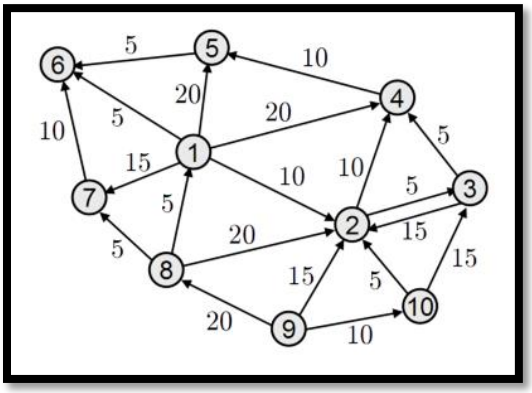
Mini-project #1 – Python Programming

Background

This project is concerned with the implementation of mathematical graphs and their usages. The basic definition of a graph is a set of **nodes** where some pairs of nodes are connected by **edges**, and where these edges have **weights**. In some graphs the edges have directions, and then the graph is called **directional**. A node is another's **neighbor** (or adjacent) if there is an edge connecting the former to the latter (considering direction if relevant). If it is possible to travel from a specific node to another by going through neighbors, then we say there is a **path** connecting these nodes.



To illustrate, the figure on the right shows a directional graph with 10 nodes and 20 edges. Node 6 is a neighbor of node 1, while the opposite is not true, and the weight of the edge directed from node 1 to node 6 is 5. There is a path from node 2 to node 6 (2-4-5-6), but there is no path from node 6 to node 2.

Part I – Non-directional graph

Task 1 – The *Node* class

Implement the *Node* class with the following properties:

- Attributes
  - *name* (str) – the "name" of the node
  - *adjacents* (dict) – a dictionary of the nodes that are paired with the node
    - Key – neighbor name
    - Value – edge weight
- Methods
  - *__init__(self, name)*
  - *__str__(self)*
  - *__eq__(self, other)* – based on the *name* attribute
  - *__ne__(self, other)*
  - *neighbors(self)* – returns a list of the names of the adjacent nodes

- *is_neighbor(self, name)* – returns True if there is an edge connecting the node to *name*
- *add_neighbor(self, name, weight=1)*
- *remove_neighbor(self, name)*
- *get_weight(self, name)* – returns the weight of the relevant edge (if exists)
- *is_isolated(self, name)* – returns *True* if *name* has no neighbors

The file node.py contains the class “skeleton” for your convenience.

Task 2 – The Graph class

Implement the Graph class with the following properties:

- Attributes
  - *nodes* (dict) – the full description of the graph
    - Key – node name
    - Value – corresponding node object
- Methods
  - *__init__(self, *nodes)*
  - *__str__(self)*
  - *__len__(self)* – returns the number of nodes in the graph
  - *__contains__(self, name)* – return *True* if a node called *name* is in the graph
  - *__getitem__(self, name)* – returns the Node object whose name is *name*
  - *__add__(self, other)* – returns a new Graph object that includes all the nodes and edges of *self* and *other* (excluding duplicates)
  - *names(self)* – returns a list of the names of all the nodes in the graph
  - *edges(self)* – returns a list of all the edges (tuples) in the graph
  - *add_node(self, node_obj)*
  - *remove_node(self, node_name)*
  - *is_edge(self, frm_name, to_name)* – returns *True* if *to_name* is a neighbor of *frm_name*
  - *add_edge(self, frm_name, to_name, weight=1)*
  - *remove_edge(self, frm_name, to_name)*
  - *get_edge_weight(self, frm_name, to_name)*
  - *get_path_weight(self, path)* – returns the total weight of the given path, where path is an iterable of nodes names.

- *find_path(self, frm_name, to_name)* – returns a path from *frm_name* to *to_name* (if exists)
- *find_shortest_path(self, frm_name, to_name)* – returns the path from *frm_name* to *to_name* which has the minimum total weight
  - note: path finding is usually implemented with recursion. We didn't learn recursion in our course, so I recommend implementing a non-recursive algorithm like "breadth-first search" or "depth-first search".

The file node.py contains the class "skeleton" for your convenience.

Task 3 – The social network implementation

The file social.txt describes chronologically the intrigues among 14 friends. Use the data in the file and the classes you've defined to answer the following questions.

Question 1

What was the highest number of simultaneous friendships?

Question 2

What was the maximum number of friends Reuben had simultaneously?

Question 3

At the current graph (considering all the data of the file), what is the maximal path between nodes in the graph?

Question 4

Implement a function called *suggest_friend(graph, node_name)* that returns the name of the node with the highest number of common friends with *node_name*, which is not already one of his friends.

Part II – Directional graph

Task 1 – The DirectedGraph class

Implement the DirectedGraph class.

Task 2 – The roadmap implementation

The files travelsEW.csv and travelsWE.csv record a large number of travels made by people from five regions in the country.

From each file create a graph whose nodes are the country regions, and whose edges are the roads (if a travel was not recorded between country regions, then it means such road does not exist). The weight of each edge is defined as the average of all the travels done on that road.

When the two graphs are ready, add them together to create the complete graph of the roadmap.