Übungsblatt 1

Einführung in Netzwerktheorie

Netzwerke und komplexe Systeme

F. Klimm und B.F. Maier

Schülerakademie 5.2 (Roßleben 2016)

Aufgabe 1.1 Graphen mit Python erstellen.

Wir wollen die Programmiersprache Python nutzen um Graphen zu erstellen und zu untersuchen. Die folgende Funktion `pfadgraph` in der Datei `pfadgraph.py` erzeugt zum Beispiel die Adjazenzmatrix eines Pfadgraphen.

```
1  # -*- coding: utf-8 -*-
2  # Importieren von benoetigten Bibliotheken
3  import numpy as np
4  # Definieren der Funktion
5  def pfadgraph( n ):
6      "Erzeugt die Adjazenzmatrix eines Pfadgraphen auf n Knoten.
           "
7      A=np.zeros((n,n)) # Leere Adjazenzmatrix der Groesse n x n
8      for x in range(0, n-1): # gehe ueber jeden Knoten
9          A[x,x+1]=1 # und verbinde ihn mit dem naechsten
10     A=A+np.transpose(A) # Transponieren und Addieren der Matrix
           um zu symmetrisieren
11     return A # das Objekt A wird von der Funktion zurueck
           gegeben
12
13 pfad10=pfadgraph(10) # Ruft die Funktion auf um einen Pfad aus
       10 Knoten zu erstellen
14 print(pfad10) # gibt die Matrix als Text in der Konsole aus
```

Teilaufgabe 1.1.1 Programm verstehen und verändern

Gehe Zeile für Zeile über den Programm-Code um ihn zu verstehen. Beantworte insbesondere die folgenden Fragen:

1. Was sind In- und Output der Funktion `pfadgraph( n )`?
2. Wie sieht die Adjazenzmatrix in jedem Schritt der FOR-Schleife aus? überprüfe deine Vermutung indem du nach Zeile 8 die folgende Zeile einfügst `print(A)`
3. Warum geht die FOR-Schleife von 0 bis $n-1$?
4. Was passiert in Zeile 10? Wie würde die Adjazenzmatrix aussehen wenn diese Zeile nicht eingefügt wäre? überprüfe deine Vermutung indem du die Zeile auskommentierst.

Das gute an Funktionen ist, dass wir sie wieder und wieder verwenden können. Erweitere das obige Programm, so dass dir nach der Adjazenzmatrix eines Pfades bestehend aus 10 Knoten auch den für 15 Knoten ausgibt.

Teilaufgabe 1.1.2 Netzwerke erstellen

Nutze das obige Programm als Ausgangspunkt um Funktionen zu schreiben die die folgenden Graphen erzeugen:

1. Nullgraph
2. Kreisgraph
3. Vollständiger Graph
4. Vollständiger Bipartiter Graph mit Gruppengrößen m und n

Aufgabe 1.2 Netzwerke mit Python darstellen

Wir erstellen Graphen meist als Adjazenzmatrix. Diese sind nützlich um die sie weiter mathematisch zu analysieren. Manchmal ist es allerdings von Vorteil ein Netzwerk darzustellen. Hierzu können wir zum Beispiel die Bibliothek `networkx` benutzen:

```
# -*- coding: utf-8 -*-
# Importiere Bibliotheken
import matplotlib.pyplot as plt
import networkx as nx

# lade die Funktion pfadgraph aus der Datei pfadgraph.py
# (die Datei zaehlt als python-Modul)
from pfadgraph import pfadgraph

pfad10 = pfadgraph(10) # Ruft die Funktion auf um einen Pfad
    aus 10 Knoten zu erstellen
G = nx.from_numpy_matrix(pfad10) # schreibe die Adjazenzmatrix
     in ein anderes Objekt
nx.draw(G)  # die Funktion die das eigentliche zeichnen ü
    bernimmt
```

Wir müssen nur die nötigen Bibliotheken laden und die Adjazenzmatrix in ein anderes Objekt umwandeln. Dieses kann dann von der Funktion `draw()` genutzt werden um eine Darstellung zu erzeugen.

Nutze `networkx` um verschiedene Graphen der Größe $n = 8$ darzustellen.

Aufgabe 1.3 Netzwerke mit Python untersuchen

Nun werden wir die in Aufgabe 1 erstellten Funktionen nutzen um diese Netzwerke zu untersuchen.

Teilaufgabe 1.3.1 Kanten zählen

Wir wollen untersuchen wie die Anzahl m von Kanten im Netzwerk sich in Abhängigkeit von der Anzahl der Knoten n verhält. Führe dafür die folgenden Schritte aus:

1. Schreibe eine Funktion die als Input eine Adjazenzmatrix und als Output die Anzahl von Kanten hat. (*Hinweis: Die numpy-Funktion* `sum()` *könnte nützlich sein.*)
2. Nutze eine FOR-Schleife um Netzwerke variierender Größe n zu erstellen und deren Kanten zu zählen.
3. Nutze `maplotlib` um diese Messungen darzustellen.
4. Vergleiche diese Werte mit den am Vormittag ermittelten analytischen Formeln.

Teilaufgabe 1.3.2 Vergleich mit empirischen Netzwerken

Bis jetzt haben wir uns nur mit Netzwerkmodellen beschäftigt. Nun wollen wir diese real existierenden (empirischen) Netzwerke vergleichen.

Nutze die Funktion `loadtxt` um eine Adjazenzmatrix aus einer Textdatei zu laden.

```
1 H = np.loadtxt("hennen.txt")
```

Bei dem Netzwerk handelt es sich um ein Netzwerk das angibt welche von 32 Hühner miteinander kämpfen.

Zähle die Kanten, stelle das Netzwerk dar und vergleiche mit den Netzwerkmodellen die wir eingeführt haben. Welches der Modelle beschreibt diese Daten am besten?

Wiederhole dies mit einem zweiten Netzwerk `florence.txt`. Ist eines der Modelle in der Lage die Daten gut zu beschreiben?

Dieses Netzwerk beschreibt die ehelichen Beziehungen von Adelsfamilien im Florenz des 13. Jahrhunderts. In der Datei `florence_families.txt` findet ihr die Namen der Adelsgeschlechter (Zeilennummer korrespondiert mit Knotennummer im Netzwerk). Welche Familie hat die meisten Verbindungen?

Aufgabe 1.4 Zusatzaufgaben

Teilaufgabe 1.4.1 Multipartite Netzwerke

Ähnlich wie die bipartiten Netzwerke mit zwei Gruppen von Knoten gibt es auch multipartite (oder p-partite) Netzwerke mit mehreren Gruppen. Schreibe eine Funktion die vollständige p-partiten Netzwerke erzeugt wobei jede der p Gruppen die gleiche Größe g besitzt.

Finde analytische Formeln für die Anzahl n der Knoten im gesamten Netzwerk und der Anzahl m der Kanten und überprüfe sie numerisch.

Teilaufgabe 1.4.2 Knoten mit gleichem Grad

Beweise, dass jeder Graph G der mehr als einem Knoten mindestens zwei Knoten mit gleichem Grad besitzt.

Teilaufgabe 1.4.3 3-reguläre Graphen

Ein Graph ist r-regulär wenn jeder Knoten den Grad r besitzt. Beweise, dass es genau dann einen 3-regulären Graphen mit n Knoten gibt, wenn $n > 3$ und n gerade ist.

Teilaufgabe 1.4.4 Dijkstra-Algorithmus

Implementieren den Dijkstra-Algorithmus. überprüfe ob seine Zeitkomplexität $\mathcal{O}(N^2)$ beträgt.