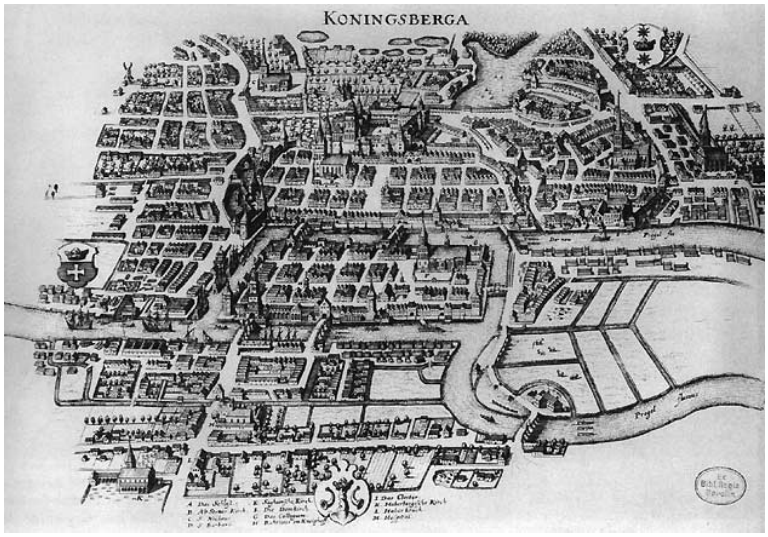
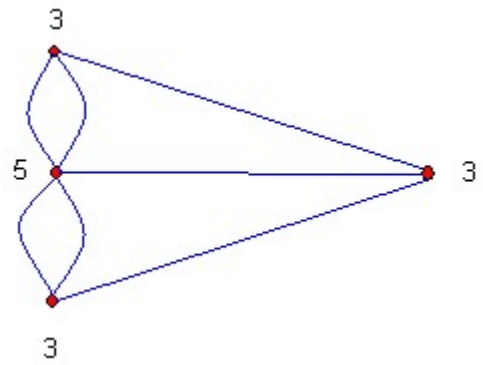
3. Die Datenstruktur Graph

3.1 Einleitung: Das Königsberger Brückenproblem

Das **Königsberger Brückenproblem** ist eine mathematische Fragestellung des frühen 18. Jahrhunderts, die anhand von sieben Brücken der Stadt Königsberg illustriert wurde. Das Problem bestand darin, zu klären, ob es einen Weg gibt, bei dem man alle sieben Brücken über den Pregel genau einmal überquert, und wenn ja, ob auch ein Rundweg möglich ist, bei dem man wieder zum Ausgangspunkt gelangt. Wie Leonhard Euler 1736 bewies, war ein solcher Weg bzw. „Eulerscher Weg“ in Königsberg nicht möglich, da zu allen vier Ufergebieten bzw. Inseln eine ungerade Zahl von Brücken führte. Es dürfte maximal zwei Ufer (Knoten) mit einer ungeraden Zahl von angeschlossenen Brücken (Kanten) geben. Diese zwei Ufer könnten Ausgangs- bzw. Endpunkt sein. Die restlichen Ufer müssten eine gerade Anzahl von Brücken haben, um sie auch wieder verlassen zu können.



Das Brückenproblem ist kein klassisches geometrisches Problem, da es nicht auf die präzise Lage der Brücken ankommt, sondern nur darauf, welche Brücke welche Inseln miteinander verbindet. Es handelt sich deshalb um ein topologisches Problem, das Euler mit Methoden löste, die wir heute der Graphentheorie zurechnen.

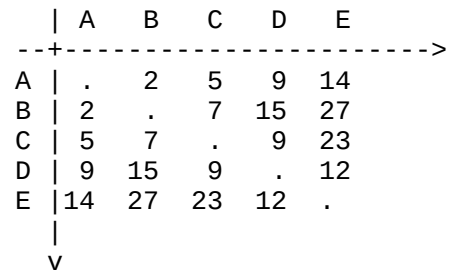


3.2 Unterschiede Graph - Baum

Auch ein Graph besteht wie der Baum aus Knoten und Kanten. Allerdings gibt es bei einem Graphen keine Wurzel und die Anzahl der Kanten ist nicht beschränkt. Jeder Knoten kann Kanten zu allen anderen Knoten besitzen, wenn man so will sogar zu sich selbst. Als Kante bezeichnet man wie beim Baum die Verbindung zwischen zwei Knoten.

Der Baum ist also eine spezielle Form eines Graphen. Während wir beim Binärbaum maximal zwei Nachfolger-Knoten hatten, so kann ein Knoten im Graphen Kanten zu allen anderen Knoten besitzen.

Kanten lassen sich sehr leicht durch eine Tabelle darstellen, bei der sowohl in der ersten Zeile als auch in der ersten Spalte alle Knoten aufgelistet sind. Du kennst bestimmt Entfernungstabellen zwischen Städten. Diese sind ja nichts anderes als die Kanten zwischen den Knoten.

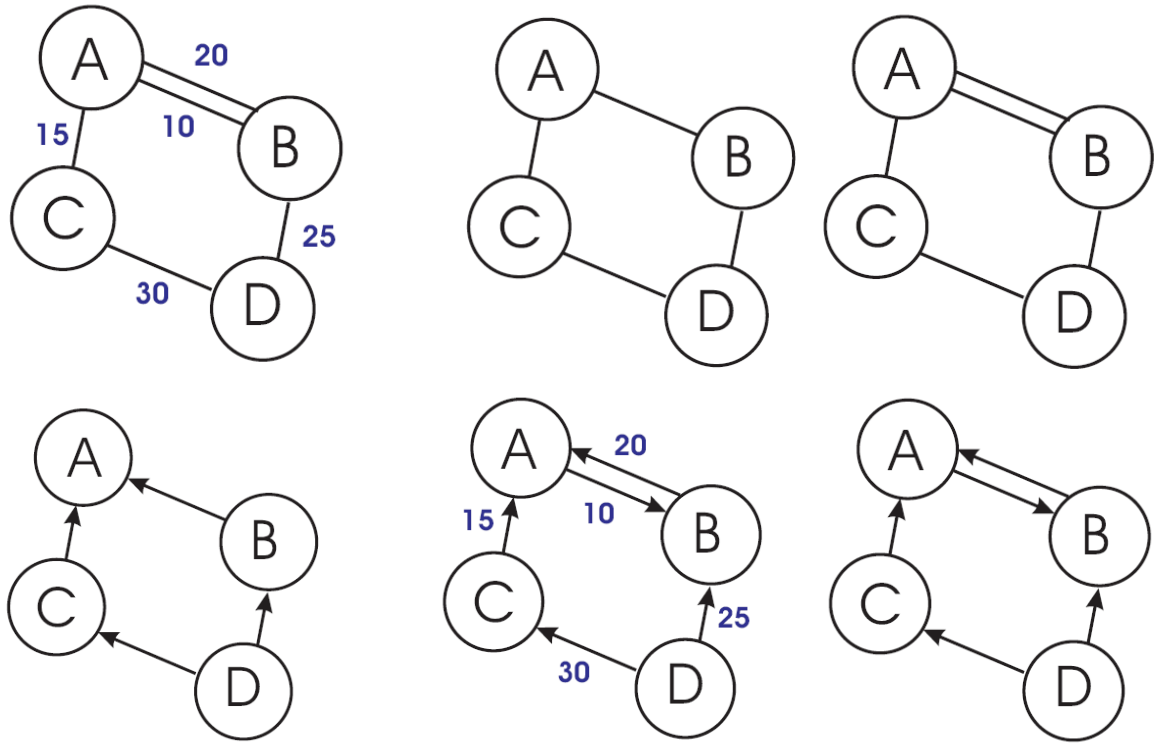


Adjazenzmatrix

Zwischen den Städten A bis E sind die Entfernungen angegeben. In der Informatik spricht man allerdings nicht von Entfernungstabellen, da nicht zwangsläufig Entfernungen angegeben sind, es kann jedes Kostenmaß Verwendung finden. In der Informatik nennt man eine solche Tabelle eine **Adjazenzmatrix**. In unserer Abbildung bildet die Diagonale eine Symmetrieachse. Das muss aber nicht in jedem Fall so sein. An mehreren Beispielen lässt sich zeigen, dass die Kosten in einer Richtung andere sind als in die andere Richtung sein können:

- Der Transport eines Containers von Shanghai nach Hamburg ist wegen der unterschiedlich hohen Auslastung der Schiffe teurer als von Hamburg nach Shanghai.
- Die Zeit, die man für eine Wanderung zwischen zwei Orten benötigt, ist unterschiedlich lang, wenn es in eine Richtung überwiegend bergauf geht.
- Einbahnstraßen führen zu unterschiedlichen Wegen zwischen zwei Knoten.

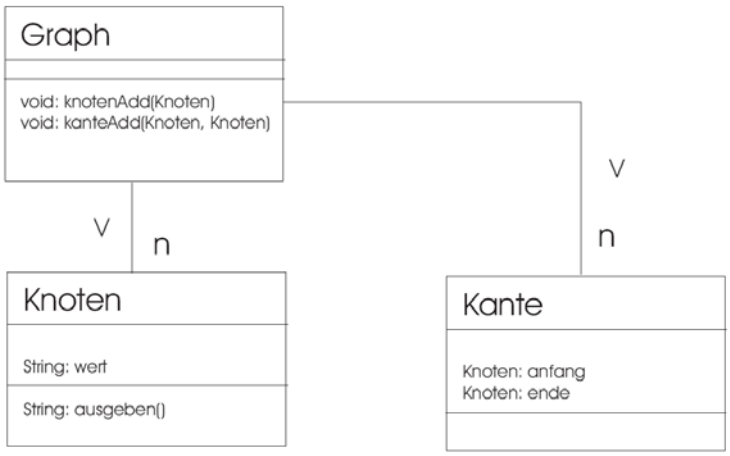
Die Beispiele zeigen, dass es neben ungerichteten Graphen auch gerichtete Graphen geben muss. Daneben kann eine Kante auch mit dem Attribut „Gewicht“ versehen werden, das ein beliebiges Kostenmaß für die Strecke darstellt. Zwischen zwei Knoten kann es auch mehrere Kanten geben. In unseren Beispielen werden wir uns aber darauf beschränken, dass zwischen zwei Knoten in eine Richtung maximal eine Kante verlaufen soll.



Unterschiedliche Graphen, mit und ohne Gewichtung, gerichtet und ungerichtet, mit einer oder mehrerer Kanten zwischen zwei Knoten.

3.3 Modellierung eines Graphen

Die Modellierung eines Graphen erscheint nicht viel anders als die Modellierung eines Baumes. Es gibt aber keine Wurzel und ein Knoten hat keine Nachfolger, sondern eine vergleichsweise unbestimmte Anzahl von Kanten, von der wir nun wissen, dass sie zwischen 0 und der Anzahl der Knoten im Graphen liegt. In einer sehr einfachen Modellierung sieht der Graph dann so aus:



In der Klasse Graph können die Beziehungen zu den Klassen Knoten und Kante durch zwei ArrayList realisiert werden. Die Verwaltung der Kanten ist aber ein schwieriges Unterfangen. Daher verwendet man in der Praxis als Datenstruktur eine der beiden Möglichkeiten Adjazenzliste oder Adjazenzmatrix, um die Kanten im Rechner zu repräsentieren.

Bei der Adjazenzliste besitzt jeder Knoten eine Referenz auf eine eigene Liste, in der die Referenzen aller Nachbarn dieses Knotens gespeichert sind. Allerdings ist der Verwaltungsaufwand für diese Datenstruktur nicht so einfach wie bei der Adjazenzmatrix.

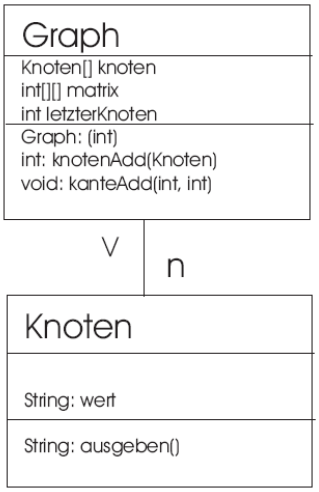
Bei einer Matrix (oder Matrize) handelt es sich um ein zweidimensionales Feld, also eine Nachbarschaftstabelle (oder Entfernungstabelle), wie sie oben schon angesprochen wurde.

In Java lässt sich diese Tabelle leicht als zweidimensionales Array des Datentyps `int` anlegen. Anzumerken ist dabei allerdings, dass dies keine objektorientierte Art der Modellierung darstellt. Die Objekte Kanten repräsentieren nicht mehr eine Klasse Kante. Das Klassendiagramm muss entsprechend angepasst werden.

```
public class Graph
{
  int[][] matrix;
  …

  public Graph(int maxKnoten){
    …
    // zweidimensionales Feld anlegen
    matrix=new int[maxKnoten][maxKnoten];
    for (i=0; i<maxKnoten; i++){
      for (j=0; j<maxKnoten; j++){

         // keine Kante erhält den Wert -1
         if (i!=j) matrix[i][j]=-1;

         // Gewicht für die Kante zu sich selbst ist 0
         else matrix[i][j]=0;

       }
    }
  }

}
```

Nachteilig erweist sich für das Feld, dass schon zu Beginn die maximale Anzahl an Knoten bekannt sein muss.

Die Knoten selbst können als ArrayList oder als Array implementiert werden. Das sollte keine Schwierigkeiten bereiten.



Im Vergleich zum ersten Klassendiagramm hat sich einiges geändert: Die Klasse Kante ist weggefallen, da alle Kanten durch das zweidimensionale Feld `matrix` repräsentiert werden. Die Knoten wurden hier als `Array` vorgegeben und nicht als ArrayList. Der Konstruktor erwartet als Parameter einen int-Wert, mit dem die maximale Anzahl an Knoten vorgegeben wird. Insofern ist es sinnvoll, das Feld für die Knoten genau auf die Länge festzulegen. Kanten werden mit der Methode kanteAdd(int, int) angegeben, die beiden Integer-Werte stehen für den Start und das Ziel der Kante. Gemeint ist jeweils der Index i und j in der Matrix. Da die Stelle eines Knotens kein Attribut des Knotens selbst ist, soll die Methode `knotenAdd(Knoten)` einen Integer-Wert zurückliefern. Dies übernimmt die Aufgabe einer Referenz, um die Kanten in der Matrize richtig eintragen zu können. Denn wenn Knoten erzeugt werden, dann hält man seine Position in der Matrix als Integer-Wert einer Variable fest. Kanten können dann für diese Knoten leicht eingetragen werden. Das folgende Listing zeigt ein Beispiel, wie in einer Test-Klasse ein Graph aufgebaut wird.

```
public class Test
{
  public los(){
    // der Graph hat maximal 10 Knoten
    Graph graph=new Graph(10);
    // zwei Knoten warden erzeugt
    Knoten k1=new Knoten(“Hamburg”);
    Knoten k2=new Knoten(“Berlin”);

    // die Knoten warden dem Array graph.knoten hinzugefügt
    // in der int-Variable speichern wir die Position des Knotens im Feld
    int hamburg=graph.addKnoten(k1);
    int berlin=graph.addKnoten(k2);

    // in der Adjazenzmatrix wird die Kante eingetragen
    graph.addKante(haburg,berlin);

  }

}
```

Wie bei der Liste ist es aber sinnvoll, nicht direkt auf eine Klasse Knoten zu implementieren, sondern auf eine Schnittstelle und davon ein Datenelement abzuleiten. Später können die Datenelemente leicht ausgetauscht werden, um die Struktur des Graphs auf andere Aufgabenstellungen anzupassen.

Exkurs: Vergleichende Betrachtungen zwischen Adjazenzmatrix und Adjazenzliste

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. In der Praxis verwendet man daher meist diese Form der Repräsentation.

http://kik.informatik.fh-dortmund.de/visual/grapheditor.html