A short guide to `networkx`

What is `networkx`?

`networkx` is a popular Python library for implementing networks. It has a variety of networks, and a large number of preimplemented algorithms (like Dijkstra's). It also has the benefit of being other people's code.

Installing `networkx`

`networkx` is not a part of your standard 233 installation. Use

```
conda activate 233env
conda install networkx
```

to install it. We will be working with version 2.4 (the latest stable version at time of writing).

If you are using the base conda installation (not using the 233env), you will likely already have this. Check with:

```
conda list networkx
```

If your version is not 2.4, upgrade it:

```
conda upgrade networkx
```

Getting started with the Auckland network

For this project, we are providing you with a prebuilt network of (most of) Auckland's public transport system and rest homes. Utility functions in project_utils.py will help you get started.

Reading in the data

We can use the `read_network` and `get_rest_homes` functions to read in the data:

```
from project_utils import *
auckland = read_network('network.graphml')
rest_homes = get_rest_homes('rest_homes.txt')
```

The `auckland` object will be a `networkx.Graph` instance, which contains the network information. The `rest_homes` object is a list of the names of the rest homes.

The structure of the graph

Nodes in the `auckland` graph will consist of two types, which have different name types:

- nodes with numerical names: these represent nodes that make up the road system of Auckland's public transport system
- nodes with string names: these are the rest homes, and are enumerated in `rest_homes`

All nodes have two attributes:

- `lat`: Latitude
- `lng`: Longitude

Edges will connect nodes together. All edges have one attribute:

- `weight`: The time (in hours) it takes to travel along the edge.

Algorithms on the graph

We can use in-built `networkx` algorithms to compute shortest paths. For example, finding a shortest path between Everil Orr and Kumeu Village can be done as follows:

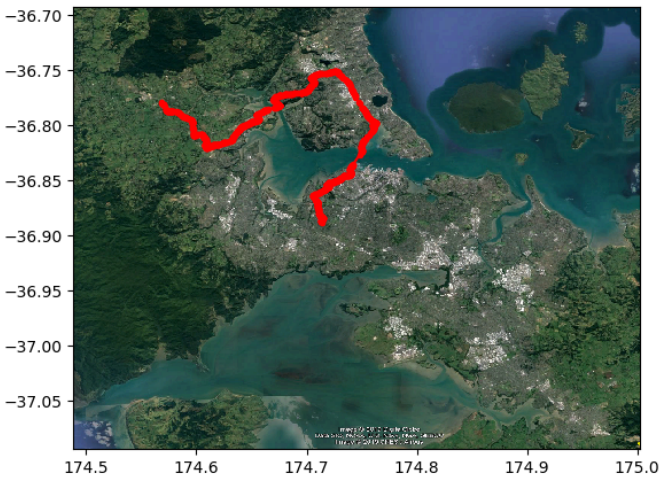
```
import networkx as nx

path = nx.shortest_path(auckland,
                        'Everil Orr', 'Kumeu Village',
                        weight='weight')
distance = nx.shortest_path_length(auckland,
                                   'Everil Orr', 'Kumeu Village',
                                   weight='weight')
```

Here, we specify that we use the `weight` attribute to weight the edges for the shortest path algorithm.

We can plot the path using the `plot_path` function from `project_utils`:

```
# continuing from above
plot_path(auckland, path, save=False)
```



Additional tips and tricks

Node information

We can access a node's information through the `nodes` attribute of the network:

```
# Getting the location of Auckland Airport
auckland.nodes['Auckland Airport']
```

```
>>> {'lat': -37.0082, 'lng': 174.785}
```

```
# Accessing the information in the node
auckland.nodes['Auckland Airport']['lat']
```

```
>>> -37.0082
```

Node neighbours

We can access a node's neighbours in two ways:

```
# Method 1
list(auckland.neighbors('Auckland Airport'))
```

```
>>> [1048403197]
```

```
# Method 2
list(auckland['Auckland Airport'])
```

```
>>> [1048403197]
```

Note that if we do not cast the output of each of those calls to a list, we get an *iterator*:

```
# Method 1
auckland.neighbors('Auckland Airport')
```

```
>>> <dict_keyiterator at 0x1a1711c8d18>
```

```
# Method 2
auckland['Auckland Airport']
```

```
>>> AtlasView({1048403197: {'weight': 0.3629710768924994}})
```

We cannot index these, but we can use these in a for loop:

```
# Using Method 1 as an example
for neighbour in auckland.neighbors('Auckland Airport'):
    print(neighbour)
    print(auckland.nodes[neighbour])
```

```
>>> 1048403197
>>> {'lat': -37.0050009, 'lng': 174.7842862}
```

Edge lookups

```
airport_edge = auckland.get_edge_data('Auckland Airport', 1048403197)
airport_edge
```

```
>>> {'weight': 0.3629710768924994}
```

```
airport_edge['weight']
```

```
>>> 0.3629710768924994
```

An algorithm for generating pairs from a sequence

```
nodes = [1, 2, 3, 4]
pairs = [pair for pair in zip(nodes[:-1], nodes[1:])]

pairs
```

```
>>> [(1, 2), (2, 3), (3, 4)]
```

Sorting using Python built-ins

```
# Sort the points by their distance from the origin
points = [(-1, 3), (2, 1), (3, 4), (-1, 1), (5, 0)]
```

```
def distance_from_origin(point):
    # by Pythagoras' Theorem
    return (point[0]**2 + point[1]**2)**(0.5)

# the key argument acts as a distance function
sorted_points = sorted(points, key=distance_from_origin)
sorted_points
```

```
>>> [(-1, 1), (2, 1), (-1, 3), (3, 4), (5, 0)]
```

Further documentation

The networkx 2.4 official documentation: https://networkx.github.io/documentation/stable/