Python Summer School, Day 1
Exercises, Part II

Instructors: Pietro Berkes and Niko Wilbert

Exercise 1 - Design and implement a graph library

Goal: Design a set of classes to represent graphs; implement the library and use unittests.

a) Discuss with your partner a possible design for a set of objects that represent graphs. The graph class should be general enough to implement directed and weighted edges. The nodes should be able to carry additional data or behaviour. Think about the parts that compose a graph, and how they relate to each other.

   - Discuss possible object oriented ways to design this.
   - What are possible applications of graphs? Which parts of the code would vary in these applications?

   Don't read the rest of the exercise before completing a).

b) Implement three classes, one representing nodes (Node), one for edges (Edge), and one the whole graphs (Graph). Nodes and edges should be able should be able to give information like:
Who are your neighbors? Which nodes do you connect? Graph is an object that manages a set of nodes and edges. It has methods to add / remove new nodes and edges.

c) Write test cases for these classes. For example, you should test that when a node is removed from a graph, the edges that connect it to its neighbors should also be removed.

d) Discuss advantages and disadvantages of this design in a file called `design_discussion.txt` (just some brief notes or keywords). What happens to the design in an application with thousands of nodes and what could you do about it?

Exercise 2 - Travel planner

Goals: Extend the graph library to solve a search problem.

In this exercise, your goal is to write a travel planning application based on the classes of Exercise 1. We want to represent a set of cities as nodes in a graph, with edges between nodes representing different kinds of transportation.

a) Create a network of cities that are connected by different modes of transportation (Train, Plane, Boat). The edges should be directed and have two kinds of weights: travel time and cost.

b) Extend the Graph class with a method that is able to search for the quickest or the cheapest path between two given cities. You'll have to implement an algorithm to find the shortest path in a weighted graph, starting from any node. This can be done using Dijkstra's algorithm, which is described at `http://en.wikipedia.org/wiki/Dijkstra_algorithm`. Note that due to the weights you will have to iterate through the edges or nodes in the order of the least overall path cost.

   The Python library contains a `heapq` module, which might be useful for the implementation. The weight function will be in the first case the cost of the transport, in the second the travel time.

   - The output of the search method should be a nice representation of the shortest path, e.g., as a string ("Taking the plane from city A to B, then taking the plane from B to C,...").
   - Don't forget to write tests for the algorithm (for example, does it work for multiple edges from one city to another).
   - Think about flexibility of your design. How hard would it be to add additional modes of transportation or add a new type of cost (like carbon footprint)?

c) Implement the following city graph as an example and print the quickest and cheapest path from Berlin to Cologne: