DSC 40B - Homework 06

Due: Wednesday, May 17

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 at 11:59 p.m.

Problem 1.

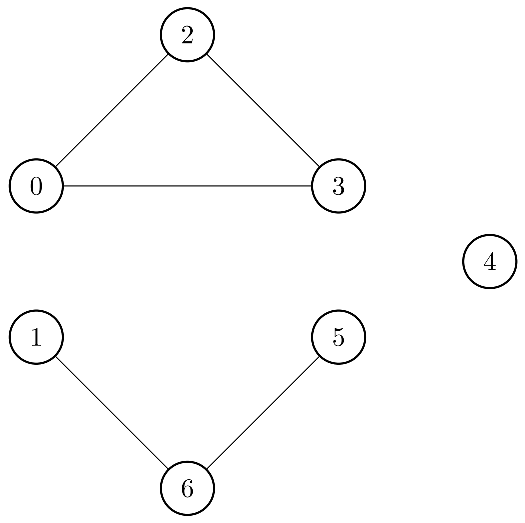
In the following, let

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

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

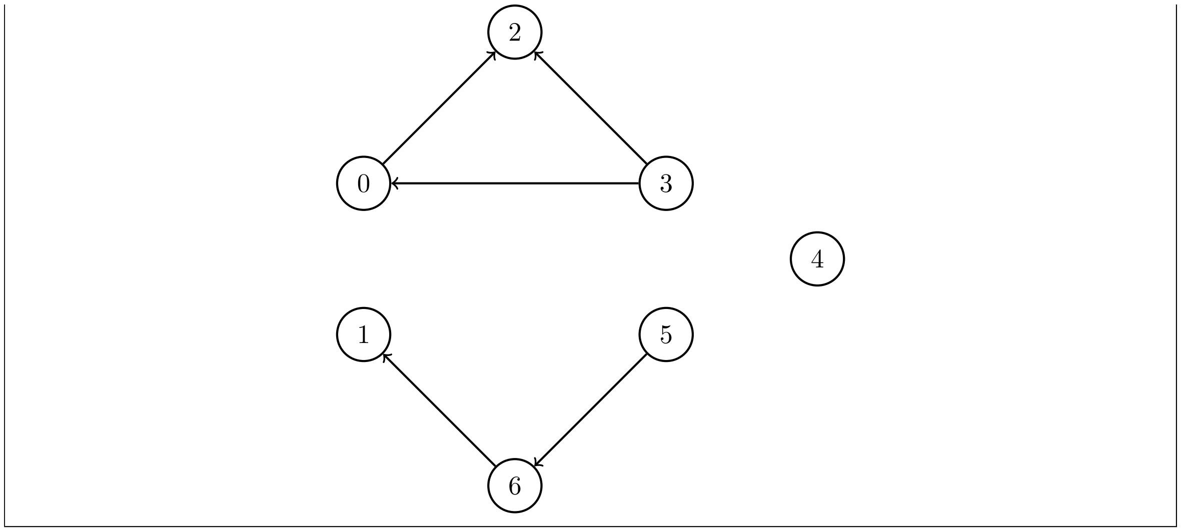
a) Draw the *undirected* graph $G = (V, E)$. Remember that when writing the edges of an undirected graph, we often abuse notation and write (u, v) when we really mean $\{u, v\}$; we have done so here.

Solution:



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

Solution:



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

Solution: From the above drawing we see that the connected components are:

$$\{0,2,3\},\{1,5,6\},\{4\}$$

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

Solution:

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

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

Solution:

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

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.

Solution: Recall that the (i, j) entry of the product of two $n \times n$ matrices X and Y is given by:

$$(XY)_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}.$$

In other words, $(XY)_{ij}$ is the dot product of the ith row of X and the jth column of Y.

Hence the (i, j) entry of A^2 is given by:

$$(A^2)_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}.$$

Now, any path from node i to node j is of the form (i, k, j), where k is any node in the graph. This will be a valid path if and only if edges (i, k) and (k, j) are in the graph. If both of these edges are in the graph, then both a_{ik} and a_{kj} are one, and so their product is too. On the other hand, if either of these edges is missing, then at least one of a_{ik} and a_{kj} are zero, and so their product is zero. Therefore:

$$a_{ik} a_{kj} = \begin{cases} 1, & \text{if there is a path of length 2 between } i \text{ and } j \text{ through } k. \\ 0, & \text{otherwise.} \end{cases}$$

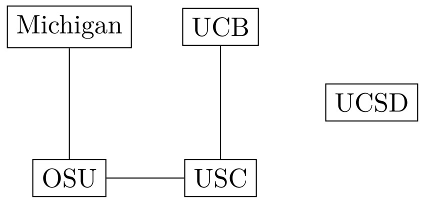
As a result, the total number of paths of length two between i and j is found by summing over all possible intermediate nodes, k:

$$\sum_{k=1}^{n} a_{ik} a_{kj}.$$

This is exactly $(A_{ij})^2$.

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 `assign_good_and_evil.py`, 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'
}
```

If in the above graph, there is also an edge between `Michigan' and `USC', then there does not exist such a label and your funtcion should return `None`.

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

Solution: This can be solved using a graph search algorithm. We start by labeling the source node as either "good" or "evil" – it doesn't matter which. We then label all of its neighbors with the opposite label. From there on, any time we visit a node and loop through the neighbors, we give each neighbor a label that is opposite the label of the node we are currently visiting.

When looking through the neighbors, however, we may come across a node which already has already been assigned a label. If that label is the opposite of label of the node we are currently visiting, then all is fine. If the two labels are the same, then we have discovered that it is not possible to label the graph in this way.

We will implement this idea by modifying the "full" BFS. We use the "full" version of the algorithm because the graph may not necessarily be connected (the example graph is not), yet we still wish to assign labels to every node. The code is shown below:

```python
from collections import deque

def assign_good_and_evil(graph):
    label = {}
    status = {node: 'undiscovered' for node in graph.nodes}
    for node in graph.nodes:
        if status[node] == 'undiscovered':
            possible = good_and_evil_bfs(graph, node, status, label)
            if not possible:
                return None

    return label

def good_and_evil_bfs(graph, source, status, label):
    status[source] = 'pending'
    label[source] = 'good'
    pending = deque([source])

    # while there are still pending nodes
    while pending:
        u = pending.popleft()
        for v in graph.neighbors(u):
            # explore edge (u,v)
            if status[v] == 'undiscovered':
                status[v] = 'pending'
                if label[u] == 'good':
                    label[v] = 'evil'
                else:
                    label[v] = 'good'
                # append to right
                pending.append(v)
            elif label[u] == label[v]:
                return False
        status[u] = 'visited'

    return True
```