Klaus Bovermann
02.05.23

Graphen im Unterricht
mit Hilfe von Python und Jupyter-Notebooks

Einführung

An vielen Stellen des Alltags begegnen uns sogenannte *Graphen*. Beispielsweise ist das Netz der öffentlichen Nahverkehrs ein Graph; die einzelnen Haltestellen (*Knoten*) sind durch Linien (*Kanten*) miteinander verbunden. Oder man kann mit Hilfe von Graphen in einem sozialen Netzwerk die Beziehungen der Menschen modellieren, also z.B. darstellen, wer mit wem befreundet ist.

Schülerinnen und Schüler sollten im Informatikunterricht der Sekundarstufe 2 diese wichtige Datenstruktur kennenlernen und grundlegende Algorithmen verstehen, entwickeln und implementieren:

- Kürzeste Wege; der *Dijkstra-Algorithmus*
- Spannende Bäume; *Algorithmus von Kruskal* oder *Algorithmus von Prim*
- MaxFlow-MinCut; *Ford-Fulkerson-Algorithmus*
- Travelling salesman problem; z.B. genetische oder andere approximative Algorithmen

In diesem Artikel wird aufgezeigt, wie man den Lernenden die Idee des Dijkstra-Algorithmus vermitteln kann.

Technische Details

Als Programmiersprache bietet sich z.B. *Python* an, da die Syntax besonders einfach ist, so dass ein Algorithmus, der bereits umgangssprachlich in einer Art Pseudocode entwickelt wurde, leicht programmiert werden kann. Dazu stellt Python eine Unmenge von Bibliotheken bereit, die es gestatten, sich bei der Implementation auf die wesentlichen Aspekte konzentrieren zu können, ohne viele technische Details kennen zu müssen. Daneben wurde eine weitere Bibliothek entwickelt, mit der man Sprechweisen von Graphen-Algorithmen direkt nutzen kann.

Als Entwicklungsumgebung ist der Anaconda-Navigator mit den *Jupyter-Notebooks* besonders geeignet:

- leicht zu implementieren
- für alle Plattformen verfügbar

Jupyter-Notebooks (JNBs) erlauben es, den Schülerinnen und Schülern Lernmaterial zur Verfügung zu stellen. Ein JNB enthält Lehrtexte und Python-Code und erfüllt mehrere Zwecke. Ein JNB ist ein

1. vorbereitetes digitales dynamisches Arbeitsblatt
   - ein Tutorial mit Erklärtexten und -bildern, vorbereitetem Code, Aufgaben…
2. computational Essay
   - ein digitales Essay, mit dem die lesende Person interagieren kann.
3. worked Example
   - mit lauffähigen und veränderbaren Programmen

Typischerweise enthält also ein JNB Lehrtexte, die wie ein traditionelles Schulbuch die fachlichen Grundlagen vermitteln. Anders als ein Schulbuch kann ein JNB dynamische Bilder, anklickbare Links, Videos, … enthalten. Mit Hilfe von MarkDown können diese Inhalte formatiert werden.

Daneben findet sich in einem JNB ausführbarer Python-Code. Schülerinnen und Schüler lernen so die Syntax und Semantik von Python, können den Code interaktiv verändern und erweitern, verbessern und vervollständigen.

Eine Python-Bibliothek für die Schule

Neben der Python-Bibliothek `networkx`, die im Internet erhältlich ist und in der Anaconda-Umgebung installiert werden kann, wurde ein eigene Python-Bibliothek mit dem Namen `nrw_graph` entworfen, mit der man Graphen-Algorithmen einfach und verständlich implementieren kann. Zentrale Idee dieser Bibliothek ist es, ungerichtete Graphen zu verwalten.

1. Knoten

Zunächst gibt es die Knoten des Graphen. Es eignen sich dafür alle Python-Objekte, die hashfähig sind. Typischerweise ist ein Knoten charakterisiert durch einen Namen, doch auch andere Objekte sind denkbar.

In dem Beispiel

```
import nrw_graph as ng
MeinGraph = ng.nrw_graph()
MeinGraph.fuegeKnotenHinzu("HalloPython")
MeinGraph.fuegeKnotenHinzu(True)
MeinGraph.fuegeKnotenHinzu(42)
```

wird der String „HalloPython", der boolsche Wert True und die Zahl 42 dem zunächst leeren Graphen hinzugefügt.

Jeder Knoten kann beliebig viele beliebige Attribute bekommen, wobei Attribute mit Hilfe von Namen spezifiziert werden:

```
MeinGraph.setKnotenAttribut(42, "schoenheit", "Toll")
MeinGraph.setKnotenAttribut("HalloPython", "wert", -5)
```

Man kann auch einem Knoten bereits beim Hinzufügen gewünschte Attribute mitgeben:

```
MeinGraph.fuegeKnotenHinzu("Erna",gewicht=55,
                                   wohnort="Hamburg")
```

Der Aufruf

```
MeinGraph.alleKnoten()
```

erzeugt dann eine Liste aller Knoten:

```
['HalloPython', True, 42, 'Erna']
```

während der Aufruf

```
MeinGraph.getKnoten("Erna")
```

ein Python-Dictionary erzeugt:

```
{'gewicht': 55, 'wohnort': 'Hamburg'}
```

Der Aufruf

```
MeinGraph.getKnotenAttribut(42, "schoenheit")
```

liefert dann den String

```
“Toll”
```

Weitere Vereinfachungen bieten Funktionen an, mit denen Knoten markiert und als besucht gekennzeichnet werden können:

- `markiereKnoten(knoten, marke)`
  - der angegebene Knoten erhält die angegebene Marke.
- `getKnotenMarke(knoten)`
  - die dem angegebenen Knoten zugefügte Marke wird geliefert.
  - Hinweis: ein neu hinzugefügter Knoten besitzt noch keine Marke!
- `besucheKnoten(knoten)`
  - der angegebene Knoten wird als besucht markiert.
- `verlasseKnoten(knoten)`
  - der angegebene Knoten wird als unbesucht markiert.
- `knotenIstBesucht(knoten)`
  - liefert genau dann True, wenn der Knoten als besucht markiert wurde.
  - Hinweis: für einen neu hinzugefügten Knoten ist der besucht-Status undefiniert!

2. Kanten

Analog kann man ungerichtete Kanten verwalten. Das Python-Programm

```
MeinGraph.fuegeKanteHinzu("Erna", 42)
MeinGraph.fuegeKanteHinzu(42, True, laenge=123, farbe="pink")
MeinGraph.setKantenAttribut("Erna", 42, "gewicht", 5)

print(MeinGraph.alleKanten())
print(MeinGraph.getKante(42,True))
print(MeinGraph.getKante("Erna", 42))
print(MeinGraph.getKantenAttribut(True, 42, "farbe"))
```

erzeugt dann die Ausgaben:

```
[(True, 42), (42, 'Erna')]
{'laenge': 123, 'farbe': 'pink'}
{'gewicht': 5}
pink
```

Das folgende Python-Programm zeigt das Beispiel des bekannten *HausVomNikolaus*.

```
HDN = ng.nrw_graph()
HDN.fuegeKanteHinzu('a', 'b')
HDN.fuegeKanteHinzu('a', 'c')
HDN.fuegeKanteHinzu('a', 'd')
HDN.fuegeKanteHinzu('b', 'c')
HDN.fuegeKanteHinzu('b', 'd')
HDN.fuegeKanteHinzu('c', 'd')
HDN.fuegeKanteHinzu('c', 'e')
HDN.fuegeKanteHinzu('d', 'e')
```

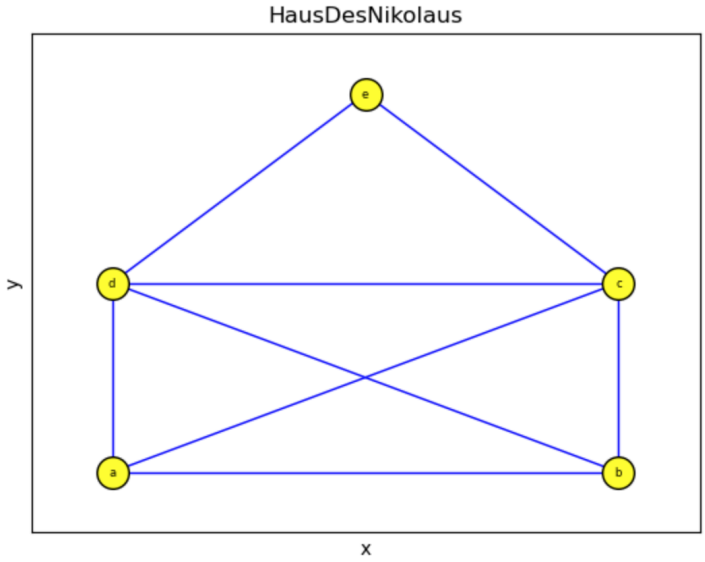
Nutzt man zusätzlich weitere Bibliotheken:

```
import pandas as pd
import matplotlib.pyplot as plt
import nrw_graph as ng
import networkx as nx
```

kann man den Graphen sogar sichtbar machen:

```
pos = {'a' : (0, 0),
       'b' : (1, 0),
       'c' : (1, 1),
       'd' : (0, 1),
       'e' : (0.5, 2)}

node_options = {
    "node_color": "yellow",
    "edgecolors": "black",
    "node_size": 290,
    "linewidths": 1,
                }
edge_options = {
    "edge_color": "blue",
    "width": 1,
}

label_options = {
    "font_size": 6,
    "font_color" : "black",
}

nx.draw_networkx_nodes(HDN, pos, **node_options)
nx.draw_networkx_edges(HDN, pos, **edge_options)
nx.draw_networkx_labels(HDN, pos, **label_options)

ax = plt.gca()
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_title("HausDesNikolaus")

ax.margins(0.1)
plt.axis("on")
plt.show()
```

so dass dann dieses Bild ausgegeben wird:



Das zentrale Jupyter-Notebook

Das erste Jupyter-Notebook ….. ImBau!