Zusammenfassung - Informationssysteme

21 October 2014 09:32

Version: 2.1.0
Studium: 1. Semester, Bachelor in Wirtschaftsinformatik
Schule: Hochschule Luzern - Wirtschaft
Author: Janik von Rotz (http://janikvonrotz.ch)

Zahlensysteme

30 September 2014 08:45

Natürliche Zahlen

- Allgemein: Natürliche Zahlen n werden dargestellt durch:

$$n = \sum_{i=0}^{N-1} b_i \, B^i$$

n = natürliche Zahl
B = Basis des Zahlensystems
b = Ziffern
N = Anzahl Stellen
i = Position

N = 4
i= 3210

$$n = (2A03)_{16} = b_3 * 16^3 + b_2 * 16^2 + b_1 * 16^1 + b_0 * 16^0$$

$$n = (2A03)_{16} = 2 * 16^3 + A * 16^2 + 0 * 16^1 + 3 * 16^0 = 10755$$

$10 * 16^2$

Reele Zahlen

Festpunktzahl z.B. 17.439

Festpunktzahl Charakteristiken:

- der **Punkt** (d.h. das "**Komma**") steht immer an einer bestimmten festgelegten Stelle (zwischen z_0 und z_{-1})
- die Zahl hat die Länge n + m
  (n Stellen vor dem Punkt, m Stellen nach dem Punkt)

Formel gilt auch für 3er, 4er, ...x-er- System!

Allgemein:

zahl = $(z_{n-1}z_{n-2}...z_1z_0 \, z_{-1}z_{-2}...z_{-m})_{(2)}$ d.h. $\sum_{i=-m}^{n-1} z_i 2^i$

Beispiel: (im 2-er System!)

$(11.011)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}$
$= 2 + 1 + 0 \cdot 0.5 + 0.25 + 0.1\,2\,5$
$= (3.375)_{10}$

1 0 (n Stellen) -1 -2 -3 (-m Stellen)

Eine Formel, die man nicht vergessen soll:

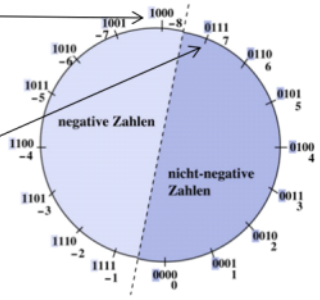
$$n^{-m} = \frac{1}{n^m}$$

Grundlagen Zweierkomplement

- Allgemein:
  - Kleinste darstellbare negative Zahl: $-B^{s-1}$
  - Grösste darstellbare **positive** Zahl: $B^{s-1}-1$

B = Zahlensystem
s = Registergrösse (=Anzahl Bit)

- für Zweiersystem (B = 2) **kleinste negative Zahl**
  bei **s** = 4: −24−1 = −23 = **−8**
  bei **s** = 8: −28−1 = −27 = −128
  bei **s** = 16: −216−1 = −215 = −32768
  bei **s** = 32: −232−1 = −231 = −2147483648
- für Zweiersystem (B=2) **grösste positive Zahl**
  bei **s** = 4: 24−1 − 1 = 23 − 1 = **7**
  bei **s** = 8: 28−1 − 1 = 27 − 1 = 127
  bei **s** = 16: 216−1 − 1 = 215 − 1 = 32767
  bei **s** = 32: 232−1 − 1 = 231 − 1 = 2147483647



- Vorgehen: z.B. 2er-Komplement von 5:
  - jedes einzelne Bit wird für den Wert (hier z.B. 5) umgekehrt UND
  - nachher wird immer 1 dazugezählt

5	0	1	0	1
	1	0	1	0
+	0	0	0	1
2er-Komplement d.h. -5:	1	0	1	1

- 2er-Komplement von -6:
  - jedes einzelne Bit wird für den Wert (hier z.B. -6) umgekehrt UND
  - nachher wird immer 1 dazugezählt

-6	1	0	1	0
	0	1	0	1
+	0	0	0	1
				1
2er-Komplement d.h. 6:	0	1	1	0

Beispiel

1010 > DEC

$$\sum z_i \cdot 2^i = 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 10$$

2-4 in Binär

2 > 0010
-4 > 4 Invertieren > 1011 > +1 Bit > 1100

0010
+1100
=**1**110

1110 > -1 Bit > 1101 > Invertieren > 0010 > Lösung: -2

Darstellung Gleitpunktzahl

10 January 2015 13:20

IEEE Darstellung Gleitpunktzahl

Formel zur Darstellung einer Gleitpunktzahl im IEEE-Format:

$(-1)^S \cdot (2^{B-bias}) \cdot (1.f_N.....f_0)$

- S = *SIGN*: S = 0 (positiv); S = 1 (negativ)
- (2^{B-bias}) = *EXPONENT*: bias = 127 (`float`), bias = 1023 (`double`); B = Biased Exponent (zu speichernder Exponent)
- $(1.f_N.....f_0)$ = *SIGNIFICAND*: N=22 (`float`=23 Stellen), N=51 (`double`=52 Stellen)

- Beispiel:

  1 10001101 01100100000000000000000

  => -27776.0 d.h. $-1.6953125 * 2^{14}$

Nach IEEE gilt Folgendes für `float` (einfach) und `double` (doppelt):

	einfach	doppelt
Vorzeichen-Bits	1	1
Exponenten-Bits	8	11
Mantissen-Bits	23	52
Bits insgesamt	32	64
BIAS	127	1023

Beispiel:

Binär zu Fliesskomma Dezimalzahl

1'**1000'0001**'11+0x21

10000001 = 129
129-127=2
11=0.11 -> 2^-1+2^-2=0.75 -> +1=1.75 -> 1.75*2^2=7.0
Resultat: -7

Fliesskomma Dezimalzahl zu Binär

-0.625

0.625=0.101
Verschiebung um -1 Exponent
1.01 -> Entfernen 1 -> 01

x-127=-1 -> x=126
128=0111'1111
Resultat: 1'1000'0000'01+0x21

3.625

6.625=110.101
Verschiebung um 2 Exponenten
1.10101 -> Entfernen 1 -> 10101
x-127=2 -> x=129
Resultat: 0'1000'0001'10101+0x18

Umrechnen von Zahlensystemen

10 January 2015 13:19

Während die Umrechnung von Dualzahlen in Dezimalzahlen keine Schwierigkeiten bereitet, braucht es für den umgekehrten Weg einen Algorithmus, d.h. eine Rechenvorschrift, die für den Startwert ‚natürliche Zahl im Dezimalsystem' den Endwert ‚diese Zahl im Dualsystem' liefert.

Ein solcher **Algorithmus** lautet für die natürliche Zahl n wie folgt:

1. $n : 2 = m + \mathrm{Rest}\ r$
2. Falls $m \neq 0$, so mache m zum neuen n und wiederhole Schritt 1. Ansonsten fahre mit Schritt 3 fort.
3. Die ermittelten Reste r werden von unten nach oben gelesen und ergeben aneinander gereiht die gesuchte Darstellung von n.

Bsp. Sei $n = 71$.

$71 : 2 = 35 + \mathrm{Rest}\ 1$

$35 : 2 = 17 + \mathrm{Rest}\ 1$

$17 : 2 = 8 + \mathrm{Rest}\ 1$

$8 : 2 = 4 + \mathrm{Rest}\ 0$

$4 : 2 = 2 + \mathrm{Rest}\ 0$

$2 : 2 = 1 + \mathrm{Rest}\ 0$

$1 : 2 = 0 + \mathrm{Rest}\ 1$

Folglich ist $71 = 1000111_{(2)}$. Dies bestätigt das Ergebnis des vorangehenden Beispiels.

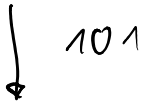
DEZ zu HEX

45054/16=2815.875 > 0.875*16=14 > E
2815/16=175.9375 > 0.9375*16=15 > F
175/16=10.9375 > 0.9375*16=15 > F
10/16=0.625 > 0.625*16=10.0 > A

$AFFE_{(16)}$

Flieskomma

0.625*2=1.25 -> Ziffer: 1
0.25*2=0.5 -> Ziffer: 0
0.5*2=1 -> Ziffer: 1



101

Hilfstabelle: Dual zu Dezimal

0000'0000'0000'0000'0001 = 2^0 = 1
0000'0000'0000'0000'0010 = 2^1 = 2
0000'0000'0000'0000'0100 = 2^2 = 4
0000'0000'0000'0000'1000 = 2^3 = 8
0000'0000'0000'0001'0000 = 2^4 = 16
0000'0000'0000'0010'0000 = 2^5 = 32
0000'0000'0000'0100'0000 = 2^6 = 64
0000'0000'0000'1000'0000 = 2^7 = 128
0000'0000'0001'0000'0000 = 2^8 = 265
0000'0000'0010'0000'0000 = 2^9 = 512
0000'0000'0100'0000'0000 = 2^{10} = 1'024
0000'0000'1000'0000'0000 = 2^{11} = 2'048
0000'0001'0000'0000'0000 = 2^{12} = 4'096

0000'0010'0000'0000'0000 = 2^{13} = 8'192
0000'0100'0000'0000'0000 = 2^{14} = 16'384
0000'1000'0000'0000'0000 = 2^{15} = 32'768
0001'0000'0000'0000'0000 = 2^{16} = 65'536
0010'0000'0000'0000'0000 = 2^{17} = 131'072
0100'0000'0000'0000'0000 = 2^{18} = 262'144
1000'0000'0000'0000'0000 = 2^{19} = 524'288
1'0000'0000'0000'0000'0000 = 2^{20} = 1048576

Hilfstabelle: Dezimal zu Hexadezimal

Zusammenfassung - Tabelle

Decimal	Hexadecimal	Binary	Oktadecimal
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	8	1000	10
9	9	1001	11
10	A	1010	12
11	B	1011	13
12	C	1100	14
13	D	1101	15
14	E	1110	16
15	F	1111	17

Beispiel: Hex zu Bin: FA = 1111 1010 = 128 + 64 + 32 + 16 + 8 + 0 + 2 + 0 = 250

Datentypen Berechnung

10 January 2015 13:39

Annahme wir haben eine Dezimalzahl und wollen diese in dualen Datentypen verschiedener Grösse und wechselndem Vorzeichen speichern.

Dezimalzahl: 39036
Vorgehen:

- x=Anzahl Bits Datentyp
- unsigned: $39036/2^x=z.y$
  - Ist z=0: dann hat der Wert "platz"
  - Ist z>0: dann übersteigt der Wert den Speicher
    - $y*2^x=d$ ergibt den Dezimalwert der Bits, die die Speicherstelle überschreiben.
- signed: $39036/2^{x-1}=z.y$
  - Ist z=0: dann hat der Werte "platz"
  - Ist z>0: dann übersteigt der Wert den Speicher
    - Ist z ungerade: handelt es sich um einen negativen Wert
      - $y*2^{x-1}-2^{x-1}=d$ ergibt den negativen Dezimalwert, der sich aus der Überschreibung ergibt.
    - Ist z gerade: handelt es sich um einen positiven Wert
      - $y*2^{x-1}=d$ ergibt den positiven Dezimalwert, der sich aus der Überschreibung ergibt.

	signed	unsigned
Byte (8)	124	124
Short (16)	39036	-26500
int (32)	39036	39036

Algorithmen

07 October 2014 08:24

Algorithmen

Beschreiben eine Verarbeitungsvorschrift, dies kann sein:
Vorschrift zur Lösung einer Aufgabe
Vorschrift zur Lösung eines Problems

Begriffe

Begriff	**Beschreibung**
terminiert	Algorithmus ist nach n-Schritten beendet, hat er kein bestimmtes Ende, spricht man einem nicht-terminiertem Algorithmus
Determinismus	Für alle Eingaben ist der Ablauf des Algorithmus eindeutig bestimmt, anderenfalls heisst er nicht-deterministisch
Determiniertheit	Ein Algorithmus heißt determiniert, wenn er bei gleichen zulässigen Eingabewerten stets das gleiche Ergebnis liefert. Andernfalls heißt er nicht-determiniert.

Wichtige Aussagen: Ein deterministischer Algorithmus ist immer determiniert,
d. h. er liefert bei gleicher Eingabe immer die gleiche Ausgabe.
Die "Umkehrung" aber gilt nicht: So gibt es Algorithmen, die nicht-deterministisch,
aber trotzdem determiniert sind (d. h. das gleiche Ergebnis liefern).

Kodierung

28 October 2014 08:25

Zeichensatz

- Liste von Zeichen

Zeichenkodierung

- Eindeutige Zuordnung von Zeichen zu einer Zahl

ASCII - American Standard for Coded Information Interchange

- Klein- und Grossbuchstaben, Ziffern + Sonderzeichen
- Codierung in 1 Byte => 265 Zeichen möglich

16					0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
					0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
					0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
					10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	0	0	0	1	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	QS	RS	US
2	0	0	1	0	32	SP		"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	0	1	1	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	0	1	0	0	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	0	1	0	1	80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	0	1	1	0	96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	0	1	1	1	112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8	1	0	0	0	128																
9	1	0	0	1	144																
A	1	0	1	0	160																
B	1	0	1	1	176																
C	1	1	0	0	192																
D	1	1	0	1	208																
E	1	1	1	0	224																
F	1	1	1	1	240																

Abbildung 8: ASCII mit bin. / dez. / hex.-Notationen (Fischer, 2001)

A = 65_{10} = 0100 0001$_2$ = 41_{16}

Unicode

- Unicode ist ein internationaler Standard, in dem langfristig für jedes sinntragende Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird.
- Ziel ist es, die Verwendung unterschiedlicher und inkompatibler Kodierungen in verschiedenen Ländern oder Kulturkreisen zu beseitigen. U
- Unicode wird ständig um Zeichen weiterer Schriftsysteme ergänzt (heute in der Version 6.2)
- Die Nummerierung ist hexadezimal in der Schreibweise U+XXXXXXXX
- z.B. Beispiel: U+00B6 ist das "Pilcrow-Zeichen", wie wir es aus Word kennen: "¶".
- Üblich sind eine 4-/6- oder 8-stellige Hex-Schreibung (16/24/32 Bits) – Obiges ist ein Beispiel

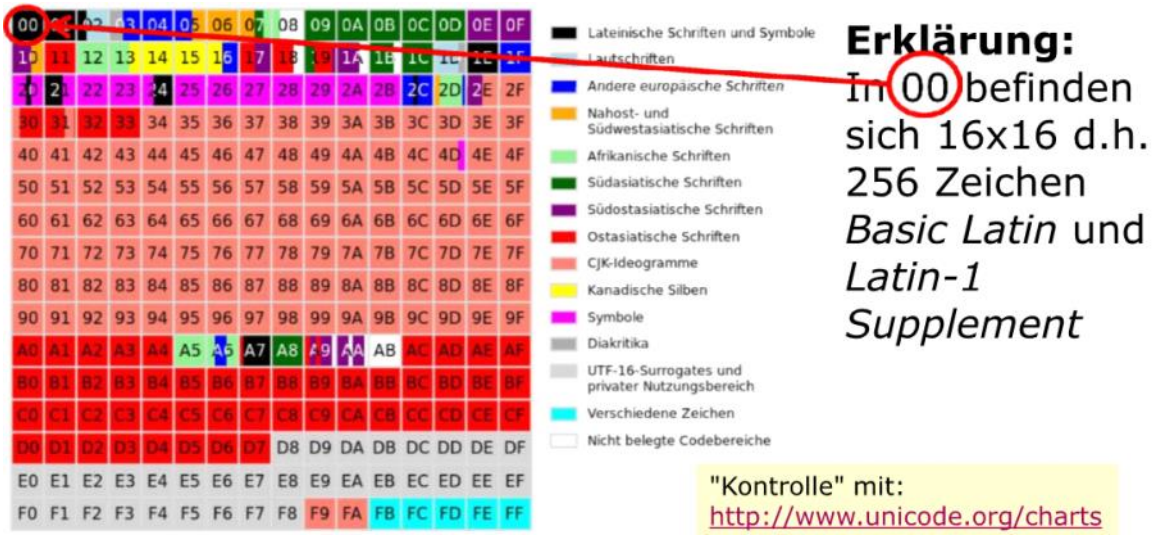
für 4 Stellen.

- Führende Nullen können in dieser Schreibung paarweise weggelassen werden.
- Die Identifikation durch eine Bezeichnung ist definiert und kann übersetzt werden

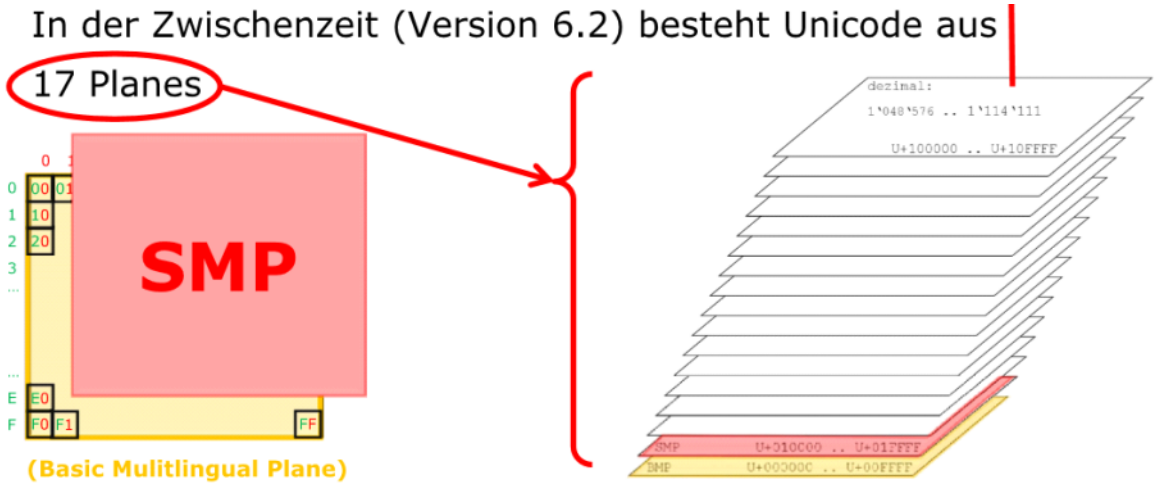
Struktur

Die Zeichen des Unicode sind in so genannten Planes organisiert. Eine Plane ist eine quadratische Tabelle von 256 Zeilen zu 256 Spalten, also 216 = 65'536 Feldern.
Dies sind die Code Points und werden innerhalb der Plane fortlaufend nummeriert.
Unicode war ehemals ein 16-Bit Code, bestand also aus einer einzigen Plane.
Inzwischen, mit Version 5.0.0 vom Juli 2006, sind weitere 16 Planes dazugekommen!

Aufbau der BMP (Basic Multilingual Plane)



Aufbau SMP (Supplementary Multilingual Plane) -> Unicode wächst jedes Jahr um abertausende Zeichen.



UTF - Unicode Transformation Formate

UTF ist also eine Methode, Unicode-Zeichen auf Folgen von Bits abzubilden.

Für die Repräsentation der Unicode-Zeichen zum Zweck der elektronischen Datenverarbeitung gibt es verschiedene UTFs.

- UTF-8 empfiehlt sich für Texte mit vorwiegend lateinischen Buchstaben.
- UTF-16 empfiehlt sich für Texte mit z.B. asiatischen Zeichen.
- UTF-32 empfiehlt sich künftig für höchste Performanz (bei grösstem Speicherbedarf).

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

Abbildung 14: Umrechnung von Codepunkt in UTF-8 (http://www.ietf.org/rfc/rfc3629.txt)

UTF-8 muss gemäss den Internet-Behörden (IETF, IANA, Internet Mail Consortium, IMC) künftig durch alle Internet-Protokolle berücksichtigt werden. IANA gibt die Schreibweise als UTF-8 vor - wie auch die exakte Schreibweise der anderen Formate (siehe Abbildung weiter unten). Die RFC 3629 legt die Codierung und die zugehörigen Algorithmen fest.

Buchstabe *y*	U+0079	00000000 01111001	01111001	0x79
Buchstabe *ä*	U+00E4	00000000 11100100	11000011 10100100	0xC3 0xA4
Zeichen für eingetragene Marke ®	U+00AE	00000000 10101110	11000010 10101110	0xC2 0xAE
Euro-Zeichen €	U+20AC	00100000 10101100	11100010 10000010 10101100	0xE2 0x82 0xAC

Zeichen	Unicode	Unicode binär	UTF-8 binär	UTF-8 hexadezimal

Big und Little Endian

- Womit arbeitet Mac / Windows?
  - Windows mit Little Endian (Intel CPU)
  - Mac mit Big Endian (Motorola CPU)

- Was ist unter welchen Umständen besser?
  - Big Endian:
    The numbers are also stored in the order in which they are printed out, so binary to decimal routines are particularly efficient.

  - Little Endian:
    Also, because of the 1:1 relationship between address offset and byte number (offset 0 is byte 0), multiple precision math routines are correspondingly easy to write.

- Wofür braucht man BOM (Byte Order Mark)? Hint:
  http://www.unicode.org/faq/utf_bom.html
  - Gibt die verwendete Codierung an.

- Was ist das Unicode Consortium?
  - Zielsetzung
    - Entwicklung eines weltweit gültigen Zeichensatzes für alle lebenden und toten Sprachen.
  - Seit wann?
    - 3. Januar 1991
  - Aus welcher "Ecke" (wie so vieles) kommen auch diese Ideen?
    - Sitz in Kalifornien, bestehend aus den big Playern der Branche

ASCII Conversion Chart

04 November 2014 08:45

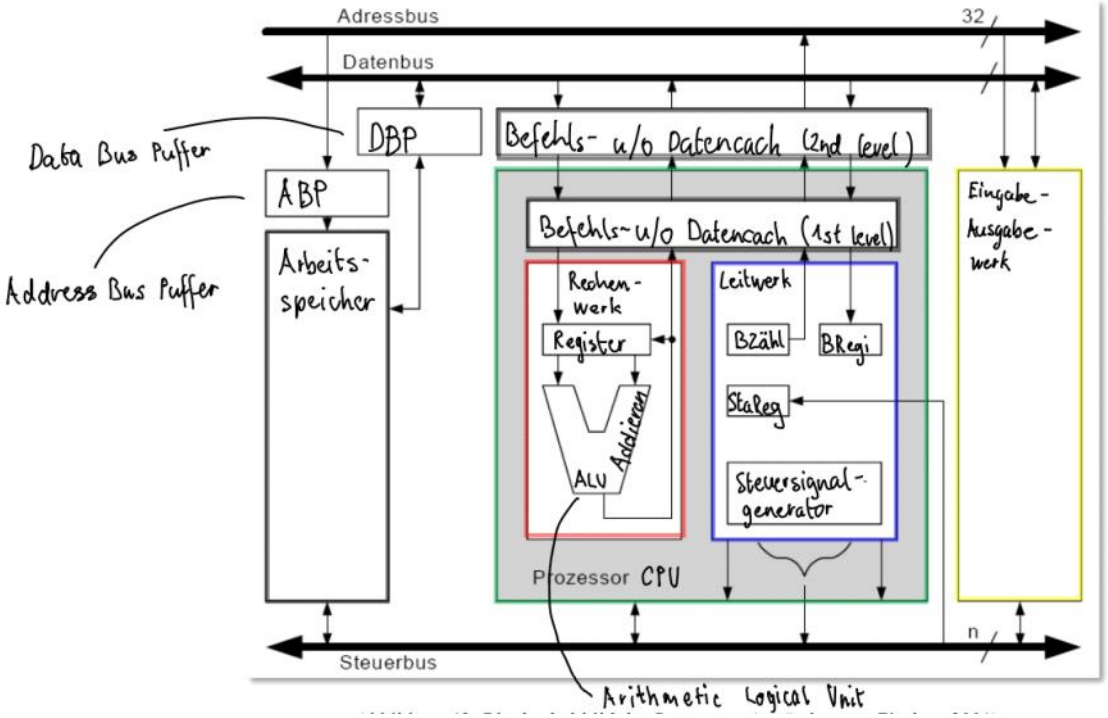
Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Architektur Mikroprozessor

07 October 2014 09:26

Aufbau Mikroprozessor



Rechenwerk und Cache

ALU

- Arithmetic Logical Unit
- Kann nur addieren
- Trifft logische Entscheide (XOR)
- FPU = Floating Point Unit
- Moderne Prozessoren bestehen aus mehreren ALU's und FPU's
- Register = Prozessor eigene 64bit Speicherbereiche

Leitwerk und Steuerwerk

BZ = Befehlszähler (Counter)
Zeigt auf den nächsten Befehl

BR = Befehlsregister
Holt Befehl aus Katalog

SR = Statusregister
Status der Hardware

Arbeitsspeicher

- Aufbau wie ein Hochhaus
- Daten werden von oben nach unten gespeichert
- Programmcode wird von unten nach oben gespeichert

Bus

Parallele Leistungssystem

- Es gibt Adress-, Daten- und Steuerbus
- Heutige Busse sind 64 Bit
- Der Holzi hat ein nur ein Steuerbus für Read und Write (minimal)

Busstypen:

- Adressbus - Verantwortlich für die Geschwindigkeit, überträgt Adressen (i7 auf 36 Leitungen)
- Datenbus - Verantwortlich für das Speichervolumen, überträgt Daten (i7 auf 64 Leitungen)

Holzi

06 January 2015 14:43

Assembler Sprache — Zeit

Hochschule Luzern
Wirtschaft

Mnemonic	Code	Worte	Zyklen	Beschreibung
				Datentransferbefehle
MVI R0	0100	2	8	unmittelbar folgendes Wort in Register 0
MVI R1	0101	2	8	unmittelbar folgendes Wort in Register 1
STO R0	0000	2	10	Register 0 in RAM; 2. Wort ist Ziel-Adresse
STO R1	0001	2	10	Register 1 in RAM, 2. Wort ist Ziel-Adresse
LD R0	0010	2	10	RAM in Register 0; 2. Wort ist Quell-Adresse
LD R1	0011	2	10	RAM in Register 1; 2. Wort ist Quell-Adresse
MOV R1, R0	1001	1	5	Register 0 in Register 1 kopieren
MOV R0, R1	1010	1	5	Register 1 in Register 0 kopieren
				Input-/Outputbefehle
IN	1000	1	7	Input-Schnittstelle in Register 0
OUT	1011	1	7	Register 0 an Output-Schnittstelle
				Arithmetische Befehle
ADD R1	1101	1	5	R1 + R0 (ohne carry); Resultat in R0, 5. Bit in carry
				Rotationsbefehle
1. ASL	1110	1	5	Register 0 links schieben, MSB in carry
2. RAR	1111	1	5	R0 zyklisch rechts rollen, LSB in carry
				Sprungbefehle
JMP	0110	2	8	Sprung zu ROM-Adresse im 2. Wort
JC	0111	2	8	Sprung zu ROM-Adresse im 2. Wort, wenn carry = TRUE
JNC	1100	2	8	Sprung zu ROM-Adresse im 2. Wort, wenn carry = FALSE

1. Jumpbo
2. Adress

Entscheidung

Abbildung 45: Befehlssatz des Holzi (Fischer, 2001)

RAM-Adressen in 2-Wort-Befehlen:
0000 RAM 16
0001 RAM 17
0010 RAM 18
0011 RAM 19

Holzi-Beispiele im Unterricht *(notieren Sie sich die Codes gründlich):*

1. Unverzweigt: Erstellen Sie ein Programm in Holzi-Assembler, welches am Eingang zwei Summanden abholt, sie addiert und das Resultat an den Ausgang legt.
2. Verzweigt: Erstellen Sie ein Programm in Holzi-Assembler, welche eine an Input liegende Ganzzahl wie folgt zu untersucht: Ist die Zahl grösser als 7, ist sie an Output zu geben. Andernfalls ist der Wert 0000 an Output zu geben.

1. 1011 → 0110) ASL → verdoppelt

2. 1011 → 0101) RAR → Division :10

Aufgabe 1

IN	1000
MOV R1, R0	1001
IN	1000
ADD R1	1101
OUT	1011

Aufgabe 2

ROM Adress	Command	Assembler

1	1000	IN
2	1110	ASL
3	1100	JC
4	0111	7
5	0100	MVI R0
6	0000	0
7	1111	RAR
8	1011	OUT

8.4 Stichworte zur Leistungssteigerung bei Prozessoren:

Die meisten technologischen Innovationen in der Weiterentwicklung von (Mikro-) Prozessoren gelten der Geschwindigkeit in der Befehls- und Datenverarbeitung. Einige der folgenden Stichworte werden in der Vorlesung angesprochen, andere sind im Lexikon zu finden:

- Taktfrequenz und Übertaktung
- Harvard Architektur im Innern
- Co-Processoring
- Caching, Cache Levels
- Superskalarität
- Hyperthreading
- Multi Coring / Multi Processoring
- Prefetching
- Pipelining (siehe Illustration)
- Branch Prediction, Speculative Execution

Most frequently used

spekulative Verarbeitung

Voraussage

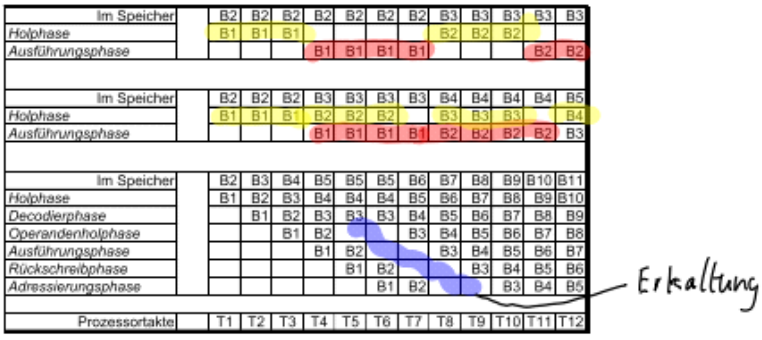


Abbildung 46: Prefetching und Pipelining (Fischer, 2001)

Multi-Coring: Mehrere eigenständige Cores

Hyperthreading: Paralelisierung der Befehle

Harvard: Eigener Speicher für Operatoren & Operanden

Prefetching & Pipelining: Bereitstellung Befehl 2 während Ausführungsphase Befehl 1

Hardware-Komponenten

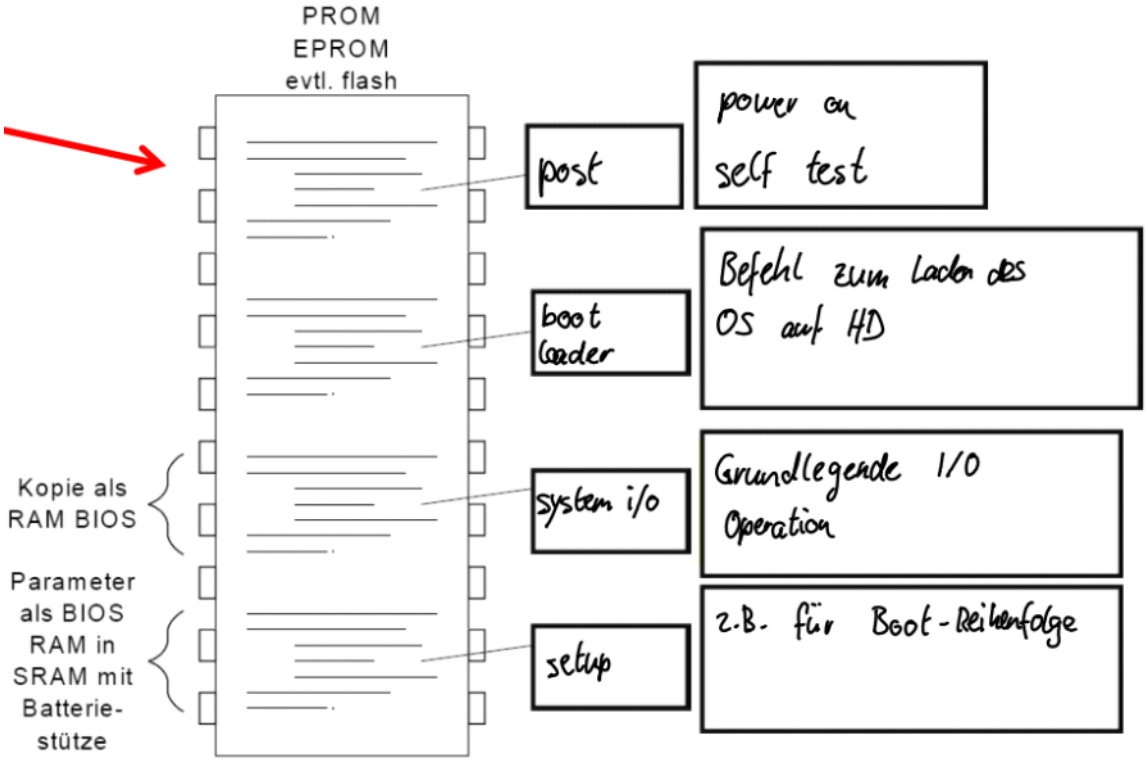
18 November 2014 09:00

Bios

Eine Chip der die Computer Firmware enthält.

Neuerdings heissen diese:
EFI - Extensible Firmware Interface
UEFI - Universalt EFI (ab 1998 von Intel)

Architektur des EPROM/PROM Speichers, welcher diese Komponente enhält:



Codegewinnung

28 October 2014 08:26

Diskretisierung

Bei der Diskretisierung wird in veschiedenen zeitlichen Abstand anhand einer Referenztabelle ein Signal digitalisiert.

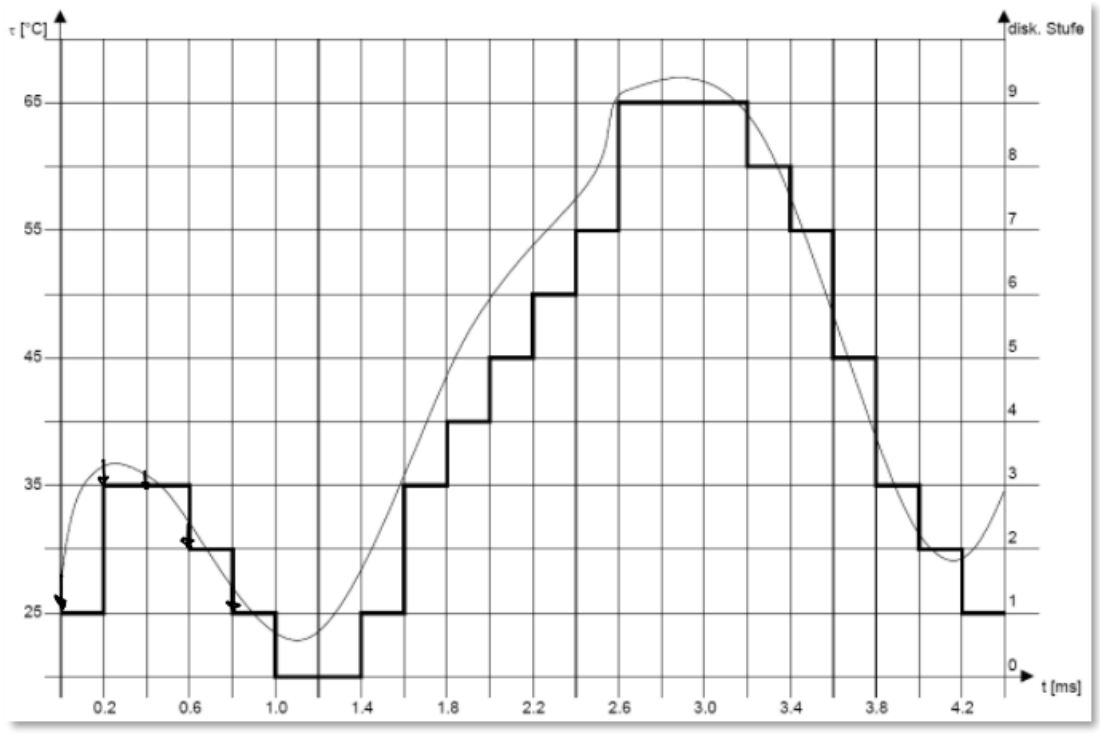


Abbildung 28: Diskretisierung eines kontinuierlichen Signals

Immer abrunden

diskrete Stufe	binäres Wort			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

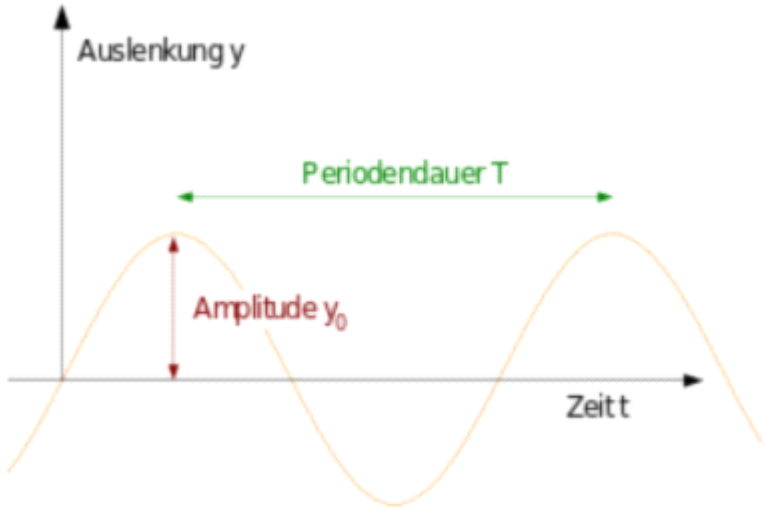
Abbildung 29: Digitalisierungstabelle

Zeitpunkt	*0.0 ms*	*0.2 ms*	*0.4 ms*	*0.6 ms*	*0.8 ms*	*1.0 ms*	*1.2 ms*	*1.4 ms*	*1.6 ms*	*1.8 ms*	*2.0 ms*
diskreter Wert	1	3	3	2	1	0	0	1	3	4	5
binärer Wert	0001	0011	0011	0010	0001	0000	0000	0001	0011	0100	0101

Abbildung 30: Übertragungsrate als Funktion von Abtastfrequenz *und* Auflösung

Modulierung

Beschreibt einen Vorgang in der Nachrichtentechnik zur Übertragung eines Nutzsignals. Dabei verändert das Nutzsignal seinen Täger.



Kehrwert der Periodendauer ist die Frequenz

F= 1/T

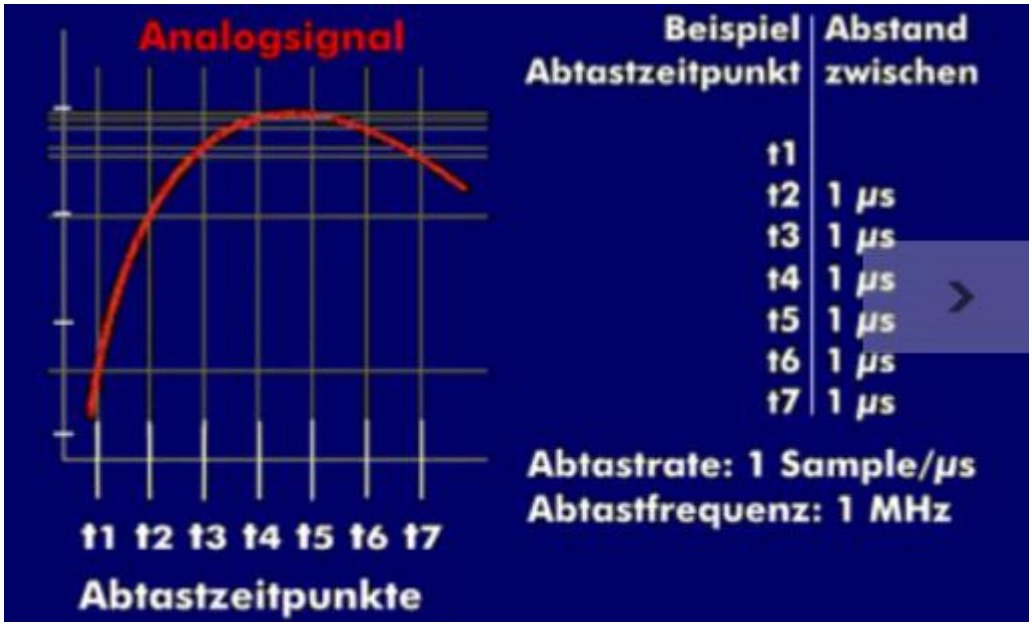
Die Einheit der Frequenz ist Hertz

1 Hz = 1 s^{-1} = 1 * 1/s

Abtastrate

Beschreibt die Frequenz mit der ein analoges Signal gemessen wird.

Samples per Second S/s



Eine ergänzende Methode ist sample and hold.

Dabei wird bei einer Entnahme einer Singlaprobe aus einer analogen Spannung der agbetastete Spannungwert für einen bestimmten Zeitraum gehalten.

Die Digitialisierung von analogen Singalen ist mit AD-Wandlern möglich.

Codierung

28 October 2014 08:47

Ziel der codierung ist es Daten mig rössmöglicher Qualität zu übertragen.

Redundanz

Kanalcodierung bzeichnet man in der Nachrichtentechnik das Verfahren digitale Daten bei der Übertragung über gestörte Kanäle mit Redundanz zu schüten.

+ Daten sind geschützt
- Es braucht mehr Bandbreite

Codesicherung

Transportkomponenten haben und machen fehler, meist in Form der Invertierung einzelner Bits.

Binäre Codes sind *dicht* oder *redundant (Erklärung in der Vorlesung)*:

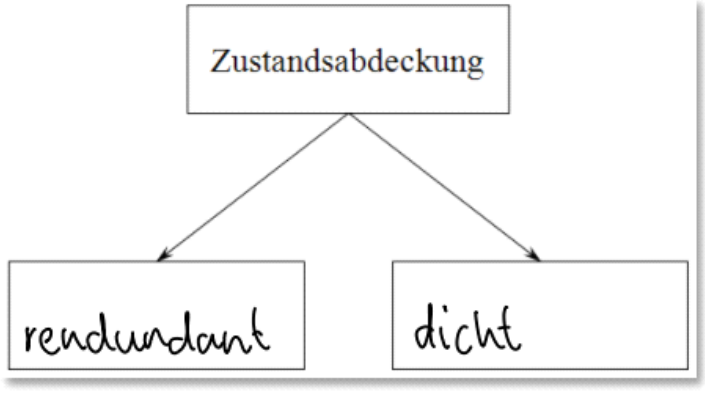


Abbildung 35: Systematik bezüglich Zustandsabdeckung

Binäre Codes sind *fehlererkennend* oder *fehlerkorrigierend (dito)*:

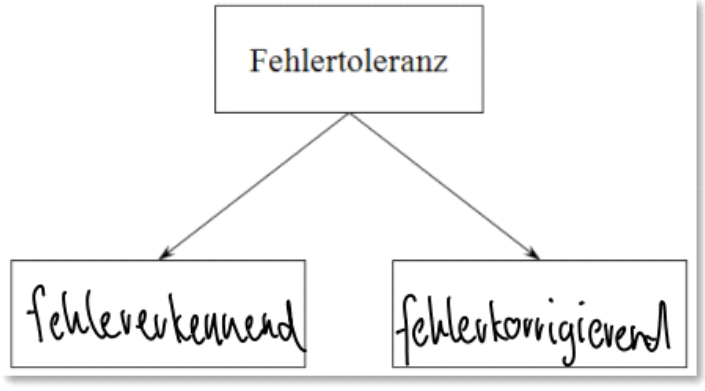


Abbildung 36: Systematik bezüglich Fehlertoleranz

EDC: error detecting codes
ECC: error correcting codes
Defekt: Mangen d.h. beeinträchtigung der Funktion.
Fehler: Dauerhafter Abbruch der Funktionalität.

Error detecting Code

Beispiel einer Korrektur mit mehreren Parity-Bits.

Codierung von "par" mit ASCII:

p: 0111 0000 > even **1**
a: 0110 0001 > even **1**
r: 0111 0010 > even **0**

Mit dem Parity-Even wird ein zustätzliches Bit mit der Information ob Quersumme an erster Stelle ungerade ist oder nicht.

Die Fehlerkorrektur hat immer eine Nachfrage zurfolge, d.h. der Datenblock muss nochmals übertragen werden.

Error correcting Code

Zwei Arten von Codes, Block-Code und Faltungs-Code.

Ein Blockcode ist eine Art von Kanalcodierung, wobei alle benutzten Codewörter diesebe Anzal ans Sysmbolen (z.B. 16 Bit) aus einem Alphabet (z.B. {0,1} haben.

Masseinheiten für Bytes

05 November 2014 19:40

Maßeinheiten für Bytes

Maßeinheit		Anzahl von Bytes	KBytes	MBytes
Byte		1		
Kilobyte (KByte)	2^{10}	1024	1	
Megabyte (MByte)	2^{20}	1.048.576	1024	1
Gigabyte (GByte)	2^{30}	1.073.741.824	1.048.576	1024
Terabyte (TByte)	2^{40}	1.099.511.627.776	1.073.741.824	1.048.576
Petabyte (PByte)	2^{50}	1.125.899.906.842.624	1.099.511.627.776	1.073.741.824
Exabyte (EByte)	2^{60}	1.152.921.504.606.846.976	1.125.899.906.842.624	1.099.511.627.776

Fehlerkorrektur

28 October 2014 09:39

Bei der Übetragung können Signale ungenauer, verszert oder verzögert werden.

Fehlerursachen

Rauschen

In einem Medium können die Leitmaterialien druch verschmutzung zur Verzerrung des Signals führen.

Kurzzeitstörungen

Elektronische Funken oder Kratzer auf CDs die kurzzeitig die Spannung beeinflussen.

Signalverformung

Das Übetragungs Signal wird verform [ ] >> ( ).

Nebensprechen

Benachbarte Digitalkanäle haben Einfluss auf das zu übertragende Signal, z.B. durch kapazitive Kopplung.

Fehlerarten

Einzelbitfehler

Treten unabhängig von anderen auf.

Bündelfehler

Treten abhängig von anderen Fehler auf >> Folgefehler.

Synchronisationsfehler

Ist ein längerer Bündelfehler bei dem neben dem Verlust des Inhalts auch die Information verloren geht wieviele Symbole verloren gegangen sind.

Dies hat zur Folge, dass auch die folgenden Informationen nicht mehr korrekt gelesen werden können.

Fehlerkorrektur-Codes (ECC)

- ***Annäherung 1*: ECC auf dem Stern Duoland**
- Das Alphabet auf Duoland kennt die Buchstaben j und n.
- Zwecks Fehlerkorrektur wird ein **Drei-Bit-Code** fürs Kommunizieren im Duonet vereinbart:
- j = 111, n = 000
- Bei der Übermittlung können 1-Bit-Fehler vorkommen.

Empfangen	Gesendet
000	000
001	000
010	000
100	000
011	111
101	111
110	111
111	111

ECC - Parity-Prüfung

04 November 2014 08:55

Beispiel einer einfachen 1-Bit-ECC-Codierung

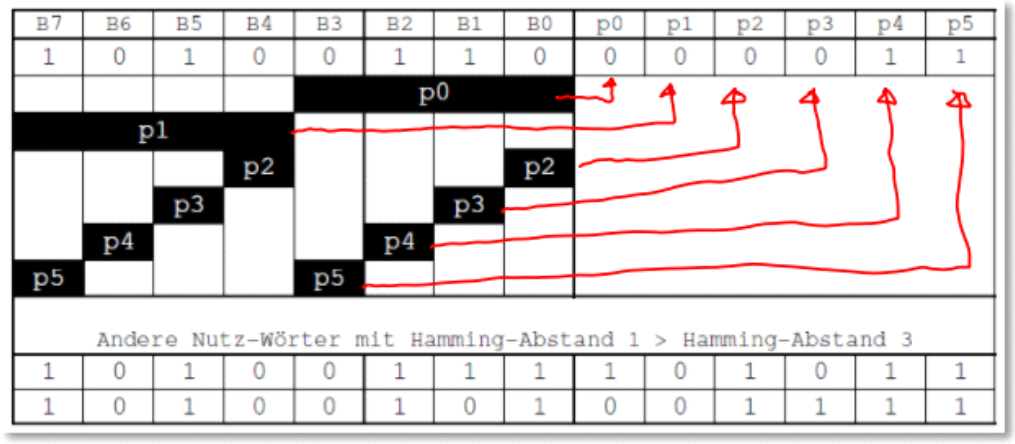


Abbildung 37: Illustration der Idee der fehlerkorrigierenden Codierung (www.pfischer.doz.fhz.ch)

Jedes Bit wird doppelt kontrolliert.

Eindimensionale Prüfung

Ziel ist es aus einem "Nicht-fehlererkennenden" code einen 1-fehlererkennenden Code zu machen.

Die Umsetzung erfolgt mit dem Hinzufügen von einem zusätlichen Parity-Bit.

Für dieses Bit filt die even parity Regel:

```
Anzahl 1 in Codewort + Parity-Bit = Gerade
```

Beispiel mit einem 7 Bit Codewort:

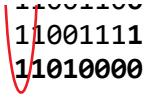
```
1001 0000
1001 0011
```

Zweidimensionale Prüfung

Bei der Zweidimensionalen Prüfung werden ganze Code-Blöcke überprüft.

Dabei wird mit der gleichen parity Regel für jede Zeile eine Parity-Bit gesetzt und in der Schluss-Zeile wird für jede Spalte ein Bit gesetzt.

```
11000011
11000101
11000110
11001001
11001010
11001100
11001111
11010000
```



Mit dieser Art von Prüfung können nun die Fehlerhaften Bits gefunden werden.

Codes mit zweidimensionaler Parity-Prüfung haben mindestens den Hammingabstand 3.
-> 3 fehlererkennend und 1 fehlerkorrigierend.

k aus n Code

28 October 2014 10:15

n ist die länge des Bitwortes
k ist die Anzahl gesetzten (1) Bits im Codewort

00011 >> n=5, k=2

Hammingabstand eines Codes

Das Hamminggwicht entspricht dem k.

Der Hammingabstand entspricht den Anzahl von Binärstellen an denen sich zwei Bitwörter unterscheiden.

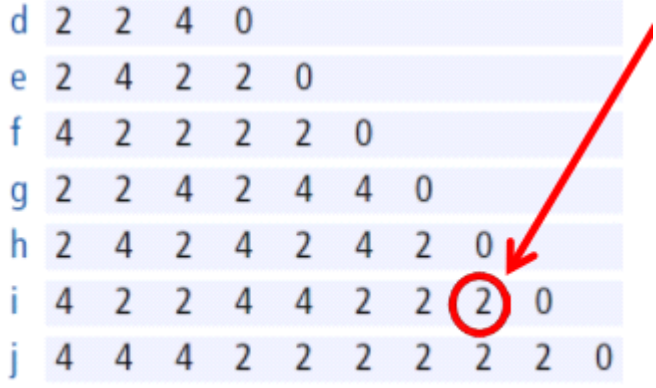
Beispiel mit 2 aus 5 Code:

c 00110
g 1**0001**

>> Hammingabstand ist 4

Mit den Codewörtern {a, …, j} kann man eine Tabelle erstellen:

	a	b	c	d	e	f	g	h	i	j
a	0									
b	2	0								
c	2	2	0							
d	2	2	4	0						
e	2	4	2	2	0					
f	4	2	2	2	2	0				
g	2	2	4	2	4	4	0			
h	2	4	2	4	2	4	2	0		
i	4	2	2	4	4	2	2	2	0	
j	4	4	4	2	2	2	2	2	2	0



Der Hammingabstand d dieses Codes ist der kleinste auftretende Abstand, sprich 2.

Es können Fehler erkannt werden, die weniger als d Bits betreffen (<2).

Es können Fehler korrigiert werden, die weniger als d/2 Bits betreffen (<2/2)

>> Der 2 aus 5 Code kann 1 Fehler erkennen.
>> Der Code kan keine Korrektur durchführen.

Hamming Codes

03 November 2014 19:50

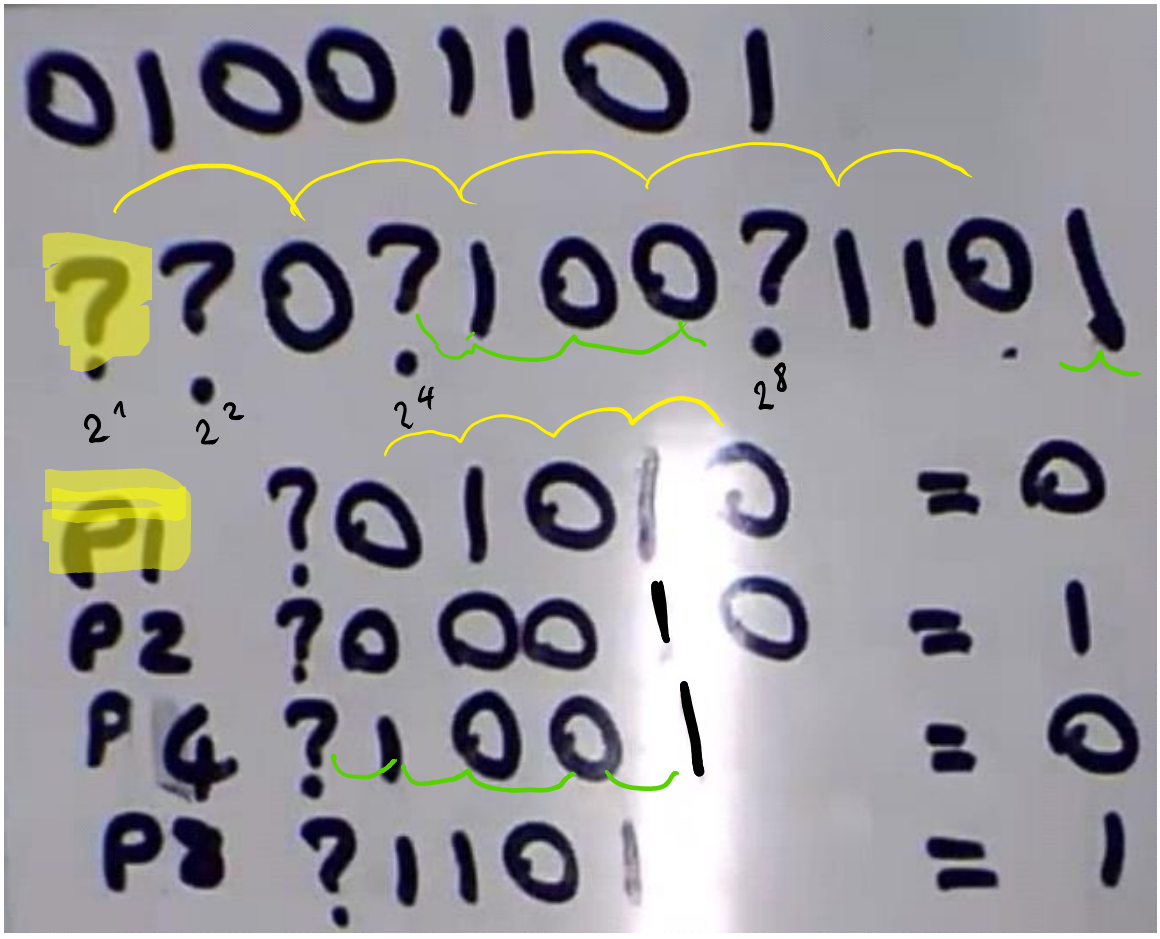
Der einfachste Hamming-Code ist ein (7,4)-Code, der eine Länge von 7 Bits hat, wovon allerdings nur 4 Bits Nutzinformationen sind und die restlichen 3 Bits zur Fehlerkorrektur dienen.
Dieser Hamming-Code ist ein 1-fehlerkorrigierender Code mit einem Hammingabstand von 3.

Berechnung

Im Allgemeinen gilt:

Es gibt Hamming-Codes der Länge:
`2r-1`
sofern folgendes gilt:
`r>=2`
Anzahl Korrekturbits:
`r`
Informationsbits:
`2r-1-r`

Beispiel:

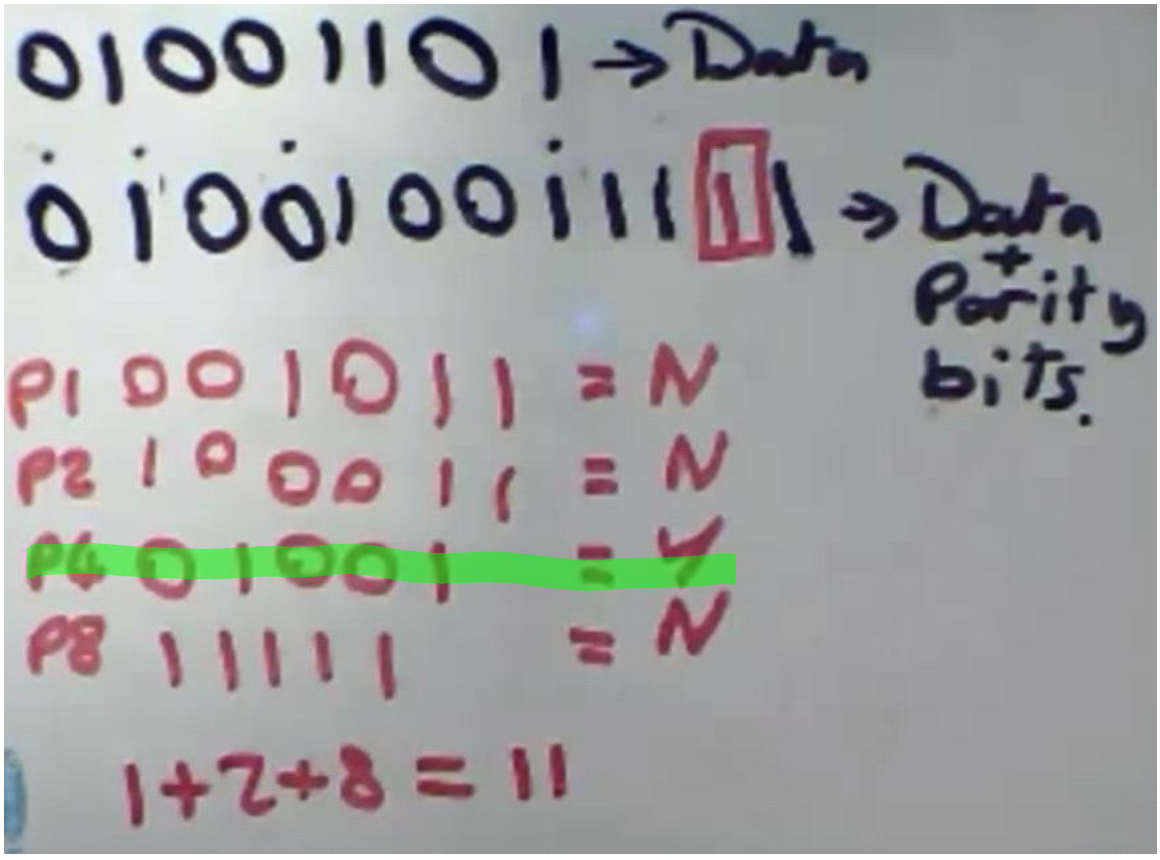


Die Berechnung des Parity Bits besteht aus den Bits die der Stelle des Parity Bits folgen.
Dabei werden immer Anzahl entsprechend der Parity Bit Stelle übersprüngen.

P1 > Start bei Position 1, Überspringen 1 Bit
…

P4 > Start bei Position 4, Überspringen 4 Bit

…



Bei der Fehlerüberprüfung werden dann die Parity Bits neu berechnet. Die Positionen, die nicht stimmen werden zusammengezählt, das Ergebnis entspricht dann der Position des fehlerhaften Bits.

CRC-Kodierung

04 November 2014 09:45

- CRC erkennt Fehler bei der Übertragung oder Speicherung.
- CRC kommst gerade beim Ethernet Standard zum Einsatz.
- Die Berechnung des CRC Werts beruht auf Polynomdivision.

Beispiel:

https://www.youtube.com/watch?v=MSAog5MEhrs

Als erstes wird ein neues Codewort durch Polynom

```
1001 1101
x^7+x^4+x^3+x^2+x^0
```

$X^4 + X^2 + X + 1$
1 0 1 1 1

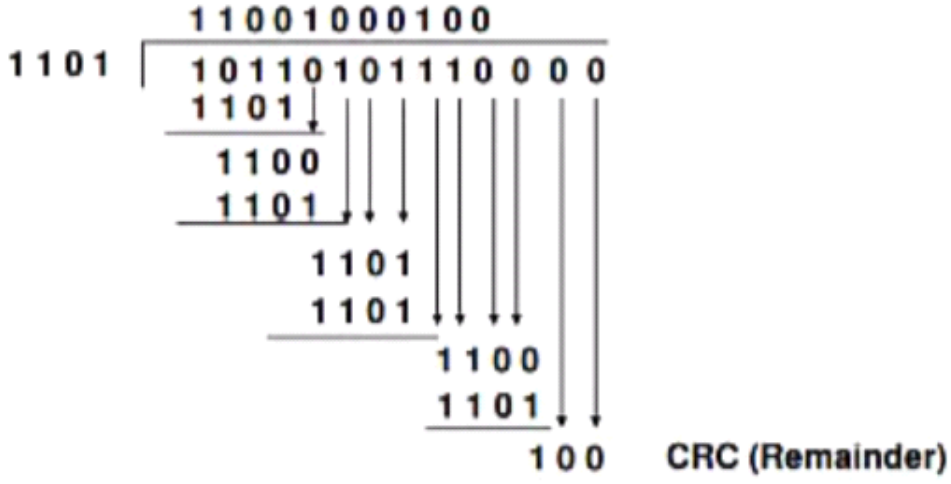
$X^5 + X^4 + X$
1 1 0 0 1 0

We want the sender and receiver to agree on a generator polynomial. The degree of the generator polynomial is the size of the CRC.

Size of the CRC = 4

Verfahren zur Berechnung

Original Message: 10110101110
Generator: 1101
Message after 3 zero bits appended: 10110101110000



Transmitted Frame: 1 0 1 1 0 1 0 1 1 1 0 1 0 0

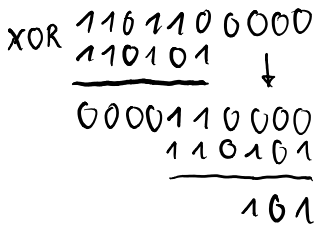
CRC Beispiel

10 January 2015 13:35

Generatorpolynom 5. Grades: 110'101
Anzahl Zeichen: 6

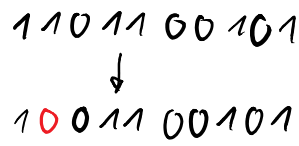
Nutzdaten zum Übertragen 1'1011
Anhang: 6-1=5
Nutzdaten mit Anhang ergänzen: 11'011**0'0000**

Berechnung CRC Anhang mit XOR:

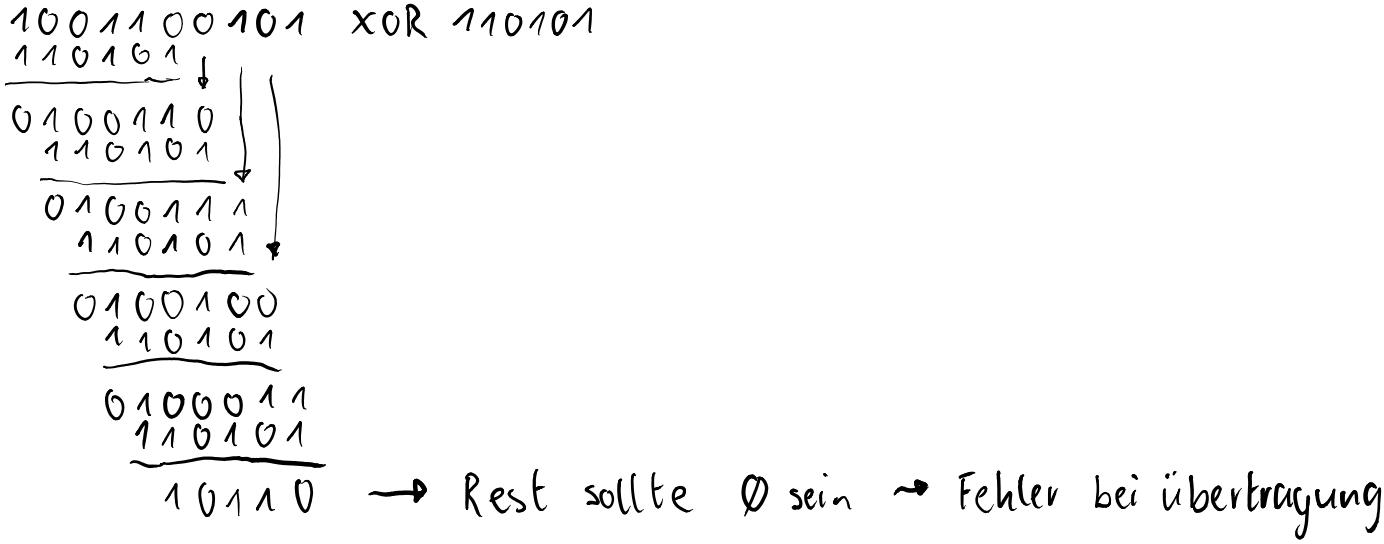


Nutzlast = Nutzdaten + Anhang + Rest

Annahme: Bei Übertragung 1 fehlerhaftes Byte



Prüfung des übertragenen Wertes



Booleschen Algebra

04 November 2014 10:02

In der Booleschen Algebra werden mit logischen Operatoren gerechnet.

OR, AND und NOT

Wahrheitstabelle **OR**:

a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

Symbol **OR**:

Wahrheitstabelle **AND**:

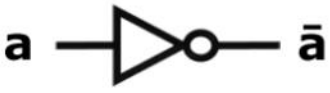
a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

Symbol **AND**:

Wahrheitstabelle **NOT**:

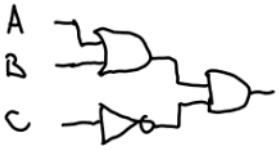
a	NOT a
0	1
1	0

Symbol **NOT**:



Beispiel:

Zeichnen Sie mit Bleistift und Papier und offiziellen Symbolen den booleschen Ausdruck **(A + B)** $\overline{C}$



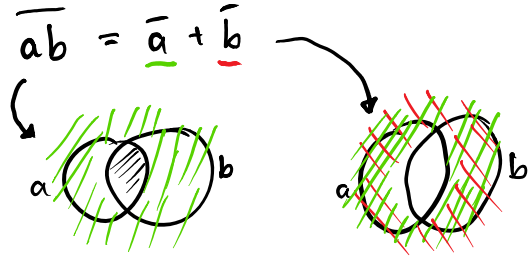
Beispiel Und-Ausdruck:

Wichtige Gesetze

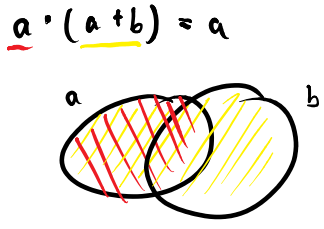
Name	Formel	Kurzschreibweise
Kommutativgesetze	a * b = b * a a + b = b + a	$ab = ba$
Assoziativgesetze	a * (b * c) = (a * b) * c a + (b + c) = (a + b) + c	$a(bc) = (ab)c$
Distributivgesetze	a * (b + c) = (a * b) + (a * c) a + (b * c) = (a + b) * (a + c)	$a(b + c) = ab + ac$ $a + bc = (a + b)(a + c)$
Identitätsgesetze	a * 1 = a a + 0 = a	$a1 = a$
Null-/Einsgesetze	a * 0 = 0 a + 1 = 1	$a0 = 0$
Komplementärgesetze	a * -a = 0 a + -a = 1	$a\bar{a} = 0$ $a + \bar{a} = 1$
Idempotenzgesetze	a * a = a a + a = a	$aa = a$
Verschmelzungsgesetze	a * (a + b) = a a + (a * b) = a	$a(a + b) = a$ $a + ab = a$
De Morgan'sche Gesetze	-(a * b) = -a + -b -(a + b) = -a * -b	$\overline{ab} = \bar{a} + \bar{b}$ $\overline{a + b} = \bar{a}\bar{b}$
Doppeltes Negationsgesetz	-(-a) = a	$\bar{\bar{a}} = a$

* = UND
+ = ODER
- = NOT

Beweis De Morgan'sche Gesetz:



Beweis Verschmelzungsgesetz:



Wahrheitstabelle

$\overline{a \cdot b} = \bar{a} + \bar{b}$

a	b	$\bar{a}$	$\bar{b}$	$\bar{a} + \bar{b}$	$a \cdot b$	$\overline{a \cdot b}$
0	0	1	1	1	0	1
0	1	1	0	1	0	1
1	0	0	1	1	0	1
1	1	0	0	0	1	0

How to real:

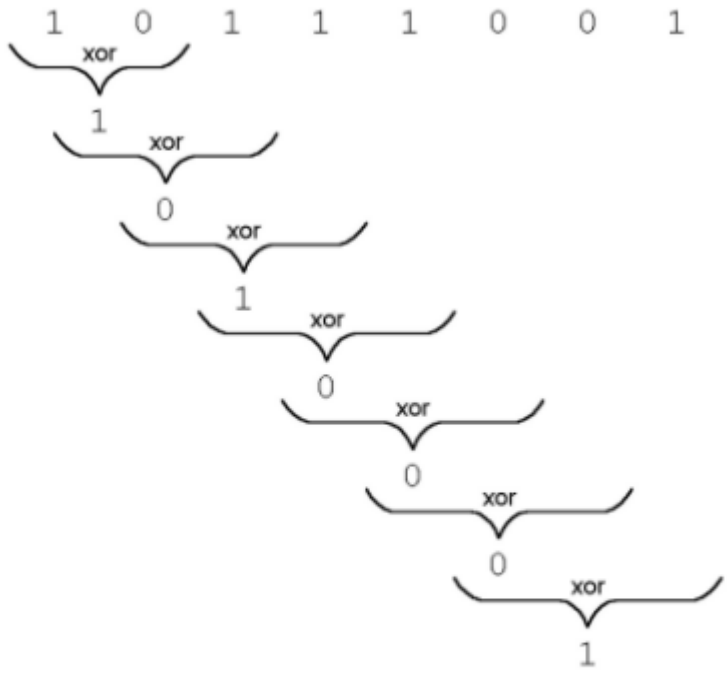
a+(-a+b)=a+b

a+b:
>**0111**

u:-a*b: a+u
1100 0011
0101 0100
>0100 >**0111**

Antivalenz

So bestimmt man die gerade parität eines n-Bitworts.



P(n-Wort) = LSB xor P(n-Wort div 2)
P(0) = 0

Disjunktive Normalform

11 November 2014 09:23

Folgende Sätze existieren:

1: Alle zweistelligen booleschen Funktionen können mit Hilfe der Negation (-) , der Konjunktion (* und der Diskunktion (+) dargestellt werden.

2: Alle zweistelligen booleschen Funktionen können entweder mit Hilfe der Negation und der Konjunktion, oder mit Hilfe der Negation und der Disjunktion dargestellt werden.

3: Alle zweistelligen booleschen Funktionen können entweder mit Hilfe der NAND-Verknüpfung oder mit Hilfe der NOR-Verknüpfung dargestellt werden.

Disjunktion (+)

$\overline{ab} = \bar{a} + \bar{b}$ $\qquad$ $\overline{a+b} = \bar{a}\,\bar{b}$

$ab = \overline{\bar{a} + \bar{b}}$ $\qquad$ $a+b = \overline{\bar{a}\,\bar{b}}$

Beispiel

Ausdruck: $\bar{a}b + a\bar{b}$

Nur mit Diskunktion und Negation: $\overline{a+\bar{b}} + \overline{\bar{a}+b}$

Konjunktion (*)

$\overline{a+b} = \bar{a} \cdot \bar{b}$

$a+b = \overline{\bar{a} \cdot \bar{b}}$

Beispiel

Ausdruck: $\bar{a}b + a\bar{b}$

Nur mit Konjuktion und Negation: $\overline{\bar{a}b} \quad \overline{a\bar{b}}$

Liftsteuerung

Ein Steuerung für einen Lift der nur im 0 und 4-7 Stock halten soll wird mit einer UND Schaltung gebaut. Der ganze Ausdruck wird dann verkürtzt.

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

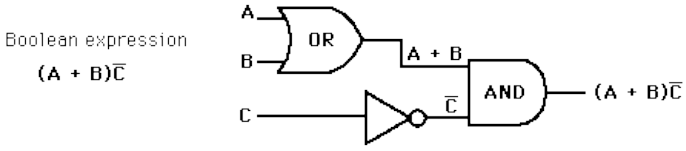
$$
\begin{aligned}
s &= \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c} + abc \\
&= \bar{a}\bar{b}\bar{c} + a\bar{b}(\bar{c} + c) + ab(\bar{c} + c) \\
&= \bar{a}\bar{b}\bar{c} + a\bar{b} + ab \\
&= \bar{a}\bar{b}\bar{c} + a(\bar{b} + b) \\
&= \bar{a}\bar{b}\bar{c} + a
\end{aligned}
$$

NAND Gate

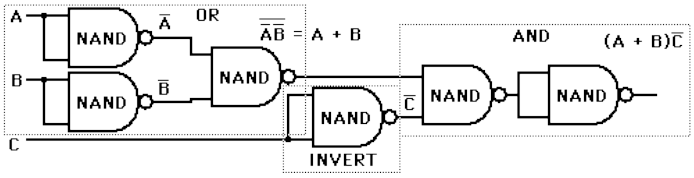
06 January 2015 16:40

NAND Gate Application

Suppose you want a high output when either A or B is high but C is low. The boolean expression and straightforward gate version of this are:

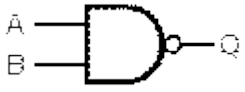


But the same task can be accomplished with NAND gates only since NAND's are universal gates. Integrated circuits such as the 7400 make this practical.



NAND

A NAND gate is an inverted AND gate. It has the following truth table:



$\overline{ab}$

Q = NOT(**A** AND **B**)

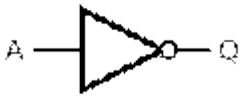
Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

NOT

A NOT gate is made by joining the inputs of a NAND gate together. Since a NAND gate is equivalent to an AND gate followed by a NOT gate, joining the inputs of a NAND gate leaves only the NOT gate.

Desired NOT Gate **NAND Construction**



Q = NOT(**A**) = NOT(**A** AND **A**)

Input A	Output Q
0	1
1	0

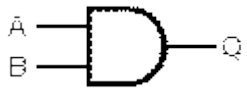
Truth Table

$\overline{aa}$

AND

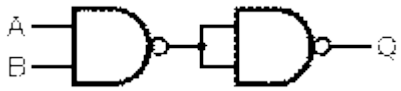
An AND gate is made by following a NAND gate with a NOT gate as shown below. This gives a NOT NAND, i.e. AND.

Desired AND Gate



Q = **A** AND **B**

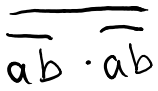
NAND Construction



= NOT[NOT(**A** AND **B**) AND NOT(**A**AND **B**)]

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table



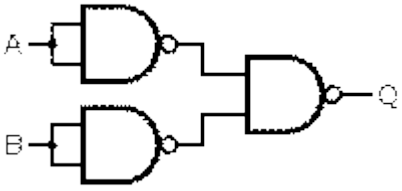
OR

If the truth table for a NAND gate is examined or by applying De Morgan's Laws, it can be seen that if any of the inputs are 0, then the output will be 1. To be an OR gate, however, the output must be 1 if any input is 1. Therefore, if the inputs are inverted, any high input will trigger a high output.

Desired OR Gate

Q = **A** OR **B**

NAND Construction



= NOT[NOT(**A** AND **A**) AND NOT(**B**AND **B**)]

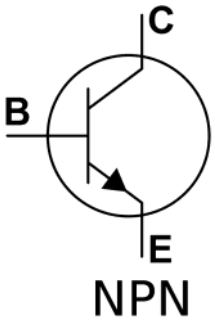
Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table

Le Transistor

11 November 2014 08:34

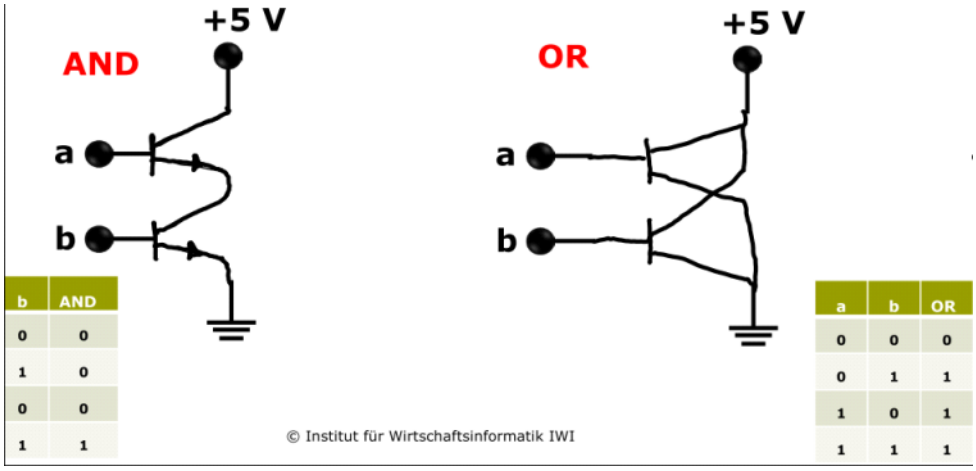
Mittels geschickter Kombination von Transisitoren kann man NOT, AND und OR-Schaltungen erzeugen.
Transistoren sind elektronische Schalter, die zwei Zustände haben: Ein und Aus.



Kleiner Basistrom (B-Basis) kann grossen Stromfluss (C-Collector) steuern. Die Ausgabe (E-Emitter) wird dann an den Verbraucher geleitet.

Das verhältniss von E und B ist die Verstärkung

Logische Schaltungen



b	AND
0	0
1	0
0	0
1	1

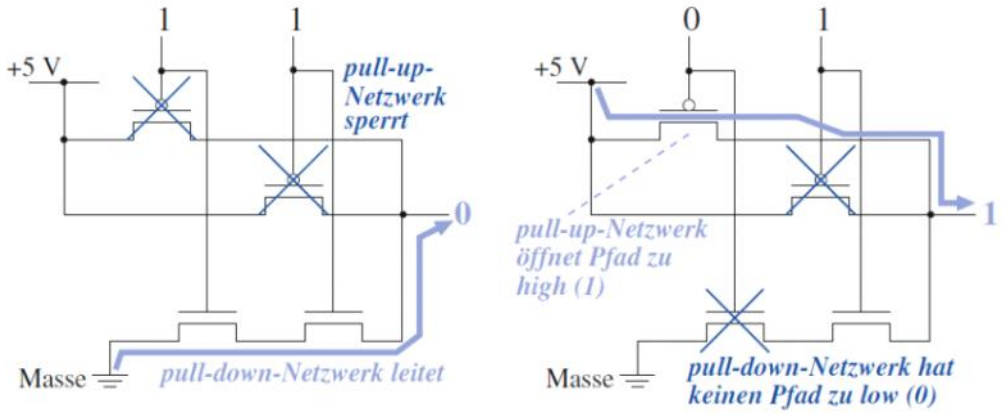
a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

NAND- und NOR-Schaltungen

a	b	NAND
0	0	1
0	1	1
1	0	1
1	1	0

a	b	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Die NAND-Schaltung im Detail

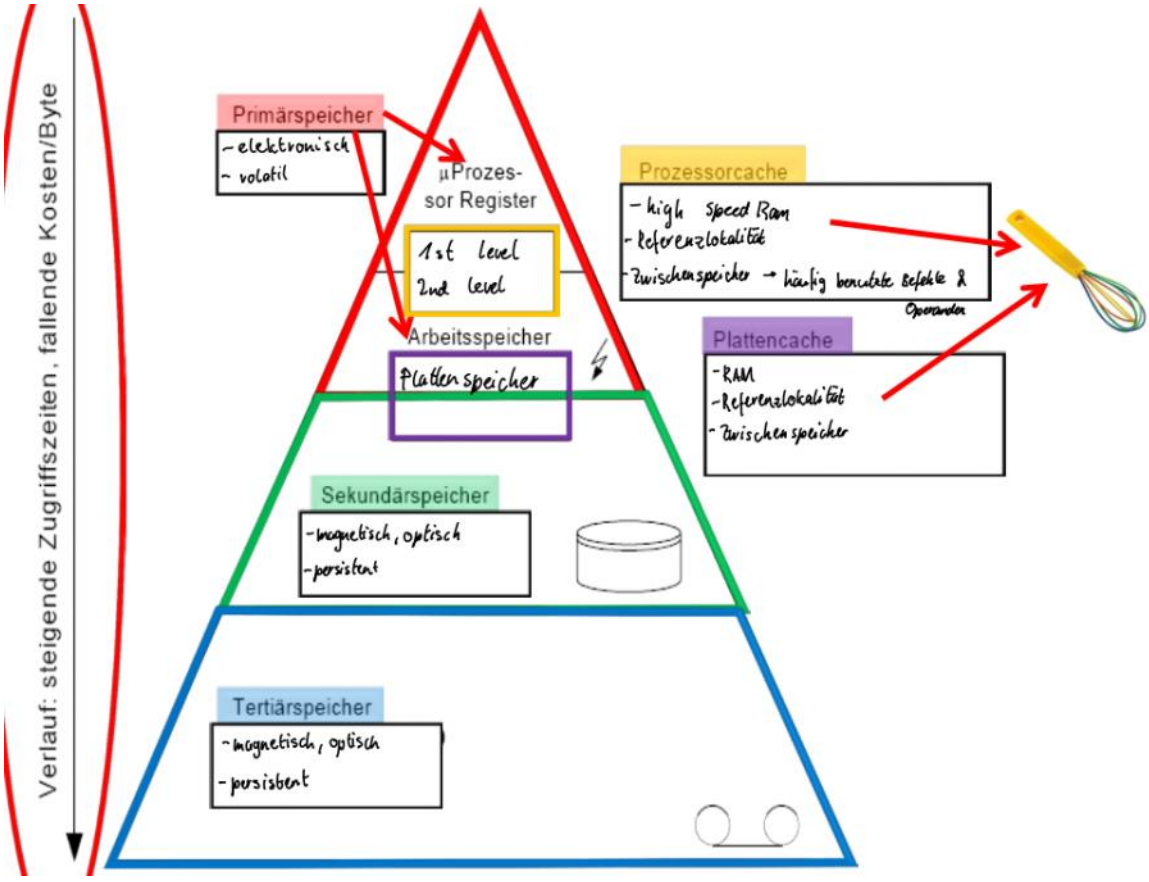

1
1
+5 V
pull-up-
Netzwerk
sperrt
0
Masse
pull-down-Netzwerk leitet
0
1
+5 V
pull-up-Netzwerk
öffnet Pfad zu
high (1)
1
Masse
pull-down-Netzwerk hat
keinen Pfad zu low (0)

Speicher

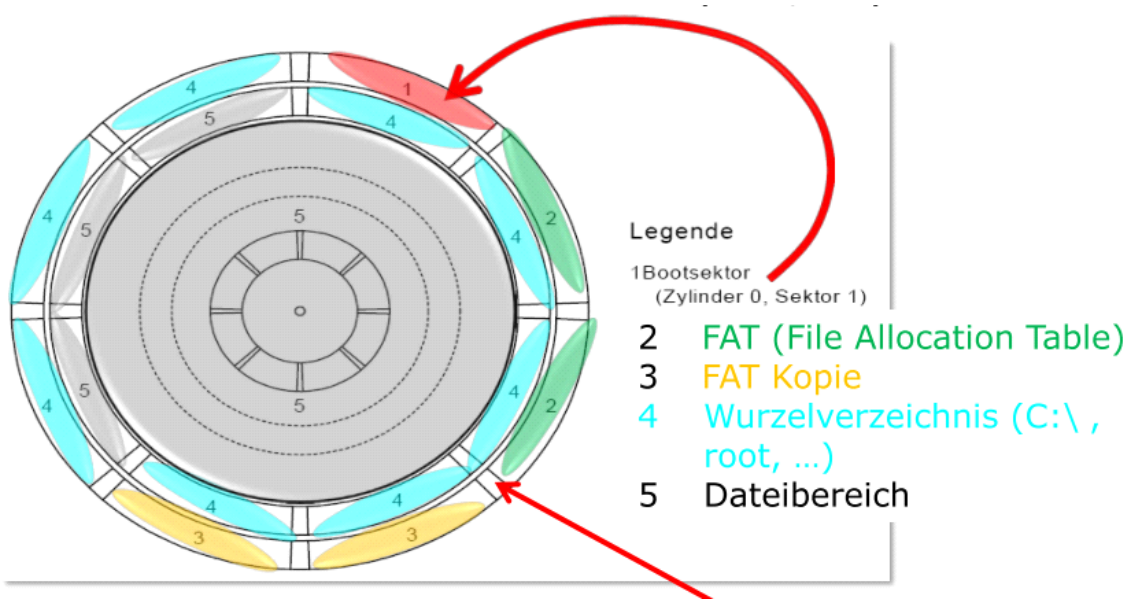
18 November 2014 08:41

Die Speicherhierarchi beschreibt den Aufbau von Speicher im Verlauf der Zugriffszeiten.

Dabei gilt: Je längere Zugriffszeiten, desto billiger ist der Speicher.

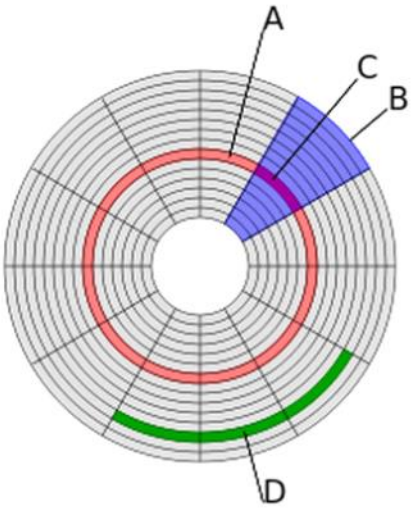


Aufbau nach Formatierung



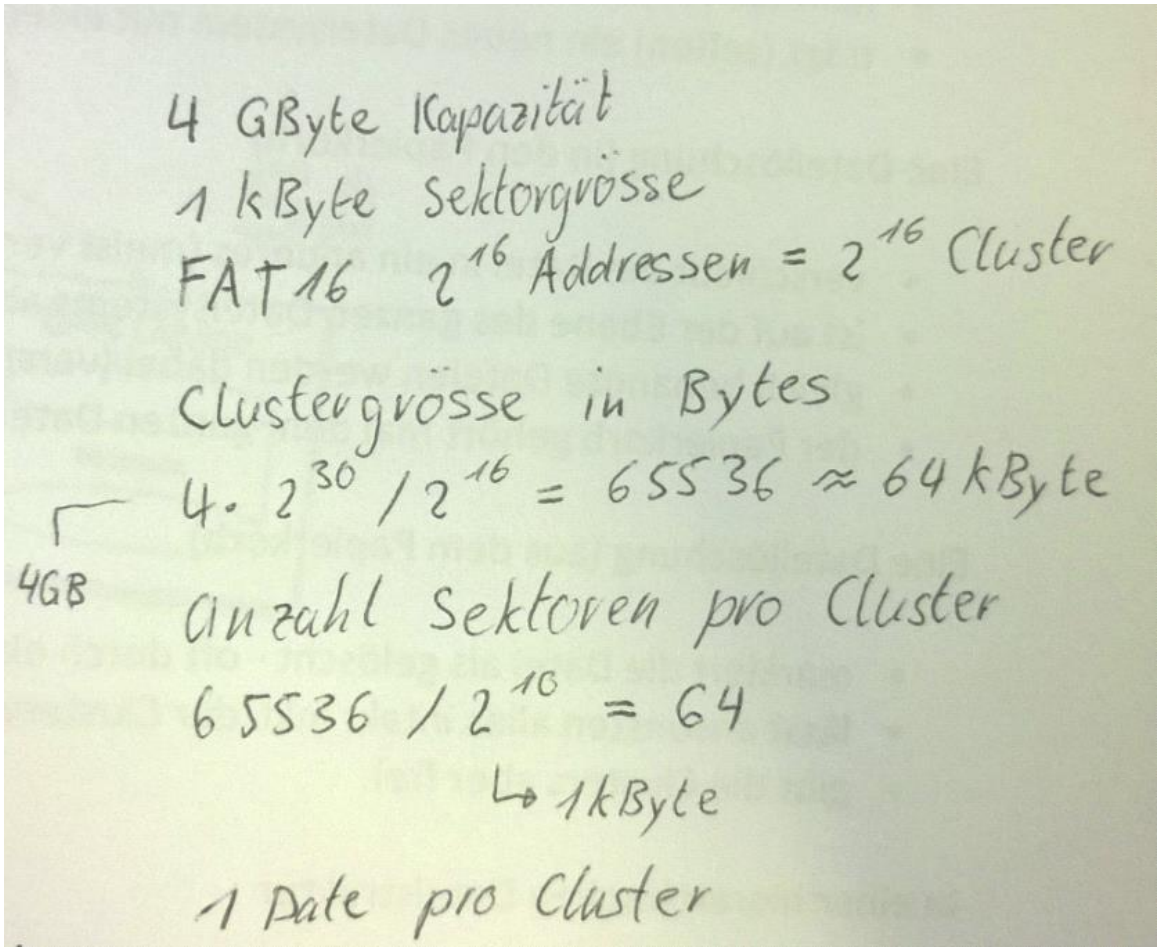
Disk-Aufbau

In computer file systems, a cluster or allocation unit is a unit of disk space allocation for files and directories. To reduce the overhead of managing on-disk data structures, the filesystem does not allocate individual disk sectors by default, but contiguous groups of sectors, called clusters.



Disk structure:
(A) track
(B) geometrical sector
(C) track sector
(D) cluster

Track (Spur): ein Ring auf einer Disk
Cylinder: meherere Disks gestapelt
Zone (geomtrischer Sektor): Ein Auschnitt der Disk
Sektor: Stück von Track in einem geometrischen Sektor
Cluster (sektorenverbund): mehrere Sektoren verbunden



FAT32 -> 228 -> 268'435'456 Sektoren

Cluster Grösse 512 Byte (-32768 Byte -> 31kB)
HD Grösse -> 2^28*512=1E11
2^28*32768=8E12

Je nach Grösse der Datensätze lohnt es ich die Cluster Grösse zu vergrössern oder zu verkleinern.

Formatierung

Schnellformatierung: Löschen der FAT
Volle Formatierung: Löschen aller Daten auf der Disk
Dateilöschung: Adresse zu Datei wird gelöscht
Dateilöschung aus Papierkorb: Datei wird physisch auf Disk gelöscht

- Low Level Formatierung
- initialisiert die Platte magnetisch
- trägt die physikalischen Eigenschaften wie
  - Interleaving-Faktor und
  - Zone Bit Recording (siehe je Lexikon)
  - oder vergleichbare Technologien auf.
- Sie wird heute durch die Hersteller vorgenommen und kann/sollte vom Anwender nicht mehr verändert werden.

Volle (High Level) Formatierung

- löscht alle Dateieintragungen
- löscht damit auch den Papierkorb
- überprüft die Qualität der Clusters und markiert sie u.U. als schlecht

- nullt alle Status-Eintragungen (ausser Bad Cluster)
- trägt ein neues Dateisystem mit einem leeren Index ein.

Schnellformatierung

- löscht alle Dateieintragungen im Index
- löscht damit auch den Papierkorb
- nullt (selten) alle Statuseintragungen (ausser Bad Cluster)
- trägt (selten) ein neues Dateisystem mit einem leeren Index ein.

Eine Dateilöschung (in den Papierkorb)

- verschiebt die Datei in ein anderes (meist verstecktes) Verzeichnis (recycled, trash, ...)
- ist auf der Ebene des ganzen Dateisystems also eine reine Umbenennung
- gleich benannte Dateien werden dabei (versteckt) umbenannt
- der Papierkorb gehört mal dem ganzen Dateisystem, mal nur einem Laufwerk.

Eine Dateilöschung (aus dem Papierkorb)

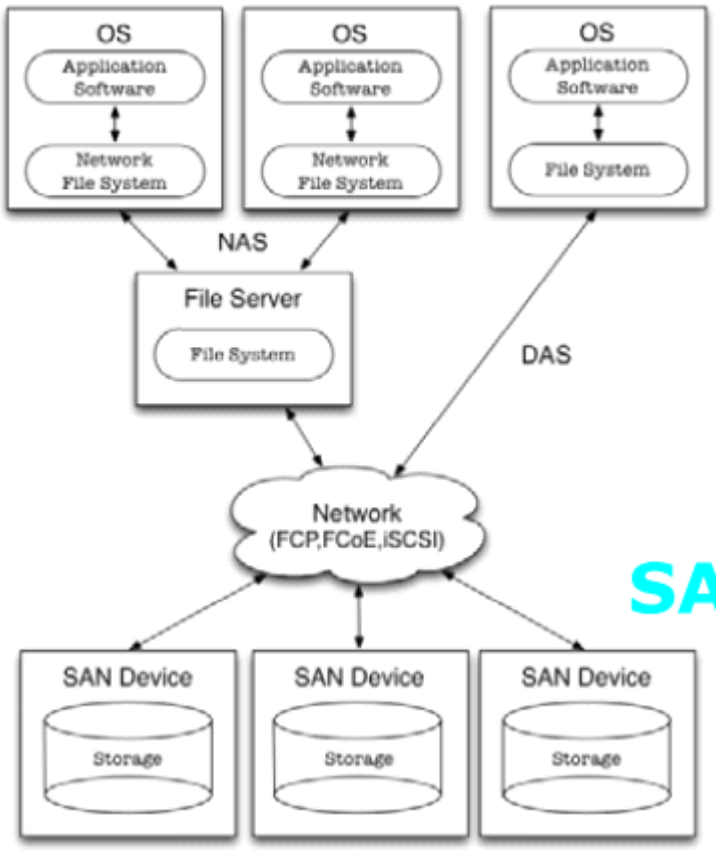
- markiert die Datei als gelöscht - oft durch eine Änderung des Dateinamens
- lässt ansonsten alles intakt inkl. der Clusterverkettung
- gibt die Clusters aber frei.

In einer hierarchischen Dateistruktur

- hat das Wurzelverzeichnis einen festen Standort: es ist statisch und hat eine beschränkte Anzahl Einträge
- werden die Unterverzeichnisse als spezielle Dateien im Datenbereich eingetragen: ihre Ablage ist dynamisch und ihre Anzahl lediglich durch die Anzahl möglicher Dateien - also letztlich durch die Anzahl Clusters - beschränkt
- haben Dateien einen partitionsweit einmaligen Namen - den absoluten Dateinamen
- ist eine Dateiverschiebung eine Umbenennung
- ist eine Verschiebung in den Papierkorb eine Umbenennung.

Business Storage

DAS: Direct Attached Storage
NAS: Network Attached Storage
SAN: Sorage Attached Network
Cloud Storage


OS
Application Software
Network File System
OS
Application Software
Network File System
OS
Application Software
File System
NAS
File Server
File System
DAS
Network
(FCP,FCoE,iSCSI)
SA
SAN Device
Storage
SAN Device
Storage
SAN Device
Storage

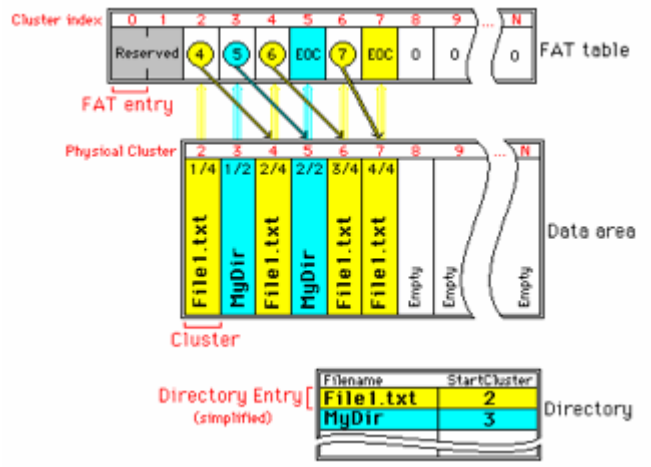
Fragmentierung

25 November 2014 08:42

Dateiindex

Such nach einer Datei geht wie folgt:

1. Such anhand Dateinamens im Dateiindex
2. Lesen der Attribute und Adresse des ersten Clusters
3. Sprung zum Index und Nachschlagen der Cluster Adress-Kette
4. Cluster-Kette sequenziell auslesen



Der Cluster index kann wie folgt aussehen:

Status	*Bedeutung*
0000	frei
XXXX	Adresse Folge-Cluster
FFF7	Bad Cluster
FFF8	Reserved Cluster
FFFF	End of File

In der FAT table wird angezeigt wo eine Datei im Data area gespeichert ist. Mit EOC wird angezeigt, dass der letzte Cluster der Datei and dieser Speicherstelle liegt.

Beispiel:

script.doc hat die Grösse M
Mc ist die Memorygrösse des Clusters
Dabei gilt:
0 < M <= MC MC < M <= 2*MC 2*MC < M <= 3*MC

	Cluster	Status
	…	….
skript.doc ->	**3ABB**	**FFFF**
	3ABC	0000
	3ABD	FFF7
	3ABE	0000
	3ABF	0000
	3AC0	0000
	3AC1	0000
	3AC2	0000
	3AC3	0000
	3AC4	0000
	3AC5	0000
	3AC6	0000
	3AC7	0000
	3AC8	0000
	3AC9	0000

	Cluster	Status
	…	….
skript.doc ->	**3ABB**	**3ABC**
	3ABC	**FFFF**
	3ABD	FFF7
	3ABE	0000
	3ABF	0000
	3AC0	0000
	3AC1	0000
	3AC2	0000
	3AC3	0000
	3AC4	0000
	3AC5	0000
	3AC6	0000
	3AC7	0000
	3AC8	0000
	3AC9	0000

	Cluster	Status
	…	….
skript.doc ->	**3ABB**	**3ABC**
	3ABC	**3ABE**
	3ABD	FFF7
	3ABE	**FFFF**
	3ABF	0000
	3AC0	0000
	3AC1	0000
	3AC2	0000
	3AC3	0000
	3AC4	0000
	3AC5	0000
	3AC6	0000
	3AC7	0000
	3AC8	0000
	3AC9	0000

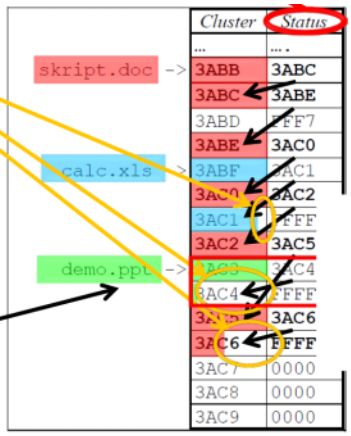
Fragmentierungstypen

Bei der Speicherung von Dateien werden Cluster fragmentiert, dabei unterscheidet man zwischen zwei Fragmentierungstypen.

Ein Cluster kann nur Informationen aus einer ganzen oder einem Teil einer datei aufnehmen. Bleibt "hinten" Platz frei, ist er verschwendet -> interne Fragmentierung.

Erstreckt sich eine Datei über mehr als einen Cluster und sind diese nicht zusammenhängend ist das externe Fragmentierung.

Interne Fragmentierung
Wenn im letzten Cluster der Platz nicht ganz aufgebraucht wird, bleibt dieser Platz "offen" d.h. für die restlichen Dateien verloren!
Externe Fragementierung
(= Plattenfragmentierung):
Die Zerstückelung von Dateien in nicht zusammenhängende Cluster!
Nur "**demo.ppt**" ist nicht fragmentiert!

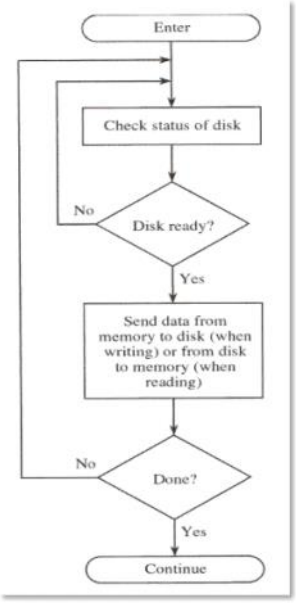


Bei der internen Fragmentierung ergibt sich also Speicherverlust (aus Sicht bytes ja, aus Sicht Cluster nein -> EOF).

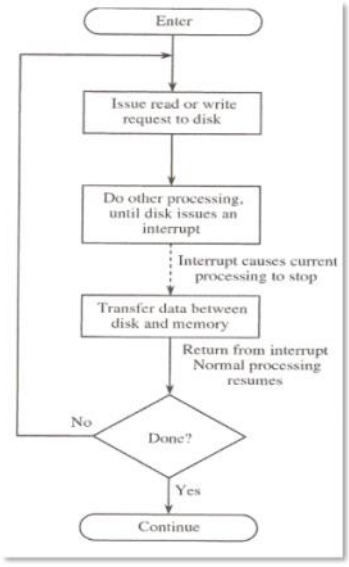
Polling und Interrupting

18 November 2014 09:09

Methoden wie 2 Partner kommunizieren können.



Polling (Busiy Waiting, Round Robin)



Interrupt Requesting

SSD

25 November 2014 08:39

When deciding what type of flash to use for an application, it is important to understand the differences between flash technologies. The following document explains the pros and cons of the three types of flash, SLC, MLC and TLC.

SLC- Single Layer Cell

- High performance
- Lower power consumption
- Faster write speeds
- 100,000 program/erase cycles per cell
- Higher cost
- **A good fit for industrial grade devices**, embedded systems, critical applications.

MLC- Multi Layer Cell

- Lower endurance limit than SLC
- 10,000 program/erase cycles per cell
- Lower cost
- **A good fit for consumer products.** Not suggested for applications which require frequent update of data.

TLC- Three Layer Cell

- Higher density
- Lower endurance limit than MLC and SLC
- TLC has slower read and write speeds than conventional MLC
- 5,000 program/erase cycles per cell
- Best price point
- **A good fit for low-end basic products.** Not suggested for critical or important applications at this time which require frequent updating of data.

SLC vs. MLC vs. TLC as explained with a glass of water

This glass of water analogy demonstrates how SLC NAND Flash outperforms MLC NAND Flash.

- SLC Flash has only two states: erased (empty) or programmed (full).
- MLC Flash has four states: erased (empty), 1/3, 2/3, and programmed (full).
- TLC Flash has eight states: erased (empty), 1/7, 2/7, 3/7, 4/7, 5/7, 6/7 and programmed (full).

It's easier to read the correct fill status when a glass is either empty or full, as in SLC NAND Flash. When a glass is partially full, as in MLC NAND Flash, the fill status is more difficult to read, taking more time and energy.

Source: http://centon.com/flash-products/chiptype

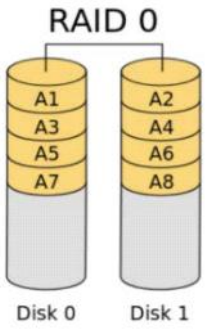
RAID

18 November 2014 10:29

Heisst: Redundant Array of Independent Disks
Es gibt verschiedene RAID-Levels

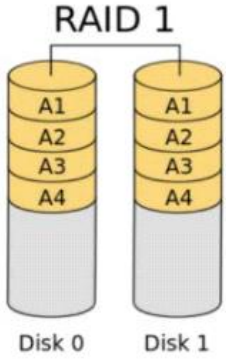
RAID 0 - Striping

- Dateien werden auf verschiedene physische Disk aufgeteilt.
- Keine Redundanz bei Disk-Verlust
- Gute Performance



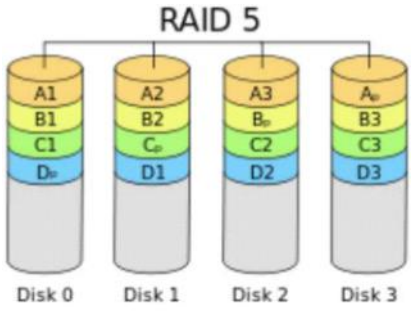
RAID 1 - Mirroring

- Disk werden vollständig dupliziert
- Braucht offensichtlich doppelten Speicherplatz
- Hohe Redundanz



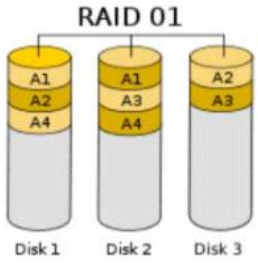
RAID 5 - Striping with Parity

- Vereint beide Ansprüche von 0 und 1
- Mithilfe des Parity-bits kann eine Disk wiederhergestellt werden
- Effiziente Disk benutzung
    - Dateien werden nicht dupliziert, zusätzliche Benutzung durch Parity bit
- Parity berechnung kann unperformant sein

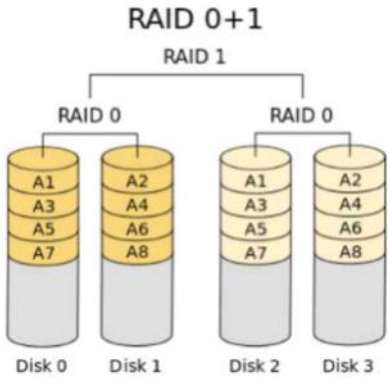


RAID 01

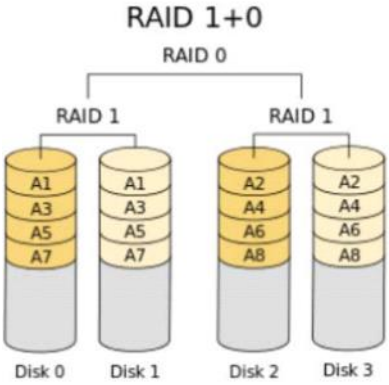
Wird selten bneutzt.



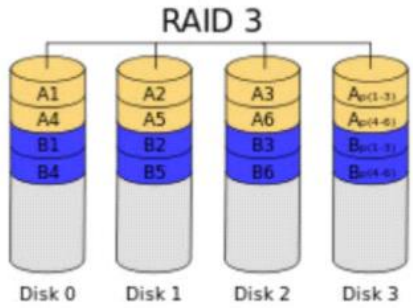
RAID 0+1



RAID 1+0

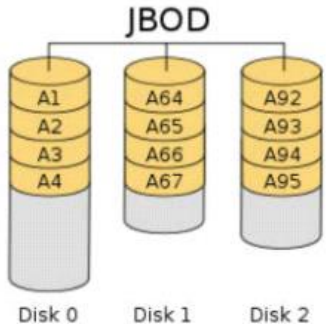


RAID 3



JBOD

Just a bound of disk



RAID Implementation

Software-RAID

- Eine OS-Feature
- Setzt keine spezielle Hardware voraus
- Schlechtere performance als hardware-Raid

Hardware-RAID

- Eine Drive-Controller feature
- Konfiguration unabhängig vom OS -> unsichtbar für OS
- Hohe performance -> designed for speed

Wechsel

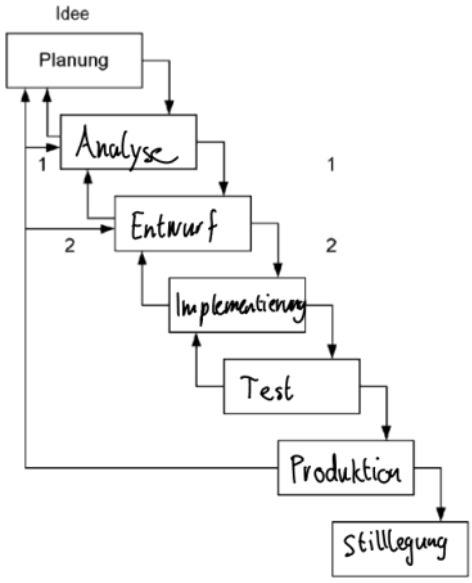
Hot Replacing, Swapping, Plugging -> Wechsel einer Disk im laufenden Betrieb
Host Spare (Sparing) -> Laufendes nicht verwendetes Laufwerk, einsatz bei Ausfall

Prozessmodelle

25 November 2014 10:03

Schwergewichtige (klassische) Modelle

- Wasserfall-Modell



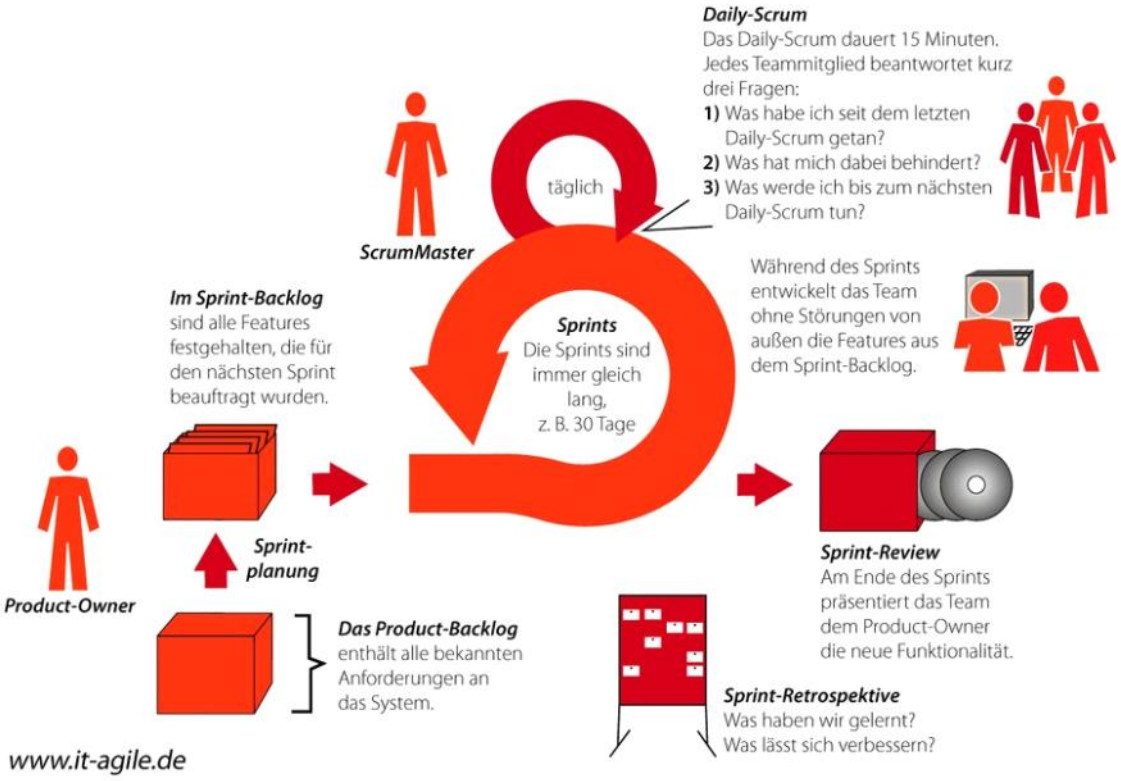
- V-Modell (ähnlich Wassefall jedoch kongruent v-seitig Testphasen zu jedem Step)
- Inkrmentelle und iterative Prozessmodelle
    - Spiralmodell (böhm)
    - Rational Unified Proces
    - Open Unified Process

Leitgewichtige (agile) Vorgehensmodelle

- Extreme Programming (XP)
    - Programmierung zu zweit während eines Zeitraums -> Fazit: zu streng

Scrum

Im Mittelpunkt von Scrum steht das selbstorganisierte Entwicklerteam, das ohne Projektleiter auskommt. Um dem Team eine störungsfreie Arbeit zu ermöglichen, gibt es den ScrumMaster, der als Methodenfachmann dafür sorgt, dass der Entwicklungsprozess nicht zerbricht.
Der ScrumMaster stellt auch die Schnittstelle zum Produktverantwortlichen (Product-Owner) dar, dem die Aufgabe zukommt, Anforderungen zu definieren, zu priorisieren und auch zu tauschen. Allerdings ist in Scrum klar geregelt, wann der Produktverantwortliche neue oder geänderte Anforderungen beauftragen darf - so gibt es ungestörte Entwicklungszyklen von 2-4 Wochen (Sprints), in denen ihm untersagt ist, das Entwicklerteam zu "stören". Während eines Sprints wird deshalb der Product Owner seine Vorstellungen von der weiteren Entwicklung ins Product Backlog eintragen und so für kommende Sprints einplanen.

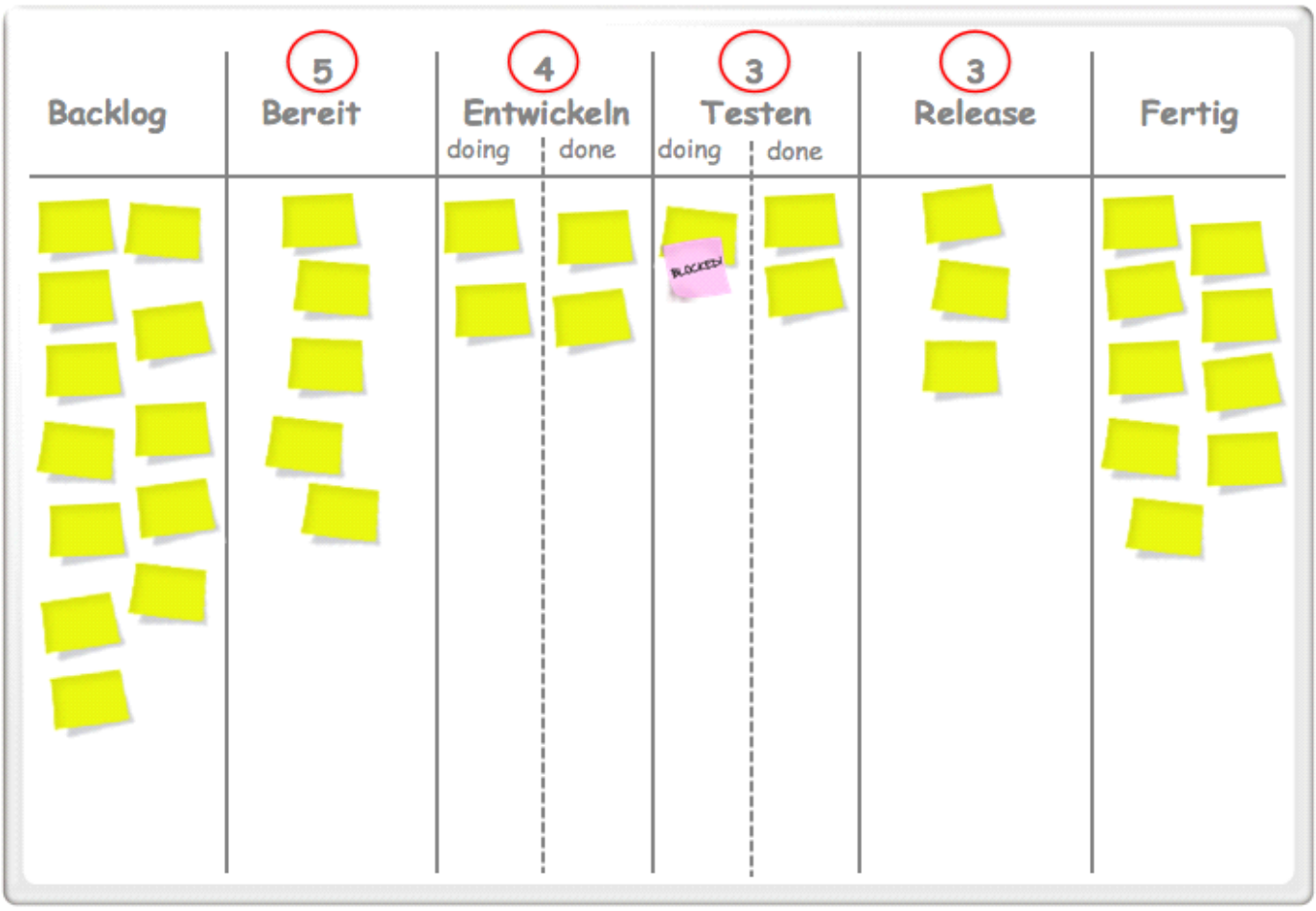


Kanban

- Kanban ist eine agile Methode für evolutionäres Change Management. Das bedeutet, dass der bestehende Prozess in kleinen Schritte (evolutionär) verbessert wird.
- Indem viele kleine Änderungen durchgeführt werden (anstatt einer großen), wird das Risiko für jede einzelne Maßnahme reduziert.
- Darüber hinaus führt der eher sanfte Stil von Kanban in der Regel zu weniger Widerständen bei den Beteiligten.

So funktioniert Kanban:

Der erste Schritt bei der Einführung von Kanban besteht darin, den bestehenden Workflow, die vorhandene Arbeit sowie Probleme zu visualisieren.
Dies wird in Form eines Kanban-Boards getan, das z.B. aus einem einfachen Whiteboard und Haftnotizen oder Karteikarten besteht. Jede Karte auf dem Board repräsentiert dabei eine Aufgabe.



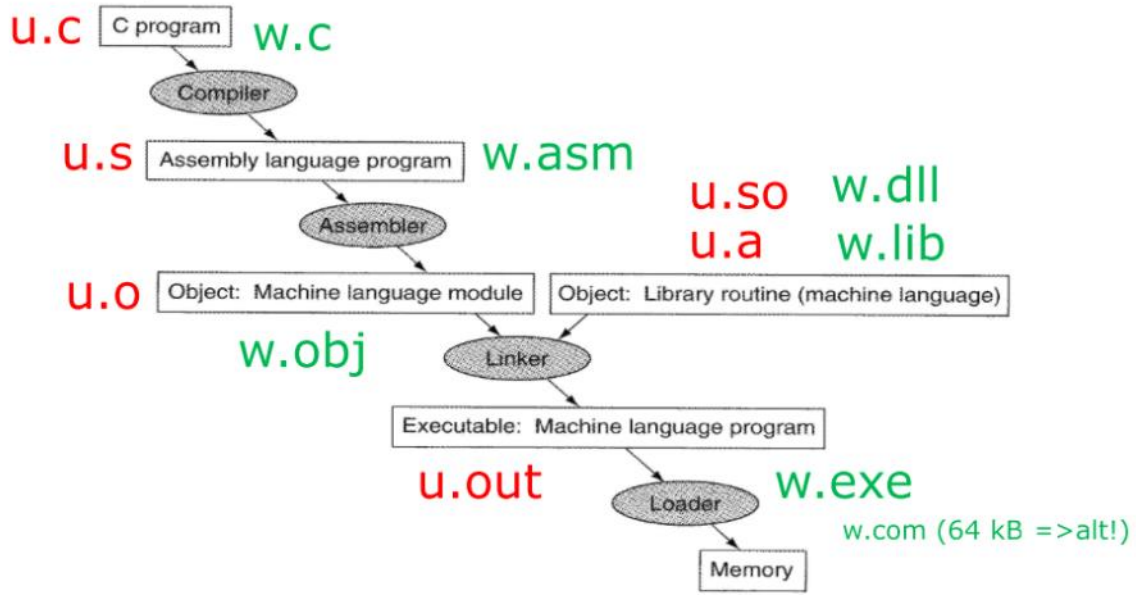
Allein diese einfache Maßnahme führt zu viel Transparenz über die Verteilung der Arbeit sowie bestehende Engpässe.

- Kanban ist ein übergreifender Ansatz, der prinzipiell auf jeden bestehenden Prozess aufgesetzt werden kann - egal ob agil oder nicht.
- Viele Prinzipien und Techniken aus Scrum ergänzen sich gut mit Kanban (etwa die Daily Scrums und die Burndown-Charts)
- und der ScrumMaster kann seine Rolle nutzen, um Kanban einzuführen.

Compilieren und Interpretieren

02 December 2014 08:33

Entiwcklung mit C:



Die Kompilierung erfolgt in 3 Schritten:

Lexikalische Analye (Scanner, Lexer)

- Strom aus Zeichen wird squentiell gelsen und in Symbole (Tokens) separiert
- Tokens werden mit Position in Quelltext assozieert.
- Lexikalischer Fehler: Zeichen oder Zeichenfolge, die keinem token zugeordnet werde kann, z.B. Bezeichner die mit Zahlen beginne "3foo".

Syntaktische Analyse (Parser)

- Symbole werden zu grammatikalischen Einheiten zusammengefasst
- Zusammenfassung erfolgt nach Syntaxregeln
- Visualisierung durch Syntaxbaum
- Syntaxfehler: Fehlende Klammer steht zu Beginn einer Methode oder es fehlt eine ";" am Ende einer Anweisung.

Semantische Analyse

- Erzeugung eines mit weiteren Attributen versehenen Sytanxbaum, d.h. attributierten Syntaxbaum
- Untersuchung von semantischen Fehler
  - Verwendete Variable muss deklariert sein
  - Datentypen müssen bei Zuweisung verträglich sein
  - Verwendung von nicht vorher definierten Bezeichner

Backend eines Compilers

- Auswertung des attributierten Syntaxbaumes
- Optimierung am Code (Syntaxbaum)
- Transformation des Syntaxbaumes in die Enddarstellung -> Maschinencode eines OS

Programmiersprachen

02 December 2014 08:48

Die Unterteilung der Generationen von Programmiersprachen wird in Paradigment unterteilt, d.h. eine grundsätziche Denkweise.

Name	funktional	imperativ	objektorientiert	deklarativ	logisch	nebenläufig
Ada		X	X			X
C		X				
Prolog				X	X	
Scheme	X	X	(X)	X		(X)
Haskell	X	(X)		X		(X)

Dabei wird nebst OOP auch zwischen diesen Paradigmen unterschieden:

Prozedurale Programmierung

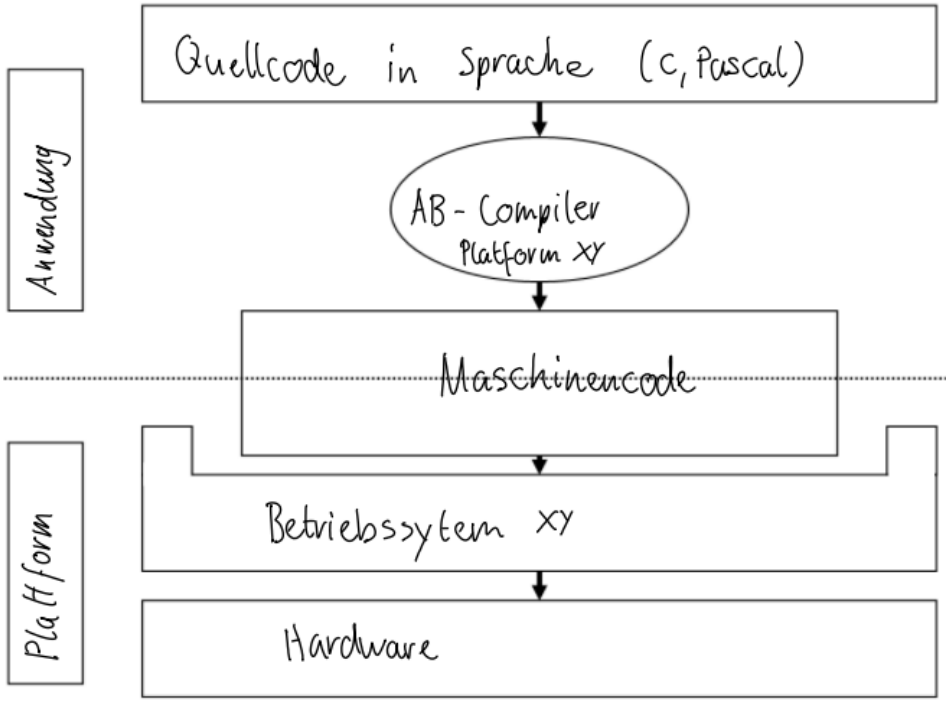
- Wird am nächsten mit Programmierung in Verbindung gesetzt
- Quellcode wird von oben nach unten gelsen -> Liste von Instruktionen
- Sprachen: C, C++ und Java -> nicht nur OOP sondern auch prozedural
    - Sind immer auch imperative Programmiersprachen
    - -> Befehl pro Linie an Compiler
    - Anweisung kann man als Zustand des Programmes werten, z.B. Variable wird inkrementiert
- Imperative Programmierung beschreibt wie etwas berechnet werden soll
- Maschinencod

Funktionale Programmierung

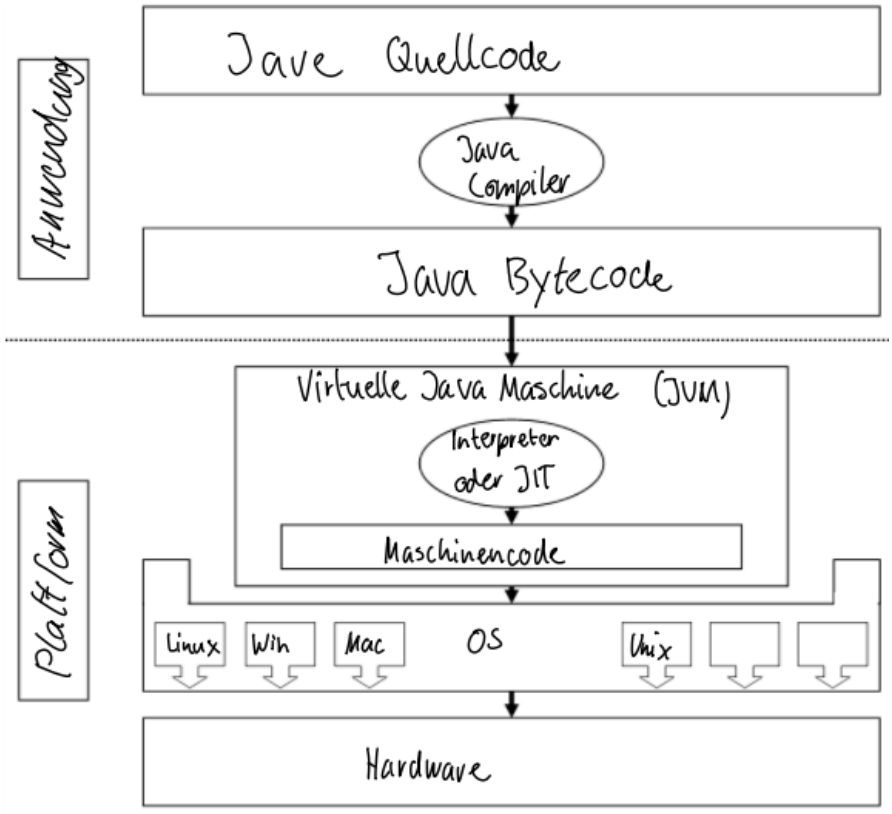
- Funktion im mathematischen Sine
- Zustandlos -> Eingabeparameter verändern Zustand des Programmes nicht
- Funktion kann ihren eigenen Zustand verändern -> concurrency, mehrere Instanzen laufen unabhängig
- Funktionen können andere Funktionen aufrufen (Rekursion)
- -> Entwicklung von Funktionen
- Funktionale Programmiersprachen sind immer deklarative Programmiersprachen
- -> Deklarative beschreibt was berechnet werden Soll
    - SQL z.B. beschreibt das Resulatat, der Lösungsweg mach die Programmiersprache
- Beispiele: Haskell, LISP oder Erlang -> beliebt in akademischen Krisen als Mittel der Mathematik

Enwicklungsparadigmen

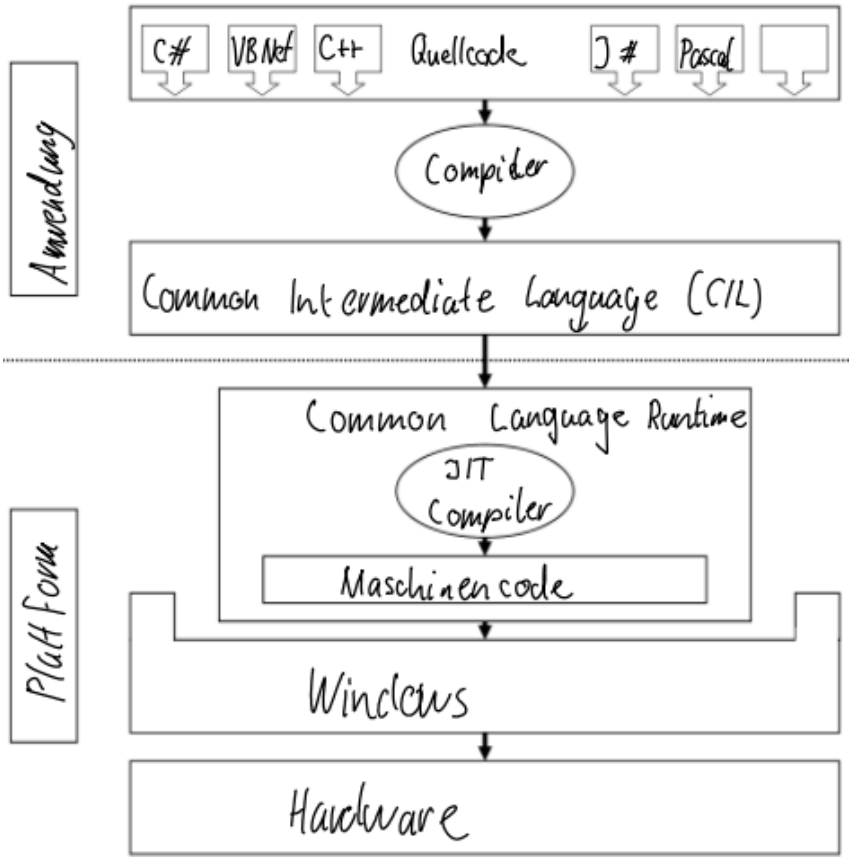
Klassisch:



JEE (Java Enterprise)



.Net (Microsoft)


Anwendung
C#
VB.Net
C++
Quellcode
J#
Pascal
Compiler
Common Intermediate Language (CIL)
Plattform
Common Language Runtime
JIT Compiler
Maschinencode
Windows
Hardware

Algorithmus II

02 December 2014 09:11

Bezeichnet eine Verarbeitungsvorschrift.

Ein Algorithmus ist eine in der Beschreibung und Ausführung endliche, deterministische und effektive Vorschrift zur Lösung eines Problems, die effizient sein sollte.

endlich: nach einer endlichen Zeit wird der Algorithmus beendet,

deterministisch: nur definierte und reproduzierbare Zustände treten auf d.h. bei gleicher Eingabe folgt immer gleiche Ausgabe und zusätzlich wird die gleiche Folge von Zuständen durchlaufen. Zu jedem Zeitpunkt ist der nachfolgende Abarbeitungsschritt des Algorithmus eindeutig festgelegt. Es gibt auch "nicht-deterministische" Algorithmen!

effektiv: Grad (Mass) für die Zielerreichung: Es gibt Aufschluss darüber, wie nahe ein erzieltes Ergebnis dem angestrebten Ergebnis gekommen ist. Wir erwarten bem Programmieren implizit meist 100% ige Zielerreichung!

effizient: Mass für die Wirtschaftleichkeit z.B. geringer Verbrauch an Ressourcen wie Speicherplatz und Rechenzeit.

Abstrahierung: Ein Algorithmus löst eine ganze Klasse von gleichartigen Problemen. Die Wahl des speziellen Problems erfolgt über Parameter.

Finitheit statisch: Die Beschreibung des Algorithmus selbst ist endlich. - Finitheit dynamisch: Ein in Bearbeitung befindlicher Algorithmus hat zu jedem Zeitpunkt eine endliche Fülle von Datenstrukturen und Zwischenergebnissen. Er belegt deshalb endlich viele Ressourcen im System.

Sequenzialität: Ein Algorithmus ist aus Einzelschritten aufgebaut. In jedem dieser Schritte wird eine einfache Operation ausgeführt, wie z.B. eine Addition oder eine Zuweisung zu einer Variablen.

Realisierbarkeit: Die genannten Operationen müssen tatsächlich in der Praxis durchführbar sein. Die Quadratur des Kreises oder die Division durch Null sind also nicht algorithmisch lösbar, ebenso wenig wie die Bestimmung der Masse der Erde auf ein Gramm genau.

Terminierung: Normalerweise gehen wir davon aus, dass ein Algorithmus terminiert, das heisst, nach einer absehbaren Zeit kontrolliert abbricht. Gewisse Algorithmen – und Programme – laufen potenziell endlos wie z.B. Betriebssysteme, Prozessleitsysteme usw.

Determinismus: Ein Algorithmus ist dann deterministisch, wenn zu jedem Zeitpunkt nur eine Möglichkeit des weiteren Ablaufs, oder des Abbruchs, besteht. Ist ein nichtdeterministischer Algorithmus durch Wahrscheinlichkeiten oder Zufälle gesteuert, dann heisst er stochastisch.

Determiniertheit: Ein Algorithmus ist dann determiniert, wenn er bei gleichen Startparametern und Eingabewerten auf gleiche Art terminiert und gleiche Ergebnisse liefert (dabei aber möglicherweise nicht immer die gleiche Sequenz von Einzelschritten wählt).

Wichtig:

Rekurive Programmierung

Beispiel für rekursive Programmierung in Java:

```
public class Calculator {
     public static void main(String[] args) {
          System.out.println(getFactorial(32));
     }
     public static double getFactorial(double n){
          if(n==1){
               return 1;
          }
          return n * getFactorial(n-1);
     }

     public static int getFibonacci(int z){
               if(z==0){
                    return 0;
               }
               if(z==1){
                    return 1;
               }
               return getFibonacci(z-1) + getFibonacci(z-2);
          }
     }
}
```

Der base case ist das IF-Statement, die Rekursion natürlich der wiederholte Funktionsaufruf.

Zeichenkette Invertierung

Einfache Invertierung einer Zeichenkette

```
using System;

class InvertMain
{
   public static string Invert(string old)
   {
       if(old.Length < 2) return old;
       return old.Substring(old.Length-1) + Invert(old.Remove(old.Length-1, 1));
   }

   public static void Main()
   {
       string old = Console.ReadLine();
       Console.WriteLine(Invert(old));
   }
}
```

Prosit

Ermittelt wie oft die Gläser klingen, wenn n Personen, jede mit jedem anstossen.

```
class PrositMain
{
    private static long Prosit(long n)
    {
        if (n > 2) return n-1 + Prosit(n-1);
        else return 1;
    }

    public static void Main()
    {
        Console.Write("Wie viele Gäste sind an der Party: ");
        int gaeste = Int32.Parse(Console.ReadLine());
        Console.WriteLine("Die Gläser klingen {0} mal.", Prosit(gaeste));
        Console.ReadKey();
    }
}
```

Ressourcenkomplexität

09 December 2014 08:24

Die Komplexitätstheorie als Teilgebiet der Theoretischen Informatik befasst sich mit der Komplexität von algorithmisch behandelbaren Problemen auf verschiedenen mathematisch definierten formalen Rechnermodellen. Die Komplexität von Algorithmen wird in deren Ressourcenverbrauch gemessen, meist Rechenzeit oder Speicherplatzbedarf.

Beispiel - Zahlenreihe:

Nun möchte man herausfinden, welcher zusammenhängende Abschnitt von Zahlen, die grösste Summe aufweist.

Kubischer Algorithmus

```
int maxfolge1(int z[], int n) {
    int i , j , k, sum, max = −10000000;
        for ( i = 0; i < n; i++)
            for ( j = i ; j < n; j++) {
                sum = 0;
                for (k = i ; k <= j; k++)
                    sum += z[k];
                if (sum > max)
                    max = sum;
            }
        return max;
}
```

- Zeiaufwand: 3 geschachtelte for-Schlaufen
- Aufwand: etwa proportional n^3

Quadratischer Algorithmus

```
int maxfolge2(int z[], int n) {
    int i , j , sum, max = −10000000;
    for ( i=0; i<n; i++) {
        sum = 0;
        for ( j=i; j<n; j++) {
            sum += z[j];
            if (sum > max)
                max = sum;
        }
    }
    return max;
}
```

- Zeiaufwand: 2 geschachtelte for-Schlaufen
- Aufwand: etwa proportional n^2

Linearer Alorithmus

```
int maxfolge3(int z[], int n) {
    int i , s, gesamtmax = −10000000, endesumme = 0;
    for ( i=0; i<n; i++) {
        if ((z[i] < 0 && Math.abs(z[i]) <  gr_min_zahl)){
          gr_min_zahl = Math.abs(z[i]);}
        endesumme = ((s=endesumme+z[i]) > 0) ? s : 0;
        if (endesumme > gesamtmax)
            gesamtmax = endesumme;
    }
    return gesamtmax;
}
```

- Zeiaufwand:1 for-Schlaufen
- Aufwand: etwa proportional n

Exponentieller Algorithmus

```
int prim(int zahl, int teiler ) {
    if (zahl < 2 || zahl%2 == 0 || zahl%teiler == 0)
        return 0; /* keine Primzahl */
    else if ( teiler *teiler > zahl)
        return 1; /* Primzahl */
    return prim(zahl, teiler+1);
}
```

- Ist am schnellsten.
- Zeiaufwand: 0 Schleifen (Schaltkreis entscheiden)
- Aufwand: 2^n

Die Wahl des Algortithmus ist gravierend für das Zeitverhalten des Programms.
Zusammenfassend für 10'000 Zahlen kann man folgende Aussage machen:

`maxfolge1()`: $t(n) = 10000^3 = 10^{4^3} = 10^{12} = 1$ Billion

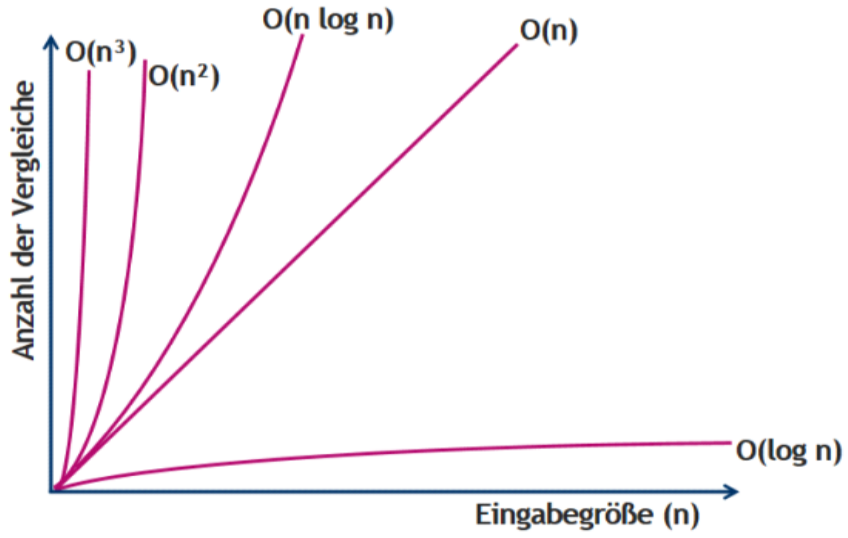
`maxfolge2()`: $t(n) = 10000^2 = 10^{4^2} = 10^8 = 100$ Millionen
also 10000 mal schneller als `maxfolge1()`

`maxfolge3()`: $t(n) = 10000^1 = 10^4$
also 10000 mal schneller als `maxfolge2()` und
100 Millionen mal schneller als `maxfolge1()`.

Die verschiedenen Algorithmen lassen sich also in mathematische Funktionen unterteilen:

1	*konstant*	Jede Anweisung eines Programms wird höchstens einmal ausgeführt. Dies ist der Idealzustand für einen Algorithmus.
$\log n$	*logarithmisch*	Speicher- oder Zeitverbrauch wachsen nur mit der Problemgröße *n*. Die Basis des Logarithmus wird häufig 2 sein, d. h. vierfache Datenmenge verursacht doppelten Ressourcenverbrauch, 8-fache Datenmenge verursacht 3-fachen Verbrauch und 1024-fache Datenmenge 10-fachen Verbrauch.
n	*linear*	Speicher- oder Zeitverbrauch wachsen direkt proportional mit der Problemgröße *n*.
$n \log n$	$n \log n$	Der Ressourcenverbrauch liegt zwischen n (*linear*) und n^2 (*quadratisch*).
n^2	*quadratisch*	Speicher- oder Zeitverbrauch wachsen quadratisch mit der Problemgröße. Solche Algorithmen lassen sich praktisch nur für kleine Probleme anwenden.
n^3	*kubisch*	Speicher- oder Zeitverbrauch wachsen kubisch mit der Problemgröße. Solche Algorithmen lassen sich in der Praxis nur für sehr kleine Problemgrößen anwenden.
2^n	*exponentiell*	Bei doppelter, dreifacher und 10-facher Datenmenge steigt der Ressourcenverbrauch auf das 4-, 8- bzw. 1024-fache. Solche Algorithmen sind praktisch kaum verwendbar.

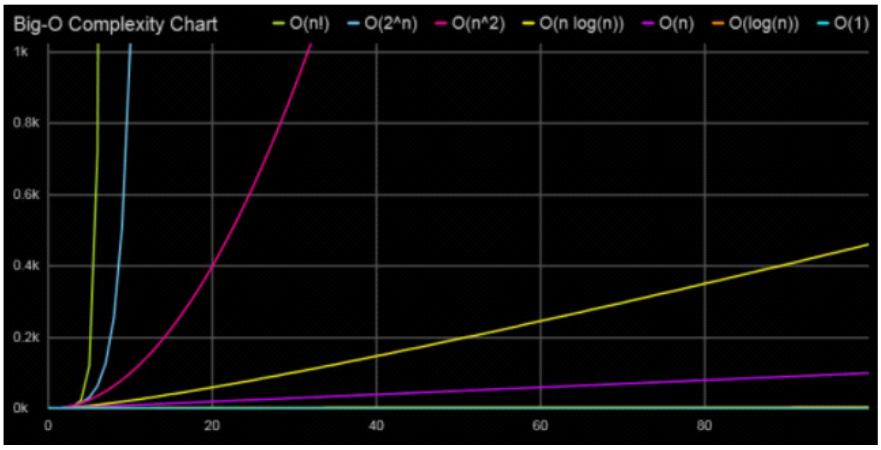
Eine Visualisierung der Anzahl der Vergeliche, die eine solcher Algorithmus macht lässt sich einfach aufzeigen:



Sortierung

09 December 2014 08:52

Algorithm	Data Structure	Time Complexity			Worst Case Auxiliary Space Complexity
		Best	Average	Worst	Worst
Quicksort	Array	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Mergesort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Heapsort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Select Sort	Array	O(n^2)	O(n^2)	O(n^2)	O(1)
Bucket Sort	Array	O(n+k)	O(n+k)	O(n^2)	O(nk)
Radix Sort	Array	O(nk)	O(nk)	O(nk)	O(n+k)



Bubble sort

```
BubbleSort(Array A)
for (n=A.size; n>1; n=n-1){
   for (i=0; i<n-1; i=i+1){
      if (A[i] > A[i+1]){
          A.swap(i, i+1)
      } // ende if
   } // ende innere for-Schleife
} // ende äußere for-Schleife
```

1. Beginnend beim ersten wird Element mit Nachbarverglichen
2. Grösse Zahl landet in der letzten Position
3. Erneuter Vergleich minus der Position bis alle Elemente durchiteriert worden sind

Komplexität: n^2

Ripple sort

```
void RippleSort(int n, int z []) {
   for (int i=0; i<n-1; i++){
      for (int j=i+1; j<n; j++){
         if (z[i ] > z[j ]) {
            int t=z[i]; z[i]=z[j];
            z[j]=t;
         }
      }
   }
}
```

Ähnlich dem Bubble-Sort

1. Jedes Element wird mit dem letzten Element vergleichen und vertauscht
2. Nach jedem Durchgang liegt am Ende das grösste Element
3. Für jeweils eine Position weniger wird dieser Vorgang rekursiv durchgeführt

Komplexität: n^2

Insertion sort

```
void InsertionSort(vektor){
   for (int i = 1; i < vektor.Length;
                                    i++) {
      int temp = vektor[i];
      for (int j = i-1; j >= 0; j--) {
         if (vektor[j] > temp) {
            vektor[j+1] = vektor[j];
            vektor[j] = temp;
         }
      }
   }
}
```

Vergleichbar mit Jassen beim Aufnehmen der Karten

1. Schleife: Werte iterieren
2. Beginnend beim zweiten Elemente wird dieser mit allen vorgängigen Position verglichen
3. Der grössere Wert wird jeweils rechts eingeschoben
3. Dieser Vorgang wird rekursiv durchgeführt

Komplexität: n^2

Selection ort

```
SelectionSort(vektor){
   for (int i=vektor.size-1; i>0;i--){
      int max = 0;
      int temp =0;
         for (int j=1; j<=i;j++){
            if (vektor[j]>vektor[max]){
               max=j;  // merke
            }          // grösste Zahl
         }
      temp = vektor[i];
      vektor[i] = vektor[max];
      vektor[max] = temp;
  }
}
```

Ähnlich Ripple, nur wird erst kontrolliert um am Schluss getauscht.

1. Für jede Position wird geschaut welcher Werte der kleinste oder der grösste der vorausgehenden Position ist.
2. Das kleinste wird an die aktuelle Stelle platziert.
3. Das wird solange wiederholt bis man an der letzten Position ist.

Beim Ripple sort kann es zu mehreren Tausch kommen bevor der Maximal Wert an der entsprechenden Position liegt.

Shell sort

```
ShellSort(Array A)
static void  shellsort (int[] a, int n)
{
   int i, j, k, h, t;

   int[] spalten = {1743392200, 581130733, 193710244,
64570081, 21523360, 7174453, 2391484, 797161, 265720,
 88573, 29524, 9841, 3280, 1093, 364, 121, 40, 13, 4, 1};

   for (k = 0; k < spalten.length; k++)
   {
      h = spalten[k];
      // Sortiere die "Spalten" mit Insertionsort
      for (i = h; i < n; i++)
      {
         t = a[i];
         j = i;
         while (j >= h && a[j-h] > t)
         {
            a[j] = a[j-h];
            j = j - h;
         }
         a[j] = t;
      }
   }
}
```

...hmm:
3n + 1
eine Möglichkeit!

Bedient sich Prinzip vom Insertion sort.
Jedoch möchte man vermeiden, dass Elemente über weite Strecken verschoben werden müssen.
Deshalb wird die Sequenz in Untersequenzen zerlegt und einzeln sortiert.

1. Aufteilung in z.B. in 4er Blöcke
2. Soritierung blockweise -> 4-sortiert
3. Wiederholen für 2 und 1 Blöcke
4. Letzter als 1er Block entspricht dan wieder Insertion sort.

Quick sort

Ganze Familie von sorts, es gibt etliche Versionen.
Ist ein Rekusiver Alogrithmus

```
int partition(int z [], int l , int r) {
  int x = z[r], i = l-1, j = r ;
  while (1) {
    while (z[++i] < x)
      ;
    while (z[--j] > x)
      ;
    if ( i < j)
      swap(&z[i], &z[j ]);
    else {
      swap(&z[i], &z[r ]);
      return i ;
    }
  }
}
void quick sort(int z[], int l , int r) {
  if ( l < r) {
    int pivot = partition(z, l , r );
    quick sort(z, l, pivot-1);
    quick sort(z, pivot+1, r);
  }
}
```

1. Teilen der Liste in zwei Teillisten (links und rechts)
2. Vergleich der Werte anhand Pivotelement (das letzte Element im Array)
3. Linke Liste enthält kleinere Elemente und rechte die Grösseren
4. Diesen Vorgang (1-3) nun für jede Teilliste rekursiv durchführen
5. Ist Grösse der Liste eins oder 0 erfolgt Abbruch der Rekursion

Als Pivot Element empfiehlt sich

Merge sort

```
void merge(int z[], int l , int m, int r) {
  int i , j , k;
  for ( i=m+1; i>l; i--)
    hilf [ i-1] = z[i-1];
  for ( j=m; j<r; j++)
    hilf [r+m-j] = z[j+1];
  for (k=l; k<=r; k++)
    z[k] = ( hilf [ i ] < hilf[ j ]) ?
             hilf [ i++] : hilf [ j--];
}
void merge sort(int z[], int l , int r) {
  if ( l < r) {
    int mitte = ( l+r)/2;
    merge sort(z, l, mitte);
    merge sort(z, mitte+1, r);
    merge(z, l, mitte, r );
  }
}
```

1. Die gesamte Menge wir als erstes in kleinere Listen zerlegt.
2. Diese Teillisten werden sortiert
3. Die Teillisten werden dann nach dem Reisverschlussprinzip zusammengefügt.
4. Jede neue Kombination von Listen wird dan ebenfalls nach dem Reisverschlussprinzip zusammengefügt.

Datenstrukturen

09 December 2014 10:29

Datenstrukturen können sein:

Arrays

```
int[] array = {4, 7, 2, 1}
```

- Adressierbarkeit über Index
- 1 bis n dimensional
- Komponenten können primitive Datentypen oder ganze Objekte sein

Listen
Passendes Beispiel wäre Linked List

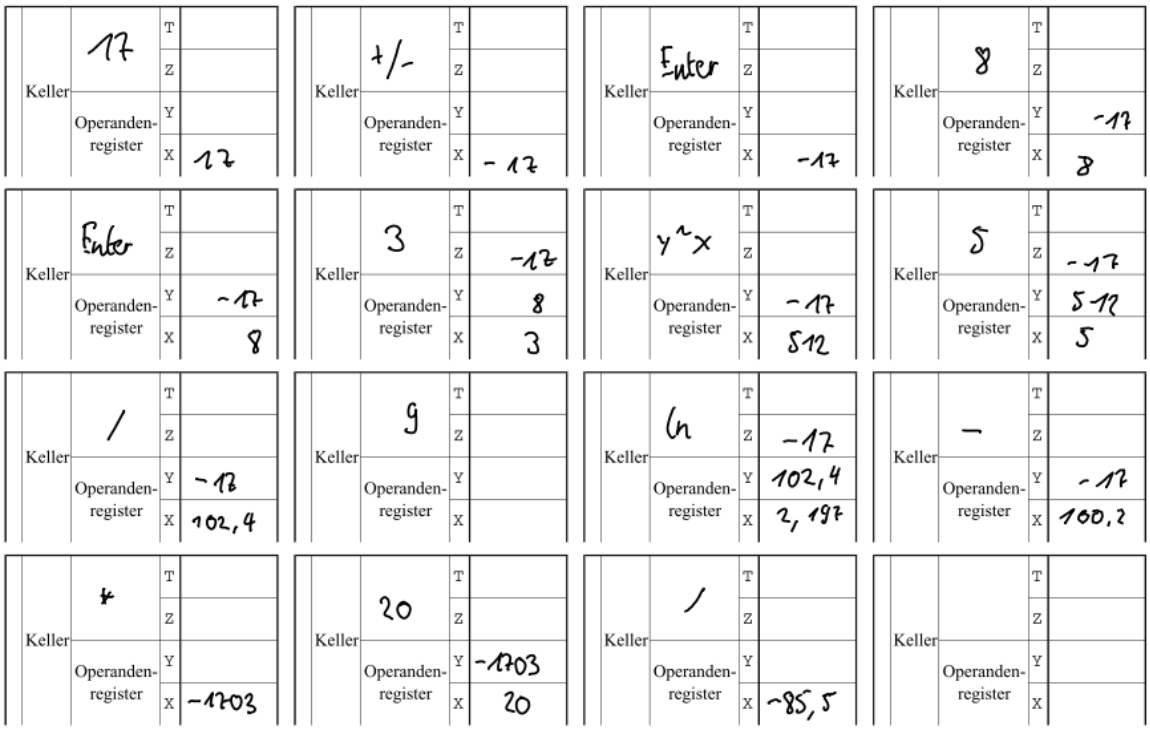
- homogen, gleiche Strukturen verketten
- dynmische, zu Laufzeit veränderbar

Stacks
Für LIFO-Bearbeitung (Last In First Out) der Komponenten
Ist ebenfalls homogen, dynamisch und abstrakt.

Beispiel - UPN Rechner:

$$\frac{-17\cdot\left(\frac{8^3}{5}-\ln 9\right)}{20}$$

Vor Tausend Jahren kannten Taschenrechner noch keine Klammern, folge jeder Hersteller hatte eine eigene Notation. Dazu eine Beispiel von HP:

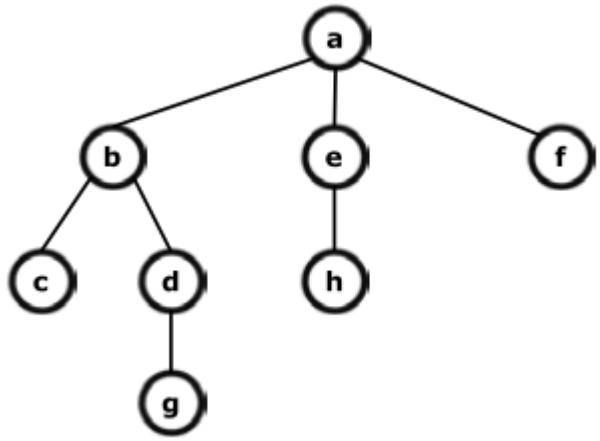


Schlangen und Ringpuffer
für die FIFO-Bearbeitung der Komponenenten

Bäume
Schnelle Suche nach Komponenten

Trees

09 December 2014 10:30



Bäume sind im Vergleich zu anderen Datenstrukturen nichtlinear.
In Bäumen kann man

- Nicht nur Daten, sondern auch
- relative Beziehungen der Daten untereinander,
- Ordnuungsgeziehrungen (siehe geordnete Bäume)
- hierarchische Beziehungen
- und Gewichte auf den Kanten (Zeile) speichern

Vorteil von Baumstrukturen bietet die Durchsuchung, ebenfalls könne viele Operationen mit niedriger O-Komplexität durchgeführt werden (Suchen, Insert, Delete, …)

Rekursive Definition

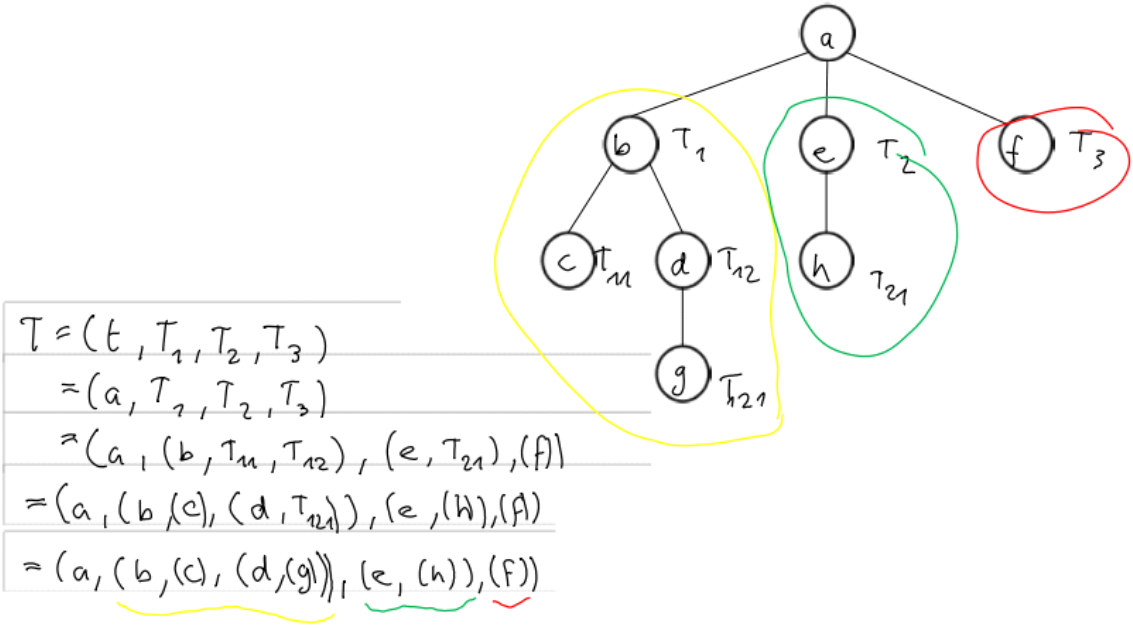
Ein endlicher Baum ist eine endliche Menge T, die in n+1 (n ∈ N) paarweise disjunkte Teilmengen T0, ..., Tn zerlegt ist mit:
1. |T0| = 1
2. Ti ist wieder ein Baum (1<= i <= n)
Alle Teilbäume haben immer +1 Child.

Eine mathematische Notierung sieht dann so aus:

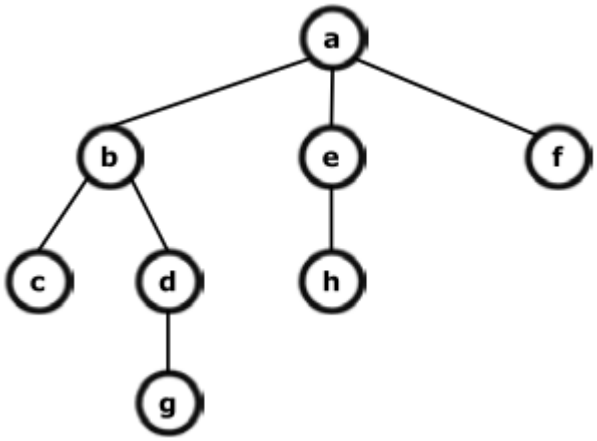
$$T = (t, T_1, ..., T_n) \; mit : T = \{t\} \cup \bigcup_{i=1}^{n} T_i$$

t: Wurzel (nur Name)
Ti: Teilbäume von T
n: z.B. n = 0 dann Besteht Baum aus nur einer Wurzel und hat keine Teilbäume.
u: Vereint
U: Summe

Eine Formale Definition kann dann so aussehen:



Elemente



Knoten (nodes): a-f bzw. N1-Ni
Kanten: Verbindungen (a->f)
a ist der Elternknoten des Baumes
b ist ein direkter Kindknoten
Ein Knoten ohne Kindknoten ist Blatt (leaf)
externe Knoten (äussere Knoten) und innere Knoten = Alle Knoten

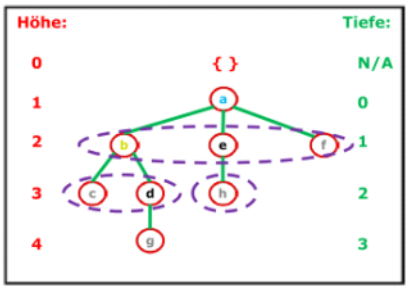
- externe Knoten (c, g, h, f)
- innere Knoten (b, d, e, a)

Knoten ohne Elterknoten ist die Wurzel (a)
Jeder Kindknoten ist Wurzel eines Teilbaums (ti) (b,c, d, g, e, h, f)
Eine Pfad (path) ist der Weg von einem Knoten zu einem anderen {h, e, a, b, c}
Die Länge l ist die Mächtigkeit der Menge der knoten in einem Pfad (c zu h -> l = 5)
Betrachtet man nur absteigene Pfade so gilt:

- a nach b -> b ist Nachkomme (kind oder child) von a (successor)
- a ist Vorfahre (Eltern, parent, predecessor)
- d ist Vorfahre von g, f ist eine Nachomme von a
- a ist ein Vorfahr von c, aber ekein direkter Vorfahre.

Höhe und Tiefe

Zwischen Höhe und Teife muss ganz klar unterschieden werden, je nach Definition unterscheiden sich diese Werte.



Tiefe ist Attribut des Knotens

Höhe ist Attribut des Baumes

- Die direkten Nachkommen eines Elternknotens (="Geschwister") bilden ein **Niveau** oder eine **Ebene** (*layer*).
- Die **Tiefe** eines Kontens ist die Anzahl Bögen (**Kanten**) bis zur Wurzel oder auch die Länge des Pfades von der Wurzel zu ihm.
- Ein Baum hat immer ein **Höhe** h:
  - Ein leerer Baum **{ }** hat die Höhe 0
  - ein Baum aus "nur" einem Wurzelknoten hat die Höhe 1
  - => unser Baum oben hat die **Höhe** 4
  - => unser Baum oben hat die **Tiefe** 3
- Es gilt die Gleichung: Höhe = Tiefe +1

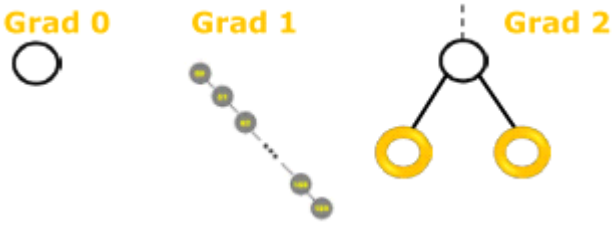
Hinweis: Im zugehörigen Lernmittel gilt Tiefe=Höhe.

Eigenschaften

Geordnete Bäume
Bei geordneten Bäumen ist klar die Reihenfolge von direkten Nachfolger bei jedem Knoten festgelegt.

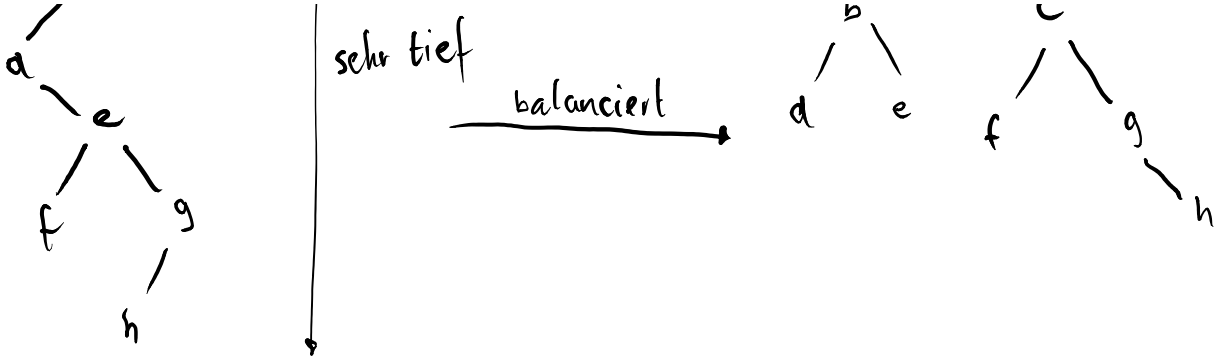
Grad
Auch Knotengrad oder Valen ist ein Begriff aus der Graphentheorie.
Beschreibt die Anzahl ausgehenden Kanten von einem Knoten.
Entspricht der maximal zulässigen Anzahl direkter Kindknoten.

- Grad 2 wäre ein Binärbaum
- > Grad 2 sind Vielwegbäume
- Grad 1 ist eine Liste (Linked List, Arrays)
- Grad 0 ist ein Baum als Blatt



Balancierter Baum
Ist eine Spezialfall der Datenstruktur Baum.
Es gilt:

- Maximal Höhe c*log(n)
- Webei n Anzahl Elemente
- c und n sind unabhängige Konstanten

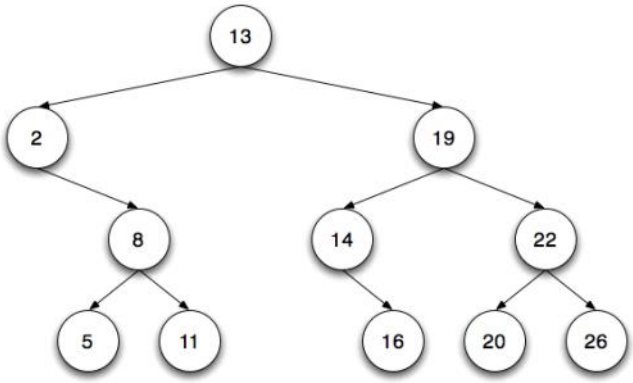


Die Suche bei balancierten Bäumen ist viel effizienter.

Binärbäume

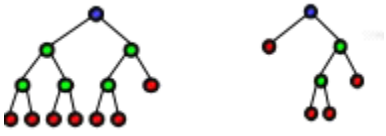
Ist ein geordneter Baum mit Grad 2.
Alle höheren Grade können auf binäre Bäume reduziert werden.

Bei binären Suchbäumen gilt:

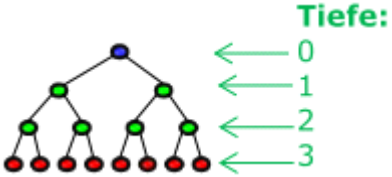


- Elemente werden auf einer Zeile von Links nach Rechts sortiert
- Die Wurzel des Baumes ist der Pivot für die Werte, spricht kleinere Elemente werden rechts eingefügt.
  - link < 13
  - rechts > 13
- Für jede Eingabe wird diese Prüfung an jedem Knoten gemacht

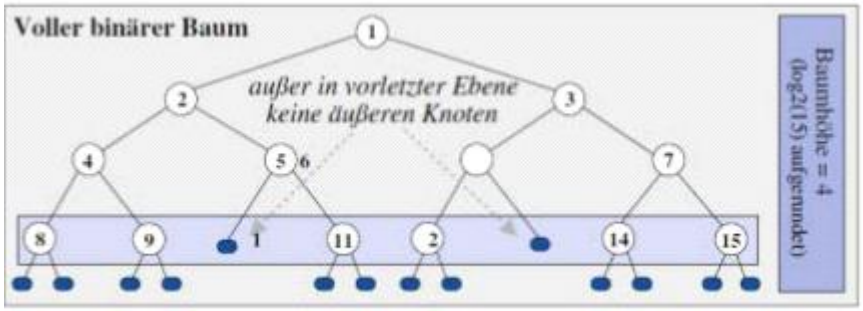
Eigenschaften



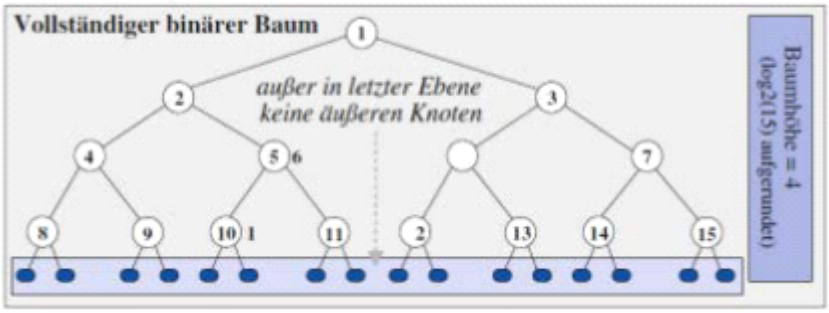
Voll: Jeder Knoten ist entweder ein Blatt oder beseitzt zwei Kinder (also keine Halbblätter)



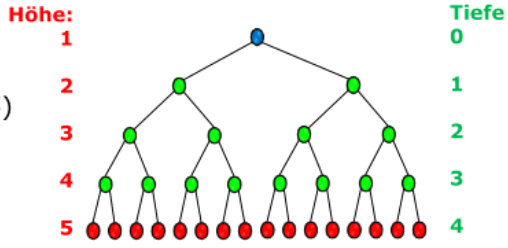
Vollständig: alle Blätter haben die gleiche Tiefe (defniert durch Anzal Bögen bis zur Wurzel). Ist sehr selten der Fall
Für binäre Bäume gilt:



Vorletzte und letzte Ebene können äussere Knoten haben.



Nur die letzte Ebene hat äussere Knoten

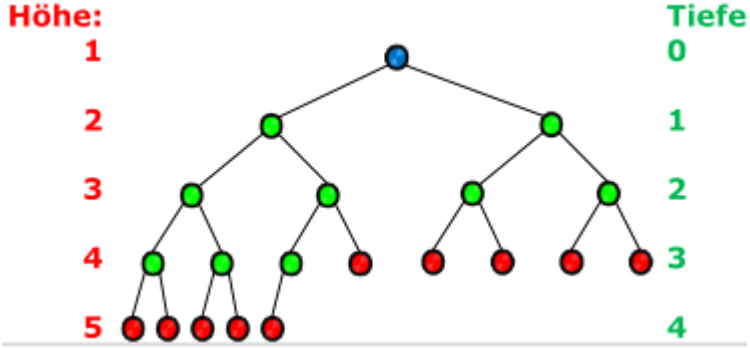


Bei einem **vollständiger Binärbaum**
mit der Höhe h gilt: (z.B. für h = 5)
Es gibt 2^h-1 Knoten => $2^5 - 1 = 31$
Es gibt 2^{h-1}-1 innere Knoten => $2^4 - 1 = 15$
Es gibt 2^{h-1} Blätter => $2^4 = 16$
Es gibt auf der **Höhe h** genau
2^{h-1} Knoten
z.B. h = 3 => $2^{3-1} = 4$
oder gleiche Aussage
Es gibt auf der **Tiefe t**
(0<= **t** <= h-1)
genau 2^t Knoten z.B. t = 2 => $2^2 = 4$
Es gilt ja: **h = t + 1**

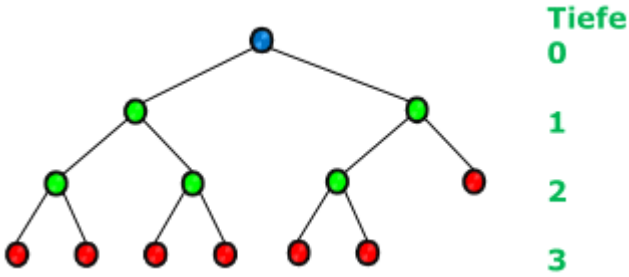
Die minimale Höhe h eines Binärbaumes lässt sich aus der Anzahl Knoten n berechnen:

$\log_2(n) = h$

Lösung immer abrunden.
Beispiel:



Log10(20)/log10(2)=4.321928094887362
Achtung!: Ergibt immer die Tiefe

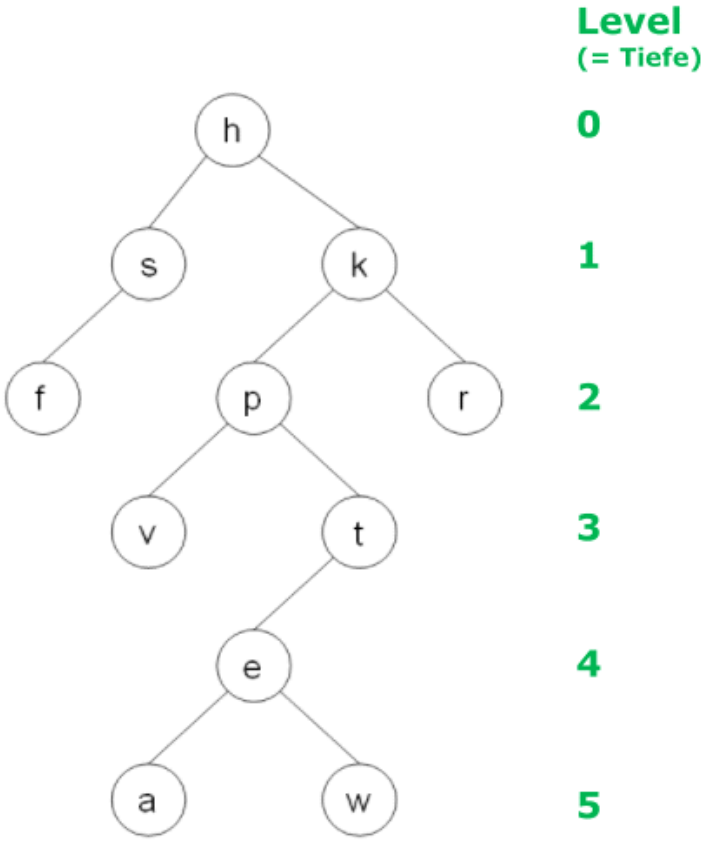


Log10(13)/log10(2)=3.700439718141092

Travesierung von Bäumen

16 December 2014 10:00

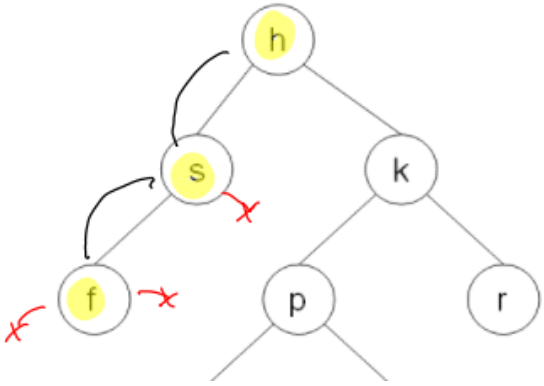
Gegeben ein nicht ausbalancierter Baum.
Dieser soll in verschiedenen Methoden durchlaufen werden:



Allgemein gilt:
Ist kein Schritt durchführbar, geht man einen Teilbaum zurück. Da das ganze rekursiv durchgeführt wird.

Preorder (WLR)
Durchlaufe den Baum rekursiv in folgenden Schritten:

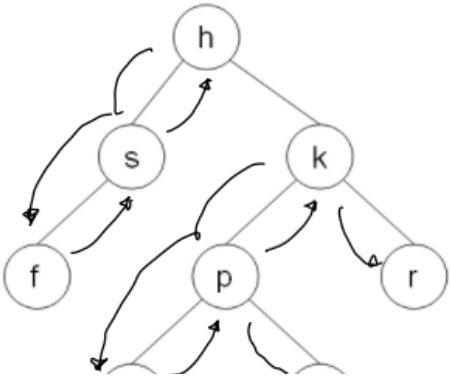
1. Lies Wurzel
2. Traversiere linken Teilbaum in Preorder
3. Traversiere rechten Teilbaum in Preorder



h	s	f	k	p	v	t	e	a	w	r

Inorder (LWR)

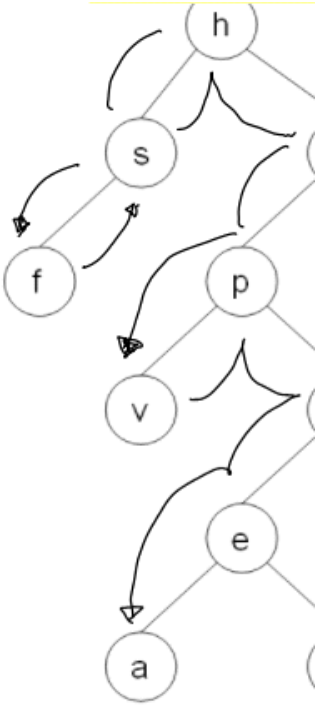
1. Traversiere linken Teilbaum in Inorder
2. Lies Wurzel
3. Traversiere rechten Teilbaum in Inorder



f	s	h	v	p	a	e	w	t	k	r

Postorder (LRW)

1. Traversiere linken Teilbaum in Postorder
2. Traversiere rechten Teilbaum in Postorder
3. Lies Wurzel

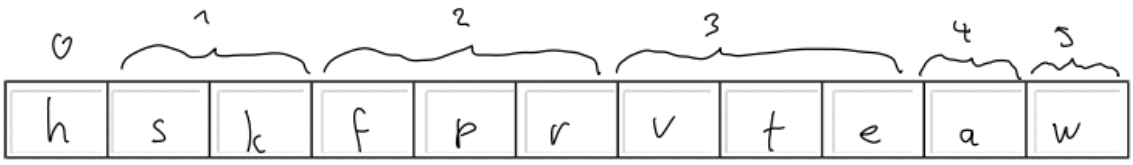


f	s	v	a	w	e	t	p	r	k	h

Levelorder

Beginnend beim ersten Level werden alle Elemente pro Level gelesen.

1. LevNo:=0
2. solange Ebene levNo
    a. Lies die Knoten EvenelevNo von links nach rechts
    b. levNo:=levNo+1


0
1
2
3
4
5
h
s
k
f
p
r
v
t
e
a
w