Klaus Bovermann

29.09.23

Graphen im Unterricht

Mit Hilfe von Python und Jupyter-Notebooks

Am Beispiel des Dijkstra-Algorithmus

Abstract

Graphen spielen eine große Rolle in vielen Alltagssituationen sowie in der wissenschaftlichen Informatik. Sie sind Teil der Informatik-Lehrpläne (mindestens der Leistungskurse) der gymnasialen Oberstufe. Beispielsweise findet man in dem Beispiel eines schulinternen Lehrplans von NRW (siehe z.B. https://www.schulentwicklung.nrw.de/lehrplaene/lehrplannavigator-s-ii/gymnasiale-oberstufe/informatik/hinweise-und-beispiele/schulinterner-lehrplan/uv_lk_q1_v.html):

1. Analyse von Graphen in verschiedenen Kontexten

(a) Grundlegende Begriffe (Graph, gerichtet – ungerichtet, Knoten, Kanten, Kantengewicht)

(b) Aufbau und Darstellung von Graphen anhand von Graphenstrukturen in verschiedenen Kontexten (Adjazenzmatrix, Adjazenzliste)

2. Die Datenstruktur Graph im Anwendungskontext unter Nutzung der Klassen Graph, Vertex und Edge.

(a) Erarbeitung der Klassen Graph, Vertex und Edge und beispielhafte Anwendung der Operationen

(b) Bestimmung von Wegen in Graphen im Anwendungskontext (Tiefensuche, Breitensuche)

(c) Bestimmung von kürzesten Wegen in Graphen im Anwendungskontext (Backtracking, Dijkstra).

(d) Bestimmung von minimalen Spannbäumen eines Graphen im Anwendungskontext.

Die im folgenden aufgeführten Teile dieses Artikels sind entstanden, um Graphen im Unterricht für die Schülerinnen und Schüler erfahrbar zu machen:

- *Zwei Python-Bibliotheken, mit denen die Verwaltung von ungerichteten Graphen einfach gelingt.*
- *Eine für die Schülerinnen und Schüler geeignete sehr kurze Einführung in die Begriffswelt von Graphen.*
- *Ein Jupyter-Notebook, das den Schülerinnen und Schülern als Arbeitsmaterial vorgelegt wird.*

Einführung

An vielen Stellen des Alltags begegnen uns sogenannte *Graphen*:

- Das Netz des öffentlichen Nahverkehrs bildet einen Graph; die einzelnen Haltestellen (*Knoten*) sind durch Linien (*Kanten*) miteinander verbunden.
- Mit Hilfe von Graphen kann ein soziales Netzwerk die Beziehungen der Menschen modelliert werden, indem die Personen (Knoten), die miteinander befreundet sind, mit Kanten verbunden werden.

Schülerinnen und Schüler sollten im Informatikunterricht der Sekundarstufe 2 (besonders sicherlich in einem Leistungskurs) diese wichtige Datenstruktur kennenlernen und grundlegende Algorithmen verstehen, entwickeln und implementieren, wie z.B.:

- Tiefen- und Breitensuche in einem Graphen
- *Euler-Weg* und *Hamilton-Weg*
- Kürzeste Wege; der *Dijkstra-Algorithmus*
- Spannbäume; *Algorithmus von Kruskal* oder *Algorithmus von Prim*
- MaxFlowProblem; *Ford-Fulkerson-Algorithmus*
- TravellingSalesmanProblem; z.B. genetische oder andere approximative Algorithmen

Im Rahmen der Unterrichtseinheit *Graphen* sollten die Grundbegriffe der Graphentheorie vorgestellt werden. *Euler-Wege* und die damit verbundenen Probleme, Sätze und Methoden können dort gute Dienste leisten.

Der Dijkstra-Algorithmus

Im Rahmen dieses Artikels muss dieser Algorithmus sicherlich nicht vorgestellt zu werden. Wichtig ist die methodisch-didaktische Aufbereitung und Präsentation. Das wird verdeutlicht in dem beigefügten Jupyter-Notebook (s.u.). Das Problem des Kürzesten Weges ist sicherlich besonders geeignet, für das Thema *Graphen* zu motivieren.

Die Programmiersprache *Python*

Als Programmiersprache bietet sich *Python* an:

- Die Syntax ist besonders einfach, so dass ein Algorithmus, der bereits umgangssprachlich in einer Art Pseudocode entwickelt wurde, leicht implementiert werden kann.
- Die Semantik der Basiskonzepte ist leicht verständlich.
- Ein Python-Programm erfordert weniger Code (als Java), daher kann ein Programm schneller und verständlicher erzeugt werden.

- Programmbausteinen können einfach getestet werden (Vorteil einer Interpretersprache); damit unterstützt Python das sog. *Prototyping*.
- Python stellt eine Unmenge von Bibliotheken bereit, die es gestatten, sich bei der Implementation von Algorithmen auf die wesentlichen Aspekte konzentrieren zu können, ohne viele technische Details kennen zu müssen.
- Die Installation von Python sowie von schüler- bzw. schulgerechten IDEs (wie z.B. Jupyter, s.u.) auf allen Betriebssystemen und allen Geräten gelingt problemlos.

Zwar ist zur Zeit in den Schulen die Programmiersprache Java weit verbreitet. Doch in den Lehrplänen (mindestens dem Lehrplan in NRW) wird keine Programmiersprache vorgeschrieben, und der Informatikunterricht ist sicher kein Programmierkurs, sollte also nicht den Eindruck hinterlassen, dass das Erlernen einer bestimmten Programmiersprache im Mittelpunkt stünde. Ziel des Informatikunterrichts sollte u.a. darin bestehen, wichtige zentrale Ideen und Strukturen aus Informatik zu vermitteln.

Dabei benötigt man sicherlich auch(!) eine Programmiersprache, damit z.B. Algorithmen implementiert werden können und ihre Funktionalität überprüft werden können. Es dient nicht zuletzt der Motivation, wenn das lauffähige Programm das gewünschte Ergebnis produziert.

Die Rolle von *Jupyter-Notebooks*

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

1. Ein JNB ist ein vorbereitetes digitales *dynamisches Arbeitsblatt*
   - Es ist ein Tutorial mit Erklärtexten und -bildern, vorbereitetem Code, Aufgaben…
2. Ein JNB ist ein sog. *Computational Essay*
   - Es ist ein digitales Notizbuch, mit dem die lesende Person interagieren kann.
3. Ein JNB ist ein sog. *Worked Example*
   - Es enthält lauffähige und veränderbare Programme

Typischerweise enthält ein JNB Lehrtexte, die wie ein traditionelles Schulbuch die fachlichen Grundlagen vermitteln. Anders jedoch als ein Schulbuch kann ein JNB dynamische Bilder,

anklickbare Links, Videos, … enthalten. Mit Hilfe von *MarkDown*[1] (eine Art HTML-Light) 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.

Als Entwicklungsumgebung ist der *Anaconda-Navigator*[2] mit den *Jupyter-Notebooks*[3] besonders geeignet:

- kostenlos
- leicht zu installieren
- Browser-basiert; in allen gängigen Browsern lauffähig
- für alle Plattformen verfügbar

Bei der Installation von Anaconda wird auch automatisch (in der Regel die neueste Version) von Python installiert.

Jupyter-Notebooks sind spezielle Dateien, die mit dem Jupyter-System geöffnet werden können. Sie haben die Dateiendung “ipynb”.

Die Python-Bibliothek `networkx`

Es handelt sich um im Internet unter *https://networkx.org/* zugängliche Bibliothek. Dort findet man eine sehr umfangreiche Dokumentation aller verfügbaren Funktionen.

Insbesondere ist es mit Hilfe dieser Bibliothek und der dort bereitgestellten Klasse `Graph` möglich, ungerichtete Graphen zu verwalten.

Details zu den Bibliotheken `attrDirGraph` und `simpleGraph`

Die oben erwähnte Bibliothek `networkx` ist zwar sehr leistungsstark, jedoch aus Sicht des Informatikunterrichts an Schulen komplex und für Schülerinnen und Schüler gerade bei der Einführung in das Thema *Graphen* aus meiner Sicht nur bedingt geeignet. Deswegen wurden zwei Bibliotheken entwickelt, die alle Klassen und Funktionen aus der erwähnten Bibliothek erben, jedoch für Lernende einen einfachen Umgang mit den notwendigen Funktionen erlauben.

---

[1] siehe z.B. https://markdown.de/

[2] siehe z.B. https://www.anaconda.com/download

[3] siehe z.B. https://jupyter.org

Beide Bibliotheken können von GitHub heruntergeladen werden:

https://github.com/klausNetSchulbuch/GraphWithPython.git

Die Bibliothek `attrDirGraph` stellt eine Python-Klasse bereit, mit der ungerichtete Graphen verwaltet werden können. Jeder Knoten und jede Kante eines solchen Graphen kann beliebig viele, vom Benutzer frei zu benennende Attribute erhalten. Insbesondere kann jede Kante ein Gewicht haben, so dass dann der Graph (im Sinne der Graphentheorie) ein gewichteter Graph[4] ist.

Ein kurzes Beispiel kann den Umgang mit dieser Bibliothek verdeutlichen:

```
In [1]: import attrDirGraph as adg

        g = adg.attrDirGraph()

        g.fuegeKnotenHinzu("Berlin")
        g.fuegeKnotenHinzu("München")
        g.fuegeKanteHinzu("Berlin", "Potsdam")
        g.setKnotenAttribut("Berlin", "Farbe", "blau")

        print(g.alleKnoten())
        print(g.alleKanten())
        print(g.alleNachbarknoten("Berlin"))
        print(g.getKnotenAttribut("Berlin", "Farbe"))
```

```
['Berlin', 'München', 'Potsdam']
[('Berlin', 'Potsdam')]
['Potsdam']
blau
```

Besonders in Zusammenhang mit dem zu implementierenden Dijkstra-Algorithmus benötigt man spezielle Markierungen der Kanten und Knoten. Dazu stellt die Bibliothek `simpleGraph` weitere Vereinfachungen zur Verfügung.

Auch hierzu ein kurzes Beispiel:

[4] In diesem Sinne ist als ein gewichteter Graph nur ein Sonderfall eines attributierbaren Graphen.

```
In [2]: import simpleGraph as sg

        g = sg.simpleGraph()

        g.fuegeKanteHinzu("B", "K")
        g.markiereKnoten("B", "gross")
        g.besucheKnoten("K")

        if g.knotenIstBesucht("K"):
            print("Ist besucht")

        print(g.getKnotenMarke("B"))

Ist besucht
gross
```

Eine Dokumentation aller Funktionen dieser Klassen ist in den Bibliotheksdateien in Form von Python-Kommentaren zu finden. Der Umgang mit den beiden Klassen kann optimal bei der Benutzung des Jupyter-Notebooks erfahren werden.

Anmerkung zu den `assert-Statements` in den Bibliotheken

Alle Funktionen in den beiden Bibliotheksklassen mit sog. `assert-Statements` abgesichert. Wenn man beispielsweise ermitteln möchte, ob der Knoten "H" bereits besucht wurde, dieser Knoten jedoch in dem Graphen nicht enthalten ist, wird eine aussagekräftige Fehlermeldung erzeugt, die das Programm beendet oder mit einer entspr. Aktion behandelt werden kann. Diese Fehlerprüfungen sind jedoch sehr zeitintensiv. Wenn man sicher ist[5], dass solche Situationen niemals auftreten, kann man auf diese Prüfungen verzichten.

Dazu gibt es beide Bibliotheken als unsafe-Versionen, unter den Namen `attrDirGraphUnsafe` und `simpleGraphUnsafe`.

Die beigefügten Dateien

1. In dem Verzeichnis `GraphBib` findet man die Bibliotheken.
2. Im dem Verzeichnis `Daten` sind einige csv-Dateien, in denen Graphen beschrieben sind und in den Jupyter-Notebooks eingelesen werden können.
3. Das Jupyter-Notebook `Dijkstra-Lernen.ipynb` ist zentral für den Einsatz im Unterricht. Mit diesem Notebook können Schülerinnen und Schüler das Dijkstra-Verfahren kennen lernen. Dieses Notebook verdeutlicht den Ablauf der Unterrichtsreihe. Es verdeutlicht die Idee, die Jupyter-Notebooks charakterisieren (*dynamisches Arbeitsblatt, Computational Essay, Worked Example*).
4. Die Datei `Dijkstra-Lernen.pdf` ist ein PDF-Export des gleichnamigen Jupyter-Notebooks.
5. Die Datei `Einfuehrung.pdf` bietet einen sehr kurzen Abriss der Graphentheorie.

---

[5] Das Jupyter-Notebook `Dijkstra-Loesen.ipynb` findet automatisch kürzeste Wege. Nutzt man die sicheren Versionen der Bibliotheken, benötigt die Suche je nach Rechner bis zu 20 Minuten!