

„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.“

„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.“

schrrieb 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.“

schrrieb 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

Universität Karlsruhe, Fakultät für Informatik

Wintersemester 2008/2009

Codierungen

Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

Huffman-Codierung

- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes

Codierungen

Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

Huffman-Codierung

- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes

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

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

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

$$\text{Num}_{10} : Z_{10}^* \rightarrow \mathbb{N}_0$$

- ▶ Schule:

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

- ▶ Pünktchenvermeidung:

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

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

- ▶ geboren 1. Juli 1646 (Leipzig), gestorben am 14. November 1716 (Hannover)
- ▶ baute zum Beispiel die erste Maschine, die zwei Zahlen multiplizieren 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.

Pour l'Addition par exemple. ☺	$\begin{array}{r} 110 \\ 111 \\ \hline 1101 \end{array} \Bigg \begin{array}{l} 6 \\ 7 \\ 13 \end{array}$	$\begin{array}{r} 101 \\ 1011 \\ \hline 10000 \end{array} \Bigg \begin{array}{l} 5 \\ 11 \\ 16 \end{array}$	$\begin{array}{r} 1110 \\ 10001 \\ \hline 11111 \end{array} \Bigg \begin{array}{l} 14 \\ 17 \\ 31 \end{array}$
	$\begin{array}{r} 1101 \\ 111 \\ \hline 110 \end{array} \Bigg \begin{array}{l} 13 \\ 7 \\ 6 \end{array}$	$\begin{array}{r} 10000 \\ 1011 \\ \hline 101 \end{array} \Bigg \begin{array}{l} 16 \\ 11 \\ 5 \end{array}$	$\begin{array}{r} 11111 \\ 10001 \\ \hline 1110 \end{array} \Bigg \begin{array}{l} 31 \\ 17 \\ 14 \end{array}$
Pour la Sou- straction.			

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

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

$$\text{num}_2(0) = 0$$

$$\text{num}_2(1) = 1$$

$$\text{Num}_2(\varepsilon) = 0$$

$$\forall w \in Z_2^* \forall x \in Z_2 : \text{Num}_2(wx) = 2 \cdot \text{Num}_2(w) + \text{num}_2(x)$$

- ▶ Z. B.:

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

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

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

- ▶ Zuordnung von Wörtern zu Zahlen gegeben durch

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

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

- ▶ die Alphabete Z_2 , Z_3 , usw. sind nicht disjunkt
- ▶ Darstellungen mehrdeutig; z. B. ist **111**
 - $\text{Num}_2(\mathbf{111})$ die Zahl sieben,
 - $\text{Num}_8(\mathbf{111})$ die Zahl dreiundsiebzig,
 - $\text{Num}_{10}(\mathbf{111})$ die Zahl einhundertelf und
 - $\text{Num}_{16}(\mathbf{111})$ 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 $\text{Num}_k(w)$ 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 $\text{repr}_k(i)$ 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 $\text{Num}_k(w) = n$
 - ▶ für die naheliegende Definition von Num_k
- ▶ gleich: solche Wörter gibt es immer
- ▶ Und wenn, dann auch immer gleich unendlich viele: wenn $\text{Num}_k(w) = n$, dann auch $\text{Num}_k(0w) = n$

```
// 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  
     $r \leftarrow y \bmod k$   
     $w \leftarrow \text{repr}_k(r) \cdot w$   
     $y \leftarrow y \text{ div } k$   
od  
// am Ende:  $n = \text{Num}_k(w)$ 
```

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

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

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

- ▶ Beachte:
 - ▶ umgekehrt im allgemeinen

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

- ▶ weil „überflüssige“ führende Nullen wegfallen.

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

Codierungen

Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

Huffman-Codierung

- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes

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

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

- ▶ wesentlicher Punkt:
 - ▶ $\mathbf{A3}$ und $\mathbf{10100011}$ haben *die gleiche Bedeutung*
 - ▶ nämlich die Zahl einhundertdreiundsechzig
- ▶ So etwas wollen wir eine **Übersetzung** nennen.

- ▶ 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“

- ▶ 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 $\text{sem}_A : L_A \rightarrow \text{Sem}$ und $\text{sem}_B : L_B \rightarrow \text{Sem}$
- ▶ Eine Abbildung $f : L_A \rightarrow L_B$ heie eine **Übersetzung** bezüglich sem_A und sem_B , wenn f die Bedeutung erhält, d. h.

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

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

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

- ▶ zum Beispiel bei der Übersetzung von Programmiersprachen
- ▶ Was ist Sem?
 - ▶ Das lernt man in Vorlesungen zur „Semantik von Programmiersprachen“.
- ▶ Andeutung: Man kann als Semantik von

$$x \leftarrow 5$$

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

- ▶ **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

- ▶ Forderung $\text{sem}_A(w) = \text{sem}_B(f(w))$ 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 dann die Bedeutung sem_B im wesentlichen *definieren* durch die Festlegung $\text{sem}_B(f(x)) = \text{sem}_A(x)$.
 - ▶ Beachte: Dafür ist Injektivität von f wichtig!
- ▶ Wenn f injektiv ist, heie das eine **Codierung**
- ▶ Die Menge $\{f(w) \mid w \in L_A\}$ heit dann auch ein **Code**.

Wie spezifiziert man eine Übersetzung?

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

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

$$h^*(\varepsilon) = \varepsilon$$

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

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

$$h(\mathbf{babb}) = h(\mathbf{b}) \cdot h(\mathbf{a}) \cdot h(\mathbf{b}) \cdot h(\mathbf{b}) = 001 \cdot 10 \cdot 001 \cdot 001 = 00110001001$$

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

- ▶ Gegeben: Abbildung $h : A \rightarrow B^*$
- ▶ Frage: ist Abbildung $h : A^* \rightarrow B^*$ 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 $h(x_2)$.

- ▶ Die Codierung einzelner Zeichen wird gleich definiert
- ▶ Wörter werden zeichenweise codiert.

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

- ▶ 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. ...

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

Codierungen

Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

Huffman-Codierung

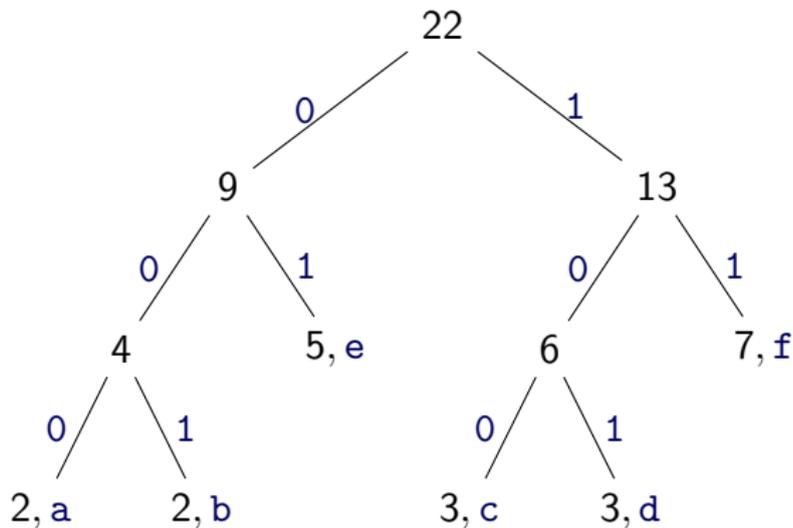
- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes

- ▶ Gegeben: ein Alphabet A und ein Wort $w \in A^*$
- ▶ Eine sogenannte **Huffman-Codierung** von w ist
 - ▶ eine Abbildung $h : A^* \rightarrow Z_2^*$,
 - ▶ die ein ε -freier Homomorphismus ist.
 - ▶ h also eindeutig festgelegt durch die $h(x)$ für alle $x \in A$.
- ▶ Jedes Symbol $x \in A$ kommt mit einer gewissen absoluten Häufigkeit $N_x(w)$ in w vor.
- ▶ Bei Huffman-Codes werden häufigere Symbole durch kürzere Wörter codiert und seltener vorkommende Symbole durch längere.
- ▶ Huffman-Codierungen werden zum Beispiel von Kompressionsverfahren wie gzip oder bzip2 verwendet.
- ▶ Im folgenden
 - ▶ erst der Algorithmus
 - ▶ anschließend einige Bemerkungen

- ▶ 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)

- ▶ Beispiel: $w = \text{afefbecaffdeddccefbeff}$
- ▶ Baum am Ende:



- ▶ Homomorphismus (präfixfrei!)

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

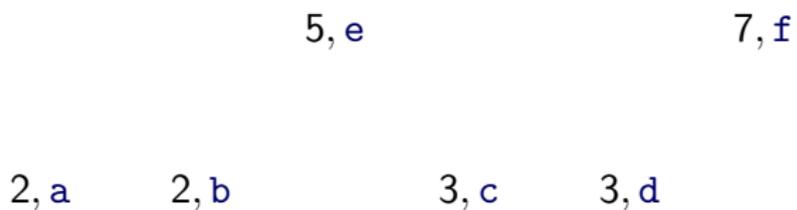
Konstruktion des Huffman-Baumes (1)

- ▶ 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

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

Konstruktion des Huffman-Baumes (2)

Initialbaum im Beispiel:



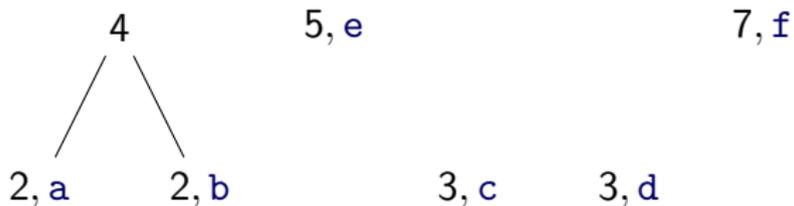
Iterationsschritt des Algorithmus:

- ▶ Solange Menge M_i noch mindestens zwei Paare enthält, Bestimme Menge M_{i+1} wie folgt:
 - ▶ Wähle zwei Paare (X_1, k_1) und (X_2, k_2) , deren Häufigkeiten zu den kleinsten noch vorkommenden gehören.
 - ▶ Entferne diese Paare aus M_i und fügt statt dessen das eine Paar $(X_1 \cup X_2, k_1 + k_2)$ 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 (X_1, k_1) und (X_2, k_2) eingefügt worden waren.

Konstruktion des Huffman-Baumes (4)

Beispiel:

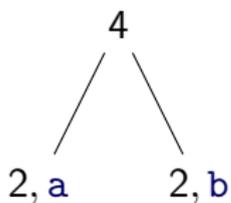
$$M_1 = \{ (\{a, b\}, 4), (\{c\}, 3), (\{d\}, 3), (\{e\}, 5), (\{f\}, 7) \}$$



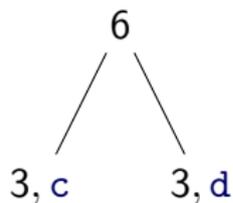
Konstruktion des Huffman-Baumes (5)

Beispiel:

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



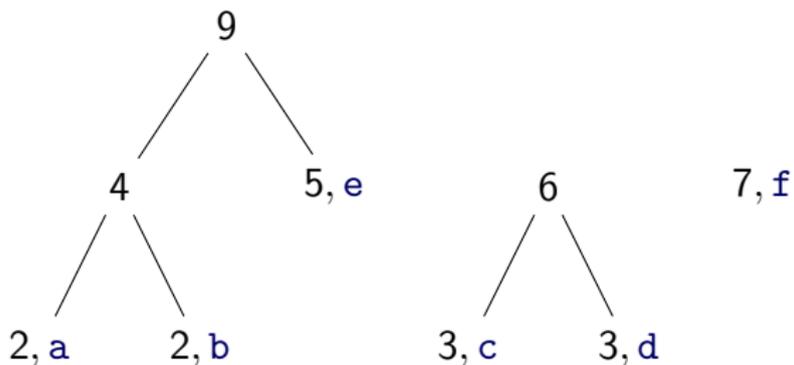
5, e



7, f

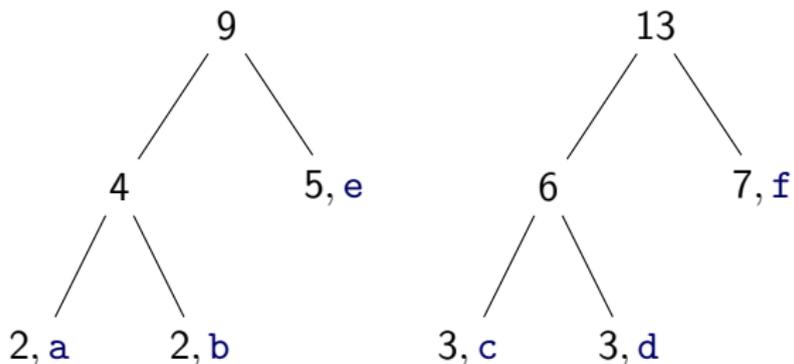
Beispiel

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



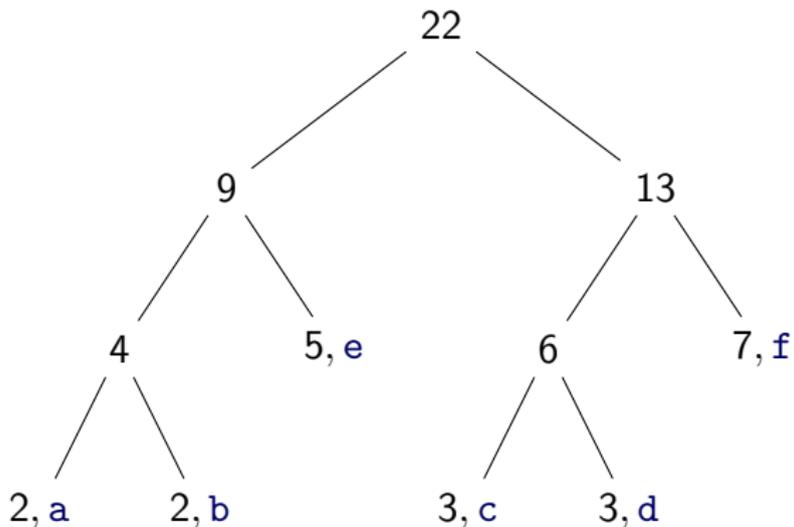
Beispiel

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



Beispiel

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



- ▶ 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.

- ▶ 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.

- ▶ 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 $h : (A^b)^* \rightarrow B^*$.

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

- ▶ Zahldarstellungen
- ▶ Übersetzungen und Codes
- ▶ Huffman-Codierung