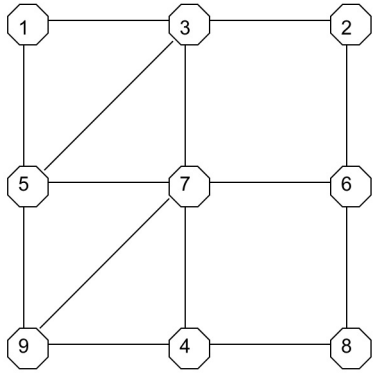
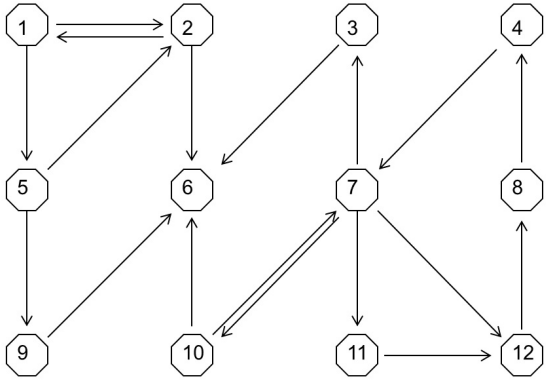
Graphen: Grundlegende Definitionen

- Graph G: abstrakte Struktur, die eine Menge von Objekten, oder *Knoten* V, mit zwischen diesen Objekten bestehenden Verbindungen, oder *Kanten* E, repräsentiert.
- *Notation:* Graph $G(V, E)$ hat n Knoten und m Kanten.



Graphen: Grundlegende Definitionen

- *Mehrfachkanten* sind möglich (z.B. U-Bahn Schienennetz)
- *Gerichtete Graphen:* Jede Kante hat eine Richtung



- *Ungerichtete Graphen:* Kanten verlaufen in beide Richtungen

Graphen: Grundlegende Definitionen

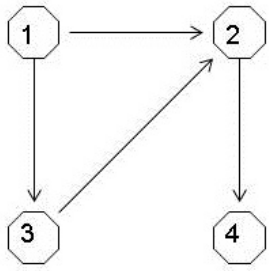
- *Weg:* Folge von Knoten $v_1, v_2, \ldots, v_k$, die durch Kanten verbunden sind.
- *Pfad:* Weg, bei dem sich alle Knoten voneinander unterscheiden.
- *Zyklus:* Weg, bei dem sich nicht alle Knoten voneinander unterscheiden.
- Graph mit Zyklus heißt *zyklisch*. Graph ohne Zyklen heißt *azyklisch*.

Repräsentation von Graphen

- *Adjazenzliste:* jeder Knoten v kennt eine Liste mit den Endpunkten der von v ausgehenden Kanten
  - Platzbedarf $O(n+m)$
  - Um Frage zu beantworten, ob Kante (u,v) existiert, benötigt man $O(deg(u)) = O(n)$ Zeit, $deg(u)$ = Grad von u
- *Adjazenzmatrix:* binäre Matrix der Dimension $n \times n$, die an Stelle (u,v) genau dann eine 1 hat, wenn die Kante (u,v) in G ist
  - Platzbedarf $O(n^2)$
  - Um Frage zu beantworten, ob Kante (u,v) existiert, benötigt man $O(1)$ Zeit

Gerichtete Azyklische Graphen
Directed Acyclic Graphs (DAG)

- Kommen in vielen Anwendungen (z.B. Studienordnung) vor



- *Quelle:* Knoten mit Eingangsgrad Null
- *Senke:* Knoten mit Ausgangsgrad Null
- **Satz:** Jeder DAG hat mindestens eine Quelle und mindestens eine Senke.

Topologische Sortierung

- Sortierung, in der für jede gerichtete Kante (u, v) der Knoten u vor dem Knoten v in der Sortierung kommt, heißt *topologische Sortierung*
- **Satz:** Ein gerichteter Graph G besitzt genau dann eine topologische Sortierung, wenn G azyklisch ist.
- Algorithmus zum Finden einer topologischen Sortierung (Laufzeit $O(n + m)$):
  1. Finde eine Quelle q von G
  2. Füge q als nächstes Element in der Sortierung ein
  3. Entferne q und alle von q ausgehenden Kanten von G
  4. Wiederhole bis G keine Knoten mehr enthält

Tiefensuche
Depth First Search (DFS)

- Möglichkeit zur Sortierung eines generellen Graphen
- Idee: G wird in einer Reihenfolge abgesucht, die immer zuerst in die Tiefe geht
- Jeder Knoten u von G bekommt
  - $u.d$: Zeitstempel der ersten Prüfung des Knotens
  - $u.f$: Zeitstempel, an der Algorithmus mit Knoten fertig ist
  - $u.predecessor$: Vorgänger von u in der Suche

Tiefensuche
Depth First Search (DFS)

Algorithm 1: Tiefensuche Algorithmus:

```
1  for jeden Knoten u von G do
2  |   u.d = NULL
3  |   u.predecessor = NULL
4  end
5  time = 0
6  for jeden Knoten u von G do
7  |   if u.d == NULL then
8  |   |   DFS-VISIT(G, u)
9  |   end
10 end
```

Tiefensuche
Depth First Search (DFS)

Algorithm 2: DFS-VISIT(G, u):

```
1  time = time+1
2  u.d = time
3  for jeden Knoten v mit (u, v) ∈ G do
4      if v.d == NULL then
5          v.predecessor = u
6          DFS-VISIT(G, v)
7      end
8  end
9  time = time+1
10 u.f = time
```

Tiefensuche
Depth First Search (DFS)

- Da jeder Knoten ein Mal bearbeitet wird und die Summe aller Kanten $O(m)$ ist, ist die Laufzeit des Algorithmus $O(m+n)$
- Kanten $(v.predecessor, v)$ bilden einen Wald, der im Folgenden als *Tiefenwald* bezeichnet wird
- Tiefensuche hat viele Eigenschaften und Anwendungen. Hier wird nur eine Anwendung behandelt.

Starke Zusammenhangskomponenten

Definitionen: Graph G heißt

- *Stark zusammenhängend von Knoten u aus* falls es für einen beliebigen Knoten v einen gerichteten Weg von u nach v in G gibt.
- *Stark zusammenhängend* falls G von jedem Knoten aus stark zusammenhängend ist.
- Teilgraph G' ist eine *starke Zusammenhangskomponente* falls G' stark zusammenhängend ist und es keinen stark zusammenhängenden Teilgraphen von G gibt, der G' echt enthält.

Starke Zusammenhangskomponenten

Algorithm 3: Starke Zusammenhangskomponenten finden (Algorithmus von Tarjan):

```
1  for jeden Knoten u von G do
2  |   u.d = NULL
3  |   u.lowlink = NULL
4  end
5  time = 0
6  S = leerer Stack
7  for jeden Knoten u von G do
8  |   if u.d == NULL then
9  |   |   STRONG-CONNECT(G, u)
10 |   end
11 end
```

Starke Zusammenhangskomponenten

Algorithm 4: STRONG-CONNECT(G, u):

```
1  time = time+1
2  u.d = time
3  u.lowlink = time
4  S.push(u)
5  for jeden Knoten v mit (u, v) ∈ G do
6  |   if v.d == NULL then
7  |   |   STRONG-CONNECT(G, v)
8  |   |   u.lowlink = min(u.lowlink, v.lowlink)
9  |   end
10 |   else if v ∈ S then
11 |   |   u.lowlink = min(u.lowlink, v.d)
12 |   end
13 end
14 if u.lowlink == u.d then
15 |   Beginne eine neue starke Zusammenhangskomponente Z
16 |   repeat
17 |   |   v = S.pop()
18 |   |   Füge v in Z ein
19 |   until v! = u;
20 end
```

Starke Zusammenhangskomponenten

- Grundidee des Algorithmus: Zyklen werden identifiziert
  - Wurzel w ist der Knoten einer starken Zusammenhangskomponente, der vom Algorithmus zuerst durchlaufen wird
  - Variable *lowlink* nimmt die Zeit $w.d$ der Wurzel an
  - Es ist immer $u.lowlink \leq u.d$ mit $=$ für die Wurzel
  - Wird ein Knoten als Wurzel w identifiziert, so ist jeder Knoten in S, der nach w eingefügt wurde in derselben starken Zusammenhangskomponente wie w
- Laufzeit (mit vorsichtiger Implementierung des Inhalts von S): analog zum Tiefensuchalgorithmus $O(n + m)$