Grundbegriffe der Informatik

Einheit 9: Speicher

Prof. Dr. Tanja Schultz

Karlsruher Institut für Technologie, Fakultät für Informatik

Wintersemester 2012/2013

Überblick

Speicher

- Bit und Byte
- Speicher als Tabellen und Abbildungen
- Binäre und dezimale Größenpräfixe

Überblick

Speicher

- Bit und Byte
- Speicher als Tabellen und Abbildungen
- Binäre und dezimale Größenpräfixe

Überblick

Bit und Byte

- Das Wort „Bit“ hat verschiedene Bedeutungen.
  - Bezeichnung für eine Binärziffer (binary digit):
    Ein Bit ist ein Zeichen des Alphabetes $\{0, 1\}$.
  - Maßeinheit für die Datenmenge bei digitaler Speicherung oder Übertragung von Daten
  - Maßeinheit für den Informationsgehalt (Shannon)
    (siehe Vorlesung „Theoretische Grundlagen“ im 3. Semester)
- Byte: ein Wort aus acht Bits - genauer Oktett - Octet
- Historisch Byte = Anzahl Bits zur Kodierung eines einzigen Schriftzeichens in einem Computer, bzw. kleinste addressierbare Datenmenge eines technischen Systems
  (ASCII - 7 bit; IBM-PC - 8 bit; Nixdorf 820 - 12 bit ...)
- Abkürzungen
  - für Bit: möglichst „bit“, denn „b“ ist Flächeneinheit „barn“
  - für Byte: meist „B“, aber „B“ auch „Bel“, „deziBel“ - dB
  - für Octet: „o“

Überblick

im folgenden

- Formalisierung von
  - Speicher
  - Lesen aus dem Speicher
  - Schreiben in den Speicher
- Diskussion, wozu diese Formalisierungen

Gesamtzustand eines Speichers

- aktueller Gesamtzustand eines Speichers:
  - für jede Adresse, zu der etwas gespeichert ist:
  - welcher Wert ist unter dieser Adresse abgelegt
- Vorstellung: Tabelle mit zwei Spalten:
  - links alle Adressen
  - rechts die zugehörigen Werte

allgemein	
Adresse 1	Wert 1
Adresse 2	Wert 2
Adresse 3	Wert 3
Adresse 4	Wert 4
⋮	⋮
Adresse n	Wert n

Halbleiterspeicher	
000	10110101
001	10101101
010	10011101
011	01110110
100	00111110
101	10101101
110	00101011
111	10101001

Gesamtzustand eines Speichers

- aktueller Gesamtzustand eines Speichers:
  - für jede Adresse, zu der etwas gespeichert ist:
  - welcher Wert ist unter dieser Adresse abgelegt
- Vorstellung: Tabelle mit zwei Spalten:
  - links alle Adressen
  - rechts die zugehörigen Werte

allgemein	
Adresse 1	Wert 1
Adresse 2	Wert 2
Adresse 3	Wert 3
Adresse 4	Wert 4
⋮	⋮
Adresse n	Wert n

Halbleiterspeicher	
000	10110101
001	10101101
010	10011101
011	01110110
100	00111110
101	10101101
110	00101011
111	10101001

Formalisierung von Speicher

- Tabelle: Abbildung von Adressen auf Werte, also Definitionsbereich: Menge aller Adressen, Wertebereich: Menge aller speicherbaren Werte

$$m : \mathrm{Adr} \to \mathrm{Val}$$

- Z. B. Halbleiterspeicher in einem PC mit „4 Gigabyte“:

$$m : \{0,1\}^{32} \to \{0,1\}^{8}$$

- Speicher im Zustand m hat an Adresse $a \in \mathrm{Adr}$ ($2^{32} = 4294967296$) den Wert $m(a) \in \mathrm{Val}$ ($2^8 = 256$) gespeichert.
- bei Hauptspeicher:
  - Menge der Adressen ist fest
  - Adresse bezeichnet einen physikalischen Ort auf dem Chip
  - entspricht einer Angabe auf einem Brief, etwa wie „Adenauerring 4, 76131 Karlsruhe“

Formalisierung von Lesen aus dem Speicher

- Formalisierung als Abbildung $\mathrm{memread}$
  - Argumente:
    - der gesamte Speicherinhalt m des Speichers und
    - die Adresse a aus der ausgelesen wird
  - Resultat: der in m an Adresse a gespeicherte Wert, d.h.

$$\mathrm{memread} : \mathrm{Mem} \times \mathrm{Adr} \to \mathrm{Val}$$
$$(m, a) \mapsto m(a)$$

- Dabei sei Mem die Menge aller möglichen Speicherzustände, also die Menge aller Abbildungen von Adr nach Val.

- auf das „Warum überhaupt so eine Formalisierung?“ kommen wir noch zu sprechen

Bemerkungen zu $\mathrm{memread}$

- nicht verwirren lassen:
  - Funktion $\mathrm{memread}$ bekommt als Argument eine (andere) Funktion $m : \mathrm{Adr} \to \mathrm{Val}$
  - Beispiel zeigt: kein Problem; man denke an Tabellen
- die Menge aller Abbildungen der Form $f : A \to B$
  - so etwas kommt noch öfter vor
  - Notation: B^A
  - beachte Reihenfolge
  - für endliche Mengen A und B gilt: $|B^A| = |B|^{|A|}$.
- hätten also auch schreiben können:
  - $\mathrm{Mem} = \mathrm{Val}^{\mathrm{Adr}}$ und
  - $$\mathrm{memread} : \mathrm{Val}^{\mathrm{Adr}} \times \mathrm{Adr} \to \mathrm{Val}$$
    $$(m, a) \mapsto m(a)$$

Formalisierung von Schreiben in den Speicher

Das Speichern eines neuen Wertes v an eine Adresse a in einem Speicher m kann man auch als Funktion notieren, sieht ein wenig komplizierter aus als beim Lesen aus dem Speicher:

- $$\mathrm{memwrite} : \mathrm{Val}^{\mathrm{Adr}} \times \mathrm{Adr} \times \mathrm{Val} \to \mathrm{Val}^{\mathrm{Adr}}$$
  $$(m, a, v) \mapsto m'$$
  - wobei m' festgelegt durch die Forderung, dass für alle $a' \in \mathrm{Adr}$ gilt:

$$m'(a') = \begin{cases} v & \text{falls } a' = a \\ m(a') & \text{falls } a' \neq a \end{cases}$$

  - d.h. Speicher m' enthält an Addresse a den Wert v
- Das klingt plausibel ...
  - für Hauptspeicher ist es das auch,
  - aber es gibt auch Speicher die anders arbeiten, z. B. sogenannte Cache-Speicher (siehe VL Technische Informatik)

Eigenschaften von Speicher

- Was ist „das wesentliche“ an Speicher?
- Der allerwichtigste Aspekt überhaupt:
  Für alle $m \in \mathrm{Mem}$, $a, a' \in \mathrm{Adr}$ mit $a' \neq a$ und $v' \in \mathrm{Val}$ gilt:

$$\mathrm{memread}(\mathrm{memwrite}(m, a, v), a) = v$$

- d.h. wenn man Wert v an Adresse a in einen Speicher m schreibt und danach diesen Speicher liest, dann ist der Wert v.
- Reicht das als Spezifikation für Speicher aus? - Nicht für alle Speicher (z.B. Cache)
- Für die oben definierten Funktionen gilt nämlich auch:
  Für alle $m \in \mathrm{Mem}$, $a, a' \in \mathrm{Adr}$ mit $a' \neq a$ und $v' \in \mathrm{Val}$:

$$\mathrm{memread}(m, a) = \mathrm{memread}(\mathrm{memwrite}(m, a', v'), a)$$

- d.h. das Ergebnis des Lesens einer Speicherstelle hängt nicht davon ab, was vorher an einer anderen Adresse gespeichert wurde.

Wozu diese Formalisierungen?

- Die Gleichungen sagen etwas darüber aus, wie „sich der Speicher verhalten“ soll
- d.h. sie liefern eine Möglichkeit für den *Spezifizierer* von Speicher, auszudrücken
  - „wie sich Speicher verhalten soll“
  - algebraische Spezifikation (in späteren VLs lernen Sie noch komplexere Formen der algebraischen Spezifikation von Datentypen kennen)
- auch eine Möglichkeit für den *Impementierer*, sich über Testfälle klar zu werden
  - Erstellen von Testfällen, Testdurchführung
  - Testen kann nicht Korrektheit einer Implementierung beweisen,
  - aber immerhin, dass sie falsch ist.

Was ist wichtig

Das sollten Sie mitnehmen:

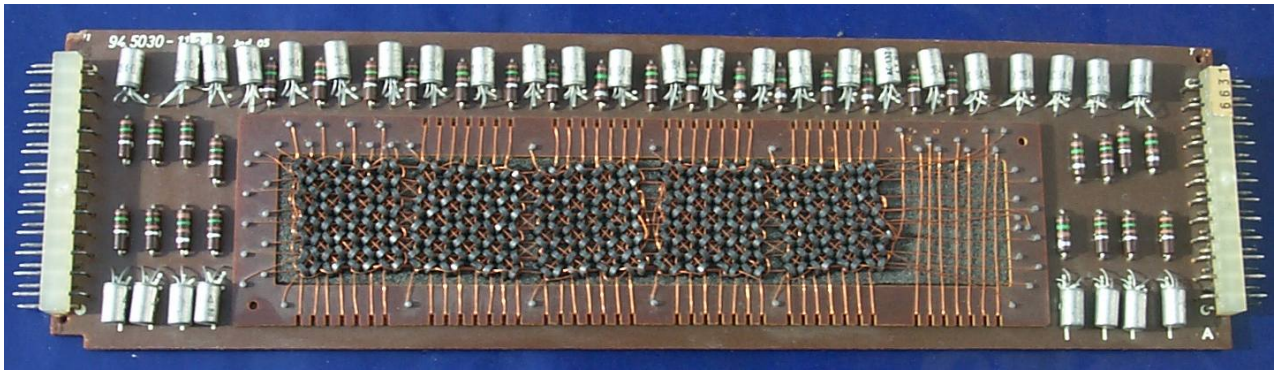
- es gibt nicht nur Hauptspeicher
- Adressen sind manchmal etwas anderes als „physikalische Koordinaten“
- Abbildungen kann man sich manchmal am besten als Tabelle vorstellen

Das sollten Sie üben:

- Gewöhnung an Abbildungen, die Abbildungen auf Abbildungen abbilden

Klein und groß

- früher: Speicher klein, z. B. ein paar Hundert Bits



Bildquelle: `http://de.wikipedia.org/w/index.php?title=Bild:Kernspeicher1.jpg`

- heute: groß, z. B.
  - Hauptspeicher: $2^{32} = 4\,294\,967\,296$ Bytes
  - Festplatten: so was wie $1\,000\,000\,000\,000$ Bytes
- Zahlen nur noch schlecht zu lesen
- Benutzung von Präfixen für kompaktere Notation: Kilometer, Mikrosekunde, Megawatt, usw.

Dezimale Größenpräfixe

10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}	10^{-18}
1000^{-1}	1000^{-2}	1000^{-3}	1000^{-4}	1000^{-5}	1000^{-6}
milli	**mikro**	**nano**	**pico**	**femto**	**atto**
m	μ	n	p	f	a
10^{3}	10^{6}	10^{9}	10^{12}	10^{15}	10^{18}
1000^{1}	1000^{2}	1000^{3}	1000^{4}	1000^{5}	1000^{6}
kilo	**mega**	**giga**	**tera**	**peta**	**exa**
k	M	G	T	P	E

Binäre Größenpräfixe

- In Rechnern häufig Größen, die Potenzen von 2 oder 2^{10} sind, und *nicht* Potenzen von 10 bzw. 1000.
- Die *International Electrotechnical Commission* hat 1999 Präfixe eingeführt, die für Potenzen von $2^{10} = 1024$ stehen.
- motiviert durch Kunstworte „kilobinary“, „megabinary”, usw.
- Präfixe *kibi*, *mebi*, *gibi*, usw.
- abgekürzt Ki, Mi, Gi, usw.

2^{10}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}
1024^1	1024^2	1024^3	1024^4	1024^5	1024^6
kibi	mebi	gibi	tebi	pebi	exbi
Ki	Mi	Gi	Ti	Pi	Ei

Wir halten fest

Das sollten Sie mitnehmen:

- Bit und Byte / Oktett,Octet
- binäre Größenpräfixe

Das sollten Sie üben:

- Rechnen mit binären Größenpräfixen

Grundbegriffe der Informatik

Einheit 10: Codierungen

Prof. Dr. Tanja Schultz

Karlsruher Institut für Technologie, Fakultät für Informatik

Wintersemester 2012/2013

Überblick

Überblick

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	$\cdots$	8	9
$\mathrm{num}_{10}(x)$	*null*	*eins*	*zwei*	$\cdots$	*acht*	*neun*

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

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

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

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

- Beispiel aus der Schule: $164 = 10^2 \cdot 1 + 10^1 \cdot 6 + 10^0 \cdot 4, k = 3$

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

- Pünktchenvermeidung:

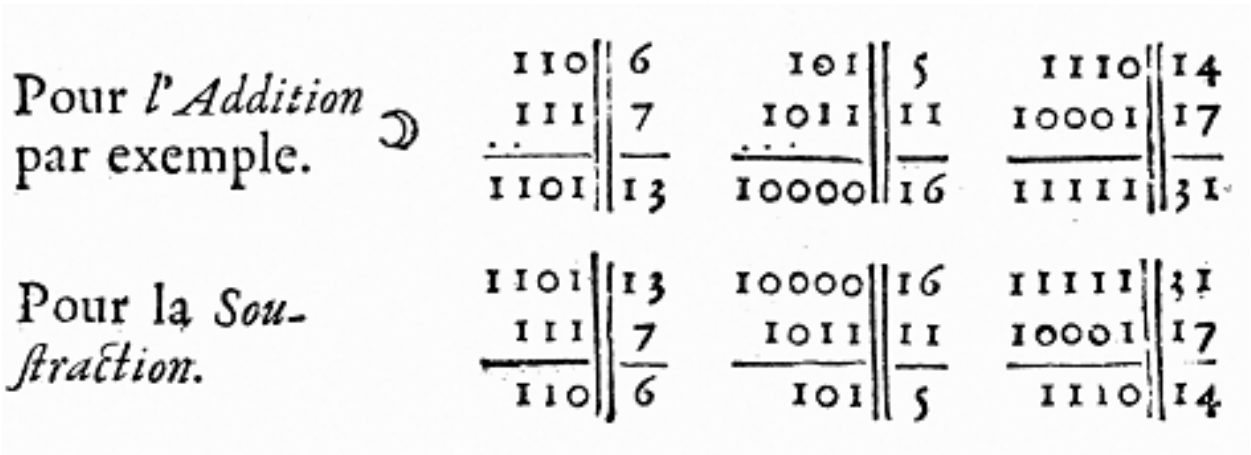
$$\mathrm{Num}_{10}(\varepsilon) = 0$$

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

Gottfried Wilhelm Leibniz

- geboren 1. Juli 1646 in Leipzig
  gestorben am 14. November 1716 in Hannover
- baute erste Maschine, die zwei Zahlen multiplizieren konnte

- in einem Brief vom 2. Januar 1697 an den Herzog von Braunschweig-Wolfenbüttel:
  - auch wenn man nur zwei Ziffern 0 und 1 hat, kann man nichtnegative Zahlen darstellen und vernünftig rechnen



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

Binärdarstellung von Zahlen

- Ziffernmenge $Z_2 = \{0, 1\}$
- definiere: $\mathrm{num}_2(0) = 0$ und $\mathrm{num}_2(1) = 1$ und

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

- Z. B.:

$$\begin{aligned}
\mathrm{Num}_2(1101) &= 2 \cdot \mathrm{Num}_2(110) + 1 \\
&= 2 \cdot (2 \cdot \mathrm{Num}_2(11) + 0) + 1 \\
&= 2 \cdot (2 \cdot (2 \cdot \mathrm{Num}_2(1) + 1) + 0) + 1 \\
&= 2 \cdot (2 \cdot (2 \cdot (2 \cdot \mathrm{Num}_2(\varepsilon) + 1) + 1) + 0) + 1 \\
&= 2 \cdot (2 \cdot (2 \cdot (2 \cdot 0 + 1) + 1) + 0) + 1 \\
&= 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 \\
&= 13
\end{aligned}$$

Hexadezimaldarstellung

- Ziffern $Z_{16} = \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}, \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}\}$.
- 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; Beispiel `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

Von Zahl zu ihrer Darstellung

- Man kann zu $w \in Z_k^*$ die repräsentierte Zahl $\mathrm{Num}_k(w)$ berechnen.
- Umgekehrt geht es auch:
  man kann zu $n \in \mathbb{N}_0$ die sogenannte k-äre Darstellung (also Wort $w \in Z_k^*$) 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
- gibt es immer unendlich viele passende w:
  wenn $\mathrm{Num}_k(w) = n$, dann auch $\mathrm{Num}_k(0w) = n$
  (Beweis durch Induktion!)

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

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

$y \leftarrow y$ **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

- ($r \leftarrow y$ **mod** k , $w \leftarrow \mathrm{repr}_k(r) \cdot w$ und $y \leftarrow y$ **div** k)
- 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 $\mathrm{Num}_k(w) = n$.
- Wir schreiben dafür $\mathrm{Repr}_k(n)$.
- Es gilt stets

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

- Beachte:
  - Die Umkehrung gilt im allgemeinen nicht

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

  - weil „überflüssige“ führende Nullen wegfallen, bspw. für $w = 004711$ gilt $\mathrm{Num}_k(004711) = \mathrm{Num}_k(4711)$

noch ein Beispiel (1)

- Ziffernmenge $Z = \{\mathtt{\text{ɪ}}, \mathtt{0}, \mathtt{1}\}$
- definiere

$$\begin{aligned}\mathrm{num}(\mathtt{\text{ɪ}}) &= -1\\ \mathrm{num}(\mathtt{0}) &= 0\\ \mathrm{num}(\mathtt{1}) &= 1\end{aligned}$$

- definiere $\mathrm{Num} : Z^* \to \mathbb{Z}$

$$\mathrm{Num}(\varepsilon) = 0$$
$$\forall w \in Z^*\ \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- Z. B.: $\mathrm{Num}(\mathtt{\text{ɪ}01}) = -3^2 + 0 + 3^0 = -8$ und $\mathrm{Num}(\mathtt{1\text{ɪ}01}) = +3^3 - 3^2 + 0 + 3^0 = 19$

noch ein Beispiel (2)

- $\mathrm{num}(\mathtt{1}) = -1$, $\mathrm{num}(\mathtt{0}) = 0$ und $\mathrm{num}(\mathtt{1}) = 1$ und

$$\mathrm{Num}(\varepsilon) = 0$$
$$\forall w \in Z^* \; \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- definere $\mathrm{inv} : Z^* \to Z^*$
  - $\mathrm{inv}(\mathtt{1}) = \mathtt{1}$, $\mathrm{inv}(\mathtt{1}) = \mathtt{1}$ und $\mathrm{inv}(\mathtt{0}) = \mathtt{0}$.
  - $\mathrm{inv}(\varepsilon) = \varepsilon$ und $\forall w \in Z^* : \forall x \in Z : \mathrm{inv}(wx) = \mathrm{inv}(w)\mathrm{inv}(x)$

  also z. B. $\mathrm{inv}(\mathtt{1}\mathtt{1}\mathtt{0}\mathtt{1}) = \mathtt{1}\mathtt{1}\mathtt{0}\mathtt{1}$
- dann gilt für alle $w \in Z^*$: $\mathrm{Num}(\mathrm{inv}(w)) = -\mathrm{Num}(w)$.
- $(1 + 1 = 3 - 1 = 1\mathtt{1}$; $-1 - 1 = -3 + 1 = \mathtt{1}1$; $0 + 1 = 1)$

noch ein Beispiel (2)

- $\mathrm{num}(\text{ɪ}) = -1$, $\mathrm{num}(0) = 0$ und $\mathrm{num}(1) = 1$ und

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

$$\forall w \in Z^* \; \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- definere $\mathrm{inv} : Z^* \to Z^*$
  - $\mathrm{inv}(1) = \text{ɪ}$, $\mathrm{inv}(\text{ɪ}) = 1$ und $\mathrm{inv}(0) = 0$.
  - $\mathrm{inv}(\varepsilon) = \varepsilon$ und $\forall w \in Z^* : \forall x \in Z : \mathrm{inv}(wx) = \mathrm{inv}(w)\mathrm{inv}(x)$

  also z. B. $\mathrm{inv}(1\text{ɪ}01) = \text{ɪ}10\text{ɪ}$
- dann gilt für alle $w \in Z^*$: $\mathrm{Num}(\mathrm{inv}(w)) = -\mathrm{Num}(w)$.
- „Addition“ wie „in der Schule“:

$$\begin{array}{cccc} 1 & \text{ɪ} & 0 & 1 \\ & \text{ɪ} & 0 & 1 \\ \hline \end{array}$$

- $(1 + 1 = 3 - 1 = 1\text{ɪ};\; -1 - 1 = -3 + 1 = \text{ɪ}1;\; 0 + 1 = 1)$

noch ein Beispiel (2)

- $\mathrm{num}(\text{ɪ}) = -1$, $\mathrm{num}(0) = 0$ und $\mathrm{num}(1) = 1$ und

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

$$\forall w \in Z^* \; \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- definere $\mathrm{inv} : Z^* \to Z^*$
  - $\mathrm{inv}(1) = \text{ɪ}$, $\mathrm{inv}(\text{ɪ}) = 1$ und $\mathrm{inv}(0) = 0$.
  - $\mathrm{inv}(\varepsilon) = \varepsilon$ und $\forall w \in Z^* : \forall x \in Z : \mathrm{inv}(wx) = \mathrm{inv}(w)\mathrm{inv}(x)$

  also z. B. $\mathrm{inv}(1\text{ɪ}01) = \text{ɪ}10\text{ɪ}$
- dann gilt für alle $w \in Z^*$: $\mathrm{Num}(\mathrm{inv}(w)) = -\mathrm{Num}(w)$.
- „Addition“ wie „in der Schule“:

$$\begin{array}{cccc} 1 & \text{ɪ} & 0 & 1 \\ & \text{ɪ} & 0 & 1 \\ & & 1 & \\ \hline & & & \text{ɪ} \end{array}$$

- $(1 + 1 = 3 - 1 = 1\text{ɪ};\; -1 - 1 = -3 + 1 = \text{ɪ}1;\; 0 + 1 = 1)$

noch ein Beispiel (2)

- $\mathrm{num}(\text{1}) = -1$, $\mathrm{num}(\text{0}) = 0$ und $\mathrm{num}(\text{1}) = 1$ und

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

$$\forall w \in Z^* \; \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- definere $\mathrm{inv} : Z^* \to Z^*$
  - $\mathrm{inv}(\text{1}) = \text{1}$, $\mathrm{inv}(\text{1}) = \text{1}$ und $\mathrm{inv}(\text{0}) = \text{0}$.
  - $\mathrm{inv}(\varepsilon) = \varepsilon$ und $\forall w \in Z^* : \forall x \in Z : \mathrm{inv}(wx) = \mathrm{inv}(w)\mathrm{inv}(x)$

  also z. B. $\mathrm{inv}(\text{1101}) = \text{110}$
- dann gilt für alle $w \in Z^*$: $\mathrm{Num}(\mathrm{inv}(w)) = -\mathrm{Num}(w)$.
- „Addition“ wie „in der Schule“:

$$\begin{array}{cccc} \text{1} & \text{1} & \text{0} & \text{1} \\ & \text{1} & \text{0} & \text{1} \\ & \text{0} & \text{1} & \\ \hline & & \text{1} & \text{1} \end{array}$$

- $(1 + 1 = 3 - 1 = 1\text{1};\; -1 - 1 = -3 + 1 = \text{1}1;\; 0 + 1 = 1)$

noch ein Beispiel (2)

- $\mathrm{num}(\text{ɪ}) = -1$, $\mathrm{num}(0) = 0$ und $\mathrm{num}(1) = 1$ und

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

$$\forall w \in Z^* \; \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- definere $\mathrm{inv} : Z^* \to Z^*$
  - $\mathrm{inv}(1) = \text{ɪ}$, $\mathrm{inv}(\text{ɪ}) = 1$ und $\mathrm{inv}(0) = 0$.
  - $\mathrm{inv}(\varepsilon) = \varepsilon$ und $\forall w \in Z^* : \forall x \in Z : \mathrm{inv}(wx) = \mathrm{inv}(w)\mathrm{inv}(x)$

  also z. B. $\mathrm{inv}(1\text{ɪ}01) = \text{ɪ}10\text{ɪ}$
- dann gilt für alle $w \in Z^*$: $\mathrm{Num}(\mathrm{inv}(w)) = -\mathrm{Num}(w)$.
- „Addition“ wie „in der Schule“:

$$\begin{array}{cccc} 1 & \text{ɪ} & 0 & 1 \\ & \text{ɪ} & 0 & 1 \\ \text{ɪ} & 0 & 1 & \\ \hline & 1 & 1 & \text{ɪ} \end{array}$$

- $(1 + 1 = 3 - 1 = 1\text{ɪ};\; -1 - 1 = -3 + 1 = \text{ɪ}1;\; 0 + 1 = 1)$

noch ein Beispiel (2)

- $\mathrm{num}(\mathtt{1}) = -1$, $\mathrm{num}(\mathtt{0}) = 0$ und $\mathrm{num}(\mathtt{1}) = 1$ und

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

$$\forall w \in Z^* \; \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- definere $\mathrm{inv} : Z^* \to Z^*$
  - $\mathrm{inv}(\mathtt{1}) = \mathtt{1}$, $\mathrm{inv}(\mathtt{1}) = \mathtt{1}$ und $\mathrm{inv}(\mathtt{0}) = \mathtt{0}$.
  - $\mathrm{inv}(\varepsilon) = \varepsilon$ und $\forall w \in Z^* : \forall x \in Z : \mathrm{inv}(wx) = \mathrm{inv}(w)\mathrm{inv}(x)$

  also z. B. $\mathrm{inv}(\mathtt{1101}) = \mathtt{110}$
- dann gilt für alle $w \in Z^*$: $\mathrm{Num}(\mathrm{inv}(w)) = -\mathrm{Num}(w)$.
- „Addition“ wie „in der Schule“:

$$\begin{array}{cccc} \mathtt{1} & \mathtt{1} & \mathtt{0} & \mathtt{1} \\ & \mathtt{1} & \mathtt{0} & \mathtt{1} \\ \mathtt{1} & \mathtt{0} & \mathtt{1} & \\ \hline \mathtt{0} & \mathtt{1} & \mathtt{1} & \mathtt{1} \end{array}$$

- $(1 + 1 = 3 - 1 = 1\mathtt{1};\; -1 - 1 = -3 + 1 = \mathtt{1}1;\; 0 + 1 = 1)$

noch ein Beispiel (2)

- $\mathrm{num}(\text{ɪ}) = -1$, $\mathrm{num}(0) = 0$ und $\mathrm{num}(1) = 1$ und

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

$$\forall w \in Z^* \; \forall x \in Z : \mathrm{Num}(wx) = 3 \cdot \mathrm{Num}(w) + \mathrm{num}(x)$$

- definere $\mathrm{inv} : Z^* \to Z^*$
  - $\mathrm{inv}(1) = \text{ɪ}$, $\mathrm{inv}(\text{ɪ}) = 1$ und $\mathrm{inv}(0) = 0$.
  - $\mathrm{inv}(\varepsilon) = \varepsilon$ und $\forall w \in Z^* : \forall x \in Z : \mathrm{inv}(wx) = \mathrm{inv}(w)\mathrm{inv}(x)$

  also z. B. $\mathrm{inv}(1\text{ɪ}01) = \text{ɪ}10\text{ɪ}$
- dann gilt für alle $w \in Z^*$: $\mathrm{Num}(\mathrm{inv}(w)) = -\mathrm{Num}(w)$.
- „Addition“ wie „in der Schule“:

1	ɪ	0	1	19
	ɪ	0	1	− 8
ɪ	0	1		
0	1	1	ɪ	11

- $(1 + 1 = 3 - 1 = 1\text{ɪ};\; -1 - 1 = -3 + 1 = \text{ɪ}1;\; 0 + 1 = 1)$

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:

- 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:

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

- $*$ weil wie bereits erwähnt: $\mathrm{Num}_k(\mathrm{Repr}_k(n)) = n$

Komplizierte Menge Sem

- Im allgemeinen kann die Menge der Bedeutungen Sem kompliziert sein
- Zum Beispiel bei der Übersetzung von Programmen
- Was ist Sem, wenn man ein Java-Programm in eine Folge von Bytecodes übersetzt?
  - siehe Vorlesungen zu „Semantik von Programmiersprachen“.

Wozu braucht man Übersetzungen?

- Legalität: Manchmal ist ein bestimmter Zeichensatz vorgeschrieben (z. B. ASCII in Emails)
- Lesbarkeit: Übersetzungen evtl. kürzer und besser lesbar: vergleiche `A3` oder `163` mit `10100011`
- Verschlüsselung: gerade das Umgekehrte, d.h. Erschweren der Lesbarkeit
  - am liebsten gar keine Lesbarkeit, jedenfalls für Außenstehende
  - siehe Vorlesungen über Kryptographie
- Kompression: Übersetzungen können kürzer sein
  - und zwar *ohne* größeres Alphabet
  - das sehen wir gleich: Huffman-Codes
- Fehlererkennung und Fehlerkorrektur:
  - mache Texte werden durch die Übersetzung länger,
  - so dass, falls ein korrekter Funktionswert $f(w)$ zufällig „kaputt“ geht (Übertragungsfehler),
  - u. U. Fehlererkennung oder gar Fehlerkorrektur möglich
  - siehe Vorlesungen über Codierungstheorie

Codierungen bzw. Codes

- In Spezialfällen ist die Forderung $\mathrm{sem}_A(w) = \mathrm{sem}_B(f(w))$ kein Problem:
- z. B. bei Verschlüsselung und manchen Anwendungen von Kompression:
  - man will vom Übersetzten $f(x)$ eindeutig zum ursprünglichen x zurückkommen
  - f ist injektiv (linkseindeutig: wenn $x_1 \neq x_2$ dann $f(x_1) \neq f(x_2)$).
- 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
    (aus dem Griechischen, homos für „gleich“ und morphe für „Form“), sind strukturerhaltende Abbildungen
  - 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 $h(\mathtt{b}) = \mathtt{001}$
  - $h^{**}(\mathtt{bab}) = h(\mathtt{b}) \cdot h(\mathtt{a}) \cdot h(\mathtt{b}) = \mathtt{001} \cdot \mathtt{10} \cdot \mathtt{001} = \mathtt{00110001}$

- Eine solche Abbildung h^{**} nennt man einen Homomorphismus.
- 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 $h(x_2)$.

Erläuterung und Beispiel

- Ein Code ist präfixfrei, wenn kein gültiges Codewort der Beginn eines anderen gültigen Codewortes ist.
- Für einen nicht präfixfreien Code müsste man die Symbole mit Hilfe von Trennzeichen markieren.
- Beispiel: Text ohne Trennsymbole „Urinstinkt“ könnte man dekodieren in „Urin stinkt“ - das Trennsymbol sorgt für Eindeutigkeit der Dekodierung
- Das zusätzliche Trennzeichen macht den Code allerdings weniger effizient.
- Durch die Bedingung der Präfixfreiheit definiert man den Code so, dass zwei beliebige Codeworte ohne Trennsymbol hintereinander geschrieben werden können, ohne die eindeutige Dekodierbarkeit zu gefährden
- In unserem Beispiel könnte man also statt Einführung der Trennzeichen die Präfixfreiheit fordern, d.h. es dürfte im Code keine Wörter „U, Ur, Uri, Urin, Urins, Urinst, Urinstin, ...“ geben

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 $h(\mathtt{a}) = \mathtt{1}$, $h(\mathtt{b}) = \mathtt{01}$, $h(\mathtt{c}) = \mathtt{001}$.
  - 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 ...

Präfixfreie Codes: Decodierung (4)

- Man spricht hier von Wohldefiniertheit
- immer ein Problem, wenn Funktionswert potenziell „auf mehreren Wegen“ festgelegt;
  dann klar machen:
  - entweder nur ein Weg „gangbar“
  - oder auf allen Wegen gleicher Funktionswert
- 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.

Beispiele von Homomorphismen

- ASCII-Code:
  - alle Zeichen werden durch Wörter gleicher Länge codiert
  - Längere Texte werden übersetzt, indem man zeichenweise decodiert.
- Unicode: könnte man analog zu ASCII machen, ABER: weil Unicode-Zeichensatz soviele Zeichen enthält, bräuchte man 4 Bytes pro Zeichen
- Idee: Platzsparende Codierung verwenden, sofern ein (deutscher) Text nur sehr wenige Zeichen verwendet $\rightarrow$ UTF-8
- 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
  - UTF-8 ist eine platzsparendere Codierung als Unicode

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

Überblick

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 $x \in A$.

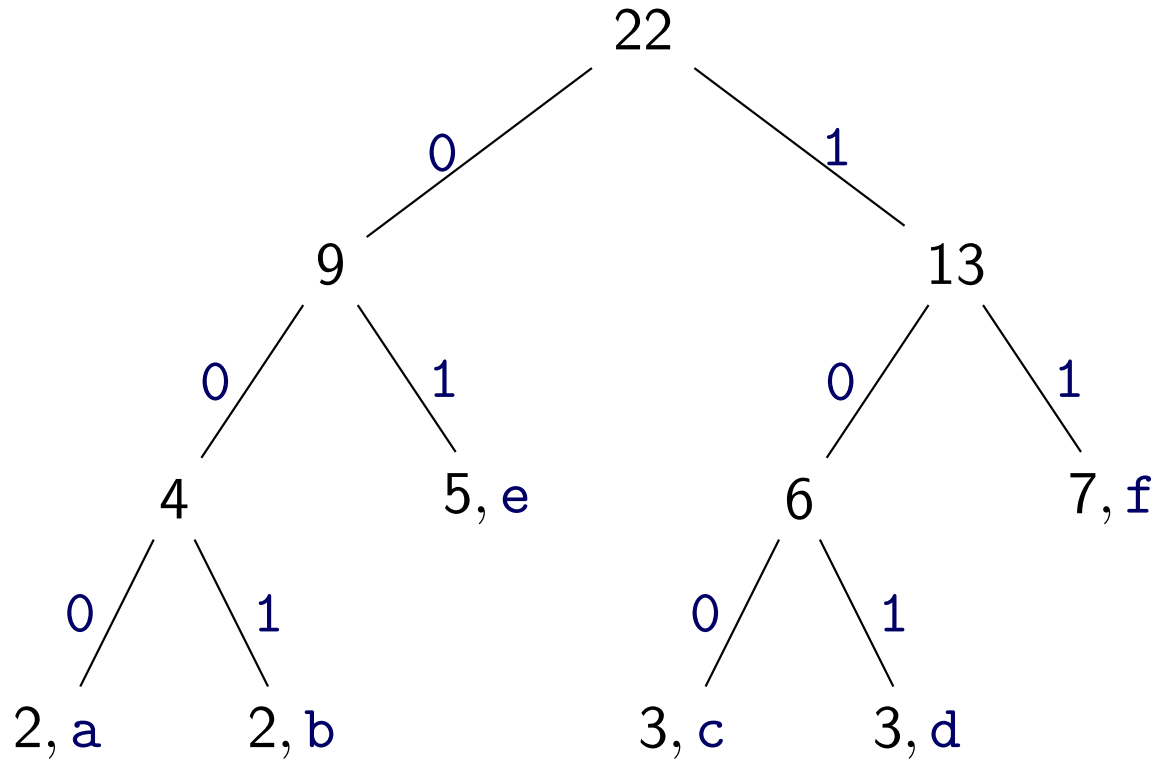
- Verwendung z. B. in Kompressionsverfahren wie gzip oder bzip2
- denn bei Huffman-Codierungen
  - häufigere Symbole durch kürzere Wörter codiert und
  - seltenere Symbole durch längere

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

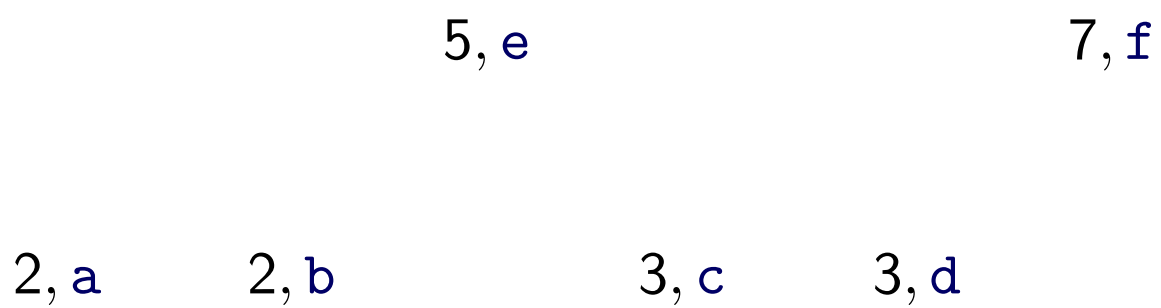
Konstruktion des Huffman-Baumes (1)

- 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 $\{(N_x(w), \{x\})\}$ für $x \in A$,
  - Als Anfang für die Konstruktion des Baumes zeichnet man für jedes Symbol einen Knoten mit Markierung $(N_x(w), \{x\})$.
- 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)

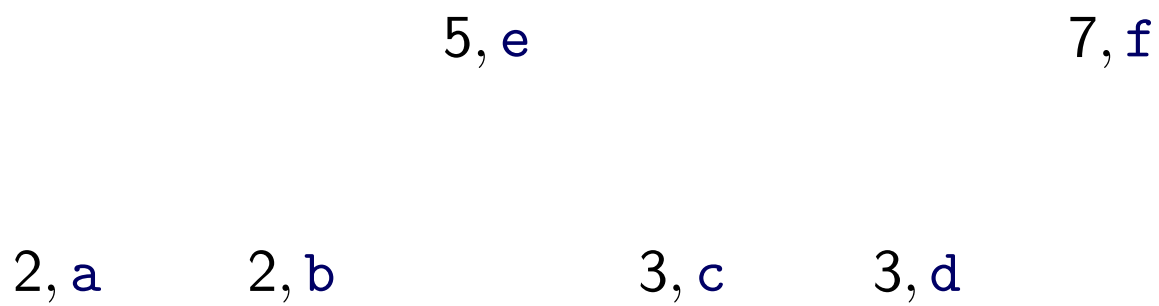
Iterationsschritt des Algorithmus:

- 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:

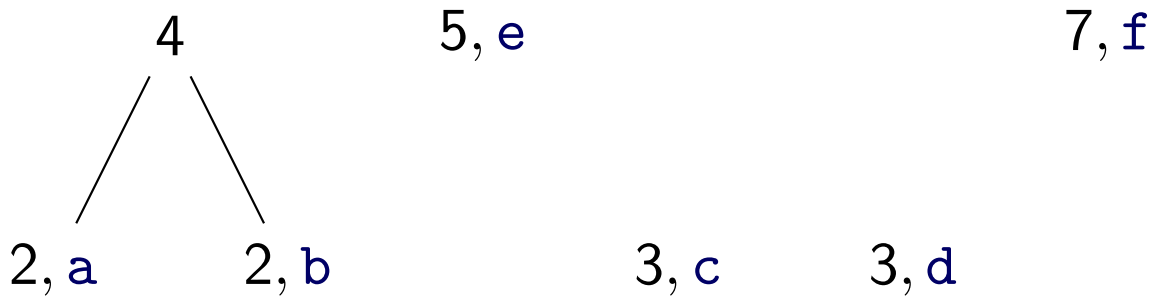
$$M_0 = \{\ (2, \{\texttt{a}\})\ ,\ (2, \{\texttt{b}\})\ ,\ (3, \{\texttt{c}\})\ ,\ (3, \{\texttt{d}\})\ ,\ (5, \{\texttt{e}\})\ ,\ (7, \{\texttt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (4)

Beispiel:

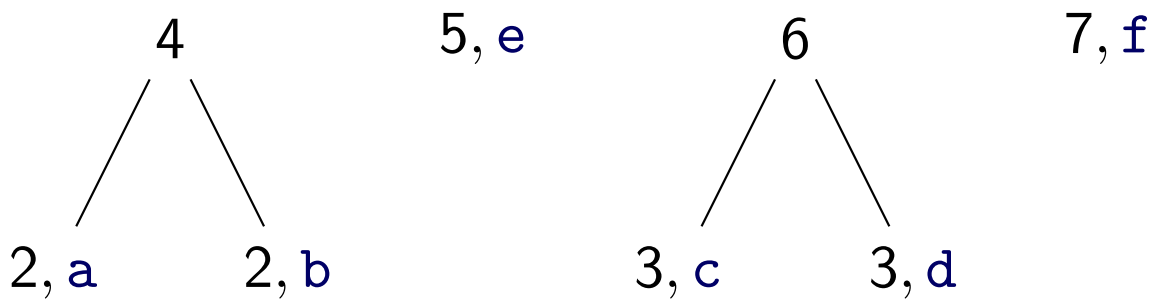
$$M_1 = \{\ (4, \{\texttt{a}, \texttt{b}\})\ ,\ (3, \{\texttt{c}\})\ ,\ (3, \{\texttt{d}\})\ ,\ (5, \{\texttt{e}\})\ ,\ (7, \{\texttt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (5)

Beispiel:

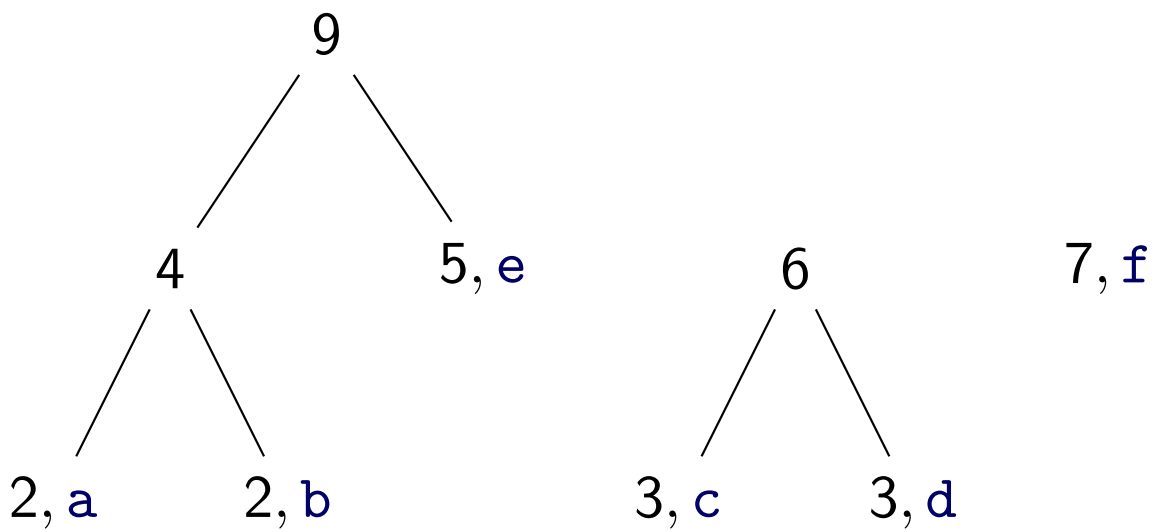
$$M_2 = \{\ (4, \{\texttt{a}, \texttt{b}\})\ ,\ (6, \{\texttt{c}, \texttt{d}\})\ ,\ (5, \{\texttt{e}\})\ ,\ (7, \{\texttt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (6)

Beispiel

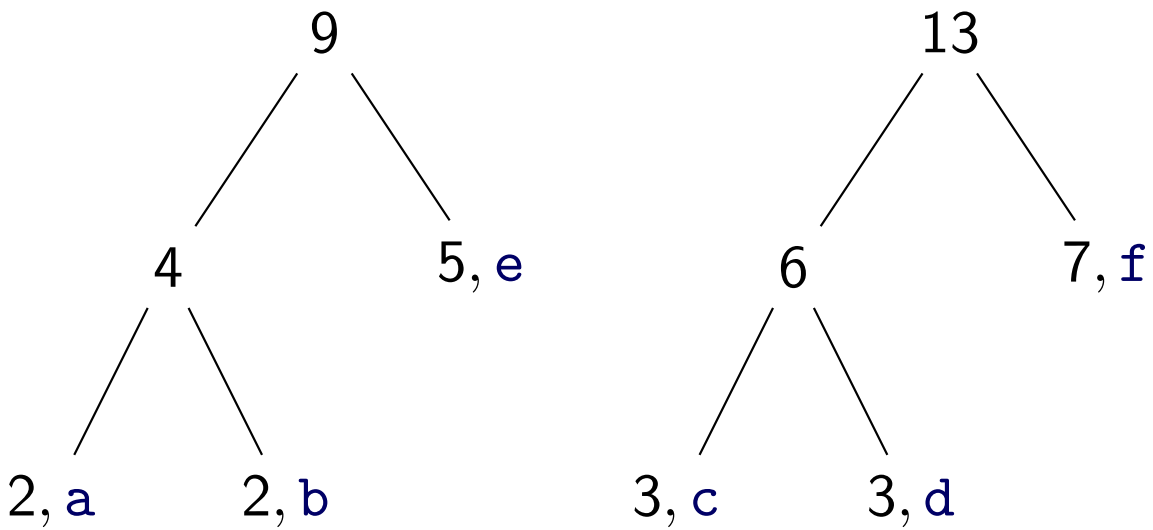
$$M_3 = \{\ (9, \{\texttt{a}, \texttt{b}, \texttt{e}\})\ ,\ (6, \{\texttt{c}, \texttt{d}\})\ ,\ (7, \{\texttt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (7)

Beispiel

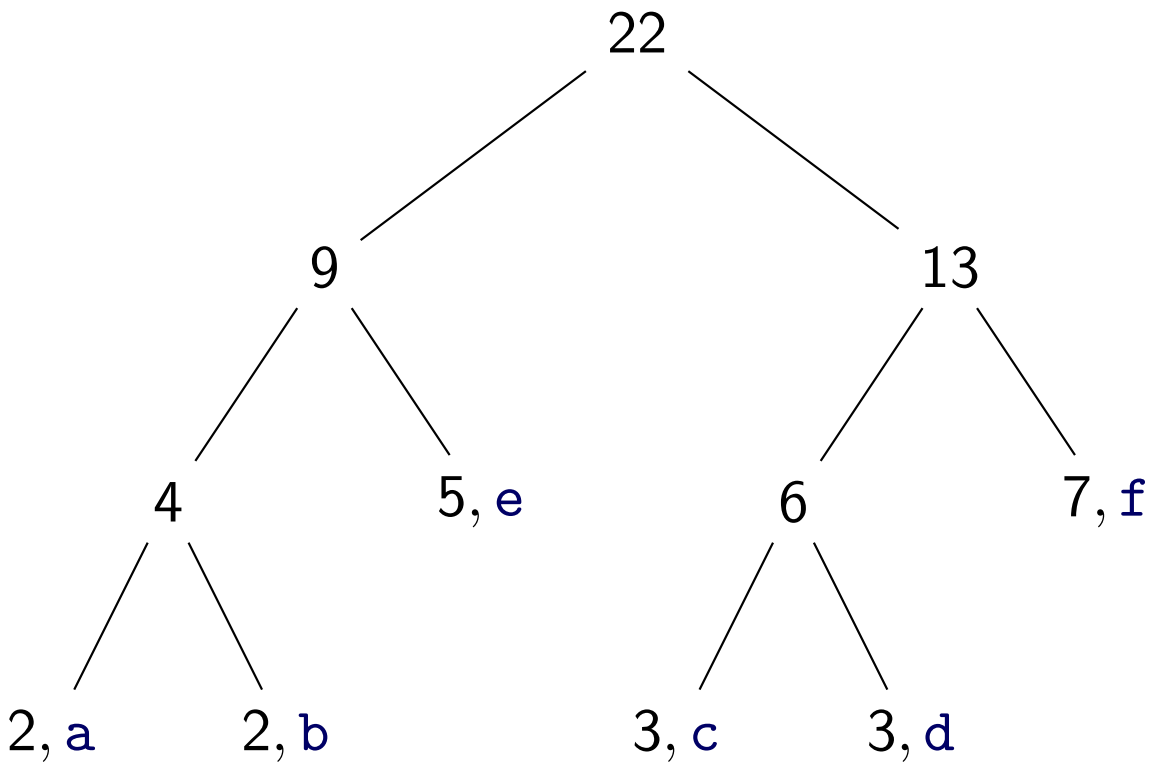
$$M_4 = \{\ (9, \{\mathtt{a}, \mathtt{b}, \mathtt{e}\})\ ,\ (13, \{\mathtt{c}, \mathtt{d}, \mathtt{f}\})\ \}$$



Konstruktion des Huffman-Baumes (8)

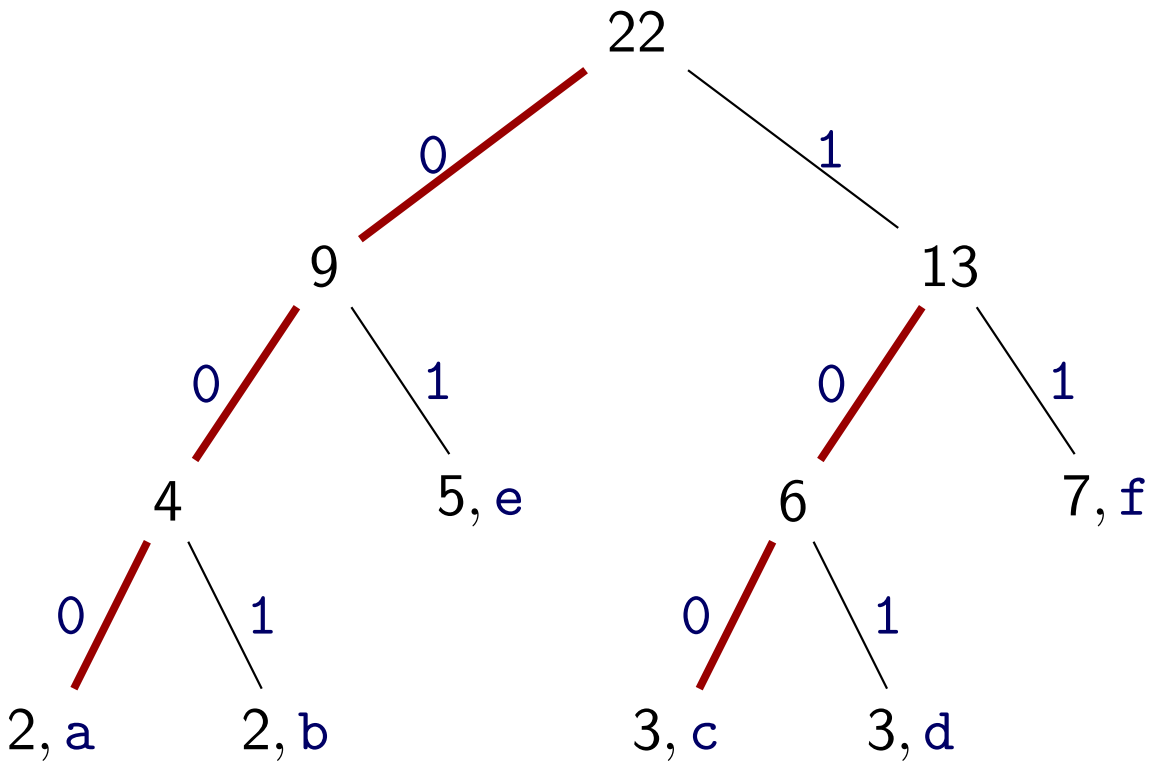
Beispiel

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



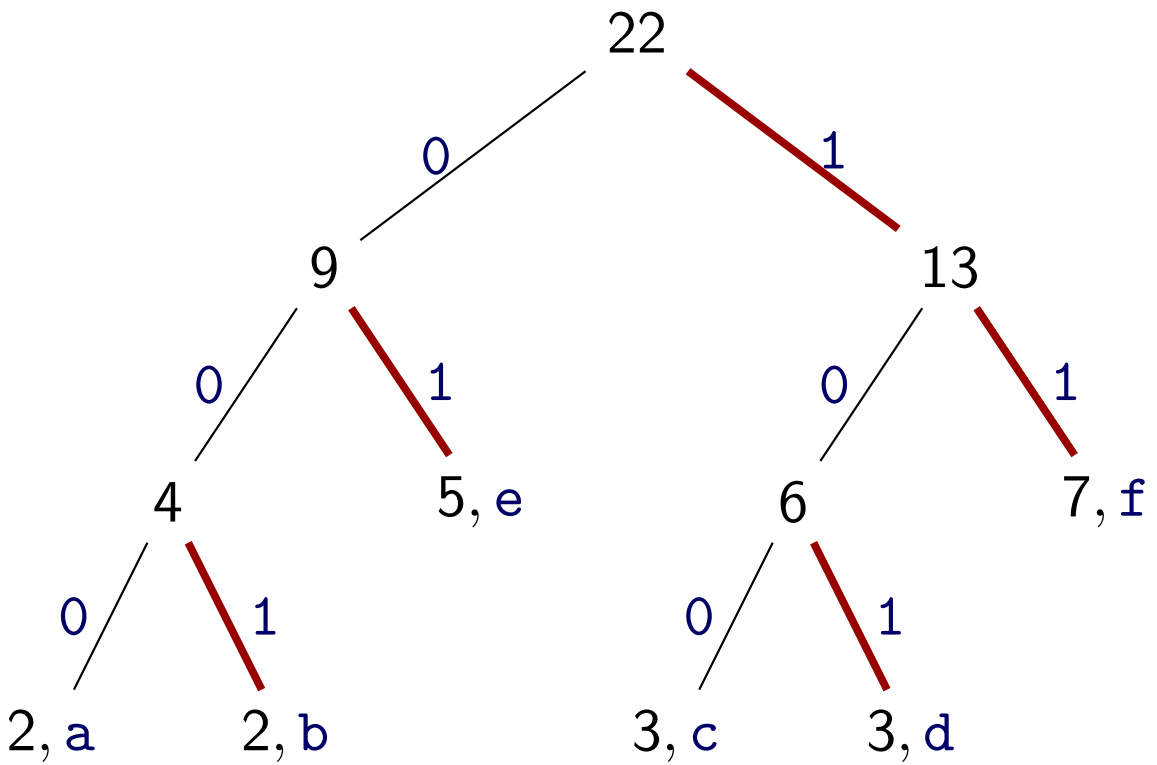
Beschriftung der Kanten

- nach links führende Kanten werden mit `0` beschriftet
- nach rechts führende Kanten werden mit `1` beschriftet



Beschriftung der Kanten

- nach links führende Kanten werden mit `0` beschriftet
- nach rechts führende Kanten werden mit `1` beschriftet

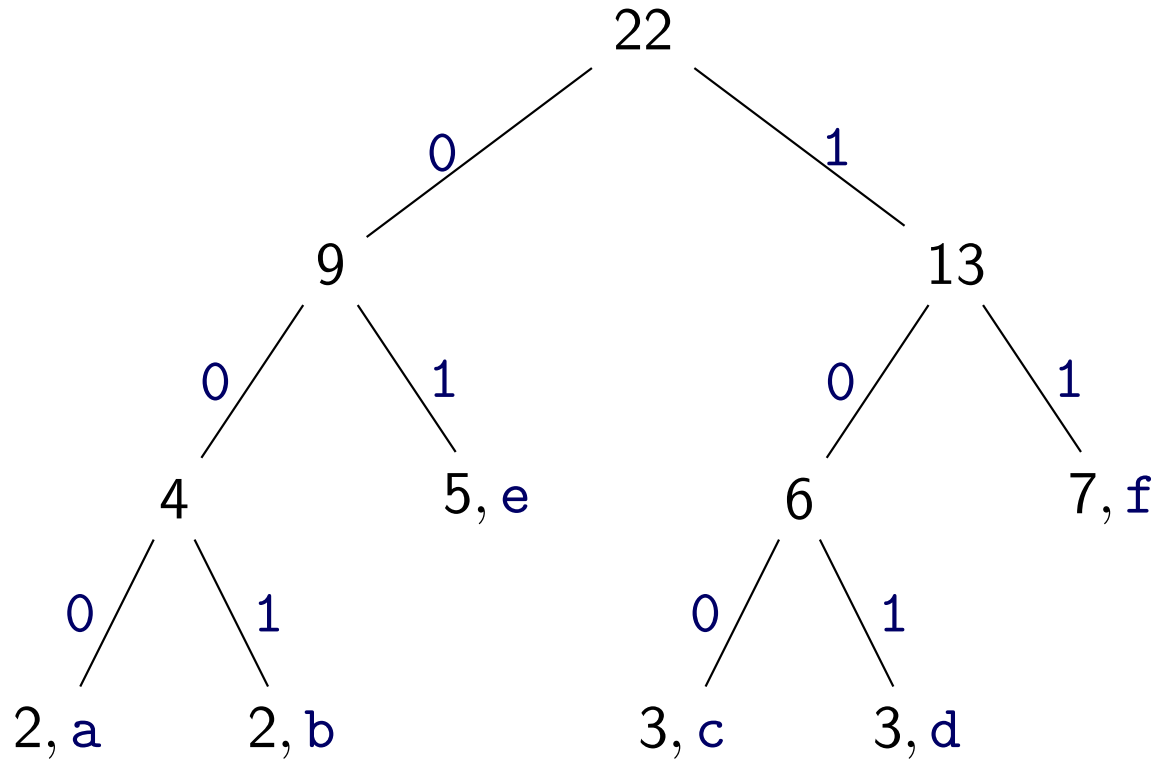


Ablesen der Codierungen

- 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

- Huffman-Code nicht eindeutig
  - im allgemeinen mehrere Möglichkeiten,
    welche zwei Knoten zu einem neuen vereinigt werden
  - im Baum links und rechts vertauschbar

- aber alle sind „gleich gut“:
  - Unter allen präfixfreien Codes führen Huffman-Codes zu kürzesten Codierungen
    des Wortes, für das die Huffman-Codierung konstruiert 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