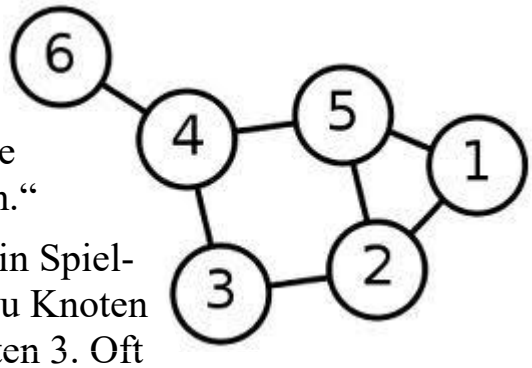
Datenstrukturen in Java: Graphen

Aus Wikipedia: „Ein Graph ist in der Graphentheorie eine abstrakte Struktur, die eine Menge von Objekten zusammen mit den zwischen diesen Objekten bestehenden Verbindungen repräsentiert. Die [...] Objekte werden dabei Knoten (auch Ecken) des Graphen genannt. Die paarweisen Verbindungen zwischen Knoten heißen Kanten [...] Häufig werden Graphen anschaulich gezeichnet, indem die Knoten durch Punkte und die Kanten durch Linien dargestellt werden."



Der abgebildete Graph könnte beispielsweise ein Spielfeld darstellen. Zwar kann man von Knoten 1 zu Knoten 2 gelangen, nicht aber in einem Schritt zu Knoten 3. Oft möchte man den Kanten eine „Richtung" geben: Z.B. soll die Kante zwischen Knoten 1 und 2 nur in Richtung 1→2, nicht aber in Richtung 2→1 durchlaufen werden können. Graphen mit dieser Eigenschaft heißen „gerichtete Graphen".

In Java können Graphen folgendermaßen modelliert werden: Knoten werden durch Objekte, Kanten durch Referenzen repräsentiert:

```
public class Knoten {
    Knoten kanteA, kanteB;
    String name;
}
```

Den Graphen aus der Abbildung könnte man folgendermaßen nachbauen:

```
Knoten eins = new Knoten();
eins.name = "Knoten eins";
eins.kanteA = new Knoten();
eins.kanteA.name = "Knoten zwei";
eins.kanteA.kanteA = eins;   //... etc.
```

Durch die vorgestellte Implementierung hat man bereits einen gerichteten Graphen modelliert: würde man die letzte Zeile weglassen, so gäbe es im Javaprogramm keine Verbindung mehr von Knoten zwei zu eins.

Rekursiver Durchlauf von Graphen:

Möchte man in obiger Datenstruktur z.B. alle Knotennamen in Großbuchstaben umwandeln, so müssen alle Knoten erneut „besucht" werden. In der Regel hat man aber gar keine Referenz auf jeden Knoten des Graphen, sondern z.B. nur eine auf einen Anfangsknoten. Man kann die Struktur nun wie folgt durchlaufen: Man spendiert der Klasse Knoten die **`boolean`**-Eigenschaft **`visited`** und schreibt eine rekursive Methode:

```
public void besuche(Knoten k) {
  if (k.visited == false) {
    k.visited = true;  // + Knoten bearbeiten...
    besuche(k.kanteA); besuche(k.kanteB);
  }
}
```