DSC 40B - Homework 05

Due: Wednesday, February 18

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.

Suppose we are given two (unsorted) arrays of real valued numbers A and B, where $|A| = n$ and $|B| = m$. Suppose we wish to merge these two arranges into a single array C, but remove duplicates. We call this the MergeArray problem. For example, $A = \{3, 12, 5\}$ and $B = \{7, 12, 9, 22, 5\}$. Then the merged array C should contain the list of numbers $\{3, 12, 5, 7, 9, 22\}$ (note that numbers do not need to be in this order).

Describe an algorithm to solve this MergeArray problem (using 6 sentences or less): you can use any data structure or procedures we have descried in class. Provide the time complexity analysis of your answer. Slower algorithms receive fewer points.

Solution: First, initiate an empty array C and hash table D. Then, iterate through each element of A and B. In each iteration, first check to see if the element has already been stored in D. If not, then add it to hash table D and also array C. If the element already exists in D, then do nothing. At the end of this process we will have created array C merging the elements of A and B with no duplicates, in $\Theta(m + n)$ time.

Problem 2.

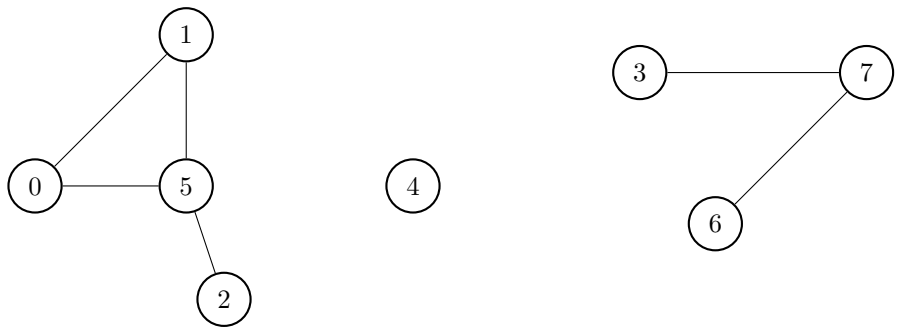
In the following, let

$$V = \{0, 1, 2, 3, 4, 5, 6, 7\},$$
$$E = \{(0, 1), (1, 5), (5, 2), (0, 5), (7, 3), (6, 7)\}.$$

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

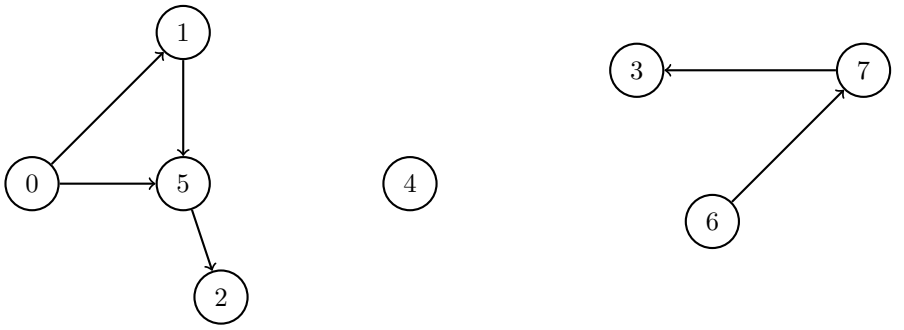
a) Draw the *undirected* graph $G = (V, E)$. Remember that when writing the edges of an undirected graph, we often abuse notation and write (u, v) when we really mean $\{u, v\}$; we have done so here.

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:

$$\{0, 1, 2, 5\}, \{3, 6, 7\}, \{4\}$$

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

Solution:

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

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

Solution:

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

Problem 3.

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 $(A_{ij})^2$.

Problem 4.

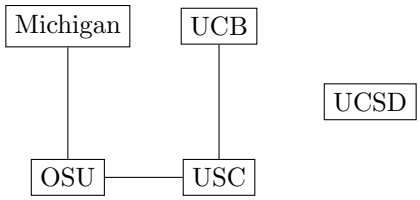
Suppose we wish to develop an algorithm HasCompOfSize(G, k), which, given a undirected graph $G = (V, E)$ and an integer k, we wish to return whether there exists a connected component of G whose size is **exactly** k. Here, the size of a graph is simply the number of nodes in this graph (and note that a connected component is itself a graph, which is subgraph of G).

Describe an efficient algorithm for this test HasCompOfSize(G, k) (in 6 sentences or less): you can refer to / use any procedures described in class. Give the time complexity of your algorithm; slower algorithms receive fewer points.

Solution: Pick any node to start BFS from, keeping track of the number of visited nodes along the way. The BFS will terminate once the component has been completely searched. If the number of visited nodes is k, then return true, otherwise continue this procedure with any other unvisited node until all components have been searched. If at the end there are no components with k visited nodes, then return false. This has the same complexity as BFS which is $\Theta(|V| + |E|)$.

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.

You can install `dsc40graph` by following one of the methods mentioned on lecture 10, slide 44.

Solution:

```
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
```