

Introduction to 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 *path graph*.

```
1 # -*- coding: utf-8 -*-
2
3 # import necessary library
4 import numpy as np
5
6 # define a function
7 def path_graph( n ):
8     """
9         Returns the (n x n) adjacency matrix
10        of a path graph with n nodes.
11    """
12
13    # empty adjacency matrix of shape (n x n)
14    A = np.zeros((n,n))
15
16    # loop over every node besides the last one
17    for u in range(0, n-1):
18
19        # connect this node to the subsequent one
20        A[u,u+1] = 1
21
22    A = A + np.transpose(A)
23
24    # the matrix will now be returned
25    return A
26
27 # call the function to obtain an adjacency matrix
28 # for a path graph of 10 nodes
29 path_10 = path_graph(10)
30
31 # put the matrix on screen in text
32 print(path_10)
```

Subexercise 1.1.1 Understanding the Program 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 the input and output of the function `pathgraph(n)`?
2. How does the adjacency matrix look like at each step of the FOR-Loop? Check your conjecture by inserting the command `print(A)` after line 17.
3. Why does the FOR-loop go from 0 to $n-1$?

4. What happens in line 22? What would happen without this command? Check your conjecture by commenting out this line.

The benefit of functions is that we can reuse them multiple times in a program. Extend the above program 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 synthetic 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 and n

Exercise 1.2 Illustrate Graphs with Python

Usually, we represent graphs as adjacency matrices. This is useful for their further mathematical analysis. Sometimes it is 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 program that creates a path graph and its visualisation.

```
1 # -*- coding: utf-8 -*-
2 import matplotlib.pyplot as plt
3 import networkx as nx
4
5 # import the function path_graph from the file pathgraph.py
6 from pathgraph import path_graph
7
8 # calls the path_graph function
9 # afterwards the variable path10
10 # contains the adjacency matrix
11 # of a P10 graph
12 path10 = path_graph(10)
13
14 # convert matrix to a networkx graph
15 G = nx.from_numpy_matrix(path10)
16
17 # draw the graph
18 nx.draw(G)
```

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

Now, adapt this program 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 to 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 takes the adjacency matrix of a network as an input and returns the number of edges as output. (Hint: The *numpy*-function `sum()` could be useful.)
2. Use a FOR-loop to create networks of varying size n and count their number of edges. Save the results into a vector.
3. Use `matplotlib` 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.

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

The network carries the information which of the 32 chickens 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 exist *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 Equal 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 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 $\mathcal{O}(N^2)$.