Algorithmen und Datenstrukturen

Prof. Dr. Volker Blanz
Lehrstuhl für Medieninformatik
Universität Siegen – Fakultät IV

4 Codierung

Version: WS 20/21

4 Codierung ...

Motivation: Ziele dieses Abschnittes

- Rechner muss Daten mittels Binärsystem speichern
- Anwendungen erfordern Zahlen, Text, Bilder, Audio, Video etc.
- Codierung: Darstellung von Daten im Rechner

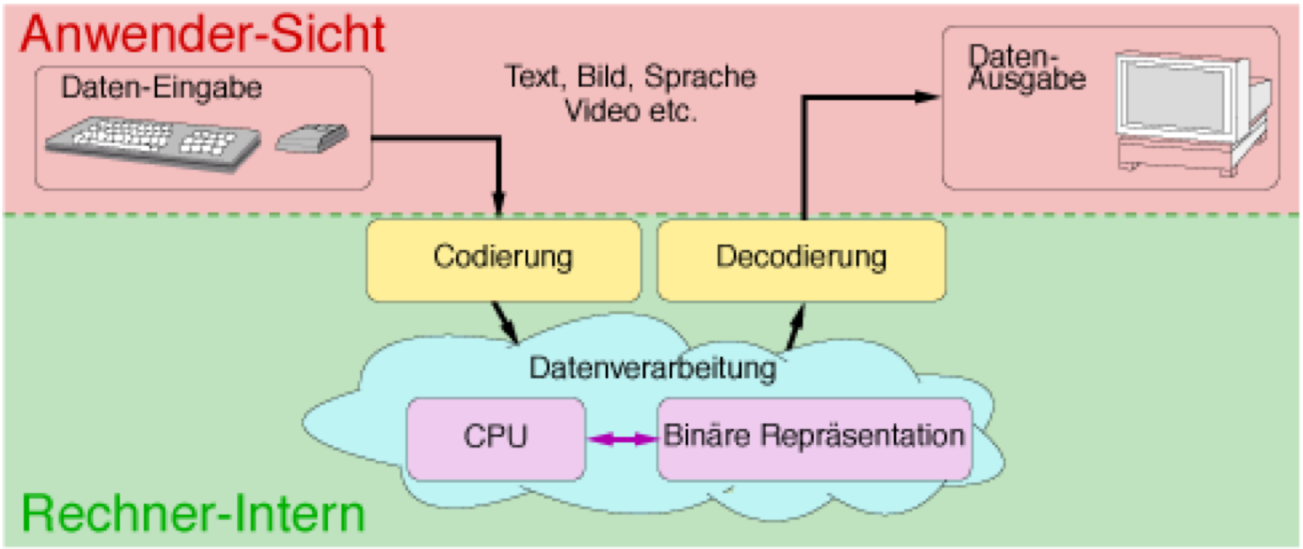
Herausforderungen:

- ungewohnte Denkweise im Umgang mit Zahlen & Zeichen
- Darstellung von Zahlen mit eingeschränkter Genauigkeit
- wichtig: Daten ≠ Information

Literatur:

- [Ernst] Auszüge aus Kapitel 1 & 3
- [Gumm] Auszüge aus Kapitel 1
- [Herold] Auszüge aus Kapitel 3 & 22
- http://www.computerbase.de/lexikon/IEEE 754
- http://www.h-schmidt.net/FloatApplet/IEEE754de.html

4 Codierung ...


Anwender-Sicht
Daten-Eingabe
Text, Bild, Sprache
Video etc.
Daten-
Ausgabe
Codierung
Decodierung
Datenverarbeitung
CPU
Binäre Repräsentation
Rechner-Intern

4.1 Zahlensysteme

Additionssystem

- Symbole (***Ziffern***) für bestimmte (Grund-)Zahlen
- konkrete Zahl ergibt sich durch Addition der Ziffern
- die Position der Ziffern ist vertauschbar (***kommutativ***)
- Probleme: Schlechte Lesbarkeit, lange Zahlen, Operationen schwierig durchführbar

Beispiel: Römisch-etruskische Zahlen

- Ziffern: $I = 1,\ V = 5,\ X = 10,\ L = 50,\ C = 100,\ \ldots$
- Konvention: Ziffern werden mit abnehmender Wertigkeit notiert
- Zusatz: Vierfaches Auftreten einer Ziffer wird durch Subtraktion umschrieben:

  $IIII = IV$ $\Rightarrow$Vertauschbarkeit geht verloren
- Beispiele: $3 = III,\ 14 = XIV,\ 149 = CIL,\ 205 = CCV$

4.1 Zahlensysteme ...

Stellenwertsystem

- geht schon auf Inder und Perser zurück
- Wichtige Kennzeichen:
  - Nutzung eines Satzes von b Ziffern für beliebige Zahlen (b ***Basis***)
  - Wertigkeit einer Ziffer ergibt sich aus ihrer Position
  - Wertigkeit steigt in der Regel von rechts nach links an

$$\text{Zahl} = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} \ldots a_1 \cdot b + a_0, a_i \in \text{Ziffernsatz}$$

$$\text{Zahl} = \sum_{i=0}^{n} a_i b^i = \left(a_n a_{n-1} \cdots a_1 a_0\right)_{\text{Basis } b}$$

- Wichtige Beispiele: Dezimalsystem (b=10), Duodezimalsystem (b=12), Dualsystem (Binärsystem, b=2), Hexadezimalsystem (b=16), Oktalsystem (b=8)

4.1 Zahlensysteme ...

Beispiel: Dezimalsystem

Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

10^4	10^3	10^2	10^1	10^0
5	6	3	8	6

$$56386_{10} = 10^4 \cdot 5 + 10^3 \cdot 6 + 10^2 \cdot 3 + 10^1 \cdot 8 + 10^0 \cdot 6$$

Beispiel: Binärsystem

Ziffern: 0, 1

2^4	2^3	2^2	2^1	2^0
1	0	1	1	0

$$10110_2 = 2^4 \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot 1 + 2^1 \cdot 1 + 2^0 \cdot 0 = 22_{10}$$

Beispiel: Oktalsystem

Ziffern: $0, 1, 2, 3, 4, 5, 6, 7$

8^4	8^3	8^2	8^1	8^0
7	3	2	6	7

$$73267_8 = 8^4 \cdot 7 + 8^3 \cdot 3 + 8^2 \cdot 2 + 8^1 \cdot 6 + 8^0 \cdot 7 = 30391_{10}$$

Beispiel: Hexadezimalsystem

Ziffern: $0, \ldots, 9, A(10), B(11), C(12), D(13), E(14), F(15)$

16^4	16^3	16^2	16^1	16^0
A	9	5	F	F

$$A95FF_{16} = 16^4 \cdot A + 16^3 \cdot 9 + 16^2 \cdot 5 + 16^1 \cdot F + 16^0 \cdot F = 693759_{10}$$

4.1 Zahlensysteme ...

Motivation: Wozu andere Stellenwertsysteme?

Dezimalsystem ungeeignet für elektronische Rechner

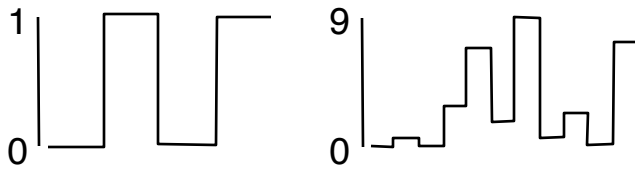
- ❍ Rechner stellt nur zwei Zustände dar (Bit): Spannung liegt an oder nicht
- ❍ Bauteile für zwei Zustände sind einfach, günstig, robust und platzsparend realisierbar
- ❍ Bezug zu Aussagenlogik: 0 = false, 1 = true

  ⇒Operationen wie <,≤,>,≥, ≠, . . . sind direkt integrierbar

Theoretisch: Mehr Zustände unterscheidbar

⇒ Aber: Bereits kleine Spannungsschwankungen führen dann zu falschen Ergebnissen

- ❍ 2 Zustände: 50% Spannungsschwankung→Fehler
- ❍ 10 Zustände: 5% Spannungsschwankung→Fehler



4.1 Zahlensysteme ...

Oktal- und Hexadezimalsystem: Warum sind diese von Bedeutung?

- Basen 8 und 16 sind 2er-Potenzen

  ⇒Umrechnungen zwischen diesen Systemen besonders einfach

- Darstellungen von Binärdateien im Oktal- oder Hexadezimalsystem

  sind deutlich kompakter & lesbarer

Beispiel: Vergleichen Sie folgende Zahlenpaare

$$11101011110110011101110111111010111_2$$
$$11101011110110011101111011111010111_2$$

$$727547357727_8$$
$$727547367727_8$$

$$EBD9DDFD7_{16}$$
$$EBD9DEFD7_{16}$$

Umrechnung: Dezimal→Binär

Vorüberlegung: Wie erfolgt die Umrechnung Binär nach Dezimal ?

$$10110_2 = 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 16 + 4 + 2 = 22_{10}$$

oder wahlweise auch umgeformt in das folgenden Schema:

$$= \left(1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\right)\cdot 2 + 0$$

$$= \left(\left(1\cdot 2^2 + 0\cdot 2^1 + 1\right)\cdot 2 + 1\right)\cdot 2 + 0$$

$$= \left(\left(\left(1\cdot 2 + 0\right)\cdot 2 + 1\right)\cdot 2 + 1\right)\cdot 2 + 0$$

d.h.

$$22 = \left(11\right)\cdot 2 + 0$$

$$= \left(\left(5\right)\cdot 2 + 1\right)\cdot 2 + 0$$

$$= \left(\left(\left(2\right)\cdot 2 + 1\right)\cdot 2 + 1\right)\cdot 2 + 0$$

$$= \left(\left(\left(\left(1\right)\cdot 2 + \mathbf{0}\right)\cdot 2 + \mathbf{1}\right)\cdot 2 + \mathbf{1}\right)\cdot 2 + \mathbf{0}$$

Also: eine Abfolge von **Operationen „mal 2 plus Rest** 0 oder 1“.
Der als **letztes** addierte Rest sind die Einer-Stellen 2^0.

Die Umkehrung erfolgt dann durch eine Abfolge von **Operationen „Dividiere durch 2 mit Rest“**.
Der als **erstes** verbleibende Rest sind die Einer,
der als letztes verbleibende Rest bezeichnet das höchstwertige Bit.

Umrechnung: Dezimal→Binär

Vorgehen: Division mit Rest (Modulo-Rechnung)

Beispiel:

$$22_{10} = ?_2$$

$$\begin{aligned}
22 &: 2 = 11 \quad \text{Rest } 0 \\
11 &: 2 = 5 \quad \text{Rest } 1 \\
5 &: 2 = 2 \quad \text{Rest } 1 \\
2 &: 2 = 1 \quad \text{Rest } 0 \\
1 &: 2 = 0 \quad \text{Rest } 1
\end{aligned}$$

$$\Rightarrow 22_{10} = 10110_2$$

Vorsicht: Reihenfolge der Ziffern im Ergebnis wird oft falsch angenommen!
Der Rest der ersten Division bezeichnet die Einer, steht also ganz **rechts** !

Vorsicht: Einstieg und Ausstieg in das Schema: Am Anfang erst dividieren, dann Rest bilden oder umgekehrt (letzteres), was passiert am Schluss mit einer 2? Rest 0 und fertig oder noch ein nächsthöheres Bit setzen (letzteres). In der Informatik ist immer der Ein- und Ausstieg in Schleifen eine Fehlerquelle.

Einschub 1: Modulo

Modulo-Funktion, Gleichheit "modulo" n

$$21 = 2 \cdot 8 + 5$$

$$21 = 2 \cdot 8 \quad \text{Rest} \quad 5$$

$$21 \bmod 8 = 5$$

Allgemein: a mod n = b

genau dann, wenn gilt: Es existiert eine natürliche Zahl c, sodass $a = c \cdot n + b, \quad 0 \le b < n$

Insbesondere: a mod n = 0 genau dann, wenn a ein Vielfaches von n ist.

Gleichheit modulo n: Es ist $(21 = 5)_{\text{mod } 8}$

Allgemein: $(p = q)_{\text{mod } n}$ genau dann, wenn (p mod n) = (q mod n)

dann gilt: $(p - q)_{\text{mod } n} = 0$, also: p - q ist Vielfaches von n.

Einschub 2: Summen von Potenzen

Bemerkung zu Binären Zahlen:

Allgemein gilt die Geometrische Summenformel
(Beweis z.B. durch Induktion oder einen einfachen Trick...)

$$\sum_{i=0}^{n} b^i = \frac{b^{n+1}-1}{b-1}$$

Insbesondere für b=2:

$$\sum_{i=0}^{n} 2^i = 2^{n+1}-1$$

$$\sum_{i=0}^{n} 2^i + 1 = 2^{n+1}$$

$$1111_2 + 1 = 10000_2$$

Addition Binärer Zahlen

Vorgehen: Stellenweise Addition, beginnend mit niedrigster Stelle, mit Übertrag

Beispiel: $10110_2 + 10100_2$

```
      1 0 1 1 0
+     1 0 1 0 0
    1 - 1 - - -   Übertrag
=   1 0 1 0 1 0
```

Multiplikation Binärer Zahlen

Vorgehen: Weiterrücken der Binärstellen, dann Summe der Komponenten

Beispiel:

$$10110_2 \cdot 10100_2 = 10110_2 \cdot (10000_2 + 100_2)$$
$$= 10110_2 \cdot 10000_2 + 10110_2 \cdot 100_2$$

$$\begin{array}{rl} & 1\,0\,1\,1\,0\,.\,1\,\mathbf{0}\,\mathbf{0} \quad = \quad 1\,0\,1\,1\,0\,\mathbf{0}\,\mathbf{0} \\ + & 1\,0\,1\,1\,0\,.\,1\,\mathbf{0}\,\mathbf{0}\,\mathbf{0}\,\mathbf{0} = 1\,0\,1\,1\,0\,\mathbf{0}\,\mathbf{0}\,\mathbf{0}\,\mathbf{0} \\ = & \qquad\qquad\qquad\quad 1\,1\,0\,1\,1\,1\,0\,0\,0 \end{array}$$

Das Weiterrücken gilt ganz allgemein bei Multiplikation mit Zahlen der Form 100… , denn

$$x \cdot (1 \cdot b^k) = \sum_{i=0}^{n} a_i b^i \cdot b^k = \sum_{i=0}^{n} a_i b^{i+k} = (a_n a_{n-1} \cdots a_1 a_0 0 \cdots 0)_{\text{Basis } b}$$

Umrechnung: Hexadezimal Binär

Vorgehen: Einfaches Aneinanderreihen bzw. Blockbildung von Binärzahlen

Beispiel:

$$A95FF_{16} = ?_2$$

$$\begin{aligned}
A_{16} &= 1010_2 \rightarrow \cdot 16^4 \rightarrow \cdot 2^{4\cdot 4} \\
9_{16} &= 1001_2 \rightarrow \cdot 16^3 \rightarrow \cdot 2^{3\cdot 4} \\
5_{16} &= 0101_2 \rightarrow \cdot 16^2 \rightarrow \cdot 2^{2\cdot 4} \\
F_{16} &= 1111_2 \rightarrow \cdot 16^1 \rightarrow \cdot 2^{1\cdot 4} \\
F_{16} &= 1111_2 \rightarrow \cdot 16^0 \rightarrow \cdot 2^{0\cdot 4}
\end{aligned}$$

$$\Rightarrow A95FF_{16} = \underbrace{1010}_{A}\ \underbrace{1001}_{9}\ \underbrace{0101}_{5}\ \underbrace{1111}_{F}\ \underbrace{1111}_{F}{}_2$$

4.2 Zahlentypen

Ziel: Darstellung verschiedener Zahlenarten (natürliche Zahlen, ganze Zahlen, reelle Zahlen, . . .) und Umsetzung von Operationen (Addition)

Zahlentypen sind ***eingeschränkte Darstellungen*** von Zahlenarten:

- ***fixe Anzahl*** von N Bits, d.h. nur 2^N Werte darstellbar

⇒ ***einschränkter Wertebereiche*** und ***endliche viele Zahlenwerte***

⇒ nicht alle Zahlen sind im Rechner darstellbar

Zahlentyp entscheidet über

- die Interpretation von Werten (Zahlenart),
- die Genauigkeit der Darstellung und
- die minimale und maximale darstellbare Zahl

4.2.1 Repräsentation von Ganzen Zahlen

Natürliche Zahlen

- Repräsentation im Binärsystem (s.o.)
- übliche Längen N: 16, 32 oder 64 bits
- Typename in C/C++:
  - unsigned int: 32 bit (unsigned=vorzeichenlos, integer=Ganzzahl)
  - unsigned short: 16 bit
  - unsigned long: 64 bit
- Wertebereich: $0, \ldots, 2^{N-1}$

Ganze Zahlen

- Typename in C/C++: int, short, long (gleiche Anzahl bits wie oben)
- Wertebereich: Hängt von der konkreten Repräsentation ab

  Beachte: Zur Wahl des optimalen Typs muss stets Wertebereich und Speicherplatz/Geschwindigkeit abgewogen werden.

4.2.1 Repräsentation von Ganzen Zahlen ...

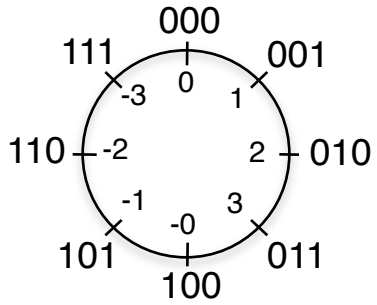
Vorzeichen-Betrag Darstellung

- „Naiver“ Ansatz:
  - Höchstwertiges Bit bestimmt Vorzeichen (0 positiv, 1 negativ)
  - Restliche Bits werden zur Codierung des Betrags verwendet
- Probleme:
  - Darstellung nicht eindeutig wegen der doppelt repräsentierten Null (+0,–0)
  - Aufwändige Umsetzung der Addition

Beispiel:

$$6_{10} = 0110_2$$
$$-6_{10} = 1110_{VZB}$$

VZB=vorzeichenbehaftet



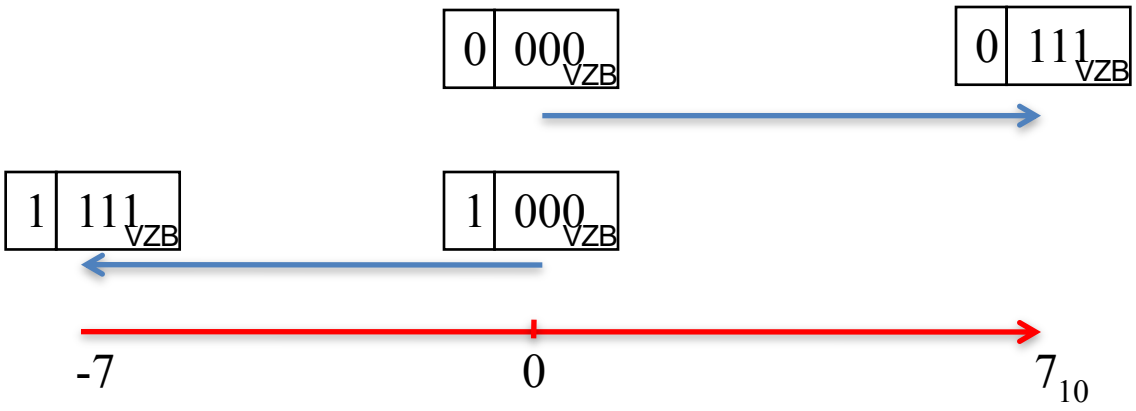
Zahlenkreis für 3 Bits

$$111_{VZB} = -(11_2) = -(2+1) = -3$$

4.2.1 Repräsentation von Ganzen Zahlen ...

Vorzeichen-Betrag Darstellung

Bei gesetztem (negativem) Vorzeichenbit wird von den übrigen Stellen in die negative Richtung gezählt.



(0 ist doppelt repräsentiert)

4.2.1 Repräsentation von Ganzen Zahlen ...

Addition und Subtraktion bei Vorzeichen-Betrag

Addition: 1. beide Zahlen haben ***dasselbe Vorzeichen*** ⇒ Bitweise Addition der nachfolgenden Bits,

VZ-Bit bleibt, (Overflow der nachfolgenden Bits wird ignoriert und liefert dann einen Fehler)

2. beide Zahlen haben ***unterschiedliche Vorzeichen*** ⇒ Bitweise Subtraktion der Beträge, VZ von betragsgrößerer Zahl

Subtraktion: Addition mit der ***Gegenzahl***, z.B. 5 – 7 = 5 + (–7)

Beispiele mit 3 Wert-Bits

	5		0101		–3		1011		–5		1101		–5		1101
+	2	+	0010	+	–4	+	1100	+	2	+	0010	–	2	+	1010
	7		0111		–7		1111_{VZB}=-7		–3		1011		–7		1111

VZ bleibt
101+010=111

VZ der betragsgrößeren Zahl. 101-010=011.

-5-2=-5+(-2)
101+010=111

Nachteil: Die Addition bei unterschiedlichen VZ erfordert eine (ungünstige) Fallunterscheidung

4.2.1 Repräsentation von Ganzen Zahlen ...

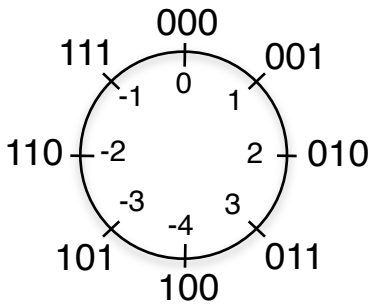
Zweierkomplement

- Gebräuchliche Darstellung ganzer Zahlen
- Typename in C/C++: int (32 bit), short (16 bit), long (64 bit)
- Darstellung für $N = n+1$ bits $b_n b_{n-1} \ldots b_1 b_0$

$$\text{Zahl} = -b_n \cdot 2^n + b_{n-1} \cdot 2^{n-1} + \ldots + b_1 \cdot 2 + b_0 = -b_n \cdot 2^n + \sum_{i=0}^{n-1} b_i \cdot 2^i$$

$\Rightarrow b_n = 1 \Leftrightarrow$ negative Zahl, da $2^n > \sum_{i=0}^{n-1} 2^i$

$\Rightarrow$ die Zahlen sind ***modulo 2^{n+1}*** kontinuierlich aufgetragen. $a \bmod b$ = Rest bei ganzzahliger Division a/b.



Zahlenkreis für 3 Bits

- Hinweis: Für ganze Zahlen a, b ∈ ℤ gilt

$$(a = b)_{\bmod N} \Leftrightarrow (a - b) \bmod N = 0$$

bzw. a/N und b/N haben denselben positiven Rest, z.B. $(12 = 4)_{\bmod 8}$

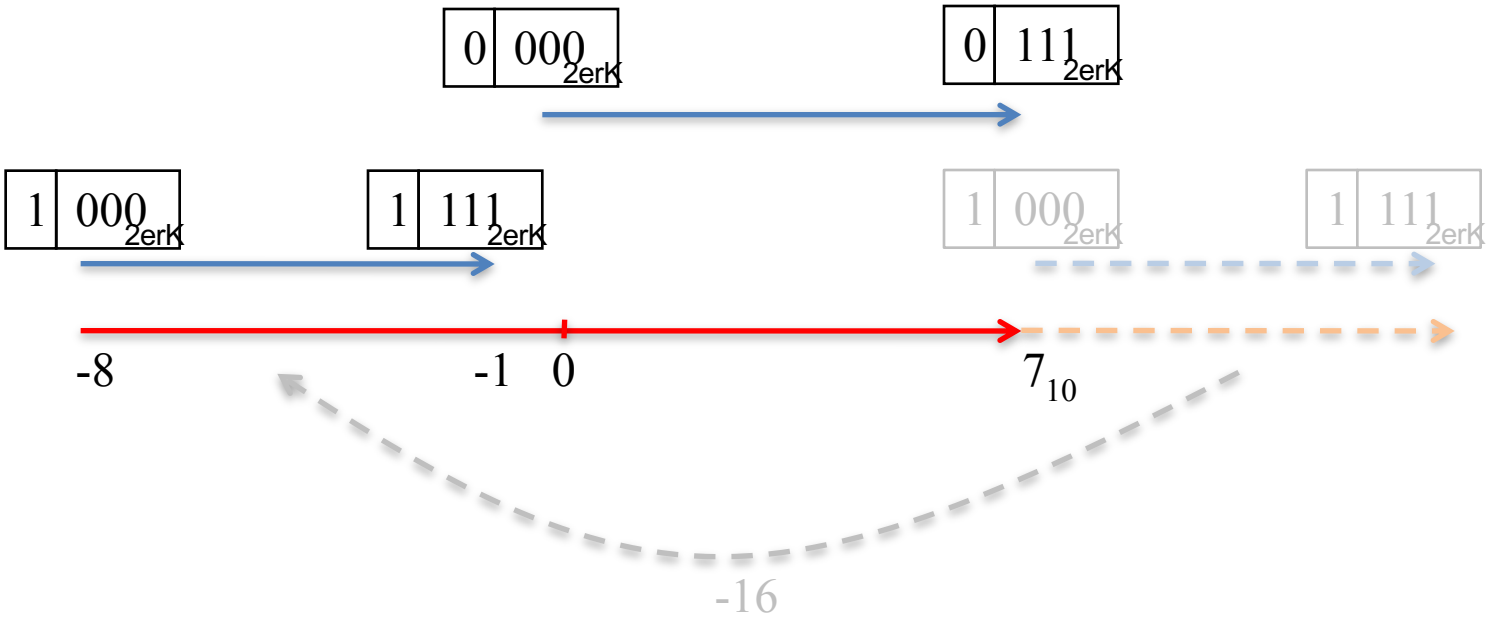
$$(3 + 1) = 4 = -4 \bmod 8$$

$$111_{VZB} = -4 + (11_2) = -4 + 3 = -1$$
$$110_{VZB} = -4 + (10_2) = -4 + 2 = -2$$
$$100_{VZB} = -4 + (00_2) = -4 + 0 = -4$$

4.2.1 Repräsentation von Ganzen Zahlen ...

Zweierkomplement-Darstellung

Das Vorzeichenbit verschiebt die Skala der übrigen Bits
durch Subtraktion statt Addition von 2^n um 2^{n+1} ins Negative.



4.2.1 Repräsentation von Ganzen Zahlen ...

Vorzeichenwechsel durch Zweierkomplement

Gegeben eine Zahl z, wie bekomme ich –z innerhalb der Zweierkomplementdarstellung ?

Regel: Invertiere Zahl bitweise (Schreibweise: $a \rightarrow \overline{a}$) und addiere 1, z.B.

$$6_{10} = 0110_2 \qquad \textbf{Überprüfung}: (--6 = 6)$$

$$\begin{aligned} -6_{10} &= \overline{0110_2} + 1_2 \\ &= 1001_2 + 1_2 \\ &= 1010_{2er-K} \end{aligned} \qquad \begin{aligned} -1010_{2er-K} &= \overline{1010_2} + 1_2 \\ &= 0101_2 + 1_2 \\ &= 0110_2 = 6_{10} \end{aligned}$$

Begründung:

$$\text{Sei } z_{2er-K} = b_n b_{n-1} \ldots b_1 b_0 \text{, dann}$$

$$z_{2er-K} + \overline{z_{2er-K}} = 11\ldots11_{2er-K} = -2^n + \underbrace{2^{n-1} + \ldots + 2 + 1}_{=2^n-1} = -1$$

$$z_{2er-K} = -\overline{z_{2er-K}} - 1$$

$$\Rightarrow -z_{2er-K} = \overline{z_{2er-K}} + 1$$

4.2.1 Repräsentation von Ganzen Zahlen ...

Addition und Subtraktion bei Zweierkomplement

Addition: Bitweise (Overflow beim Vorzeichenbit bei zwei negativen Zahlen wird ignoriert und liefert dennoch richtiges Ergebnis, solange Wertebereich {-8, ..., 7} nicht überschritten.)

Subtraktion: Rückführung auf Addition der negativen Zahl

1. Negation der zweiten Zahl mittels Zweierkomplement+1
2. Addieren: Erste Zahl mit Zweierkomplement der zweiten Zahl

Beispiele mit $N = 4$, $n = 3$ Bits

	5		0101		−3		1101		−5		1011		−5		1011
+	2	+	0010	+	−4	+	1100	+	2	+	0010	−	2	+	1110
	7		0111		−7		$1001_{2er\text{-}K} = -7$		−3		1101		−7		1001

$$-1001_{2er\text{-}K} = 0110_{2er\text{-}K} + 1$$
$$= 0111_{2er\text{-}K} = 7$$

Bemerkung: ❍ Addition und Subtraktion bauen nur auf bitweiser Addition und Komplementbildung auf und benötigen keine Fallunterscheidung

❍ Zweierkomplementdarstellung hat keine mehrdeutige Null

Addition negativer Zahlen im Zweierkomplement

Beobachtung: bei 2 negativen Zahlen sind jeweils die führenden bits gesetzt,
also entsteht in der Summe immer ein overflow über die erlaubte Bit-Anzahl hinaus.
Rechenregel: ignoriere dies, das overflow-bit läuft ins Leere.

Beispiel 1:

$(-4_{10}) + (-4_{10}) = 1100_{2erKompl} + 1100_{2erKompl} = 1\ 1000_{2erKompl} = 1000_{2erKompl} = (-8_{10})$

Beispiel 2:

$(-8_{10}) + (-8_{10}) = 1000_{2erKompl} + 1000_{2erKompl} = 1\ 0000_{2erKompl} = 0000_{2erKompl} = (0_{10})$

... also falsch, da -16 ausserhalb des Wertebereiches liegt.
Aber immerhin richtig modulo 16, also nicht ganz sinnlos: $(0 = -16)$ **mod 16**

Also: bei negativen Zahlen, deren Summe nicht aus dem Wertebereich führt,
sind die nicht-führenden, also positiv gezählten Bits zusammen so groß, dass sie
in der Summe im führenden Bit eine 1 hervorrufen und damit wieder (richtigerweise)
eine negative Zahl ergeben.

4.2.1 Repräsentation von Ganzen Zahlen ...

Addition von Zahlen im Zweierkomplement

Auch bei nicht-negativen Zahlen kann ein overflow auftreten, und zwar:

- vom führenden Bit ins Leere,
- von den übrigen Bits ins führende Bit.

Dabei gilt aber:

- die nicht-führenden Bits zeigen immer ins Positive -> Richtung stimmt immer
- modulo 2^{n+1} ist das Ergebnis nie falsch

also:

Sofern die Rechnung nicht ohnehin aus dem Wertebereich führt, bleibt sie korrekt.

4.2.1 Repräsentation von Ganzen Zahlen ...

Biased-Notation

- 00..0 repräsentiert die kleinstmögliche Zahl, 11..1 die größtmögliche
- Auf jede Zahl wird die sogenannte ***Magic Number*** addiert und das Ergebnis binär dargestellt.
- **Beispiel: 8-bit Zahl $(N = 8)$, Magic Number: 127**

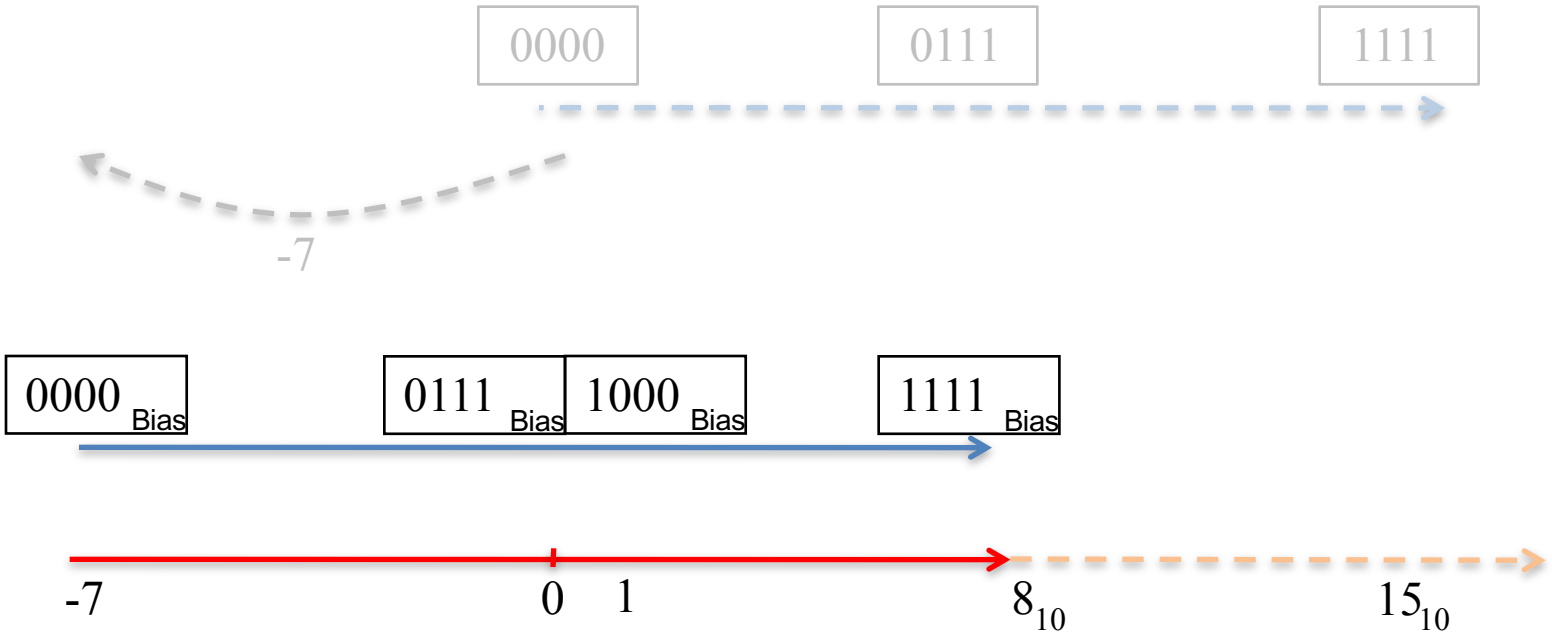
	Zahl	Berechnung der Binärdarstellung	Biased Notation
Kleinste Zahl	-127	-127+127=0	00000000_{biased}
Größte Zahl	128	128+127=255	11111111_{biased}
Null:	0	0+127=127	01111111_{biased}

- Vorteil:
  - Größenvergleiche können bitweise durchgeführt werden
- Problem:
  - Mathematische Operationen können nicht mehr direkt durchgeführt werden

4.2.1 Repräsentation von Ganzen Zahlen ...

Biased Notation

Der gesamte Zahlenbereich wird um die Magic Number verschoben.
Bei einer 4bit Zahl wäre die Magic Number 7 (= Mitte des Zahlenbereiches 0-16).
Dann folgende Situation:



4.2.2 Repräsentation von Gleitkommazahlen

Gleitkommazahlen

- Bezeichnungen: float
- Länge: 32 Bit =4 Byte (auch 16 Bit half oder 64 Bit double)
- Wertebereich: Siehe folgende Folien
- Definiert im IEEE 754 Standard
- Beachte: Operationen auf Gleitkommazahlen sind ***wesentlich*** aufwändiger und damit langsamer als für ganze Zahlen

Grundsätzliche Idee

Verwendung der wissenschaftlichen Schreibweise (***Scientific notation***)

Beispiele im Dezimalsystem:

$$1214.31 = \underbrace{1.21431}_{\textit{Mantisse}} \cdot 10^{3} \leftarrow \textit{Exponent}$$

$$0.0213178 = 2.13178 \cdot 10^{-2}$$

4.2.2 Repräsentation von Gleitkommazahlen ...

Wissenschaftliche Notation im Dezimalsystem:

Nach welcher Regel ist der Exponent zu wählen?

$$0.0213178 = 0.213178 \cdot 10^{-1} = 2.13178 \cdot 10^{-2} = 21.3178 \cdot 10^{-3}$$

Regel: Wähle Exponenten so, dass für die Mantisse m gilt: $1 \leq m < 10$

$$10^0 \leq m < 10^1$$

Exponentialdarstellung bezüglich der Basis 2:

$$4.8_{10} = 1.2_{10} \cdot 2^2 = (1.0011\ldots_2) \cdot 2^2 = m \cdot 2^e$$

Dabei gilt für die Mantisse nun

$$1 \leq m < 2$$

Beobachtung: In der Binärdarstellung von m ist damit das erste (Einer-)Bit immer gesetzt.

Einschub: Binäre Kommazahlen

Verwende Zweierpotenzen auch mit negativem Exponenten

2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

$1\ 0\ 1\ .\ 1\ 0\ 0\ 1_2 = 4 + 1 + 0.5 + 0.0625 = 5.5625$

$2^0 = 1$, $2^{-1}=1/2=0.5$ $2^{-2} = ¼ =0.25$ $2^{-3} = 1/8 = 0.125$ ...

Insbesondere sind Zahlen der Form

$1.xxxxxx_2$ immer in [1,2[und

$0.xxxxxx_2$ immer in [0,1[, da für die Zahl $1.11111..._2$ gilt:

$$\sum_{k=0}^{n} \left(\frac{1}{2}\right)^k = \frac{\left(\frac{1}{2}\right)^{n+1} - 1}{\frac{1}{2} - 1} \rightarrow \frac{0-1}{-\frac{1}{2}} = 2 \quad \text{wenn } n \rightarrow \infty.$$

4.2.2 Repräsentation von Gleitkommazahlen ...

Gleitkommazahl-Repräsentation (IEEE 754)

Aufbau: 1 Bit Vorzeichen (s), 8 Bit Exponent (e), 23 Bit Mantisse (m)

Vorzeichen: 0 positiv, 1 negativ

Exponent: Biased Notation mit Magic Number 127 (Hinweis: Im Lehrbuch [Ernst] steht hier fälschlicherweise 128)

Mantisse: Im Dualsystem immer zwischen 1 und 2, also erstes Bit immer gesetzt.

⇒Stelle vor dem Komma ist immer 1, kann also weggelassen werden

Dargestellte Gleitkommazahl:

$$(-1)^s \cdot (1.0 + m) \cdot 2^{e-127}$$

.

4.2.2 Repräsentation von Gleitkommazahlen ...

Konvertierung einer Gleitkommazahl nach IEEE 754

Gegeben: Gleitkommazahl der Form $\pm\left(g_{10}+r_{10}\right) \quad mit: g_{10} \in \mathbb{N}\ ; r_{10} \in \mathbb{R}\ , 0 \le r_{10} < 1.$

Am Beispiel von $25.421875: g_{10}=25, r_{10}=0.421875,$ Vorzeichen $s=0$

1. Bilde g_{10} auf einen binären Wert g_2 ab: $25_{10}=11001_2$
2. Bilde r_{10} auf einen binären Wert r_2 ab (s.u): $0.421875_{10}=0.011011_2$
3. Bilde vorläufige Mantisse $m=g_2+r_2$

   $m=11001_2+0.011011_2=11001.011011_2$
4. Forme m in wissenschaftliche Schreibweise um

   $m=11001.011011_2=1.1001011011\cdot 2^4\left(e=4\right)$
5. Endgültige Mantisse durch Weglassen der ersten 1: $m=1001011011$
6. Endgütliger Exponent durch Addieren von 127

   $e=4_{10}+127_{10}=100_2+01111111_2=10000011_{\text{biased}}$

4.2.2 Repräsentation von Gleitkommazahlen ...

Gesamtergebnis

$$25.421875_{10} = 0\ 10000011\ 10010110110000000000000_{\text{IEEE 754}}$$

Umwandlung von Nachkommastellen (Dezimal→Binär)

Beispiel: 0.421875_{10} (Nachkommazahl von oben)

Achtung: Direkte Umwandlung von 421875_{10} in Dualsystem und Interpretation

als Nachkommawert ist falsch!

Beobachtung: Multiplikation mit Basis bringt Nachkommastelle vor Komma: Zunächst rein dezimal:

$$0.421875_{10} \overset{.10}{\rightarrow} \boxed{4}.21875_{10}; \qquad 0.21875_{10} \overset{.10}{\rightarrow} \boxed{2}.1875_{10}$$

Dies gilt auch für Multiplikation mit anderer Basis, hier 2:

0.421875 **nach Binär:**

$$0.421875 \overset{.2}{\rightarrow} \boxed{0}.84375 \qquad 0.375 \overset{.2}{\rightarrow} \boxed{0}.75$$

$$0.84375 \overset{.2}{\rightarrow} \boxed{1}.6875 \qquad 0.75 \overset{.2}{\rightarrow} \boxed{1}.5$$

$$0.6875 \overset{.2}{\rightarrow} \boxed{1}.375 \qquad 0.5 \overset{.2}{\rightarrow} \boxed{1}.0 \rightarrow 0.011011_2$$

4.2.2 Repräsentation von Gleitkommazahlen ...

Kleine Zahlen in Gleitkomma-Repräsentation

Problem: Die Zahl 0 ist wegen (1+m) Konvention nicht darstellbar !
Zur Mantisse wird ja immer noch die Vorkomma-Eins hinzuaddiert.
Die Zahl

$$0\ 00000000\ 00000000000000000000000_{\text{IEEE 754}} = +1.0 \cdot 2^{-127} > 0$$

wäre damit die betragsmäßig kleinste darstellbare Zahl.
Ausserdem kann man noch kleinere Zahlen nahe 0 darstellen, wenn man Mantissen wie 0.00000001 zulässt. Daher:

Konvention:
Die Mantissen-Regel (1+m) gilt nur für e>00000000, also bis

$$0\ 00000001\ 00000000000000000000000_{\text{IEEE 754}} = +1.0 \cdot 2^{-126}$$

Für e=00000000 wird zu der Mantisse eine 0 als Vorkommastelle hinzugefügt, es können also Mantissen 0.m gebildet werden, z.B. 0.0000000001. Der Exponent **bleibt** -126, um eine Lücke zu verhindern

$$0\ 00000000\ 11111111111111111111111_{\text{IEEE 754}} = +0.999 \cdot 2^{-126}$$

$$0\ 00000000\ 00000000000000000000001_{\text{IEEE 754}} = +2^{-23} \cdot 2^{-126} = 2^{-149}$$

$$0\ 00000000\ 00000000000000000000000_{\text{IEEE 754}} = +0.0 \cdot 2^{-126} = 0$$

4.2.2 Repräsentation von Gleitkommazahlen ...

Darstellbare Werte in IEEE 754:

Weitere Ziele: ❍ Erhöhung der Darstellungsgenauigkeit nahe 0

❍ Codierung von Sonderfällen wie ∞ (unendlich) oder $\frac{1}{0}$ (Div. durch 0)

Normalisierte Gleitkommazahlen: Codierung wie zunächst (*vor* der vorigen Folie) vorgestellt

Denormalisierte Zahl: Bei $e = 0$ wird statt $\pm 1.m\ 2^{-127}$ kodiert: $\pm 0.m\ 2^{-126}$

⇒höhere Genauigkeit bei kleinen Zahlen, 0 wird darstellbar.

S	Exponent	Mantisse	Bedeutung
0/1	00000000_2	$0..00_2$	$+/-0$ (Null)
*	00000001_2 bis 11111110_2	beliebig	normalisierte Gleitkommazahl
*	11111110_2	$1..11_2$	Größte darstellbare Zahl $(3.4 \cdot 10^{38})$
*	00000000_2	$0..1_2$	Kleinste darstellbare Zahl $(2^{-149}=1.4 \cdot 10^{-45})$
0/1	11111111_2	$0..00_2$	$+/-$ unendlich
*	11111111_2	$\neq 0..00_2$	NaN (Not a Number)
*	00000000_2	$\neq 0..00_2$	denormalisierte Gleitkommazahl

4.3 Zeichen-Codierung

ASCII: American Standard Code for Information Interchange

Ziel:

- ❍ Standardisierter Zeichensatz zum Austausch von Text
- ❍ Zuordnung von Zeichen der Schriftsprache zu Binärzahlen

  ⇒Nur so können Computer Zeichen kodieren!

Ansatz:

- ❍ Basiert auf lateinischem Alphabet und arabischen Zahlen
- ❍ mind. 62 Standard-Zeichen (0-9, a-z, A-Z) plus Sonderzeichen

  ⇒mind. 7 bit notwendig (128 Zeichen)

Alternative Zeichensätze, z.B.

- ❍ ISCII (Indisch) oder
- ❍ ISO8859 (15 verschiedenen Codierungen zur Abdeckung aller europäischen Zeichen und Thai)

4.3 Zeichen-Codierung …

ASCII (Forts.)

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
SP	!	“	#	$	%	&	’	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	¡	=	¿	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	—	}	~	DEL

Tabelle 1: ASCII Tabelle

- Der ASCII Code ergibt sich durch Durchzählen in der Tabelle: ‚NUL‘=0, , A‘ = 65 ...
- Erste 32 Codes: Steuerzeichen wie Leerzeichen (SP = space = 32) oder Glocke (BEL=bell)
- Viele der Steuerzeichen sind nicht-druckbar und historisch bedingt (z.B. Steuerung von Druckern)

4.3 Zeichen-Codierung …

Unicode und UTF

Unicode: Legt einheitliche Codes für alle sinntragenden Zeichen aller Kulturen fest

UTF (Unicode Transformation Format):

- ❍ bildet Unicode auf Codes unterschiedlicher Länge ab
- ❍ häufige Zeichen erhalten kurze Codes
- ❍ ASCII-Zeichen in UTF-8: Führende 0 und 7 bit umfassen ASCIIZeichen
- ❍ Beispiel:

$$A = 1000001_{ASCII} = 01000001_{UTF8}$$

Textspeicherung als Zeichenketten (Folge von Zeichen)

4.4 Informations-Codierung

Wozu beschäftigen wir uns mit Information?

Grundsätzlich: Die Übertragung und Speicherung von Daten erfordert Effizienz

Beispiele: ❍ in deutschen Texten sind die Buchstaben ,,e“ und ,,s“ häufiger benötigt als andere⇒kurzer Morse-Code

❍ in einem einfarbigen Bild steckt im Grunde keine Information

Fragestellung: Wie läßt sich daraus Nutzen ziehen, um Daten kompakt darzustellen (zu ***komprimieren***)?

Komprimierungsarten:

❍ ***Verlustfrei***: Ursprüngliche Daten lassen sich vollständig wieder herstellen (wichtig z.B. bei Text)

❍ ***Verlustbehaftet***: Ursprüngliche Daten lassen sich nur unvollständig wieder herstellen (z.B. bei Bildern: jpeg)

4.4 Einschub: Morsealphabet

A	· –
B	– · · ·
C	– · – ·
D	– · ·
E	·
F	· · – ·
G	– – ·
H	· · · ·
I	· ·
J	· – – –
K	– · –
L	· – · ·
M	– –
N	– ·
O	– – –
P	· – – ·
Q	– – · –
R	· – ·
S	· · ·
T	–
U	· · –
V	· · · –
W	· – –
X	– · · –
Y	– · – –
Z	– – · ·

4.4 Informations-Codierung …

Zum Begriff ”Information“

❍ Von verschiedenen Wissenschaften (Informatik, Informationstheorie, Semiotik, . . .)

unterschiedlich definiert

❍ Allgemein: Potenziell oder tatsächlich vorhandenes nutzbares oder genutztes Wissen

❍ Wesentlich sind Wiedererkennbarkeit und Neuigkeitsgehalt
(in Bezug auf einen festgelegten vorhergehenden Wissensstand)

❍ Information ist für einen Betrachter in einem bestimmten ***Kontext*** von Bedeutung

und verändert dessen inneren Zustand (beim Menschen: dessen ***Wissen***)

❍ Vgl. Datum: Quasi ,,Einheit“ von Information, jedoch selbst keine Information,

sondern ,,Informationsträger“ (Zeichen),

Beispiel:

’xxxxxxxxxx’ = ,,10 Zeichen Daten“ aber dennoch möglicherweise ,,keine Information“

4.4.1 Information und Wahrscheinlichkeit

Wichtige Begriffe

Alphabet A:

- Besteht aus einer ***abzählbaren Menge von Zeichen*** . . .
- . . . und einer ***Ordnungsrelation*** (= feste Reihenfolge, wird im Folgenden nicht benötigt)
- Üblicherweise endlicher Zeichenvorrat

Beispiel:

- $A = \{a, b, c, \ldots z\}$ (Menge aller Kleinbuchstaben in alphabetischer Ordnung)
- $A = \{0, 1, 2, \ldots 9\}$ (Menge der ganzen Zahlen 0 bis 9 mit der „≤“ Relation)

Nachricht: Besteht aus Sequenz beliebiger Länge von Elementen von A

Länge einer Nachricht X: $|X|$ = Anzahl der Zeichen in X

Alle Nachrichten einer fester Länge n: $A^n = \{X : |X| = n\}$

Nachrichtenraum A^*: Menge aller Nachrichten über A: $A^* = \bigcup_n A^n$

4.4.1 Information und Wahrscheinlichkeit …

Shannon'scher Informationsbegriff (Claude Shannon, 1950)

Ausgangspunkte: Eine Nachricht $X = x_1 x_2 x_3 \ldots x_k \in A^*$

Auftrittswahrscheinlichkeit eines Zeichens $a \in A$ in X:

$$p_X(a) = \frac{\text{Häufigkeit von } a \text{ in } X}{\text{Länge von } X} \in [0,1], \quad \sum_{a \in X} p_X(a) = 1$$

Shannons Informationsbegriff: Axiome für den zu definierenden Informationsgehalt $I_X(a)$

- $I_X(a)$ groß, falls $p_X(a)$ klein, d.h. $I_X(a) \uparrow\downarrow p_X(a)$
- 100%-ige Auftrittswahrscheinlichkeit hat keine Information („weißes Bild"):

$$p_X(a) = 1 \Rightarrow I_X(a) = 0$$

- Informationsgehalt der Nachricht $X: I(X) = \sum_{a \in X} I_X(a)$

Konkrete Funktion für den Informationsgehalt:

$$I_X(a) = \log_2 \frac{1}{p_X(a)} = -\log_2 p_X(a)$$

4.4.1 Information und Wahrscheinlichkeit …

Shannon'scher Informationsbegriff: Hintergrund

Gegeben die Häufigkeitsverteilung, wie viele Ja-Nein Fragen muss ich stellen, um an einer gegebenen Stelle im Text das nächste Zeichen zu erfragen?

Fall 1: $p_X(a) = 1 \Rightarrow I_X(a) = 0$

0 Fragen, da von vornherein klar ist, dass auch das nächste Zeichen wieder ein a ist.

Fall 2: Gleichverteilung, jeder Buchstabe ist gleich häufig. $p_X(a_1) = p_X(a_2) = \frac{1}{|A|}$
Mit jeder Ja-Nein Frage kann ich die Zahl der verbleibenden Möglichkeiten halbieren (*„erste Hälfte oder zweite Hälfte des Alphabets?"* usw.) wie beim Intervallhalbierungsverfahren aus der Mathematik.
Mit k Fragen kann ich 2^k Buchstaben unterscheiden, umgekehrt ergibt sich daher

$$I_X(a) = \log_2(|A|) = \log_2 \frac{1}{p_X(a)} = -\log_2 p_X(a)$$

Fall 3: Ungleiche, nicht-triviale Häufigkeitsverteilung.
Hier gibt die Formel die zu erwartende Zahl von Fragen an, wenn man *optimal* fragt, indem man immer Fragen mit 50-50 Wahrscheinlichkeiten stellt, etwa am Anfang „*Vokal oder Konsonant?*".

4.4.1 Information und Wahrscheinlichkeit …

Entropie

Ausgangspunkt: Eine Nachricht $X = x_1 x_2 x_3 \ldots x_k \in A^*$

Bekannt: Statistischer Informationsgehalt $I_X(a)$ aller Zeichen von A

Mittlerer Informationsgehalt (Entropie) der Nachricht X:

$$H(X) = \sum_{a \in A} I_X(a) \cdot p_X(a) = -\sum_{a \in A} \log_2\left(p_X(a)\right) \cdot p_X(a)$$

H **entspricht der theoretischen Untergrenze** von Bit pro Zeichen zur Codierung

der Nachricht X

Beachte: Nicht die Abfolge, sondern nur die Häufigkeit der Zeichen wird verwendet

⇒ ,,geschickte“ Codierung kann mit weniger als H Bits auskommen
Etwa durch Codierung nicht auf Buchstaben- sondern auf Wortebene, wenn manche Worte oft in der Nachricht auftauchen.

Hintergrund des Mittleren Informationsgehalts:
Dies ist die zu statistisch zu erwartende Anzahl der ja-nein Fragen, die man stellen muss, um ein Zeichen x_i in X zu erhalten, wenn man die *Wahrscheinlichkeiten kennt* und mit dieser Kenntnis *optimal fragt*. Da wir noch nicht wissen, welcher Buchstabe folgt, gewichten wir jedes $I_X(a)$ mit der Wahrscheinlichkeit, dass dieser Buchstabe auftritt: ein sogenannter Erwartungwert.

Beispiel:

- Nachricht: $AABBDCDDAD$
- Alphabet: $\{A,B,C,D\}$
- Absolute Häufigkeiten: $\{3, 2, 1, 4\}$
- Relative Häufigkeiten: $\left\{\frac{3}{10},\frac{2}{10},\frac{1}{10},\frac{4}{10}\right\}$
- $H = -\left(\log_2\left(\frac{3}{10}\right)\cdot\frac{3}{10}+\log_2\left(\frac{2}{10}\right)\cdot\frac{2}{10}+\log_2\left(\frac{1}{10}\right)\cdot\frac{1}{10}+\log_2\left(\frac{4}{10}\right)\cdot\frac{4}{10}\right)\approx 1.85$

  ⇒ Codierung eines Zeichens erfordert (theoretisch) mind. 1.85 Bit

  ⇒ Codierung der Nachricht (10 Zeichen) erfordert (theoretisch) mind. $10 * 1.85 = 18.5$ Bit

- Weitere Beispiele:

  Rel. Häufigkeiten: $\{1,0,0,0\}$ $\quad H = -\left(\log_2(1)\cdot 1+3\cdot\log_2(0)\cdot 0\right)=0$

  Rel. Häufigkeiten: $\left\{\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4}\right\}$ $\quad H = -\left(4\cdot\underbrace{\log_2\left(\frac{1}{4}\right)}_{=-2}\cdot\frac{1}{4}\right)=2$

4.4.2 Runlength-Encoding

Ziel: Sehr einfacher, verlustfreier Kompressionsalgorithmus

Ansatz: Verkürzte Sequenz für wiederholte Zeichen durch Speicherung ihrer Anzahl

Beispiel: ❍ Alphabet $A = \{S, W\}$ (schwarz-weiss in einem Bild)

❍ Vor der Anwendung: $SSSSSSSSSSWSSSSSSSSSSWW$

❍ Nach RLE: $10SW10S2W$

Probleme: Codierung der Wiederholzahl, wenn Zahlen selbst Teil des Alphabets sind

Kontrollzeichen (Escape-Sequenz): Verwendung eines Sondersymbols (Escape-Zeichen), z.B. '\';

Beispiel:

❍ '\7' steht für Wiederholzahl 7

❍ '\\' steht für Zeichen '\'

4.4.3 Vorüberlegungen: Morsealphabet

A	· –
B	– · · ·
C	– · – ·
D	– · ·
E	·
F	· · – ·
G	– – ·
H	· · · ·
I	· ·
J	· – – –
K	– · –
L	· – · ·
M	– –
N	– ·
O	– – –
P	· – – ·
Q	– – · –
R	· – ·
S	· · ·
T	–
U	· · –
V	· · · –
W	· – –
X	– · · –
Y	– · – –
Z	– – · ·

Grundidee: Häufige Buchstaben bekommen kurzen Code.

T E S T = - . ... -

Versuch: Übersetzung in einen Binärcode: . = 0 und - = 1
T E S T = 100001

Es fehlt die Trennung zwischen den Zeichen.
Beachte die Pausen beim Morsen!

- . . . . - könnte z.B. auch gelesen werden als
D U

Eindeutigkeit wird erreicht durch die
Fano-Bedingung (die beim Morsealphabet nicht gilt):
Kein Wort des Codes darf Anfang eines anderen Wortes des selben Codes sein.

4.4.3 Huffman-Codes

Huffman-Codierung (David Huffman, 1952)

Alternative Bezeichnung: Entropie-Codierung

Ziel: Effiziente Codierung von Nachrichten unter Berücksichtigung der Auftrittswahrscheinlichkeiten der Zeichen und der Fano-Bedingung

Idee: Verwendung von kurzen Codes für häufiger auftretende Zeichen

⇒Zeichen-Codes mit variabler Länge & abhängig von Nachricht

Herausforderung: Fano Bedingung, hier ein weiteres Beispiel einer Mehrdeutigkeit:

- 3 Zeichen (A,B,C) mit steigender Wahrscheinlichkeit $(p(A) < p(B) < p(C))$
- Ungeschickte Codierung: $A = 10$, $B = 01$, $C = 0$
- Nun kann die Zeichenfolge 10010 entweder ABC oder ACA bedeuten

Lösung durch Fano-Bedingung: Kein Code für ein Zeichen darf Anfang eines anderen Codes sein.

Im Beispiel ist die Codierung für C Anfang der Codierung für B

4.4.3 Huffman-Codes …

Algorithmus: Huffman-Codierung

1. Sortiere Zeichen aufsteigend nach ihrer Wahrscheinlichkeit
2. Zusammenfassung der beiden Zeichen mit der geringsten Wahrscheinlichkeit
   - ❍ Vergabe von (zusätzlichen) Bits für beide Zeichen
   - ❍ Addition der Wahrscheinlichkeiten beider Zeichen
3. Wiederhole die Schritte, bis sich nur noch ein Zeichen in der Sortierung befindet

Beispiel: Nachricht „HAAAALLO“

Z	p	Code	Z	p	C	Z	p	C	Z	p	Code
H	1/8	-	H	2/8	0	H	4/8	00	H	8/8	000
O	1/8	-	0		1	0		01	0		001
L	2/8	-	L	2/8	-	L		1	L		01
A	4/8	-	A	4/8	-	A	4/8	-	A		1

Entropie: 1.75 (bit/Zeichen)

Codierung der Nachricht (8 Zeichen)**:**
Theor. Untergrenze: 14 bit
Tatsächlich: 14 bit

HAAAALLO = 000 1 1 1 1 01 01 001 = 00011110101001, 14 Zeichen

4.4.3 Huffman-Codes …

Bemerkungen:

- Die Zuordnung der Bits 0/1 ist jeweils willkürlich.
- Ebenso ist es bei gleicher Häufigkeit egal, welche Knoten man als erste verknüpft.

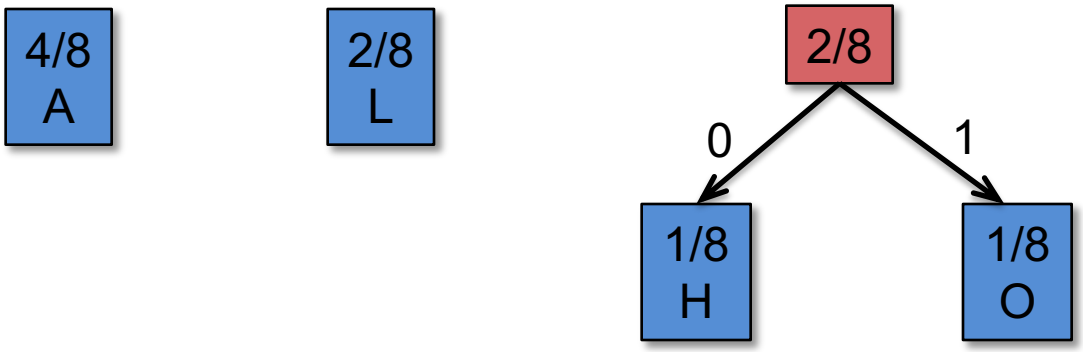
Es gibt daher viele Möglichkeiten, den Huffman-Algorithmus auszuführen.
Dies zeigt die folgende Folie, die eine andere grafische Darstellung verwendet.

4.4.3 Huffman-Codes …

Nochmals das Beispiel: “HAAAALLO”

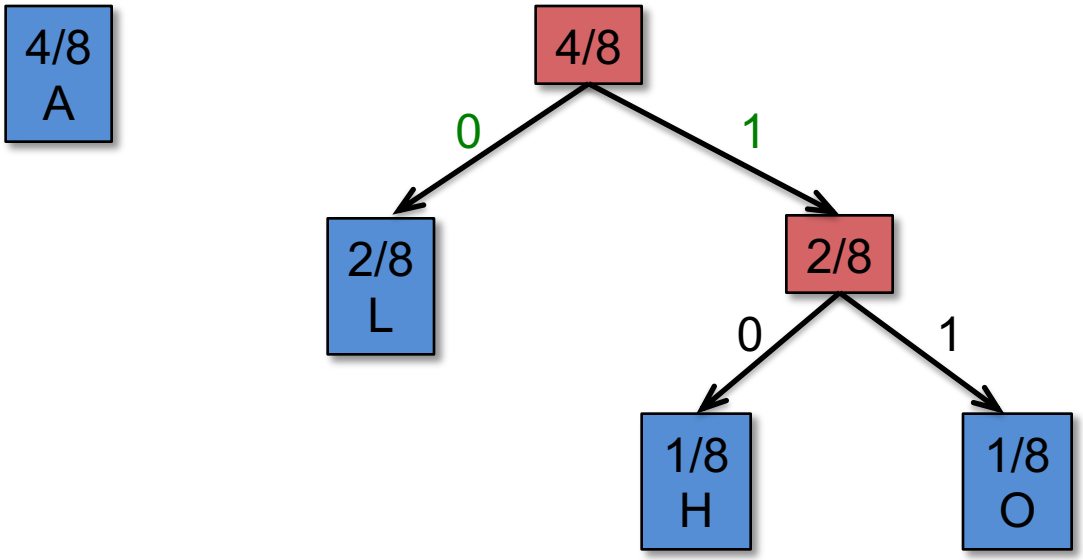
4.4.3 Huffman-Codes …

Nochmals das Beispiel: “HAAAALLO”



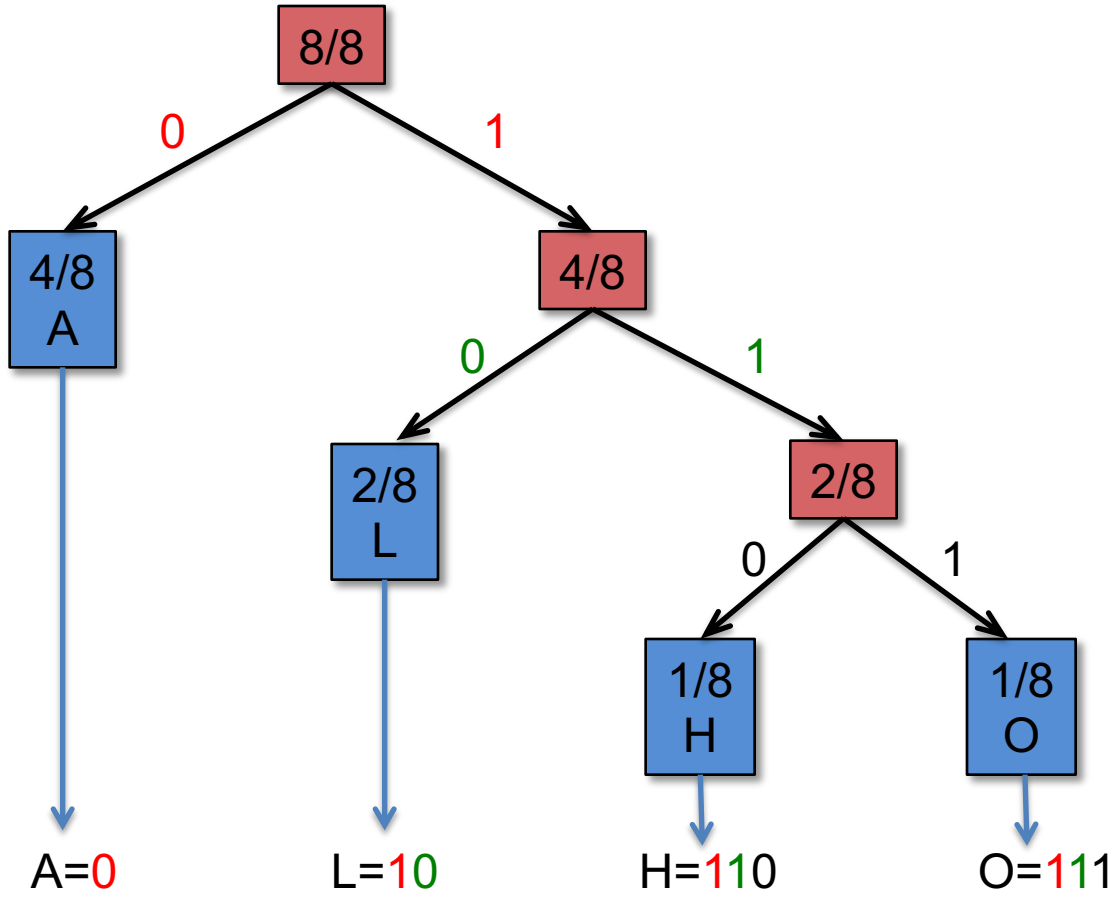
4.4.3 Huffman-Codes …

Nochmals das Beispiel: “HAAAALLO”



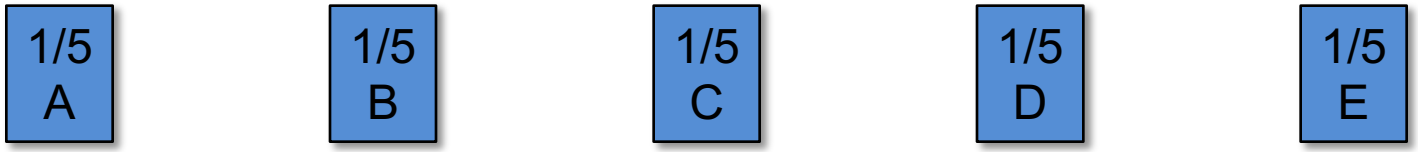
4.4.3 Huffman-Codes …

Nochmals das Beispiel: “HAAAALLO”.



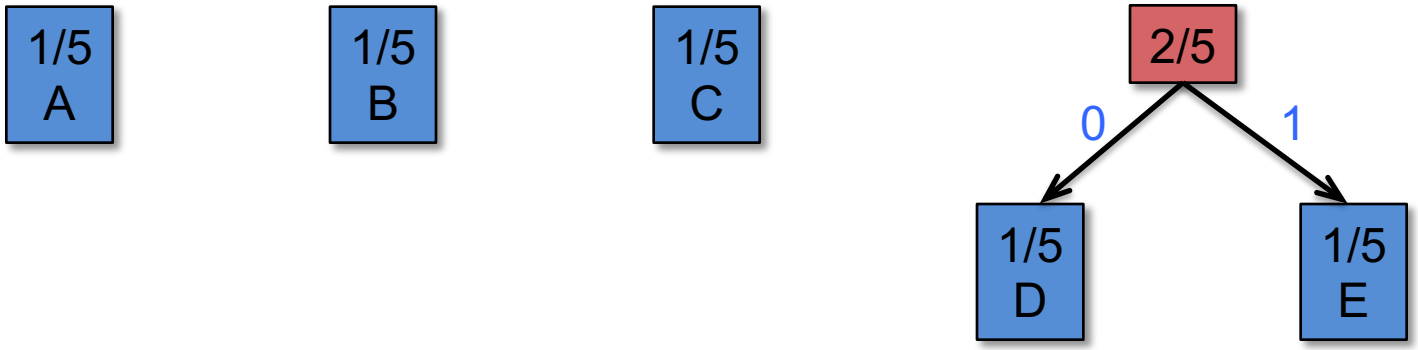
4.4.3 Huffman-Codes …

Weiteres Beispiel: “ABCDE”



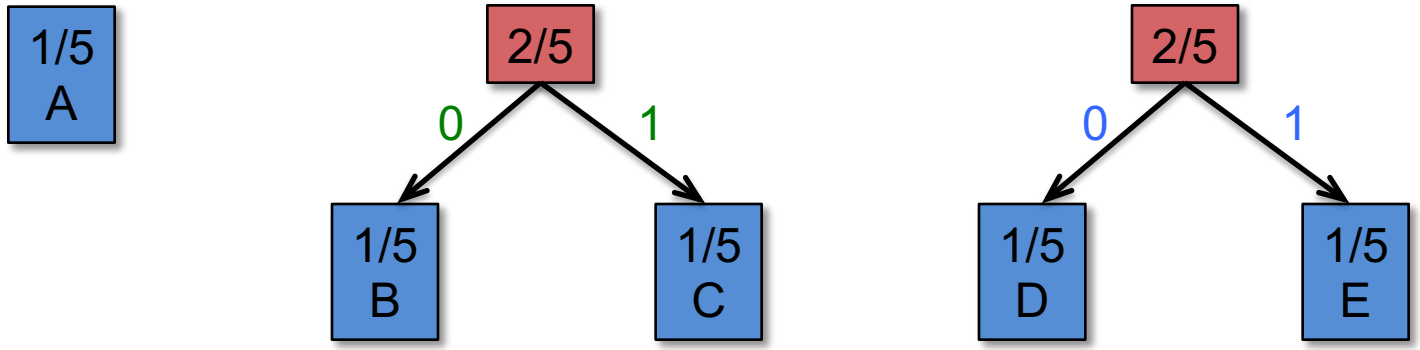
4.4.3 Huffman-Codes …

Weiteres Beispiel: “ABCDE”



4.4.3 Huffman-Codes …

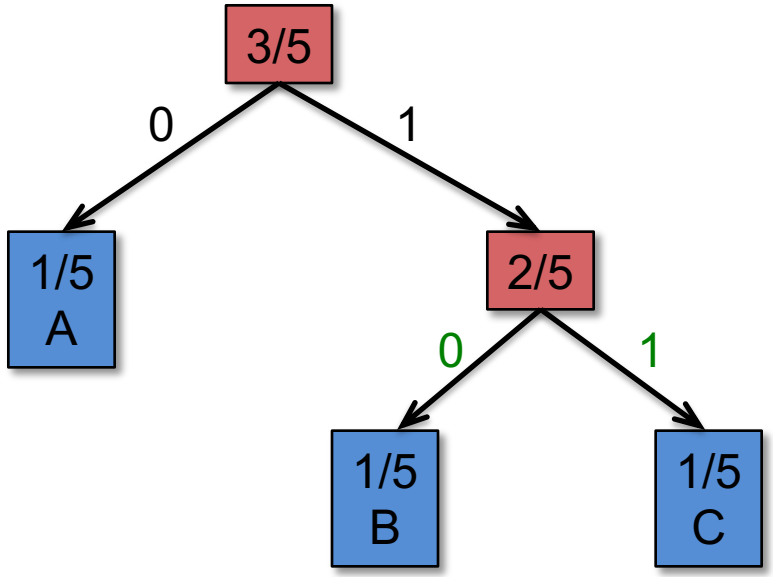
Weiteres Beispiel: "ABCDE". Beachte: bei allen 0,1 Ziffern wird es sich um Endziffern handeln.

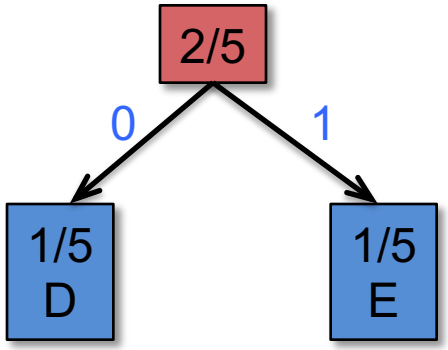


Wichtig: hier wird ein neuer Baum angelegt, die Buchstaben D, E werden zunächst nicht weiterverfolgt. Also: nicht immer nur an existierenden Baum anbauen!

4.4.3 Huffman-Codes …

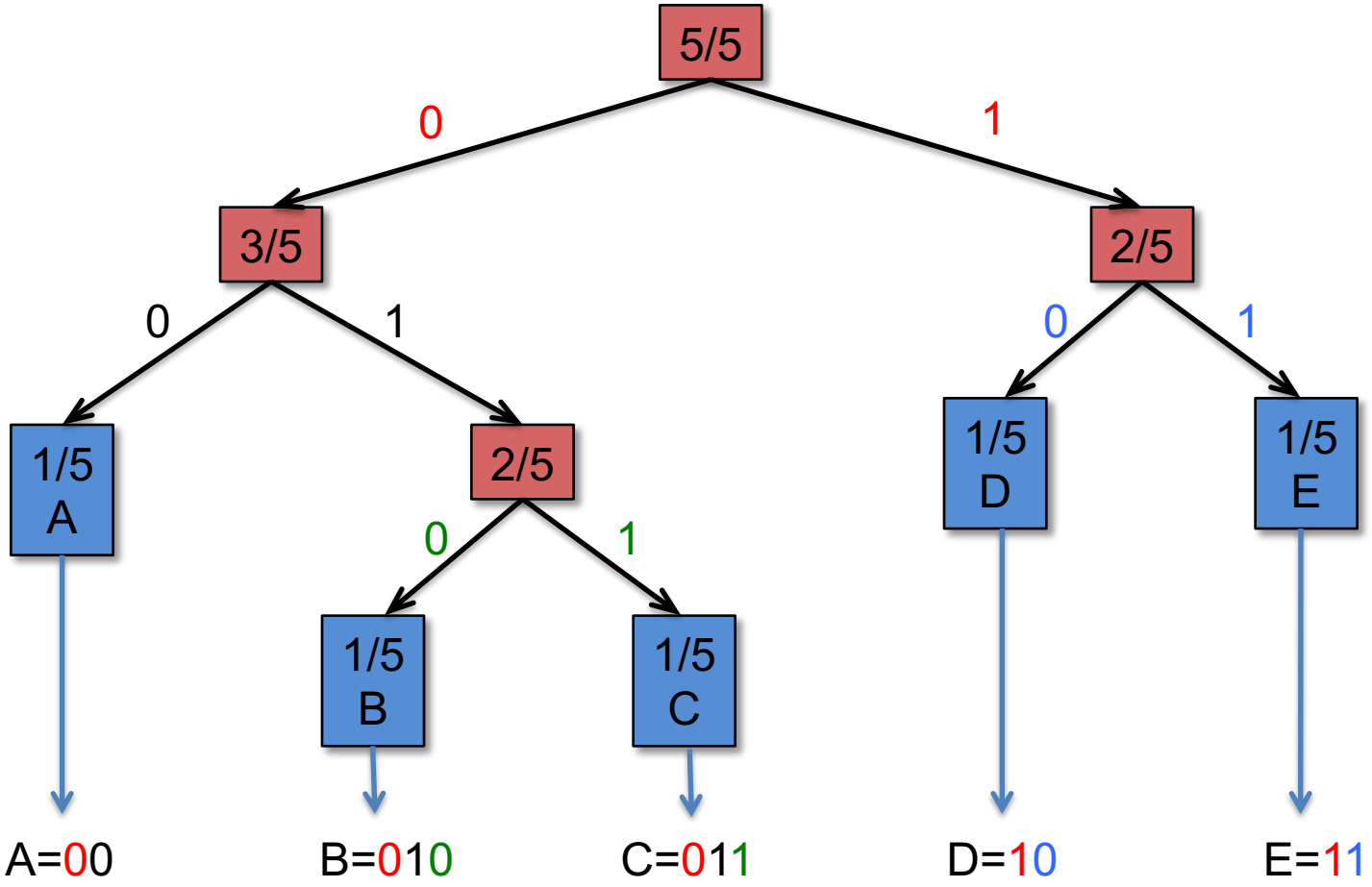
Weiteres Beispiel: "ABCDE". Das A kann mit B,C oder D,E zusammengefasst werden.





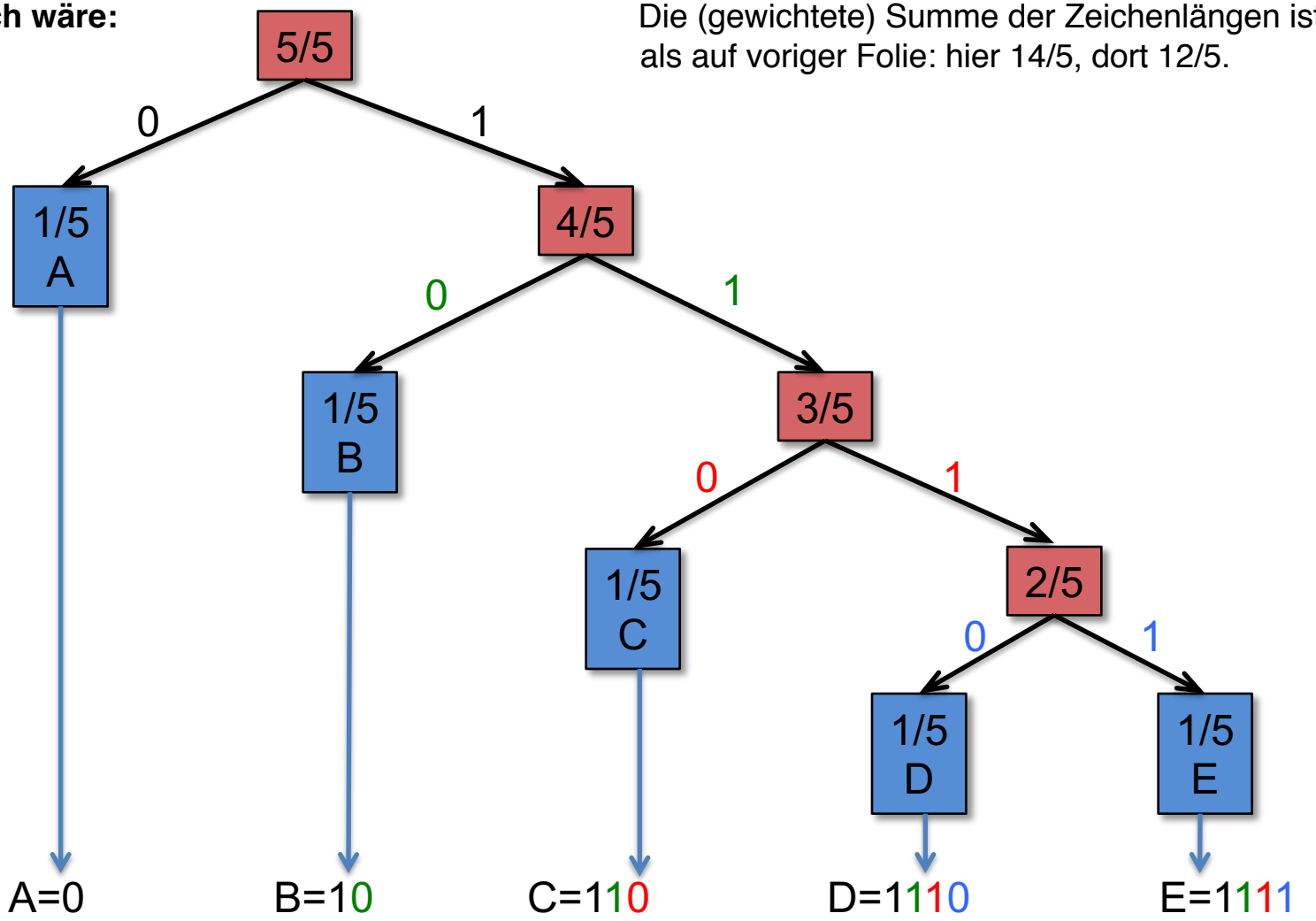
4.4.3 Huffman-Codes …

Weiteres Beispiel: “ABCDE”.



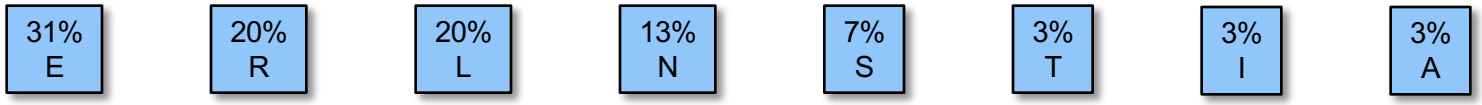
4.4.3 Huffman-Codes: Falsche Lösung!

Falsch wäre:

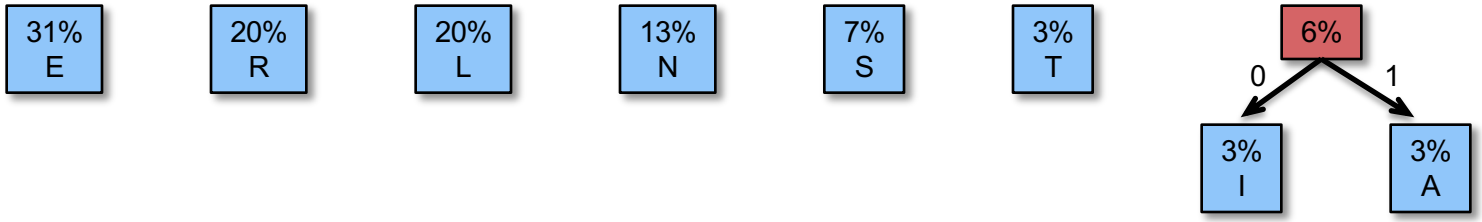


Die (gewichtete) Summe der Zeichenlängen ist höher als auf voriger Folie: hier 14/5, dort 12/5.

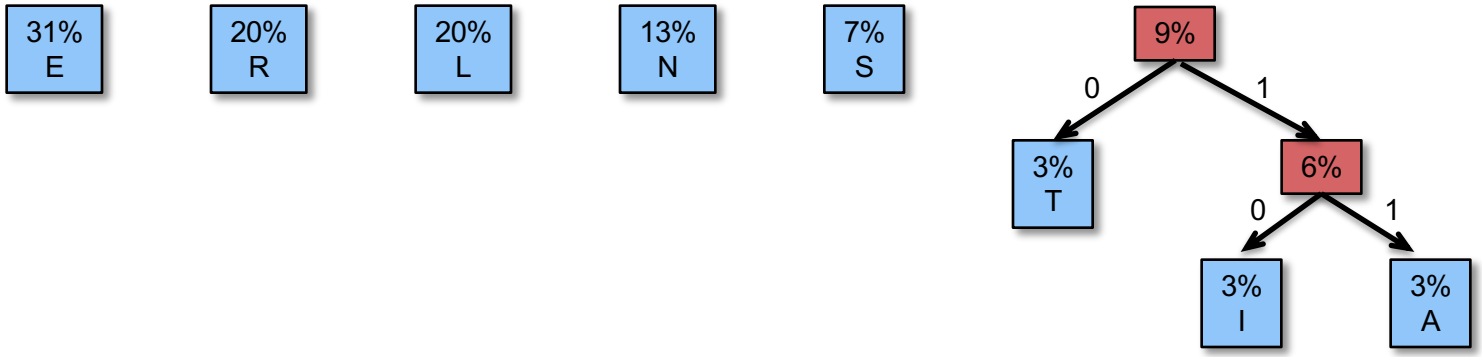
4.4.3 Huffman-Codes …



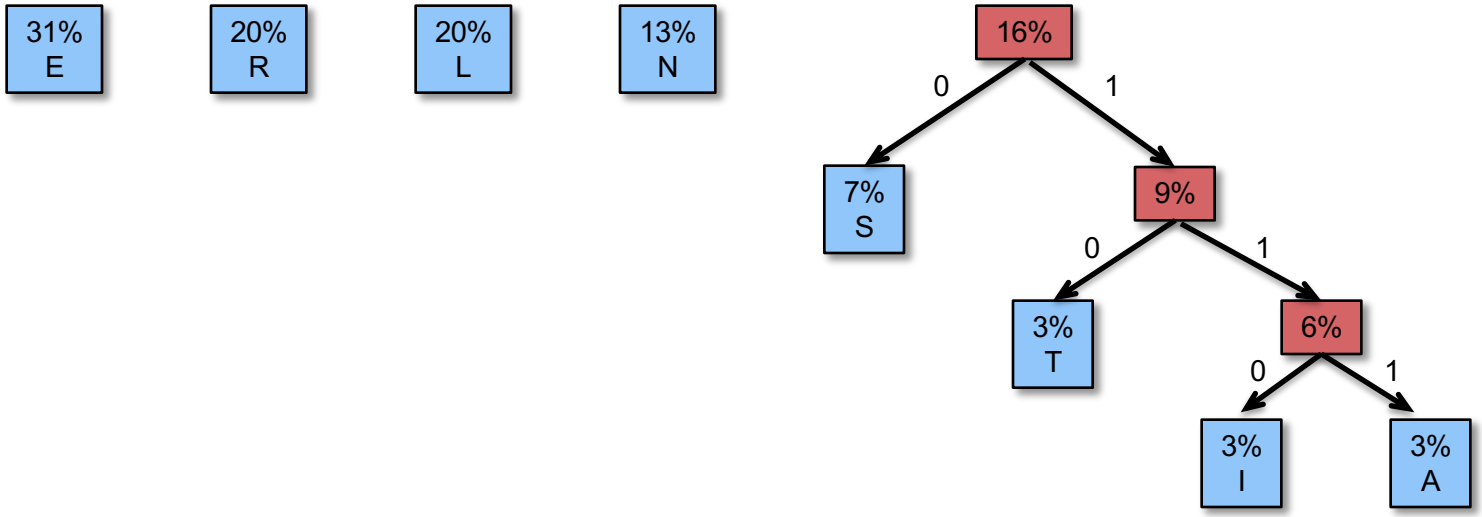
4.4.3 Huffman-Codes …



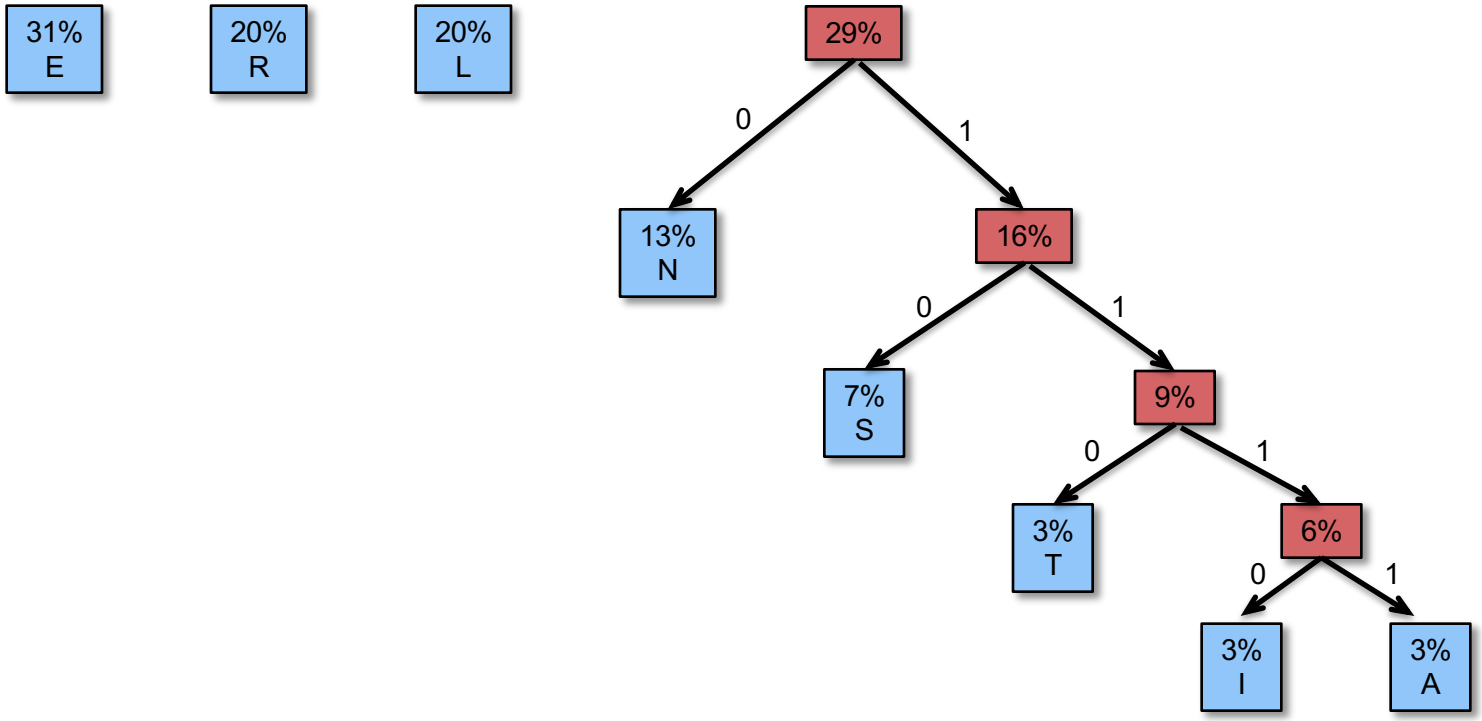
4.4.3 Huffman-Codes …



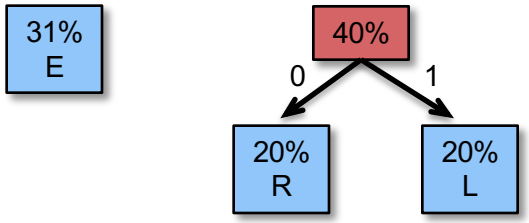
4.4.3 Huffman-Codes …



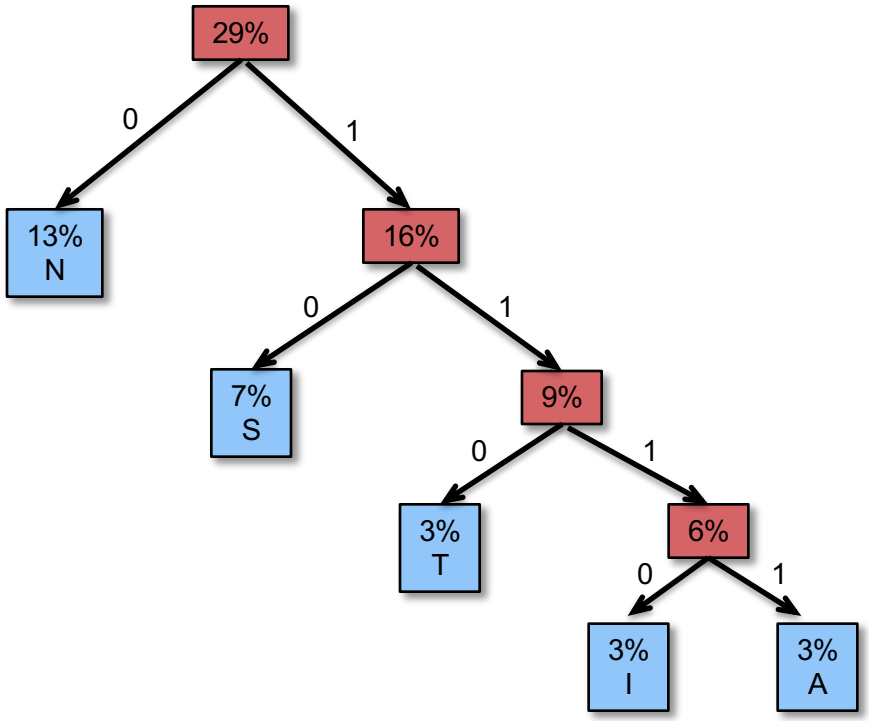
4.4.3 Huffman-Codes …



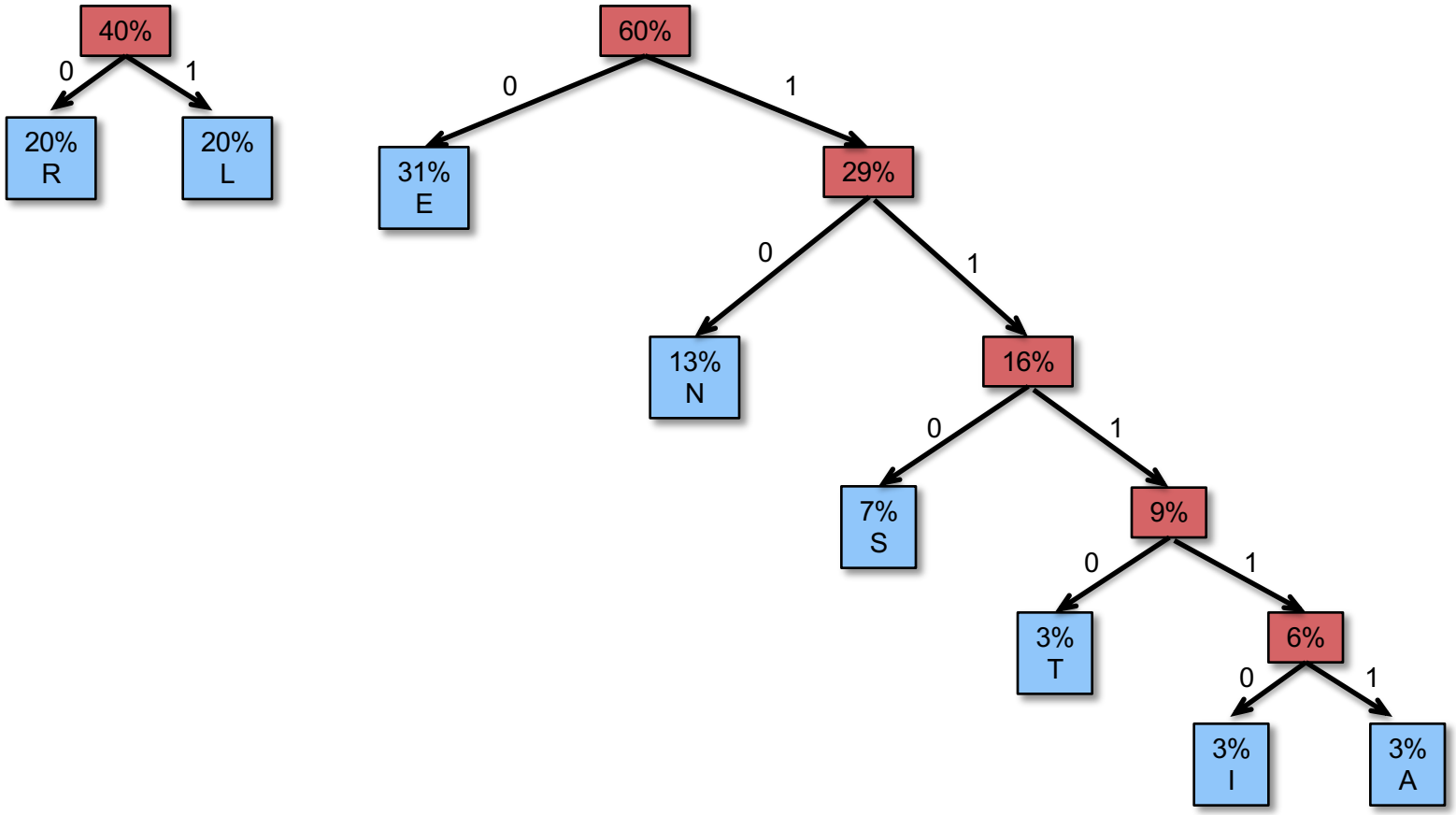
4.4.3 Huffman-Codes …



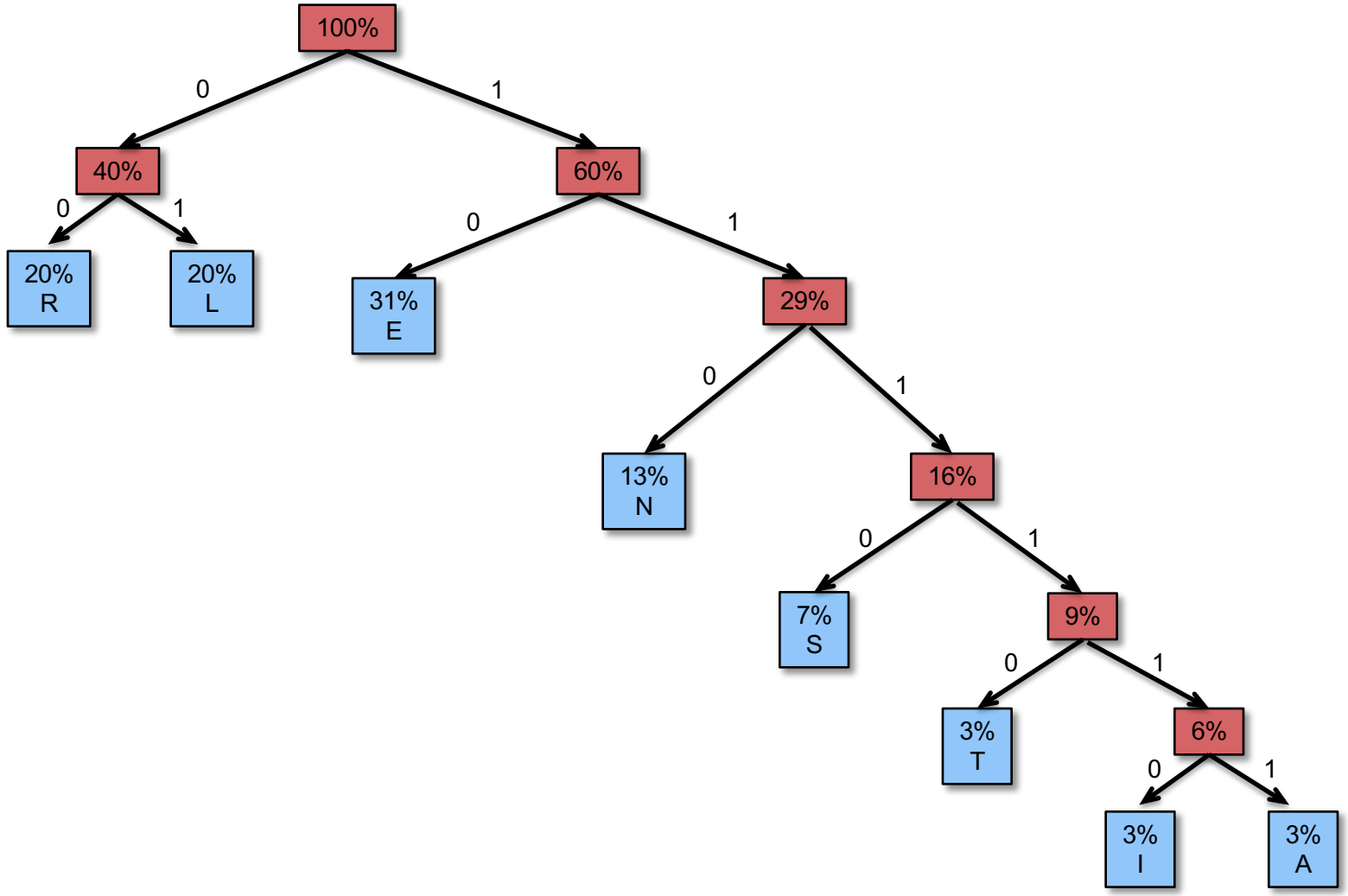
Verschiebe das E
zum rechten Baum!



4.4.3 Huffman-Codes …



4.4.3 Huffman-Codes …



R=00 L=01 E=10 N=110 S=1110 T=11110 I=111110 A=111111

4.5 Zusammenfassung

Wichtige Erkenntnisse und Inhalte dieses Abschnitts:

Codierung befasst sich mit der binären Repräsentation ,,beliebiger“ Daten

Zahlensysteme der Informatik: Binär (1 Bit), oktal (3 Bits), hexadezimal (4 Bits)

Repräsentation von Zahlen im Rechner durch Zahlentypen

- ❍ Einschränkung durch fixe Anzahl Bits: Feste Ober- u. Untergrenze, eingeschränkte Genauigkeit
- ❍ Natürliche Zahlen (unsigned int) mittels Binärsystem
- ❍ Ganze Zahlen mittels Vorzeichen-Betrag, Zweierkomplement (Standard für Ganzzahldarstellung) oder Biased Darstellung
- ❍ Gleitkommazahlen im IEEE-754 Format auf Basis der wissenschaftlichen Schreibweise

Repräsentation von Schriftzeichen bzw. Text

- ❍ ASCII Zeichensatz und internat. Erweiterungen (ISO-8859, UTF-8)

Informations-Codierung zur Komprimierung von (Text-)Daten