**Last name** _____

**First name** _____

**LARSON—OPER 731—CLASSROOM WORKSHEET 18**
**Linear Programming in Sage.**

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*).

   Here's the linear program we'd like to solve: $max\{c^T x : Ax \leq b, x \geq \mathbb{O}\}$, where

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix},\ b = \begin{pmatrix} 10 \\ 11 \end{pmatrix},\ c = \begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix}$$

2. Evaluate `LP = MixedIntegerLinearProgram(maximization=True)`

   Now we will let $x$ be the name of the variable vector, and also require that the vector entries be non-negative.

3. Evaluate `x = LP.new_variable(nonnegative=True)`.

4. Now we will tell Sage what the objective function is. Evaluate: `LP2.set_objective(sum(c[i][0]*x[i] for i in range(n)))`. Notice that we are also implicitly giving the components of vector $x$ the names $x[i]$, where $i$ is a column index (starting at 0).

Now let's add our constraints.

5. Evaluate:

```
for i in range(m):
    LP2.add_constraint(sum(A[i][j]*x[j] for j in range(n))<=b[i][0])
```

6. Now evaluate `LP.solve()` to solve the linear program. What do you get?

   This should print the maximum possible value of the objective function. (And there is probably a teeny error - all LP solvers are subject to some numerical instability.) All the work was done in the last step. If the LP is big this could take some time. Now let's see a feasible solution that attains the optimal value.

7. Evaluate `LP.get_values(x)`. What do you get?

Now lets set up the dual of the last linear program. We found that the dual is: $min\{b^T y : A^T y \geq c, y \geq \mathbb{O}\}$.

We'll call this system LPdual. Here are all the steps.

8. Evaluate:

```
LPdual = MixedIntegerLinearProgram(maximization=False)
y = LPdual.new_variable(nonnegative=True)

At=A.transpose()
LPdual.set_objective((sum(b[j][0]*y[j] for j in range(m)))
for i in range(n):
    LPdual.add_constraint(sum(At[i][j]*y[j] for j in range(m))>=c[i][0])
LPdual.solve()
LPdual.get_values(y)
```

   What did you get? What does it mean?

9. Let's try this for a **large** LP. We'll need to define an appropriate $A$, $b$ and $c$. Let's try to cook up examples where we can guess what an answer should be.

**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.

If the variables are restricted to integers this is an *Integer Programming* (IP) problem. If we allow the variables to be 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.

Here's how we can write this in Sage in a general way.

```
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()
```

10. Type in the above definition and evaluate.

11. Evaluate the following to get an upper bound for the vertex packing number of the complete graph on three vertices, $k3$:

```
k3=graphs.CompleteGraph(3)
k3.show()
vertex_packing_LP(k3)
```

12. Now find the LP upper bound for the Petersen graph Evaluate:

```
g=graphs.PetersenGraph()
g.show()
vertex_packing_LP(g)
```

13. 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.