Grapheneigenschaften

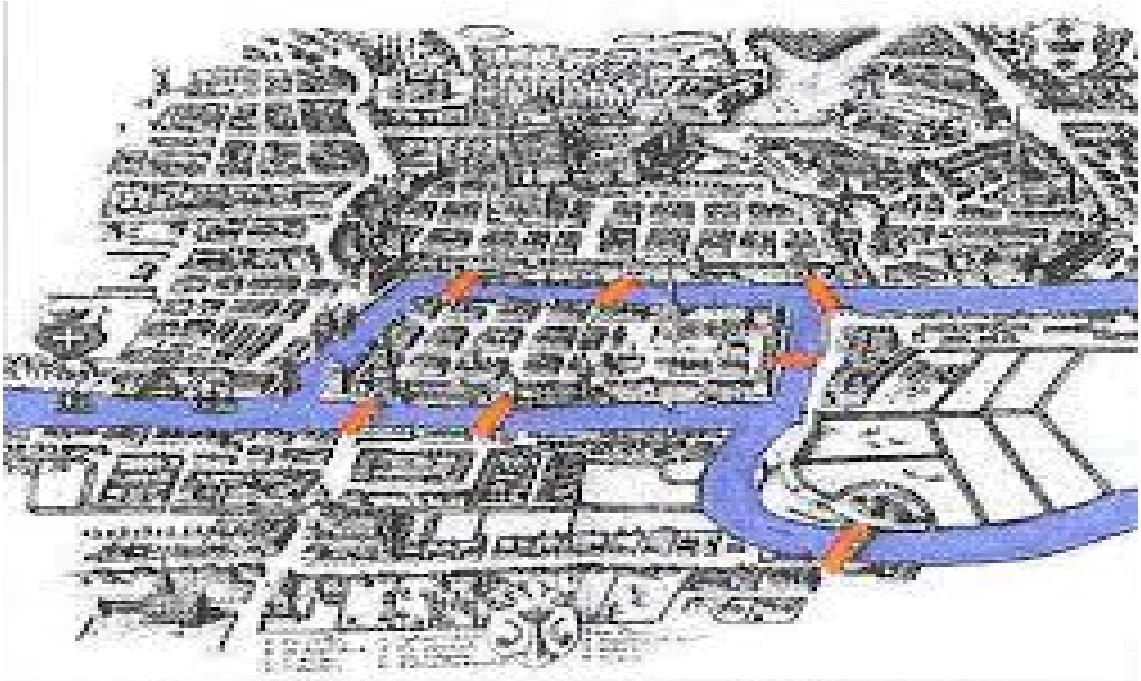
Einführung

Die Graphentheorie ist als Wissenschaft noch eine recht junge Disziplin. Das Problem des *Königsberger sieben Brücken* wurde von **Leonard Euler (1707-1783)** Jahre 1736 formuliert und auch gelöst. Damit gilt er als der Begründer dieser Wissenschaft.

Mit der Entdeckung der Elektrizität und der Entwicklung des Transport- und Verkehrswesens im19. Jahrhundert wurden ganz neue Probleme graphentheoretisch formuliert. **Gustav Robert Kirchhoff (1824-1887)** lieferte für die Darstellung und Berechnung von elektrischen Schaltkreisen entschiedende Beiträge.

Gerade in der heutige Zeit der Koordinierung von Verkehrsströmen ist die Frage nach Optimierung eine ganz wesentliche Frage.
Durch die rasanten Fortschritte in der Computertechnik nach 1945 erhielt die Graphentheorie neue Impulse. Viele Optimierungsaufgaben konnten von nun an mit ihrer Hilfe gelöst werden.

Beispiel: Königsberger Siebenbrückenproblem: Gibt es einen Rundweg, der jede Brücke in Königsberg genau einmal überquert?



Beschreibung von Graphen

Definition: Graph

Ein **Graph** $G = (V, E)$ besteht aus einem Paar zweier Mengen, der Knotenmenge V (Vertex Set) und E der Kantenmenge (Edges). Es gilt $E \subseteq \{\{v_i, v_j\} | \, v_i, v_j \in V\}$ für ungerichtete Graphen oder $E \subseteq V \times V, \{(v_i, v_j) | \, v_i, v_j \in V\}$ für gerichtete Graphen.
Die Anzahl $|V|$ heißt **Ordnung** von G. Die Anzahl $|E|$ heißt **Größe** von G. Es gilt: $E \subseteq \binom{n}{2}$.

Die Notation von Kanten in gerichteten Graphen der Art (v_i, v_j) hat die Bedeutung eines geordneten Paares.
Ein Graph mit maximaler Kantenzahl nennt man *vollständig*. Ein Graph mit Zahlenwerte an den Kanten nennt man *gewichtet*.

Eine Kante $\{v_i, v_i\}$bzw.$[v_i, v_i]$ heißt *Schleife*. Ein Graph ohne Schleife heißt *schleifenfrei*.
In ungerichteten Graphen definiert man weiterhin:

Definition:

Ein Knoten $v \in V$ und eine Kante $e \in E$ in einem Graphen *inzidieren* einander, falls $v \in e$. Zwei Knoten $u, v \in V$ heißen *adjazent*, falls $\{u, v\} \in E$. Die *Nachbarschaft* eines Knotens ist die Menge aller Nachbarn. Man nennt diese Nachbarn auch *Rand*. Die Anzahl aller Nachbarn nennt man Grad des Knotens. Der Grad von G ist der maximale Grad eines Knoten in G.

In gerichteten Graphen unterscheidet man für einen Knoten den Ausgangsgrad von v, der die Anzahl der Kanten, die in v beginnen, angibt. Entsprechend wird der Eingangsgrad definiert.

Wenn jeder Knoten von jedem anderen Knoten über einen Pfad erreichbar ist, so nennt man den Graphen *zusammenhängend*. Dafür definieren wir:

Definition: Wege und Pfade

Ist G ein Graph mit Knoten v_i, so heißt eine Folge von Knoten, eine Knotenfolge bzw. *Weg W*. Sind die Knotenverbindungen gerichtet, so bezeichnet man W als *gerichteten Weg W* in G. Falls G ungerichtet ist, so ist W ein *ungerichteter Weg*. Den Knoten v_1 nennt man Startknoten von W und den Knoten v_n Endknoten von W.
Ist $v_1 = v_n$, so heißt die Knotenfolge *geschlossen*.

Sind alle Knoten verschieden, so nennt man den Weg *Pfad*.

Hinweis: Es gibt auch Autoren, die einen Weg als Folge von Kanten $[v_i, v_j]$ definieren. Dann lauten die Definitionen:

Definition: Wege und Pfade (als Kantenfolgen)

Ist G ein Graph mit Knoten v_i, so heißt eine Folge von Kanten, die v_1 mit v_n verbindet, eine Kantenfolge bzw. *Weg* von v_1 nach v_n. Wir bezeichnen diese Folge $[v_1, v_2], [v_2, v_3], \ldots, [v_{n-1}, v_n]$ auch abkürzend mit $v_1, v_2, v_3, \ldots v_n$. Ist $v_1 = v_n$, so heißt die Kantenfolge *geschlossen*.

Eine Kantenfolgen ohne Knotenwiederholungen nennt man *Pfad*. Die Anzahl der Kanten eines Weges heißt *Länge des Weges*. In gerichteten Graphen ist die Länge des Weges die Summe aller Kantengewichte.

Hinweis: Ich werde, wenn nicht explizit anders betont, stets die erste Definition von Wegen verwenden.
Besondere Wege und Pfade sind die geschlossenen Knotenfolgen.

Definition: Kreis oder Zyklus

Ein *Zyklus* ist ein Weg, bei dem Start- und Endknoten identisch sind, das heißt falls $v_1 = v_n$. Ein Zyklus ist ein *Kreis*, wenn zusätzlich ein Pfad vorliegt. Das bedeutet also, das in Kreisen **nur** Anfangs- und Endknoten identisch sind. Alle anderen Knoten v_i, v_j gilt: $v_i \neq v_j$.

Man sagt auch: Ein Kreis ist ein kreuzungsfreier Zyklus.

Der Begriff des *Graphenzusammenhangs* bekommt für die Untersuchung von Netzwerken eine Bedeutung. Wie kann man Flüsse von Informationen von einem Knoten zum anderen leiten, ist jeder Knoten erreichbar und gibt es mehrere Wege?

Definition: Zusammenhang

Ein ungerichteter Graph G heißt *zusammenhängend*, wenn je zwei seiner Knoten durch einen Weg verbunden sind.

Ist ein Graph zusammenhängend, so besitzt er eine *Zusammenhangskomponente*. Anschaulich besitzt ein Graph mehr als eine Zusammenhangskomponente, wenn er in mehrere Teilgraphen zerfällt.
Gibt es zwei Wege zu einem Knoten, so ist er *zweifach zusammenhängend*.

Definition: starker Zusammenhang

Wenn G gerichtet ist, dann besteht die *starke Zusammenhangskomponente* von v aus allen Knoten w,
- die sowohl von v aus erreichbar sind,
- die aber auch v selbst durch einen in w beginnenden Weg erreichen.

G heißt stark zusammenhängend, wenn die starke Zusammenhangskomponente irgendeines Knotens aus allen Knoten von G besteht.

Ein Graph heißt kreuzungsfrei, wenn sich keine zwei Kanten schneiden.

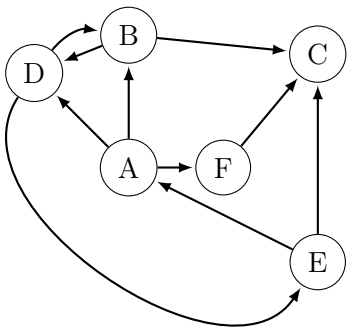
Definition:

Ein Graph heißt *planar* (oder plättbar), wenn er sich ohne Kreuzungen in der Ebene darstellen lässt.

Eine interessante Frage ist nun, unter welchen Bedingungen die Planarität von Graphen hergestellt bzw. bei Veränderung des Graphen erhalten werden kann.

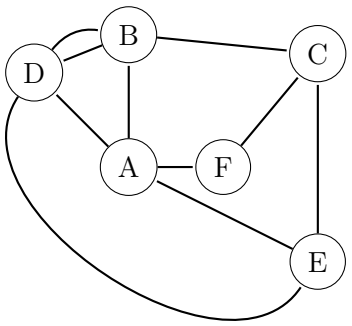
Implementierungen von Graphen

Eine Darstellung von Graphen wird meist in Form einer Adjazenzmatrix oder -liste realisiert. Die Adjazenzmatrix unterscheidet sich für gerichtete und ungerichtete Graphen in der Belegung der Matrixelelemente. Bei ungerichteten Graphen gibt es zwei Einträge für ein Kante. Die Matrix ist somit symmetrisch zur Hauptdiagonale. Bei schwach besetzten Graphen sind entsprechend viele Matrixelemente leer.
Darstellung von gerichteten Graph



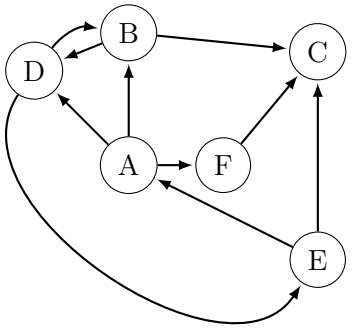
$$G_f = \begin{pmatrix} & A & B & C & D & E & F \\ A & 0 & 1 & 0 & 1 & 0 & 1 \\ B & 0 & 0 & 1 & 1 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 1 & 0 \\ E & 1 & 0 & 1 & 0 & 0 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

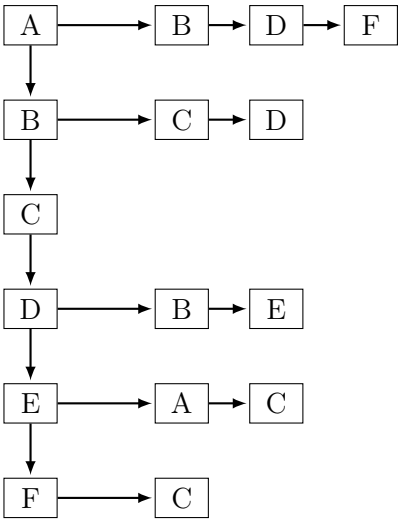
Darstellung von ungerichteten Graphen



$$G_f = \left(\begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 0 & 1 & 0 & 1 & 1 & 1 \\ B & 1 & 0 & 1 & 1 & 0 & 0 \\ C & 0 & 1 & 0 & 0 & 1 & 1 \\ D & 1 & 1 & 0 & 0 & 1 & 0 \\ E & 1 & 0 & 1 & 1 & 0 & 0 \\ F & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right)$$

Sind die Graphen schwach besetzt oder verändert sich häufig die Anzahl der Knoten, so ist die Implementierung mithilfe einer Adjazenzliste effizienter.





Wie werden nun Graphen konkret implementiert? Wir betrachten obigen ungerichteten Graphen. Die Darstellung der Knoten und deren Nachbarn erfolgt als Dictionary entsprechend der Adjazenzlistendarstellung.

Ein Framework für Graphen

Wir wollen nun eine erste Klasse für Graphen entwickeln. Die Eigenschaften (Menge von Knoten und Kantenmenge) kann man direkt implementieren. Weitere Funktionen sollen die Benutzung der Graphen vereinfachen.

```python
# graph.py
# Ein Graph ist ein Menge von Knoten und eine Menge von ungerichteten Kanten
# Die Darstellung erfolgt in einer Adjazenzliste mithilfe eines Dictionary
# Knotenwerte sind vorerst nur Zahlen

class Graph():

    def __init__(self, graph={}) -> None:
        self.__graph=graph

    def vertex_count(self) -> int:
        return len(self.__graph) # Anzahl der Knoten

    def vertices(self):
        """ gibt die Knoten des Graphen aus """
        return list(self.__graph.keys())

    def edges(self):
        """ gibt die Kanten eines Graphen aus """
        return self._generate_edges()

# Die wohlformatierte Ausgabe eins Graphen ermoeglichen
    def __str__(self) -> str:
      res = "vertices: "
      for k in self.__graph:
          res += str(k) + " "
      res += "\nedges: "
      for edge in self._generate_edges():
          res += str(edge) + " "
      return res

      if __name__ == "__main__":
    # Grundlegende Konstruktion des Graphen testen

    d={1:[1],5:[2,3]}

    graph = Graph(d)
    print(graph)
    print("Edges of graph:")
    print(graph.edges())
    graph.add_edge({2,3})
    print(graph.edges())
    print(graph)
```

Listing 1: Implementierung von Graphen für Zahlen

Algorithmen auf Graphen

Aufgabe: Studiere den Quelltext und probiere das Framework aus. Implementiere weitere Funktionen in der Klasse **Graph**, z.B. `add_vertex()` oder `add_edge()`. Auch die Funktion `generate_edges()` fehlt noch.

Nun geht es in die ersten Probleme, die Graphen betreffen. Eine grundlegende Frage ist die Frage der Suche innerhalb von Graphen. Schreibe einen Algorithmus `enthalten()` in Python, der zu einem Graphen $\mathcal{G}$ und einem Knotenelement e herausfindet, ob dieses Element im Graphen enthalten ist.

Etwas schwerer: Finde einen Algorithmus der überprüft, ob ein gerichteter Graph Zyklen beinhaltet.