DSC 40B - Homework 06

Due: Monday, November 7

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope on Monday at 11:59 p.m.

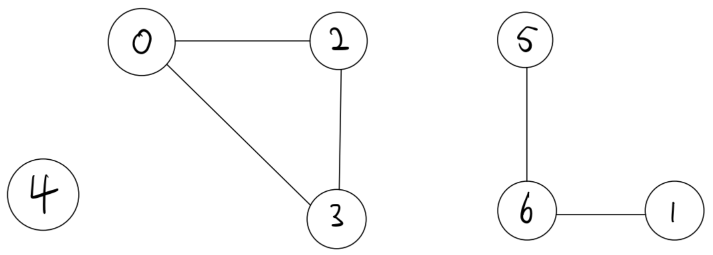
Problem 1.

In the following, let

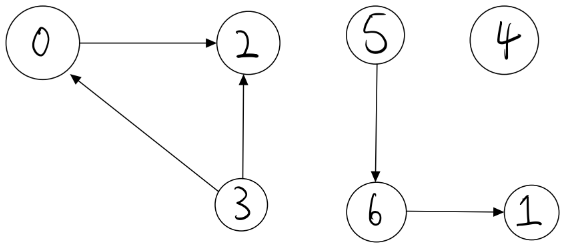
$$V = \{0, 1, 2, 3, 4, 5, 6\},$$
$$E = \{(0, 2), (3, 2), (5, 6), (6, 1), (3, 0)\}.$$

For this problem, you do not need to show your work.

a) Draw the *undirected* graph $G = (V, E)$. Remember that when writing the edges of an undirected graph, we often abuse notation and write (u, v) when we really mean $\{u, v\}$; we have done so here.



b) Draw the graph $G = (V, E)$, assuming that G is directed.



c) Write down the connected components of G, assuming that G is undirected.

$C_1 = \{0, 2, 3\}$, $C_2 = \{1, 5, 6\}$, $C_3 = \{4\}$

d) Write the adjacency matrix representation of G, assuming that G is undirected.

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

e) Write the adjacency matrix representation of G, assuming that G is directed.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Problem 2.

Suppose A is the adjacency matrix of an undirected graph. Let A^2 be the *squared* matrix, obtained by matrix multiplying A by itself. Show that the (i, j) entry of A^2 is the number of ways to get from i to j in exactly two hops (i.e., the number of paths of length two between node i and node j). *Hint:* Consult your linear algebra notes/textbook to remember a formula for the (i, j) entry of the product of two matrices.

We are going to prove this fact for all n

Theorem: Raising an adjacency matrix A to the power of n for any graph G, gives the total number of n-length walks between 2 vertices for the $v_{i^{th}}$ and $v_{j^{th}}$ vertex.

Case when n = 1 It is easy to see that when n = 1, the adjency matrix gives the number of walks of length 1 it i - j as,

$$A^1_{ij} = 1 \text{ if } \{v_i, v_j\} \in E, \ \ 0 \text{ otherwise}$$

That tells us that there is a 1 in A_{ij} if there exist one edge between the 2 vertices v_i, v_j which corresponds to a walk of length 1.

Now assuming, that A^n_{ij} gives the number of n length walks from v_i, v_j is true, we are going to prove that the theorem holds for A^{n+1}. Since we assume that it is true for A^n I set that the number of n-length walks from v_i, v_k is A^n_{ik} and that the number of walks of length 1 from v_k, v_j is equal to A^1_{kj}. Then to get all walks of n + 1 length, that it is the sum of all walks from v_i, v_k to v_k, v_j for all the possibilities of k, which yields us,

$$\sum_{k=1} A^n_{ik} A^1_{kj}$$

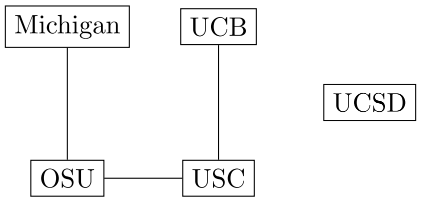
Which is equal to the matrix mutliplication of $A^n \times A^1$. Thus proving the theorem.

The following programming problem will ask you to write functions which accept a graph as input. The graph will be an instance of the `UndirectedGraph` class (or `DirectedGraph` class, if the problem says so) from the `dsc40graph` module. You can install this module on your own computer by running `pip install dsc40graph`. Alternatively, you can download the module from `https://raw.githubusercontent.com/eldridgejm/dsc40graph/master/dsc40graph.py` and place it in the same directory as your code. Or, if you'd prefer, you can simply use DataHub to write your code – we have pre-installed dsc40graph for you.

The documentation for the graph library, as well as examples of it in action, can be found at https://eldridgejm.github.io/dsc40graph/. This graph class behaves exactly like the graphs we have seen in lecture; namely it has a .neighbors() method and a .nodes attribute.

Programming Problem 1.

We can use a graph to represent rivalries between universities. Each node in the graph is a university, and an edge exists between two nodes if those two schools are rivals. For instance, the graph below represents the fact that OSU and Michigan are rivals, OSU and USC are rivals, UCB and USC are rivals, but UCSD does not have a rival.



In a file called None, write a function assign_good_and_evil(graph) which determines if it is possible to label each university as either “good” or “evil” such that every rivalry is between a “good” school and an “evil” school. The input to the function will be a graph of type UndirectedGraph from the dsc40graph package. If there is a way to label each node as “good” and “evil” so that every rivaly is between a “good” school and an “evil” school, your function should return it as a dictionary mapping each node to a string, 'good' or 'evil'; if such a labeling is not possible, your function should return None.

For example:

```
>>> example_graph = dsc40graph.UndirectedGraph()
>>> example_graph.add_edge('Michigan', 'OSU')
>>> example_graph.add_edge('USC', 'OSU')
>>> example_graph.add_edge('USC', 'UCB')
>>> example_graph.add_node('UCSD')
>>> assign_good_and_evil(example_graph)
{
    'OSU': 'good',
    'Michigan': 'evil',
    'USC': 'evil',
    'UCB': 'good',
    'UCSD': 'good'
}
```

Of course, there might be several ways to label the graph – your code need only return one labeling.