Last name _____

First name _____

LARSON—MATH 556–SAGE WORKSHEET 11

Spannng Trees

1. Log in to your Sage/Cocalc account.

   (a) Start Chrome browser.

   (b) Go to `http://cocalc.com`

   (c) Click "Sign In".

   (d) Click project **Classroom Worksheets**.

   (e) Click "New", call it **s11**, then click "Sage Worksheet".

   A **spanning tree** of a connected graph G is a subgraph of G which is a tree and contains all the vertices of G. We proved that every connected graph has a spanning tree.

   An intuitive idea for finding a spanning tree is to iteratively remove any edge which does not disconnect the graph (and, thus, is in a cycle). We will write an algorithm that implements this idea and then test it.

```
def find_spanning_tree(g):
    while not g.is_tree():
        E = g.edges()
        for e in E:
            g.delete_edge(e)
            if g.is_connected():
                break
            else:
                g.add_edge(e)
    return g
```

   This code consists of a single `while` loop. Here's how it works. If the graph g input to the while loop is not a tree then the loop gets executed. As soon as the input graph g is a tree, the loop gets skipped and we go right to the return statement.

   Inside the while loop, we first find the set of edges *of the graph input to the while loop* (this shrinks by a edge in every iteration). In the `for` loop, we iterate over the set of edges of the graph input to the while loop. During the for loop, we remove the edge and check if the resulting graph is connected. If it is not, we put the edge back and go to the next edge in our list of edges to check in the for loop. If the graph we get *is* connected we break out of the for loop, and go back to the top of the while loop.

2. Evaluate `pete = graphs.PetersenGraph()`. Now lets find a spanning tree T. Evaluate `T = find_spanning_tree(pete)`. To see what we get, evaluate `T.show()`. Draw the result.

3. Now let's draw the Petersen graph again: `pete.show()`. What happened!!?!?

   If we don't want our function to change the input graph g, we need to make a copy of g before we start removing edges. Try:

```
def find_spanning_tree(g):
    gc = copy(g)
    while not gc.is_tree():
        E = gc.edges()
        for e in E:
            gc.delete_edge(e)
            if gc.is_connected():
                break
            else:
                gc.add_edge(e)
    return gc
```

4. Evaluate `pete = graphs.PetersenGraph()`. Now use this new function to find a spanning tree T of the Petersen graph. `show` "pete" to check that our original graph hasn't been changed.

5. Let `k_3_2 = graphs.CompleteBipartiteGraph(3,2)`. Use our function to find and show a spanning tree.

6. Add `gc.show()` as the first edge of the while loop so that you can watch the evolution of your spanning tree. Run your code again on the Petersen graph. It doesn't print the final spanning tree. How can you modify your code to fix that?

7. Run your code on `k_3_2` and draw all the graphs you get.

8. Our code never checks if the input graph is connected? What should we do in the case where the input graph is not connected (and thus does not have a spanning tree)? How can we modify our code?