

A black and white close-up photograph of a computer keyboard. The focus is on several keys, including one with the letter 'A' and another with the letter 'S'. The keys are slightly blurred, creating a sense of depth. A vertical line runs down the right side of the image, and a horizontal line runs across the bottom, intersecting at the start of the text box.

***Formelsammlung  
Informatik***

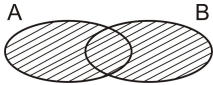
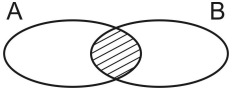
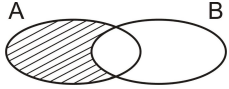
## 1. Inhaltsverzeichnis

1.	Inhaltsverzeichnis .....	2
2.	Informatik .....	3
2.1.	Mengenlehre .....	3
2.1.1.	Mengenoperationen .....	3
2.2.	Relationen – Spezielle Klassen von Relationen .....	4
2.3.	Abbildungen .....	7
2.3.1.	Rechts eindeutige Relation .....	7
2.3.2.	Links eindeutige Relation .....	7
2.3.3.	Links vollständige Relation .....	7
2.3.4.	Rechts vollständige Relation .....	8
2.3.5.	Totale Abbildung .....	8
2.3.6.	Surjektive Abbildung .....	8
2.3.7.	Injektive Abbildung .....	8
2.3.8.	Surjektive Abbildung .....	9
2.4.	Algorithmen und Datenstrukturen .....	10
2.4.1.	Rekursion .....	10
2.4.2.	Beispiele für Rekursion .....	10
2.4.3.	Vollständige Induktion .....	11
2.4.4.	Sortieralgorithmen .....	12
2.4.5.	AVL-Baum .....	15
2.4.6.	Vielwegbaum .....	16
2.5.	Aussagenlogik .....	17
2.5.1.	Grundbegriffe .....	17
2.5.2.	Implikation .....	17
2.5.3.	Beispiele für Tautologien .....	17
2.5.4.	Äquivalenz von Formeln .....	17
2.5.5.	Allgemeine Äquivalenzen .....	17
2.5.6.	Umformungsregeln .....	18
2.5.7.	Konjunktive Normalform .....	18
2.5.8.	Resolventenprinzip .....	18
2.5.9.	Hornformeln .....	19
2.5.10.	Markierungsalgorithmus für Hornformeln .....	20
2.6.	Graphentheorie .....	21
2.6.1.	Definitionen .....	21
2.6.2.	Eulerscher Weg .....	22
2.6.3.	Eulerscher Graph .....	22
2.7.	Graph kürzester Weg .....	23
2.7.1.	Flüsse in Netzwerken, Ermittlung maximaler Fluss .....	24
2.8.	Automatentheorie .....	25
2.8.1.	Grundbegriffe .....	25

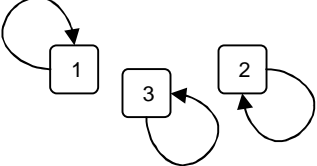
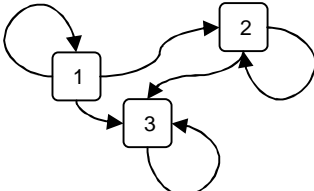
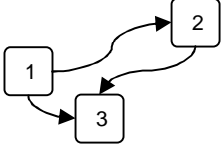
## 2. Informatik

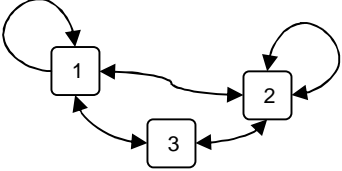
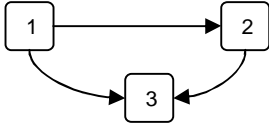
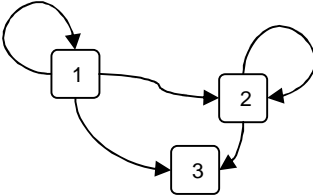
### 2.1. Mengenlehre

#### 2.1.1. Mengenoperationen

<b>Vereinigungsmenge</b> $A \cup B$	
<b>Schnittmenge</b> $A \cap B$ Alle Elemente die zu A und gleichzeitig zu B gehören. $A \cap B = \{x \mid x \in A \wedge x \in B\}$	
<b>Differenzmenge</b> $A \setminus B$	
<b>Produktmenge</b> $A \times B$	<b>Beispiel:</b> $A = \{a; b; c\}$ $B = \{u; v\}$ $A \times B = \{(a; u); (a; v); (b; u); (b; v); (c; u); (c; v)\}$

## 2.2. Relationen – Spezielle Klassen von Relationen

<b>Identische Relation</b>	$I = \{(x, x) \mid x \in M\}$ - jedes $x$ steht nur in Relation zu $x$ (sich selbst)																
	<b>Matrixdarstellung:</b> <table border="1" data-bbox="508 284 978 384"> <thead> <tr> <th></th> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <th>1</th> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <th>2</th> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <th>3</th> <td>0</td> <td>0</td> <td>1</td> </tr> </tbody> </table> Nur Hauptdiagonale ist belegt.		1	2	3	1	1	0	0	2	0	1	0	3	0	0	1
	1	2	3														
1	1	0	0														
2	0	1	0														
3	0	0	1														
<b>Reflexive Relation</b>	$I \subseteq R \quad + \quad x \in M \text{ ist } (x, x) \in R$ - jedes Element steht in Relation zu sich selbst - zusätzlich können andere Elemente verknüpft sein																
	<b>Matrixdarstellung:</b> <table border="1" data-bbox="508 596 978 697"> <thead> <tr> <th></th> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <th>1</th> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <th>2</th> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <th>3</th> <td>0</td> <td>0</td> <td>1</td> </tr> </tbody> </table>		1	2	3	1	1	1	1	2	0	1	1	3	0	0	1
	1	2	3														
1	1	1	1														
2	0	1	1														
3	0	0	1														
<b>Irreflexive Relation</b>	- kein Paar mit gleichen Elementen gehört zur Relation - für alle $a$ gilt $\neg(aRa)$																
	<b>Matrixdarstellung:</b> <table border="1" data-bbox="508 927 978 1027"> <thead> <tr> <th></th> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <th>1</th> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <th>2</th> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <th>3</th> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table> Hauptdiagonale ist nicht belegt		1	2	3	1	0	1	1	2	0	0	1	3	0	0	0
	1	2	3														
1	0	1	1														
2	0	0	1														
3	0	0	0														

<b>Symmetrische Relation</b>	$(x, y) \in R \text{ folgt } (y, x) \in R \quad R = R^{-1}$ - zu jedem Pfeil gibt es einen Umkehrpfeil																
	<i>Matrixdarstellung:</i> <table border="1" data-bbox="505 252 692 352"> <thead> <tr> <th></th> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <th>1</th> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <th>2</th> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <th>3</th> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table> Spiegelung an Hauptdiagonale möglich, Diagonale muss keine 1er enthalten, da bei Spiegelung nicht relevant.		1	2	3	1	1	1	1	2	1	1	1	3	1	1	0
	1	2	3														
1	1	1	1														
2	1	1	1														
3	1	1	0														
<b>Asymmetrische Relation</b>	$(x, y) \in R \text{ folgt } (y, x) \in R / R \cap R^{-1} = \emptyset$ - weder Umkehrpfeile noch Ringpfeile - gehört $1 \rightarrow 2$ zur Relation, gehört $2 \rightarrow 1$ nicht dazu																
	<i>Matrixdarstellung:</i> <table border="1" data-bbox="505 590 692 691"> <thead> <tr> <th></th> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <th>1</th> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <th>2</th> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <th>3</th> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table> Kein Element ist mit sich selbst verknüpft und kein Element ist rückverknüpft Irrreflexiv und antisymmetrisch		1	2	3	1	0	1	1	2	0	0	1	3	0	0	0
	1	2	3														
1	0	1	1														
2	0	0	1														
3	0	0	0														
<b>Antisymmetrische Relation</b>	$(x, y) \in R \text{ u. } (y, x) \in R \text{ folgt } x = y$ - keine Umkehrpfeile, aber Ringschleifen sind erlaubt																
	<i>Matrixdarstellung:</i> <table border="1" data-bbox="505 917 692 1018"> <thead> <tr> <th></th> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <th>1</th> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <th>2</th> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <th>3</th> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table> Elemente dürfen mit sich selbst verknüpft sein, aber keine Rückverknüpfungen zwischen Elementen		1	2	3	1	1	1	1	2	0	1	1	3	0	0	0
	1	2	3														
1	1	1	1														
2	0	1	1														
3	0	0	0														

<b>Transitive Relation</b>	$(x, y) \in R \quad (y, z) \in R \quad \text{folgt} \quad (x, z) \in R$ - zu je 2 Pfeilen gibt es einen Überbrückungspfeil Matrixdarstellung:																
	<table border="1" style="display: inline-table;"> <thead> <tr> <th></th> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <th>1</th> <td>1</td> <td>1</td> <td>1</td> </tr> <tr> <th>2</th> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <th>3</th> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table> Wann immer 2 Pfeile aufeinander folgen gibt es einen Pfeil der diese überbrückt		1	2	3	1	1	1	1	2	0	1	1	3	0	0	0
	1	2	3														
1	1	1	1														
2	0	1	1														
3	0	0	0														
Bsp.: $1 \rightarrow 3$ und $1 \rightarrow 2 \rightarrow 3$																	

<b>Aquivalenzrelation</b>	- ist reflexiv, symmetrisch und transitiv
---------------------------	---

<b>Ordnung</b>	- reflexiv, antisymmetrisch und transitive Relation von R in M
<p style="text-align: center;"><b>Totale Ordnung</b></p> <p style="text-align: center;">Beispiel „kleiner gleich“</p>	<p style="text-align: center;"><b>Partielle Ordnung:</b></p> <p style="text-align: center;">Beispiel „teilt“</p>
Darstellung von Beziehungen wie z.B. langsamer, schneller, dümmer/klüger können dargestellt werden.	

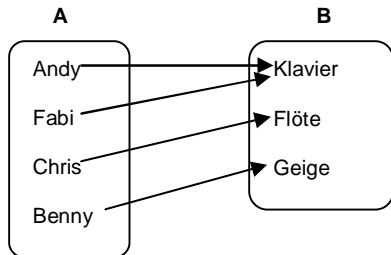
<b>Totale / partielle Ordnung</b>	2 Elemente von M sind miteinander vergleichbar, <b>ansonsten</b> partielle Teilordnung
-----------------------------------	--

<b>Strikte Ordnung</b>	- eine asymmetrische, transitive Relation in M
<p style="text-align: center;">Beispiel „alkoholreicher als“</p>	

## 2.3. Abbildungen

### 2.3.1. Rechts eindeutige Relation

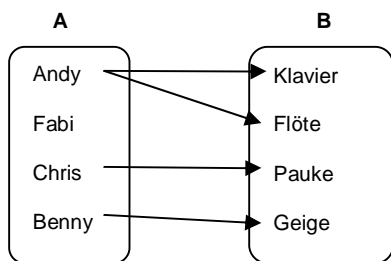
- jedem Element von A (links) wird höchstens ein Element der rechten Menge B zugeordnet



- die Pfeile nach rechts sind eindeutig

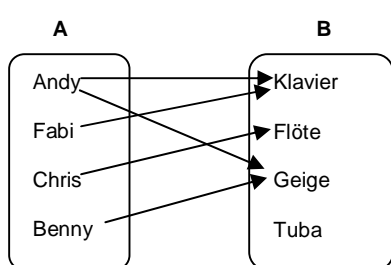
### 2.3.2. Links eindeutige Relation

- jedes Element von rechts kommt höchstens einmal als Bild eines Elements der linken Menge A vor

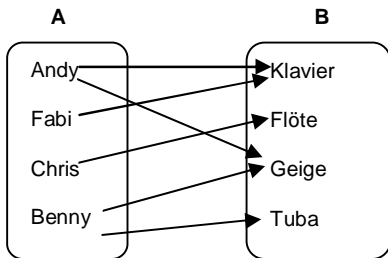


- auf jedes Element von B zeigt genau ein Pfeil

### 2.3.3. Links vollständige Relation



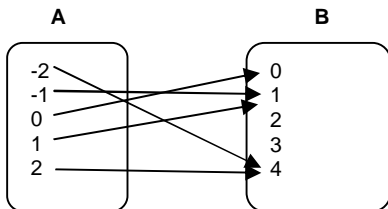
- jedem Element von A (links) wird mindestens ein Element aus B zugeordnet

**2.3.4. Rechts vollständige Relation**

- jedes Element kommt mindestens einmal als Bild vor

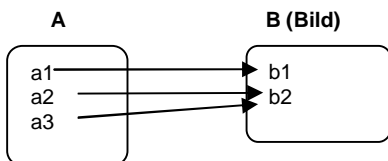
**2.3.5. Totale Abbildung**

Beispiel:  $x^2$

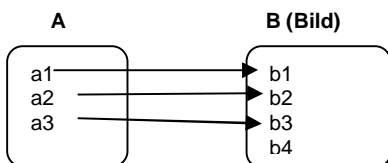


- rechts eindeutig und links vollständig
- jedes Element v. A darf mit maximal einem Element aus B in Beziehung stehen

Definitionsbereich: Werte aus A  
Wertebereich: Werte aus B

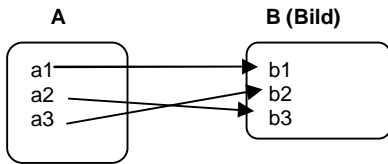
**2.3.6. Surjektive Abbildung**

- jedes Element der Zielmenge kommt als Bild vor (zu jedem Element von B führt ein Pfeil)

**2.3.7. Injektive Abbildung**

- jedes Element der Zielmenge kommt höchstens einmal als Bild vor



**2.3.8. Surjektive Abbildung**

- ist injektiv und surjektiv
- jedes Element der Zielmenge kommt genau einmal als Bild vor

## 2.4. Algorithmen und Datenstrukturen

Statische Finitheit: Algorithmus darf nicht unendlich groß sein (Quelltext hat ein Ende)

Dynamische Finitheit: benötigter Speicher darf nicht unendlich groß sein

### 2.4.1. Rekursion

Allgemeiner Ansatz:

```
int sum(int zahl) {
    int sum = 0;
    if (zahl==1) {
        sum=1;
        return sum;
    }
    else {
        sum=zahl + sum(zahl-1);
        return sum;
    }
}
```

| einfacher Fall  
|  
|  
| rekursiver Aufruf

### 2.4.2. Beispiele für Rekursion

Summe der ersten n natürlichen Zahlen	geschlossene Formel: $\left(\frac{n*(n+1)}{2}\right)$ $sum(n) = \begin{cases} 0 & // n = 0 \\ sum(n-1) + 1 & // n \geq 1 \end{cases}$
Fakultät	$fak(n) = \begin{cases} 1 & // n \leq 1 \\ n * fak(n-1) + 1 & // n \geq 1 \end{cases}$ Näherungsformel: $n! \approx \sqrt{2\pi n} * \left(\frac{n}{e}\right)^n$
Euklidischer Algorithmus	$ggT(x, y) = \begin{cases} y = 0 \Rightarrow x & // x \text{ mod } y = 0 \\ ggT(y, x \text{ mod } y) \end{cases}$
Summe der ersten n ungeraden Zahlen	geschlossene Formel: $n^2$ $quadrat(n) = \begin{cases} 1 & // n = 1 \\ (2n-1) + quadrat(n-1) \end{cases}$
Summe der ersten n geraden Zahlen	$sumgerad(n) = \begin{cases} 2 & // n = 1 \\ (2n) + sumgerad(n-1) \end{cases}$
Fibonacci	$fib(n) = \begin{cases} f_0 = 0 & f_1 = 1 & // n < 2 \\ fib(n-1) + fib(n-2) & // n \geq 2 \end{cases}$

**2.4.3. Vollständige Induktion**

Summe der ersten n Zahlen:  $\sum_{i=1}^n i = \frac{n \cdot (n+1)}{2}$

Induktionsanfang	$n=1 \quad \frac{1 \cdot (1+1)}{2} = 1$
Induktionsbehauptung	$\sum_{i=1}^{n+1} i = \frac{(n+1) \cdot n(n+2)}{2}$
Induktionsbeweis	$\sum_{i=1}^{n+1} i = 1 + 2 + 3 + \dots + n + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$ $(1 + 2 + 3 + \dots + n \rightarrow \frac{n \cdot (n+1)}{2})$ $\frac{n \cdot (n+1)}{2} + (n+1) = \frac{(n+1) \cdot (n+2)}{2}$ $\frac{n^2 + n}{2} + (n+1) = \frac{n^2 + 3n + 2}{2}$ $\frac{n^2 + n + 2n + 2}{2} = \frac{n^2 + 3n + 2}{2}$ $\frac{n^2 + 3n + 2}{2} = \frac{n^2 + 3n + 2}{2}$ <hr style="border: none; border-top: 1px solid black; width: 100%;"/> <p><b>q.e.d!</b></p>

- wenn P(1) wahr ist und auch P(n+1) wahr ist, so ist P(n) wahr für alle n!

## 2.4.4. Sortieralgorithmen

### 2.4.4.1. Straight Insertion Sort

<b>Beispiel:</b> 5, 3, 2, <u>8</u> , 9   8 <hr/> 5, 3, <u>2</u> , 8, 9   2 → 2 < 8 → 2 in Marke <hr/> 5, <u>3</u> , 2, 8, 9   2 → 3 > 2 Tausch, 3 in Marke <hr/> <u>5</u> , 2, 3, 8, 9   3 → 5 > 2, 5 in Marke <hr/> 2, 5, 3, 8, 9   3 → 5 > 3 → Tausch <hr/> 2, 3, 5, 8, 9	– fange beim letzten Element an → in Marke schreiben – vgl. 8 mit 9 → kein Tausch notwendig  <b>Algorithmus:</b> – es wird von hinten im Array ausgegangen, beim vorletzten Element wird angefangen – Element wird in Marke geschrieben – Element wird mit den nachfolgenden verglichen, solange Element größer als Nachfolger ist, wird getauscht
---	--

#### Quelltext:

```
int x = 0;
int j = 0;

for (int i = arValues.Length - 3; i >= 0; i--)
{
    x = arValues[i];
    arValues[arValues.Length-1] = x;
    j = i + 1;

    while (x > arValues[j])
    {
        arValues[j - 1] = arValues[j];
        j++;
    }

    arValues[j - 1] = x;
}
```

#### Komplexität:

Durchlauf: 1 → 2 Vergleiche  
 2 → 3 Vergleiche  
 3 → 4 Vergleiche

daraus folgt:  $Vergleiche_{\max} = \frac{n \cdot (n+1)}{2} - 1$        $Tausch = \frac{n \cdot (n+1)}{2}$

### 2.4.4.2. Bubble Sort

- es werden immer 2 benachbarte Elemente verglichen
- ist  $x > x+1 \rightarrow$  tauschen
- nach jedem Durchlauf steht das größte Element oben

<b>Beispiel:</b>	<b>Komplexität:</b>
<pre> 5 3 → Vergleich     → Grenze äußere Schleife  5 3   2 8 1   3 5 2   8 1   3 2 5 8   1   3 2 5 8 1   3 2   5 1   8 2 3 5 1   8 2 3 5 1   8 2 3   1 5 8 2 3 1   5 8 2 1   3 5 8 1 2 3 5 8 </pre>	<p>Vergleiche: <math>\frac{n(n-1)}{2}</math></p> <p><math>BC: 0</math></p> <p>Zuweisung: <math>WC: 3 * \frac{n(n-1)}{2}</math></p>

**2.4.4.3. Heap Sort**

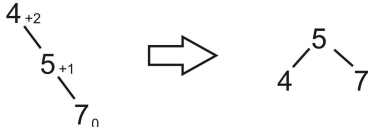
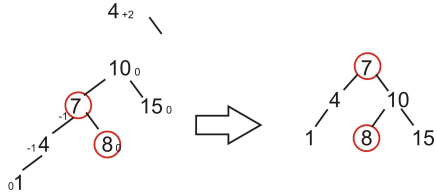
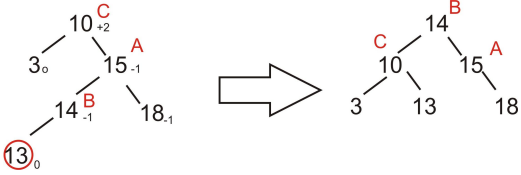
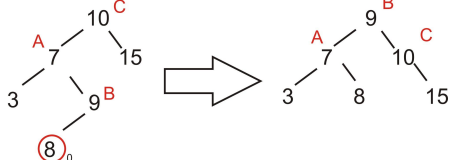
- Aufbau des Arrays als Binärbaum
- Beispiel: 17, 31, 05, 59, 13, 41, 43, 67, 11, 23
- letzte Hälfte des Arrays sind die Blätter
- Zugriff auf Kindknoten:  $\text{arrayIndex} * 2$  (links)  
 $\text{arrayIndex} * 2 - 1$  (rechts)

(1 basierte Arrays!)

<p><u>1. Schritt: Aufbau des Heaps</u></p>	<ul style="list-style-type: none"> <li>- Anfang bei Array-Länge / 2 (Mitte des Arrays)</li> <li>- Beginn bei <math>\text{arrayIndex}[5]</math></li> <li>- größeres Kind wird mit Vater getauscht</li> <li>- <math>[5] \rightarrow 23 &gt; 13 \rightarrow</math> Tausch</li> <li>- <math>[4] \rightarrow 67 &gt; 59 \rightarrow</math> Tausch</li> </ul>
	<ul style="list-style-type: none"> <li>- <math>[3] \rightarrow 43 &gt; 5 \rightarrow</math> Tausch</li> <li>- <math>[2] \rightarrow 67 &gt; 31 \rightarrow</math> Tausch</li> <li>- <math>\rightarrow 31</math> muss weiter versickert werden!</li> <li>- <math>\rightarrow 59 &gt; 31 \rightarrow</math> Tausch</li> </ul>
<p><u>2. Schritt: Abbau des Heaps</u></p>	<ul style="list-style-type: none"> <li>- nach Abschluss steht das größte Element oben im Heap</li> <li>- größtes Element wird genommen und mit letztem Array-Element getauscht</li> <li>- <math>\rightarrow 67</math> tausch 13</li> <li>- das letzte Element ist nun sortiert, und wird nicht mehr berücksichtigt bei den weiteren Schritten</li> <li>- 13 steht oben und muss neu versickert werden</li> <li>- Vorgang wird wiederholt bis Heap komplett abgebaut ist</li> </ul>
<p>Array: ..., 23, 31, 41, 43, 59, 67</p>	

### 2.4.5. AVL-Baum

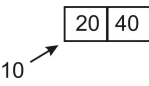
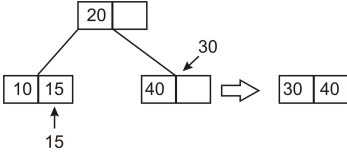
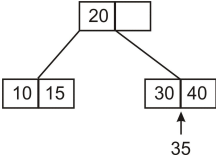
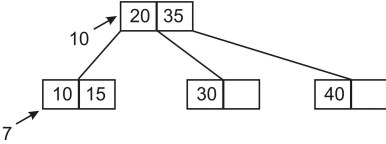
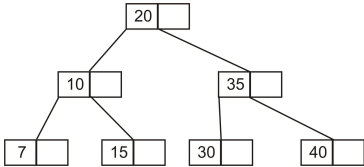
- maximaler Höhenunterschied 1 zwischen Knoten
- wenn Bedingung verletzt  $\rightarrow$  Umbau

Fall 1:	<p>Beispiel 1:</p>  <p>Beispiel 2:</p> 
Fall 2:	<p>Beispiel 1:</p>  <p>Beispiel 2:</p> 

### 2.4.6. Vielwegbaum

- mehrere Elemente pro Stufe
- Baum wird von unten nach oben aufgebaut

Beispiel: 20, 40, 10, 30, 15, 35, 7

	<ul style="list-style-type: none"> <li>- 20 u. 40 werden eingefügt</li> <li>- für 10 Knoten zu klein</li> <li>- Knoten wird aufgeteilt in 2 Knoten, mittleres Element wandert nach oben</li> </ul>
	<ul style="list-style-type: none"> <li>- ist im Knoten noch Platz, dann sortiert eintragen</li> </ul>
	<ul style="list-style-type: none"> <li>- Knoten platz, 35 wandert Ebene nach oben</li> <li>- Knoten 30, 40 wird in Elemente aufgeteilt</li> </ul>
	<ul style="list-style-type: none"> <li>- Knoten platz durch 7, die 10 wandert nach oben, dadurch platz auch dieser Knoten!</li> </ul>
	<ul style="list-style-type: none"> <li>- fertig!</li> </ul>



## 2.5. Aussagenlogik

### 2.5.1. Grundbegriffe

Aussagensymbole:  $A, B, A_1, A_2, \dots$   
 Junktoren:  $\wedge$  (und),  $\vee$  (oder),  $\neg$  (nicht),  $\rightarrow$  (Implikation),  $\leftrightarrow$  (Äquivalenz)  
 Tautologie: immer wahr  
 Kontradiktion: immer falsch

### 2.5.2. Implikation

Beispiel: „Wenn es regnet wird die Straße nass.“

Prämisse: „es regnet“

Konklusion: „die Straße wird nass“

Ist die Prämisse falsch, so ist die Aussage dennoch wahr,  $\mathbf{a \rightarrow 1}$

da die Straße auch aus anderen Gründen nass sein kann.  $\mathbf{0 \rightarrow a}$

### 2.5.3. Beispiele für Tautologien

- (1)  $(p \wedge q) \rightarrow p$  oder  $p \rightarrow (p \vee q)$       (6)  $((p \rightarrow q) \wedge (q \rightarrow p)) \leftrightarrow (p \leftrightarrow q)$   
 (2)  $(q \rightarrow p) \vee (\neg q \rightarrow p)$   
 (3)  $(p \rightarrow q) \leftrightarrow (\neg p \vee q)$   
 (4)  $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$   
 (5)  $(p \wedge (p \rightarrow q)) \rightarrow q$

### 2.5.4. Äquivalenz von Formeln

2 Formeln sind logisch äquivalent ( $p \equiv q$ ), wenn die Formel  $(p \leftrightarrow q)$  eine Tautologie ist.

### 2.5.5. Allgemeine Äquivalenzen

Kommutativität	$(p \wedge q)$ $(p \vee q)$	$\equiv$ $\equiv$	$(q \wedge p)$ $(q \vee p)$
Assoziativität	$(p \wedge (q \wedge r))$ $(p \vee (q \vee r))$	$\equiv$ $\equiv$	$((p \wedge q) \wedge r)$ $((p \vee q) \vee r)$
Distributivität	$(p \wedge (q \vee r))$ $(p \vee (q \wedge r))$	$\equiv$ $\equiv$	$((p \wedge q) \vee (p \wedge r))$ $((p \vee q) \wedge (p \vee r))$
Idempotenz	$(p \wedge p)$ $(p \vee p)$	$\equiv$ $\equiv$	$p$ $p$
Doppelnegation	$(\neg(\neg p))$	$\equiv$	$p$
de Morgan	$(\neg(p \wedge q))$ $(\neg(p \vee q))$	$\equiv$ $\equiv$	$((\neg p) \vee (\neg q))$ $((\neg p) \wedge (\neg q))$
Tautologieregeln	Falls $q$ eine Tautologie (wahr) ist, gilt: $(p \wedge q)$ $(p \vee q)$	$\equiv$ $\equiv$	$p$ $q$ $\equiv$ wahr

Kontradiktionsregeln	Falls q eine Kontradiktion (falsch) ist, gilt:		
	$(p \wedge q)$	$\equiv$	$q$ $\equiv$ falsch
	$(p \vee q)$	$\equiv$	$p$
Absorbtionsgesetze	$\neg A \vee (A \wedge B)$	$\equiv$	$\neg A \vee B$
	$\neg A \wedge (A \vee B)$	$\equiv$	$\neg A \wedge B$

### 2.5.6. Umformungsregeln

$$\begin{array}{llll}
 (A \wedge B) \leftrightarrow & (\neg(\neg A \vee \neg B)) & (A \rightarrow B) \leftrightarrow & (\neg A \vee B) \\
 (A \vee B) \leftrightarrow & (\neg(\neg A \wedge \neg B)) & (A \leftrightarrow B) \leftrightarrow & (A \rightarrow B) \wedge (B \rightarrow A) \\
 & & & (\neg A \vee B) \wedge (\neg B \vee A)
 \end{array}$$

### 2.5.7. Konjunktive Normalform

- Konjunktion (logische UND-Verknüpfung) von Disjunktionstermen
- Disjunktionsterme sind Disjunktionen (ODER-Verknüpfung) von Literalen (negierte oder nicht negierte Variablen)
- jede Formel lässt sich in eine KNF umwandeln

### 2.5.8. Resolventenprinzip

Grundidee: jede Formel kann in eine äquivalente Konjunktion von Klauseln (=Disjunktionsterm) transformiert werden.

$$\begin{array}{l}
 \text{Mengenschreibweise:} \quad (\neg A \vee B \vee C) \quad \leftrightarrow \quad \{-A, B, C\} \\
 \text{leere Menge} \rightarrow \text{leere Klausel } \{\}
 \end{array}$$

$$\begin{array}{l}
 \text{Beispiel:} \quad \{-A, B, C\}, \{B, C\}, \{A, C\} \\
 \text{entspricht:} \quad (\neg A \vee B \vee C) \wedge (B \vee C) \wedge (A \vee C)
 \end{array}$$

- Klausel wird wahr, wenn eine Variable wahr wird (da Disjunktion)
- Klauselmenge wird wahr, wenn alle Klauseln wahr sind
- leere Klausel erhält den Wahrheitswert „falsch“  $\rightarrow$  Klauselmenge mit leerer Klausel ist immer unerfüllbar

#### Resolvente:

- kann aus 2 Klauseln gebildet werden, wenn eine Variable in einer Klausel vorkommt und in einer anderen als Negation
- z.B.:  $\{A, B\}, \{\neg A, B\} \rightarrow$  Resolvente:  $\{B, C\}$
- Resolvente wird der Klauselmenge angefügt und bleibt zur ursprünglichen äquivalent

#### Resolventenprinzip:

Behauptung  $\rightarrow A$  sei wahr

- zur Klauselmenge wird ein Widerspruch hinzugefügt  $\{\neg A\}$
- anschließend Resolventen bilden, diese werden der ursprünglichen Formel hinzugefügt
- lässt sich die leere Menge als Resolvente ableiten, so ist die Behauptung wahr

*Beispiel A sei wahr:*

Formel	$\{A, B, \neg C\}, \{A, B, C\}, \{A, \neg B\}$
Hinzufügen von $\{\neg A\}$ und Bildung von Resolventen	$\{\neg A\}, \{A, B, \neg C\}, \{A, B, C\}, \{A, \neg B\}$ 1.        2.        3.        4. 2. u. 4. $\{A, \neg C\}$ 5. 2. u. 3. $\{A, B\}$ 6. 4. u. 6. $\{A\}$ 7. 1. u. 7. $\{\}$ $\Rightarrow$ Widerspruch $\Rightarrow$ erfüllbar

### 2.5.9. Hornformeln

Aussagenlogische Formel mit maximal einem Atom in der Konklusion, bzw. Formel in KNF und höchstens einem positiven Literal pro Klausel (Umwandlung Implikation  $\rightarrow$  Disjunktionsterm)

z.B.:  $(A \rightarrow B) \Leftrightarrow (\neg A \vee B)$  (maximal ein Atom in der Prämisse und maximal ein positives Literal  $\rightarrow$  Hornformel)

weiteres Beispiel:

$$(A \vee \neg B) \wedge (\neg C \vee \neg A \vee D) \wedge (\neg A \vee \neg B) \wedge D$$

$$(B \rightarrow A) \wedge (C \wedge A \rightarrow D) \wedge (A \wedge B \rightarrow 0) \wedge (1 \rightarrow D)$$

$$D \Leftrightarrow (1 \rightarrow D)$$

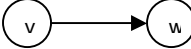

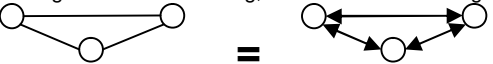
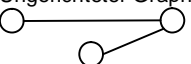
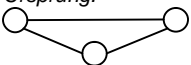
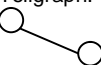
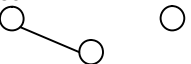
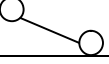
### 2.5.10. Markierungsalgorithmus für Hornformeln

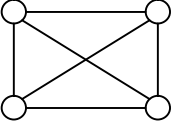
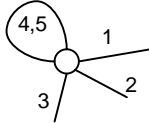
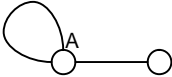
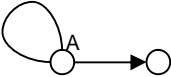
Beispiel 1:	$(E \rightarrow 0) \wedge (C \rightarrow A) \wedge C \wedge B \wedge (G \rightarrow D) \wedge G$
1. Schritt	<p>Markierung aller Vorkommen von einer atomaren Formel x, falls es eine Teilformel der Form (Wahr <math>\rightarrow</math> 1):</p> $(E \rightarrow 0) \wedge (*C \rightarrow A) \wedge *C \wedge *B \wedge (*G \rightarrow D) \wedge *G$
2. Schritt	<p>Prüfen ob es eine Teilformel in den Formen</p> $G = (A_1 \wedge A_2 \dots \wedge A_n \rightarrow B) \quad (\text{Typ 1})$ $G = (A_1 \wedge A_2 \dots \wedge A_n \rightarrow 0) \quad (\text{Typ 2})$ <p>gibt, welche in der Prämisse alle Atome markiert haben, aber in der Konklusion noch nicht.</p> <p>Wenn Teilformel von Typ 1 ist, markiere alle B, sonst (Teilformel von Typ 2) gib „<u>unerfüllbar</u>“ aus und Stoppe.</p> <p><math>\rightarrow</math> Schritt 2 wird wiederholt, solange es Teilformeln ohne markierte Konklusion gibt (Typ 1 oder Typ 2)</p> $(E \rightarrow 0) \wedge (*C \rightarrow A) \wedge *C \wedge *B \wedge (*G \rightarrow D) \wedge *G$ <p>Markiere A (<math>*C \rightarrow A</math>) und D (<math>*G \rightarrow D</math>)</p> $(E \rightarrow 0) \wedge (*C \rightarrow *A) \wedge *C \wedge *B \wedge (*G \rightarrow *D) \wedge *G$ <p><math>\rightarrow</math> alle Teilformeln mit markierter Prämisse sind vom Typ 1 <math>\rightarrow</math> Schritt 3</p>
3. Schritt	Gib erfüllbar aus.
Beispiel 2:	$(A \wedge E \rightarrow 0) \wedge (1 \rightarrow A) \wedge (A \wedge C \rightarrow E) \wedge (A \wedge F \rightarrow C) \wedge (1 \rightarrow F)$ <p>1. Markiere A, F wegen <math>(1 \rightarrow A)</math> und <math>(1 \rightarrow F)</math></p> $(*A \wedge E \rightarrow 0) \wedge (1 \rightarrow *A) \wedge (*A \wedge C \rightarrow E) \wedge (*A \wedge *F \rightarrow C) \wedge (1 \rightarrow *F)$ <p>2. Markiere C wegen <math>(*A \wedge *F \rightarrow C)</math></p> $(*A \wedge E \rightarrow 0) \wedge (1 \rightarrow *A) \wedge (*A \wedge *C \rightarrow E) \wedge (*A \wedge *F \rightarrow *C) \wedge (1 \rightarrow *F)$ <p>3. Markiere E wegen <math>(*A \wedge *C \rightarrow E)</math></p> $(*A \wedge *E \rightarrow 0) \wedge (1 \rightarrow *A) \wedge (*A \wedge *C \rightarrow *E) \wedge (*A \wedge *F \rightarrow *C) \wedge (1 \rightarrow *F)$ <p><math>\rightarrow</math> unerfüllbar wegen <math>(*A \wedge *E \rightarrow 0) = \text{Typ 2}</math></p>

## 2.6. Graphentheorie

### 2.6.1. Definitionen

Ein Graph stellt eine Menge von Objekten zusammen mit einer Beziehung dar, z.B. Personen, A kennt B.

Kante	Verbindung von 2 Knoten (Pfeil)  - Kante $(v,w)$ ist <u>inzident</u> zu $v$ und $w$ - $v$ u. $w$ sind <u>adjazent</u> (benachbart)
Gerichteter Graph	Pfeile zeigen in bestimmte Richtung 
Ungerichteter Graph	Pfeile zeigen in keine Richtung, bzw. in beide Richtungen 
Schlichter Graph	Ungerichteter Graph ohne Mehrfachkanten und Schlingen 
Teilgraph	Teil eines Graphen, z.B. <i>Ursprung:</i>  <i>Teilgraph:</i> 
Spannender Teilgraph	Sämtliche Knoten des Ursprungsgraphen müssen vorhanden sein: 
Gesättigter Teilgraph	Teilmenge der Knoten, aber mit allen Kanten 

Vollständiger Graph	<p>Jeder Knoten ist mit jedem anderen verbunden</p> 
Grad eines Knotens	<p>- Anzahl der inzidenten Kanten  - Schlingen werden doppelt gezählt</p> <p>- bei gerichteten Graphen:  Ausgangsgrad  Eingangsgrad</p>   <p>A Grad: 3</p>  <p>A Ausgangsgrad: 2  A Eingangsgrad: 1</p>
Weg	Verbindung auf Graph den man „ablaufen“ kann
Kreis	Anfang = Ende, sonst jeder Knoten nur einmal ansteuerbar
elementarer Weg	ohne Schlingen
Länge des Weges	- bei normalen Graphen, Anzahl Kanten - bei gewichteten → Wichtung

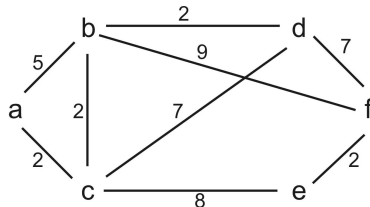
### 2.6.2. Eulerscher Weg

- Graph hat einen Weg (a,b) der alle Kanten genau einmal enthält
- a, b sind die einzigen Knoten mit ungeraden Grad (z.B. Haus vom Nikolaus)

### 2.6.3. Eulerscher Graph

- jeder Knoten von G hat einen geraden Grad

2.7. Graph kürzester Weg



	0	1	2	3	4	5
$l_a$	<u>0</u>	0	0	0	0	0
$l_b$	$\infty$	5	<u>4</u>	4	4	4
$l_c$	$\infty$	<u>2</u>	2	2	2	2
$l_d$	$\infty$	$\infty$	9	<u>6</u>	6	6
$l_e$	$\infty$	$\infty$	10	10	<u>10</u>	10
$l_f$	$\infty$	$\infty$	$\infty$	$\infty$	13	<u>12</u>
$p_a$	*	*	*	*	*	*
$p_b$	*	a	c	c	c	c
$p_c$	*	a	a	a	a	<b>a</b>
$p_d$	*	*	c	b	b	b
$p_e$	*	*	c	c	c	<b>c</b>
$p_f$	*	*	*	b	b	<b>e</b>

- Startknoten a, Weg nach A = 0
- kürzester Weg wird in alle  $l_a$  eingetragen
- alle Weg von a zu anderen Knoten eintragen, wenn kein Weg möglich  $\infty$
- vorgetragene Wege zu Knoten müssen nicht mehr berücksichtigt werden, z.B. bei  $1 \rightarrow 2$
- Knotenwege werden eingetragen
- vortragen wie bei Schritt 1
- in der letzten Spalte kann der Weg abgelesen werden
- **$f \rightarrow e \rightarrow c \rightarrow a$**
- Kürzester weg ist:  $a \rightarrow c \rightarrow e \rightarrow f$

Bearbeitete unbearbeitete Knoten:

	u	b
<b>0</b>	a,b,c,d,e,f	
<b>1</b>	b,c,d,e,f	a
<b>2</b>	b,d,e,f	a,c
<b>3</b>	d,e,f	a,b,c
<b>4</b>	e,f	a,b,c,d
<b>5</b>	f	a,d,c,d,e

Knoten die bearbeitet wurden werden hier eingetragen und müssen für weitere Wegezähl nicht mehr berücksichtigt werden (vortragen in Schritt 1)

**2.7.1. Flüsse in Netzwerken, Ermittlung maximaler Fluss**

<p><b>Netzwerk:</b></p>	
<p><b>1. Restgraph erstellen:</b></p>	<p><b>2. Ermittlung Vorwärtspfade:</b>  <math>4 \ 3 \ 2 \ 4 \ 3</math>  <math>q-b-a-c-d-s</math>                  2 = maximal noch an Kapazität auf Pfad                  → Pfad mit der größten Kapazität wird genommen, ansonsten der kürzeste</p>
<p><b>3. Neuen Restgraphen erstellen</b></p>	<p>Hinweg: -2                  Rückweg: +2                   anschließend neuen Vorwärtspfad ermitteln und Schritte wiederholen, sonst Ende</p>
<p><b>4. Bestimmung maximaler Fluss</b></p>	<p>- einzeichnen der neuen Flusswerte in Ursprungsgraph</p>



## 2.8. Automatentheorie

### 2.8.1. Grundbegriffe

Alphabet	- nichtleere endliche Menge von Symbolen ( $A = \{a_1, a_2, \dots, a_n\}$ ) <i>Beispiele:</i> boolesches Alphabet: $A_b = \{0, 1\}$ lateinisches Alphabet: $A_{lat} = \{A, \dots, Z\}$
Wort	- endliche Folge an Symbolen eines Alphabets $w = a_i \dots a_j \mid a_k \in A, i \leq k \leq j$  $A^*$ = Menge der Wörter über A inkl. des leeren Wortes (Länge 0 bis beliebig) $\varepsilon$ = leeres Wort (Länge 0) $A^+$ = wie $A^*$ , aber ohne leeres Wort (Länge 1 bis beliebig)
Formale Sprache	- eine Menge von Wörtern über einem Alphabet A ist eine Menge von Wörtern über A <i>Beispiel:</i> $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ Menge der Dezimaldarstellungen natürlicher Zahlen die durch 7 teilbar sind, z.B.: 35, 714, ...
Potenzen von Sprachen	$L^n$ = n-fache Verkettung von L mit $L^0 = \{e\}$ <i>Beispiel:</i> $L = \{0, 01\}$ $L^3 = \{111, 1101, 10101, 010101, 0111, 01011, 1011, 01101\}$
Kleen-Stern	Iteration von L: $L^* = \bigcup_{n \geq 1} L^n$  $L^0 = \{\varepsilon\} \quad L^{i+1} = L^i * L$