

# Organising information: graphs

## Author(s)

[Silvio Peroni](#) – [silvio.peroni@unibo.it](mailto:silvio.peroni@unibo.it) – <https://orcid.org/0000-0003-0530-4305>

Digital Humanities Advanced Research Centre (DHARC), Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

## Keywords

Edges and nodes; Euler; Graph; Königsberg

## Copyright notice

This work is licensed under a [Creative Commons Attribution 4.0 International License](#). You are free to share (i.e. copy and redistribute the material in any medium or format) and adapt (e.g. remix, transform, and build upon the material) for any purpose, even commercially, under the following terms: attribution, i.e. you must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. The licensor cannot revoke these freedoms as long as you follow the license terms.

## Abstract

This chapter introduces the last data structure presented in this course, i.e. the *graph*. The historic hero introduced in these notes is Leonhard Euler, a great scientist of the 18<sup>th</sup> century who introduced a new mathematical field called graph theory for the very first time.

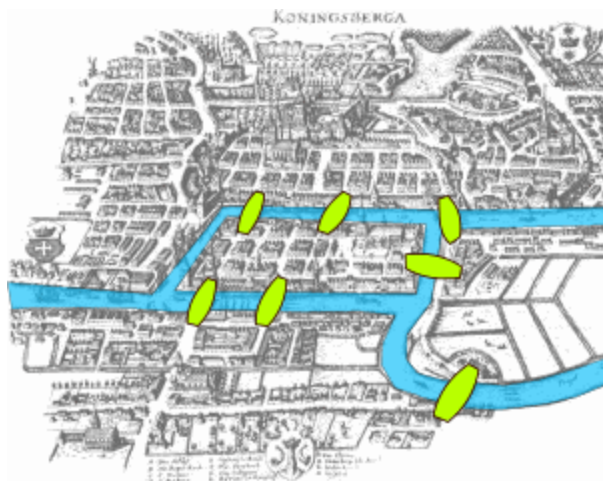
## Historic hero: Euler

[Leonhard Euler](#) (shown in [Figure 1](#)) was one of the most influential men of Science of whole history. His contributions in Mathematics, Physics, Astronomy, Logic, among others, were disruptive and even started pretty new disciplines. He spent most of his life in Saint Petersburg in Russia. Among the mathematical problems he dealt with, one related to a hilarious story that he solved by initiating a new field in mathematics called [graph theory](#).

The (mathematical) story told about the [seven bridges of the city of Königsberg](#), illustrated in [Figure 2](#). We can state the problem as follows: is it possible to walk around the town and cross each bridge once and only once? Several people have tried to propose a solution to this enigma before Euler. Finally, he demonstrated it through purely mathematical (and non-debatable) proof [\[Euler, 1741\]](#).

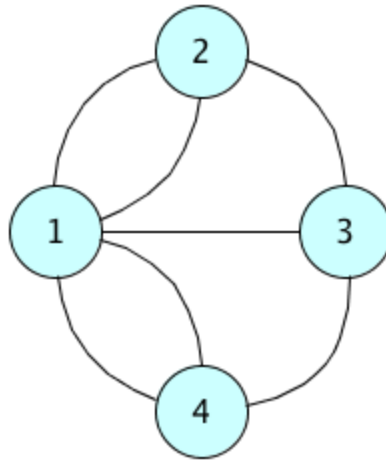


**Figure 1.** A portrait of Leonard Euler by Emanuel Handmann. Picture by Oursana, source: [https://en.wikipedia.org/wiki/File:Leonhard\\_Euler.jpg](https://en.wikipedia.org/wiki/File:Leonhard_Euler.jpg).



**Figure 2.** A representation of the seven bridges in Königsberg. Figure by Bogdan Giușcă, source: [https://commons.wikimedia.org/wiki/File:Königsberg\\_bridges.png](https://commons.wikimedia.org/wiki/File:Königsberg_bridges.png).

He abstractly described the four lands in Königsberg divided by the river as *nodes* of a network, where each *edge* between two nodes represents one of the city's bridges. [Figure 3](#) shows his abstract representation of the problem. Using this abstract notion, known as *graph*, he demonstrated that there is *no solution* to the problem of the seven bridges of Königsberg.



**Figure 3.** An abstract representation of the seven bridges of Königsberg of Königsberg using a graph.

He based the solution to the problem on the following intuition. The idea was that each node, excepting the starting node and the final node, should have an even number of edges. It is a practical implication: one should pass through two different bridges (i.e. arts) to enter and then go out from a node. Thus, non-starting and non-ending nodes must have an even number of edges for being satisfactorily traversed one or more times. However, all the nodes in [Figure 3](#) have an odd number of edges, which contradicts the aforementioned requirement.

## Graphs

[Graphs](#) are one of the principal data structures in Computer Science and Computational Thinking. For example, they describe routes between cities, connections to people in social networks, the organisation of links between Web pages, etc. [\[Albert and Barabasi, 2002\]](#). Graphs are entirely derived from the mathematical structure invented by Euler, as illustrated in [Section "Historic hero: Euler"](#). In particular, we can distinguish two different kinds of graphs: *undirected graphs* and *directed graphs*. In [undirected graphs](#), used by Euler in the seven bridges problem, one can traverse an edge in one way or the other indifferently. Instead, the edge specifies the direction for crossing it in [directed graphs](#).

In Python, as it happens for the trees, there is no built-in class defining this type of object. However, several external libraries implement them. Among the most used and famous there is

[NetworkX](#). This library makes available the common constructs for creating and traversing graphs and additional functions for analysing them for different purposes, such as for the analysis of social networks.

## Undirected graphs

We can create an undirected using the constructor `Graph()`. Then, we make all the nodes and edges of such a new graph by using its available methods.

```
from networkx import Graph

# create a new graph
my_graph = Graph()

# create four nodes
my_graph.add_node(1)
my_graph.add_node(2)
my_graph.add_node(3)
my_graph.add_node(4)

# create five edges
my_graph.add_edge(1, 2)
my_graph.add_edge(1, 3)
my_graph.add_edge(1, 4)
my_graph.add_edge(2, 3)
my_graph.add_edge(3, 4)
```

**Listing 1.** A simple undirected graph with four nodes and five edges. The source code of this listing is available [as part of the material of the course](#).

The NetworkX package allows us to associate an **immutable object** as a node. We can connect such a node through one or more edges. In particular, it is possible to execute the following methods on a graph object:

- `<graph>.add_node(<node>)` adds `<node>` as a node of the graph – note that, if a node with that value is already present, the method does not affect the graph;
- `<graph>.add_edge(<node_1>, <node_2>)` adds an edge between `<node_1>` and `<node_2>` – note that, since we are dealing with undirected graphs, inverting the position of the input nodes does not change the result;
- `<graph>.remove_node(<node>)` removes `<node>` from the graph as well as all the edges that involve it directly;
- `<graph>.remove_edge(<node_1>, <node_2>)` removes the particular edge between the two nodes specified.

[Listing 1](#) introduces an example of a graph. It creates a structure similar to the one presented in [Figure 3](#), except it is impossible to make multiple arcs between two nodes. Thus, using this specific constructor, it is impossible to create the same structure requested by Euler for solving the mathematical problem introduced in [Section "Historic hero"](#).

```
from networkx import MultiGraph

# create a new graph
my_graph = MultiGraph()

# create four nodes
my_graph.add_node(1)
my_graph.add_node(2)
my_graph.add_node(3)
my_graph.add_node(4)

# create seven edges
my_graph.add_edge(1, 2)
my_graph.add_edge(1, 2)
my_graph.add_edge(1, 3)
my_graph.add_edge(1, 4)
my_graph.add_edge(1, 4)
my_graph.add_edge(2, 3)
my_graph.add_edge(3, 4)
```

**Listing 2.** Another undirected graph that maps precisely the situation depicted in [Figure 3](#) since it allows the creation of multiple arcs between the same two nodes. The source code of this listing is available [as part of the material of the course](#).

To enable the creation of multiple edges between two nodes, we have to use a different kind of undirected graph using the constructor `MultiGraph()`. This graph accepts multiple edges between nodes by calling the method `<graph>.add_edge(<node_1>, <node_2>)` several times, and the method `<graph>.remove_node(<node>)` will remove all the edges involving that input node, as usual. [Listing 2](#) introduces an example of this kind of graph that maps precisely the one introduced in [Figure 3](#).

Two additional methods are fundamental to understanding how a graph is composed and which nodes link to the others. They are `<graph>.nodes()` and `<graph>.edges()`, each returning particular kind of lists (called `NodeView` and `EdgeView` respectively) that can be iterated by means of a *for-each* loop as usual. It is also possible to understand what are the nodes linked by a target node using the adjacency variable `<graph>.adj[<node>]`. This operation returns an `AtlasView`: a kind of dictionary containing all the nodes reachable from `<node>`, where each dictionary key represents one of these nodes.

```

from networkx import Graph

# create a new graph
my_graph = Graph() # it works also with MultiGraph

my_graph.add_node(1) # no additional data
my_graph.add_node(2, name="John", surname="Doe") # additional data
my_graph.add_node(3)

my_graph.nodes()
# Returns NodeView (tuple) with all the nodes:
# NodeView((1, 2, 3))

my_graph.nodes(data=True)
# Returns a NodeDataView (like a dictionary) with nodes + data:
# NodeDataView({1: {}, 2: {'name': 'John', 'surname': 'Doe'}, 3: {}})

my_graph.add_edge(1, 2) # no additional data
my_graph.add_edge(1, 3, weight=4) # additional data

my_graph.edges()
# Returns an EdgeView (of two-item tuples) with all the edges:
# EdgeView([(1, 2), (1, 3)])

my_graph.edges(data=True)
# Returns an EdgeDataView (of three-item tuples) with edges + data:
# EdgeDataView([(1, 2, {}), (1, 3, {'weight': 4})])

my_graph.adj[1]
# This returns an AtlasView (like a dictionary) containing all the
# nodes that are reachable from an input one + data of edges:
# AtlasView({2: {}, 3: {'weight': 4}})

```

**Listing 3.** The use of additional data for enriching nodes and edges of graphs. The source code of this listing is available [as part of the material of the course](#).

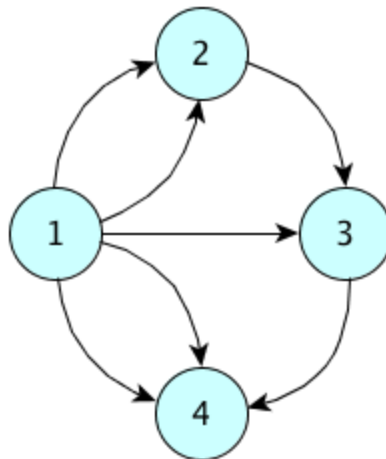
The value associated with each node, in this case, is another dictionary that is initialised empty if one did not specify any additional information explicitly. This information, or *attribute* in NetworkX, can be specified when one build the edge connecting the two nodes. In particular, we use one or more pairs of a parameter and the value assigned to him via =, as shown in [Listing 3](#). We can do the same kind of assignments to nodes. In addition, these information can be also shown by executing the aforementioned methods `nodes()` and `edges()` by specifying the named parameter `data` as `True`, i.e. `<graph>.nodes(data=True)` and `<graph>.edges(data=True)`. This use of naming the parameters explicitly in Python when

one wants to execute a method (or a function) is permissible by Python, as explained in [its documentation](#).

## Directed graphs

According to the NetworkX package, we can create a directed graph with the constructor `DiGraph()`. In NetworkX, a directed graph has the same methods of undirected graphs, presented in [Section "Undirected graphs"](#). However, in this case, the order between `<node_1>` and `<node_2>` in the methods for adding and removing an edge is meaningful, since an edge specifies a particular direction: `<node_1>` is the source node, while `<node_2>` is the target node.

Also, it is possible to specify more than one edge between two nodes by using the constructor `MultiDiGraph()`. For instance, [Figure 4](#) shows the abstract diagram of the graph implemented in [Listing 2](#) if the constructor `MultiDiGraph()` would be used instead of `MultiGraph()`.



**Figure 4.** The diagram of the graph depicted in [Figure 3](#) and implemented in [Listing 2](#) if we use `MultiDiGraph()` instead of `MultiGraph()`.

## Exercises

1. Consider the list of co-authors of Tim Berners-Lee as illustrated in the right box at <http://dblp.uni-trier.de/pers/hd/b/Berners=Lee:Tim>. Build an undirected graph containing Tim Berners Lee as the central node, linking it to five nodes representing his top-five co-authors. Also, specify the *weight* of each edge as an attribute, where the value of the weight is the number of bibliographic resources (articles, proceedings, etc.) Tim Berners-Lee has co-authored with the person linked by that edge.

2. Create a directed graph that relates the actors [Brad Pitt](#), [Eva Green](#), [George Clooney](#), [Catherine Zeta-Jones](#), [Johnny Depp](#), and [Helena Bonham Carter](#) to the following movies: [Ocean's Twelve](#), [Fight Club](#), [Dark Shadows](#).

## Acknowledgements

The author wants to thank one of the students of the [Digital Humanities and Digital Knowledge second-cycle degree of the University of Bologna](#), [Chantal Lengua](#) and [Carlo Teo Pedretti](#), for having suggested corrections to the text of this chapter.

## References

Albert, R., Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74 (47): 47-97. DOI: <https://doi.org/10.1103/RevModPhys.74.47>, freely available at <https://arxiv.org/pdf/cond-mat/0106096.pdf>

Euler, L. (1741). *Solutio problematis ad geometriam situs pertinentis*. *Commentarii academiae scientiarum Petropolitanae*, 8 (1741): 128-140. <http://eulerarchive.maa.org/docs/originals/E053.pdf> (last visited 10 December 2017)