LARSON—MATH 255–CLASSROOM WORKSHEET 34
Graph Theory & Computationally Hard Problems.

1. (a) Start the Chrome browser.
   (b) Go to `http://cocalc.com`
   (c) Login using **your VCU email address** .
   (d) Click on our class Project.
   (e) Click "New", then "Worksheets", then call it **c34**.
   (f) For each problem number, label it in the Sage cell where the work is. So for Problem 2, the first line of the cell should be `#Problem 2`.

Graphs & Graph Theory

A **graph** is a mathematical object consisting of *dots* and *lines* (also called *vertices* and *edges*). The *order* of a graph is the number of vertices it has. The *size* of a graph is the number of edges it has.

Now we will define a new graph function for computing the hard-to-compute *stability number* of a graph. The *stability number* of a graph is the largest number of vertices in the graph that have no edges between them. So if `g=graphs.PetersenGraph()`, its stability number is 4.

2. Find the stability number of `c5=graphs.CycleGraph(5)` by hand. What is the largest stable set of vertices that you can find?

3. Find the stability number for `p2=graphs.PathGraph(2)`. Use `p2.show()` to help you visualize.

4. Find the stability number for `p3=graphs.PathGraph(3)`. Use the `show()` method to help you visualize.

5. Find the stability number for `p4=graphs.PathGraph(4)`. Use the `show()` method to help you visualize.

6. Find the stability number for `p5=graphs.PathGraph(5)`. Use the `show()` method to help you visualize.

7. Find the stability number for `c5=graphs.CompleteGraph(5)`. Use `c5.show()` to help you visualize.

Now we will write an algorithm to find the stability number.

8. First let's write a test for whether or not a specific set S of vertices is stable. Then one possible algorithm involves testing every subset of vertices. If S is stable then the test for (i, j) will not be an edge (will not be in the list of edges) of graph g for each possible pair i, j of vertices.

```
def is_stable(g, S):
    E=g.edges(labels=False)
    for i in S:
        for j in S:
            if (i,j) in E:
                return False
    return True
```

9. Test this for some (sub)sets of vertices of the Petersen graph.

Subsets and Counting

The "naive" (and inefficient) way to find a largest stable set in a graph is to test every subset of vertices, check if it is stable, and then keep track of the largest one you've seen up to that point. Let's see why its inefficient.

10. Suppose you have a very small graph, with just 3 vertices. How many subsets of the vertex set would you have to look at? Run: `Subsets([0,1,2])`.

11. How many is that? We can turn the *Subset generator* into a list and use the Python list counting function *len*. Run *len(Subsets([0,1,2]))*. (Don't ever try this on a set that's much larger—the list gets created in memory and might freeze your computer as it runs out of RAM).

12. How many subsets will we create for a graph that has n vertices? For each vertex v we will generate every possible set containing v and every possible set not containing v. So that will be 2 possibilities for each of the n vertices. That's 2^n subsets. (For our earlier example, thats $2^3 = 8$ subsets).

13. (**Challenge**). Write a function `stability_number(g)` that takes a graph g as input, forms every possible subset of vertices, checks if that set of vertices is stable, and updates the largest stable set found so far, and returns the number of vertices in a largest stable set.

Getting your classwork recorded

When you are done, before you leave class...

(a) Click the "Make pdf" (Adobe symbol) icon and make a pdf of this worksheet. (If CoCalc hangs, click the printer icon, then "Open", then print or make a pdf using your browser).

(b) Send me an email with an informative header like "Math 255 - c34 worksheet attached" (so that it will be properly recorded).