stellen u_i= 0, 1, 2

$$g_0(x) = x^2 + 2x + 2$$
$$g_0(0) = 2$$

$$g_1(x) = x^2 + 3x$$
$$g_1(1) = 1 + 3 = 4$$

$$g_2(x) = x^2 + 4x$$
$$g_2(2) = 4 + 8 = 12 = 2$$

$$P(x) = \sum_{i=0}^{m-1} \frac{P(u_i)}{g_i(u_i)} g_i(x)$$

RS – Decodierung – Beispiel

- Bestimmung der multiplikativen Inversen $g_i^{-1}(u_i)$
  - diese existieren immer, da Körper
  - verwende z.B. erweiterten euklidischen Algorithmus

$g_0(0) = 2 \rightarrow g_0^{-1}(0) = 3$ (Test: 2 · 3 = 6 = 1)
$g_1(1) = 4 \rightarrow g_1^{-1}(1) = 4$ (Test: 4 · 4 = 16 = 1)
$g_2(2) = 2 \rightarrow g_2^{-1}(2) = 3$ (Test: 2 · 3 = 6 = 1)

$$P(x) = \sum_{i=0}^{m-1} \frac{P(u_i)}{g_i(u_i)} g_i(x)$$

(1, 4, 4, ε, ε)

- Produkt $P(u_i)g_i^{-1}(u_i)$

$$P(0)g_0^{-1}(0) = 1 \cdot 3 = 3$$
$$P(1)g_1^{-1}(1) = 4 \cdot 4 = 16 = 1$$
$$P(2)g_2^{-1}(2) = 4 \cdot 3 = 12 = 2$$

- Ergebnis:

$$P(x) = \sum_{i=0}^{2} \frac{P(u_i)}{g_i(u_i)} g_i(x) = 3g_0(x) + 1g_1(x) + 2g_2(x)$$

$$= 3(x^2 + 2x + 2) + (x^2 + 3x) + 2(x^2 + 4x)$$

$$= 6x^2 + 17x + 6$$

$$= x^2 + 2x + 1$$

→ ursprüngliche Nachricht war (1, 2, 1)

RS – Decodierung – Fehlerkorrektur

- RS(q, m, n) hat eine Hamming-Distanz von n – m + 1
- damit lassen sich also (n – m) / 2 Fehler korrigieren

Beweis:

- für $n \geq m$ können zwei Polynome nur an m – 1 Stellen die gleichen Werte haben
  - sonst wären sie identisch und die Nachrichten auch
  - die Werte der Polynome unterscheiden sich also an n – m + 1 Stellen
    (= minimaler Abstand zwischen zwei Codewörtern)

- Man nehme zwei Polynome mit noch unbekannten Koeffizienten:
  - $f(x) = f_0 + f_1x + f_2x^2 + \cdots$ vom Grad $\left\lceil\frac{n-m}{2}\right\rceil$
  - $g(x) = g_0 + g_1x + g_2x^2 + \cdots$ vom Grad $\left\lceil\frac{n-m}{2}\right\rceil + m - 1$
- Konstruiere daraus ein neues Polynom
  - $p(x, y) = yf(x) + g(x)$
- Bestimme die Koeffizienten von $p(x, y)$ so, dass gilt
  - $p(u_i, y_i) = 0$,
    wobei $y_i = P(u_i)$ die empfangene (fehlerhafte) Nachricht ist
- Die ursprünglich gesendete Nachricht ergibt sich aus den Koeffizienten des Polynoms

$$-\frac{g(x)}{f(x)}$$

RS – Decodierung – Beispiel

- RS(q, m, n) mit q = 5, m = 3, n = 5 wie vorher
  - (n – m) / 2 = (5 – 3) / 2 = 1 → 1 Fehler korrigierbar
- Auswertung von P(x) erfolgte an den Stellen 0, 1, 2, 3, 4
- gesendetes Codewort **c** = (1, 4, 4, 1, 0)
  - eine Stelle falsch → empfangen (1, 4, 0, 1, 0)
- Polynome:
  - $f(x) = f_0 + f_1 x$ vom Grad $\left\lceil \frac{n-m}{2} \right\rceil = 1$
  - $g(x) = g_0 + g_1 x + g_2 x^2 + g_3 x^3$ vom Grad $\left\lceil \frac{n-m}{2} \right\rceil + m - 1 = 3$
- ergibt
  - $p(x, y) = y f(x) + g(x) = f_0 y + f_1 x y + g_0 + g_1 x + g_2 x^2 + g_3 x^3$

- $p(x,y) = yf(x) + g(x) =$
  $$f_0 y + f_1 xy + g_0 + g_1 x + g_2 x^2 + g_3 x^3$$
- Paare (u_i, y_i) für $p(u_i, y_i) = 0$
  empfangen (1, 4, 0, 1, 0):
  (0,1), (1,4), (2,0), (3,1), (4,0)
- Gleichungssystem:

(0;1) $f_0 + g_0 = 0 \longrightarrow g_0 = -f_0 = 4f_0$ einsetzen in restliche Gleichungen + Reduktion mod 5

(1;4) $4f_0 + 4f_1 + g_0 + g_1 + g_2 + g_3 = 0$

(2;0) $g_0 + 2g_1 + 4g_2 + 8g_3 = 0$

(3;1) $f_0 + 3f_1 + g_0 + 3g_1 + 9g_2 + 27g_3 = 0$

(4;0) $g_0 + 4g_1 + 16g_2 + 64g_3 = 0$

Achtung: Rechnung mod 5!

- ergibt

$$
\begin{aligned}
3f_0 + 4f_1 + g_1 + g_2 + g_3 &= 0 \\
4f_0 + 2g_1 + 4g_2 + 3g_3 &= 0 \\
3f_1 + 3g_1 + 4g_2 + 2g_3 &= 0 \\
4f_0 + 4g_1 + g_2 + 4g_3 &= 0
\end{aligned}
$$

- Lösen des Gleichungssystems
  - z.B. mit Gauß-Elimination
  - 5 Unbekannte, 4 Gleichungen → eine frei wählbar
  - beachte: endlicher Körper, Inverse bzgl. Multiplikation: 1 ↔ 1 , 2 ↔ 3, 3 ↔ 2, 4 ↔ 4

- Ergebnis:

$$f_0 = 1,\ f_1 = 2,\ g_0 = 4,\ g_1 = 1,\ g_2 = 0,\ g_3 = 3$$

RS – Decodierung – Beispiel

- Polynome:
  - $f(x) = f_0 + f_1 x = 1 + 2x$
  - $g(x) = g_0 + g_1 x + g_2 x^2 + g_3 x^3 = 4 + x + 3x^3$

- Berechne $\frac{g(x)}{f(x)}$

$$
\begin{array}{l}
(3x^3 + x + 4) : (2x + 1) = 4x^2 + 3x + 4 \\
- \underline{(3x^3 + 4x^2)} \\
\qquad (x^2 + x + 4) \\
\quad - \underline{(x^2 + 3x)} \\
\qquad\qquad (3x + 4) \\
\qquad\quad - \underline{(3x + 4)} \\
\qquad\qquad \text{---------}
\end{array}
$$

- Nachricht = $-\frac{g(x)}{f(x)} = -(4x^2 + 3x + 4) = x^2 + 2x + 1$
  → gesendet wurde (1, 2, 1)

- Decodierung in der Praxis
  - mit schnelleren (und komplizierteren) Verfahren
  - z.B. mit Berlekamp–Massey Algorithmus

- typische RS-Codes
  - CD: zwei hintereinander geschaltete RS-Codes
    - CIRC: Cross-Interleaved Reed-Solomon Coding
    - RS(33, 28, 32) und RS(29, 24, 28)
    - Burstfehler bis 4000 Bit (ca. 2,5mm Kratzer) exakt korrigierbar
    - Fehler hier = Ausfälle
  - DVD: ähnlich wie CD, aber größere Codes
    - RS(209, 192, 208) und RS(183, 172, 182)
  - QR: nicht lesbare Teile des Codes = Ausfälle