





Hamming-Codes: per fehlendes Code  
 Basis Vektoren: Menge lin. unabh. Vektoren, mit denen sich alle Plate erzeugen lassen  
 Pastition: perfekte Code (disjunkte Mengen, die alles einschließen)  
 Menge der Wörter, die zu 000 dekodiert werden:  $\{5000, 100, 010, 001\}$   
 beim Code der Länge  $n$  mit  $d(C)=k$   
 $|S(v)| = \frac{n}{k} \cdot \binom{n}{k}$   
 $|S(v)| = \sum_{i=0}^{n-k} \binom{n}{i} d(v,w) \leq k$

## Breitensuche

- alle Kn. undiscovered
- Startknoten disc. und in queue
- • kn. dequeuen und auf Suchgröße prüfen
- undisc. Nachbarknoten zu queue hinzuf. und disc.

## Tiefensuche

Wie Breitensuche, bloß auf stack anstatt queue

1.0 2. stack: 10 2. 1.1

## Topsort

$G=(V,E)$  mit  $(u,v) \in E \Rightarrow t(u) < t(v)$   
kein Topsort für Kreis/Schlinge

- alle Kn. weiß
- in Tiefensuche besuchte weiß grau setzen.  
→ wenn auf grau befind. Fehler! (Kreis)
- bei Rekursion der Tiefensuche Knoten schwarz markieren und an Liste anfügen  
→ Liste rückwärts ist topsort

LZ Tiefen-/Breitensuche/Topsort:  
 $O(|V|+|E|)$

Lineare Suche: LZ:  $O(n)$

Voraussetzung: sortiertes Array

- in der Mitte starten (abrunden)
- wenn gesuchte Zahl kleiner als gewählte ist, linkes Teilarray auf gleiche Weise untersuchen. Sonst rechtes.

## Suchbaum

- binärer Wurzelbaum
- linker Teilbaum enthält kleinere Knoten
- rechter Teilbaum größere
- suche über Folgen der Pfade (Art binäre Suche)

LZ:  $O(\log(n))$  vollst. bal.  
 $O(n)$  linear entartet

## Hashing

Abbildung Schlüssel → pos. in Array  
 $h(s) = s \bmod m$  → Größe der Hash-tabelle

bei Überlauf: Überlaufliste:  
pos. in Array hat Liste mit allen entspr. Hashes.  
im günstigen Fall:  $O(1)$

Bsp:  $G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$  PCC des Länge 4  
→ aus  $(b_1, b_2, b_3)$  wird  $C = \{(b_1, b_2, b_3, \sum_{i=1}^3 b_i)\}$

## Quicksort

- 1. elem. markieren (Pivot)  
→ alle elem., die kl. sind als Pivot links einsortieren, die größer sind rechts
- Rekursiver Aufruf quicksort über linke/rechte Teillisten
- nicht gut bei linear entarteten Listen

LZ:  $O(n \log n)$  worst-case:  $O(n^2)$

## Mergesort

- Liste halbieren (abgerundet)
  - immer weiter halb. bis nur noch 1-elem. Listen
  - 1-elem. Listen in Rekursion zusammenführen & dabei sortieren
- $\begin{matrix} 5 & 6 & 3 \\ \uparrow & \uparrow & \uparrow \\ 5 & 6 & 3 \end{matrix} \Rightarrow \begin{matrix} 5 & 6 & 3 \\ \uparrow & \uparrow & \uparrow \\ 5 & 6 & 3 \end{matrix}$
- LZ:  $O(n \log(n))$  [ $O(n)$  auf jeder Ebene]

Codierung: inv. Elem.:  $a + a^{-1} = 0$  Bsp:  $(\mathbb{Z}_5, +)$

## Parität

$b_n = (\sum_{i=1}^n b_i) \bmod 2$  aus Folge  $b_1, \dots, b_{n-1}$   
→ Codewort  $b_1, \dots, b_n$

$\sum_{i=1}^n b_i \equiv 0 \pmod{2}$  Parity-Check-Code (PCC)

Mächtigkeit aller PCCs der Länge  $n$ :  
 $|P_n| = 2^{n-1}$  [alle Bits bis auf das letzte können frei gewählt werden]

PCC: 1-fehlererkennend  
Rekonstruktion von fehlenden Bit mögl.

ISBN: 10-stellig: 3<sup>o</sup> Buchnr. + 1 Prüfziffer

Prüfziffer:  $z_{10} = (\sum_{i=1}^9 i \cdot z_i) \bmod 11$

bzw.  $0 = \sum_{i=1}^{10} i \cdot z_i \pmod{11}$

ISBN: 1-fehlererkennend, erkennt Zahlenfehler

## Fehlerkorrektur

(Hamming-) Abstand:  $d(u,v)$  → Anzahl von Stellen, an denen sich  $u$  &  $v$  unterscheiden

Minimalabstand von Code  $C$ :  
 $d(C) = \min \{d(u,v) \mid u,v \in C, u \neq v\}$

bei lin. Code:  $d(u,v) = \sum d(u_i, v_i) = d(u-v, 0)$   
 $d(C) = \min \{d(c, 0) \mid c \in C - \{0\}\}$

bei PCC:  $d(PCC) = 2$

$d(C) \geq k+1 \Rightarrow k$ -fehlererkennend

$d(C) \geq 2k+1 \Rightarrow k$ -fehlerkorrigierend

## Lineare Code

Parity-Check-Matrix

•  $C$  ist linear, wenn für Matrix  $A = (1 \dots 1)$  gilt:  $C = \{w \mid A \cdot w^T = 0\}$

• lin. Code  $C$  ist Vektorraum über  $\mathbb{Z}_2(+, \cdot)$

Matrix  $G$ , deren Zeilen eine Basis eines lin. Codes  $C$  bilden heißt Generatormatrix

Umrechnung: Nachricht  $\cdot G =$  Codewort  
 $A \cdot \text{Codewort}^T = 0$