



# Informatik 1

Informationsdarstellung und Algorithmen



**Irene Rothe**

Zi. B 241

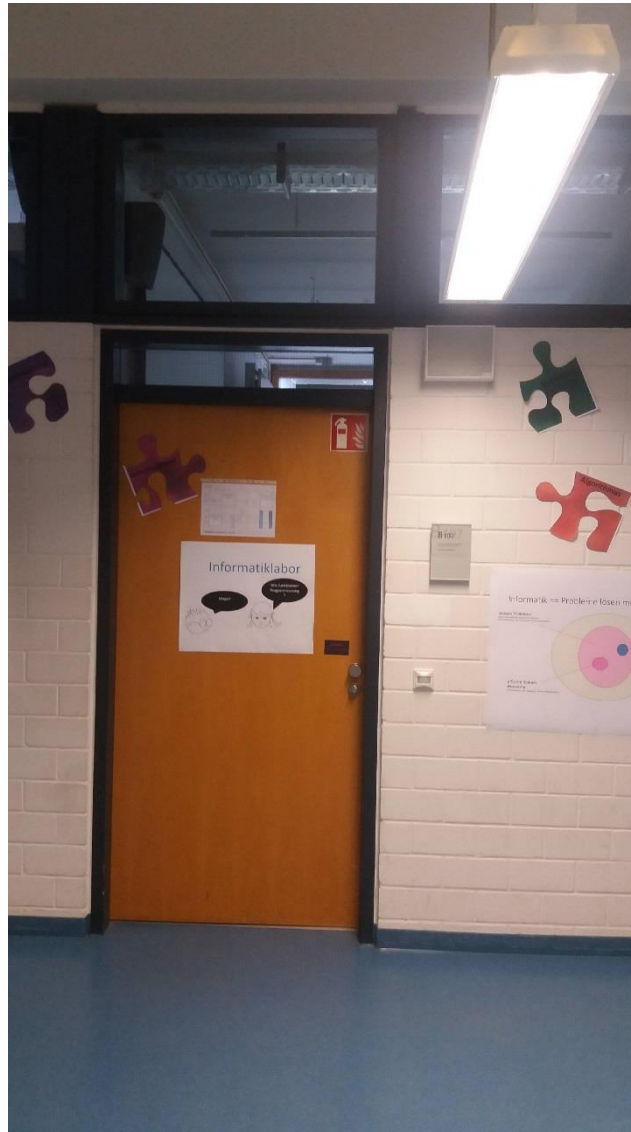
[irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)

Instagram: irenerothesdesign



Hochschule  
Bonn-Rhein-Sieg

**Vorlesung\_D\_Algorithmen**



# Informatiklabor

Läuft Dein Leben stets nach Plan,  
fährst Du selten Deutsche Bahn.



# Informatik: 2 Semester für Ingenieure

## Informatik = Lösen von Problemen mit dem Rechner

✓ Zum Lösen von Problemen mit dem Rechner braucht man **Programmierfähigkeiten** (nur mit Übung möglich): Was ist Programmieren?

✓ Was ist ein Flussdiagramm?

### → **Programmiersprache C:**

- ✓ Elementare Datentypen
- ✓ Deklaration/Initialisierung
- ✓ Kontrollstrukturen: if/else, while, for
- ✓ Funktionen
- Felder (Strings)
- Zeiger
- struct
- Speicheranforderung: malloc
- Listen
- Bitmanipulation

→ Wie löst der Rechner unsere Probleme? → mit **Dualdarstellung** von Zeichen und Zahlen und mit Hilfe von **Algorithmen**

→ Ein Beispiel für ein Problem: **Kryptografie**

→ Sind Rechner auch Menschen? → **Künstliche Intelligenz**

→ Für alle Probleme gibt es viele Algorithmen. Welcher ist der Beste? → **Aufwand** von Algorithmen



# Informatik: ein Semester für TJs und VTs

Informatik = Lösen von Problemen mit dem Rechner

- ✓ Zum Lösen von Problemen mit dem Rechner braucht man **Programmierfähigkeiten (nur mit Übung möglich)**: Was ist Programmieren? Kleine Beispiele mit Code und Flussdiagramm → Vorbereitung auf die Projektwoche

→ Wie löst der Rechner unsere Probleme? → mit **Dualdarstellung** von Zeichen und Zahlen und mit Hilfe von **Algorithmen**

→ Was ist ein Algorithmus? Beispiele von Algorithmen: **Sortieren** und **Suche**

→ Ein Beispiel für ein Problem: **Kryptografie**

→ Noch ein Beispiel für ein Problem: **Bildverarbeitung**





→ Sind Rechner auch Menschen? → **Künstliche Intelligenz**

→ Für alle Probleme gibt es viele Algorithmen. Welcher ist der Beste? → **Aufwand** von Algorithmen

→ ...



# Design der Folien

-  hinterlegt sind alle Übungsaufgaben. Sie sind teilweise sehr schwer, bitte absolut nicht entmutigen lassen! Wir können diese in Präsenz besprechen oder über Fragen im Forum.
-  hinterlegte Informationen und grüne Smileys sind wichtig und klausurrelevant.
- Alles hinter „**Achtung**“ unbedingt beachten!
-  verwende ich, wenn überraschende Probleme auftreten können. Wenn Sie schon programmiererfahrend sind, können das eventuell besonders große Überraschungen für Sie sein, wenn Sie eine andere Sprache als C kennen.
- „Tipp“ benutze ich, um Ihnen einen Weg zu zeigen, wie ich damit umgehen würde.
- „Bemerkung“ in Folien beziehen sich meist auf Sonderfälle, die nicht unbedingt klausurrelevant sind, aber für Sie beim Programmieren eine Bedeutung haben könnten
-  hinter diesem Symbol ist ein Link fürs Anhören bzw. Gucken weiterer Infos

In dieser Vorlesung:

## → Informationsdarstellung

- Codierung
- Alphabete und Sprache
- Dualzahldarstellung von Zeichen und Zahlen
- Hexadezimaldarstellung
- Addition und Komplement von ganzen Zahlen
- Darstellung von Gleitkommazahlen
- Schaltnetze
- Halbaddierer

## Algorithmen



#### Aufbau der Folien:

- Am Anfang motiviere ich gerne mit einem Beispiel, das eventuell schwer verständlich ist. Wem das nicht zusagt, dem empfehle ich, diese Folien zu überspringen.
- Weiter arbeite ich mit vielen Beispielen, die oftmals immer wieder das Gleiche erklären nur auf unterschiedliche Arten. Hat man einen Sachverhalt einmal verstanden, braucht man eventuell diese Beispiele nicht.
- Folien, die mit **Einschub** beginnen, beinhalten Zusatzinformationen, die nicht nötig für das Verständnis des Themas sind.
- Grün hinterlegte Informationen sind das, was Sie aus der Vorlesung rausnehmen sollen, alles andere sind vertiefende Informationen und Motivation.



# Codierung: Motivation - Zahlendarstellung

Warum rechnen wir mit **10** Zahlen?

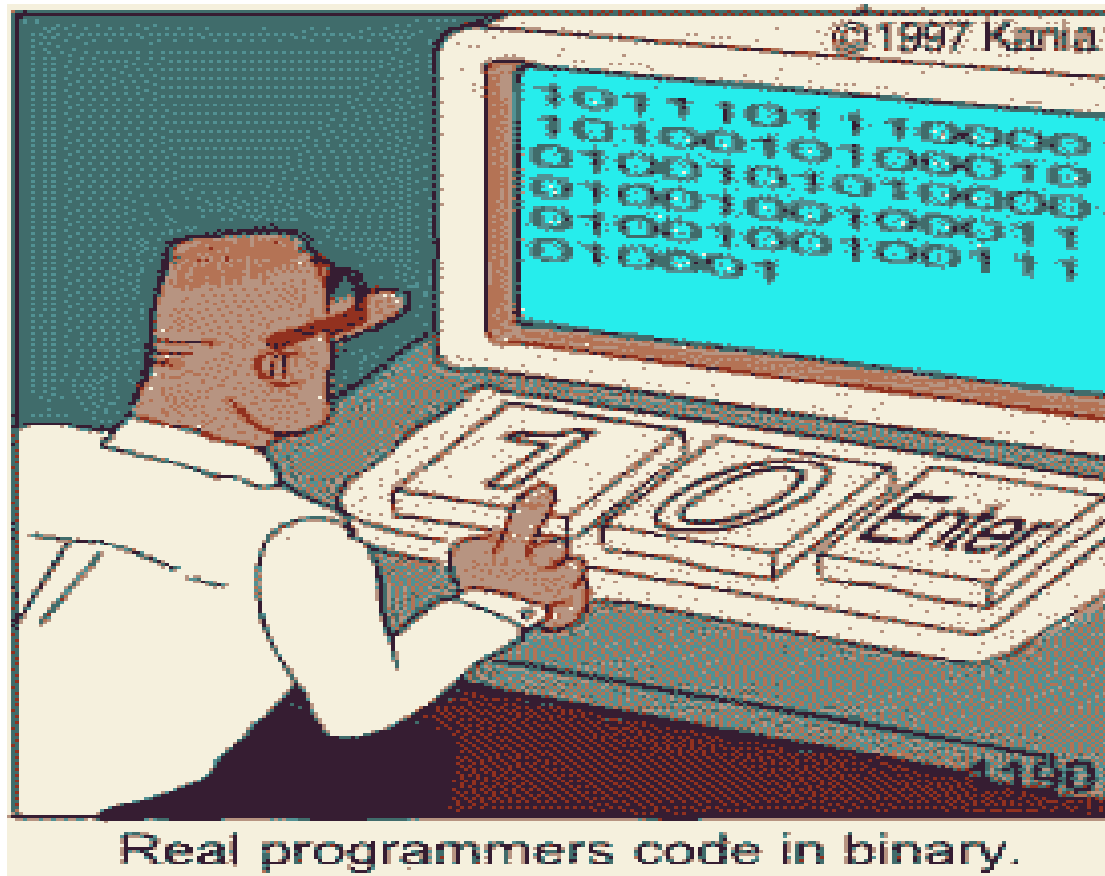
Wie viele Finger hat ein Rechner?

Kann man damit alle Zeichen der Tastatur darstellen?

„There are only 10 types of people, those who understand binary and those who don't.“



# Informationsdarstellung im Rechner



[mendrixx.de/images.html](http://mendrixx.de/images.html)

# Kommunikation mit dem Rechner

## Menschensprache:

1. Lese Speicherzelle 394 aus
2. Addiere Speicherzelle 395
3. Schreibe das Ergebnis in Speicherzelle 396

## Rechnersprache:

1. 10001000 00000001 10001010
2. 11001001 00000001 10001011
3. 10001001 00000001 10001100

Bemerkung: 394 = 1 10001010 in Dualzahldarstellung

## Assemblersprache:

1. RD 018A
2. ADD 018B
3. WRI 018C

Bemerkung: 0000=0 in Hex, 0001=1 in Hex, 1000=8 in Hex, 1010=A in Hex

## C:

1. int a,b,c;
2. c=a+b;

Leider habe ich die Quelle verloren, wo die Binärdarstellung für RD, ADD und WRI herkommt. Hier eine andere Quelle für Assemblerbefehle:

[https://de.wikibooks.org/wiki/Assembler-Programmierung\\_f%C3%BCr\\_x86-Prozessoren/\\_Befehlsliste](https://de.wikibooks.org/wiki/Assembler-Programmierung_f%C3%BCr_x86-Prozessoren/_Befehlsliste)



Videolink

<https://youtu.be/BHW0ha0QDI0>



# Informationsdarstellung: Fragen

- Wie repräsentiert der Rechner Zahlen, Buchstaben, Programme und Zeichen?
- Was ist der ASCII-Code?
- Was ist eine Binärzahl?
- Was ist eine Hexadezimalzahl?
- Wie wird eine Dezimalzahl in eine Binärzahl umgewandelt?

# Informationsdarstellung: Wie repräsentiert ein Rechner Informationen?

Da Rechner bis jetzt nur mit Strom arbeiten, kann man NUR 2 verschiedene Zustände darstellen:

- Strom da / Strom nicht da = Licht an / Licht aus (z.B. 0 Volt / 5 Volt)
- ODER: 1 / 0
- Ein **Bit** bedeutet die kleinste Speichereinheit in einem Rechner, sie kann entweder 0 oder 1 sein, andere Möglichkeiten gibt es vorerst praktikabel nicht.

# Codierung

- Man muss durch Nullen und Einsen alles darstellen können, was wir z.B. in der deutschen Sprache auch können.
- Codierung von Daten: eindeutige Abbildung eines Zeichensatzes (z.B. deutsche Buchstaben und alle Zahlen) in einen anderen Zeichensatz (0,1)
- für Rechner brauchen wir dies, um unsere Daten (Zahlen und Zeichen und damit auch Wörter) in digitalen Systemen repräsentieren zu können
- ist nicht gemeint im Sinne der Verschlüsselung zur Geheimhaltung
- Beispiele: ASCII (American Standard Code for Information Interchange), Morsealphabet



# Alphabete und Sprachen

- ein **Alphabet** ist eine Menge von Symbolen
- ein **Wort** ist die endliche Aneinanderreihung von diesen Symbolen
- eine **Sprache** ist eine Menge von Worten über einem Alphabet
- Beispiele:
  - Dezimaldarstellung (10 Elemente): Menge der Symbole = {0,1,2,3,4,5,6,7,8,9}
  - Dual(Binär)-darstellung (2 Elemente): Menge der Symbole = {0,1}
  - Hexadezimaldarstellung (16 Elemente): Menge der Symbole = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Beispiel

dezimal	0	1	2	3	4	5	6	7	8	9
binär	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Intelligenztest (Suche nach Außerirdischen): Wer versteht diese Codierung?

# Darstellung von Zeichen mit 0 und 1

- Wie viel Speicher (Bits) braucht man mindestens, um alle wichtigen Zeichen des Englischen oder gar jeder europäischen Sprache darstellen zu können?
- mit **7 Bits** kann man 128 Zeichen darstellen, was gut für ein paar Steuerzeichen, den Zahlen und dem englischen Alphabet (Klein- und Großbuchstaben) reicht (ASCII, siehe später)
- mit **8 Bits** kann man 256 Zeichen und damit alle europäischen Sprachzeichen darstellen
- für chinesisch oder japanisch braucht man viel mehr Bits: 16 Bits, genannt Unicode

Zeichen	ASCII-Code (7Bits)
A	1000001 (dezimal 65)
B	1000010 (dezimal 66)
C	1000011 (dezimal 67)
D	1000100 (dezimal 68)



# Darstellung von Zeichen: ASCII-Tabelle

ASCII	Zeichen	ASCII	Zeichen	ASCII	Zeichen	ASCII	Zeichen	ASCII	Zeichen	ASCII	Zeichen
000	(Null)	046	.	092	\	138	ê	184	ô	230	ú
001	!	047	/	093	]	139	í	185	í	231	û
002	"	048	0	094	^	140	î	186	î	232	ü
003	£	049	1	095	_	141	ï	187	ï	233	ü
004	\$	050	2	096	`	142	À	188	À	234	Û
005	¥	051	3	097	a	143	á	189	á	235	Ü
006	•	052	4	098	b	144	Â	190	â	236	Ý
007	°	053	5	099	c	145	Û	191	Û	237	Ÿ
008	¨	054	6	100	d	146	Ä	192	Ä	238	
009	©	055	7	101	e	147	Å	193	Å	239	
010	ª	056	8	102	f	148	Ö	194	Ö	240	-
011	«	057	9	103	g	149	ó	195	ó	241	£
012	¬	058	:	104	h	150	Ô	196	ô	242	
013	®	059	;	105	i	151	Ù	197	ù	243	¼
014	¯	060	<	106	j	152	Ú	198	ú	244	½
015	°	061	=	107	k	153	Û	199	û	245	¾
016	±	062	>	108	l	154	Ü	200	ü	246	*
017	²	063	?	109	m	155	Ý	201	ý	247	
018	³	064	@	110	n	156	Û	202	Û	248	
019	´	065	A	111	o	157	Ü	203	ü	249	-
020	µ	066	B	112	p	158		204		250	-
021	¶	067	C	113	q	159		205		251	*
022		068	D	114	r	160		206		252	*
023		069	E	115	s	161		207		253	*
024		070	F	116	t	162		208		254	
025		071	G	117	u	163		209		255	(Leer)
026		072	H	118	v	164		210			
027		073	I	119	w	165		211			
028		074	J	120	x	166		212			
029		075	K	121	y	167		213			
030		076	L	122	z	168		214			
031		077	M	123	[	169		215			
032	(Leertaste)	078	N	124	]	170		216			
033		079	O	125	^	171		217			
034		080	P	126	_	172		218			
035		081	Q	127	`	173		219			
036		082	R	128	A	174		220			
037		083	S	129	B	175		221			
038		084	T	130	C	176		222			
039		085	U	131	D	177		223			
040		086	V	132	E	178		224			
041		087	W	133	F	179		225			
042		088	X	134	G	180		226			
043		089	Y	135	H	181		227			
044		090	Z	136	I	182		228			
045		091	[	137	J	183		229			

# Darstellung von Ganzzahlen

Basis-Zahlendarstellung allgemein:

$$zahl = \sum_{i=0}^n a_i b^i = a_0 b^0 + a_1 b^1 + \dots + a_{n-1} b^{n-1} + a_n b^n$$

Beispiel für Basis  $b=10$ ,  $b=2$ ,  $b=16$ :

$$(196)_{10} = 1 * 10^2 + 9 * 10^1 + 6 * 10^0$$

$$(11000100)_2 = 1 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0$$

$$(C4)_{16} = 12 * 16^1 + 4 * 16^0$$

Bemerkung zu Zusammenhang Dualzahlen zu Hexa-Zahlen: immer 4 Bits ergeben einen Hexa-Zahl.

Beispiel: 196 ist dual 1100 0100, die 1100 ist dabei dezimal die 12 und 0100 ist dezimal die 4. Das stimmt wiederum mit der Hexa-Darstellung überein: C4. So können riesige Dualzahlen sehr schön kurz in Hexa-Darstellung geschrieben werden.



# Hexadezimal-Darstellung

Symbole (16 Symbole): 0 1 2...9 A B C E F

→ immer vier Dualzahlen entsprechen einer Hexadezimalzahl

Dezimal (Basis 10)	Dual (Basis 2)	Hexa (Basis 16)
42	10 1010	2A
15	1111	F
31	1 1111	1F
14277	11 0111 1100 0101	37C5

# Dualzahlendarstellung: Übung

Stellen Sie 2017 dual dar!

Lösung: <https://youtu.be/3QjQHStkqfl>



# Addition und Multiplikation mit dualen Ganzzahlen

Additionsregeln:

- $0+0=0$
- $0+1=1$
- $1+0=1$
- $1+1=0$  mit Übertrag 1

Beispiel:  $1001101$

$$\begin{array}{r} 1001101 \\ + 1000100 \\ \hline 10010001 \end{array}$$

Übertrag

Multiplikationsregeln:

- $0*0=0$
- $0*1=0$
- $1*0=0$
- $1*1=1$

Beispiel:  $1011011 * 101$

$$\begin{array}{r} 1011011 * 101 \\ \hline 1011011 \\ + 0000000 \\ + 1011011 \\ \hline 111000111 \end{array}$$

# Addition: Übung:

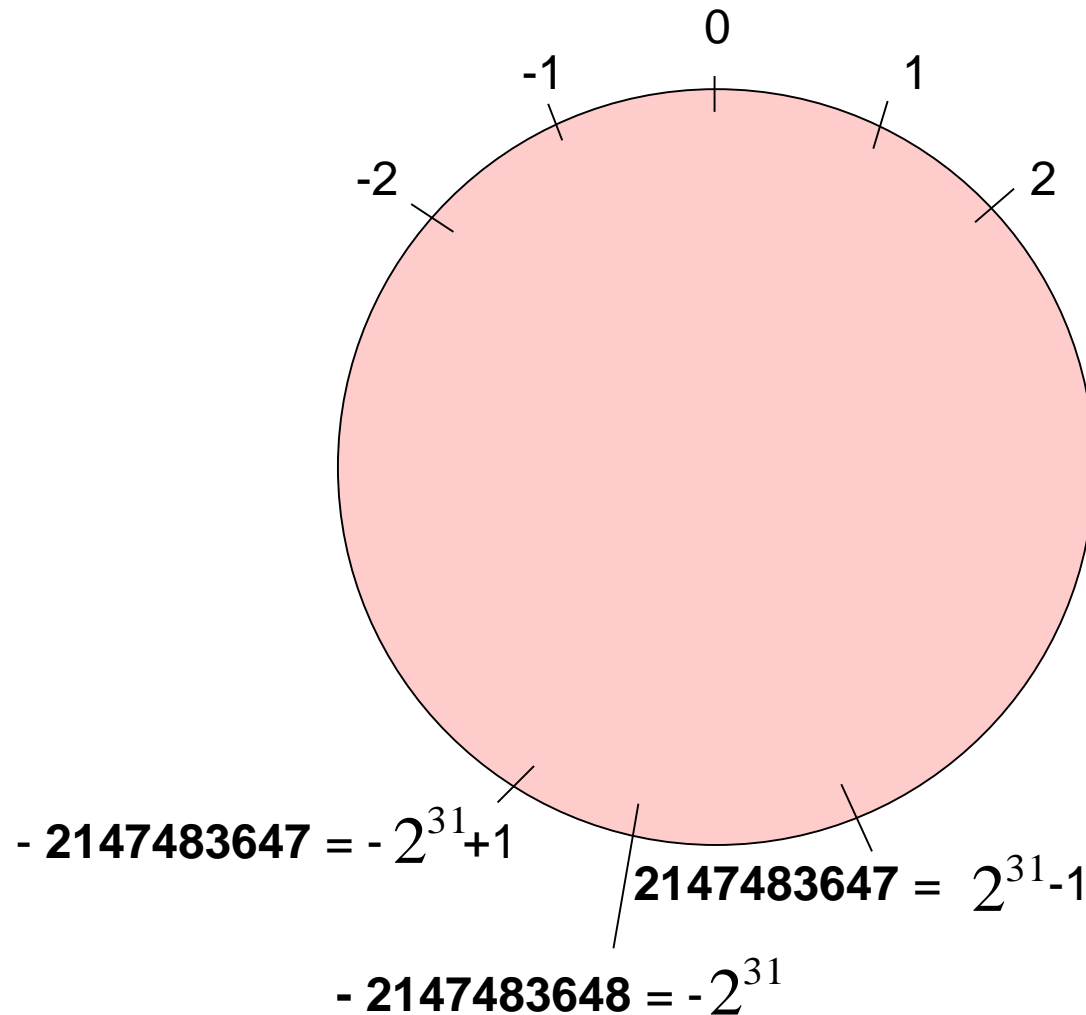
Addieren Sie und überprüfen Sie Ihr Ergebnis dezimal!

$$\begin{array}{r} 1110111 \\ + 1010101 \\ \hline \end{array}$$

Lösung: <https://youtu.be/zLCf1vFEqrs>



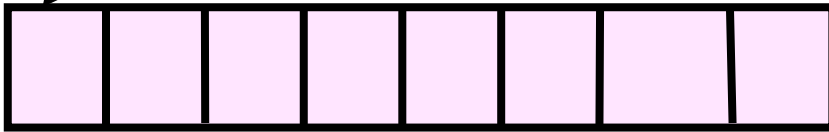
# Duale ganze Zahlen – Laufen immer im Kreis



Es entstehen so  
wenigstens nie Abstürze

# Vorzeichen bei Ganzzahlen

Das erste Bit bei der Ganzzahldarstellung wird als Vorzeichen interpretiert. Hier ein Beispiel für einen 8 Bitrechner.



7 Bits können zur eigentlichen Zahldarstellung dann nur noch verwendet werden.



# Negative duale Ganzzahlen

1. Positive Zahl in Dualdarstellung
2. Bildung des Komplements dieser Zahl
3. Zahl + 1
4. Test: positive Zahl + negative Zahl = 0

Beispiel: -42 im 8 Bit-Rechner

1. Dualzahlendarstellung: 00101010
2. Komplement: 11010101
3. Addition von 1: 11010110
4. Test:  $00101010 + 11010110 = 00000000$

# Negative duale Ganzzahlen: Übung

Stellen Sie -88 dar!

8 Bit-Rechner

1. Dualzahlendarstellung von 88:
2. Komplement:
3. Addition von 1:
4. Test: ... + ... = 00000000



# Darstellung von Gleitkommazahlen

$$zahl = + / - m \cdot b^e$$

b: Basis, z.B. 10 oder 2

m: Mantisse,  $1/b < |m| < 1$  oder  $m = 0$

e: Exponent,  $-E \leq e \leq +E$

$$zahl = \pm 0.a_1a_2\dots a_nb^e, a_1 \neq 0, 0 \leq a_i \leq b-1$$

Achtung: Verarbeitung von Gleitkommazahlen ist nicht exakt!

# Gleitkommazahlen: Float und Double

## Float (32 Bit):

b: 2

Vorzeichen: 1 Bit

m: 23 Bit

E: 8 Bit ( $2^{\text{hoch}(8-1)} = 128$ ) → es können also Dezimalzahlen bis ca.  $10^{\text{hoch} 38}$  dargestellt werden

## Double (64 Bit):

b: 2

Vorzeichen: 1 Bit

m: 52 Bit

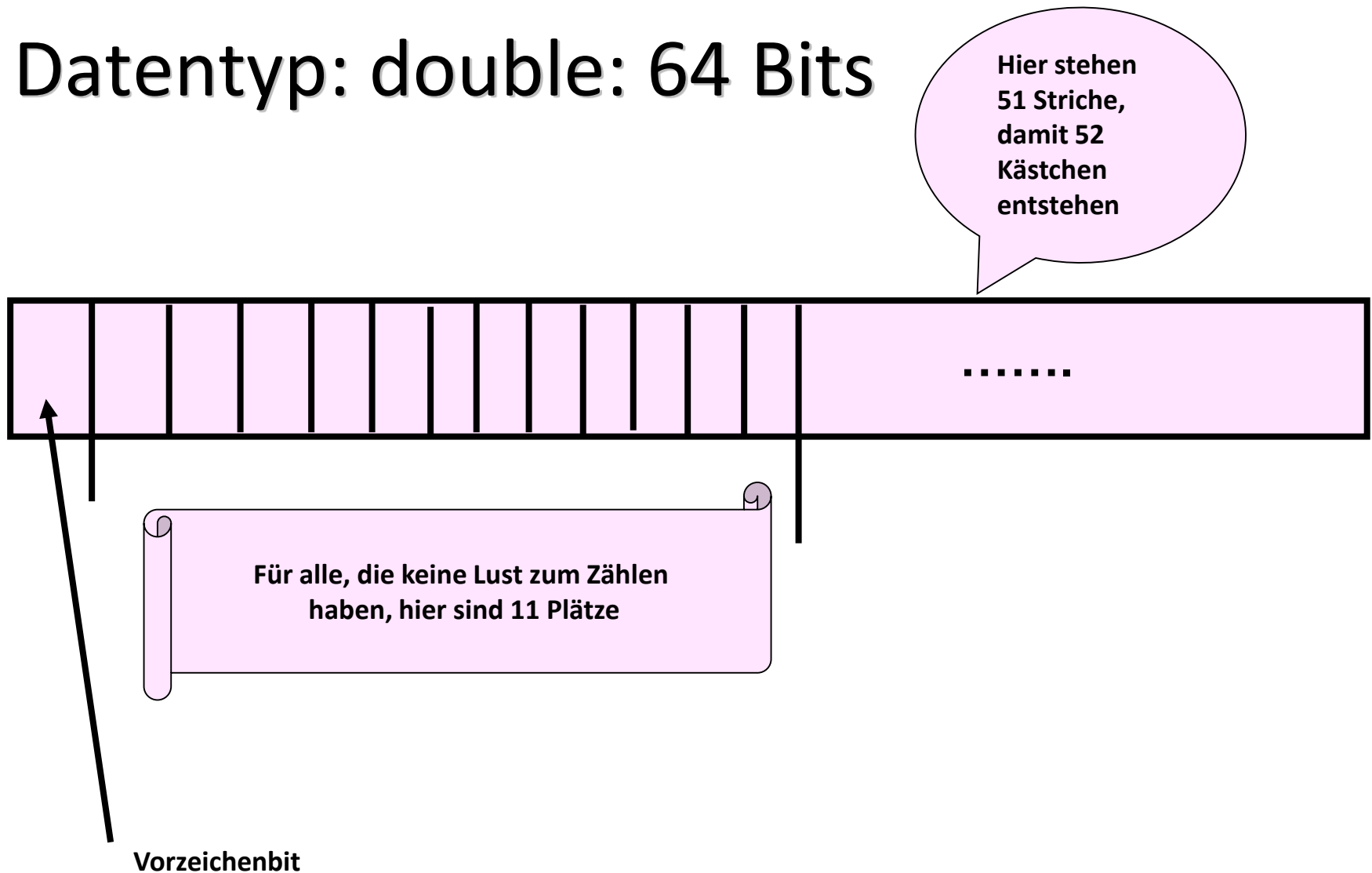
E: 11 Bit ( $2^{\text{hoch}(11-1)} = 1024$ ) → es können also Dezimalzahlen bis ca.  $10^{\text{hoch} 308}$  dargestellt werden

### Achtung:0

- Verarbeitung von Gleitkommazahlen ist nicht exakt!
- es hängt vom Rechner ab, ob 0.1111112 gleich 0.1111111 ist
- positives Unendlich: INF, z.B. zahl/0
- negatives Unendlich: -INF, z.B. -zahl/0
- NaN (not a number) oder IND, z.B. 0/0 oder Wurzel von -1



# Datentyp: double: 64 Bits



# Double: Beispiel - Dezimalzahl 0.1 in Dualzahl

$$\begin{aligned}(0.1)_{10} &= 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + \\&= 0.0625 + 0.03125 + 0.0039063 + 0.0019531 + \dots \\&= 0.0996094 + \dots \\&= 0.0001 \quad 1 \quad 001 \quad 1 \quad \dots \\&= 0.00011001100110011\dots 0011001100\dots\end{aligned}$$

→ Das heißt also, 0.1 in Dualzahldarstellung ist periodisch.



# Achtung: Beispiel - unexakte Darstellung von Gleitkommazahlen

Double: 8 Byte=64 Bits: 11 Bits für Exponenten, 52 Bits für Mantisse, 1 Bit für Vorzeichen

- 2.3 -> 0 10000000000 0010011001... 001 -> 2.29999999
- 230 -> 0 100000000110 1100110000... 000 -> 230

Periode 001

Das bedeutet, dass beim Programmieren Folgendes zu beachten ist:  $(\text{int}) 2.3 * 100 \neq 230$



# Zahlendarstellung: Übung

Rechnen Sie um:

Dezimal	Binär	Hexadezimal	Basis 3
<b>131</b>			
	<b>1001 0100</b>		
		<b>A 3</b>	
			<b>12200</b>

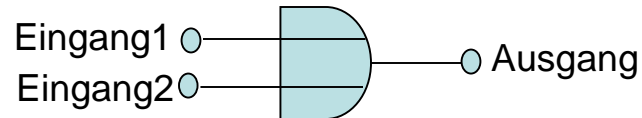




## Einschub: Wie setzt der Rechner Operationen in der CPU um? → Schaltnetze

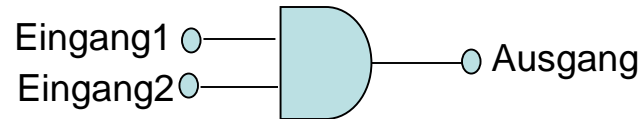
Eingang1	Eingang2	Ausgang
0	0	0
1	0	1
0	1	1
1	1	1

ODER-Gatter umgesetzt als Parallelschaltung  
 $\text{Eingang1} + \text{Eingang2} = \text{Ausgang}$



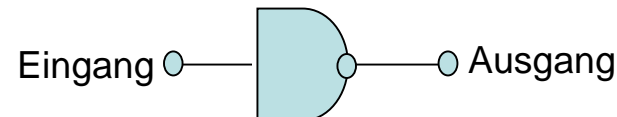
Eingang1	Eingang2	Ausgang
0	0	0
1	0	0
0	1	0
1	1	1

UND-Gatter umgesetzt als Reihenschaltung  
 $\text{Eingang1} * \text{Eingang2} = \text{Ausgang}$

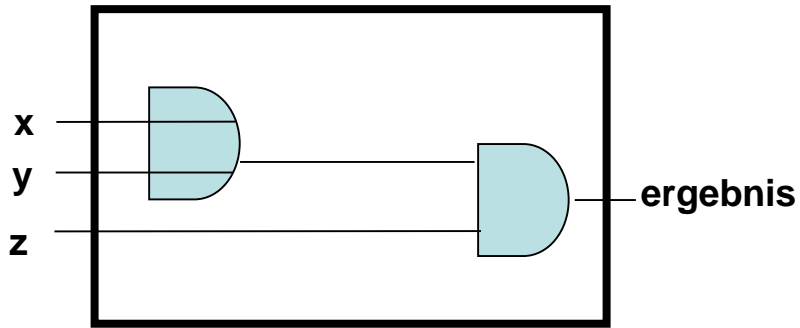


Eingang	Ausgang
0	1
1	0

NICHT-Gatter



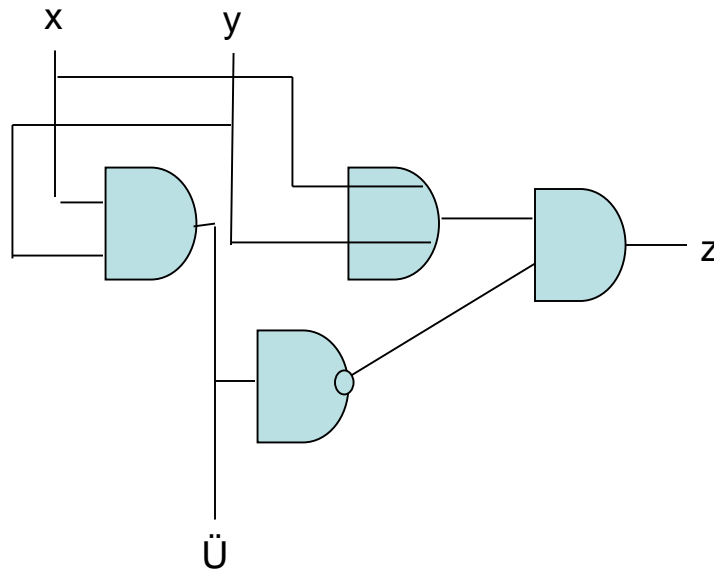
# Einschub: im Rechner: Schaltnetz und Logiktablelle (Beispiel)



x	y	z	Ergebnis ((x  y)&&z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

# Einschub: Addition im Rechner - Halbaddierer

x	y	z	Ü
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1



„Der Computer kann alles, sonst nichts.“

Siehe auch: <http://www.megaprocessor.com/>

# ✓ Informationsdarstellung

## Algorithmen

- Algorithmendefinition
- Sortieren:
  - Insertionsort
  - Selektionsort
  - Bubblesort
  - Quicksort
  - Vergleich der Algorithmen
- Suche:
  - Sequentielle Suche
  - Binärsuche
- Algorithmus für kürzeste Weg von Dijkstra



# Algorithmen: Motivation

Kennen sie Algorithmen?

Benutzen Sie Algorithmen?

Haben Sie schon mal eine Bedienungsanleitung gelesen und benutzt?

Sehen Sie einen Zusammenhang zwischen Algorithmen und Anleitungen?



# Algorithmen: Beispiele

- Kochrezept
- Bedienungsanleitung („Jetzt, wo ich weiß, wie es geht, verstehe ich auch die Anleitung.“)
- Sieb des Eratosthenes: [https://de.wikipedia.org/wiki/Sieb\\_des\\_Eratosthenes](https://de.wikipedia.org/wiki/Sieb_des_Eratosthenes)  
(rechts super Animation, wie das funktioniert)
- Sortieren einer unsortierten Liste von Namen
- Suche nach einem bestimmten Buch
- Berechnung der Zahl  $e = 2.7182\dots$

# Algorithmus: Definition

Was wünscht man sich von einer guten Bedienungsanleitung?

Ein **Algorithmus** ist eine **endliche** Folge von Schritten, die zur Lösung eines Problems abgearbeitet werden müssen. Die Schritte müssen **präzise** formuliert sein und **effektiv** (tatsächlich ausführbar).

Der Prozess der Abarbeitung der Schritte auf eine gegebene Eingabe **stoppt** nach einer **endlichen** Anzahl von Schritten.

Zu jedem Zeitpunkt der Abarbeitung des Algorithmus benötigt der Algorithmus nur **endlich** viele Ressourcen.

# Algorithmen: Übung

Sind das Algorithmen?

- Problem Wegbeschreibung:

1. Fahre von Deiner Wohnung bis zur nächsten Kreuzung.
2. Biege rechts ab.
3. Fahre bis zum nächsten Kreisverkehr.
4. Bleibe im Kreisverkehr, bis Dir ein Engel erscheint.

- Problem Geld:

1. Spare, bis Du 100 Euro hast.
2. Kaufe Aktien, die gerade sehr billig sind.
3. Verkaufe die Aktien, wenn sie am teuersten sind.

- Problem Multiplikation:

Multiplikation zweier ganzen Zahlen, wie Sie es in der Schule gelernt haben





# Algorithmen: Beispiele: Sortieren und Suchen

Warum werde ich Ihnen zuerst etwas übers *Sortieren* erzählen und danach übers *Suchen*?

Können Sie Bücher nach Autoren sortieren?

Finden Sie ein Buch in einer Bibliothek, wenn Sie den Autor kennen?

Frage: Warum können Sie das? Wer hat Ihnen das beigebracht? Irgendwie lernt man das so nebenbei.

# Algorithmen: Sortieren

Sortieren ist ein grundlegendes Problem in der Informatik, Rechner verbringen angeblich 25% ihrer Zeit mit Sortieren, wer neue bessere Idee zum Sortieren hat, kann eine Menge Geld verdienen

## Aufgabe:

- Ordnen von Datensätzen (z.B. Namensliste, Einkaufsliste, Bücher, Dateien), die Schlüssel (natürliche Zahlen) enthalten
- Umordnen der Datensätze, so dass klar definierte Ordnung der Schlüssel besteht
- alle Suchalgorithmen gehen von einer sortierten Liste aus

→ Vereinfachung: nur Betrachtung der Schlüssel (natürlichen Zahlen)

Beispiele: Sortieren durch **Einfügen**, durch **Auswählen**, durch **Mergen**, **Bubble-Sort**, **Quick-Sort**



# Algorithmen: Sortieren durch Auswählen (Englisch: Selection Sort)

**Idee:** Brute-Force-Ansatz

1. Suche jeweils kleinsten Wert, und tausche diesen mit dem Wert an der ersten Stelle,
2. fahre dann mit der um 1 kleineren Liste fort, also dem 2. Wert und
3. dann immer so weiter

**Nachteil:** Laufzeit hängt sehr von der Sortierung der Daten ab

**Vorteil:** braucht nur sehr wenige Umsortierungen (ist gut, wenn die Datensätze riesig sind und der Schlüssel klein)

**Vorführung:** <https://de.wikipedia.org/wiki/Selectionsort> weiter unten rechts die Animation ansehen



# Algorithmen: Sortieren durch Einfügen (Insertion Sort)

**Idee:** angeblich die Umsetzung der typischen menschlichen Vorgehensweise beim Sortieren eines Stapels von Karten (z.B. beim Skat)

1. Starte mit der ersten Karte einen neuen Stapel (Aufnehmen in die Hand)
2. Nimm jeweils nächste Karte des Originalstapels: Füge diese an der richtigen Stelle in den neuen Stapel ein

**Nachteil:** sehr ineffizient

Bemerkung: wird in der Klausur nicht abgefragt



# Algorithmen: Sortieren durch Vertauschen (Bubble-Sort)

**Idee:** verschieden große aufsteigende Blasen ("Bubbles") in einer Flüssigkeit sortieren sich quasi von alleine, da größere Blasen die kleineren „überholen“

1. In einer gegebenen Liste tauscht man immer 2 Elemente, wenn das erstere größer als das folgende ist.
2. Dies macht man solange, bis obiger Fall gar nicht mehr auftritt, also bis keine Vertauschungen mehr nötig sind.

**Vorteil:** einfach zu programmieren

**Nachteil:** bei sehr großen Datenmengen sehr langsam, da viele Vertauschungen notwendig sind

→ Bubblesort tanzen: <https://www.youtube.com/watch?v=lyZQPjUT5B4>

→ In Präsenz: mit freiwilligen Studierenden vorführen



# Einschub: Bubble-Sort - Funktion in C

```
void bubblesort (int f[], int anz) {  
    int i=0,j,hilfe,sortiert=0;  
    while ( i<anz && sortiert ==0 ){  
        // Annahme, dass Folge sortiert ist  
        sortiert = 1;  
        for ( j=0 ; j<anz-1 ; j=j+1 ){  
            if ( f[j] > f[j+1] ) {  
                // Vertauschung notwendig  
                hilfe = f[j];  
                f[j]= f[j+1];  
                f[j+1]= hilfe;  
                // Folge war doch nicht sortiert  
                sortiert=0;  
            }  
        }  
        i++;  
    }  
}
```

Ich weiß, dass Sie nicht alles kennen, was in diesem Programm benutzt wird, aber TROTZDEM...

f steht für ein Feld von Werten hintereinander angeordnet, die sortiert werden sollen

# Einschub: Bubble-Sort - Hauptprogramm

```
//Test: 4 6 5 1 2 9
//->1 2 4 5 6 9
#include<stdio.h>
int main(){
    int folge[6] = {4,6,5,1,2,9};
    int anzahl = 6;
    int i;
    bubblesort(folge, anzahl);
    for (i=0; i<anzahl; i=i+1) {
        printf("Folge: %i\n",folge[i]);
    }
    return 0;
}
```

# Algorithmen: Merge-Sort

## Idee

1. Teile die zu sortierende Liste in zwei Teillisten
2. Sortiere diese (rekursives Verfahren!)
3. Mische (sortiere) die Ergebnisse ineinander



# Algorithmen: Quick-Sort

## Idee

- ähnlich wie Merge-Sort durch rekursive Aufteilung: Teile und Herrsche Prinzip mit rekursiven Vorgehen
- weit verbreitet, lädt zum Herumbasteln ein, theoretisch gut analysiert, ist kaum noch zu verbessern
- Vermeidung des Mischvorgangs durch Aufteilung der Teillisten in zwei Hälften bezüglich eines Referenzelementes, wobei
  - in einer Liste alle Elemente größer als das Referenzelement sind
  - in der anderen Liste alle Elemente kleiner sind

**Vorführung:** <https://de.wikipedia.org/wiki/Quicksort>

Quicksort tanzen: <https://www.youtube.com/watch?v=3San3uKKHgg> (ein klein wenig anders als in der Vorlesung)

Bemerkung: Vereinigung verschiedener allgemeiner Konzepte (Rekursion, Teile und Herrsche, Pivotelement)



# Algorithmen – Sortieren: Übung

Sortieren Sie:

- 8 6 7 1 4 2 5 3 via Quicksort
- 6 1 4 2 5 3 via Bubblesort
- 8 6 7 1 4 2 5 3 via Selectionsort

Lösung: <https://youtu.be/PJHuM0kTCn0>



# Einschub: Vergleich der Sortieralgorithmen

	Insertion	Selection	Bubble	Quick
Laufzeit im Mittel	$\frac{n^2 + n - 2}{2}$	$\frac{n^2 + n}{2}$	$O(n^2)$	$O(n \log n)$

Welches Programm würden Sie nehmen?

Diese O-Notation  
wird im 2. Semester  
erklärt. Es  
ermöglicht einen  
Vergleich von  
Algorithmen.

# Algorithmen: Suche

**Wie** suchen Sie einen Namen in einem Telefonbuch?



# Algorithmen: Sequenzielle Suche

Suche nach einem Element in einer sortierten Liste:

1. Starte mit ersten Element der Liste
2. Ist dies nicht das gesuchte Element, überprüfe das nächsten Element in der Liste
3. Solange das gesuchte Element nicht gefunden wurde, gehe von Element zu Element in der Liste

Aufwand in Anzahl der Vergleiche bei einer Liste der Länge  $n$ :

- im besten Fall: 1
- im schlechtesten Fall:  $n$
- Durchschnittswert (bei erfolgreicher Suche):  $n/2$
- Durchschnittswert (bei erfolgloser Suche):  $n$

# Algorithmen: Binäre Suche

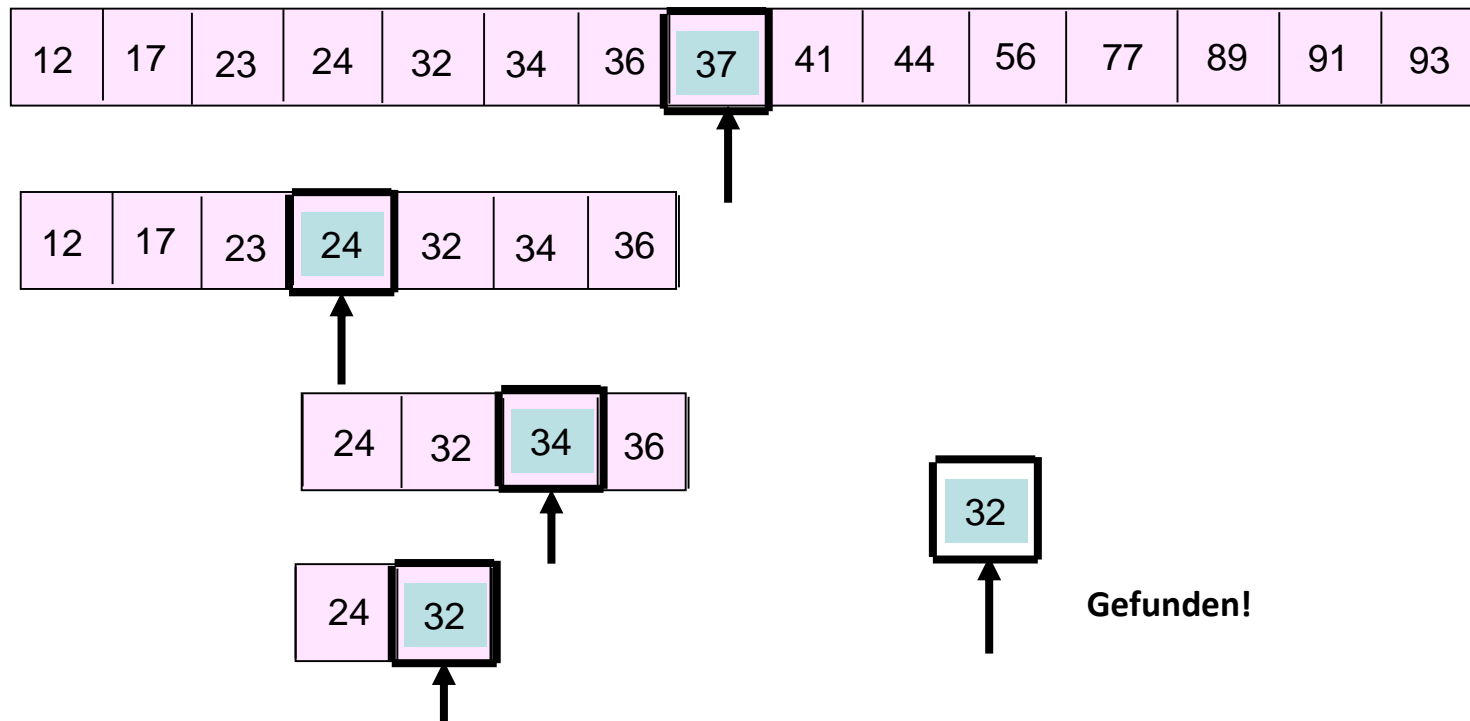
Suche nach einem Element in einer sortierten Liste:

Wähle das mittlere Element der Liste und

1. Prüfe ob gesuchter Wert in der ersten oder in der zweiten Hälfte der Liste ist,
2. Fahre **rekursiv** mit der Hälfte fort, in der sich das gesuchte Element befinden muss.

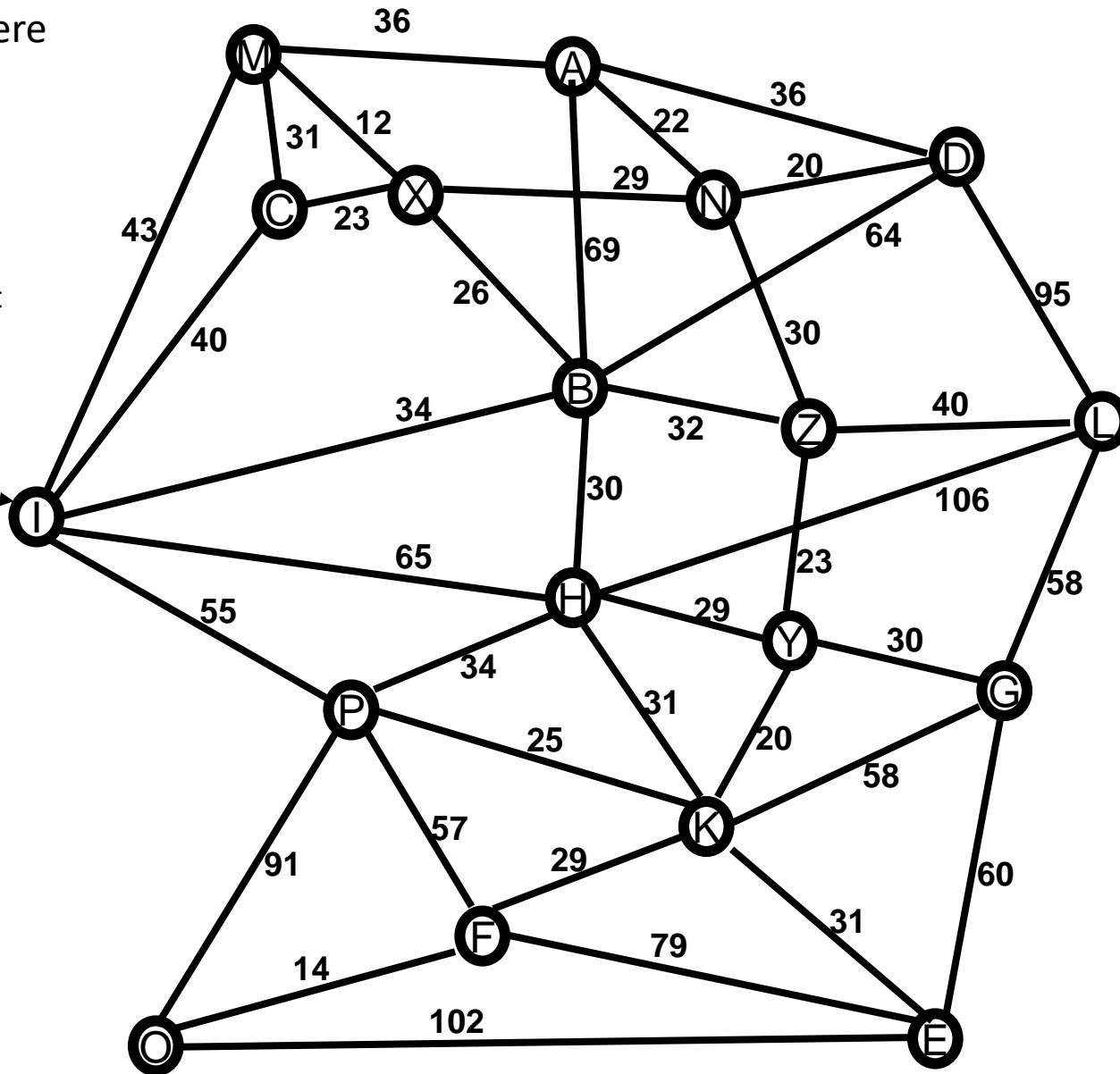
# Algorithmen: Binäre Suche - Beispiel

Suche nach Element **32** in einer sortierten Liste:



Bemerkung: nicht klausurrelevant, ab hier kann die Vorlesung beendet werden

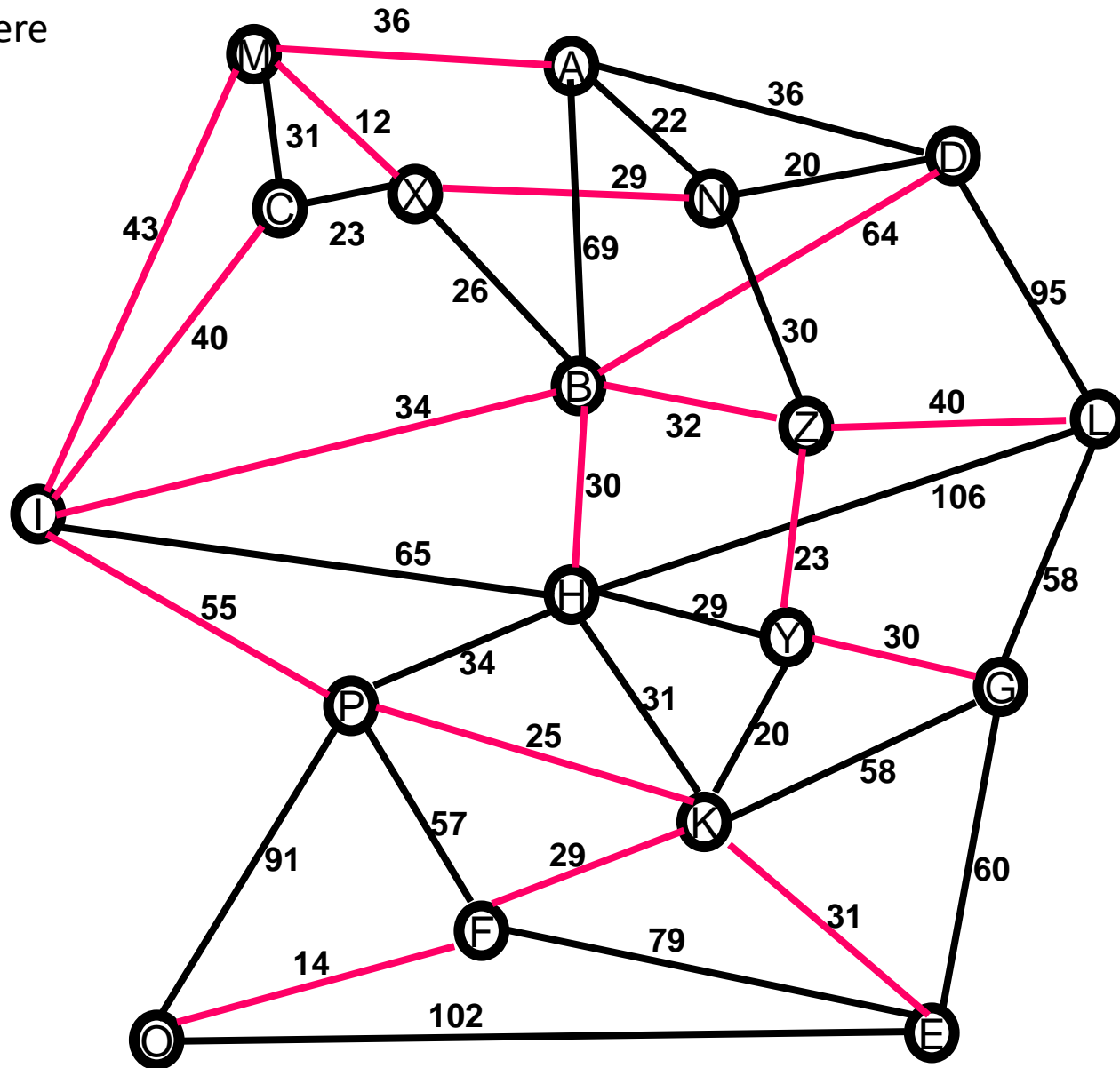
Kürzeste  
Wege von I —  
(Algorithmus  
von Dijkstra)



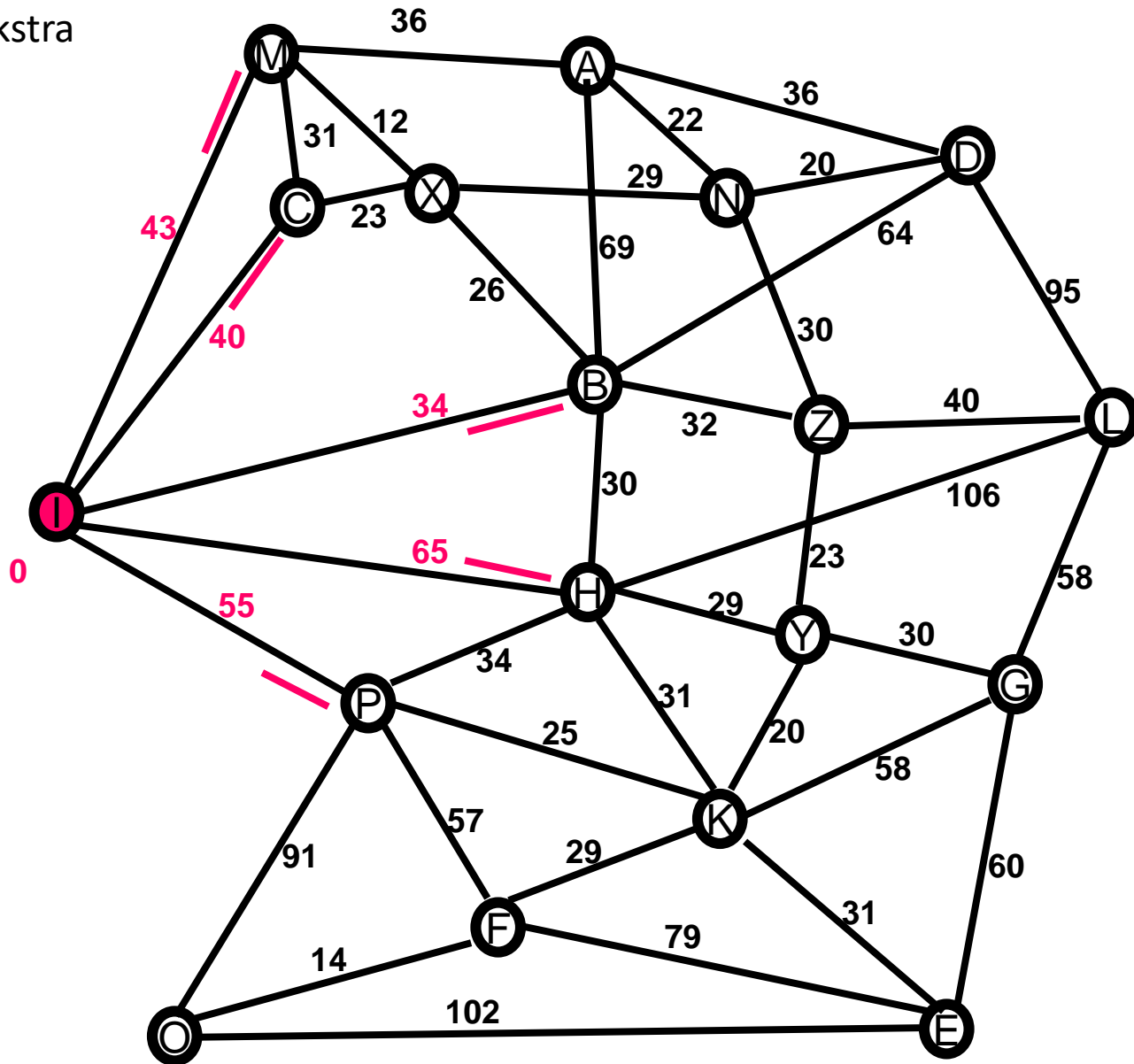


Einschub: andere  
spannende  
Algorithmen

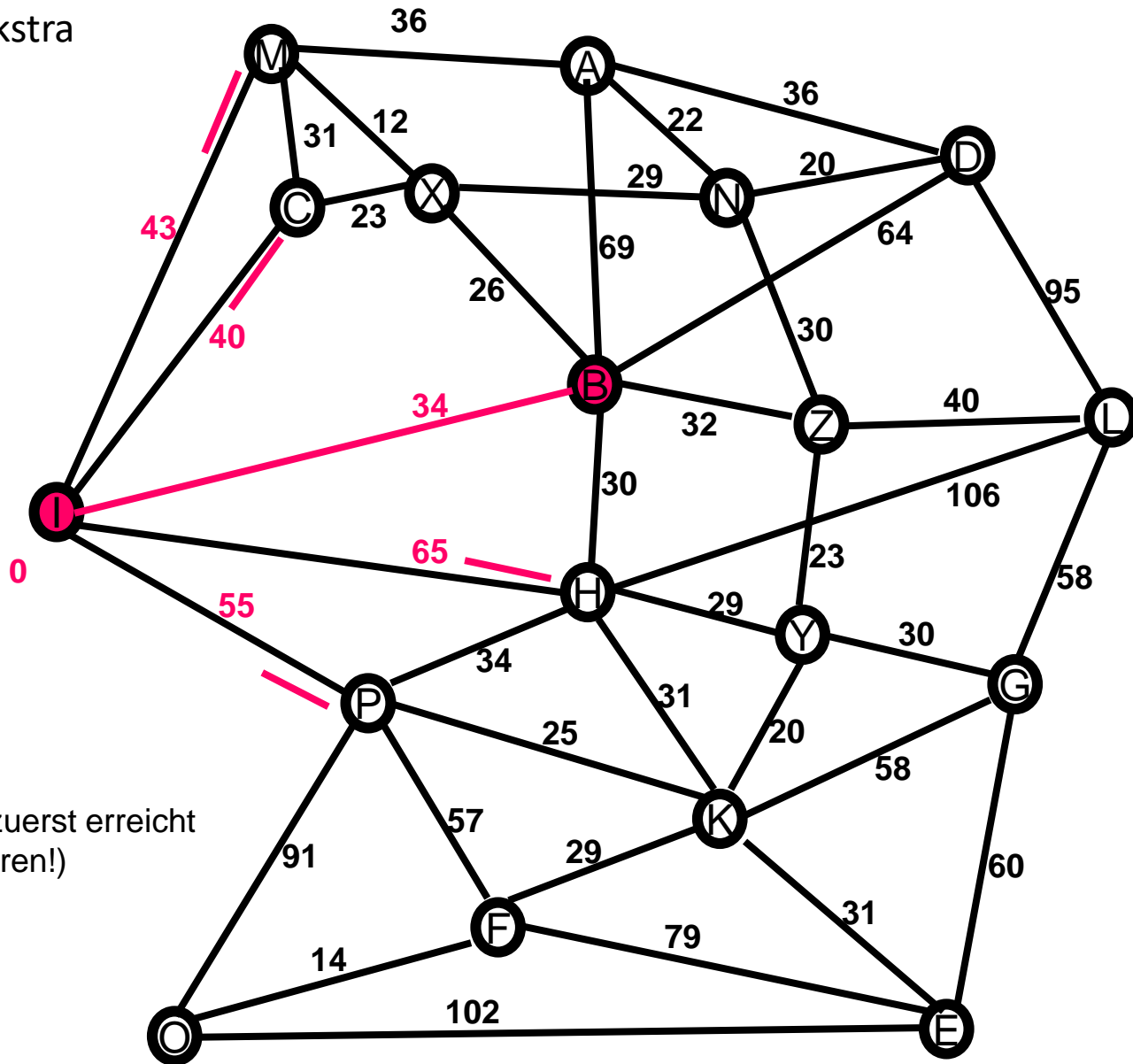
Kürzeste  
Wege von I



I ist Startstadt



## Einschub: Dijkstra Algorithmus

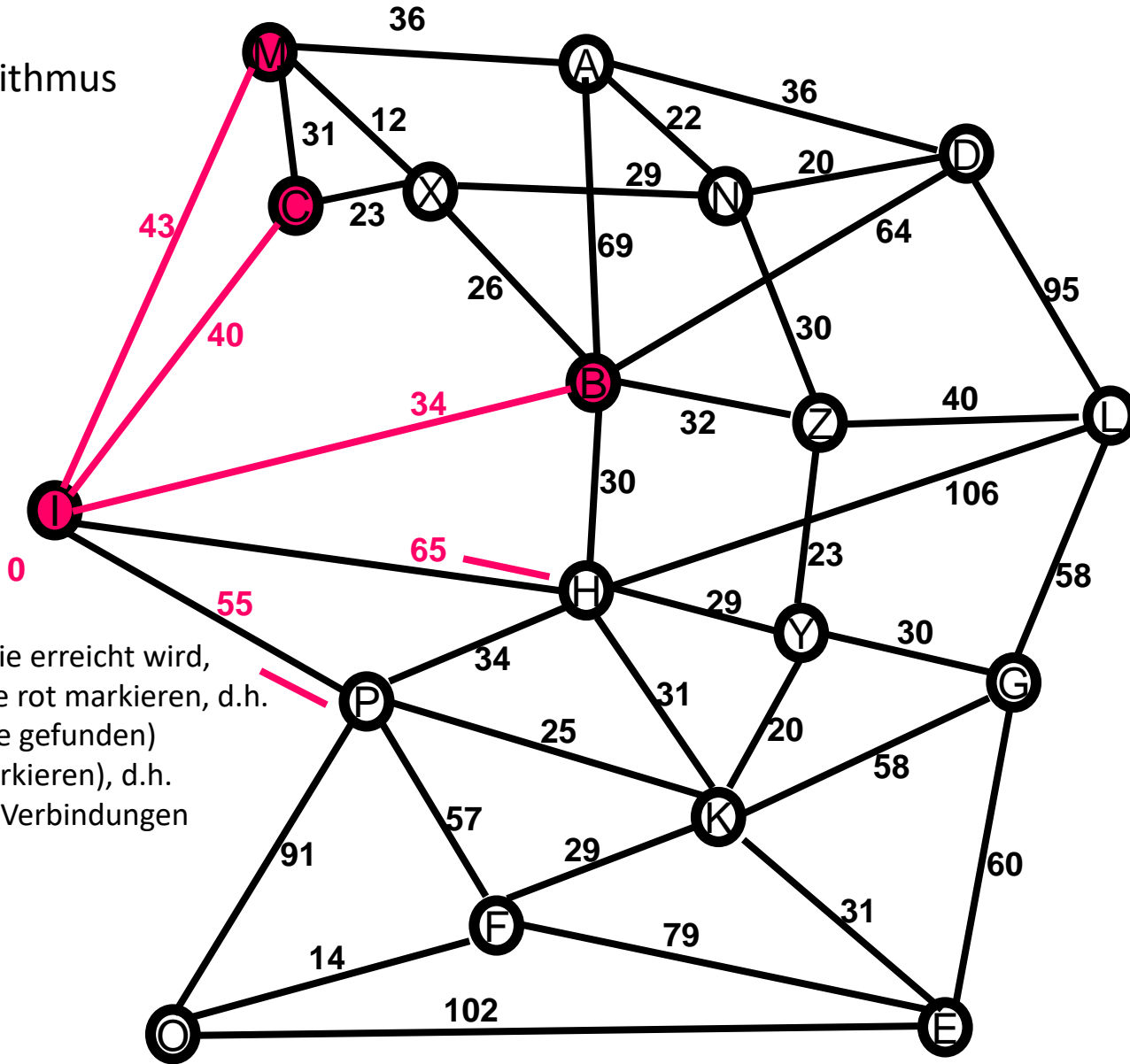


B ist Stadt, die zuerst erreicht  
Wird (rot markieren!)



## Einschub:

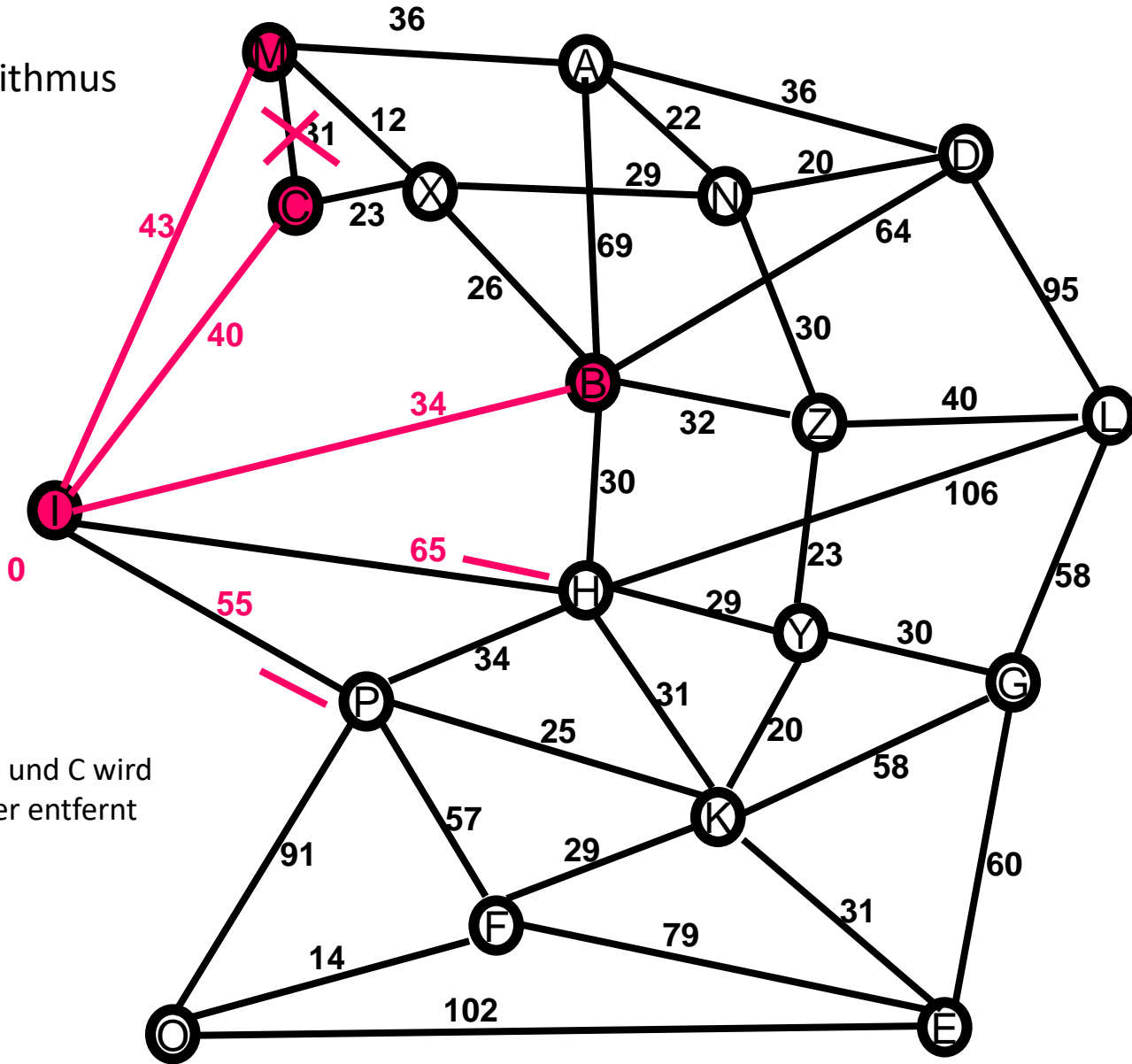
# Dijkstra Algorithmus



nächste Stadt nach B, die erreicht wird,  
ist C (Stadt und Strecke rot markieren, d.h.  
kürzeste Strecke wurde gefunden)  
und danach M (rot markieren), d.h.  
es gibt keine kürzeren Verbindungen  
zu C und M.

## Einschub:

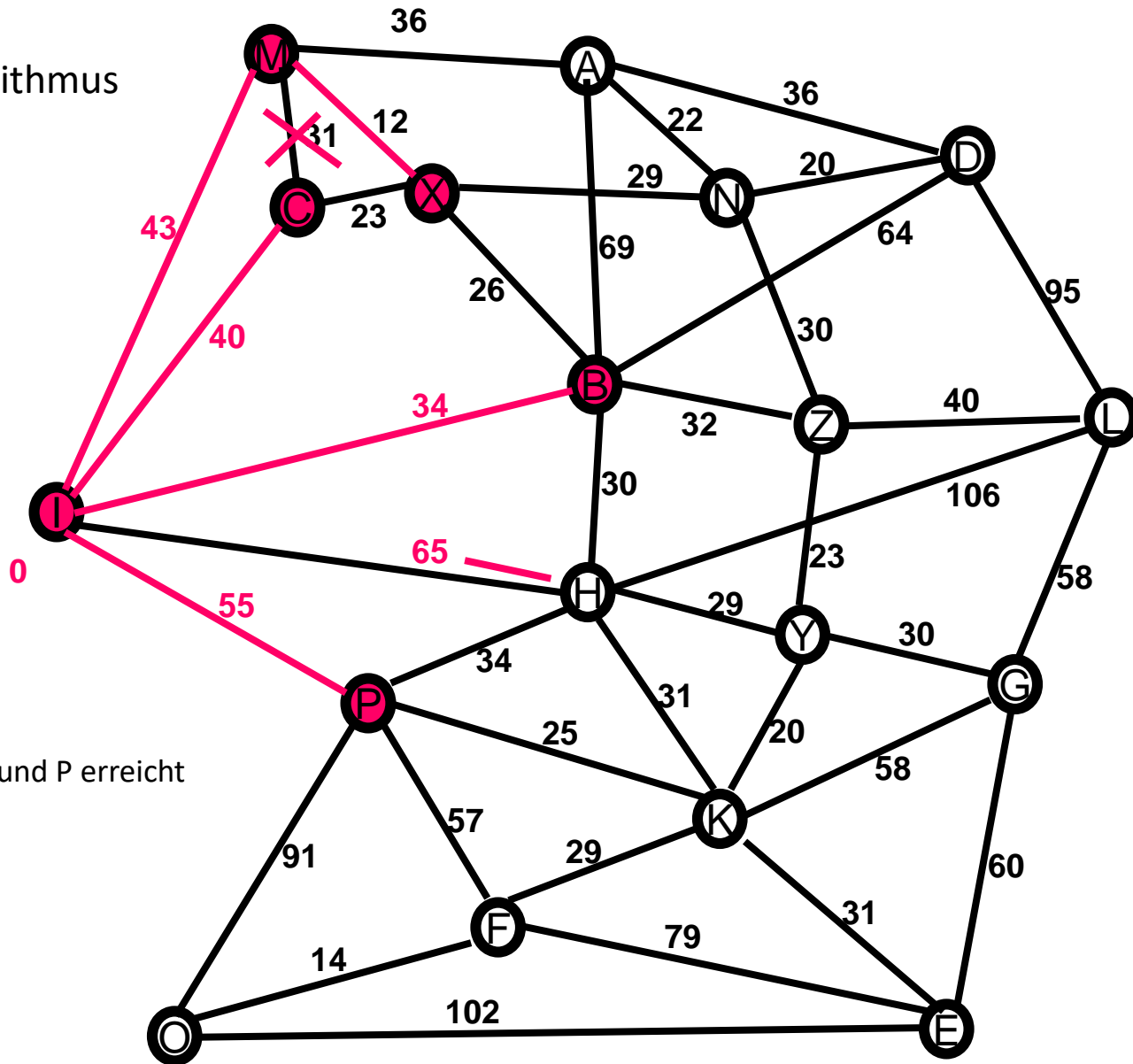
# Dijkstra Algorithmus



Strecke zwischen M und C wird gestrichen, da weiter entfernt

## Einschub:

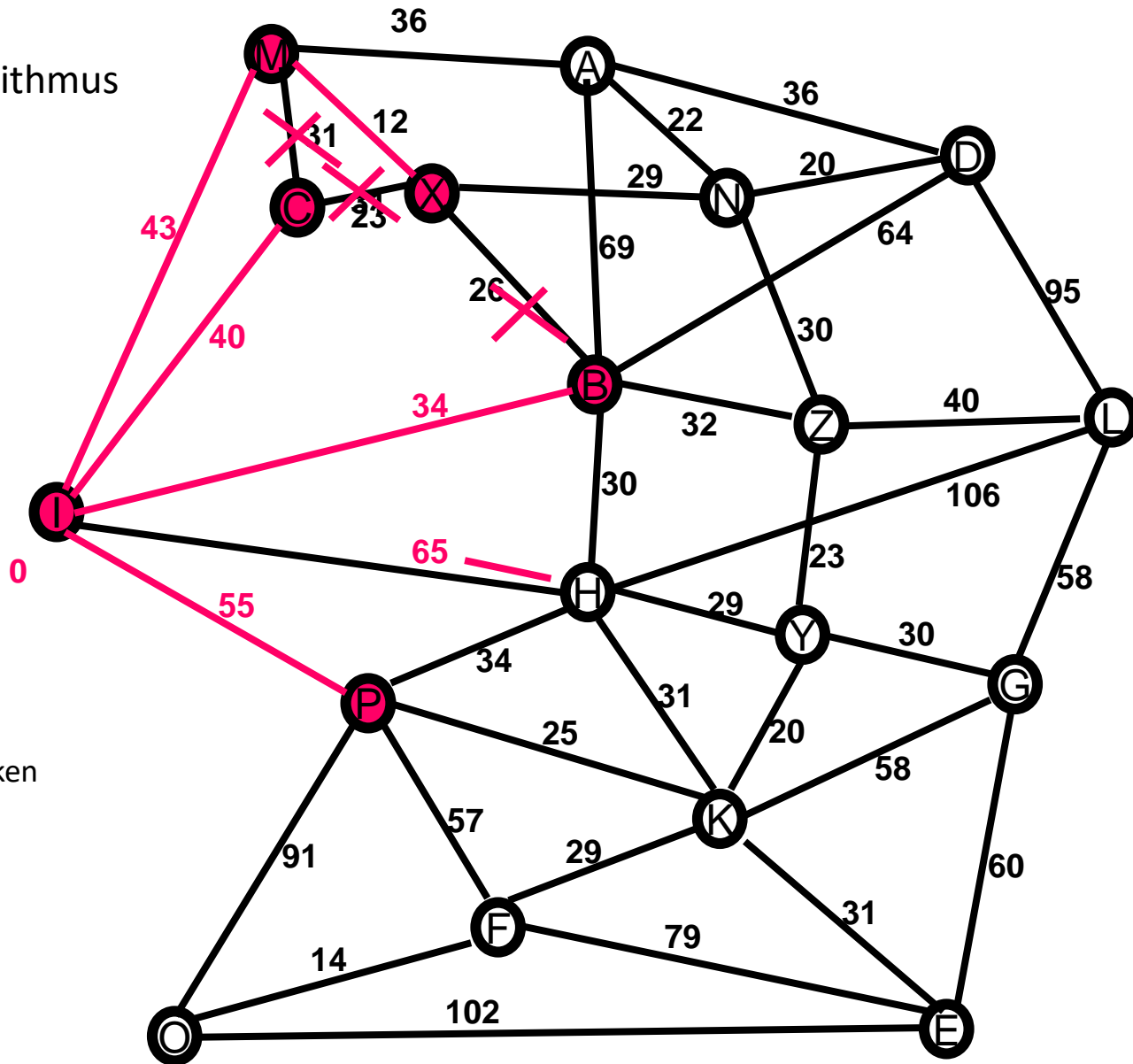
# Dijkstra Algorithmus



Nach 55 km wird X und P erreicht  
(rot markieren)

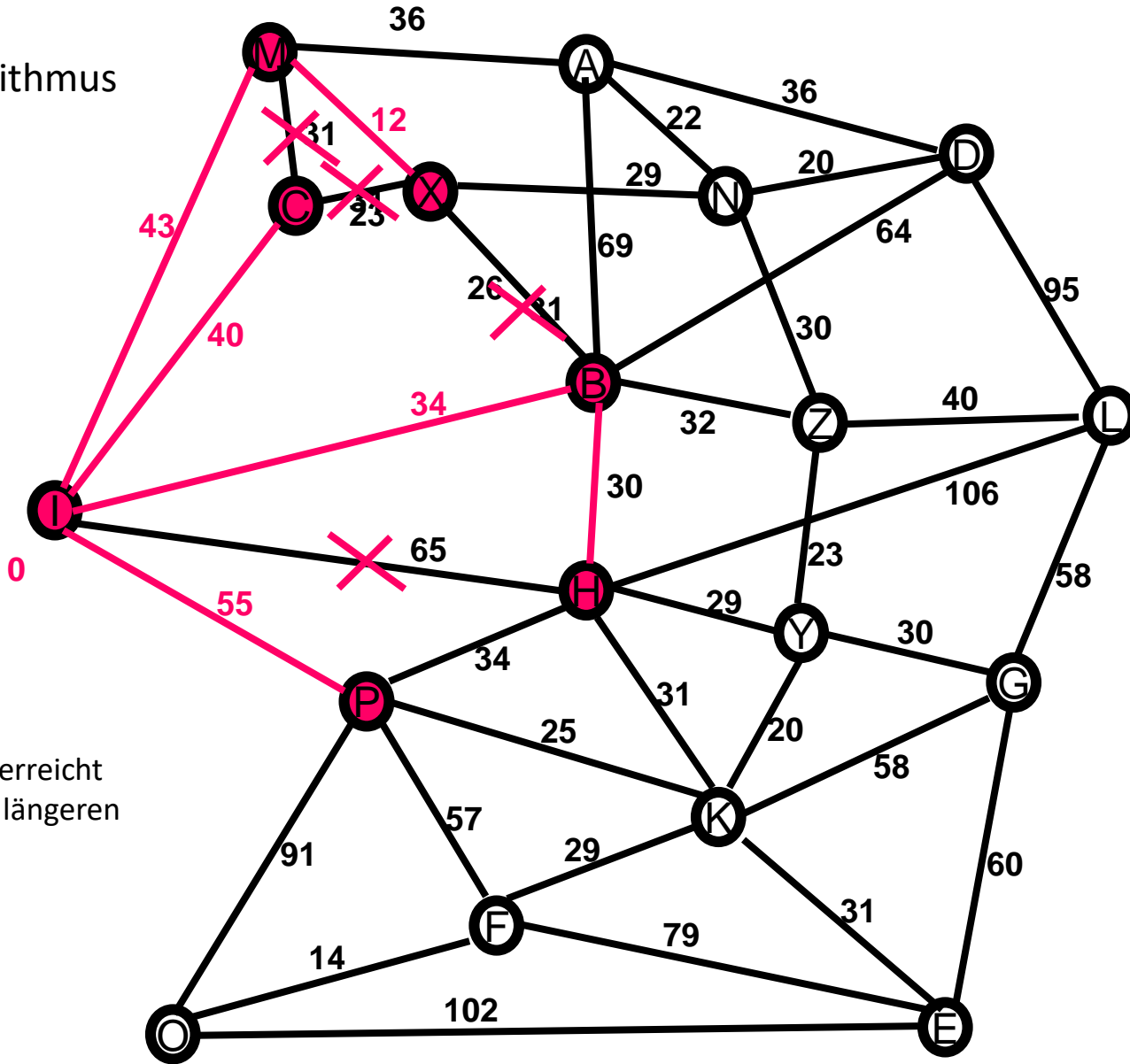
## Einschub:

# Dijkstra Algorithmus



Alle längeren Strecken streichen

# Dijkstra Algorithmus



Nach 64 km wird H erreicht  
(rot markieren, alle längeren  
Strecken streichen)



# Einschub: Algorithmus von Dijkstra

1. Markiere **Startstadt** rot, weise ihr **Kennzahl** 0 zu. Bezeichne diese als **aktuelle Stadt**.
2. Gehe von der **aktuellen Stadt** zu allen direkt erreichbaren **Nachbarstädten**  
Und führe das Folgende für jede Nachbarstadt durch:  
Errechne die **Summe** aus der Kennzahl an der **aktuellen Stadt** und der Länge der Strecke dorthin.
  - Ist die **Nachbarstadt** bereits rot markiert, mache nichts.
  - Hat die **Nachbarstadt** keine **Kennzahl**, weise ihr die **Summe** als **Kennzahl** zu und markiere die Strecke zur **aktuellen Stadt**.
  - Hat die **Nachbarstadt** eine **Kennzahl** kleiner der **Summe**, mache nichts.
  - Hat die **Nachbarstadt** eine **Kennzahl** größer der **Summe**, streiche die dortige **Kennzahl** sowie die Markierung. Weise ihr danach die **Summe** als neue **Kennzahl** zu. Markiere die Strecke zur **aktuellen Stadt**.
3. Betrachte alle Städte, die zwar eine **Kennzahl** haben, aber noch nicht rot markiert sind. Suche die Stadt mit der kleinsten **Kennzahl**.
4. Bezeichne diese als die **aktuelle Stadt**. Weisen mehrere Städte die kleinste **Kennzahl** auf, wähle eine beliebige davon als **aktuelle Stadt**.
5. Markiere die **aktuelle Stadt** rot, zeichne die dort markierte Strecke in rot ein.
6. Falls es noch Städte gibt, die nicht rot markiert sind, weiter bei 2.

# Einschub: Korrektheit von Algorithmen

Der Nachweis der Korrektheit eines Algorithmus bezieht sich immer darauf, was er tun **soll**.

Insofern handelt es sich immer um eine **relative** Korrektheit.

- Verifikation(falscher Begriff) = Bestätigen: Testen, ob aus bestimmten Eingabewerten bestimmte Ausgabewerte berechnet werden
- Echte Korrektheit: Beweis, dass der Algorithmus korrekt ist, ist nur für Miniprogramme möglich bis jetzt (sehr sehr aufwendig)

Testen kann man nur die Anwesenheit von Fehlern, nicht jedoch die Abwesenheit.

# Einschub: statt Korrektheit Robustheit und Zuverlässigkeit

**Robustheit:** sinnvolles Reagieren auf falsche Eingaben (auf alle Fälle nicht das System abstürzen lassen!!)

**Zuverlässigkeit:**

- Wie oft treten Fehler auf?
- Wie oft tritt ein und derselbe Fehler auf?
- Wie viele unterschiedliche Fehler gibt es?
- Führt der Fehler zum Systemabsturz oder zum Verlust von Daten?

# Literatur:

- Thomas Corman, Charles Leiserson, Ronald Rivest, Clifford Stein: „Algorithms“
- T. Ottmann, P. Widmayer: „Algorithmen und Datenstrukturen“
- Uwe Schöning: „Algorithmik“