Vorlesung

12

# Graphen und Bäume

# Inhalt


**1 Graph** **1**

1.1 Kategorisierung von Graphen 2

1.2 Graphen als Datenstruktur 6

1.2.1 Adjazenzmatrix 6

1.2.2 Inzidenzmatrix 9

1.2.3 Adjazenzliste 8

**2 Baum** **9**

**3 Heap** **11**

**4 Reading** **12**

**5 Glossar** Fehler! Textmarke nicht definiert.

Vorlesung

12

*Lernziele:*

# 1 Graphen

Ein **Graph** ist anschaulich ein Gebilde aus **Knoten** (auch Ecken oder Punkte), die durch **Kanten** verbunden sein können. Obwohl Graphen vielfach durch eine Zeichnung dargestellt werden, sind Graphen eigentlich „nur" mathematische Strukturen, die zum Beispiel durch die Zeichnung repräsentiert, genauer visualisiert, werden. Dies bedeutet vor allem, dass verschiedene Bilder denselben Graphen darstellen können.

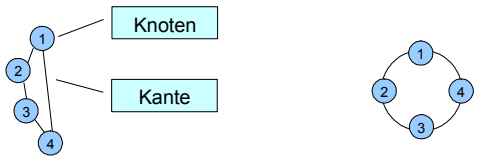



**Abb. xx: Alternative Visualisierungen desselben Graphen**

**Definition:** Ein Graph G ist ein geordnetes Paar zweier Mengen: G = (V, E).

Dabei bezeichnet **V** die Menge der im Graph enthaltenen **Knoten** und **E** die Menge der **Kanten** des Graphen. Die Bezeichnungen der Mengen entstammen dem Englischen: V für vertex (engl. für Knoten) und E für edge (engl. für Kante).

Graphen sind informatische/mathematische **Modelle** insbesondere für Netzstrukturen, beispielsweise für das Netz der deutschen Autobahnen (Knoten: Auf- und Abfahrten, Dreiecke und Kreuze) oder das Streckennetz einer Fluggesellschaft.

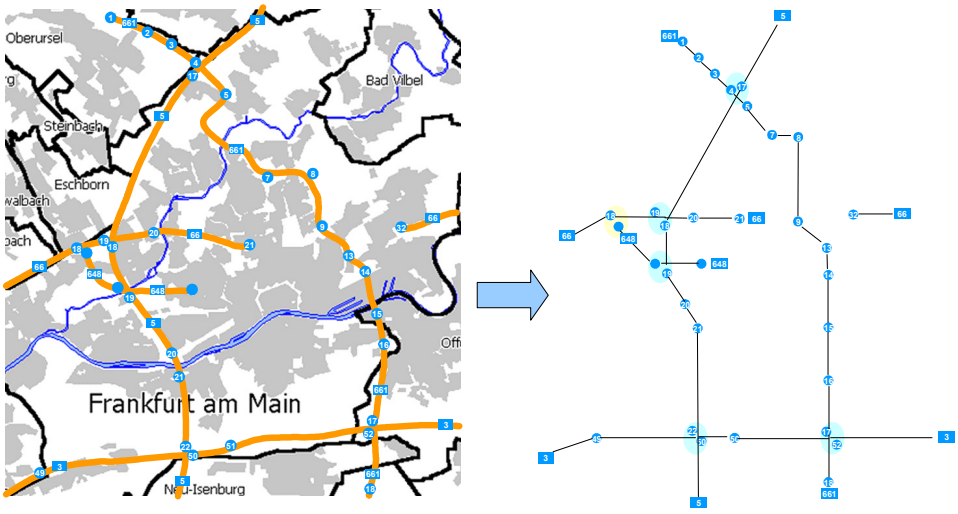



Abb. XX: Das Autobahnnetz um Frankfurt am Main als Beispiel für einen Graphen

(Quelle: Die Grundlage der linken Karte findet sich in der Wikipedia http://de.wikipedia.org/wiki/Bild:Mk_Frankfurt_Nachbargemeinden.png , **Autor**: Michael König, September 5, 2005, **License**: GNU-FDL. Die dort vorgefundene Graphik wurde um das Autobahnnetz ergänzt und in den „amerikanischen" Stil übertragen: Die Graphstruktur wird unmittelbar deutlich.
Bitte beachten Sie: Quasi alles verfügbare Kartenmaterial und andere Bilder, auch aus dem Internet, sind urheberrechtlich geschützt und dürfen nicht frei genutzt werden, bis auf ganz wenige Ausnahmen (z.B. oben). Bitte beachten Sie dies!

Wir können in diesem Beispiel die Elemente Auffahrt dadurch eindeutig machen, das wir diese mit (Autobahnnummer-Auffahrtnummer kennzeichnen), also z.B. „A3-49" für die Auffahrt Kelsterbach (unten links). Die Dreiecke und Kreuze würden dann allerdings zwei Bezeichnungen tragen, für das Frankfurter Kreuz A3-50 und A5-22; was man durch eine geeignet gewählte Elementbezeichnung, z.B. „A3-50;A5-22".

Offensichtlich hat unser Beispiel einige besondere Eigenschaften, z.B. führt nie eine Kante von v nach v, auch gibt es immer höchstens eine Kante von v nach w. Dies führt uns direkt dazu, verschiedene Unterarten von Graphen zu unterschieden und kurz zu betrachten.

## 1.1 Kategorisierung von Graphen

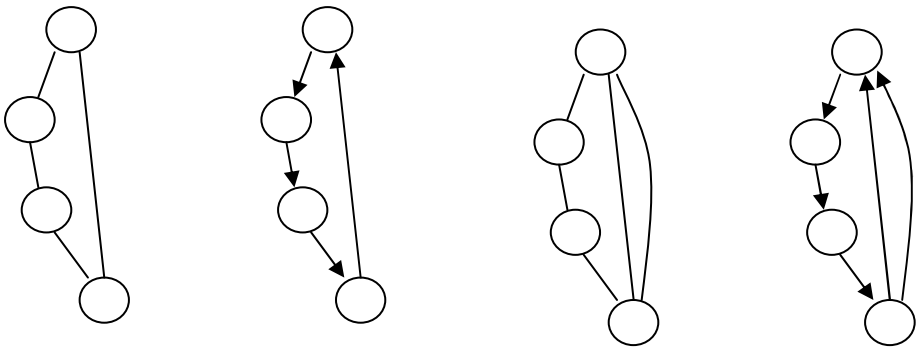
Man unterscheidet in der Graphentheorie vor allem zwischen **ungerichteten** und **gerichteten** Graphen sowie Graphen mit Mehrfachkanten (Multigraphen) und ohne Mehrfachkanten sowie Hypergraphen. Knoten und Kanten können auch mit **Namen** versehen sein, dann spricht man von einem **benannten** Graphen. In **Multigraphen** können zwei Knoten durch mehrere Kanten verbunden sein, was in einfachen Graphen nicht erlaubt ist. Statt mehrere Linien zwischen zwei Punkten zu zeichnen, kennzeichnet man Mehrfachkanten auch häufig durch ihre Vielfachheit.

In **gerichteten Graphen** oder auch **orientierten Graphen** werden Kanten statt durch Linien durch Pfeile gekennzeichnet, wobei der Pfeil vom ersten zum zweiten Knoten zeigt.

Bei **Hypergraphen** verbindet eine Kante (auch Hyperkante genannt) nicht nur zwei, sondern mehrere Knoten gleichzeitig. Bei Hypergraphen mit vielen Kanten wird diese Darstellung sehr schnell unübersichtlich.

Dabei ist $E$ in

- **ungerichteten Graphen ohne Mehrfachkanten** (auch schlichter oder einfacher Graph genannt) eine Teilmenge aller 2-elementigen Teilmengen von $V$: $E \subseteq \left\{ \{i, j\} \mid i, j \in V \right\}$. (Unser obiges Autobahn-Beispiel, aber dieses ist noch spezieller, siehe unten)
- **gerichteten Graphen ohne Mehrfachkanten** eine Teilmenge des kartesischen Produktes $V$ x $V$: $E \subseteq \left\{ (i, j) \mid i, j \in V \right\}$ (Beachte den unterschied zu ungerichteten Graphen: Hier ist ein Element von E ein geordnetes Paar, gibt also die Richtung, meist (von, nach) oder (**Startknoten i** und **Endknoten j** ); wird dann als Pfeil gezeichnet.) Eine andere Bezeichnung für gerichtete Graphen ist **Digraph** (Directed Graph).
- **ungerichteten Graphen mit Mehrfachkanten** eine Multimenge[1] über der Menge aller 2-elementigen Teilmengen von $V$,
- **gerichteten Graphen mit Mehrfachkanten** eine Multimenge über dem kartesischen Produkt $V$ x $V$,
- **Hypergraphen** eine Teilmenge der Potenzmenge von $V$.



ungerichteter Graph ohne Mehrfachkanten | gerichteter Graph ohne Mehrfachkanten | ungerichteter Graph mit Mehrfachkanten (**Multigraph)** | gerichteter Graph mit Mehrfachkanten **(Multigraph)**

Man sagt, falls $G$

[1] gegenüber dem gewöhnlichen Mengenbegriff können in einer Multimenge Elemente mehrfach vorkommen. Dementsprechend haben die für Multimengen verwendeten Mengenoperationen eine modifizierte Bedeutung.

- ein ungerichteter Graph ohne Mehrfachkanten ist und $e$ zu $E(G)$ gehört, $e$ ist eine **ungerichtete Kante** von $G$,
- gerichteter Graph ohne Mehrfachkanten ist und $e$ zu $E(G)$ gehört, $e$ ist eine **gerichtete Kante** von $G$.

In einer gerichteten Kante $e=(v,w)$ bezeichnet man $v$ als **Startknoten** und $w$ als **Endknoten** von $e$.

Das World-Wide-Web ist ein wichtiges Beispiel für einen gerichteten Graphen. Hierbei modellieren wir die Webseiten als Knoten und setzen eine Kante von Webseite v nach Webseite w, wenn die Webseite v einen Link auf die Webseite w hat.

Hat eine Kante $e$ Graphen die Form $(v, v)$ oder $\{v, v\}$, so spricht man von einer **Schleife** (oder Schlinge).. Gerichtete Graphen ohne Schleifen nennt man **schleifenlos** oder **schleifenfrei**. Das ist in unserem Autobahnbeispiel offensichtlich der Fall.

Gelegentlich schließt man diesen Fall schon in der Definition eines Graphen aus. Man ergänzt die Definition also um i≠j, also

ungerichteter Graph: $E \subseteq \{\{i, j\} \mid i, j \in V, i \neq j\}$
gerichteter Graph: $E \subseteq \{(i, j) \mid i, j \in V, i \neq j\}$

Als **Knotenzahl** $n(G)=|V(G)|$ eines Graphen $G$ bezeichnet man die Anzahl seiner Knoten, als **Kantenzahl** $m(G)=|E(G)|$ bezeichnet man die Anzahl seiner Kanten (in Multigraphen summiert man über die Vielfachheit der Kanten).

Einen Graph, dessen Knotenmenge endlich ist, nennt man **endlicher Graph**. Zwangsläufig ist in diesen auch die Kantenmenge endlich. Im Gegensatz dazu nennt man einen Graph, dessen Knotenmenge unendlich ist, **unendlicher Graph**. Meist betrachtet man nur endliche Graphen und lässt daher das Attribut "endlich" weg, während man "unendliche Graphen" explizit kennzeichnet.

Oft erweitert man Graphen $G=(V,E)$ zu **knotengefärbten Graphen**, indem man das Tupel $(V,E)$ zu einem Tripel $(V,E,f)$ ergänzt, wobei $f$ eine Abbildung von $V$ in die Menge der natürlichen Zahlen ist. Anschaulich gibt man jedem Knoten damit eine Farbe. Man könnte dies in unserem Beispiel nutzen, um einfache Auffahrten von Dreiecken und Kreuzen

Statt der Knoten kann man in Graphen ohne Mehrfachkanten und in Hypergraphen auch die Kanten färben und spricht dann von einem **kantengefärbten Graph**. Dazu erweitert man ebenfalls das Tupel $(V,E)$ zu einem Tripel $(V,E,f)$, wobei $f$ aber eine Abbildung von $E$ (statt von $V$) in die Menge der natürlichen Zahlen ist. Anschaulich gibt man jeder Kante damit eine Farbe. In Graphen mit Mehrfachkanten ist dies zwar prinzipiell auch möglich, aber schwieriger zu definieren, insbesondere, wenn Mehrfachkanten entsprechend ihrer Vielfachheit mehrere verschiedene Farben zugeordnet werden sollen.

Statt von knoten- bzw. kantengefärbten Graphen spricht man von **knoten-** bzw. **kantengewichteten** Graphen, falls $f$ statt in die natürlichen Zahlen in die reellen Zahlen abbildet. Knoten- bzw. kantengefärbte Graphen sind also Spezialfälle von knoten- bzw. kantengewichteten Graphen.

Man bezeichnet $f(v)$ bzw. $f(e)$ auch als **Gewicht** des Knotens $v$ bzw. der Kante $e$. Zur Unterscheidung spricht man auch von **Knotengewicht** bzw. **Kantengewicht**.

Analog gibt es auch **benannte Graphen** $(V, E, f, g)$, bei denen Knoten und/oder Kanten einen Namen tragen, und die Abbildungen $f$ bzw. $g$ den Knoten bzw. Kanten einen Namen zuordnen. Die zuvor abgebildeten Beispiele sind benannte Graphen, bei denen die Knoten mit Buchstaben und Zahlen benannt wurden. Dies wird oft bei Visualisierungen fast immer gemacht, um besser über den Graphen diskutieren kann.

Es gibt **einige spezielle Typen von Graphen**, die eine besondere Bedeutung haben: Hierzu folgende Definitionen:

Sei G = (V,E) ein gerichteter oder ungerichteter Graph.

(a) Zwei Knoten v; w ∈V heißen benachbart oder **adjazent**, falls sie durch eine Kante verbunden sind.

(b) Eine Kante ist mit einem Knoten **inzident**, wenn der Knoten ein Endpunkt der Kante ist.

(c) Eine Folge $(v_0, v_1, \ldots, v_m)$ heißt ein **Weg in G**, falls für jedes i $(0 \leq i < m)$

$(v_i; v_{i+1}) \in E$ (für gerichtete Graphen) oder

$\{f_{vi}; v_{i+1}\} \in E$ (für ungerichtete Graphen).

Die Weglänge ist m, also die Anzahl der Kanten. Ein Weg heißt einfach, wenn kein Knoten zweimal auftritt. Ein Weg heißt ein Kreis (oder Zyklus), wenn $v_0 = v_m$ und $(v_0, \ldots v_{m-1})$ ein einfacher Weg ist.

(d) **Ein gerichteter Graph ist azyklisch**, falls er keine Kreise besitzt.

(e) Ein ungerichteter Graph ist **zusammenhängend**, wenn es für je zwei Knoten v;w ∈ V einen Weg von v nach w gibt.).

Damit können wir eine sehr wichtige Klasse von Graphen identifizieren, nämlich:

> Ein **Baum** ist ein zusammenhängender, ungerichteter oder gerichteter, azyklischer Graph ohne Mehrfachkanten.

Ein in disjunkte Bäume zerlegbarer Graph heißt **Wald**.

Vermutlich fühlen Sie sich von all diesen Definitionen überfordert. Das ist verständlich. Merken Sie sich auf jeden Fall folgende Begriffe und Definitionen:

- gerichtete Graphen vs. ungerichtete Graphen
- Schleifen in Graphen und schleifenlose Graphen
- Weg in einem Graphen
- azyklischer Graph und zusammenhängender Graph

Deutlich mehr zu diesem Thema erfahren Sie dann im 2. Fachsemester in der Veranstaltung „Algorithmen und Datenstrukturen“.

## 1.2 Graphen als Datenstruktur

Ein Graph als Datentyp sollte mindestens die folgenden Operationen sind haben

- *Einfügen* (Kante, Knoten)
- *Löschen* (Kante, Knoten)
- *Finden* eines Objekts (Kante, Knoten).

Die bekanntesten Repräsentation von Graphen als Datenstruktur sind

- die **Adjazenzmatrix** (Nachbarschaftsmatrix)
- die **Adjazenzliste** (Nachbarschaftsliste)
- die Inzidenzmatrix (Knoten-Kanten-Matrix, seltener genutzt)

### 1.2.1 Adjazenzmatrix

Ein Graph mit *n* Knoten kann durch eine ***n*×*n*-Matrix** repräsentiert werden. Dazu nummeriert man die Knoten von 1 bis *n* durch und trägt in die Matrix die Beziehungen der Knoten zueinander ein.

Adjazent bedeutet benachbart, abgeleitet vom lateinischen adiacēns, aus ad = bei und iacēre = liegen.

In **ungerichteten Graphen** ohne Mehrfachkanten wird in die *i*-te Zeile und *j*-te Spalte eine 1 eingetragen, wenn der *i*-te und *j*-te Knoten benachbart oder adjazent, d.h. durch eine Kante verbunden sind. Andernfalls wird eine 0 eingetragen. Da die Relation „*benachbart*“ symmetrisch ist (wenn Knoten *i* mit Knoten *j* benachbart ist, dann ist auch Knoten *j* mit Knoten *i* benachbart), ist auch die Adjazenzmatrix symmetrisch. Für die Koeffizienten der Matrix gilt also $a_{ij} = a_{ji}$. Eine dazu gleichwertige Aussage ist, dass die Matrix gleich ihrer Transponierten ist: $A = A^T$. Bei symmetrischen Matrizen reicht es prinzipiell aus, alle Elemente oberhalb oder unterhalb der Hauptdiagonalen zu anzugeben. Häufig betrachtet man ungerichtete Graphen ohne Schleifen, dann steht in der Hauptdiagonalen der Matrix überall die 0.

In **gerichteten Graphen** ohne Mehrfachkanten wird in die *i*-te Zeile und *j*-te Spalte eine 1 eingetragen, wenn der *i*-te Knoten Vorgänger des *j*-ten Knoten ist, also $(i,j) \in E$. Andernfalls wird eine 0 eingetragen.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | |
| 2 | | | 1 | |
| 3 | | | | 1 |
| 4 | 1 | | | |

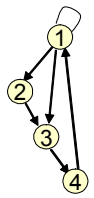



Abb. XX: Repräsentation eines gerichteten Graphen (mit einer Schleife) in einer Adjazenzmatrix

In Graphen mit Mehrfachkanten trägt man in die (i)-te Zeile und *j*-te Spalte die Vielfachheit von $\{i,j\}$ in ungerichteten Graphen bzw. $(i,j)$ in gerichteten Graphen ein. Auch an dieser Stelle sieht man leicht, warum Graphen ohne Mehrfachkanten als Spezialfälle von Graphen mit Mehrfachkanten betrachtet werden können.

In kantengewichteten Graphen trägt man häufig auch das Kantengewicht der jeweiligen Kante ein.

Hypergraphen lassen sich **nicht** durch eine Adjazenzmatrix darstellen.

In vielen Programmiersprachen sind Arrays als Primitiv vorhanden (siehe Vorlesung XX). Nicht so in Python. Hier implementieren wir ein veränderliche Arrays durch geschachtelte Listen:

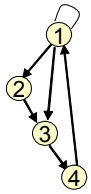
Adjazenz

### 1.2.2 Adjazenzliste

Die Adjazenzliste wird in ihrer einfachsten Form durch eine **einfach verkettete** Liste aller Knoten des Graphen dargestellt, wobei jeder Knoten eine Liste aller seiner Nachbarn (in ungerichteten Graphen) bzw. Nachfolger in gerichteten Graphen besitzt. Vielfachheiten der Kanten Knotengewichte, und Kantengewichte werden meist in Attributen der einzelnen Elemente gespeichert. Je nach Problemstellung kann es notwendig sein, statt einer einfach verketteten Liste eine doppelt verkettete Liste zu verwenden und in gerichteten Graphen zusätzlich zur Liste der Nachfolger eine Liste der Vorgänger zu verwalten. **In der Praxis verwendet man daher meist diese Form der Repräsentation.**

Unser Beispiel stellt sich dann wie folgt dar:

| | |
|---|---|
| 1 | [1,2,3] |
| 2 | [3] |
| 3 | [4] |
| 4 | [1] |




Zur direkte Umsetzung in Python könnte man also verschachtelte Listen implementieren: (Knoten-) Listen von (Kanten-) Listen:

```
Adjazenzliste = [[1,2,3],[3],[4],[1]]
```

Die elementaren Operationen lassen sich dann etwa wie folgt implementieren:

- Einfügen-Kante: prüfen, ob schon vorhanden, sonst Kantenliste.append
- Einfügen-Knoten: (ggf. prüfen, ob schon vorhanden), sonst Knotenliste.append
- Lösche-Kante: prüfen, ob vorhanden, dann Kantenliste.remove
- Lösche-Knoten: Knoten selbst löschen: Knotenliste.remove, dann aber auch alle Verweise auf diesen Knoten löschen … etwas aufwendiger, geht aber.

Adjazenzlisten sind zwar aufwändiger zu implementieren und zu verwalten, bieten aber eine Reihe von Vorteilen gegenüber Adjazenzmatrizen. Zum einen verbrauchen sie stets nur linear viel Speicherplatz, was insbesondere bei dünnen Graphen (also Graphen mit wenig Kanten) von Vorteil ist, während die Adjazenzmatrix quadratischen Platzbedarf bezüglich der Anzahl Knoten besitzt (dafür aber kompakter bei dichten Graphen, also Graphen mit vielen Kanten ist). Zum anderen lassen sich viele graphentheoretische Probleme nur mit Adjazenzlisten in linearer Zeit lösen.

Derartige Betrachtungen überschreiten aber unseren aktuellen Horizont: Auch dieses wird in der Veranstaltung „Algorithmen und Datenstrukturen“ vertieft werden.

Wir wollen statt dessen noch kurz eine Implementierung mit einem Dictionary betrachten. Wenn wir hierbei die Knotenbezeichnungen als Schlüssel nehmen, dann garantiert uns dies Eindeutigkeit (was wir ja wollen) und einen einfachen Zugriff über die Knotenbezeichnung.

```
>>> # Initialisierung
```

```
>>> Adjazenzliste = {"A":["A","B","C"],"B":["C"],"C":["D"],"D":["A"]}
>>> Adjazenzliste
{'A': ['A', 'B', 'C'], 'C': ['D'], 'B': ['C'], 'D': ['A']}
>>>
```

Wir haben damit die folgende Struktur aufgesetzt:

### 1.2.3 Inzidenzmatrix

Ein Graph mit *n* Knoten und *m* Kanten kann auch durch eine ***n*×*m*-Matrix** repräsentiert werden. Dazu nummeriert man die Knoten von 1 bis *n* und die Kanten von 1 bis *m* durch und trägt in die Matrix die Beziehungen der Knoten zu den Kanten ein.

Jede Spalte der Inzidenzmatrix enthält genau zwei von Null verschiedene Einträge. In ungerichteten Graphen zweimal die 1 und in gerichteten Graphen einmal die 1 (Endknoten) und einmal die -1 (Startknoten).

# 2 Baum

Als **Wald** bezeichnet man in der Graphentheorie einen ungerichteten Graphen ohne Kreis Zyklus). Ist dieser **zusammenhängend**, so spricht man von einem (**ungerichteten**) **Baum**. Die Zusammenhangskomponenten eines Waldes stellen in diesem Sinne für sich einen Baum dar, so dass ein Wald aus einem oder mehreren Bäumen besteht. Jeder ungerichtete Baum ist also auch ein Wald. Betrachtungen über Wälder lassen sich damit auch auf ungerichtete Bäume übertragen. Umgekehrt sind aber auch Betrachtungen über ungerichtete Bäume häufig leicht auf Wälder übertragbar.

Neben ungerichteten Bäumen betrachtetet man auch **gerichtete Bäume**, die häufig auch als **gewurzelte Bäume** bezeichnet werden und sich weiter in **In-Trees** und **Out-Trees** unterscheiden lassen. Die Struktur entspricht im wesentlichen der von ungerichteten Bäumen, jedoch gibt es einen ausgezeichneten Knoten, den man **Wurzel** nennt und für den die Eigenschaft gilt, dass alle Kanten von diesem wegzeigen (Out-Tree) oder zu diesem hinzeigen (In-Tree).

Als Datenstruktur werden meist nur **Out-Trees** verwendet. Dabei können ausgehend von der **Wurzel** mehrere gleichartige Objekte miteinander verkettet werden, so dass die **lineare Struktur der Liste aufgebrochen wird und eine Verzweigung stattfindet**.

Der maximale **Ausgangsgrad wird als Ordnung** eines Out-Trees bezeichnet und alle Knoten mit Ausgangsgrad 0 bezeichnet man als **Blätter**. Wie bei ungerichteten Bäumen bezeichnet man auch in gewurzelten Bäumen alle Knoten, die kein Blatt sind, als **innere Knoten**. Manchmal schließt man die Wurzel dabei aber aus.

Als Tiefe einen Knotens bezeichnet man die Länge des Pfades von der Wurzel zu ihm und als Höhe des Out-Trees die Länge eines längsten Pfades.

Für einen von der Wurzel verschiedenen Knoten v bezeichnet man den Knoten, durch den er mit einer eingehenden Kante verbunden ist als **Vater**, Vaterknoten, Elternknoten oder Vorgänger von v. Als **Vorfahren** von v bezeichnet man alle Knoten, die entweder Vater von v oder Vorgänger des Vaters sind. Umgekehrt bezeichnet man alle Knoten, die von einem beliebigen Knoten v aus durch eine ausgehende Kante verbunden sind als **Kinder,** Kinderknoten, Sohn oder Nachfolger von v. Als Nachfahren von v bezeichnet man Kinder von v oder deren Nachfahren. Als **Geschwister oder Geschwisterknoten** werden in einem Out-Tree Knoten bezeichnet, die den gleichen Vater besitzen.

Out-Trees lassen sich auch **rekursiv definieren**. Sie bestehen aus einem Knoten w, der die Wurzel des Baumes darstellt, welcher ausschließlich mit den Wurzeln knotendisjunkter Out-Trees T1, T2, ..., Tn verbunden ist, und zwar in Richtung der Wurzeln von T1, T2, ..., Tn

Da Bäume zu den meist verwendeten Datenstrukturen in der Informatik gehören, gibt es viele Spezialisierungen.

- So ist bei **Binärbäumen** die Anzahl der Kinder eines Knotens **höchstens zwei**.
- In **balancierten Bäumen** gilt zusätzlich, dass sich die Höhen des linken und rechten Teilbaums an jedem Knoten höchstens um eins unterscheiden.
- Bei **Suchbäumen** sind die Elemente in der Baumstruktur geordnet abgelegt, so dass man schnell Elemente im Baum finden kann.
- Man unterscheidet hier weiter in **binäre Suchbäume** mit **AVL-Bäumen** als balancierte Version und B-Bäumen sowie diversen Varianten, z.B. den B*-Bäumen (die Blattknoten sind zusätzlich in einer Liste miteinander verkettet ). (Achtung das B steht hier **nicht für Binär**, sondern für **balanciert**(!) und geordnet; B-Bäume müssen keine Binärbäume sein!) Dadurch wird neben der effizienten Suche einzelner Datenelemente auch das schnelle sequenzielle Durchlaufen aller Datenelemente unterstützt)
- Spezialisierungen von B-Bäumen sind wiederum 2-3-4-Bäume oder so genannte Rot-Schwarz-Bäume

Bäume sind in ihrem Aufbau zwar mehrdimensional jedoch in der Verkettung der Objekte oft unidirektional. Die Verkettung der gespeicherte Objekte beginnt bei der Wurzel des Baums und von dort in Richtung der Blätter des Baums. Aufgrund ihrer einfachen Struktur, kann die Komplexität von auf Bäumen arbeitenden Algorithmen meist gut abgeschätzt werden. Oft arbeiten die Algorithmen mit einem Baum als Datenstruktur schneller als andere Algorithmen für dasselbe Problem.

Um alle Knoten eines Graphen effizient betrachten zu können, werden gerne Bäume bzw. Wälder aus dem Graphen konstruiert. Bäume und Graphen werden Sie in der Theoretischen Informatik noch intensiv studieren.

# 3 Heap

Der Heap (Haufen, Halde) vereint die Datenstruktur eines Baums mit den Operationen einer Vorrangwarteschlange. Häufig hat der Heap neben den minimal nötigen Operationen wie *insert*, *remove* und *extractMin* auch noch weitere Operationen wie *merge* oder *changeKey*. Je nach Reihenfolge der Priorität in der Vorrangwarteschlange wird ein Min-Heap oder ein Max-Heap verwendet.

In einem Heap können Objekte oder Elemente abgelegt und aus diesem wieder entnommen werden. Sie dienen damit der Speicherung von Mengen. Den Elementen ist dabei ein Schlüssel zugeordnet, der die Priorität der Elemente festlegt. Häufig werden auch die Elemente selbst als Schlüssel verwendet.

Über die Menge der Schlüssel muss daher eine totale Ordnung festgelegt sein, über welche die Reihenfolge der eingefügten Elemente festgelegt wird. Beispielsweise könnte die Menge der ganzen Zahlen zusammen mit der Kleiner-Relation (<) als Schlüssel-Menge fungieren.

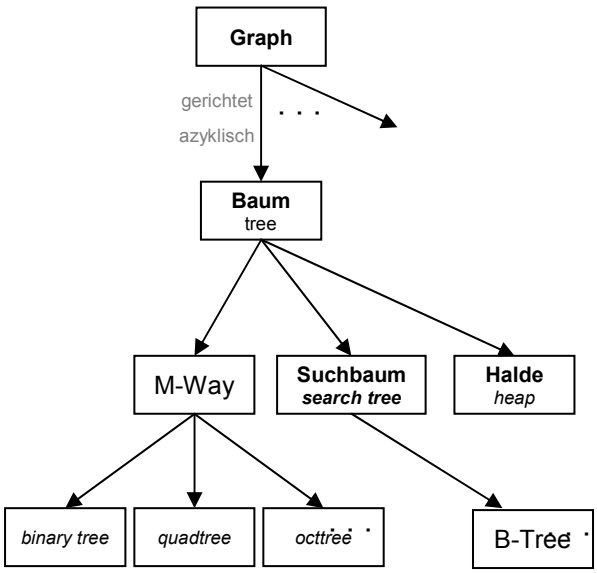
Der Begriff Heap wird häufig als bedeutungsgleich zu einem partiell geordneten Baum verstanden. Gelegentlich versteht man einschränkend nur eine besondere Implementierungsform eines partiell geordneten Baums, nämlich die Einbettung in ein Array, als Heap.

Verwendung finden Heaps vor allem dort, wo es darauf ankommt, schnell ein Element mit höchster Priorität aus dem Heap zu entnehmen, beispielsweise bei Vorrangwarteschlangen.

Spezialisierungen des Heap sind der Binäre Heap, der Binomial-Heap und der Fibonacci-Heap.

Treaps sind eine weitere Spezialisierung: Der Treap vereinigt Eigenschaften von Bäumen und Heaps in sich: **Treap** (*Binary search Tree* + Heap).

Eine grobe Übersicht über Strukturen und Spezialisierungen von Graphen ist in folgender Abbildung wiedergegeben:



# 4 Implementierungen

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.left = None
        self.right = None

    def __str__(self):
        return "[%s, %i, %i]" % (str(self.data),
            id(self.left), id(self.right))

class BinaryTree:
    def __init__(self):
        self.root = None

    def add(self, data):
        if self.root == None:
            self.root = Node(data)
        else:
            curnode = self.root
            lastnode = self.root
            while curnode != None:
                lastnode = curnode
                if data < curnode.data:
                    curnode = curnode.left
                    direction = -1      # links einfügen
                else:
                    curnode = curnode.right
                    direction = +1      # rechts einfügen
            if direction == -1:
                lastnode.left = Node(data)
            else:
                lastnode.right = Node(data)
    def _prchilds(self, node):
        if node != None:
            return "(%s; %s; %s)" % (self._prchilds(node.left),
node, self._prchilds(node.right))
        else:
            return "nil"
```

# 5 Anwendungen von Bäumen und Graphen

Szenengraph

Suchbäume + Sortieren