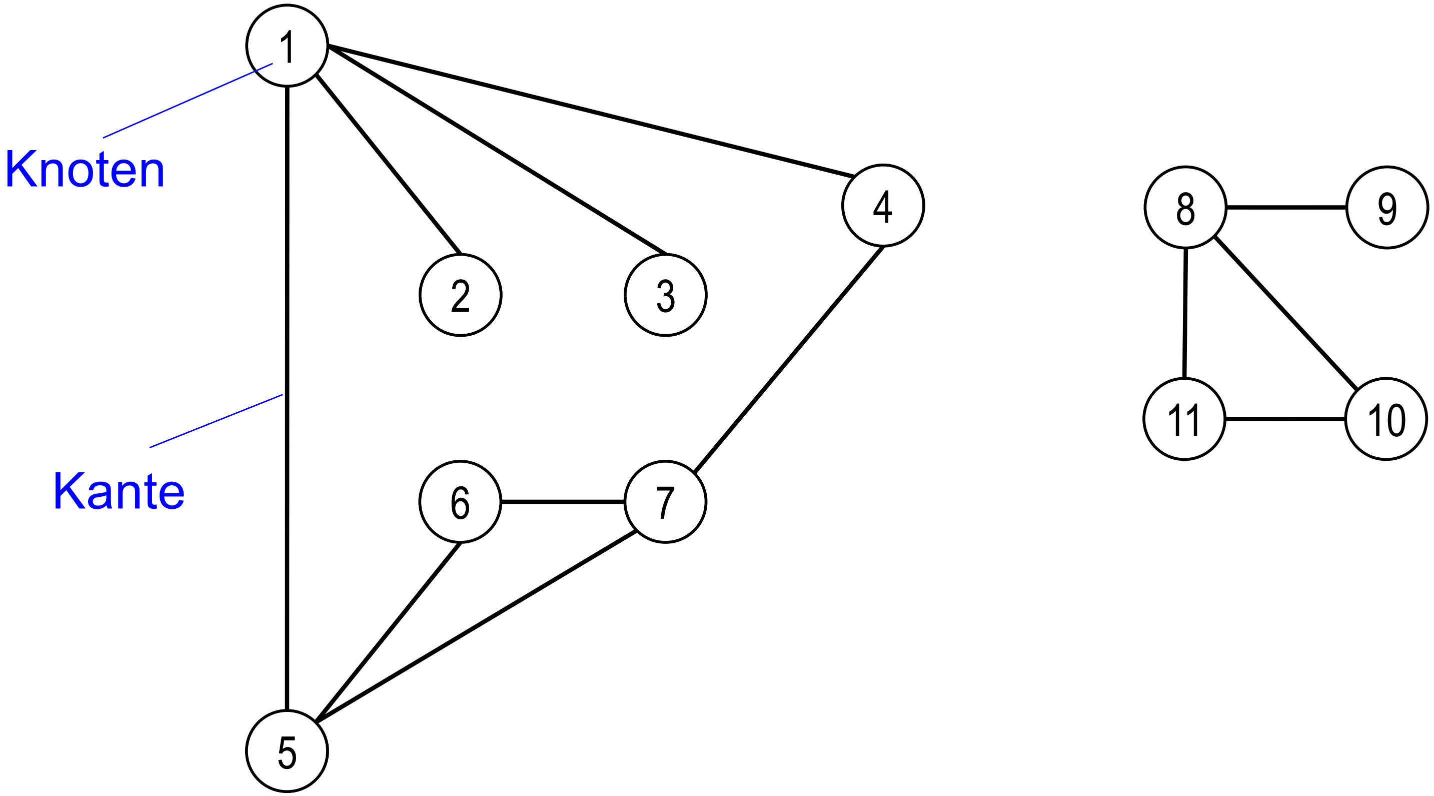
7. Einführung in Graphen

- Anwendungen
- Definitionen
- Implementierung
  - Adjazenzmatrix
  - Adjazenzliste
  - Kantenliste
  - Implementierungshinweise für Java

Graphen

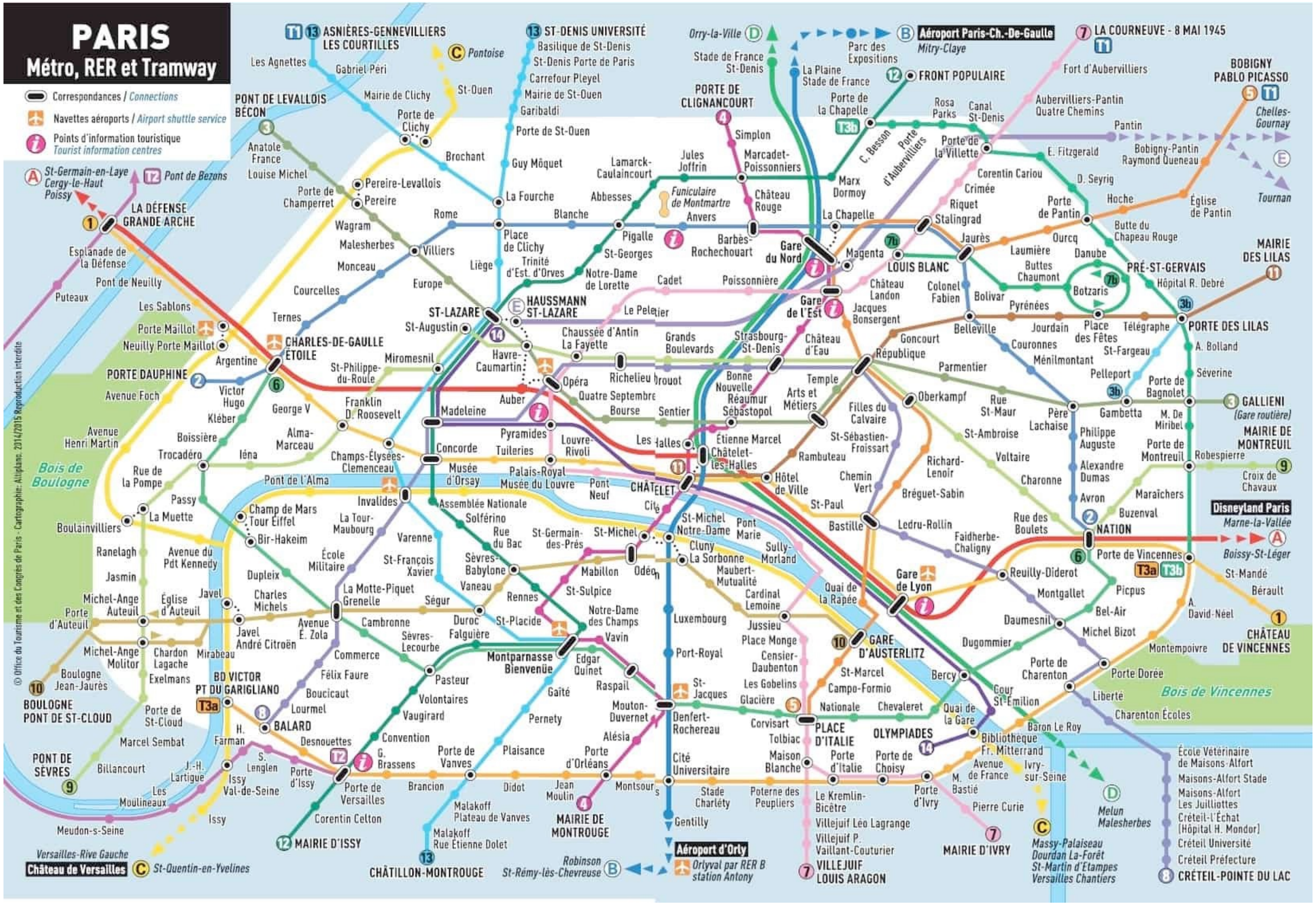
- Menge von Knoten mit Kanten
- Unzählige Anwendungen
- Viele Algorithmen



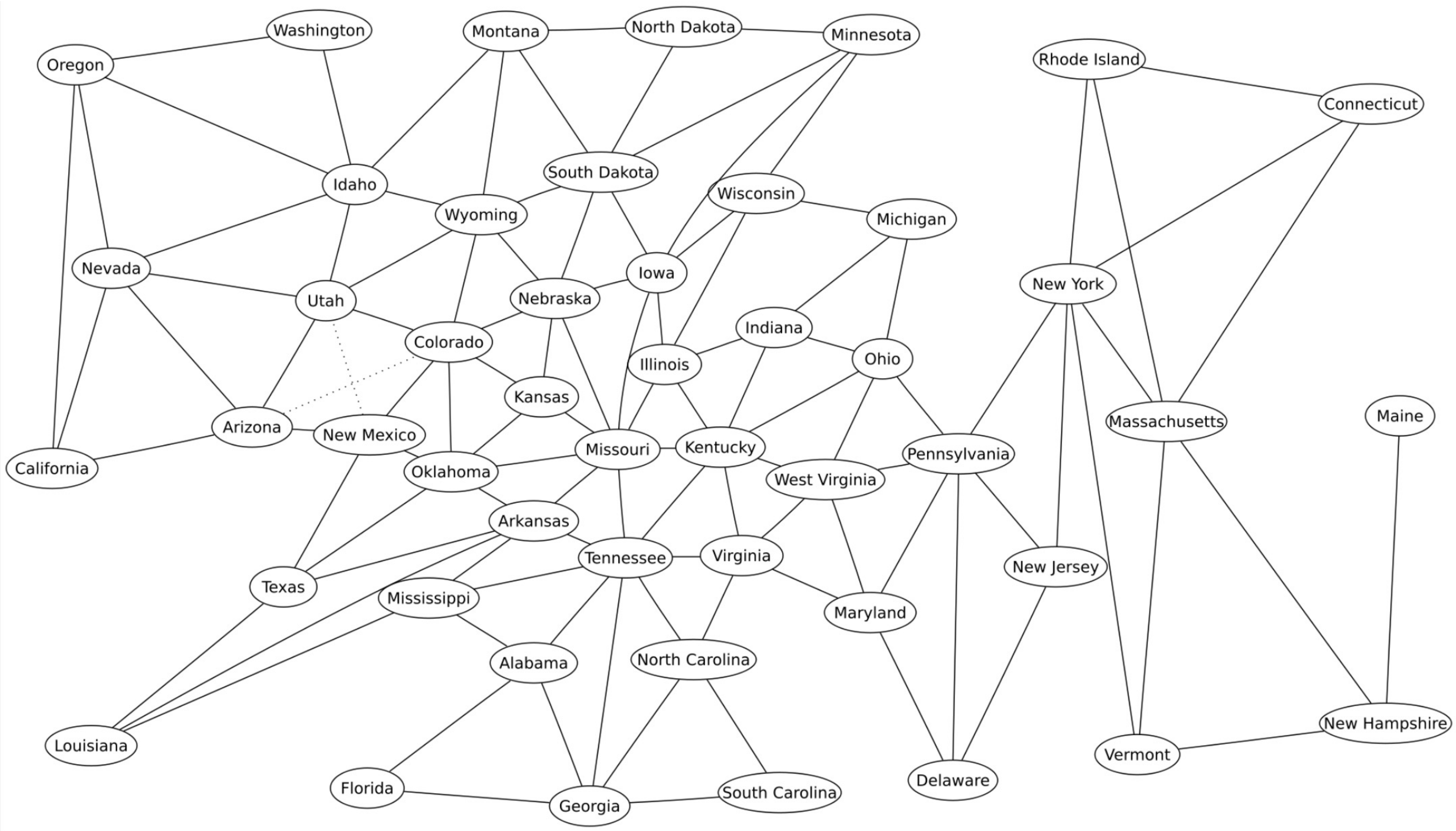
Unzählige und vielseitige Anwendungen

Graph	Knoten	Kante
Kommunikation	Rechner, Telefone	Glasfaser, Funk
Software	Module	Abhängigkeiten
Wirtschaft	Unternehmen	Transaktionen
Straßenkarten	Orte	Straßen
Internet	Web-Seiten	Links
Soziale Netzwerke	Personen	Bekanntschaften
Neuronale Netze	Neuronen	Synapsen
Moleküle	Atome	Bindungen
Gebäudeplan	Räume	Türen
Produktionsplanung	Aktivitäten	Vorrangbeziehungen
Greifarmroboter	Gelenke	Glieder
...	...	...

Metroplan Paris



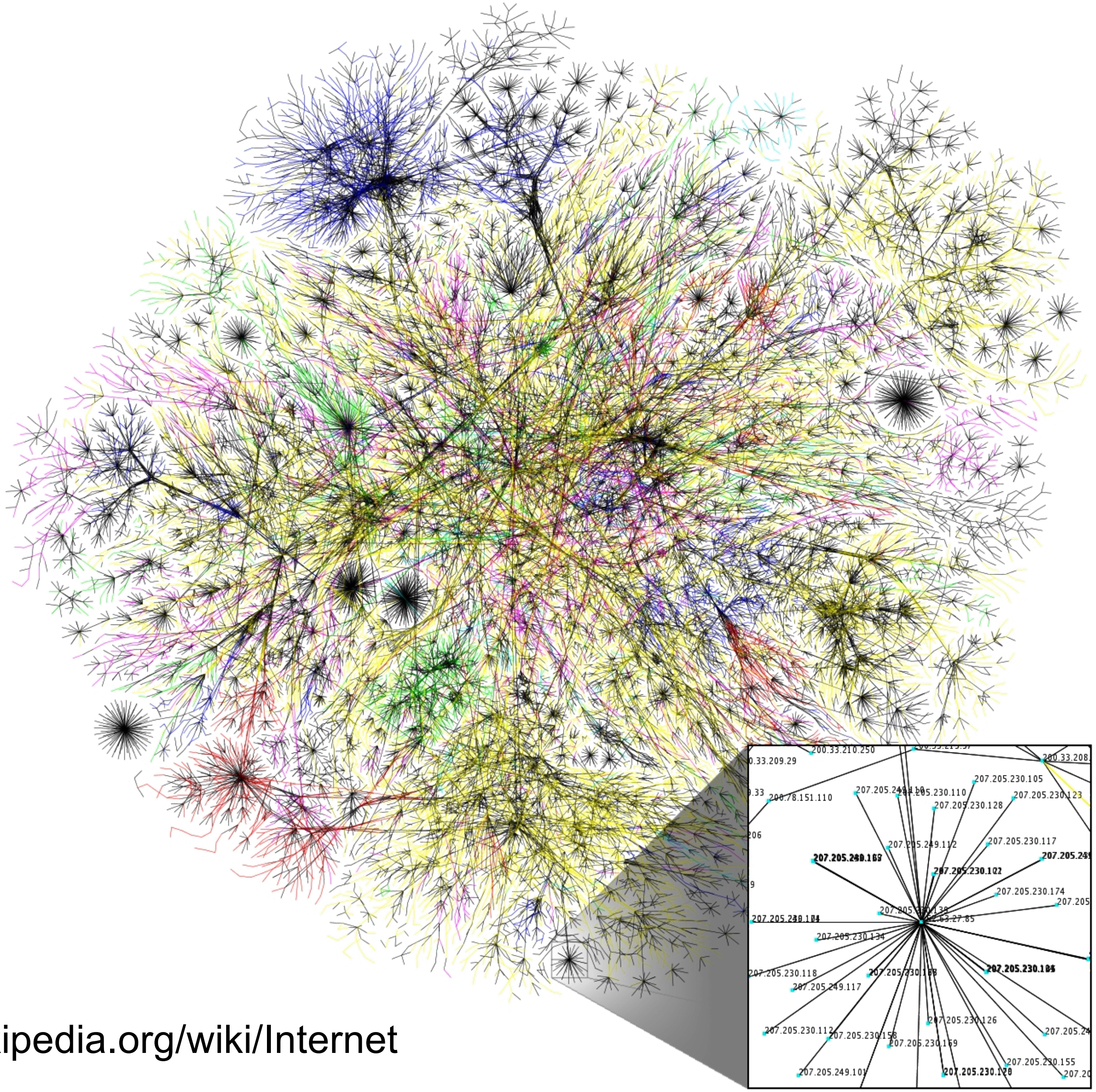
Nachbarschaftsgraph (USA)



Facebook‘s Social Network Graph

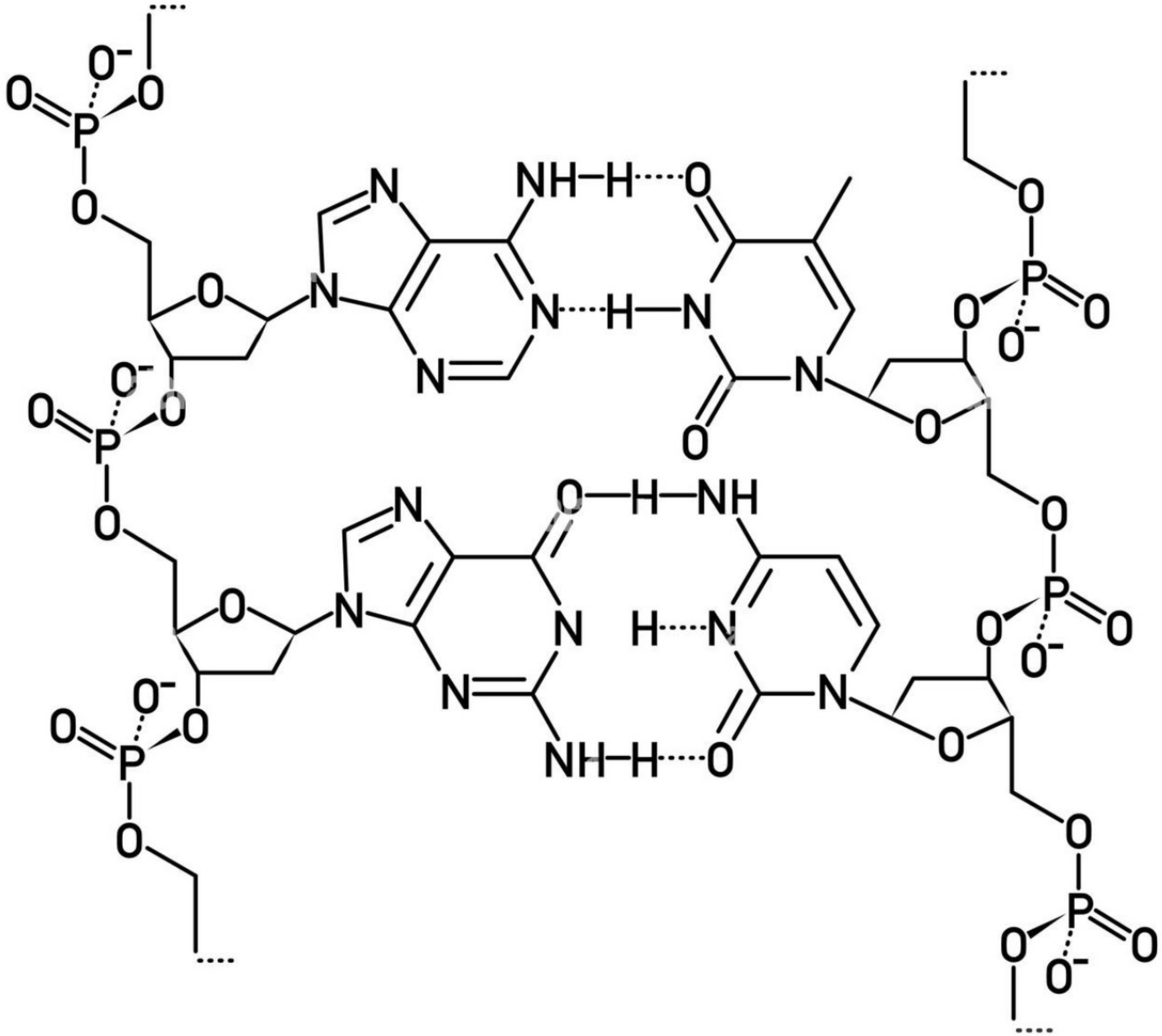
https://engineering.fb.com/2010/12/13/core-data/visualizing-friendships/

Internet Routing Paths

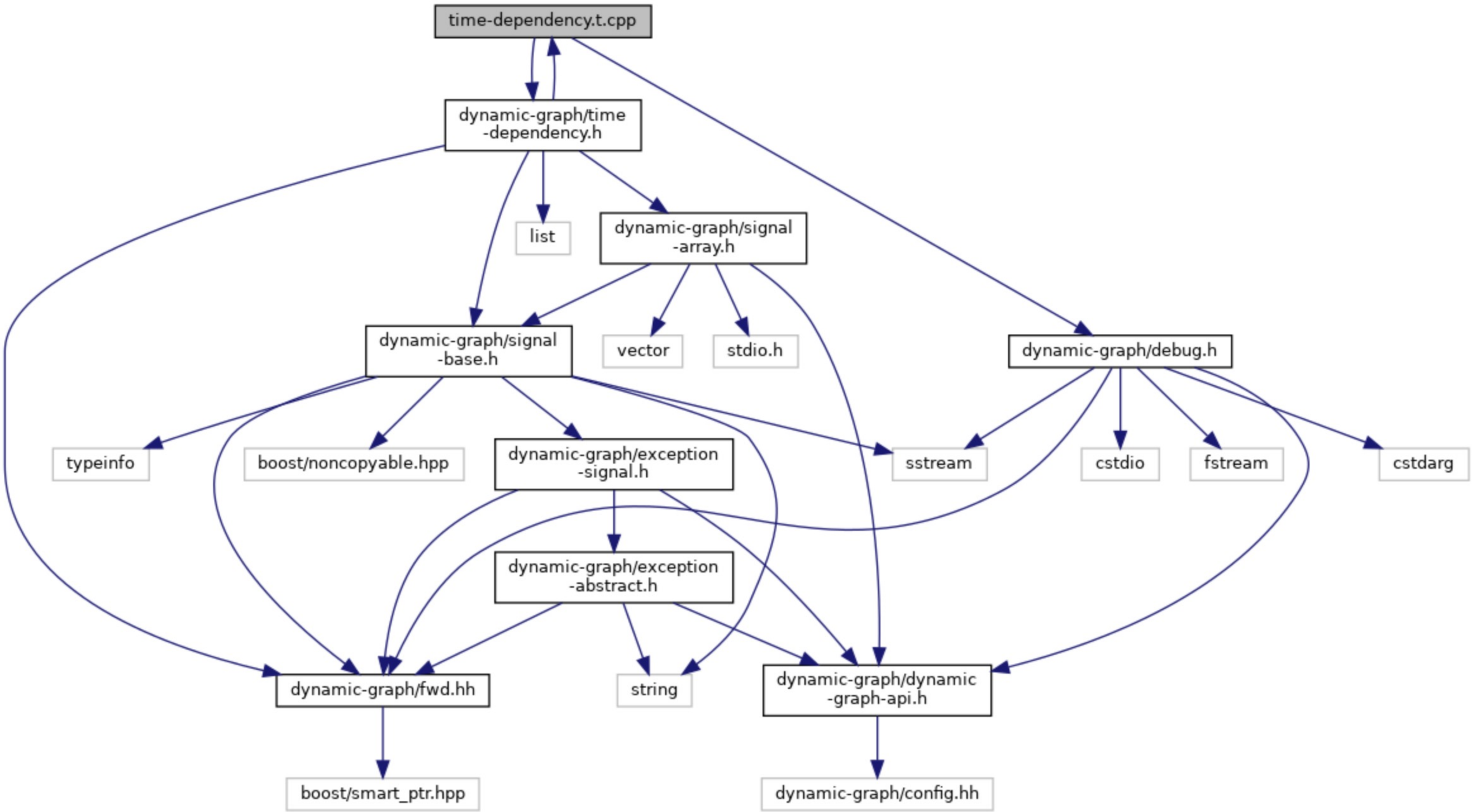


https://en.wikipedia.org/wiki/Internet

DNA Molekül-Struktur

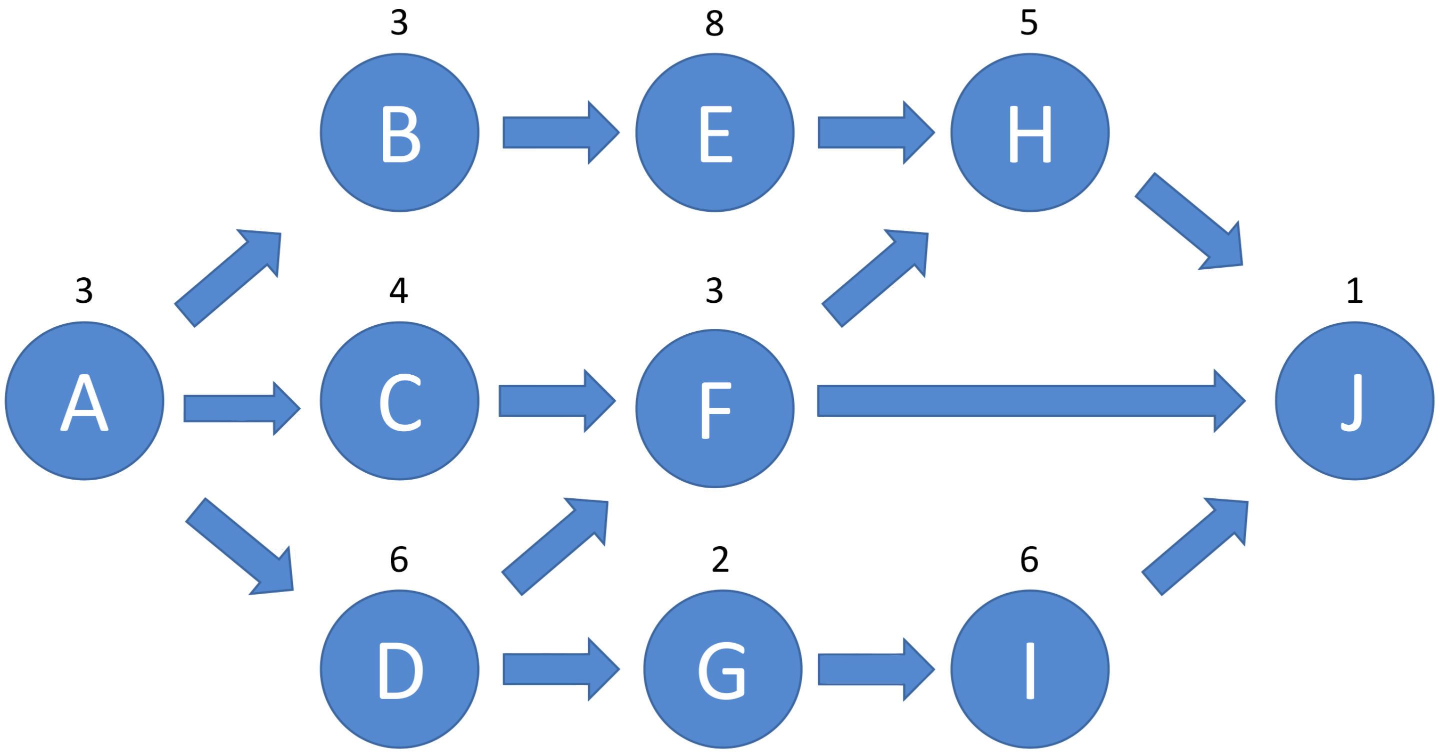


Include Dependency Graph



http://docs.ros.org/en/melodic/api/dynamic-graph/html/time-dependency_8t_8cpp.html

Produktionsplanung



Tätigkeit	Beschreibung	Ausführungszeit
A	Grundplatte	3
B	Achsen	3
C	Motor	4
D	Getriebe	6
E	Räder	8
F	Lenkstange	3
G	Keilriemen	2
H	Karosserie	5
I	Lichtanlage	6
J	Scheiben	1

https://plan-b-bremen.com/de/loesungen/montageplanung/design-for-assembly-montagefreundliche-produktgestaltung/vorranggraph

Laufroboter

Transformationsgraph:
- Knoten = Gelenke
- Kanten = Glieder

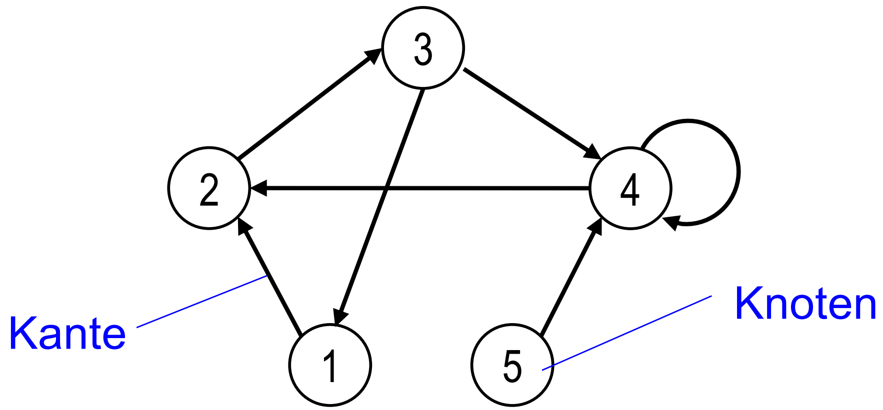
https://ndion.de/de/boston-dynamics-laesst-roboter-tanzen/

7. Einführung in Graphen

- Anwendungen
- Definitionen
- Implementierung
  - Adjazenzmatrix
  - Adjazenzliste
  - Kantenliste
  - Implementierungshinweise für Java

Gerichteter Graph

- Gerichteter Graph G = (V, E), wobei:
  - V ist eine Menge von Knoten (engl. vertices)
  - $E \subseteq V \times V$ ist eine Menge von Kanten (engl. edges).
- Eine Kante ist ein Paar von Knoten (v, w) und ist gerichtet.
  (v, w) geht von Knoten v nach Knoten w.
- Kanten der Form (v,v) heißen Schlingen.

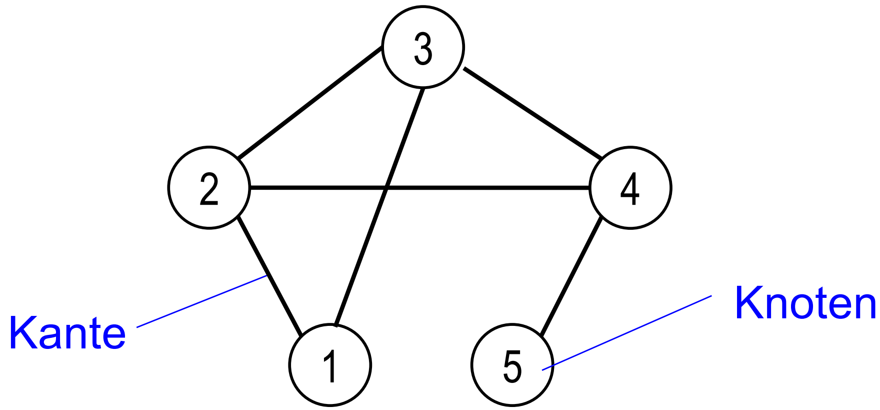


V = { 1, 2, 3, 4, 5}

E = { (1,2), (2,3), (3,1), (3,4), (4,2), (4,4), (5,4) }

Ungerichteter Graph

- Ungerichteter Graph G = (V, E), wobei:
  - V ist eine Menge von Knoten (engl. vertices)
  - $E \subseteq \{\{v,w\} \mid v \neq w \text{ und } v, w \in V\}$ ist eine Menge von Kanten (engl. edges).
- Eine Kante ist eine 2-elementige Menge von Knoten {v,w} und ist ungerichtet.
- Keine Schlingen!
- Die Darstellung soll jedoch einfach bleiben. Daher werden wir sowohl für gerichtete als auch ungerichtete Kanten dieselbe Notation (v,w) verwenden.

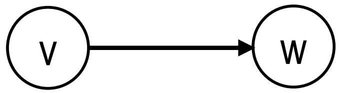


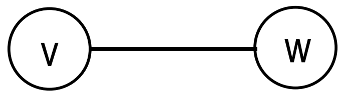
V = { 1, 2, 3, 4, 5}

E = { (1,2), (2,3), (3,1), (3,4), (4,2), (5,4) }

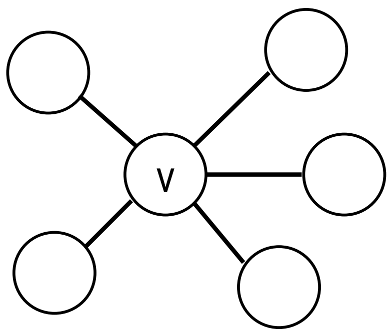
Adjazenz

- Ein Knoten w ist adjazent zu (folgt auf) einem Knoten v,
  falls (v,w) eine Kante des gerichteten bzw. ungerichteten Graphen ist.

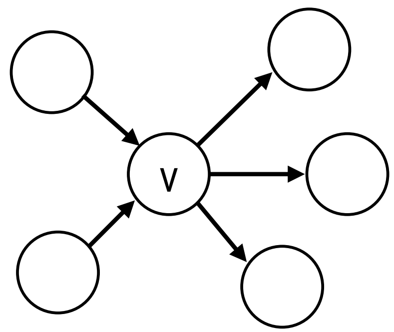




- Bei einem ungerichteten Graphen ist die Adjazenzbeziehung immer symmetrisch.
  Bei einem gerichteten Graphen ist die Adjazenzbeziehung im allg. unsymmetrisch.
- Bei einer gerichteten Kante (v,w) wird w auch Nachfolger von v und
  v Vorgänger von w genannt.
- Bei einer ungerichteten Kante (v,w) heißen v und w auch Nachbarn.



Knoten v hat 5 Nachbarn

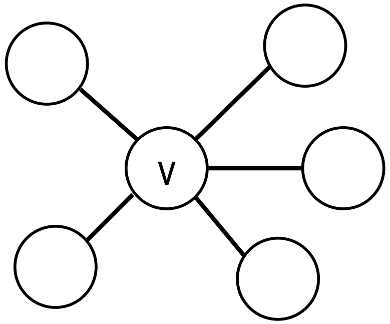


Knoten v hat 2 Vorgänger und 3 Nachfolger

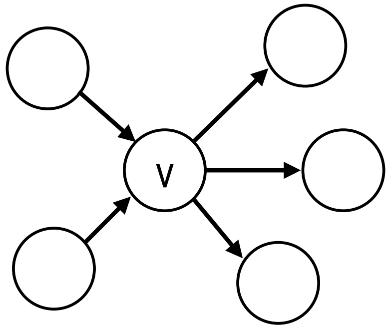
Grad

- In einem ungerichteten Graphen ist der Grad eines Knoten
die Anzahl seiner Nachbarn.

- Bei einem gerichteten Graphen wird die Anzahl der Vorgänger Eingangsgrad
und die Anzahl der Nachfolger Ausgangsgrad genannt.



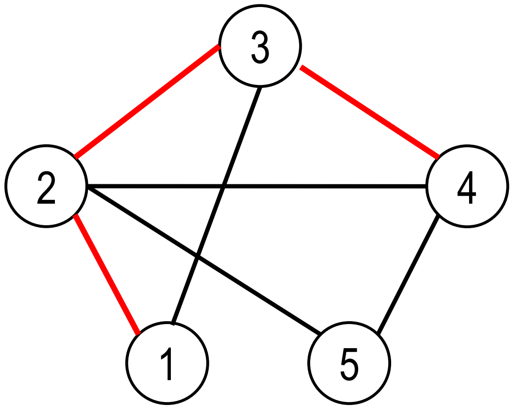
Grad von v ist damit 5.



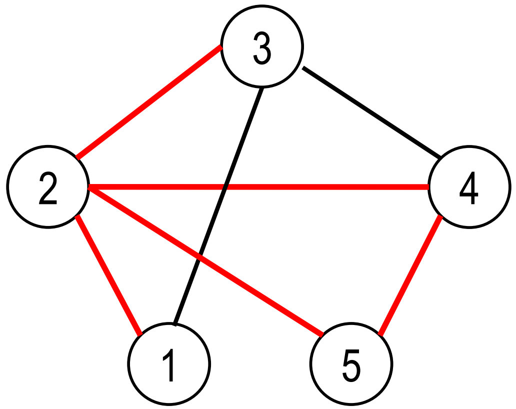
Eingangsgrad von v ist 2
und Ausgangsgrad ist 3.

Weg

- Ein Weg (engl. path) ist eine Folge von Knoten $v_1, v_2, \ldots, v_n$, wobei $n \geq 1$ und (v_i, v_{i+1}) für $1 \leq i < n$ Kanten im Graphen sind.
- Die Länge des Weges $v_1, v_2, \ldots, v_n$ ist gleich der Anzahl der Kanten und damit n-1. Ein Weg mit nur einem Knoten hat die Länge 0.
- Ein Weg heißt einfacher Weg, falls alle Knoten bis auf Anfangs- und Endknoten unterschiedlich sind.



Ein einfacher Weg
der Länge 3: 1, 2, 3, 4.



Ein (nicht einfacher) Weg der
Länge 5: 1, 2, 5, 4, 2, 3

(Knoten 2 wird zweimal besucht)

Zyklus in einem gerichteten Graphen

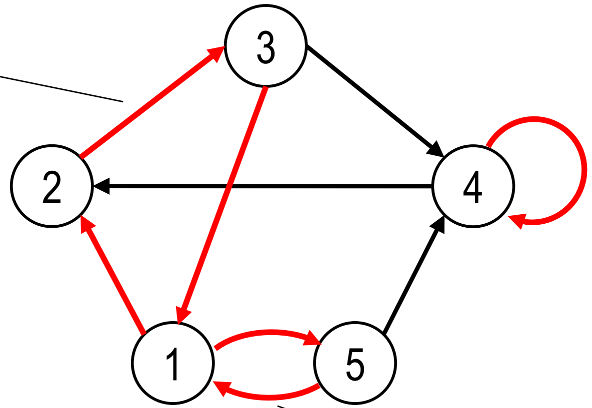
- Ein Weg $v_1, v_2, \ldots, v_n$ mit Länge ≥ 1 (d.h. wenigstens eine Kante) in einem gerichteten Graphen heisst Zyklus (cycle), falls Anfangsknoten v_1 und Endknoten v_n identisch sind.
- Zwei Zyklen sind gleich, falls sich ihre Wege (jeweils ohne Endknoten) durch eine zyklische Verschiebung unterscheiden.
- Ein Zyklus heisst einfach, falls alle Knoten bis auf Anfangs- und Endknoten unterschiedlich sind.

Zyklus der Länge 3:

1, 2, 3, 1

Derselbe Zyklus, der sich nur um eine zyklische Verschiebung unterscheidet:

2, 3, 1, 2



Zyklus der Länge 1:
4, 4

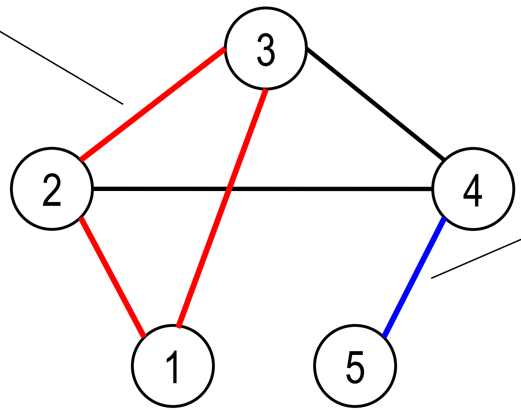
Zyklus der Länge 2:
1, 5, 1

Zyklus in einem ungerichteten Graphen

- Ein Weg $v_1, v_2, \ldots, v_n$ in einem ungerichteten Graph heisst Zyklus (cycle), falls Anfangsknoten v_1 und Endknoten v_n identisch sind und alle Kanten unterschiedlich sind.
- Folgerung: Länge eines Zyklus muss ≥ 3 sein (d.h. wenigstens 3 Kanten).
- Zwei Zyklen sind gleich, falls sich ihre Wege (jeweils ohne Endknoten) durch eine zyklische Verschiebung unterscheiden.
- Ein Zyklus heisst einfach, falls alle Knoten bis auf Anfangs- und Endknoten unterschiedlich sind.

Zyklus der Länge 3:

1, 2, 3, 1



Weg der Länge 2:

5, 4, 5

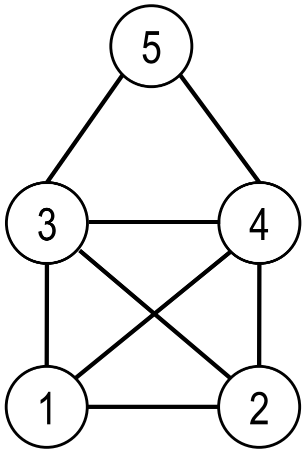
aber kein Zyklus,
da Kanten nicht unterschiedlich.

Teilgraph

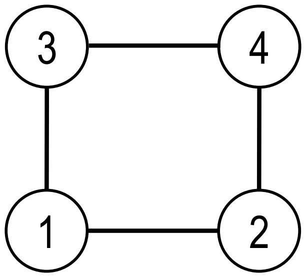
- Ein Graph G' = (V', E') ist ein Teilgraph von Graph G = (V, E), falls $V' \subseteq V$ und $E' \subseteq E$.

- Enthält darüber hinaus ein Teilgraph G' = (V', E') alle Kanten aus E, die Knoten aus V' verbinden, dann heißt G' der durch V' induzierte Teilgraph von G, d.h.

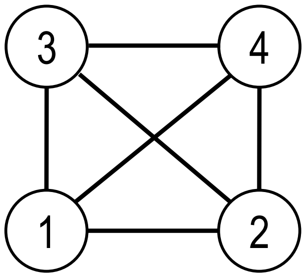
  $$E' = \{(u,v) \in E \; / \; u,v \in V'\}$$



Graph G



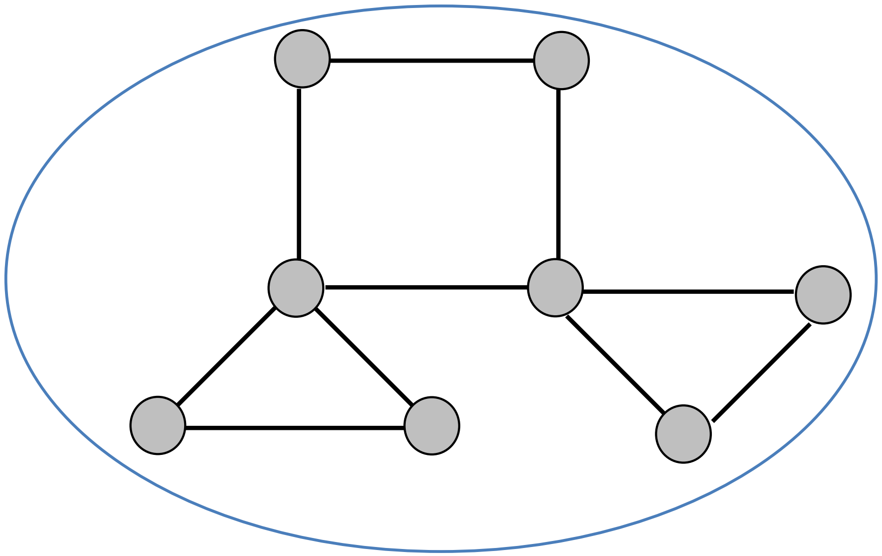
Teilgraph G' von G



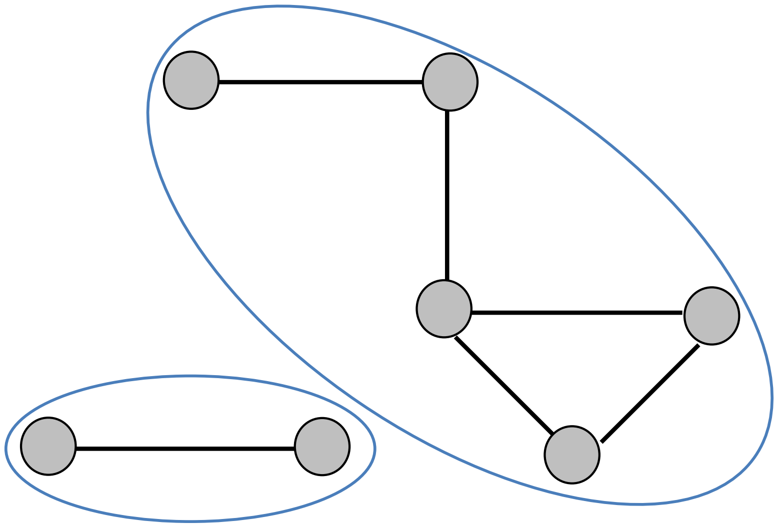
Der durch V' = {1, 2, 3, 4} induzierte Teilgraph von G

Zusammenhang bei ungerichteten Graphen

- Ein ungerichteter Graph heißt zusammenhängend (connected), falls es von jedem Knoten einen Weg zu jedem anderen Knoten gibt.
- Eine Zusammenhangskomponente eines ungerichteten Graphen G ist ein maximal zusammenhängender Teilgraph.



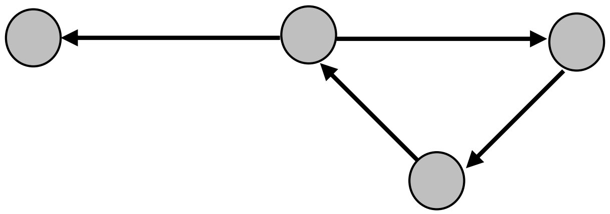
Graph ist zusammenhängend und besteht daher aus nur einer Zusammenhangskomponente



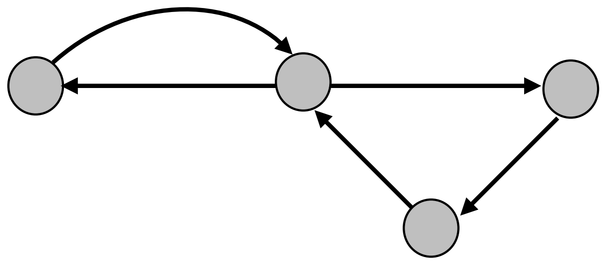
Graph ist nicht zusammenhängend und besteht aus genau zwei Zusammenhangskomponenten.

Starker und schwacher Zusammenhang bei gerichteten Graphen

- Ein gerichteter Graph heißt stark zusammenhängend (strongly connected), falls es von jedem Knoten einen Weg zu jedem anderen Knoten gibt.
- Wenn bei einem gerichteten Graphen G der entsprechende ungerichtete Graph zusammenhängend ist, dann wird G auch schwach zusammenhängend genannt (weakly connected).



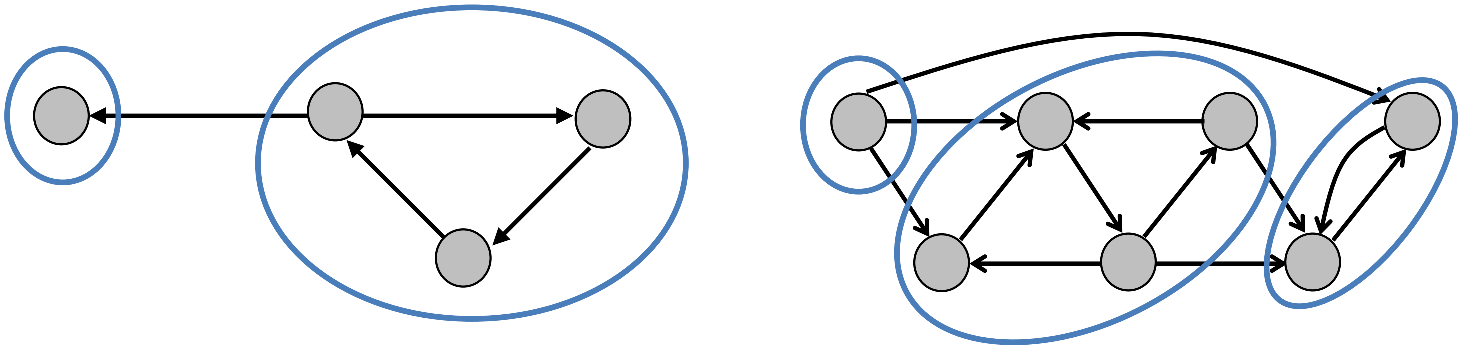
Gerichteter Graph, der schwach aber nicht stark zusammenhängend ist.



Gerichteter Graph, der stark zusammenhängend ist.

Zusammenhangskomponenten bei gerichteten Graphen

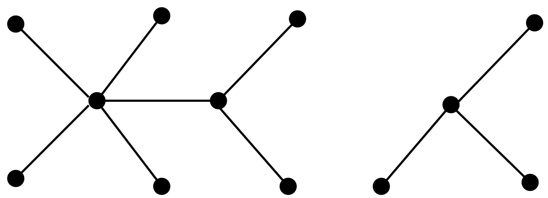
- Eine starke bzw. schwache Zusammenhangskomponente eines gerichteten Graphen G ist ein maximal stark bzw. schwach zusammenhängender Teilgraph.



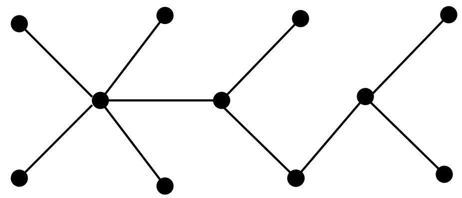
Gerichtete Graphen mit ihren
starken Zusammenhangskomponenten.

Wald und Baum

- Ein ungerichteter und azyklischer (d.h. zyklenfreier) Graph heißt auch Wald.
- Ein zusammenhängender Wald heißt auch Baum.



Wald



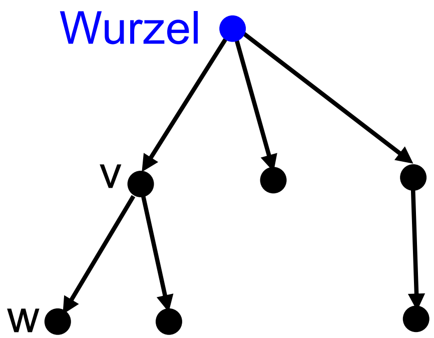
Baum

- Für jeden Baum G = (V, E) gilt:

$$|E| = |V| - 1$$

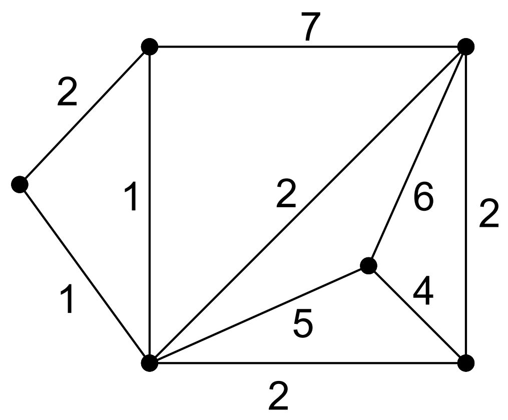
Wurzelbaum

- Ein Wurzelbaum ist ein gerichteter, azyklischer Graph, bei dem genau ein Knoten – die sogenannte Wurzel – den Eingangsgrad 0 hat und alle anderen Knoten den Eingangsgrad 1 haben.
- Bei einer Kante (v,w) heisst v auch Elternknoten von w und w Kindknoten von v.
- Knoten ohne Kinder heißen auch Blätter.
- In der graphischen Darstellung zeigen die Kanten üblicherweise nach unten.
- Die in Kapitel 3 und 4 eingeführten Suchbäume sind Wurzelbäume.



Gewichteter Graph

- Ist jeder Kante (v,w) eine reelle Zahl c(v,w) als Kosten oder Gewicht zugeordnet, spricht man von einem gewichteten Graphen.
- Sind darüber hinaus die Gewichte ≥ 0, dann heißt der Graph auch Distanzgraph.

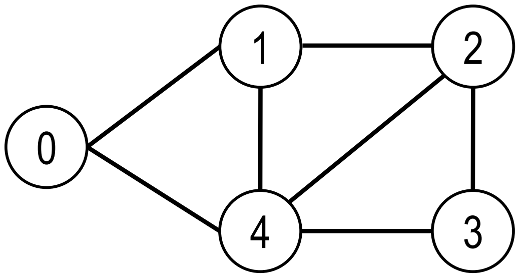


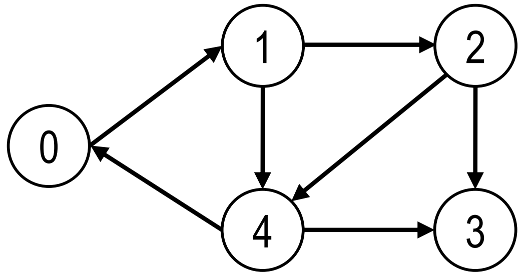
7. Einführung in Graphen

- Anwendungen
- Definitionen
- Implementierung
  - Adjazenzmatrix
  - Adjazenzliste
  - Kantenliste
  - Implementierungshinweise für Java

Adjazenzmatrix für ungewichtete Graphen

$$A[i][j] = \begin{cases} 1, \text{ falls es eine Kante von } v_i \text{ und nach } v_j \text{ gibt} \\ 0 \text{ sonst} \end{cases}$$





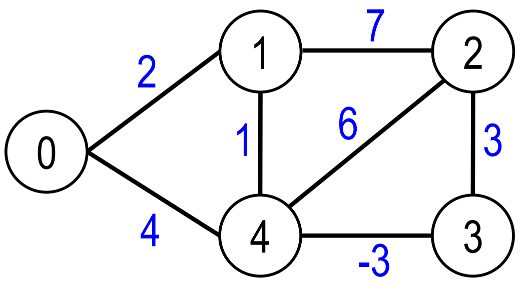
	[0]	[1]	[2]	[3]	[4]
[0]	0	1	0	0	1
[1]	1	0	1	0	1
[2]	0	1	0	1	1
[3]	0	0	1	0	1
[4]	1	1	1	1	0

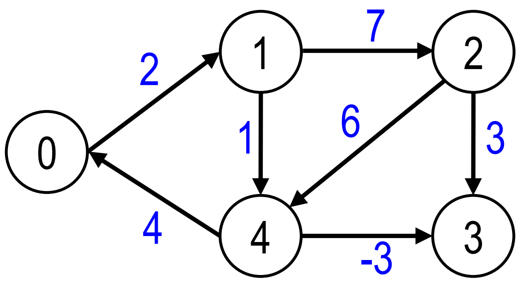
	[0]	[1]	[2]	[3]	[4]
[0]	0	1	0	0	0
[1]	0	0	1	0	1
[2]	0	0	0	1	1
[3]	0	0	0	0	0
[4]	1	0	0	1	0

Beachte: Die Adjazenzmatrizen sind bei einem ungerichteten Graphen symmetrisch und bei einem gerichteten Graphen im allgemeinen unsymmetrisch.

Adjazenzmatrix für gewichtete Graphen

$$A[i][j] = \begin{cases} c(v,w), & \text{falls es eine Kante von Knoten v nach w gibt} \\ \infty & \text{sonst} \end{cases}$$





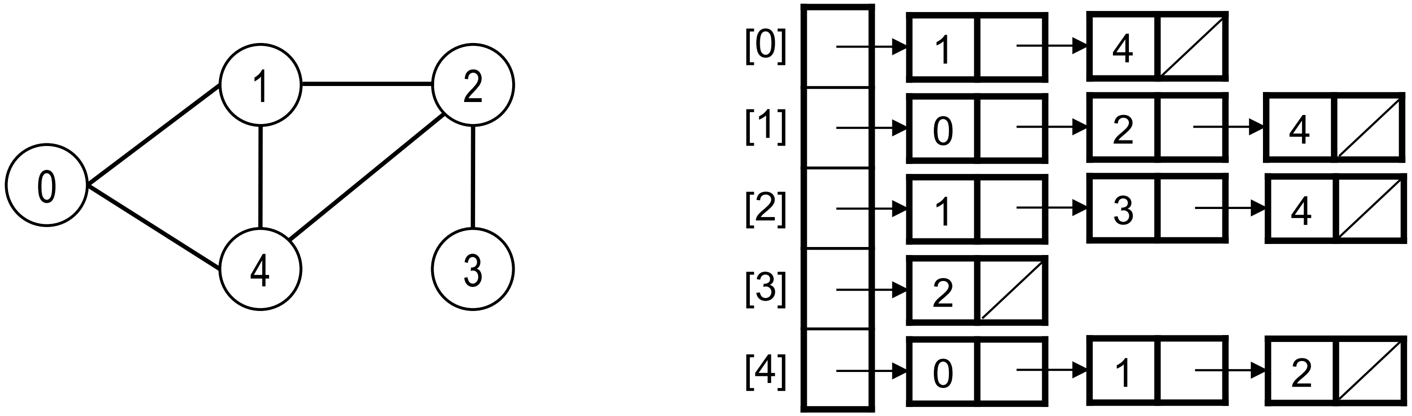
	[0]	[1]	[2]	[3]	[4]
[0]	∞	2	∞	∞	4
[1]	2	∞	7	∞	1
[2]	∞	7	∞	3	6
[3]	∞	∞	3	∞	-3
[4]	4	1	6	-3	∞

	[0]	[1]	[2]	[3]	[4]
[0]	∞	2	∞	∞	∞
[1]	∞	∞	7	∞	1
[2]	∞	∞	∞	3	6
[3]	∞	∞	∞	∞	∞
[4]	4	∞	∞	-3	∞

Beachte: Die Adjazenzmatrizen sind bei einem ungerichteten Graphen symmetrisch und bei einem gerichteten Graphen im allgemeinen unsymmetrisch.

Adjazenzliste bei ungerichteten Graphen

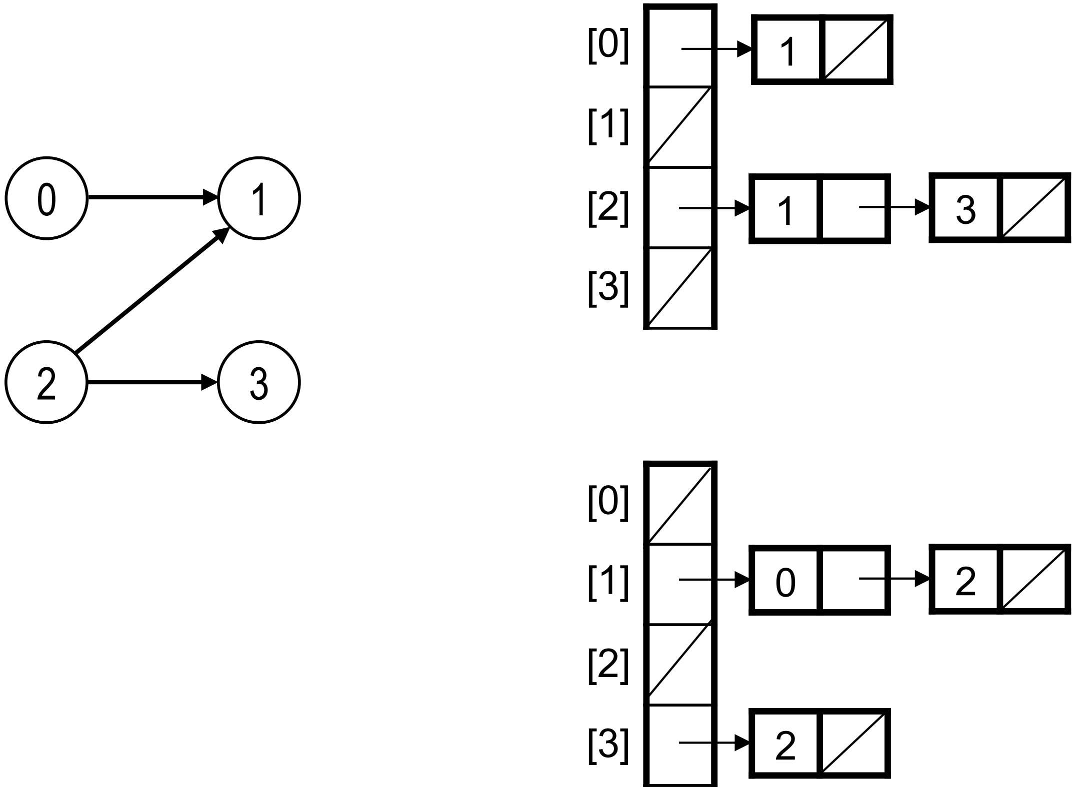
- Speichere für jeden Knoten alle Nachbarknoten in eine linear verkettete Liste (Adjazenzliste).
- Kanten werden bei beiden Endknoten eingetragen.



- Bei einem gewichteten Graphen wird noch zusätzlich das Kantengewicht eingetragen.
- Um die Dynamisierbarkeit der Datenstruktur zu verbessern, kann statt eines Feldes auch eine dynamische Datenstruktur wie ein binärer Suchbaum verwendet werden.

Adjazenzliste bei gerichteten Graphen

- Bei einem gerichteten Graphen werden zwei getrennte Listen verwaltet: eine Vorgänger-Liste und eine Nachfolger-Liste.



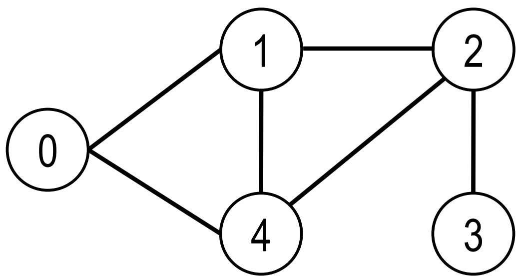
Nachfolger-Liste

Vorgänger-Liste

Kantenliste

- Speichere die Menge aller Kanten in einer sortierten Suchstruktur (sortiertes Feld oder Suchbaum).
- Aus einer linearen Ordnung für Knoten ergibt sich eine lineare Ordnung für Kanten:

$$(v, v') \leq (w, w') \text{ falls } v < w \text{ oder } v = w \text{ und } v' \leq w'$$



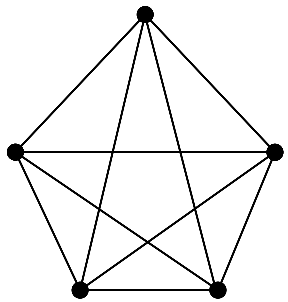
	[0]	[1]	[2]	…								[11]
i	0	0	1	1	1	2	2	2	3	4	4	4
j	1	4	0	2	4	1	3	4	2	0	1	2

Alle Nachbarknoten zu v_2 sind effizient auffindbar.

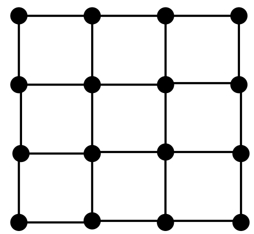
- Bei einem ungerichten Graphen wird für eine Kante von v nach w sowohl (v,w) als auch (w,v) gespeichert.
- Bei einem gerichteten Graphen wird die Kantenliste in Vorgänger- und Nachfolger-Liste aufgeteilt.

Dünn- und dichtbesetzte Graphen

- Falls $|E| = O(|V|)$ (dünnbesetzter Graph; sparse graph), dann ist die Speicherung als Adjazenzliste oder Kantenliste Speicherplatz sparender und effizienter in der Laufzeit.
- Falls $|E| = O(|V|^2)$ (dichtbesetzter Graph; dense graph), dann ist die Speicherung als Adjazenzmatrix Speicherplatz sparender und effizienter in der Laufzeit.



- Beispiel für dichtbesetzten Graph: Vollständiger Graph (alle Knotenpaare sind durch eine Kante verbunden)
- Es gilt: $|E| = |V|*(|V|-1)/2 = O(|V|^2)$.



- Beispiel für dünnbesetzten Graph: Manhattan-Graph (Graph für Manhattan mit Kreuzungen als Knoten und Strassen als Kanten)
- Es gilt: $|E| \approx 2*|V| = O(|V|)$.

7. Einführung in Graphen

- Anwendungen
- Definitionen
- Implementierung
  - Adjazenzmatrix
  - Adjazenzliste
  - Kantenliste
  - Implementierungshinweise für Java

Adjazenzmatriz als zweidimensionales Feld

- Knoten sind durchnummeriert: 0, 1, 2, ..., n-1
- Nehme Knotennummer als Index eines zweidimensionalen Felds
- Beispiel: ungerichteter und gewichteter Graph

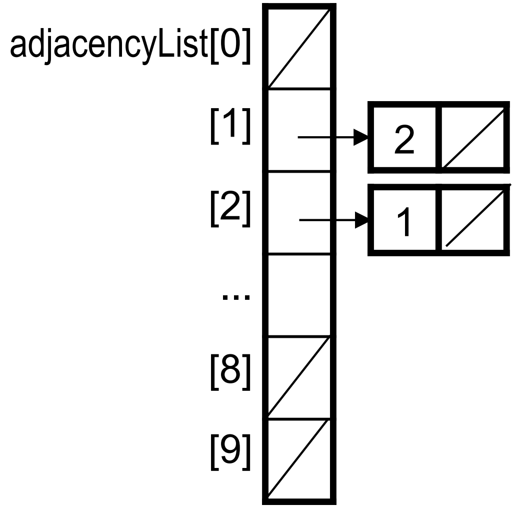
```
int n = 10; // Anzahl der Knoten
double [ ][ ] adjcencyMatrix = new double[n][n];
adjcencyMatrix[1][2] = 3.5;          // Kante von 1 nach 2 mit Gewicht 3.5 eintragen
adjcencyMatrix[2][1] = 3.5;          // Kante von 2 nach 1 mit Gewicht 3.5 eintragen
```

- Bei ungewichteten Graphen sind Gewichte 0 oder 1.
- Bei gerichteten Graphen ist Matrix i.a. nicht symmetrisch.

Feld oder Liste von Adjazenzlisten

- Knoten sind durchnummeriert: 0, 1, 2, ..., n-1.
- Beispiel: ungerichteter Graph:

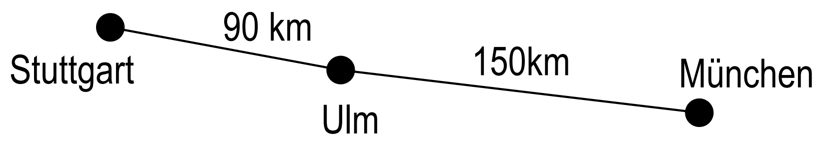
```
int n = 10; // Anzahl der Knoten
List<List<Integer>> adjacencyList = new ArrayList<>(n);
for (int i = 0; i < n; i++)
    adjacencyList.add( new LinkedList<>() );
adjacencyList.get(1).add(2);        // Kante von 1 nach 2 eintragen
adjacencyList.get(2).add(1);        // Kante von 2 nach 1 eintragen
```



- Gewichte können bei den Knoten abgespeichert werden.
- Bei gerichteten Graphen werden Nachfolger und Vorgänger in separate Listen gehalten.
- Statt die adjazenten Knoten in Listen zu speichern, ließen sich sich auch Sets (bei ungewichteten Graphen) oder Maps (bei gewichteten Graphen) einsetzen.

Interne Nummerierung für Graph-Knoten

- In der Praxis kommt es häufig vor, dass die Graph-Knoten nicht von 0 bis n-1 nummeriert sind, sondern beispielsweise als Strings gegeben sind.



- Die Schlüssel können dann mit Hilfe einer Map (TreeMap oder HashMap) in eine interne Nummerierung von 0 bis n-1 umgerechnet werden:
  - Stuttgart → 0
  - Ulm → 1
  - München → 2
- Die inverse Umrechnung von interner Nummerierung in den Schlüssel lässt sich mit einem einfachen Feld realisieren:
  - Stuttgart ← 0
  - Ulm ← 1
  - München ← 2
- Mittels der internen Nummerierung lässt sich dann eines der indizierten Zugriffsverfahren von der vorhergehenden Seite verwenden.

Graph als Map

- Eine Map ordnet jedem Knoten eine Adjazenzliste zu.
- Bei gewichteten Graphen sind die Adjazenzlisten ebenfalls Maps und ordnen jedem adjazenten Knoten ein Gewicht zu.
- Bei ungewichteten Graphen können dagegen einfachheitshalber Sets verwendet werden.
- Knotentyp V ist generisch und kann z.B. Integer oder String sein..
- Maps können entweder HashMaps oder TreeMaps sein (siehe Beispiel nächste Seite).

```
Map< V, Map<V, Double> > gewichteterGraph = ...
```

Adjazente Knoten

```
Map< V, Set<V> > ungewichteterGraph = ...
```

Beispiel: ungerichteter und gewichteter Graph als TreeMap

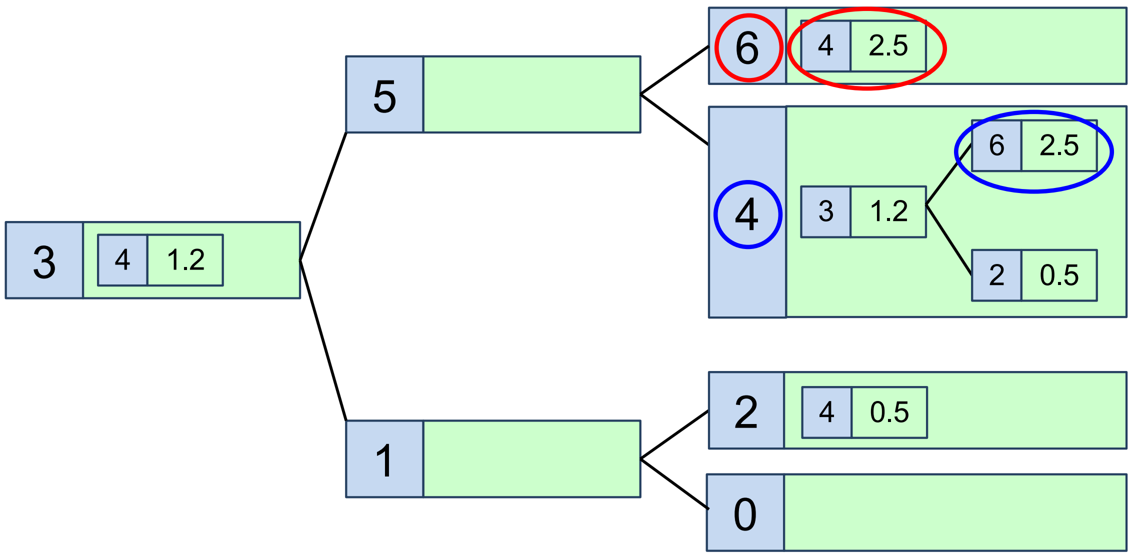
```
int n = 7;         // Anzahl Knoten
Map<Integer, Map<Integer, Double>> g = new TreeMap<>();   // ungerichteter und gewichteter Graph

for (int i = 0; i < n; i++)
        g.put(i, new TreeMap<>());

g.get(4).put(3, 1.2);        // Kante von 4 nach 3 mit Gewicht 1.2 eintragen
g.get(3).put(4, 1.2);

g.get(4).put(6, 2.5);        // Kante von 4 nach 6 mit Gewicht 2.5 eintragen
g.get(6).put(4, 2.5);

g.get(4).put(2, 0.5);        // Kante von 4 nach 2 mit Gewicht 0.5 eintragen
g.get(2).put(4, 0.5);
```



- Graph als TreeMap
- Mit Knoten als Schlüssel und Adjazenzlisten als Nutzdaten
- Die Adjazenzlisten sind ebenfalls TreeMaps.

Darstellung der Algorithmen in Pseudo-Code (1)

- Um einfach und von der konkreten Datenstruktur für Graphen unabhängig zu bleiben, wird bei der Formulierung der Algorithmen folgender Pseudo-Code benutzt:

- Ungerichtete und gerichtete Graphen:

```
for ( jeden adjazenten Knoten w von v ) {
    …
}
```

```
for ( jeden Knoten v ) {
    …
}
```

```
for ( jede Kante (v,w) ) {
    …
}
```

- Ungerichtete Graphen:

```
for ( jeden Nachbarn w von v ) {
    …
}
```

- Gerichtete Graphen:

```
for ( jeden Nachfolger w von v ) {
    …
}
```

```
for ( jeden Vorgänger w von v ) {
    …
}
```

Darstellung der Algorithmen in Pseudo-Code (2)

- Die hier in Pseudo-Code dargestellten Schleifen lassen sich in Java mit den besprochenen Datenstrukturen (Adjazenzmatrix, Adjazenzliste und Kantenliste) einfach realisieren.

Pseudo-Code:

```
for ( jeden Knoten v ) {
        …
}
```

Java:

```
// Map-basierte Adjazenzliste für ungewichtete Graphen:
Map<Vertex, Set<Vertex>> adjacencyList  = new TreeMap<>();

for (Vertex v : adjacencyList.keySet() ) {
        ...
}
```

```
for ( jeden Nachfolger w von v ) {
        …
 }
```

```
// adjacencyList  wie oben definiert

for (Vertex w : adjacencyList.get(v) ) {
        ...
}
```

Darstellung der Algorithmen in Pseudo-Code (3)

- Bei der Formulierung der Algorithmen sollen auch die Datenstrukturen möglichst einfach gehalten werden.
- Oft muss für jeden Knoten eine Information gespeichert werden.
- Im Pseudo-Code werden einfachheitshalber Felder mit Knoten als Indizierung gewählt.
- Sind die Knoten durchnummeriert, dann kann der Pseudo-Code direkt in Java übernommen werden. Bei einem beliebigen Knotentyp kann eine Map gewählt werden.

Pseudo-Code:

```
int[ ] inDegree;
...
for ( jeden Knoten v ) {
        inDegree[v] = Anzahl der Vorgänger;
        if (inDegree[v] == 0)
                ...
}

for ( jeden Nachfolger w von v ) {
        if (--inDegree[w] == 0)
                ...
}
```

Java:

```
Map<Vertex, Integer> inDegree = new ...;
...
for  ( jeden Knoten v ) {
        inDegree.put(v, Anzahl der Vorgänger);
        if (inDegree.get(v) == 0)
                ...
}

for ( jeden Nachfolger w von v ) {
        inDegree.put(w, inDegree.get(w)-1);
        if (inDegree.get(w) == 0)
                ...
}
```