1 Datenstrukturen

1.5 Graphen

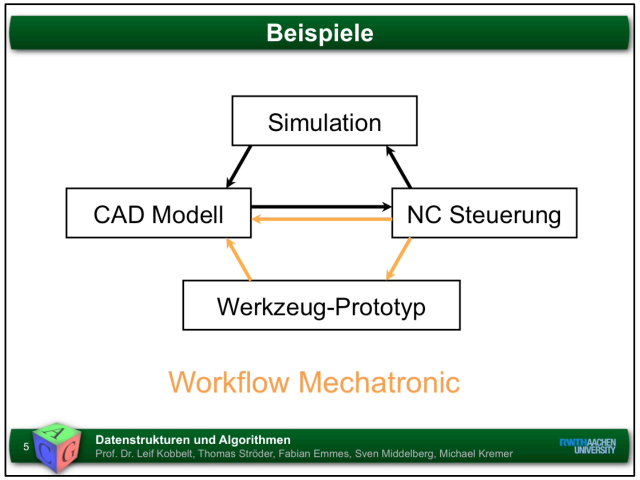
- Darstellung allgemeiner Beziehungen zwischen Objekten/Elementen
  - Objekte = Knoten:
    $V = \{v_1, \ldots, v_n\}$
  - Beziehungen = Kanten:
    $E = \{(a_1,b_1), \ldots (a_m,b_m)\}$
- Kanten können gerichtet (Vorgänger/ Nachfolger) oder ungerichtet (Nachbarschaft) definiert werden.

Beispiele

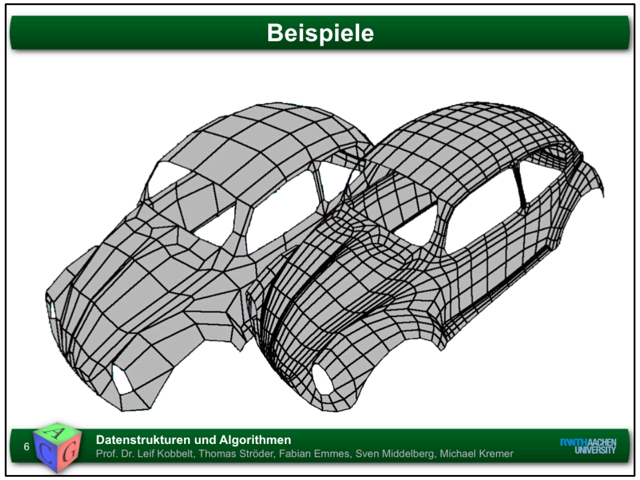
- Flugverbindungen (ungerichtet)
- Straßennetz in Aachen (gerichtet)
- Abhängigkeiten von Arbeitspaketen in einem Großprojekt (gerichtet)
- 3D Polygonmodelle in der Computergraphik (ungerichtet)
- Querbezüge in einem Hypertext-Dokument (HTML) (gerichtet)
- ...

Beispiele

U-Bahn Netz Berlin



Graphen als Darstellung von Produktions/Entwicklungs-Prozessen.



Polygonnetze sind auch Graphen, bei denen die Knoten eine Einbettung in einen Vektorraum haben (zum Beispiel R^3).

Begriffe

- Zusammenhängend (stark)
- Vollständig
- Isomorph
- Planare Graphen

Begriffe

- Zusammenhängend (stark)
  - Für jedes Paar v_i, v_j von Knoten existiert ein Pfad von Kanten von v_i nach v_j oder (und) von v_j nach v_i.
- Vollständig
- Isomorph
- Planare Graphen

Begriffe

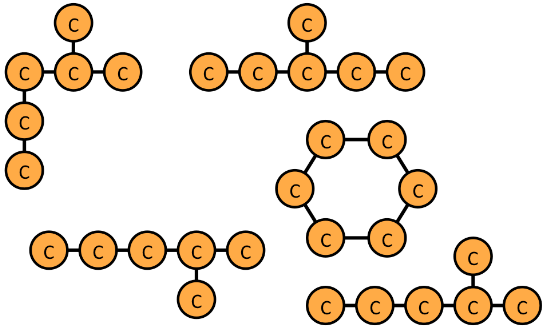
- Zusammenhängend (stark)
- Vollständig
  - Zwischen je zwei Knoten existiert eine Kante
  - $m = n \times (n-1)/2$
- Isomorph
- Planare Graphen

Begriffe

- Zusammenhängend (stark)
- Vollständig
- Isomorph
  - Graphen (V_1,E_1) und (V_2,E_2) durch Permutation der Knoten(indizes) ineinander überführbar.
  - Notwendige Kriterien: $n_1 = n_2$, $m_1 = m_2$, Anzahl der Knoten mit Grad k, ...
- Planare Graphen

Verschieden Anzahl in Knoten oder Kanten => nicht isomorph

Kohlenwasserstoffe

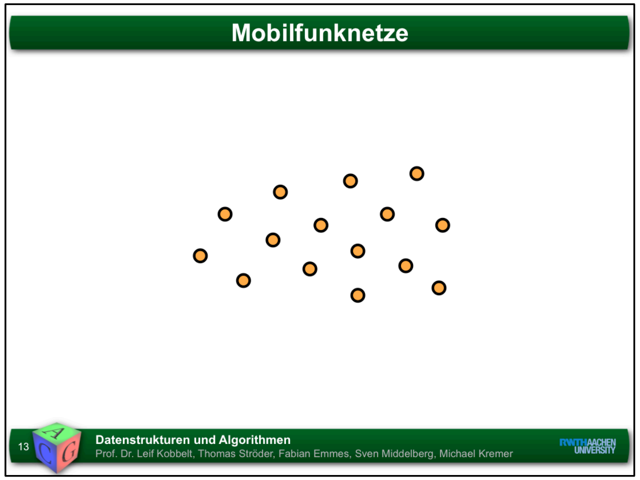


Auch chemische Moleküle können als Graphen dargestellt werden.

Begriffe

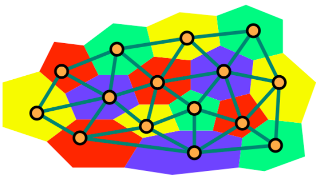
- Zusammenhängend (stark)
- Vollständig
- Isomorph
- Planare Graphen
  - Zerlegung der Ebene in disjunkte Zellen
    - Kompatibilitätsbedingungen zwischen Kanten

Viele in der Praxis relevante Graphen sind planare Graphen.



Zum Beispiel die Darstellung der Abdeckung von Mobilfunknetzen.

Mobilfunknetze



Repräsentationen

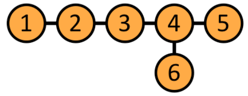
- Allgemeine Graphen
  - Adjazenzmatrix
  - Adjazenzliste
- Planare Graphen
- Simplex-Strukturen

Man benötigt Datenstrukturen, um effizient mit Graphen zu arbeiten

Adjazenzmatrix

- Knoten: $V = \{v_1,\ldots,v_n\}$
- $A \in \mathrm{Bool}^{n \times n}$
- $a_{ij} = 1$ falls $(i,j) \in E$, $a_{ij} = 0$ sonst
- j-te Spalte: alle Vorgänger von v_j
- i-te Zeile: alle Nachfolger von v_i
- A symmetrisch für ungerichtete Graphen

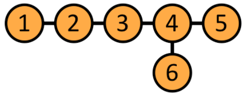
Adjazenzmatrix



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Gängige (aber nicht platzsparende) Darstellungsart eines Graphen sind Adjazenzmatrizen. In ungerichteten Graphen (d.h. die Kanten haben keine Orientierung) sind
diese Matrizen immer symmetrisch und somit um den Faktor 2 redundant. Man könnte dementsprechend auch nur eine Dreiecksmatrix zum Kodieren derselben Information benutzen.

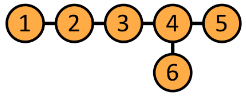
Zusammenhangskomponenten



$$B = A + I = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Addition der Einheitsmatrix: Füge (virtuell) einen Zykel für jeden Knoten ein. Dieser verbindet jeden Knoten mit sich selbst.

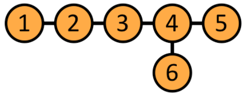
Zusammenhangskomponenten



$$B^2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Die Matrix B^2 kodiert, welche Knoten von jedem Knoten aus in maximal zwei Schritten erreichbar sind.

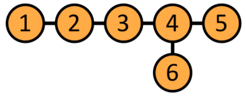
Zusammenhangskomponenten



$$B^3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

In drei Schritten erreichbare Knoten.

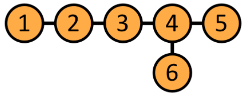
Zusammenhangskomponenten



$$B^4 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

In vier Schritten sind alle Knoten von allen Knoten aus erreichbar.

Zusammenhangskomponenten



$$B^{4} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

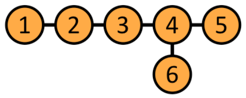
Durchmesser

Die Potenz wird auch Durchmesser genannt, d.h. im Durchmesser d eines Knotens liegen alle benachbarten Knoten, die in maximal d Schritten erreichbar sind.

Adjazenzlisten

- Bei komplexen Graphen bestehen oft nur lokale Knotenbeziehungen
- Die meisten Einträge der Adjazenzmatrix sind „0“ (dünn besetzte Matrix)
- Speichere nur die existierenden Kanten

Adjazenzlisten



1: 2
2: 1,3
3: 2,4
4: 3,5,6
5: 4
6: 4

Im Gegensatz zu Adjazenzmatrizen kodieren Adjazenzlisten nur die Information, die wirklich benötigt wird und sind somit deutlich platzsparender für große Graphen. Beide Darstellungsmöglichkeiten repräsentieren denselben Graphen.

Adjazenzlisten

- Für gerichtete Graphen werden Vorgänger- und Nachfolger-Listen verwaltet.
- Speicherverbrauch: (#Nachbarn ≤ k)
  - A-Matrix: n^2 bit
  - A-Listen: $n \times k \times b = n \times k \times \log n$
- Beispiel:
  Regionalbahnen in Aachen und Wladiwostok

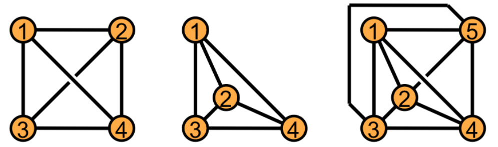
k ist die maximale Valenz eines Knotens in dem zu kodierenden Graphen. Die Valenz eines Knoten ist gleich der Zahl seiner durch eine Kante verbundenen Nachbarn.

Planare Graphen

- ... beschreiben eine Partition der Ebene in disjunkte Zellen.
- Knoten sind durch Kanten verbunden und diese bilden Zellen (Maschen).
- Planar: es ist möglich die Knoten so zu platzieren, dass sich die Kanten nicht kreuzen.

Planare Graphen

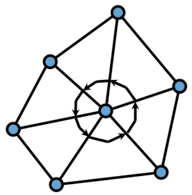
- Kompatibilitätsbedingungen
  - eindeutiger Umlaufsinn für die Nachbarn jedes Knotens
  - eindeutiger Umlaufsinn für die Kanten jeder Zelle / Masche



Für den ersten Graphen (links) gibt es trotz der hier gewählten Darstellung eine Einbettung in die Ebene, nämlich genau die Darstellung in der Mitte.
Ein Graph ist planar, wenn es mindestens eine solche Einbettung gibt. Für den Graphen auf der rechten Seite gibt es eine solche Einbettung nicht.

Kantenstrukturen

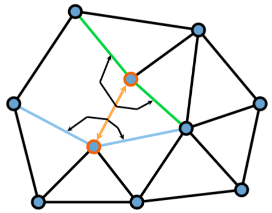
- Kanten verbinden Knoten
- Kanten verweisen auf benachbarte Kanten (Umlaufsinn)
- Garantiert die Planarität



Wenn man für jeden Knoten einen festen Umlaufsinn der inzidenten Kanten definieren kann (wie in dem Bild angedeutet), dann ist der Graph planar.

Kantenstrukturen

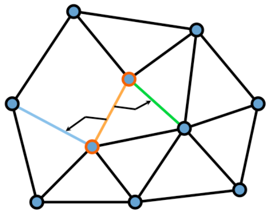
- class Edge {
  Node A,B
  Edge N_1,N_2
  Edge P_1,P_2
  }



Eine Möglichkeit, einen Graphen als Datenstruktur zu implementieren. Hier werden die Nachbarschaftsbeziehungen in jeweils beide Richtungen für die Navigierung direkt mitkodiert.

Kantenstrukturen

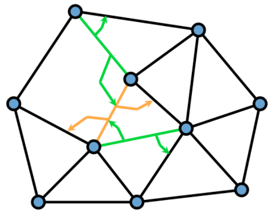
- class Edge {
  Node A,B
  Edge Next
  Edge Prev
  }



Eine etwas reduzierte Darstellung, die nur die Hälfte der eben kodierten Nachbarschaftskanten miteinbezieht.

Kantenstrukturen

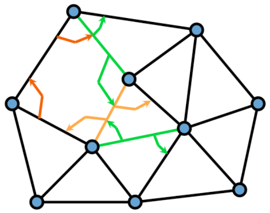
- class Edge {
  Node A,B
  Edge Next
  Edge Prev
  }



Die vollständige Navigation ist in dieser Darstellung jedoch...

Kantenstrukturen

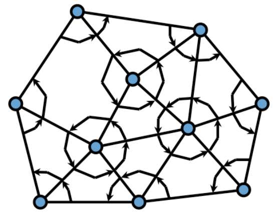
- class Edge {
  Node A,B
  Edge Next
  Edge Prev
  }



...nicht möglich, da die rot eingefärbten Adjazenzen nicht kodiert sind.

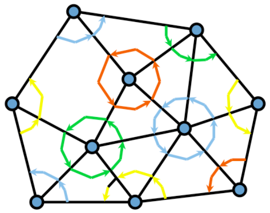
Kantenstrukturen

- class Edge {
  Node A,B
  Edge Next
  Edge Prev
  }



Kantenstrukturen

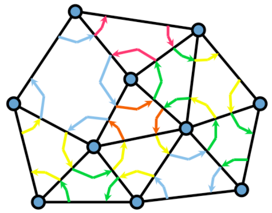
- class Edge {
  Node A,B
  Edge Next
  Edge Prev
  }



Man könnte die Nachbarschaftsbeziehung und den Umlaufsinn auch pro Knoten...

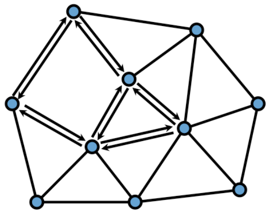
Kantenstrukturen

- class Edge {
  Node A,B
  Edge Next
  Edge Prev
  }



...oder pro Face definieren.

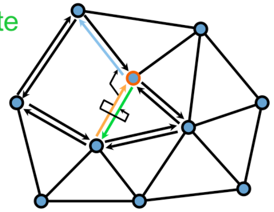
Halbkantenstrukturen



In der Praxis hat sich etabliert, alle Kanten in Paare von entgegengesetzten Halbkanten aufzuteilen. Dies kodiert eine inhärente Orientierung des Graphen und man muss nur...

Halbkantenstrukturen

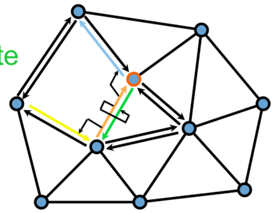
```
class HEdge {
  Node  A
  Edge  Next
  Edge  Opposite
}
```



... eine kleine Menge an Nachbarschaftsbeziehungen pro Halbkante speichern. Dies gewährleistet, dass die Navigation auf dem
Graphen vollständig und eindeutig ist.

Halbkantenstrukturen

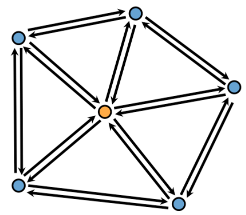
- class HEdge {
  Node A
  Edge Next
  Edge Prev
  Edge Opposite
  }



Prev ist ein Zeiger, der hinzugenommen werden kann, um die Kanten eines Faces effizient im Uhrzeigersinn zu adressieren. Dieser Pointer
Verursacht natürlich einen größeren Speicheraufwand.

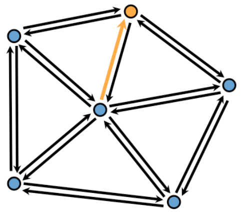
Liste der Nachbarn

1. Start at vertex



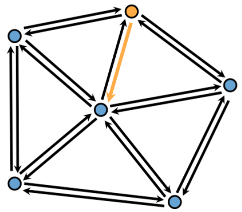
Liste der Nachbarn

1. Start at vertex
2. Outgoing halfedge



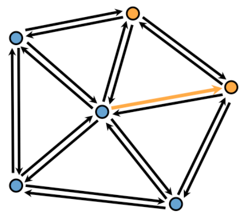
Liste der Nachbarn

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge



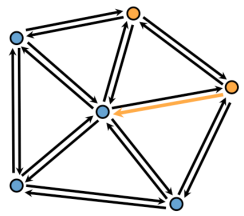
Liste der Nachbarn

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge



Liste der Nachbarn

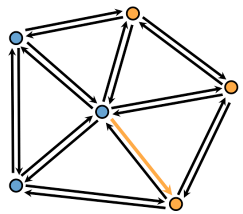
1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite halfedge



Liste der Nachbarn

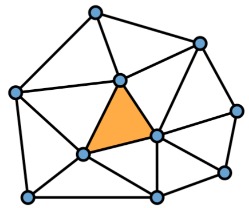
1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite halfedge
6. Next halfedge

...



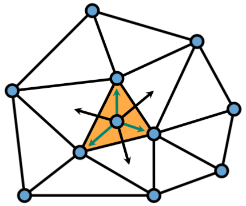
Simplex-Strukturen

- Simplices sind die einfachsten k-dim. Strukturen
  – Punkt
  – Strecke
  – Dreieck
  – Tetraeder
  – ...



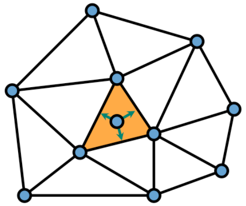
Simplex-Strukturen

- Ein k-Simplex hat
  - k+1 Knoten
  - k+1 Nachbarn



Simplex-Strukturen

- Seine k+1 Flächen sind (k−1)-Simplices



Simplex-Strukturen

- class k_Simplex {
  Node Vertices[1..k+1]
  (k−1)_Simplex Faces[1..k+1]
  }
- Darstellung beliebig dimensionaler Komplexe mit disjunkten Zellen
- Navigation / Aufzählung schwieriger