„*Ich hoffe, es werden Ew. Hochfürstliche Durchlaucht in Gnaden vermerken, daß ich sowohl dem Gebrauche, als meinem Gemüths=Triebe zu Folge, bei dem eingetretenen neuen Jahre, auf dieses und viele folgende, Denenselben in beständiger Gesundheit alle selbst verlangende hohe Fürstliche Ersprießlichkeit zu gemeinem und Dero Lande besondern Besten, aus treuem Herzen anwünsche.*“

schrieb in einem Brief datiert auf den 2. Januar 1697 an den Herzog von Braunschweig-Wolfenbüttel

„*Ich hoffe, es werden Ew. Hochfürstliche Durchlaucht in Gnaden vermerken, daß ich sowohl dem Gebrauche, als meinem Gemüths=Triebe zu Folge, bei dem eingetretenen neuen Jahre, auf dieses und viele folgende, Denenselben in beständiger Gesundheit alle selbst verlangende hohe Fürstliche Ersprießlichkeit zu gemeinem und Dero Lande besondern Besten, aus treuem Herzen anwünsche.*“

schrieb in einem Brief datiert auf den 2. Januar 1697 an den Herzog von Braunschweig-Wolfenbüttel

der Philosoph, Mathematiker, Physiker, Bibliothekar, . . .
Gottfried Wilhelm Leibniz (1646–1716)

Grundbegriffe der Informatik

Einheit 10: Codierungen

Thomas Worsch

Karlsruher Institut für Technologie, Fakultät für Informatik

Wintersemester 2009/2010

Dezimaldarstellung von Zahlen

- aus Indien
- Ziffern des Alphabetes $Z_{10} = \{0, \ldots, 9\}$.
- Bedeutung $\mathrm{num}_{10}(x)$ einer einzelnen Ziffer x als Zahl:

x	0	1	2	3	4	5	6	7	8	9
$\mathrm{num}_{10}(x)$	0	1	2	3	4	5	6	7	8	9

- Bedeutung eines Wortes $x_{k-1} \cdots x_0 \in Z_{10}^*$ von Ziffern

$$\mathrm{Num}_{10} : Z_{10}^* \to \mathbb{N}_0$$

- Schule:

$$10^{k-1} \cdot \mathrm{num}_{10}(x_{k-1}) + \cdots + 10^1 \cdot \mathrm{num}_{10}(x_1) + 10^0 \cdot \mathrm{num}_{10}(x_0)$$

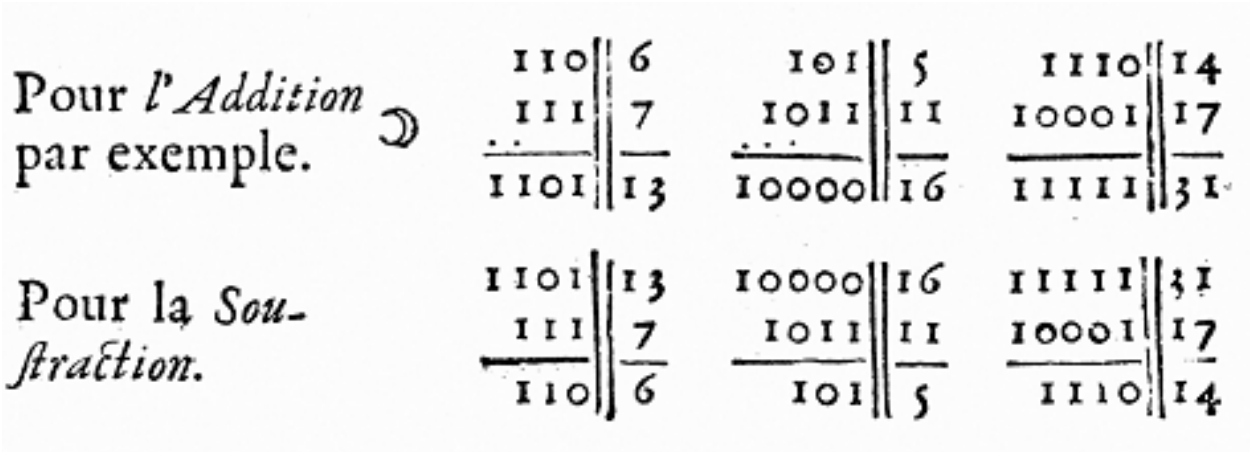
- Pünktchenvermeidung:

$$\mathrm{Num}_{10}(\varepsilon) = 0$$

$$\forall w \in Z_{10}^* \; \forall x \in Z_{10} : \mathrm{Num}_{10}(wx) = 10 \cdot \mathrm{Num}_{10}(w) + \mathrm{num}_{10}(x)$$

Gottfried Wilhelm Leibniz

- geboren 1. Juli 1646 (Leipzig), gestorben am 14. November 1716 (Hannover)
- baute zum Beispiel die erste Maschine, die zwei Zahlen multipliziern konnte
- in einem Brief vom 2. Januar 1697 an den Herzog von Braunschweig-Wolfenbüttel:
  - Man kann auch alle nichtnegativen Zahlen darstellen, und vernünftig rechnen, wenn man nur zwei Ziffern 0 und 1 benutzen darf.



Bildquelle `http://commons.wikimedia.org/wiki/Image:Leibniz_binary_system_1703.png`

Binärdarstellung von Zahlen

- Ziffernmenge $Z_2 = \{0, 1\}$
- definiere:

$$\mathrm{num}_2(0) = 0$$
$$\mathrm{num}_2(1) = 1$$
$$\mathrm{Num}_2(\varepsilon) = 0$$
$$\forall w \in Z_2^* \; \forall x \in Z_2 : \mathrm{Num}_2(wx) = 2 \cdot \mathrm{Num}_2(w) + \mathrm{num}_2(x)$$

- Z. B.:

$$\begin{aligned}
\mathrm{Num}_2(1101) &= 2 \cdot \mathrm{Num}_2(110) + 1 \\
&= 2 \cdot (2 \cdot \mathrm{Num}_2(11) + 0) + 1 \\
&= 2 \cdot (2 \cdot (2 \cdot \mathrm{Num}_2(1) + 1) + 0) + 1 \\
&= 2 \cdot (2 \cdot (2 \cdot 1 + 1) + 0) + 1 \\
&= 13
\end{aligned}$$

Hexadezimaldarstellung

- Ziffern $Z_{16} = \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}, \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}\}$.
- Manchmal Kleinbuchstaben statt Großbuchstaben
- Ziffernwerte:

x	0	1	2	3	4	5	6	7
$\mathrm{num}_{16}(x)$	0	1	2	3	4	5	6	7
x	8	9	A	B	C	D	E	F
$\mathrm{num}_{16}(x)$	8	9	10	11	12	13	14	15

- Zuordnung von Wörtern zu Zahlen gegeben durch

$$\mathrm{Num}_{16}(\varepsilon) = 0$$

$$\forall w \in Z_{16}^* \; \forall x \in Z_{16} : \mathrm{Num}_{16}(wx) = 16 \cdot \mathrm{Num}_{16}(w) + \mathrm{num}_{16}(x)$$

ein kleines Problem

- die Alphabete Z_2, Z_3, usw. sind nicht disjunkt
- Darstellungen mehrdeutig; z. B. ist `111`

  $\mathrm{Num}_2(\texttt{111})$ die Zahl sieben,
  $\mathrm{Num}_8(\texttt{111})$ die Zahl dreiundsiebzig,
  $\mathrm{Num}_{10}(\texttt{111})$ die Zahl einhundertelf und
  $\mathrm{Num}_{16}(\texttt{111})$ die Zahl zweihundertdreiundsiebzig.
- in manchen Programmiersprachen
  - Präfix `0b` für Darstellungen zur Basis 2
  - Präfix `0o` für Darstellungen zur Basis 8
  - Präfix `0x` für Darstellungen zur Basis 16

- Man kann zu $w \in Z_k^*$ die repräsentierte Zahl $\mathrm{Num}_k(w)$ berechnen.
- Umgekehrt geht es auch: zu $n \in \mathbb{N}_0$ die sogenannte k-äre Darstellung berechnen
- Alphabet Z_k mit k Ziffern
  - Bedeutung: die Zahlen in $\mathbb{G}_k$
  - für $i \in \mathbb{G}_k$ sei $\mathrm{repr}_k(i)$ das entsprechende Zeichen
  - repr_k ist also gerade die Umkehrfunktion zu num_k
- Gesucht: eine Repräsentation von $n \in \mathbb{N}_0$ als Wort $w \in Z_k^*$ mit der Eigenschaft $\mathrm{Num}_k(w) = n$
  - für die naheliegende Definition von Num_k
- gleich: solche Wörter gibt es immer
- Und wenn, dann auch immer gleich unendlich viele: wenn $\mathrm{Num}_k(w) = n$, dann auch $\mathrm{Num}_k(0w) = n$

Algorithmus für Berechnung der k-ären Darstellung einer Zahl

// Eingabe: $n \in \mathbb{N}_0$

$y \leftarrow n$

$w \leftarrow \varepsilon$

$$m \leftarrow \begin{cases} 1 + \lfloor \log_k(n) \rfloor & \text{falls } n > 0 \\ 1 & \text{falls } n = 0 \end{cases}$$

for $i \leftarrow 0$ **to** $m - 1$ **do**

$r \leftarrow y$ **mod** k

$w \leftarrow \mathrm{repr}_k(r) \cdot w$

$y \leftarrow y$ **div** k

od

// am Ende: $n = \mathrm{Num}_k(w)$

Beispielberechnung

- Nehme $k = 10$ und $n = 4711$
- Dann ist $m = 4$
- für jedes $i \in \mathbb{G}_5$ haben die Variablen r, w und y nach i Schleifendurchläufen die folgenden Werte

i	4	3	2	1	0
r	4	7	1	1	
w	4711	711	11	1	ε
y	0	4	47	471	4711

- Schleifeninvariante $y \cdot 10^i + \mathrm{Num}_k(w) = n$ drängt sich auf
- Wenn man weiß, dass am Ende $y = 0$ ist, ist man fertig.

Bemerkung

- Algorithmus liefert kürzestes Wort $w \in Z_k^*$ mit $\mathrm{Num}_k(w) = n$.
- schreiben dafür $\mathrm{Repr}_k(n)$.
- Es ist stets

$$\mathrm{Num}_k(\mathrm{Repr}_k(n)) = n\ .$$

- Beachte:
  - umgekehrt im allgemeinen

$$\mathrm{Repr}_k(\mathrm{Num}_k(w)) \neq w$$

  - weil „überflüssige“ führende Nullen wegfallen.

Das ist wichtig

Das sollten Sie mitnehmen:

- Umwandlungen zwischen Zahlen und Wörtern
- mal wieder Schleifen
  - in passenden Beispielen ist Schleifeninvariante gut zu sehen
- schon Leibniz kannte die Binärdarstellung

Das sollten Sie üben:

- selber Zahlen verschieden repräsentieren
- Algorithmen auch in Randfällen ausprobieren

Von Hexadezimal- zu Binärdarstellung

- Betrachte die Funktion $\mathrm{Trans}_{2,16} = \mathrm{Repr}_2 \circ \mathrm{Num}_{16}$
- $\mathrm{Trans}_{2,16} : Z_{16}^* \to Z_2^*$
- Z. B.

$$\begin{aligned}\mathrm{Trans}_{2,16}(\texttt{A3}) &= \mathrm{Repr}_2(\mathrm{Num}_{16}(\texttt{A3})) \\ &= \mathrm{Repr}_2(163) = \texttt{10100011}\end{aligned}$$

- wesentlicher Punkt:
  - `A3` und `10100011` haben *die gleiche Bedeutung*
  - nämlich die Zahl einhundertdreiundsechzig
- So etwas wollen wir eine Übersetzung nennen.

Übersetzungen

- Allgemein: Man schreibt Wörter aus einer formalen Sprache $L_A \subseteq A^*$ und meint aber etwas anderes, ihre Bedeutung.
- Menge der Bedeutungen je nach Anwendungsfall sehr unterschiedlich
  - einfach: Zahlen,
  - kompliziert: Ausführung von Java-Programmen
  - im Folgenden schreiben wir einfach Sem für die Menge der „Bedeutungen“
- Gegeben:
  - zwei Alphabete A und B
  - $L_A \subseteq A^*$ und $L_B \subseteq B^*$
  - zwei Abbildungen $\mathrm{sem}_A : L_A \to \mathrm{Sem}$ und $\mathrm{sem}_B : L_B \to \mathrm{Sem}$
- Eine Abbildung $f : L_A \to L_B$ heiße eine Übersetzung bezüglich sem_A und sem_B, wenn f die Bedeutung erhält, d. h.

$$\forall w \in L_A : \mathrm{sem}_A(w) = \mathrm{sem}_B(f(w))$$

Übersetzungen

- Allgemein: Man schreibt Wörter aus einer formalen Sprache $L_A \subseteq A^*$ und meint aber etwas anderes, ihre Bedeutung.
- Menge der Bedeutungen je nach Anwendungsfall sehr unterschiedlich
  - einfach: Zahlen,
  - kompliziert: Ausführung von Java-Programmen
  - im Folgenden schreiben wir einfach Sem für die Menge der „Bedeutungen“
- Gegeben:
  - zwei Alphabete A und B
  - $L_A \subseteq A^*$ und $L_B \subseteq B^*$
  - zwei Abbildungen $\mathrm{sem}_A : L_A \to \mathrm{Sem}$ und $\mathrm{sem}_B : L_B \to \mathrm{Sem}$
- Eine Abbildung $f : L_A \to L_B$ heiße eine Übersetzung bezüglich sem_A und sem_B, wenn f die Bedeutung erhält, d. h.

$$\forall w \in L_A : \mathrm{sem}_A(w) = \mathrm{sem}_B(f(w))$$

Beispiel $\mathrm{Trans}_{2,16}$

- $\mathrm{Trans}_{2,16} = \mathrm{Repr}_2 \circ \mathrm{Num}_{16}$.
- einfacher Fall: $L_A = A^* = Z_2^*$ und $L_B = B^* = Z_{16}^*$.
- Bedeutungsfunktionen: $\mathrm{sem}_A = \mathrm{Num}_{16}$ und $\mathrm{sem}_B = \mathrm{Num}_2$
- Nachrechnen, dass $\mathrm{Trans}_{2,16}$ eine Übersetzung ist:

$$\begin{aligned}\mathrm{sem}_B(f(w)) &= \mathrm{Num}_2(\mathrm{Trans}_{2,16}(w)) \\ &= \mathrm{Num}_2(\mathrm{Repr}_2(\mathrm{Num}_{16}(w))) \\ &= \mathrm{Num}_{16}(w) \\ &= \mathrm{sem}_A(w)\end{aligned}$$

Komplizierte Menge Sem

- zum Beispiel bei der Übersetzung von Programmiersprachen
- Was ist Sem?
  - viele Möglichkeiten
  - siehe Vorlesungen zu „Semantik von Programmiersprachen“.
- Andeutung: Man kann z. B. (es gitb auch anderes) als Semantik von

$$x \leftarrow 5$$

die Abbildung definieren, die jeder Speicherbelegung m die neue Speicherbelegung $\mathrm{memwrite}(m, x, 5)$ zuordnet.

Wozu Übersetzungen

- Legalität: Manchmal ist ein bestimmter Zeichensatz vorgeschrieben (z. B. ASCII in Emails)
- Lesbarkeit: Übersetzungen können zu kürzer und daher besser lesbar sein:
  vergleiche `A3` mit `10100011`
- Kompression: Übersetzungen können kürzer sein
  - und zwar *ohne* größeres Alphabet
  - gleich: Huffman-Codes
- Verschlüsselung: nicht verbesserte Lesbarkeit, sondern
  - am liebsten gar keine Lesbarkeit, jedenfalls für Außenstehende
  - siehe Vorlesungen über Kryptographie
- Fehlererkennung und Fehlerkorrektur:
  - man mache Texte durch Übersetzung so länger,
  - dass man selbst dann, wenn ein korrekter Funktionswert $f(w)$ „zufällig“ „kaputt“ (Übertragungsfehler) gemacht wird
  - man u. U. die Fehler korrigieren kann, oder zumindest erkennt, dass Fehler passiert sind.
  - typisches Beispiel: lineare Codes

Codierungen bzw. Codes

- Forderung $\mathrm{sem}_A(w) = \mathrm{sem}_B(f(w))$ in Spezialfällen kein Problem:
- z. B. bei Verschlüsselung und manchen Anwendungen von Kompression:
  - man will vom Übersetzten $f(x)$ eindeutig zurückkommen können möchte zum ursprünglichen x.
  - f ist injektiv.
- Dann kann man die Bedeutung sem_B im wesentlichen *definieren* durch die Festlegung $\mathrm{sem}_B(f(x)) = \mathrm{sem}_A(x)$.
  - Beachte: Dafür ist Injektivität von f wichtig!
- Wenn f injektiv ist, heiße eine Übersetzung auch Codierung.
- Die $f(w)$ heißen Codewörter.
- Die Menge $\{f(w) \mid w \in L_A\}$ heißt dann auch ein Code.

Wie spezifiziert man eine Übersetzung?

- Wenn L_A unendlich ist, kann man nicht alle Möglichkeiten aufzählen ...
- Auswege:
  - Homomorphismen
  - Block-Codierungen

Homomorphismen

- Gegeben:
  - zwei Alphabete A und B und
  - eine Abbildung $h : A \to B^*$.
- Zu h definiert man Funktion $h^{**} : A^* \to B^*$ vermöge

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

- Beispiel:
  - $h(\mathtt{a}) = \mathtt{10}$ und $h(\mathtt{b}) = \mathtt{001}$
  - dann ist

$$h^{**}(\mathtt{babb}) = h(\mathtt{b})\cdot h(\mathtt{a})\cdot h(\mathtt{b})\cdot h(\mathtt{b}) = \mathtt{001}\cdot\mathtt{10}\cdot\mathtt{001}\cdot\mathtt{001} = \mathtt{00110001001}$$

- Eine solche Abbildung h^{**} nennt man einen Homomorphismus.
- Häufig schreibt man statt h^{**} einfach wieder h.
- Homomorphismus heißt ε-frei, wenn $\forall x \in A : h(x) \neq \varepsilon$.

Präfixfreie Codes

- Gegeben: Abbildung $h : A \to B^*$
- Frage: ist Abbildung $h^{**} : A^* \to B^*$ eine Codierung, also injektiv?
- im allgemeinen nicht ganz einfach zu sehen.
- einfacher Spezialfall: h ist präfixfrei
- Das bedeutet: für *keine zwei verschiedenen* Symbole $x_1, x_2 \in A$ gilt: $h(x_1)$ ist ein Präfix von $h(x_2)$.

Präfixfreie Codes: Decodierung

- allgemeines Problem: nicht alle Wörter aus B^* sind Codewort, d. h. h^{**} ist im allgemeinen nicht surjektiv
- damit Decodierung trotzdem totale Abbildung u sein kann, definiere $u : B^* \to (A \cup \{\bot\})^*$.
  - wenn ein $w \in B^*$ gar nicht decodiert werden kann, dann soll in $u(w)$ das Symbol $\bot$ vorkommen

Beispiel

- Homomorphismus $h : \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \to \{\mathtt{0}, \mathtt{1}\}^*$ mit $h(\mathtt{a}) = \mathtt{1}$, $h(\mathtt{b}) = \mathtt{01}$ und $h(\mathtt{c}) = \mathtt{001}$.
  - Dieser Homomorphismus ist präfixfrei.
- dann $u : \{\mathtt{0}, \mathtt{1}\}^* \to \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \bot\}^*$
- z. B. soll gelten
  - $u(\varepsilon) = \varepsilon$
  - $u(\mathtt{001}) = \mathtt{c}$
  - $u(\mathtt{0101}) = \mathtt{bb}$
  - $u(\mathtt{0}) = \bot$ o. ä.

Präfixfreie Codes: Decodierung (2)

- wir schreiben mal hin (und diskutieren das anschließend):

$$u(w) = \begin{cases} \varepsilon & \text{falls } w = \varepsilon \\ \mathtt{a}u(w') & \text{falls } w = \mathtt{1}w' \\ \mathtt{b}u(w') & \text{falls } w = \mathtt{01}w' \\ \mathtt{c}u(w') & \text{falls } w = \mathtt{001}w' \\ \bot & \text{sonst} \end{cases}$$

- sei $w = \mathtt{100101} = h(\mathtt{acb})$; wir rechnen:

$$\begin{aligned} u(\mathtt{100101}) &= \mathtt{a}u(\mathtt{00101}) \\ &= \mathtt{ac}u(\mathtt{01}) \\ &= \mathtt{acb}u(\varepsilon) \\ &= \mathtt{acb} \end{aligned}$$

Präfixfreie Codes: Decodierung (3)

- $u(w) = \begin{cases} \varepsilon & \text{falls } w = \varepsilon \\ \mathtt{a}u(w') & \text{falls } w = \mathtt{1}w' \\ \mathtt{b}u(w') & \text{falls } w = \mathtt{01}w' \\ \mathtt{c}u(w') & \text{falls } w = \mathtt{001}w' \\ \bot & \text{sonst} \end{cases}$
- $u(\mathtt{100101}) = \mathtt{a}u(\mathtt{00101}) = \mathtt{ac}u(\mathtt{01}) = \mathtt{acb}u(\varepsilon) = \mathtt{acb}$
- Warum hat das geklappt?
- In jedem Schritt war klar, welche Zeile der Definition von u anzuwenden war.
- denn: w kann nicht gleichzeitig mit Codierungen verschiedener Symbole anfangen
- denn: kein $h(x)$ ist Präfix eines $h(y)$ für *verschiedene* $x, y \in A$
- denn: h ist präfixfrei

Präfixfreie Codes: Decodierung (3)

- $u(w) = \begin{cases} \varepsilon & \text{falls } w = \varepsilon \\ \mathtt{a}u(w') & \text{falls } w = \mathtt{1}w' \\ \mathtt{b}u(w') & \text{falls } w = \mathtt{01}w' \\ \mathtt{c}u(w') & \text{falls } w = \mathtt{001}w' \\ \bot & \text{sonst} \end{cases}$
- $u(\mathtt{100101}) = \mathtt{a}u(\mathtt{00101}) = \mathtt{ac}u(\mathtt{01}) = \mathtt{acb}u(\varepsilon) = \mathtt{acb}$
- Warum hat das geklappt?
- In jedem Schritt war klar, welche Zeile der Definition von u anzuwenden war.
- denn: w kann nicht gleichzeitig mit Codierungen verschiedener Symbole anfangen
- denn: kein $h(x)$ ist Präfix eines $h(y)$ für *verschiedene* $x, y \in A$
- denn: h ist präfixfrei

Präfixfreie Codes: Decodierung (3)

- $u(w) = \begin{cases} \varepsilon & \text{falls } w = \varepsilon \\ \mathtt{a}u(w') & \text{falls } w = \mathtt{1}w' \\ \mathtt{b}u(w') & \text{falls } w = \mathtt{01}w' \\ \mathtt{c}u(w') & \text{falls } w = \mathtt{001}w' \\ \bot & \text{sonst} \end{cases}$
- $u(\mathtt{100101}) = \mathtt{a}u(\mathtt{00101}) = \mathtt{ac}u(\mathtt{01}) = \mathtt{acb}u(\varepsilon) = \mathtt{acb}$
- Warum hat das geklappt?
- In jedem Schritt war klar, welche Zeile der Definition von u anzuwenden war.
- denn: w kann nicht gleichzeitig mit Codierungen verschiedener Symbole anfangen
- denn: kein $h(x)$ ist Präfix eines $h(y)$ für *verschiedene* $x, y \in A$
- denn: h ist präfixfrei

Präfixfreie Codes: Decodierung (4)

- Man spricht hier von Wohldefiniertheit
- immer dann ein Problem,
  wenn ein Funktionswert potenziell „auf mehreren Wegen“ festgelegt wird. Dann klar machen
  - entweder, dass in Wirklichkeit nur ein Weg „gangbar“ ist,
  - oder dass auf den verschiedenen Wegen am Ende der gleiche Funktionswert herauskommt.
  - siehe auch Kapitel über Äquivalenz- und Kongruenzrelationen
- einen präfixfreien Code kann man also so decodieren:

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

- Beachte: das heißt nur, dass man präfixfreie Codes „so einfach“ decodieren kann.

UTF-8 Codierung: ein Homomorphismus

- Die Codierung einzelner Zeichen wird gleich definiert
- Wörter werden zeichenweise codiert.
- werden gleich merken: UTF-8 ist präfixfrei

Auszug aus RFC 3629

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
0000 0000 - 0000 007F	0*xxxxxxx*
0000 0080 - 0000 07FF	110*xxxxx* 10*xxxxxx*
0000 0800 - 0000 FFFF	1110*xxxx* 10*xxxxxx* 10*xxxxxx*
0001 0000 - 0010 FFFF	11110*xxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

- Determine the number of octets required ...
- Prepare the high-order bits of the octets ...
- Fill in the bits marked x ...
  - Start by putting the lowest-order bit of the character number in the lowest-order position of the last octet of the sequence,
  - then put the next higher-order bit of the character number in the next higher-order position of that octet, ...
  - When the x bits of the last octet are filled in, move on to the next to last octet, then to the preceding one, etc. ...

UTF-8: ein Beispiel

- Integralzeichen $\int$ hat Unicode Codepoint `0x222B`
- `0x222B` in Bits `0010 0010 0010 1011`
- benutze also die Zeile

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
`0000 0800 - 0000 FFFF`	`1110`*xxxx* `10`*xxxxxx* `10`*xxxxxx*

- also `0010 0010 0010 1011` = `0010 001000 101011`
- also UTF-8 Codierung `11100010 10001000 10101011`

- Man überlege sich: UTF-8 ist präfixfrei

Das ist wichtig

Das sollten Sie mitnehmen:

- Übersetzungen sind in verschiedenen Situationen nützlich
- Homomorphismen
- Codes
- UTF-8

Das sollten Sie üben:

- Homomorphismen anwenden
- Zeichen in UTF-8 codieren

Huffman-Codierung

- Gegeben: ein Alphabet A und ein Wort $w \in A^*$
- Eine sogenannte Huffman-Codierung von w ist
  - eine Abbildung $h : A^* \to Z_2^*$,
  - die ein ε-freier Homomorphismus ist.
  - h also eindeutig festgelegt durch die $h(x)$ für alle $x \in A$.
- Huffman-Code: häufigere Symbole durch kürzere Wörter codieren und seltenere Symbole durch längere
- Verwendung z. B. in Kompressionsverfahren wie gzip oder bzip2

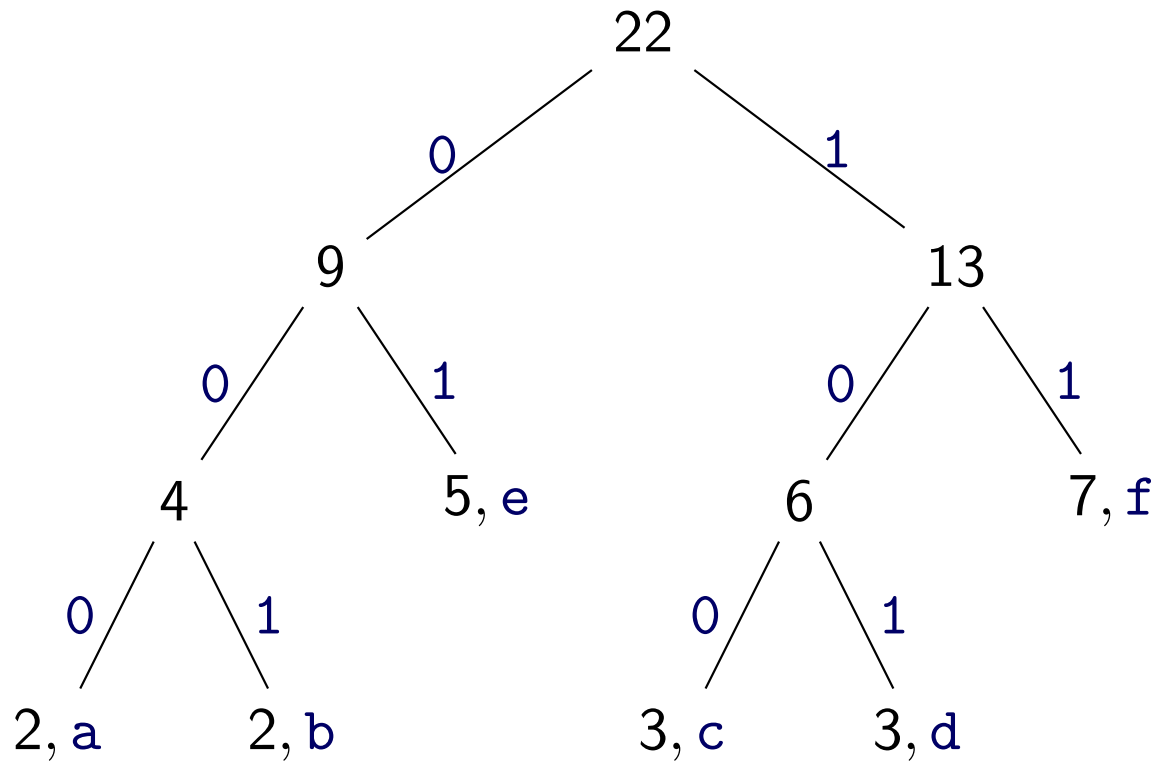
- Im folgenden
  - erst der Algorithmus
  - anschließend einige Bemerkungen

Voraussetzungen

- Gegeben
  - ein $w \in A^*$ und
  - die Anzahlen $N_x(w)$ aller Symbole $x \in A$ in w
  - o. B. d. A. alle $N_x(w) > 0$ (den Rest muss man nicht codieren)
- Algorithmus zur Bestimmung eines Huffman-Codes arbeitet in zwei Phasen:
  1. es wird ein Baum konstruiert
     - dessen Blätter den $x \in A$ entsprechen und
     - dessen Kanten mit 0 und 1 beschriftet sind
  2. Ablesen der Codes aus dem Baum (Pfadbeschriftungen)

Algorithmus für Huffman-Codes

- Beispiel: $w =$ `afebfecaffdeddccefbeff`
- Baum am Ende:



- Homomorphismus (präfixfrei!)

x	a	b	c	d	e	f
h(x)	000	001	100	101	01	11

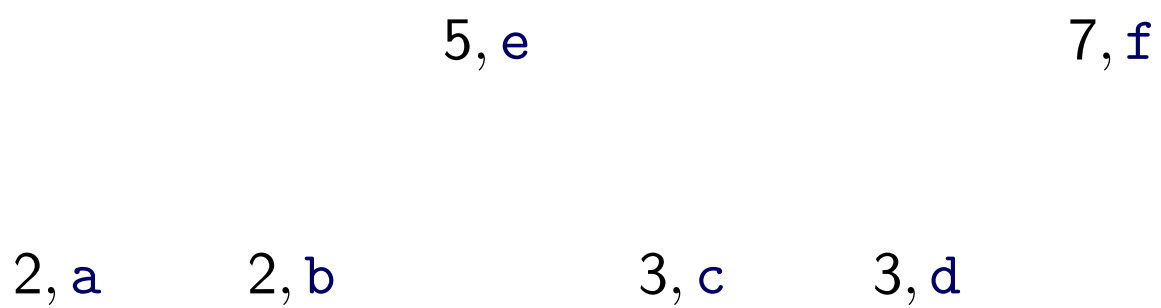
Konstruktion des Huffman-Baumes (1)

- Zu jedem Zeitpunkt hat man
  - eine Menge M_i von „noch zu betrachtenden Symbolmengen mit ihren Häufigkeiten“
  - und eine ebenso große Menge von schon konstruierten Teilbäumen
- Initialisierung:
  - M_0 ist die Menge aller $\{(x, N_x(w))\}$ für $x \in A$,
  - Als Anfang für die Konstruktion des Baumes zeichnet man für jedes Symbol einen Knoten mit Markierung $(x, N_x(w))$.
- Beispiel

$$M_0 = \{\ (2, \{\texttt{a}\})\ ,\ (2, \{\texttt{b}\})\ ,\ (3, \{\texttt{c}\})\ ,\ (3, \{\texttt{d}\})\ ,\ (5, \{\texttt{e}\})\ ,\ (7, \{\texttt{f}\})\ \}$$

Konstruktion des Huffman-Baumes (2)

Anfang im Beispiel:



Konstruktion des Huffman-Baumes (3)

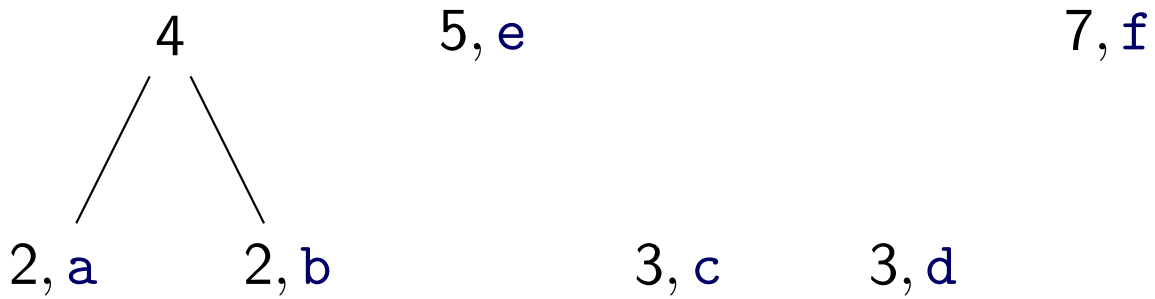
Iterationsschritt des Algorithmus:

- Solange Menge M_i noch mindestens zwei Paare enthält, Bestimme Menge M_{i+1} wie folgt:
  - Wähle zwei Paare (k_1, X_1) und (k_2, X_2), deren Häufigkeiten zu den kleinsten noch vorkommenden gehören.
  - Entferne diese Paare aus M_i und fügt statt dessen das eine Paar $(k_1 + k_2, X_1 \cup X_2)$ hinzu. Das ergibt M_{i+1}.
- im Graphen
  - Füge einen weiteren Knoten hinzu,
  - markiert mit der Häufigkeit $k_1 + k_2$ und Kanten zu den Knoten, die für (k_1, X_1) und (k_2, X_2) eingefügt worden waren.

Konstruktion des Huffman-Baumes (4)

Beispiel:

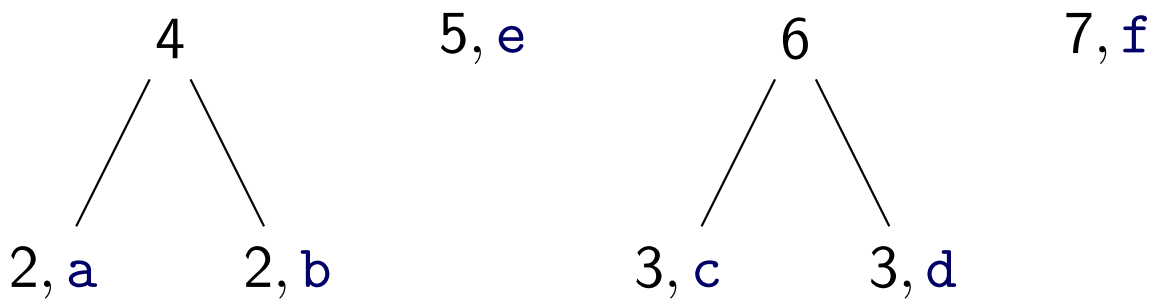
$$M_1 = \{\ (4, \{\texttt{a}, \texttt{b}\})\ ,\ (3, \{\texttt{c}\})\ ,\ (3, \{\texttt{d}\})\ ,\ (5, \{\texttt{e}\})\ ,\ (7, \{\texttt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (5)

Beispiel:

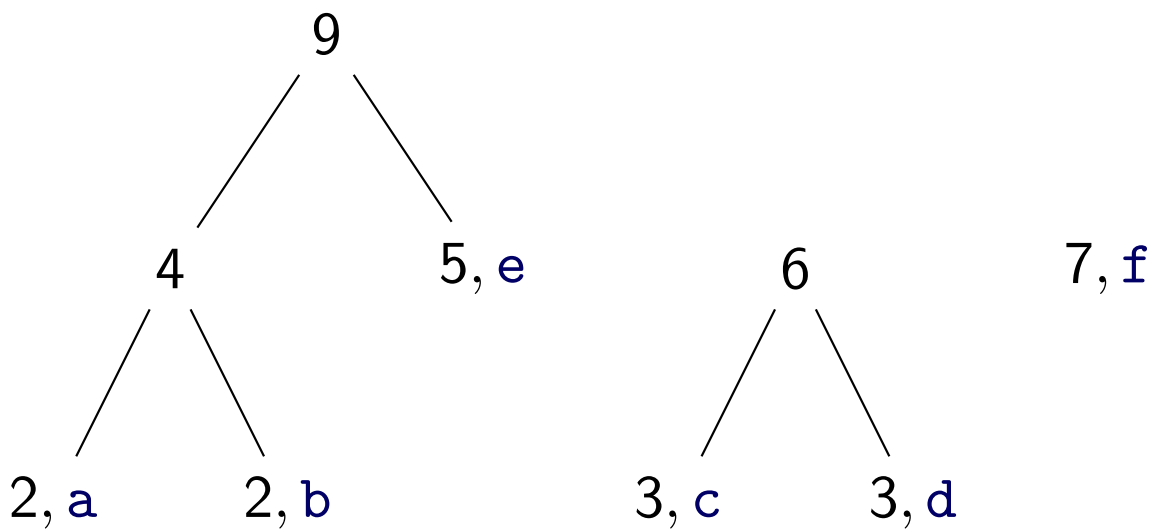
$$M_2 = \{\ (4, \{\texttt{a}, \texttt{b}\})\ ,\ (6, \{\texttt{c}, \texttt{d}\})\ ,\ (5, \{\texttt{e}\})\ ,\ (7, \{\texttt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (6)

Beispiel

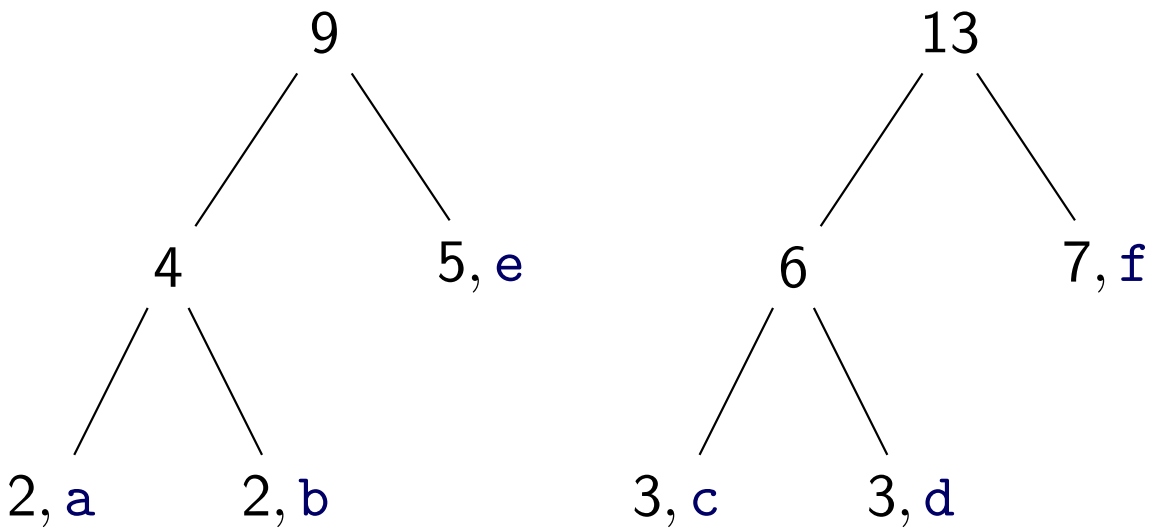
$$M_3 = \{\ (9, \{\texttt{a}, \texttt{b}, \texttt{e}\})\ ,\ (6, \{\texttt{c}, \texttt{d}\})\ ,\ (7, \{\texttt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (7)

Beispiel

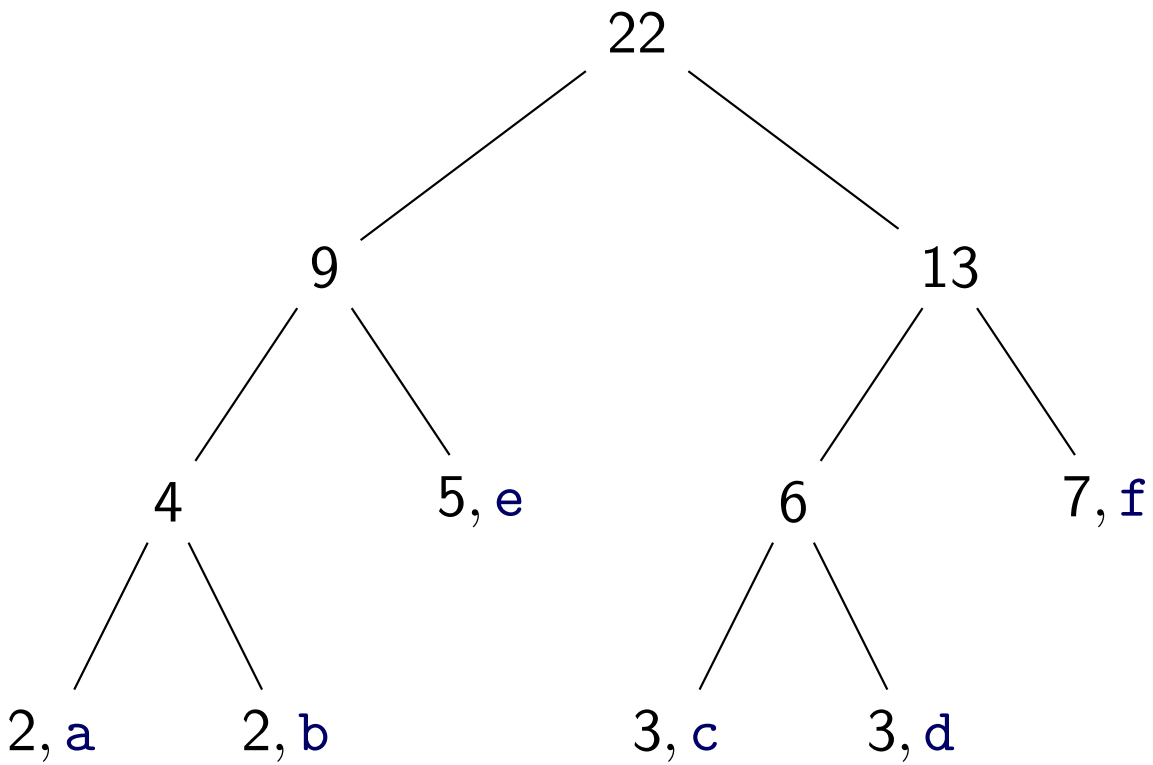
$$M_4 = \{\ (9, \{\mathtt{a}, \mathtt{b}, \mathtt{e}\})\ ,\ (13, \{\mathtt{c}, \mathtt{d}, \mathtt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (8)

Beispiel

$$M_5 = \{ \ (22, \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}\}) \ \}$$

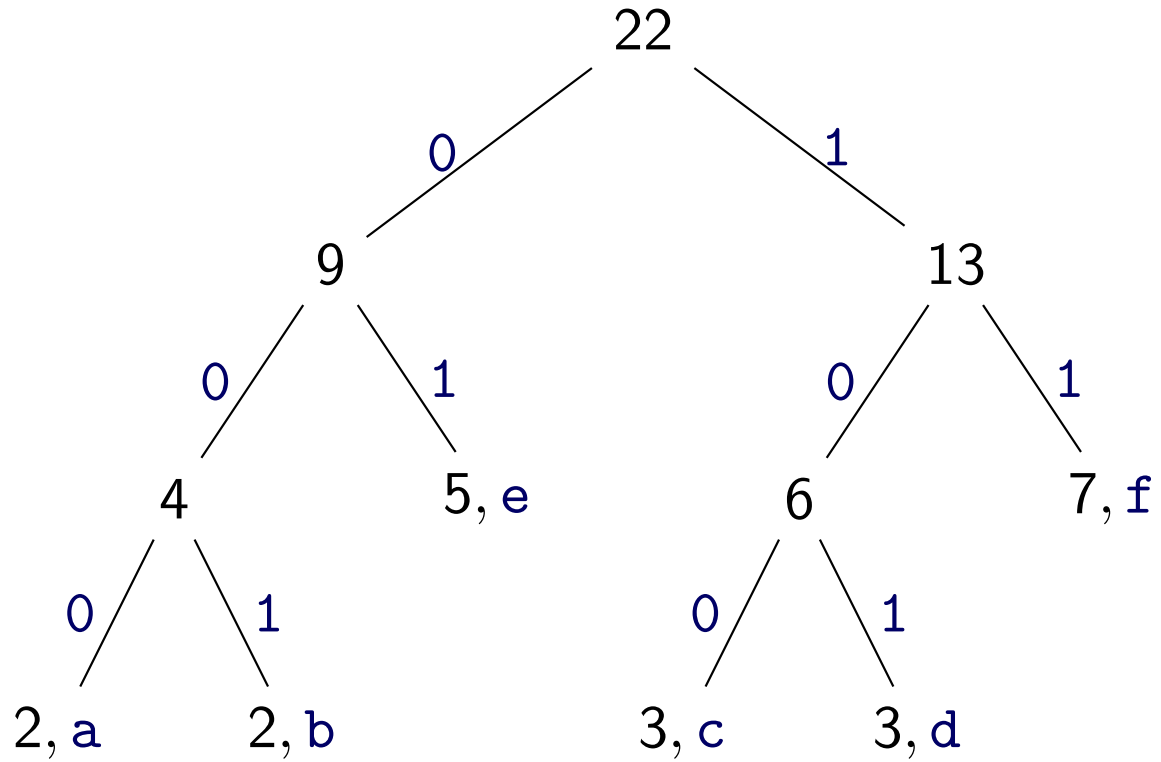


Beschriftung der Kanten

- an jedem Knoten:
  - die nach links führende Kante wird mit `0` beschriftet
  - die nach rechts führende Kante wird mit `1` beschriftet
- Codes für ein Zeichen x:
  - gehe auf kürzestem Weg von der Wurzel des Baumes zu dem Blatt, das x entspricht,
  - konkateniere der Reihe nach alle Symbole, mit denen die Kanten auf diesem Weg beschriftet sind.

Algorithmus für Huffman-Codes

- Beispiel: $w =$ `afebfecaffdeddccefbeff`
- Baum am Ende:



- Homomorphismus (präfixfrei!)

x	a	b	c	d	e	f
h(x)	000	001	100	101	01	11

Eigenschaften von Huffman-Codes

- Im Beispiel war immer klar, welche zwei Knoten zu einem neuen zusammengefügt werden mussten
- Im allgemeinen gibt es mehrere Möglichkeiten.
- Außerdem ist nicht festgelegt, welcher Knoten linker Nachfolger und welcher rechter Nachfolger eines inneren Knotens wird.
- Konsequenz dieser Mehrdeutigkeiten: Huffman-Code nicht eindeutig
- macht aber nichts: alle sind „gleich gut“.
- diese Güte kann man präzisieren: Unter allen präfixfreien Codes führen Huffman-Codes zu kürzesten Codierungen *des Wortes, für das die Huffman-Codierung konstuiert wurde.*

Block-Codierungen

- Verallgemeinerung des obigen Verfahrens:
  - Betrachte nicht Häufigkeiten einzelner Symbole,
  - sondern für Teilwörter einer festen Länge $b > 1$.
  - einziger Unterschied: an den Blättern des Huffman-Baumes stehen Wörter der Länge b.
- So etwas nennt man eine Block-Codierung.
  - Statt $h(x)$ für $x \in A$ festzulegen,
  - legt man $h(w)$ für alle Blöcke $w \in A^b$ fest, und
  - erweitert dies zu einer Funktion $h : (A^b)^* \to B^*$.

Das ist wichtig

Das sollten Sie mitnehmen:

- Huffman-Codierung liefert kürzest mögliche präfixfreie Codes
- „Algorithmus“ zur Bestimmung des Huffman-Baumes
- warum die Anführungszeichen?

Das sollten Sie üben:

- Huffman-Codes berechnen