CMPS 2200 Recitation 07

Today we'll learn more about graphs and how to navigate them. As usual, code goes in `main.py`, and short answer in `answers.md`.

1 Let's assume we're using the "Map of Neighbors" representation for undirected graphs. The provided `make_undirected_graph` function will make a graph using this representation given a list of edge tuples.

We'll start by implementing the `reachable` function, which identifies the set of nodes that are reachable from a given `start_node`.

As discussed in lecture, we'll maintain a set called `frontier` that keeps track of which nodes we will visit next. We initialize the set to be the start node. We then perform a loop where we pop a single node off the frontier, visit its neighbors, and update the `result` and `frontier` sets appropriately. At the end of the loop, `result` should contain all the nodes that are reachable from `start_node`.

Complete the `reachable` implementation and test with `test_reachable`. Think about how to make this efficient and ensure we don't revisit nodes unnecessarily.

.
.
.

2 What is the work of `reachable`, assuming n nodes and m edges?

put answer in answers.md

.
.
.

3 Next, we will use the `reachable` function to determine if a graph is connected or not. Complete the `connected` function and test with `test_connected`.

.
.
.

4 What is the worst case number of times we need to call `reachable` to determine if a graph is connected?

put answer in answers.md

.
.
.

5 What is the work of `connected`, assuming n nodes and m edges?

put answer in answers.md

.
.
.

6 Next, we'll use `reachable` to determine the number of connected components in a graph. Complete `n_components` and test with `test_n_components`. Again, think about how to minimize the number of calles to `reachable` you must make.

.
.
.

7 What if we switched the graph representation to an adjacency matrix? Would the work of `reachable` change? If so, what would it be? If not, why not?

put answer in answers.md

.
.
.