February 25, 2019
CS: DS&A
Proof School

Assignment 20: DFS on Graphs Due in class Monday, 3/04

This is a hard one. Don't put it off. Let's talk about it later this week if you get stuck.

Write your functions in a file called `assignment_20.py` and submit by Dropbox.

Problem 1. Write `DFS_1(G)` that takes a graph G and prints out the node names in the order of the depth-first search. As usual, *think this through before you code!* I found this trickier than expected. Here are some things to think about.

- With what node should the DFS start? That's up to you.
- How recursive should this be? That's up to you also! Our pseudocode was certainly recursive.
- Your code should run correctly on graphs with more than one connected component. For instance, it should run on a graph that just has n vertices and no edges!
- Your code should be able to check whether a node has begun processing in constant time.

Problem 2. Modify your code a little bit to have it return the number of connected components of the graph. (So we can figure out how many connected components there are using DFS in linear time! Yay!)

How many components does a random graph have? I think it's an interesting question. Play around with `make_random_graph` with various values for `n` and `edge_probability`. For each choice of parameters, run your code a bunch of times and find the average number of connected components. Summarize your results.

Problem 3. Write `DFS_2(G)`, which is an improvement as follows. Instead of printing the node names, it returns a list of triples of the form `(node_name, start_time, end_time)`, in the order that the nodes were processed in the DFS. The start time and end time were defined in class.

For instance, suppose G is a cyclic graph with ten nodes A through J. Suppose we start our DFS at node A, and next process B (as opposed to J, which would have been fine). Then the returned object should be:

```
[('A', 0, 19), ('B', 1, 18), ('C', 2, 17), ('D', 3, 16), ('E', 4, 15), ('F',
     5, 14), ('G', 6, 13), ('H', 7, 12), ('I', 8, 11), ('J', 9, 10)]
```