Graph Theory and Applications

Group homework — A graph problem with Python

Master 1 APE

The project

In groups of at most **five students**, choose a problem that can be modelled and studied using graph theory. You may adapt a problem from the lectures or tutorials, or propose your own problem. It may concern an economic, social or institutional network, a game, or another application of graphs. Both real data and graphs constructed by your group are acceptable.

Use **Python** to contribute to your study. Depending on your problem, your program may:

- compute a solution or implement a solution method;
- illustrate the problem, a mathematical result or a proposed solution;
- provide an application that automatically solves a specified class of instances.

These are alternative approaches; you may combine them. An application is optional. The emphasis is on understanding the mathematics and explaining the results, rather than on technical sophistication, the size of the dataset or the appearance of an interface. You may use NetworkX and other suitable Python libraries.

What your work should explain

1. **The problem.** State the question precisely. If you adapt a problem studied in class, explain the instance, extension or computational investigation undertaken by your group.
2. **The graph model.** Define the vertices, edges or arcs, and any weights or labels. State the assumptions and explain why this model represents your problem.
3. **The mathematics.** Identify the definitions and results used. Explain why they apply and justify your reasoning. Results already proved in class may be cited; any additional claim needed for your argument should be justified.
4. **The role of Python.** Explain what the code computes or illustrates, its inputs and outputs, and the method used. If you build a solver, specify which instances it can handle and whether it guarantees an exact answer.
5. **The results.** Present and interpret your findings. Check your program on a small example for which you can independently verify the answer. Explain relevant limitations. Numerical experiments can support a conjecture; they do not establish a general theorem.

Possible starting points

For example, you could investigate winning positions in a game, build a tool for analysing encounter graphs, study the planarity of a network, compare ways of ranking vertices, or model a problem of your own. Choose a question that your group can explain mathematically and investigate meaningfully with Python.

What to submit

Submit one joint package containing:

- a short report in English answering the points above;
- your Python code, as a notebook or scripts, and any data needed to reproduce the results;
- brief instructions for running the code. If you build an application, include its source and explain how to launch it.

A well-explained notebook may serve as both report and code. Cite the sources of any external data, code or mathematical results. Every group member should be able to explain the work submitted.

Submission instructions

Deadline: to be announced in class.

Send the project by email to `stephane.gonzalez@univ-st-etienne.fr`. The subject must be exactly `Homework graph theory`. List the names of all participants in the body of the email and copy all group members in CC.

How the project will be assessed

The assessment will consider the clarity of the problem and graph model, the correctness of the mathematical reasoning, the relevance and reliability of the Python work, and the quality of the explanation and interpretation. A simple project carried through carefully is entirely appropriate.

Optional preparation — Introduction to NetworkX

The following guide is adapted from the lab handout. Use the parts that help you prepare your project; submitting answers to all the guide's questions is not required.

Objectives

This guide introduces basic concepts of graph theory through hands-on practice using the `NetworkX` Python library. It covers how to:

- Create and manipulate graphs (undirected and directed);
- Visualize graphs;
- Compute basic graph properties (degrees, paths, connected components);
- Understand and interpret simple centrality measures;
- Load a graph from a text file.

Prerequisites

This lab assumes basic familiarity with Python. All code can be executed in **Google Colab**.

Installation (Google Colab)

Execute the following cell at the beginning of the notebook:

```
!pip install networkx matplotlib
```

Imports

```
import networkx as nx
import matplotlib.pyplot as plt
```

A Part 1 – Creating a Simple Undirected Graph

We start by creating a small undirected graph.

```
G = nx.Graph()
G.add_nodes_from([1, 2, 3])
G.add_edges_from([(1, 2), (2, 3)])
```

Display basic information:

```
print("Nodes:", list(G.nodes()))
print("Edges:", list(G.edges()))
```

Questions

1. How many nodes and edges does the graph have?
2. Is the graph connected?
3. What happens if you add the edge $(1, 3)$?

B Part 2 – Visualizing the Graph

Basic visualization:

```
nx.draw(G, with_labels=True, node_size=700)
plt.show()
```

Alternative layout:

```
pos = nx.spring_layout(G, seed=1)
nx.draw(G, pos, with_labels=True)
plt.show()
```

Question Why can two drawings of the same graph look different?

C Part 3 – Directed Graphs

We now consider a directed graph.

```
DG = nx.DiGraph()
DG.add_edge("A", "B")
DG.add_edge("B", "C")
print("Directed edges:", list(DG.edges()))
```

Questions

1. What is the difference between an undirected and a directed graph?
2. Is there a directed path from `C` to `A`?

D Part 4 – Degrees and Shortest Paths

Degrees:

```
print(dict(G.degree()))
```

Shortest path:

```
path = nx.shortest_path(G, source=1, target=3)
print("Shortest path from 1 to 3:", path)
```

Questions

1. What is the distance between nodes 1 and 3?
2. How does this distance change if edge $(1, 3)$ is added?

E Part 5 – Centrality Measures

We compute several classical centrality indices.

```
print("Betweenness:", nx.betweenness_centrality(G))
print("Closeness:", nx.closeness_centrality(G))
print("Eigenvector:", nx.eigenvector_centrality(G))
```

Questions

1. Which node appears the most central in this graph?
2. In which type of network would betweenness centrality be especially relevant?

F Part 6 – Cliques and Connected Components

```
print("Cliques:", list(nx.find_cliques(G)))
print("Connected components:", list(nx.connected_components(G)))
```

The command `find_cliques` lists maximal cliques (those that cannot be extended by adding a vertex), which need not all have maximum size.

Question Explain the difference between a clique and a connected component.

G Part 7 – Loading a Graph from a File

We now load a graph from a text file where each line represents an edge.
Example file `reseausimple.txt`:

```
A B
B C
C D
```

In Google Colab, you can either upload the file manually, or create it directly in a cell.
Option A (recommended in Colab): create the file in one cell

```
%%writefile reseausimple.txt
A B
B C
C D
```

Option B: upload the file manually
(Use the "Files" panel in Colab, then upload `reseausimple.txt`.)
Now, load the graph and display its basic properties:

```
G_file = nx.read_edgelist("reseausimple.txt", nodetype=str)

# nx.info() was removed in NetworkX 3.x

print(G_file)
print("Number of nodes:", G_file.number_of_nodes())
print("Number of edges:", G_file.number_of_edges())
```

Note (eigenvector centrality). `nx.eigenvector_centrality` may fail to converge on some graphs (e.g., poorly connected graphs or graphs with isolates). If you get a convergence error, check that your graph is sufficiently connected, or try removing isolated nodes or increasing the iteration limit.

Questions

1. How many nodes and edges does the graph have?
2. Is the graph connected?
3. Which node seems the most central?