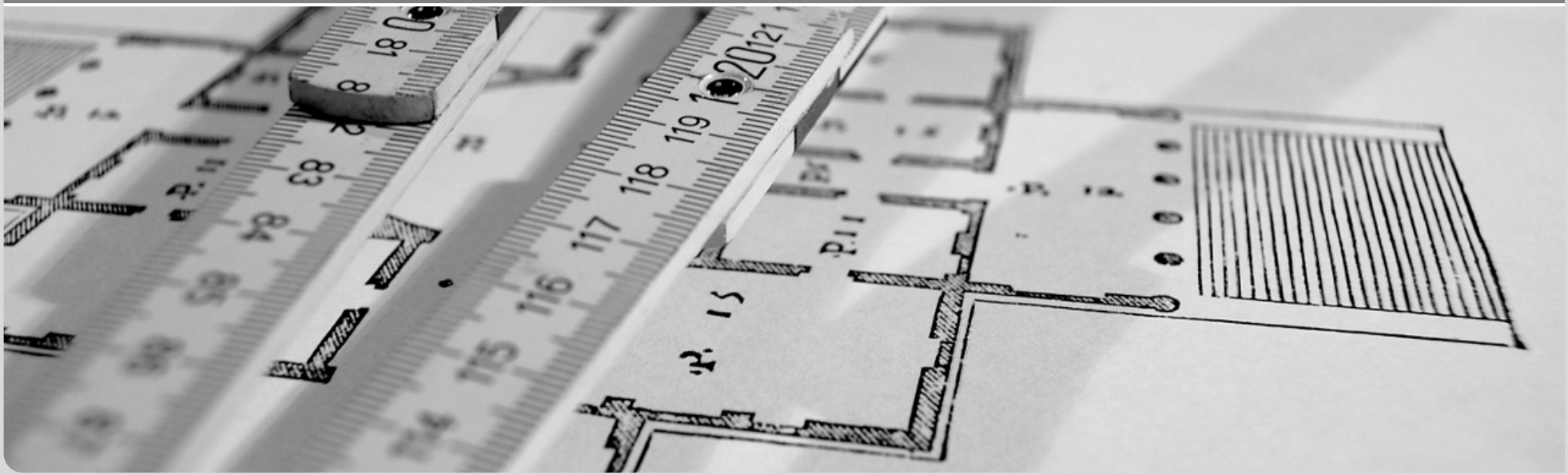
Grundbegriffe der Informatik
Tutorium 36

Termin 5 | 25.11.2016
Thassilo Helmold

KIT – Karlsruher Institut für Technologie



Inhalt

Zweierkomplement

Codierungen

Huffman-Codierung

In the previous episode of GBI...

Rückblick: Zahlendarstellung

- Zahlen sind Objekte, die einen festen numerischen Wert haben.
  Um eine Zahl aufzuschreiben, benötigen wir aber eine Darstellung.
  Die gleiche Zahl kann viele verschiedenen Darstellungen annehmen.
- Mit der Auswertungsfunktion $Num_b(\cdot)$ berechnen wir von einer Zahlendarstellung den numerischen Wert.
- Mit der Funktion $Repr_b(\cdot)$ können wir eine Zahl in einer beliebigen Darstellung angeben.

Bis jetzt haben wir das alles nur für positive Zahlen gesehen!

Wahr oder falsch?

- $\forall x \in \mathbb{Z} : x \textbf{ mod } 2 = 0 \iff x$ ist gerade W
  Und auch: $x \textbf{ mod } 2 = 1 \iff$ x ist ungerade
- $4 \cdot (x \textbf{ div } 4) = x$ F $4 \cdot (x \textbf{ div } 4) + (x \textbf{ mod } 4) = x$
- $(\{a\}^* \cdot \{b, \epsilon\})^* = \{a, b\}^*$ W
- Jede Abbildung besitzt eine Umkehrabbildung F Nur bijektive Abbildungen besitzen eine Umkehrabbildung
- Für jede Abbildung können wir das Urbild angeben W

Aufgabe: Zahlendarstellung

Berechnet die folgenden Zahlenwerte:

$$Num_2(1) = 1$$
$$Num_2(11) = 3$$
$$Num_2(111) = 7$$
$$Num_2(1111) = 15$$

Gibt es ein allgemeines Muster?
Ja, es gilt

$$Num_2(1^l) = 2^l - 1$$

und allgemein

$$Num_b((b-1)^l) = b^l - 1$$

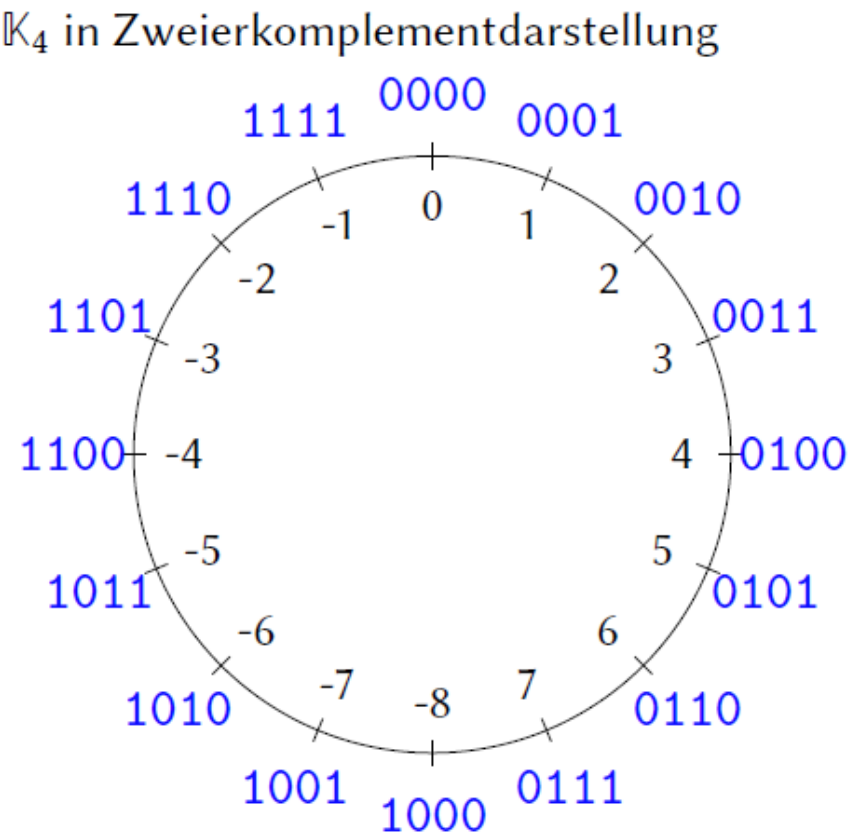
Zweierkomplement

Codierungen

Huffman-Codierung

Ein asymetrischer Zahlenbereich

$$\mathbb{K}_\ell = \{x \in \mathbb{Z} \mid -2^{\ell-1} \leqslant x \leqslant 2^{\ell-1} - 1\} .$$



Zweierkomplement

Das Zweierkomplement ist eine Möglichkeit, negative Zahlen binär darzustellen. Im Vergleich zu anderen Darstellungsarten ist es besonders vorteilhaft bei arithmetischen Rechnungen mit Hardware *(mehr dazu in Technischer Informatik)*.

Definition

$$Zkpl_l(x) = \begin{cases} 0bin_{l-1}(x) & \text{falls } x \geqslant 0 \\ 1bin_{l-1}(2^{l-1} + x) & \text{falls } x < 0 \end{cases}$$

Äquivalent:

$$Zkpl_l(x) = \begin{cases} bin_l(x) & \text{falls } x \geqslant 0 \\ bin_l(2^l + x) & \text{falls } x < 0 \end{cases}$$

ZK-Beispiel

Beispiele

$$Zkpl_5(0) = 00000$$
$$Zkpl_5(2) = 00010$$
$$Zkpl_5(15) = 01111$$
$$Zkpl_5(-1) = 11111$$
$$Zkpl_5(-6) = 11010$$
$$Zkpl_5(-16) = 10000$$

ZK: Einfache Berechnung

Zum „intuitiven“ Berechnen des Zweierkomplement können wir so vorgehen (für $x < 0$):

1. Binärdarstellung von $|x|$ berechnen
2. Mit führenden Nullen auffüllen bis zur Länge ℓ
3. Alle binären Ziffern negieren
4. 1 addieren

Beispiel

$$Zkpl_4(-2) : 2 \rightarrow 10 \rightarrow 0010 \rightarrow 1101 \rightarrow 1110$$

ZK: Einfache Berechnung

Die einzelnen Schritte können wir auch formal angeben:
(Wir operieren jeweils auf Wörtern aus $\{0,1\}^* = Z_2^*$)

1. Binärdarstellung von $|x|$ berechnen: $Repr_2(|\cdot|)$
2. Mit führenden Nullen auffüllen bis zur Länge ℓ

$$Fill_\ell : Z_2^m \to Z_2^\ell \qquad (m \leqslant \ell)$$

$$w \mapsto \begin{cases} 0^\ell & w = \varepsilon \\ Fill_{\ell-1}(w') \cdot \mu & w = w' \cdot \mu,\ w' \in Z_2^*,\ \mu \in Z_2 \end{cases}$$

oder deutlich einfacher

$$w \mapsto \begin{cases} w & |w| = \ell \\ Fill_\ell(0w) & \text{sonst} \end{cases}$$

3. / 4. Analog

Von einer Zahlendarstellung zur Anderen

Eigentlich recht intuitiv:

$$Trans_{3,5} = Repr_3 \circ Num_5$$

Aufgabe

Berechne folgende Darstellungen:
$Repr_2(42) = 101010_2$
$Trans_{4,2}(101010) = 222_4$
$Trans_{8,10}(42) = 52_8$
$Trans_{16,10}(42) = 2A_{16}$

Was ist bei allen Wörtern gleich? Die Bedeutung!
Was macht die Rechnungen 2 - 4 vergleichsweise einfach? Wir können die Struktur ausnutzen und zeichenweise vorgehen!
($Trans_{8,10}(42) = Trans_{8,2}(101 \cdot 010) = Trans_{8,2}(101) \cdot Trans_{8,2}(010)$)

Zweierkomplement

Codierungen

Huffman-Codierung

Übersetzung: Bedeutungserhaltende Abbildung

Codierung: Injektive Übersetzung
Es reicht eine injektive Abbildung: Dann können wir für jedes $f(w)$ eindeutig das erzeugende w angeben und somit die Bedeutung von $f(w)$ als die Bedeutung von w festlegen.

Beliebige Codierungen zu speichern ist sehr aufwendig, bei unendlichem Definitionsbereich sogar unmöglich.
Also bringen wir etwas Struktur ins Spiel!

Homomorphismen

Ein Homomorphismus ist eine strukturerhaltende Abbildung.

$$\Phi : A \to B$$
$$\text{mit } \forall a \in A, b \in B : \Phi(a \square b) = \Phi(a) \triangle \Phi(b)$$

Homomorphismen

In GBI: Homomorphismen auf Wörtern, dabei muss die Konkatenation erhalten werden.
A, B Alphabete, dann ist Abbildung $h : A^* \to B^*$ ein Homomorphismus, wenn

$$\forall\, x, y \in A^* : h(x \cdot y) = h(x) \cdot h(y)$$

Damit sind alle Funktionswerte durch die Bilder der einzelnen Zeichen aus A festgelegt. Das ist aber eine „endliche Tabelle“ (da A endlich ist) und lässt sich daher gut speichern und „handhaben“.

Beispiel

Sei h ein Homomorphismus mit $h(a) = 2, h(b) = 3$.
Dann gilt $h(aba) = h(a) \cdot h(b) \cdot h(a) = 232$

$Trans_{16,2}$ ist auch (fast) ein Homomorphismus (siehe dazu ÜB WS15/16)

Homomorphismen konstruieren

Wir können uns aber aus einer Abbildung der einzelnen Zeichen einen Homomorphismus auf Wörtern konstruieren.

Definition

Sei $f : A \to B^*$, definiere $f^{**} : A^* \to B^*$ als

$$f^{**}(\varepsilon) = \varepsilon$$
$$\forall w \in A^*, x \in A : f^{**}(wx) = f^{**}(w)f(x)$$

f^{**} ist der durch f **induzierte** Homomorphismus.

ε—Freiheit

Klar ist: $h(\varepsilon) = \varepsilon$

Definition

Ein Homomorphismus heißt ε—frei, wenn

$$\forall\, x \in A : h(x) \neq \varepsilon$$

Was ist das Problem mit Homomorphismen, die nicht ε—frei sind?
Es „geht Information verloren“.
Sei $h(c) = \varepsilon$, $h(b) = 2$, $w \in \{b, c\}^*$, $h(w) = 2$.
Wie kommen wir zu w?
Gar nicht! Wir wissen nicht, wie viele c in dem Wort w sind.

Präfixfreiheit

Definition

Ein Homomorphismus heißt *präfixfrei*, wenn für *keine* zwei verschiedenen Symbole $x_1, x_2 \in A$ gilt: $h(x_1)$ ist ein Präfix von $h(x_2)$.

Was ist das Problem mit Homomorphismen, die nicht präfixfrei sind?
Es „geht Information verloren“.
Sei $h(a) = 2, h(b) = 3, h(c) = 23, , w \in \{a, b, c\}^*, h(w) = 23$.
Wie kommen wir zu w?
Gar nicht! $w = c$ oder $w = ab$, wir wissen es nicht!

Zurück zu Codierungen

Beobachtung

Präfixfreie Homomorphismen sind ε—frei

Lemma

Präfixfreie Homomorphismen sind Codierungen (also injektiv).

Beobachtung

Präfixfreie Codes kann man „einfach“ decodieren:

$$u(w) = \begin{cases} \varepsilon, & \text{falls } w = \varepsilon \\ x\,u(w'), & \text{falls } w = h(x)w' \text{ für ein } x \in A \\ \bot, & \text{sonst} \end{cases}$$

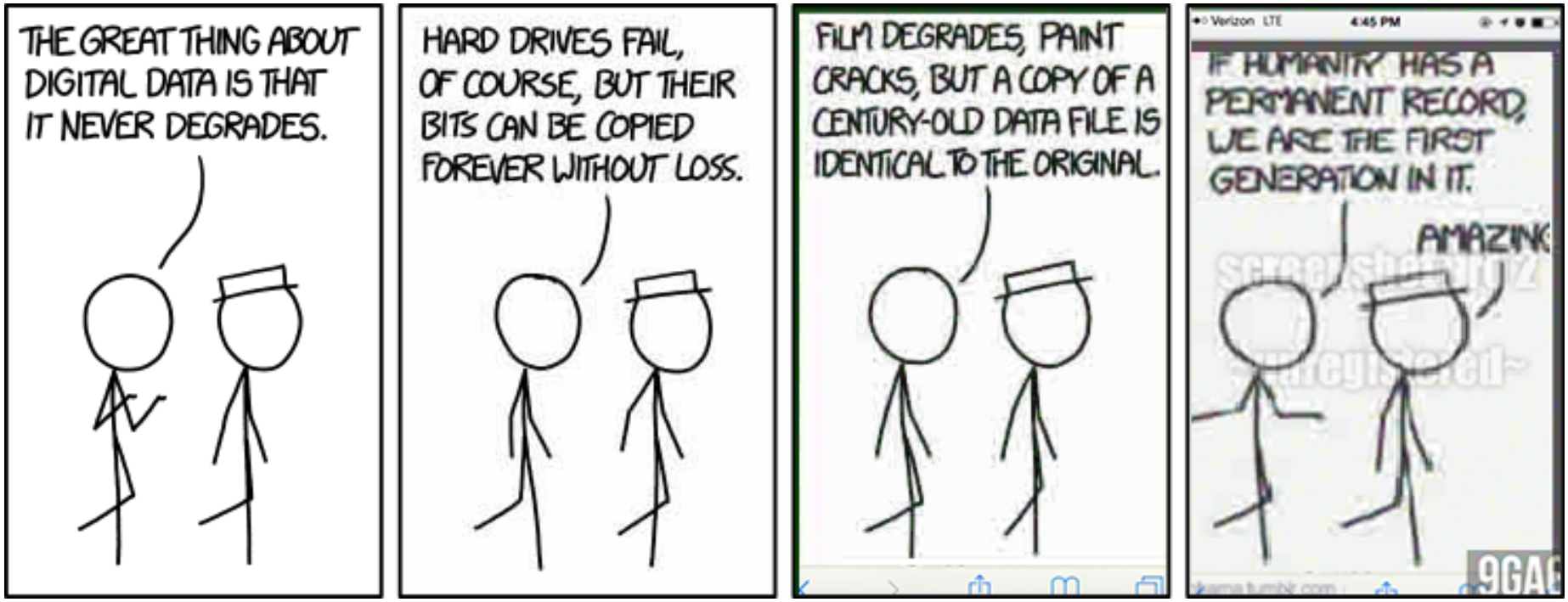


Abbildung: `https://www.xkcd.com/1683/`

Zweierkomplement

Codierungen

Huffman-Codierung

Kann man mit einer Codierung die benötigte Anzahl der Zeichen für ein Wort reduzieren und trotzdem den Sinn erhalten?

Natürlich geht das (manchmal), dieses Verfahren ist überall im Einsatz: Komprimierung!

Huffman-Codierung

Eine Huffman-Codierung ist ein präfixfreier (und demnach „einfach“ zu decodierenden) Homomorphismus, bei der die Codierung eines Zeichens umso länger wird, je seltener das Zeichen vorkommt.
Die Huffman-Codierung für ein Wort ist dabei nicht eindeutig, wie wir gleich im Konstruktionsverfahren sehen werden.

Lemma

Unter allen präfixfreien Codes führen Huffman-Codes zu kürzesten Codierungen **des Wortes, für das die Huffman-Codierung konstruiert wurde.**

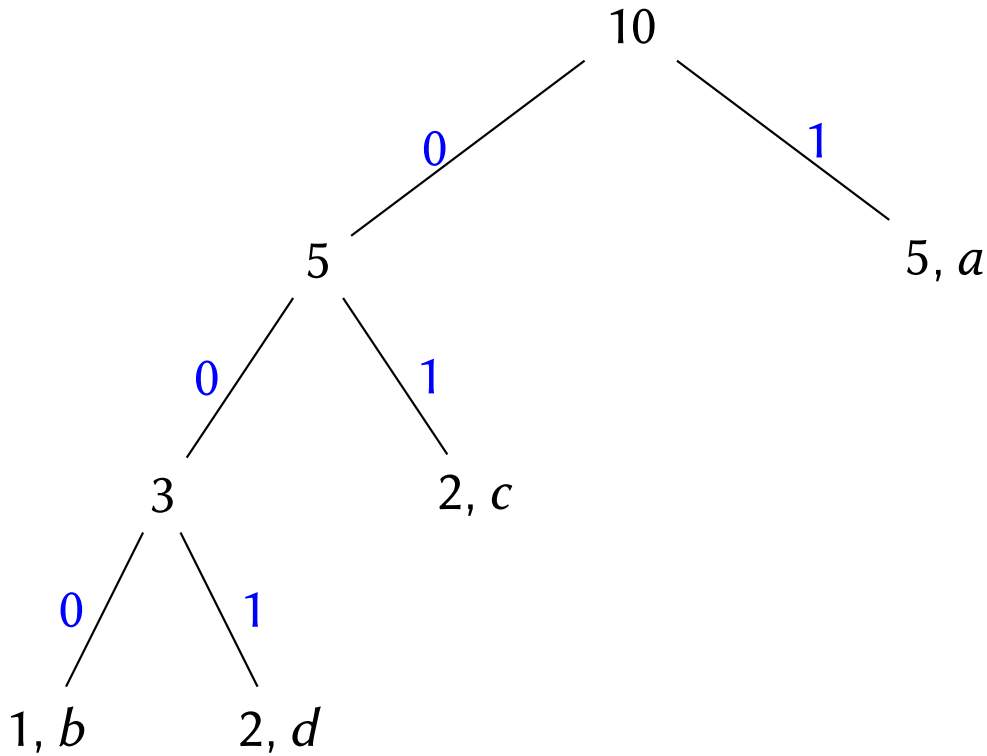
Konstruktionsverfahren

Formal: In der Vorlesung
Hier: Vorgehensweise (ausreichend!)

1. Für jedes Zeichen die Häufigkeit ermitteln
2. Alle Zeichen mit ihrer Häufigkeit als Blätter in die unterste Ebene zeichnen
3. Jeweils die zwei Knoten (nicht unbedingt Blätter!) mit den geringsten Häufigkeiten „verbinden“(also einen neuen Knoten darüber anlegen, der die Summe der Häufigkeiten erhält)
4. Fortfahren, bis der ganze Baum aufgebaut ist.
5. Die linken Äste mit 0 beschriften, die rechten Äste mit 1.
6. Codierungen der Zeichen ablesen
7. Ausgangswort codieren (wenn gefordert, vergesst das nicht!)

Beispiel

Gegeben : $w = abadcadaac$ (10 Zeichen)



x	a	b	c	d
#x	5	1	2	2
h(x)	1	000	01	001

Beispiel

Gegeben : $w = abadcadaac$ (10 Zeichen)

Wie lang wird das neue Wort ?
$5 * 1 + 1 * 3 + 2 * 2 + 2 * 3 = 18$ Zeichen
Aber wir wollen doch komprimieren? Was haben wir falsch gemacht?
Nichts. Zur Codierung eines der Zeichen a, b, c, d benötigen wir mindestens 2 Bit (da 4 Möglichkeiten). Für die Zeichen $0, 1$ brauchen wir aber nur ein Bit.
Also haben wir 18 Bit statt 20 Bit und somit komprimiert!

x	a	b	c	d
#x	5	1	2	2
h(x)	1	000	01	001

Huffman-Codierungen funktionieren immer nur gut für Wörter, die eine gleiche/ähnliche relative Zeichenhäufigkeit haben wie das Wort, für das der Code erstellt wurde.

Beispiel

$w_1 = badcfehg$, $w_2 = a^1 b^2 c^4 d^8 e^{16} f^{32} g^{64} h^{128}$ Erstellt eine Huffman-Codierung für jedes der beiden Wörter und Codiert jeweils beide Wörter mit der erstellten Codierung.
Wie verhalten sich die Längen der Codewörter?

Erweiterung

Wir können nicht nur einzelne Buchstaben codieren.
Bei $w = a^{10}b^{10}c^{10}$ lohnt es sich pro Block gleicher Buchstaben eine Codierung zu haben.

Aufgabe (WS 2008)

Das Wort

$$w = \mathbf{0000\ 0001\ 0011\ 0001\ 0011\ 0000\ 0000\ 1110\ 0001\ 0000}$$

soll komprimiert werden.

- Zerlegen Sie w in Viererblöcke und bestimmen Sie die Häufigkeiten der vorkommenden Blöcke.
- Zur Kompression soll ein Huffman-Code verwendet werden. Benutzen Sie die in Teilaufgabe a) bestimmten Häufigkeiten, um den entsprechenden Baum aufzustellen. Beschriften Sie alle Knoten und Kanten.
- Geben Sie die Codierung des Wortes w mit Ihrem Code an.

Lösung

$$w = \mathbf{00000001001100010011000000001110000010000}$$

Zerlegen Sie w in Viererblöcke und bestimmen Sie die Häufigkeiten der vorkommenden Blöcke.

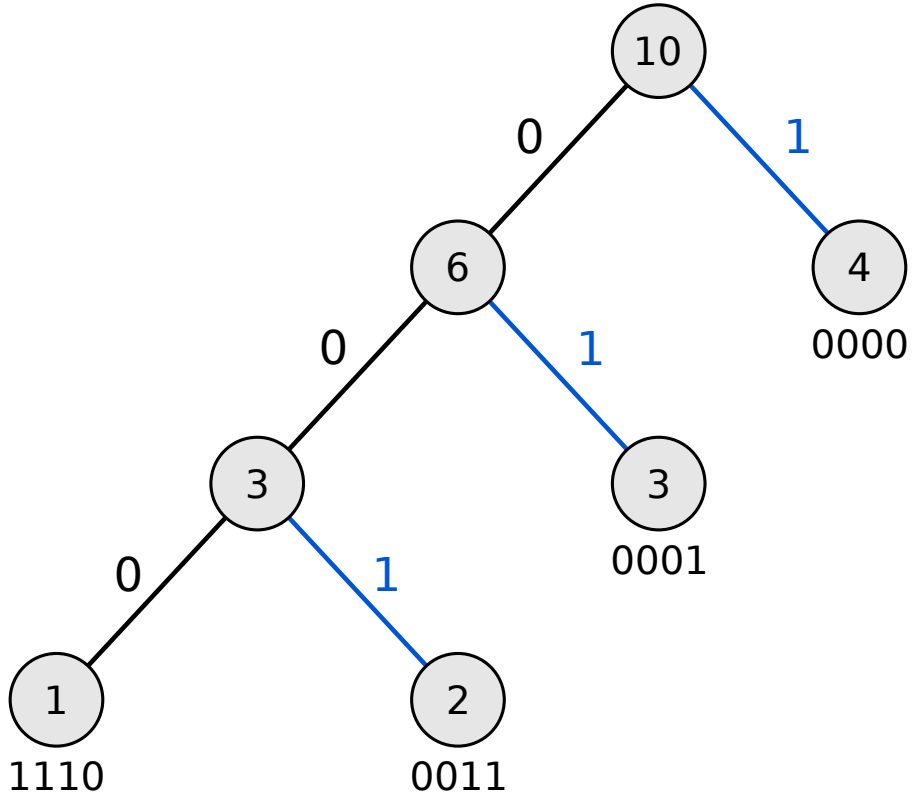
$$w = \mathbf{0000\ 0001\ 0011\ 0001\ 0011\ 0000\ 0000\ 1110\ 0001\ 0000}$$

	0000	0001	0011	1110
Absolute Häufigkeiten:	4	3	2	1
Relative Häufigkeiten:	0,4	0,3	0,2	0,1

Lösung

Zur Kompression soll ein Huffman-Code verwendet werden. Benutzen Sie die in Teilaufgabe a) bestimmten Häufigkeiten, um den entsprechenden Baum aufzustellen. Beschriften Sie alle Knoten und Kanten.

0000	0001	0011	1110
4	3	2	1



Lösung

Geben Sie die Codierung des Wortes w mit Ihrem Code an.

000000010011000100110000000011110000010000

$\rightarrow$ 1010010100111000011

Ausblick

Die Huffman-Codierung hat ein Problem: Zum Decodieren muss der Huffman-Baum, der für die Codierung verwendet wurde, bekannt sein. Im wesentlichen gibt es dafür zwei Möglichkeiten:

1. Der Codebaum wird vor dem eigentlichen Codewort angegeben.
   Problem: Das verlängert das Codewort.
2. Es wird ein vorher festgelegter Codebaum verwendet.
   Problem: Dieser Codebaum ist nicht an das spezifische Wort angepasst und kann evtl. (bei komplett anderer Zeichenhäufigkeit) zu sehr schlechten Ergebnissen führen.

Diese Probleme können durch andere Codierungsverfahren gelöst werden, indem z.B. das Wörterbuch dynamisch während der Decodierung aus dem Codewort aufgebaut wird (z.B. Lempel-Ziv-Welch-Verfahren).

Was ihr nun wissen solltet

- Wie man das Zweierkomplement bildet
- Übersetzungen und Codierungen
- Huffman-Codierung

Was nächstes Mal kommt

- Speicher - Damit wir nicht alles gleich wieder vergessen
- MIMA - Den Bits beim Arbeiten zuschauen

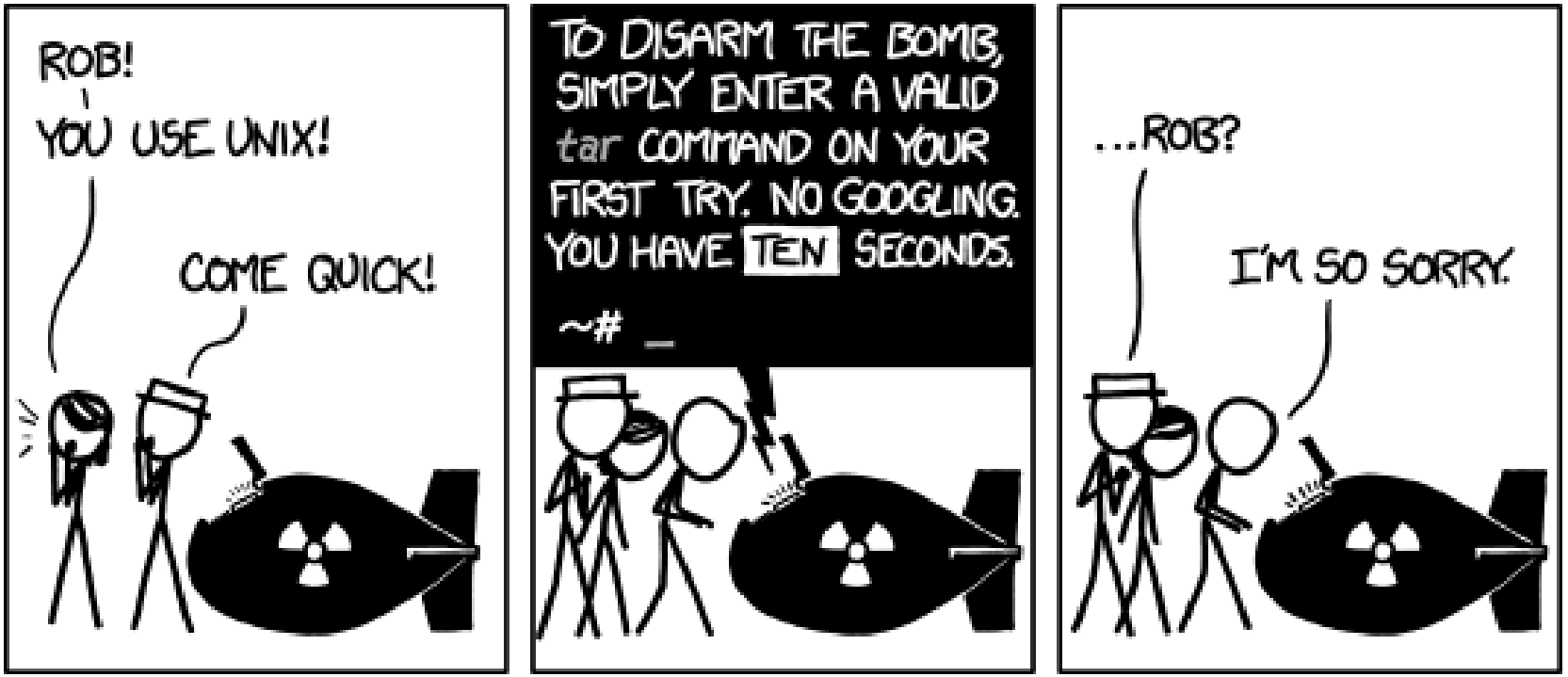


Abbildung: https://www.xkcd.com/1168/

Danksagung

Dieser Foliensatz basiert in Teilen auf Folien von:

Philipp Basler
Nils Braun
Dominik Doerner
Ou Yue