Worksheet 1

Introduction into Network Theory

Networks and Complex Systems

F. Klimm und B.F. Maier

Schülerakademie 5.2 (Roßleben 2016)

Exercise 1.1 Create Graphs with Python.

We want to use the coding language Python to create graphs and analyse them. The function `pathgraph` in the file `pathgraph.py` creates the adjacency matrix of the *pathgraph*.

```
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
```

Subexercise 1.1.1 Understanding the Programme and Adapting it

Go line by line through the code shown above to understand what is happening. Particularly answer the following questions:

1. What are input and output of the function `pathgraph( n )`?
2. How does the adjacency matrix look like for each step through the FOR-Loop? Check your supposition by inserting the command `print(A)` after line 8.
3. Why does the FOR-loop go from 0 to $n-1$?
4. What happens in line 10? What would happen without this command? Check your supposition by commenting out this line.

The benefit of functions is that we can reuse them mutliple times in a programme. Extend the above programme such that it prints not only the adjacency matrix of a path with ten but also fifteen nodes.

Subexercise 1.1.2 Creating Graphs

Create functions that create other synthetical graphs consisting of n nodes. You can use the function `pathgraph( n )` as a starting point and adapt it.

1. null graph, also called empty graph

2. cycle graph
3. complete graph
4. complete bipartite graph with group sizes m und n

Exercise 1.2 Illustrate Graphs with Python

Usually, we represent graphs as adjacency matrices. This is useful for their further mathematical analysis. Sometimes is it appropriate to visualise them with the help of a computer. One tool for this is the library `networkx`. Below is a small example of a programme that creates a pathgraph and its visualisation.

```
# -*- 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
```

First, we have to load the necessary libraries with the IMPORT command. Second, we create the adjacency matrix of a pathgraph and then make it into a 'networkx' object. Finally, this can then be used by the fucntion `draw()` to create an illustration.

Now, adopt this programme to illustrate the other networks from Subexercise 1.1.2 in size $n = 8$.

Exercise 1.3 Investigating Graphs with Python

Now we will use the functions from Subexercise 1.1.2 to analyse the networks by looking at some simple network properties.

Subexercise 1.3.1 Counting Edges

We want the investigate the relationship between the number n of nodes in a network with the number m of edges in the discussed synthetic networks. To achieve this go through the following steps:

1. Create a function that has as input the adjacency matrix of a network and as output the number of edges. (*Hint: The numpy-function* `sum()` *could be useful.*)
2. Use a FOR-loop to create networks of varying size n and count its number of edges. Save the results into a vector.
3. Use `maplotlib` to illustrate the relationship $m(n)$ for the different network models
4. Compare these numerical results with the analytical formulas we derived earlier during the lecture.

Subexercise 1.3.2 Comparing Synthetic with Empiric Networks

So far we focussed on network models. Now we want to compare these with real (empirical) networks that were created from data.

We can use the function `loadtxt` to load the adjacency matrix from a text file.

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

The network caries the information which of the 32 chicken is fighting against which other ones.

Count the number of edges in this network, illustrate it, and compare it with the network models we introduced earlier. Which model describes the data best?

Repeat this for a second network `florence.txt`. Is one of the networks able to describe the data well? This network describes the marriage relationships between the noble families in Florence in the 13th century. In the file `florence_families.txt` you find the names of the families (the line number corresponds with the node number). Illustrate the network and colour-code each node with its degree k_i and a label giving its name. Which family has the most connections?

Exercise 1.4 Optional Tasks

Subexercise 1.4.1 Multipartite Networks

Similar to bipartite networks with two groups of nodes there exists also *multipartite* networks, also called p-partite networks, which consist of p groups of nodes. Edges are only created between the groups but never inside of them. If all possible edges exist we call them *complete multipartite graphs*. Create a function that creates such p-partite networks with each of the p groups having the same size g.

Find analytic formulas for the number n of nodes in the whole graph and the number m of edges. Check your results numerically and illustrate the network.

Subexercise 1.4.2 Nodes with the same Degree

Prove that every graph G with more than one node has at least two nodes with the same degree.

Subexercise 1.4.3 3-regular graphs

A graph is called r-regular if every node has the degree r. Prove that there exists a 3-regular graph of n nodes if and only if, $n > 3$ and n is even.

Subexercise 1.4.4 Dijkstra-Algorithm

Implement the Dijkstra-Algorithm. By using it on networks of varying size n verify that its time complexity is $\mathcal{O}(N^2)$.