Last name ______________________

First name ______________________

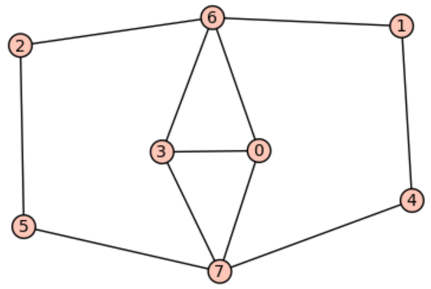
LARSON—MATH 356–SAGE HOMEWORK WORKSHEET (h07)

1. Log in to your Sage/CoCalc account.
   (a) Start the Chrome browser.
   (b) Go to `http://cocalc.com` and sign in.
   (c) You should see an existing Project for our class. Click on that.
   (d) Click “New”, call it **h07**, then click “Sage Worksheet”.
   (e) For each problem number, label it in the Sage cell where the work is. So for Problem 1, the first line of the cell should be `#Problem 1`.
   (f) When you are finished with the worksheet, click ”make pdf”, email me the pdf (at `clarson@vcu.edu`, with a header that says **Math 356 h07 worksheet attached**).

2. I’ve updated the copy of “graphs.sage” in your Handouts folder to include what we’ve added in class. *Copy* the current version from Handouts to your Home directory. *Load* your copy of “graphs.sage”. Run: `load(‘graphs.sage’)`.

   A New Graph to Study!

   You can find examples of all of these problems in the previous age worksheets. You do not need to code any new algorithms—just repeat the things we’ve already done for this new graph.



3. Use `Graph()`, and `add_edge()` to make the above graph. Call it something memorable like your name or whatever it looks like to you. Assign your graph that `name` (so you can use it later).

4. Find the `graph6_string` for this graph.

5. Use Sage to find an `incidence matrix` for this graph.

6. Use Sage to find an `adjacency matrix` for this graph.

7. Use Sage to find the `order` of this graph.

8. Use Sage to find the `size` of this graph.

9. We designed a `neighbors` function to find the neighbors of any vertex. Use that to find the neighbors of vertex 6.

10. We designed a function to test if a graph is *connected*. Use that to test if your graph is connected.

11. We designed a function to test if a graph is *bipartite*. Use that to test if your graph is bipartite.

12. Find the *complement* of your graph and *show* it (display a picture).

13. Use Sage to find the *maximum degree* of this graph.

14. Use Sage to find the *minimum degree* of this graph.

15. Use Sage to produce a list of *vertices* of this graph.

16. Use Sage to produce a list of *edges* of this graph.

17. We designed a function to test if a graph is a *tree*. Use that to test if your graph is a tree.

18. We designed a function to find a *spanning tree* of a connected graph. Use that to find a spanning tree of your graph. Call it T. And *show* it.

19. We wrote a function to find the *Prufer code* of any (integer) labeled tree. Use that to find the Prufer code of T.

20. We designed a function to test if a graph is *Eulerian*. Use that to test if your graph is Eulerian.

Dijkstra & Kruskal's Algorithm for a Weighted Graph

21. Generate and display a `random_weighted_graph` with 5 vertices. Call it R. Make sure to display the edge weights.

22. We wrote a function to find the sum of the weights of all of the edges of a weighted graph. Use that to find the sum of the weights of all of the edges of R.

23. We implemented `Dijkstra's algorithm` to find the minimum weight path between a pair of vertices. Use that to find the shortest path between vertices 0 and 3 in graph R.

24. We implemented `Kruskal's algorithm` to find a minimum weight spanning tree between a pair of vertices. Use that to find a minimum weight spanning tree of R. Call it T. And *show* it.

25. Use Sage to find the sum of the weights of all of the edges of T (that's the minimum weight of all spanning trees of R).