Organising information: graphs

Author(s)
Silvio Peroni – silvio.peroni@unibo.it
Department of Classical Philology and Italian Studies, University of Bologna, Bologna, Italy

Keywords
Edges and nodes; Euler; Graph; Königsberg

Abstract

These lecture notes introduce 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 18th century who introduced for the very first time a new mathematical field called graph theory.

Historic hero: Euler

Leonhard Euler (shown in Figure 1) was one of the most important men of Science of the whole history. His contributions in Mathematics, Physics, Astronomy, Logics, among the others, were disruptive and even started pretty new disciplines that were not explored at all before his contributions. He spent most of his life in Saint Petersburg in Russia. Among the mathematical problems he dealt with, there is one related to a particular funny 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. The problem could be stated as follows: is it possible to walk around the city and to cross each of the bridges once and only once? Several people have tried to propose a solution to this enigma before Euler, but he was able to demonstrate it by means of a 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.

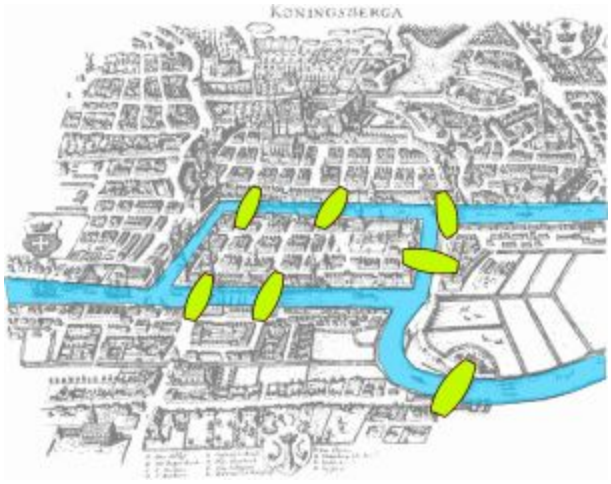

Figure 2. A representation of the seven bridges in Königsberg. Figure by Bogdan Giuşcă, source: https://commons.wikimedia.org/wiki/File:Konigsberg_bridges.png.

In order to do that, he abstractly described the four lands in Königsberg divided by the river as *nodes* of a network, where each *edge* between two nodes actually represents one of the bridges of the city. The illustration about his abstract representation is introduced in Figure 3. By using this abstract notion, known as *graph*, he was able to demonstrate that there is *no solution* to the aforementioned problem of the seven bridges of Königsberg.

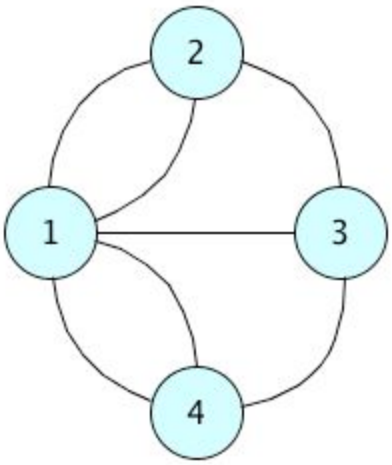

Figure 3. An abstract representation of the seven bridges in Königsberg by means of a graph.

The solution of the problem was entirely based 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. This is a practical implication derived from the moves that one has to do to enter and then go out from a node. In fact, an edge is followed every time one enters in a node, and another edge is needed to go out from that node as well. Thus, at least, each non-starting and non-ending node must have mandatorily 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 main data structure in Computer Science and Computational Thinking. They are used to describe in abstract terms several well-known situations like routes between cities, connections to people you know in social networks, and the organisation of links between Web pages [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 (like the one used by Euler for solving the seven bridge of Königsberg problem), where an edge can be traversed in one way or the other indifferently, and *directed* graphs, where the edge has a clear specification of the node-to-node direction that can be followed.

While in Python, as it happens for the trees, there is not a built-in class defining this type of objects, there are several external libraries that implement them. Among the most used and famous, there is NetworkX, which makes available the common constructs for creating and traversing graphs, as well as additional functions for analysing them for different purposes, such as for the analysis of social networks.

Undirected graphs

An undirected graph can be created by means of the constructor `Graph()`. When a new graph is created, it will be used for creating all the nodes and all the edges.

```
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.

It is worth mentioning that the NetworkX package allows us to associate as a node any possible **immutable object** definable in Python, that can be, thus, connected by means of 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 has no effect on the graph;
- `<graph>.add_edge(<node_1>, <node_2>)` adds and 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.

An example of a graph is depicted in Listing 1. It creates a structure similar to the one introduced in Figure 3 except that it is not possible to create multiple arcs between two nodes. Thus, using this specific constructor it is not possible 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.

In order to enable the creation of multiple edges between two nodes, we have to use a different kind of undirected graph by means of the constructor `MultiGraph()`. This particular graph accepts multiple edges between nodes by calling several times the method `<graph>.add_edge(<node_1>, <node_2>)`, and the method `<graph>.remove_node(<node>)` will actually remove just one of the added edges and not all of them. An example of this kind of graphs, that maps precisely the one introduced in Figure 3, is shown in Listing 2.

There are at least two additional methods that are fundamental in order understand to a graph is composed and which nodes are linked with the others. They are `<graph>.nodes()` and `<graph>.edges()` that return particular kind of lists (called `NodeView` and `EdgeView` respectively) that can be iterated by means of a foreach loop as usual. It is also possible to

understand what are the nodes to which a target node is linked with by means of the adjacency variable `<graph>.adj[<node>]`. This operation returns an `AtlasView`, which is a kind of dictionary containing all the nodes that can be reached starting from `<node>`, where each key of the dictionary actually represent 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 actually another dictionary which is initialised empty if one did not specify any additional information explicitly. This information, or *attribute* as it is called in NetworkX, can be specified when one build the edge connecting the

two nodes by using one or more pairs of a parameter and the value assigned to him via `=`, as shown in Listing 3. The same kind of assignments can be done also 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 explicitly the parameters in Python when one wants to execute a method (or a function) is totally admissible by Python, as explained in its documentation.

Directed graphs

According to the NetworkX package, a directed graph can be created with the constructor `DiGraph()`. It shares exactly the same methods presented for the undirected graphs 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 now a particular direction: `<node_1>` is the source node, while `<node_2>` is the target node.

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

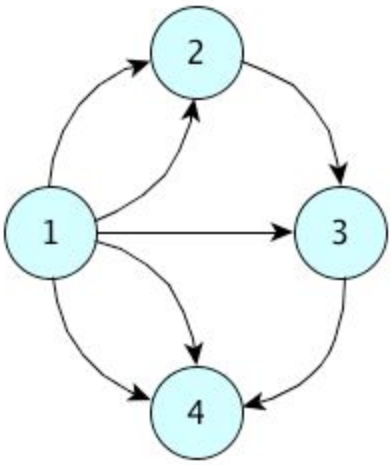

Figure 4. The diagram of the graph depicted in Figure 3 and implemented in Listing 2 if a `MultiDiGraph()` is used instead of a `MultiGraph()`.

Exercises

1. Consider the list of co-authors of Tim Berners-Lee as illustrated in the write box at http://dblp.uni-trier.de/pers/hd/b/Berners=Lee:Tim. Build an undirected graph that

contains Tim Berners Lee as the central node and that is linked to other five nodes representing his top-five co-authors. In addition, 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 which 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*.

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)