Week 12 Practice Problems

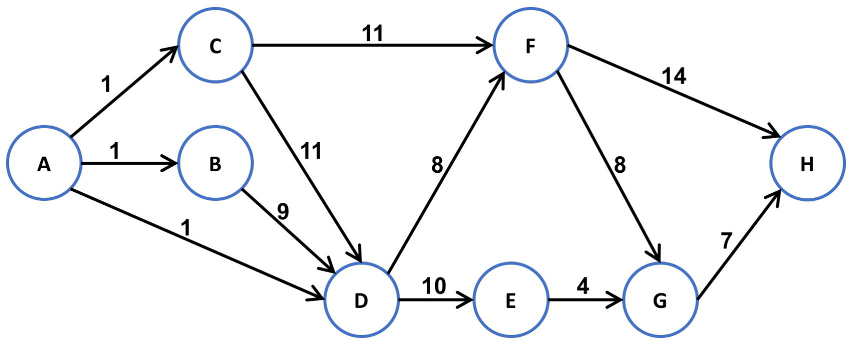
#1. A sorted linked list is a linked list where all the elements are maintained in order. For example, if your cargo are integers, the list is in ascending order. Modify the LinkedList created in lecture to implement a sorted linked list.

Implement the following methods (some will be the same as LinkedList):

- __init__, __str__
- len(): returns the number of elements in the list
- find(): returns a refernence ot the Node containing the number passed in
- add(): adds an element to the right place in the list
- remove(): removes an element containing the number passed in a returns a reference to the Node

#2. Modify the doubly linked list implementation from Week 11 Q1 to be a sorted, doubly linked list.

#3. The figure below shows a graph made up of nodes (the circles) and edges (the arrows). The letter in a node is its label, the number of the edge is its weight. Such graphs have substantial uses in mathematics, computer science, biology, and engineering (e.g., look up the Wikipedia page on "graph theory").



a) Create two classes: Node and Edge.

The Node class should represent a node's label and all of the edges that leave the node (i.e., the arrow points away). So, for example Node D needs to represent that it has two out-going edges (to E and to F). Note that each node can have a different number of outgoing edges. This suggests that the edges of a node should be represented in something like a list. Write a constructor, a `__str__` method, and a method called `add_outgoing_edge` which allows you to add an outgoing edge to a Node.

The Edge class should represent the weight of an edge and two Nodes: its head (the node where the arrow starts) and its tail (the node where the arrow ends).

b) The file graph.txt contains a text format of the graph above. (Look at the file – its format should be clear if you compare it to the graph above).

Write a function called `create_graph` which takes in a string corresponding to a filename, reads the file and constructs the graph. That is, you should create the necessary Node and Edge objects and "assemble" them. Each Node should have a list of its outgoing edges, each Edge should know its head and tail Nodes.

You should return some data structure representing the graph from the `create_graph` function. You should think about what would be best given what you are asked to do in part c.

c) Add a new method to your Node class called `find_path`. It should take in another Node object (called *end*) and return an ordered list of Nodes that connect the current node (i.e., self) to the target or None if no such path exists. The list should contain both the start node (i.e., self) and the end node and consecutive nodes must be connected by an edge in the forward direction.

Note that you can only travel along an Edge in the direction of the arrow.

This is an advanced method that you may not be able to do. It is here as a challenge.