Assignment 23, Part II: Kruskal
Due in class Thursday, 3/21

March 18, 2019
CS: DS&A
Proof School

The task for Part II is to implement Kruskal's algorithm. Your implementation should be the $O(m \log n)$ one that uses the Union-Find data structure. You'll have to write that data structure.

Since this is the second half of Assignment 23, please call your file `assignment_23_2.py`.

- Use `weighted_graph.py` again, and your binary heap. (You do *not* have to initialize the binary heap in the clever, $O(m)$ way. Inserting edge by edge is fine.)
- Your function should be called `MST_Kruskal`. As with the previous assignment, it should take a `WeightedGraph` object, and return an ordered pair. The first element of the pair is a set of `WeightedEdge` objects corresponding to the minimal spanning tree produced by Kruskal's algorithm. The second element is the sum of the weights in that tree.
- You have a choice in how to handle the Union-Find data structure. If you want, you can just use whatever extra dictionaries you need in your `MST_Kruskal` function to support the algorithm discussed in class. (You can use a vertex-name to vertex-name dictionary, or an array of indices with dictionaries converting between indices and vertex names.)
- But the "right way" to do this is to write a separate `Union-Find` class. The API for this class consists of three functions:
  - `__init__` takes a set of objects. These are the objects that will be grouped.
  - `in_same_group` takes two objects, and returns `True` or `False` depending on whether the objects are in the same group.
  - `union` takes two objects, and merges the groups containing those objects.

  In this approach, the `MST_Kruskal` function initializes a `Union-Find` object, and interacts with it via its API. There are no extra dictionaries in the `MST_Kruskal` function directly; they are all handled behind-the-scenes by the `Union-Find` class.

Have fun and good luck!!!