Last name _____

First name _____

LARSON—OPER 731—CLASSROOM WORKSHEET 19
Vertex Packing Linear Programs

1. **Log in to VCU's Athena cluster**.

   The following directions assume you have an Athena account, that you have set up Sage, and that you have set up (using `make`) the CONJECTURING program.

   (a) Start the Chrome browser.

   (b) If you are off-campus, you'll need to connect to the VPN first.

   (c) Then go to `https://athena3.hprc.vcu.edu`

   (d) Login using your VCU EID as your username, and your corresponding VCU password.

   (e) Click the Apps button and a `Sage` session. The default options are fine. This will take a couple of minutes.

   (f) Click the Apps button and start an "athena shell access" session (this will give you a terminal window, where we can issue commands).

   (g) Your Sage session will first say "Queued", then "Starting". When it is ready you will see a button that says, "Connect to Sage". Click that.

   (h) You should then get an "untitled" interactive-Python notebook (ipynb), or the last file you had open the previous time you used Athena.

   (i) When your notebook opens look on the upper-right to make sure the SageMath kernel is running (if it isn't you can change the *kernel*).

Vertex Packing

A *vertex packing* in a graph is a set of vertices that have no edges between them. The *vertex packing number* of a graph is the size (cardinality) of a largest vertex packing.

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

3. Find the vertex packing number for `pete=graphs.PetersenGraph()`. Use `c5.show()` to help you visualize.

   Now we will define a new graph function for computing the hard-to-compute *vertex packing number* of a graph. We'll want to look at each possible pair of vertices in a set S of vertices to see if there's an edge between them. One way to do that is see if a given pair is in the list of edges. We can get the edges of a graph g using `g.edges(labels=False)`. Try it for the Petersen graph and $c5$.

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

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

Subsets and Counting

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

6. 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])`.

7. 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).

8. (**Computational Aside**) A graph with 100 vertices is considered "small" (we'd like to find information about graphs with millions of vertices). How long (minutes, hours, days, years?) would it take for an algorithm to perform 2^{100} computational steps. Assume you can perform one million (10^6) steps per second.

9. Here is a function `vertex_packing_number(g)` that takes a graph g as input, forms every possible subset of vertices, checks if that set of vertices is a packing, and updates the largest vertex packing found so far, and returns the number of vertices in a largest vertex packing.

```
def naive_maximum_vertex_packing(g):
    packing = []
    L=subsets(g.vertices())
    for S in L:
        if is_packing(g,S)==True:
            if len(S) > len(packing):
                packing = S
    return packing
```

Vertex Packing IP and LP

10. We have modeled the maximum vertex packing problem and an an *Integer Programming* (IP) problem: associate a 0-1 variable x_i to each vertex v_i, require that for each edge v_iv_j that $x_i + x_j \leq 1$, and find the maximum sum $\sum x_i$.

```
p = MixedIntegerLinearProgram(maximization=True)
x = p.new_variable(nonnegative=True)
p.set_objective(sum(x[v] for v in g.vertices()))

for (u,v) in g.edge_iterator(labels=False):
    p.add_constraint(x[u] + x[v] <= 1)

p.solve()
```

11. If we allow the variables to be nonnegative real numbers instead of integers (that is, to *relax* the IP) we get a linear program (LP). The relaxation of the vertex packing IP gives an optimum value that is necessarily an upper bound for the vertex packing number. Test this with any graph g (you can run `g=pete` to use this code for the Petersen graph).

```
p = MixedIntegerLinearProgram(maximization=True)
x = p.new_variable(integer=True, nonnegative=True)
p.set_objective(sum(x[v] for v in g.vertices()))

for (u,v) in g.edge_iterator(labels=False):
    p.add_constraint(x[u] + x[v] <= 1)

p.solve()
```

12. Use `p.get_values(x)` and check that the values correspond to a vertex packing of your graph.

13. Here is code to solve the Vertex Packing LP. Test this with any graph g (you can run `g=pete` to use this code for the Petersen graph).

```
p = MixedIntegerLinearProgram(maximization=True)
x = p.new_variable(nonnegative=True)
p.set_objective(sum(x[v] for v in g.vertices()))

for (u,v) in g.edge_iterator(labels=False):
    p.add_constraint(x[u] + x[v] <= 1)

p.solve()
```

14. Here's how we can write this in Sage in a general way (where we can easily input any graph g).

```
def vertex_packing_LP(g):

    p = MixedIntegerLinearProgram(maximization=True)
    x = p.new_variable(nonnegative=True)
    p.set_objective(sum(x[v] for v in g.vertices()))

    for v in g.vertices():
        p.add_constraint(x[v], max=1)

    for (u,v) in g.edge_iterator(labels=False):
        p.add_constraint(x[u] + x[v], max=1)

    return p.solve()
```

15. Now test this by finding the LP upper bound for the Petersen graph. Evaluate: `vertex_packing_LP(pete)`.

16. It is possible to find LP bounds for graphs where calculating the true vertex packing number is impossible. Let g be a random graph on 100 vertices. Evaluate: `g=graphs.RandomGNP(100,.5)`?

    What do you expect the upper bound to be? Now calculate.