Algorithmen und Datenstrukturen

Theoretische Grundlagen der Informatik

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Algorithmen und Datenstrukturen

Sprachen zur Codierung von Daten

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

19. April 2024

Fragestellungen rund um Sprachen

1. **Sprachen als geeignetes Mittel zur Darstellung aller Ein- und Ausgaben?**

2. **Ist Σ^*_{Bool} als einzig verfügbare Sprache in einem Computer ausreichend?**

3. **Wie kann man die Beschreibung einer Sprache systematisieren?** »Die Menge aller syntaktisch korrekten JAVA-Programme« gibt wenig Aufschluss darüber, wie ein solches überhaupt aussieht.

4. **Wie kann man effektiv und effizient testen, ob $w \in L$ gilt und geht das eigentlich für jedes L?** Wie überprüft man beispielsweise, ob ein JAVA-Programm syntaktisch korrekt ist?

Computer arbeiten nach dem EVA-Prinzip:

Eingabe

Verarbeitung

Ausgabe

Zentrale Frage

Wie kann man **alle möglichen** Ein- / Ausgaben formal codieren?

Kodierung von Zahlen

Beispiel 1.11 (Kodierung von natürlichen Zahlen)

Ein Wort

$$x = x_1 x_2 \cdots x_n \in \Sigma_{Bool}^*$$

mit $x_i \in \Sigma_{Bool}$ für $1 \leq i \leq n$, kann als binäre Darstellung einer Zahl $n \in \mathbb{N}$ betrachtet werden, mit

$$Zahlwert(x) = \sum_{i=0}^{n} 2^{n-i} \cdot x_i$$

wobei wir $x_i \in \Sigma_{Bool}$ in naheliegender Weise als natürliche Zahl interpretieren.

Die Umkehrfunktion zu *Zahlwert* bezeichnen wir mit $Bin : \mathbb{N} \to \Sigma_{Bool}^*$, wobei *Bin* immer die kürzeste Binärdarstellung liefere (d.h. ohne führende Nullen).

- Kodierungen von $\mathbb{Z}$ / $\mathbb{R}$ ⇒ Vorlesung »IT-Grundlagen«

Kodierung von Zahlenfolgen

Beispiel 1.12 (Kodierung von Zahlenfolgen (eindimensionales Feld))

Eine endliche Folge von natürlichen Zahlen

$$(n_1, n_2, \cdots, n_k) \quad \text{mit} \quad n_i \in \mathbb{N} \;\; \text{für} \;\; 1 \leq i \leq k$$

kann man wie folgt als Wort über $\Sigma_{Bool} \cup \{\#\}$ darstellen:

$$Bin(n_1)\#Bin(n_2)\#\cdots\#Bin(n_k)$$

Wir nutzen das Symbol $\#$ also als Trennzeichen.

Beachte: später müssen wir uns natürlich auf 0 und 1 beschränken!

Kodierung von Matrizen 1

Beispiel 1.13 (Kodierung von Matrizen (zweidimensionales Feld))

Sei $A \in \mathbb{N}^{m \times n}$ eine $m \times n$ Matrix (mit m Zeilen und n Spalten); z.B.:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

mit $a_{ij} \in \mathbb{N}$ für $1 \leq i \leq m$ und $1 \leq j \leq n$.

Dann kodieren wir die Matrix A zeilenweise als Wort über $\Sigma_{Bool} \cup \{\#\}$ wie folgt:

$$\underbrace{b_{11}\#b_{12}\#\cdots\#b_{1n}}_{\text{Zeile 1}}\#\#\underbrace{b_{21}\#b_{22}\#\cdots\#b_{2n}}_{\text{Zeile 2}}\#\#\cdots\#\#\underbrace{b_{m1}\#b_{m2}\#\cdots\#b_{mn}}_{\text{Zeile m}}$$

mit $b_{ij} = Bin(a_{ij})$.

Lesbarkeit?

Kodierung von Matrizen 2

Beispiel 1.14 (Kodierung von Matrizen (zweidimensionales Feld))

Sei $A \in \mathbb{N}^{m \times n}$ eine $m \times n$ Matrix (mit m Zeilen und n Spalten); z.B.:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

mit $a_{ij} \in \mathbb{N}$ für $1 \le i \le m$ und $1 \le j \le n$.

Alternativ könnten wir die Matrix A auch als Wort über $\Sigma_{Bool} \cup \{;.(,)\}$ kodieren:

$$\underbrace{(b_{11}; b_{12}; \cdots; b_{1n})}_{\text{Zeile 1}} \underbrace{(b_{21}; b_{22}; \cdots; b_{2n})}_{\text{Zeile 2}} \cdots \underbrace{(b_{m1}; b_{m2}; \cdots; b_{mn})}_{\text{Zeile m}}$$

(auch hier ist $b_{ij} = Bin(a_{ij})$).

Effizienz? Einlesen einer Matrix in aus einer Datei?

Kodierung von Matrizen 3

Beispiel 1.15 (Kodierung von Matrizen (zweidimensionales Feld))

Sei $A \in \mathbb{N}^{m \times n}$ eine $m \times n$ Matrix (mit m Zeilen und n Spalten); z.B.:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

mit $a_{ij} \in \mathbb{N}$ für $1 \leq i \leq m$ und $1 \leq j \leq n$.

Manchmal hilfreich: Dimensionen der Matrix explizit in die Kodierung aufnehmen; z.B. indem man die Matrix A wie folgt als Wort über $\Sigma_{Bool} \cup \{;.(,)\}$ kodiert:

$$\underbrace{(m', n')}_{\text{Dimension}} \; \underbrace{(b_{11}; b_{12}; \cdots; b_{1n})}_{\text{Zeile 1}} \underbrace{(b_{21}; b_{22}; \cdots; b_{2n})}_{\text{Zeile 2}} \cdots \underbrace{(b_{m1}; b_{m2}; \cdots; b_{mn})}_{\text{Zeile m}}$$

wobei $b_{ij} = Bin(a_{ij})$, $m' = Bin(m)$ und $n' = Bin(n)$.

Effizienz? Speicherplatzverbrauch?

Kodierung von Graphen

Graphen sind eine wichtige Datenstruktur in der Informatik

Definition 1.21 (Graph)

Ein **Graph** $G = \langle V, E \rangle$ besteht aus einer endlichen Knotenmenge V und einer Relation $E \subseteq V \times V$ - der Kantenmenge. Abhängig von den Eigenschaften von E unterscheidet man

- **ungerichtete Graphen**: E ist symmetrisch; d.h.:

$$(v_1, v_2) \in E \;\Leftrightarrow\; (v_2, v_1) \in E$$

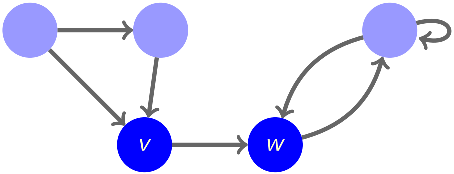
- **gerichtete Graphen**: E ist nicht symmetrisch
- **vollständige Graphen**: $E = V \times V$ (jeder Knoten ist mit jedem anderen verbunden)
- **schleifenfreier Graph**: $\forall (v, v) \in V \times V \;:\; (v, v) \notin E$

- Die Bezeichnung von Knoten und Kanten leiten sich aus dem Englischen ab:
  - V: **vertices** - die Knotenmenge
  - E: **edges** - die Kantenmenge
- Wenn wir zwischen gerichtetem und ungerichtetem Graph nicht unterscheiden wollen, sprechen wir allgemein von einem Graph

Graphen lassen sich sehr anschaulich darstellen:

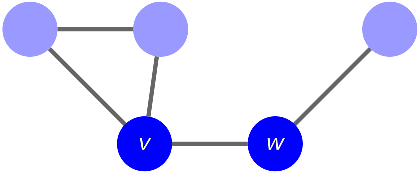
Gerichteter Graph

- v ist **Vorgänger** von w
- w ist **Nachfolger** von v
- v und w sind **Nachbarn** bzw. **adjazent**



Ungerichteter Graph

- v und w sind **Nachbarn** bzw. **adjazent**



Anwendungsgebiete für Graphen

Graphen finden vielfache Anwendung; z.B.

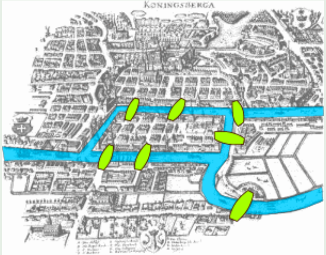
- **Beziehung zwischen Personen** (Knoten entsprechen Personen)
  - Person A kennt Person B
  - Person A ist Vater von Person B
- **Verbindungen zwischen Punkten**
  - Straßennetz
  - Eisenbahnnetz
  - Telefonnetz
  - Internet
  - elektronische Schaltkreise

Typische Fragestellungen

- Existiert eine Verbindung von A nach B?
- Existiert eine zweite Verbindung, falls diese blockiert ist?
- Was ist die kürzeste Verbindung von A nach B?
- Optimale Rundreise (*Traveling Salesman Problem*)
- Minimaler Spannbaum

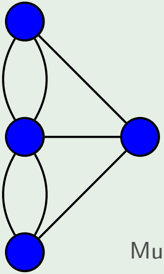
Das Königsberger Brückenproblem

- 1760 von Leonard Euler formuliert
- **Frage**: Gibt es einen Weg, bei dem man jede Brücke genau einmal überquert und schließlich wieder zum Ausgangspunkt zurückkehrt?
- **Antwort**: **Nein!**



Graphentheoretische Betrachtung

- Landmassen $\mapsto$ Knoten
- Brücken $\mapsto$ Kanten
- **Frage**: Gibt es einen geschlossenen Kantenzug (Anfangs-=Endknoten), der jede Kante genau einmal durchläuft? (Eulerscher Kreis)


Multigraph

Anwendungsbeispiel 1: Graphentheoretische Betrachtung

Definition 1.22 (Pfad / Kreis / Eulerscher- und Hamiltonscher Kreis)

Sei $G = \langle V, E \rangle$ ein Graph. Eine Folge von Knoten $p = (w_1, w_2, \cdots, w_k)$ (mit $w_i \in V$) heißt **Pfad** in G wenn

$$\forall i \in \{1, \cdots, k-1\} \; : \; (w_i, w_{i+1}) \in E$$

Ein Pfad heißt

- **Kreis**, falls $w_1 = w_k$
  (Startknoten = Endknoten)
- **Eulerscher Kreis**, falls er ein Kreis ist und
  $\forall (s,t) \in E \; : \; \exists_1 i \; : \; s = w_i \; \wedge \; t = w_{i+1}$
  (jede **Kante** wird genau einmal durchlaufen)
- **Hamiltonscher Kreis**, falls er ein Kreis ist und $\forall v \in V \; : \; \exists_1 i \; : \; v = w_i$
  (jeder **Knoten** wird genau einmal besucht)

Anmerkung: Pfade können wir offensichtlich leicht in Form von Zahlenfolgen kodieren.

Das Königsberger Brückenproblem entspricht der Suche nach einem Eulerschen Kreis in einem ungerichteten Multigraphen.

Gewichtete Graphen

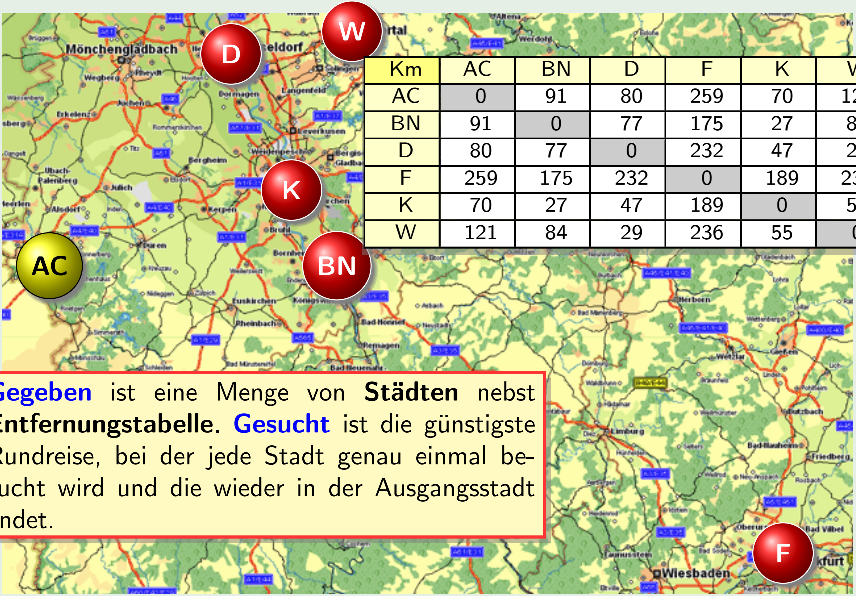
Definition 1.23 (Gewichtete Graphen)

Ein **gewichteter Graph** $G = \langle V, E, \phi \rangle$ ist ein Graph $\langle V, E \rangle$, erweitert um eine totale Funktion $\phi : E \to R$, die jeder Kante ein Gewicht zuordnet. R ist dabei eine beliebige Menge (z.B. $\mathbb{R}$).

- Gewichtete Graphen finden z.B. Anwendung bei der Modellierung von Straßennetzen (Navigationssysteme!):
    - Knoten entsprechen Kreuzungen, Einmündungen, Sackgassen, POIs etc.
    - Kanten sind gerichtet (z.B. Einbahnstraßen)
    - Gewicht entspricht Entfernung

Anwendungsbeispiel 2: Traveling Salesman Problem

Das Traveling Salesman Problem

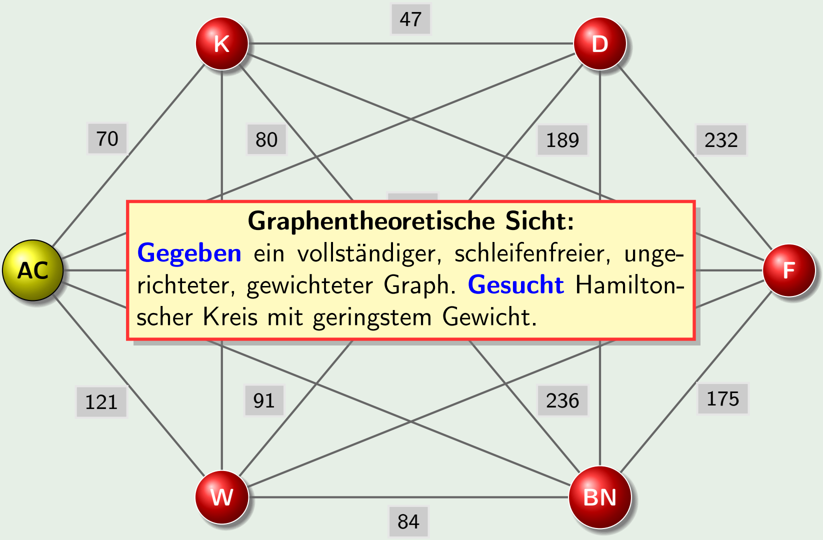


Km	AC	BN	D	F	K	W
AC	0	91	80	259	70	121
BN	91	0	77	175	27	84
D	80	77	0	232	47	29
F	259	175	232	0	189	236
K	70	27	47	189	0	55
W	121	84	29	236	55	0

Gegeben ist eine Menge von **Städten** nebst **Entfernungstabelle**. **Gesucht** ist die günstigste Rundreise, bei der jede Stadt genau einmal besucht wird und die wieder in der Ausgangsstadt endet.

Beispiel 1.16 (Traveling Salesman Problem)

Wir **modellieren** das Problem durch einen ungerichteten, gewichteten Graphen:



Graphentheoretische Sicht:
Gegeben ein vollständiger, schleifenfreier, ungerichteter, gewichteter Graph. **Gesucht** Hamiltonscher Kreis mit geringstem Gewicht.

Anwendungsbeispiel 2: Graphentheoretische Betrachtung 2

Definition 1.24 (Pfadgewicht)

Sei $G = \langle V, E, \phi \rangle$ ein gewichteter Graph und $p = (w_1, w_2, \cdots, w_k)$ ein Pfad in G. Dann beschreibt

$$\phi(p) = \sum_{i=1}^{k-1} \phi(w_i, w_{i+1})$$

das **Pfadgewicht** von p.

Erste algorithmische Idee:

1. Bestimme $L_{Hamil}(G) = \{w \mid w \text{ kodiert Hamiltonschen Kreis in } G\}$
2. Suche mittels ϕ *günstigsten* Pfad in dieser Menge

Laufzeiteffizienz, Speicherplatzverbrauch?

Wie berechnet man $L_{Hamil}(G)$ und wie viele Pfade enthält diese Menge?

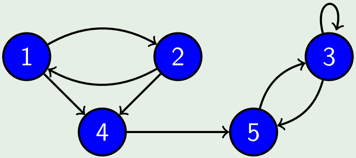
Die Adjazenzmatrix

Definition 1.25 (Adjazenzmatrix)

Sei $G = \langle V, E \rangle$ ein Graph mit $V = \{v_1, v_2, \cdots, v_n\}$. Die **Adjazenzmatrix** $A = (a_{ij})$ zu G ist eine $n \times n$-Matrix mit

$$a_{ij} = \begin{cases} 1 & \text{falls } (v_i, v_j) \in E \\ 0 & \text{sonst} \end{cases}$$

Beispiel 1.17 (Adjazenzmatrix)



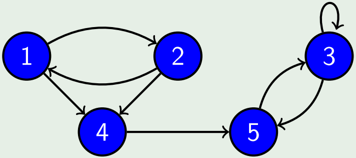
$$\begin{array}{c|c} A & \text{nach} \\ \hline \text{von} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Beachte: Für ungerichtete Graphen ist die Adjazenzmatrix offenbar symmetrisch!

Beispiel: Kodierung eines Graphen

Beispiel 1.18 (Kodierung eines Graphen)

- Ein Graph ist durch seine Adjazenzmatrix eindeutig repräsentiert:



A	nach
von	$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$

- Wir kodieren die Matrix zeilenweise als Wort über $\Sigma_{Bool} \cup \{\#\}$
- Spaltentrennzeichen hier überflüssig - ein Eintrag ist exakt ein Zeichen lang:

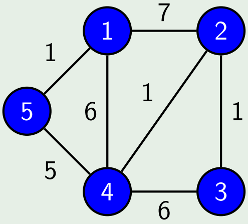
$$\underbrace{01010}_{\text{Zeile 1}} \# \underbrace{10010}_{\text{Zeile 2}} \# \underbrace{00101}_{\text{Zeile 3}} \# \underbrace{00001}_{\text{Zeile 4}} \# \underbrace{00100}_{\text{Zeile 5}}$$

Kodierung von gewichteten Graphen

Beispiel 1.19 (Kodierung von gewichteten Graphen)

- Zwei Komponenten:
  - Adjazenzmatrix A zur Kodierung von E
  - Gewichtsmatrix W zur Kodierung von ϕ mit $w_{ij} = \phi(v_i, v_j)$

Graph:



Gewichtsmatrix:

W	nach
von	$\begin{pmatrix} 0 & 7 & 0 & 6 & 1 \\ 7 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 6 & 0 \\ 6 & 1 & 6 & 0 & 5 \\ 1 & 0 & 0 & 5 & 0 \end{pmatrix}$

W könnte wieder zeilenweise kodiert werden

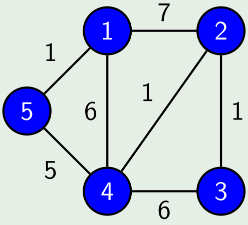
0#111#0#110#1##111#0#1#1#0##0#1#0#110#0##
110#1#110#0#101##1#0#0#101#0

Insgesamt **69** Zeichen.

Kodierung von gewichteten ungerichteten Graphen

Beispiel 1.20 (Kodierung von gewichteten Graphen)

Graph:



Gewichtsmatrix:

W	nach
von	$\begin{pmatrix} 0 & 7 & 0 & 6 & 1 \\ 7 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 6 & 0 \\ 6 & 1 & 6 & 0 & 5 \\ 1 & 0 & 0 & 5 & 0 \end{pmatrix}$

Beobachtung: Die Gewichtsmatrix ist symmetrisch und nur Nullen auf Hauptdiagonale!

Es ist ausreichend das »obere Dreieck« zu kodieren:

111#0#110#1##1#1#0##110#0##101

Insgesamt nur noch **30** Zeichen.

Standortbestimmung

Standort

- **Wir haben gelernt**
  - wie man auch komplexere Daten durch Wörter (Sprachen) beschreiben kann
- **Weitere Fragen bezüglich der Kodierung von Daten**
  1. Ist Σ_{Bool}, das Alphabet eines Computers, wirklich ausreichend?
     [Wir hatten z.B. mit # Hilfssymbole verwendet und Formeln sogar über Σ_{logic} kodiert]
  2. Wie viele Wörter enthält eine Sprache, wenn Sie nicht unendlich ist?
     [Wie viele Wörter hat z.B. $L_{Hamil}(G) = \{w \mid w \text{ kodiert Hamiltonschen Kreis in } G\}$]
- **Noch zu klären**
  - Wie kodieren wir algorithmische Probleme mithilfe von Sprachen?
    [Das sind Probleme, die wir durch Algorithmen lösen wollen]
- **Offen gebliebene Fragen zur Kodierung von Daten**
  - Gibt es eine zu einem Datum eine kürzeste Darstellung als Wort über einem beliebigem Alphabet?
    [Diese Frage hatten wir vertagt]

Bevor wir untersuchen wie man mit Sprachen algorithmische Probleme beschreiben kann, werden wir die offenen Fragen zur Kodierung von Daten beantworten.

Das EVA-Prinzip und unser Modell (bisher)

$L_{in} \subseteq \Sigma_i^*$

Eingabe

Verarbeitung

$L_{out} \subseteq \Sigma_o^*$

Ausgabe

Kodierung der Ein-/Ausgaben

1. Formale Repräsentation finden (z.B. Folge, Graph) [**Modellierung**]
2. Kodierung formaler Objekte als Wörter über Σ_i (Eingaben) bzw. Σ_o (Ausgaben) [**Implementierung**]

Beispiel: Traveling Salesman Problem

Beispiel 1.21 (Ein- und Ausgabe beim Travelling Salesman Problem)

Modell (abstrakte, formale Beschreibung):

- **Eingabe**: Graph $G = \langle V, E, \phi \rangle$ (vollständig, schleifenfrei, ungerichtet)
- **Ausgabe**: Pfad (Kreis) $P = (v_1, \cdots, v_k)$

Implementierung (später: Datentypen in konkreter Programmiersprache):

- **Eingabe**: $w \in L_{in}$ mit

$$L_{in} = \left\{ w \in \{0, 1, \#\}^* \;\middle|\; w \text{ kodiert ungerichteten, gewichteten Graph} \right\}$$

- **Ausgabe**: $w \in L_{out}$ mit

$$L_{out} = \left\{ w \in \{0, 1, \#\}^* \;\middle|\; w \text{ kodiert einen Pfad} \right\}$$

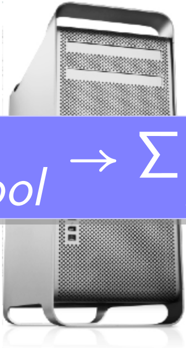
Beachte: Wir betrachten hier nur die Art (den Typ) von Ein- und Ausgabe. Wir legen noch nicht fest, dass die Ausgabe Hamiltonscher Kreis mit minimalem Gewicht sein soll.

Das EVA-Prinzip und unser Modell (bisher)

$L_{in} \subseteq \Sigma_i^*$

Eingabe

$$\Sigma_{Bool}^* \rightarrow \Sigma_{Bool}^*$$



Verarbeitung

$L_{out} \subseteq \Sigma_o^*$

Ausgabe

Beschränkung auf Σ_{Bool}^*

1. Computer arbeiten auf Wörtern aus Σ_{Bool}^*
2. Können wir alle Sprachen auf Σ_{Bool}^* abbilden? [**Compiler**]

Vorüberlegungen

- L_{in} kann unendlich viele Wörter enthalten; Σ^*_{Bool} enthält unendlich viele Wörter.
- **Aufgabe**: jedem Wort $w \in L_{in}$ ein Wort $v \in \Sigma^*_{Bool}$ zuordnen.
  - **Lösung**: Relation $encode \subseteq L_{in} \times \Sigma^*_{Bool}$

Anforderungen an *encode*:

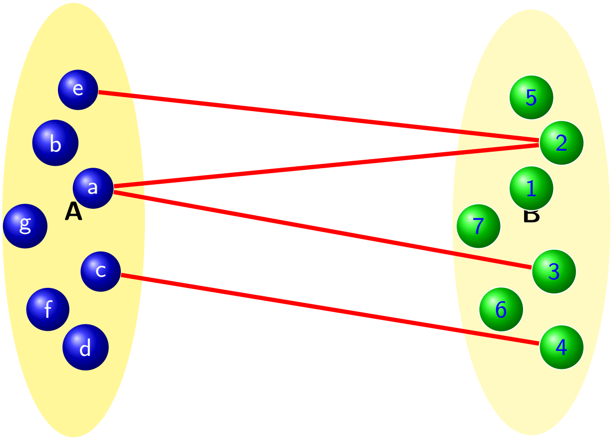
- **Kodierung muss eindeutig sein**
  (unterschiedl. Wörter erhalten unterschiedl. Code)

  Lösung: *encode* als **Funktion** $encode : L_{in} \to \Sigma^*_{Bool}$
- **Kodierte Wörter müssen sich eindeutig dekodieren lassen**

  Lösung: *encode* muss **injektiv** sein
- **Möglichst einfache, endliche Beschreibung**

  Lösung: beschreibe *encode* als **homomorphe Fortsetzung**

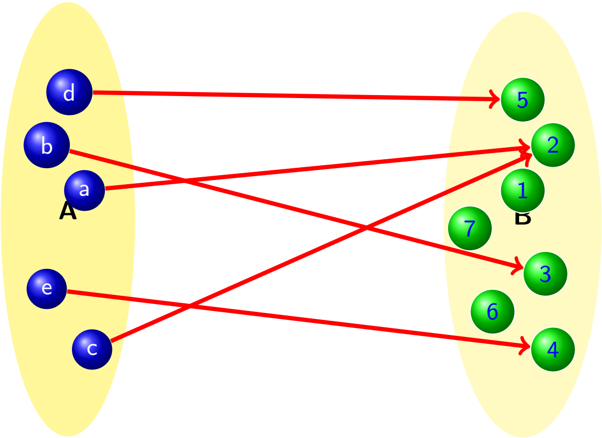
Relation: Seien A und B Mengen. Eine Relation R ist eine Teilmenge von $A \times B$. Relationen kann man auch grafisch veranschaulichen:



Anstelle von $(a, b) \in R$ schreibt man auch $a\,R\,b$, um auszudrücken, dass die Elemente a und b in Beziehung stehen. In unserem Beispiel gelten etwa $a\,R\,2$ und $e\,R\,2$.

Zu Relationen siehe auch `http://de.wikipedia.org/wiki/Relation_(Mathematik)`

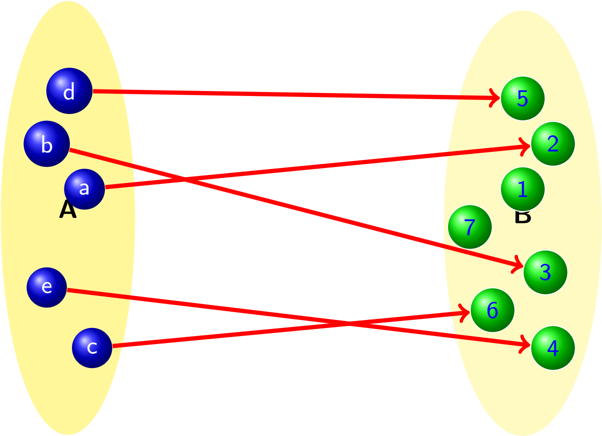
Funktion: Seien A und B Mengen. Eine Relation R heißt Funktion, wenn Sie jedem Element der Menge A genau ein Element der Menge B zuordnet. Man spricht von einer Abbildung $R : A \to B$ mit **Definitionsbereich** A und **Bildbereich** B.



Anstelle von $(a, b) \in R$ schreibt man $R(a) = b$. In unserem Beispiel gelten etwa $R(a) = 2$ und $R(d) = 5$.

Injektive Funktion: Eine Funktion $f : A \to B$ heißt injektiv (bzw. umkehrbar), falls

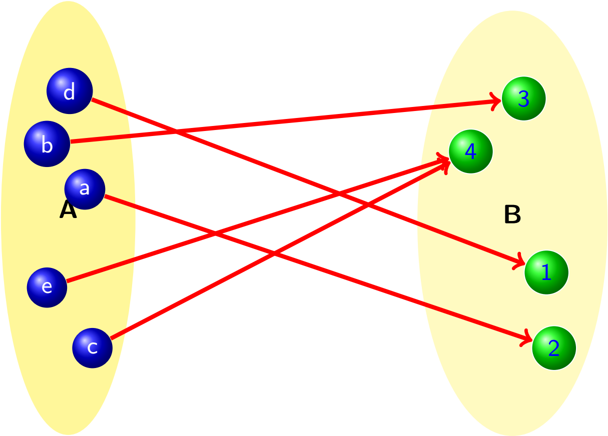
$$\forall x, y \in A \ : \ f(x) = f(y) \ \Leftrightarrow \ x = y$$



Wiederholung: Mathematische Grundlagen 4

Surjektive Funktion: Eine Funktion $f : A \to B$ heißt surjektiv, falls

$$\forall b \in B \ : \ \exists a \in A : f(a) = b$$



Eine Funktion heißt **bijektiv**, wenn Sie sowohl injektiv, als auch surjektiv ist.

Zu Funktionen siehe auch `http://de.wikipedia.org/wiki/Funktion_(Mathematik)`

Anmerkung: Funktionen in der Theoretischen Informatik

Eine Funktion ordnet **jedem** Element aus dem Wertebereich ein Element aus dem Bildbereich zu. Definitionslücken sind explizit zu berücksichtigen betrachte z.B. $\mathit{kehrwert} : \mathbb{R} - \{0\} \rightarrow \mathbb{R}$ mit

$$\mathit{kehrwert}(x) = \frac{1}{x}$$

In der Theoretischen Informatik ist man an konkreten Definitionslücken oft nicht interessiert und führt daher folgende Begriffe ein:

Definition 1.26

Sei $f : (A - D) \rightarrow B$ eine Funktion mit Definitionslücken $D \subseteq A$. f heißt

- **partielle Funktion** $f : A \dashrightarrow B$, wenn f u.U. nicht auf ganz A definiert ist
  ($\mathit{kehrwert}$ ist also eine partielle Funktion $\mathit{kehrwert} : \mathbb{R} \dashrightarrow \mathbb{R}$)
- **totale Funktion** $f : A \rightarrow B$, falls f auf ganz A definiert ist ($D = \emptyset$).

Beachte: nach dieser Definition ist jede totale Funktion auch partiell!

Homomorphismus 1

Definition 1.27 (Homomorphismus)

Seien Σ_1 und Σ_2 Alphabete. Eine Funktion $h : \Sigma_1^* \to \Sigma_2^*$ heißt Homomorphismus, gdw.

1. $h(\varepsilon) = \varepsilon$
2. $h(u \cdot v) = h(u) \cdot h(v)$ für $u, v \in \Sigma^*$

Beachte: Homomorphismen sind verträglich mit der Konkatenation - dem Konstruktor für Wörter.

Homomorphismus 2

Korollar 1.1

Sei $h : \Sigma_1^* \to \Sigma_2^*$ *ein Homomorphismus. Dann gilt für jedes Wort* $w = a_1 a_2 \cdots a_n \in \Sigma_1^*$, $n \in \mathbb{N}$ *mit* $n \geq 1$ *und* $a_i \in \Sigma_1$ $(1 \leq i \leq n)$*:*

$$h(w) = h(a_1)h(a_2)\cdots h(a_n)$$

Beweis: vollständige Induktion über die Wortlänge von w.

Sei $w = a_1 a_2$ für $a_1, a_2 \in \Sigma \cup \{\varepsilon\}$. Da h ein Homomorphismus ist, folgt das Gewünschte sofort aus der Definition, denn $h(a_1 \cdot a_2) = h(a_1) \cdot h(a_2)$, $h(\varepsilon) = \varepsilon$ und ε ist rechts- und linksneutrales Element der Konkatenation. Sei nun $|w| = n$ mit $n > 2$, also $w = a_1 a_2 \cdots a_{n-1} a_n$; dann gilt

$$\begin{aligned} h(w) &= h(a_1 a_2 \cdots a_{n-1} a_n) \\ &= h(a_1 a_2 \cdots a_{n-1}) h(a_n) \\ & \quad h \text{ ist Homomorphismus} \\ &= h(a_1)h(a_2)\cdots h(a_{n-1})h(a_n) \\ & \quad \text{Induktionsvoraussetzung} \end{aligned}$$

$\square$

! Ein Homomorphismus $h : \Sigma_1^* \to \Sigma_2^*$ ist durch seine Wirkung auf den Symbolen aus Σ_1 **eindeutig** bestimmt.

Homomorphe Fortsetzung

Definition 1.28 (Homomorphe Fortsetzung)

Sei $h : \Sigma_1 \to \Sigma_2^*$ eine Funktion. Die **homomorphe Fortsetzung** $h' : \Sigma_1^* \to \Sigma_2^*$ von h ist definiert durch

- $h'(\varepsilon) = \varepsilon$
- $h'(w \cdot a) = h'(w) \cdot h(a)$ für $w \in \Sigma_1^*$, $a \in \Sigma_1$

- **Anmerkungen**:
  - Anstelle von h' werden wir oft einfach nur h schreiben.
  - Ist $h : \Sigma_1 \to \Sigma_2^*$ injektiv, ist dies **nicht** hinreichend dafür, dass die homomorphe Fortsetzung $h' : \Sigma_1^* \to \Sigma_2^*$ injektiv ist!
- Kodierfunktionen *encode* definieren wir jetzt (bequem) als homomorphe Fortsetzung einer injektiven Funktion

Beispiel

Beispiel 1.22 (Kodierung von Graphen)

Adjazenzmatrizen hatten wir als Worte über dem Alphabet $\Sigma = \{0, 1, \#\}$ kodiert. Eine Kodierung über Σ_{Bool} $encode : \Sigma^* \to \Sigma_{Bool}^*$ erhalten wir z.B. durch homomorphe Fortsetzung von $h : \Sigma \to \Sigma_{Bool}^*$ mit

$$\begin{array}{rcl} h(0) & = & 00 \\ h(1) & = & 11 \\ h(\#) & = & 10 \end{array}$$

Dann ist z.B.

$$\begin{array}{rcl} encode(01\#10) & = & h(0) \cdot h(1) \cdot h(\#) \cdot h(1) \cdot h(0) \\ & = & 00 \cdot 11 \cdot 10 \cdot 11 \cdot 00 \end{array}$$

Standardkodierung

Definition 1.29 (Standardkodierung)

Sei $\Sigma = \{a_1, \cdots, a_n\}$ ein Alphabet mit mindestens zwei Symbolen. Die **Standardkodierung** $encode : \Sigma^* \to \Sigma_{Bool}^*$ ist die homomorphe Fortsetzung von $h : \Sigma \to \Sigma_{Bool}^*$ mit

$$h(a_i) = Bin_{\lceil \log_2 n \rceil}(i-1)$$

wobei $Bin_j(m)$ der Binärdarstellung der Zahl m mit mindestens j Stellen entspricht (ggf. Auffüllen mit führenden Nullen).

Anmerkung: Sei $r \in \mathbb{R}$, dann beschreibt

- $\lceil r \rceil$ das kleinste $n \in \mathbb{N}$ mit $n \geq r$
- $\lfloor r \rfloor$ das größte $n \in \mathbb{N}$ mit $n \leq r$

Standortbestimmung

Standort

- **Wir haben gelernt**
  - wie man auch komplexere Daten durch Wörter (Sprachen) beschreiben kann
  - dass Σ_{Bool} als Alphabet ausreichend ist (Homomorphismen)
- **Weitere Fragen bezüglich der Kodierung von Daten**
  - Wie viele Wörter enthält eine Sprache, wenn Sie nicht unendlich ist?
    [Wie viele Wörter hat z.B. $L_{Hamil}(G) = \{w \mid w \text{ kodiert Hamiltonschen Kreis in } G\}$]
- **Noch zu klären**
  - Wie kodieren wir algorithmische Probleme mithilfe von Sprachen?
    [Das sind Probleme, die wir durch Algorithmen lösen wollen]
- **Offen gebliebene Fragen zur Kodierung von Daten**
  - Gibt es eine zu einem Datum eine kürzeste Darstellung als Wort über einem beliebigem Alphabet?
    [Diese Frage hatten wir vertagt]