DSC 40B - Homework 06

Due: Wednesday, February 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 on Wednesday at 11:59 p.m.

Problem 1. *(Eligible for Redemption)*

In each of the following situations, describe whether you would use a directed or undirected graph to represent the data, and why.

Note: in some cases *both* directed and undirected graphs might be reasonable choices; in these cases your reasoning will determine whether your answer is correct or incorrect.

a) A graph of webpages, where edges represent links.

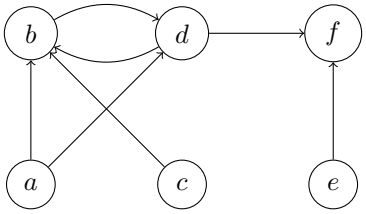
b) A contact tracing graph, where nodes are people with COVID and edges represent the spread of the disease from one person to another.

c) A computer network graphs, where edges represent a connection between two computers.

d) A prerequisite graph, where nodes are classes at UCSD and an edge shows that one class is a prerequisite of the other.

Problem 2. *(Eligible for Redemption)*

Consider this graph:



The questions below will ask you about the properties of this graph. You do not need to show work for this problem.

a) List four different paths from node a to node f. Hint: remember that paths do not need to be *simple paths*, that is, a path may visit the same node twice.

Solution: Four possibilities are:

$$(a, b, d, f),$$
$$(a, d, f),$$
$$(a, d, b, d, f),$$
$$(a, d, b, d, b, d, f).$$

The third path visits d twice – that's totally fine. The last path visits the edges (b, d) and (d, b) twice – that's also fine.

These aren't the only four paths, since we can come up with infinitely many paths from a to f by bouncing back and forth between b and d. For instance, you could've tried to annoy the grader by writing:

$$(a, b, d, b, d, b, d, b, d, b, d, b, d, b, d, b, d, b, d, f)$$

That's a valid path.

b) Recall that a *simple* path is one which has no duplicate nodes; that is, it can only visit a node once. List all of the simple paths from a to f.

Solution: The only simple paths from a to f are:

$$(a, b, d, f),$$
$$(a, d, f).$$

The path (a, d, b, d, f) is a valid path, but it is not a *simple* path since it visits the node d twice.

c) List all of the nodes which are reachable from a. Hint: there are four.

Solution: That is, list all of the nodes which are at the end of a path which starts at a. These are:

b, via, e.g., path (a, b),
d, via, e.g., path (a, d),
f, via, e.g., path (a, d, f),
a, via, e.g., path (a).

The last one might have been tricky, but remember that every node is reachable from itself.

d) What is the total degree of node b?

Solution: There are 3 edges coming into b, and one edge leaving, for a total degree of 4.

Problem 3. *(Eligible for Redemption)*

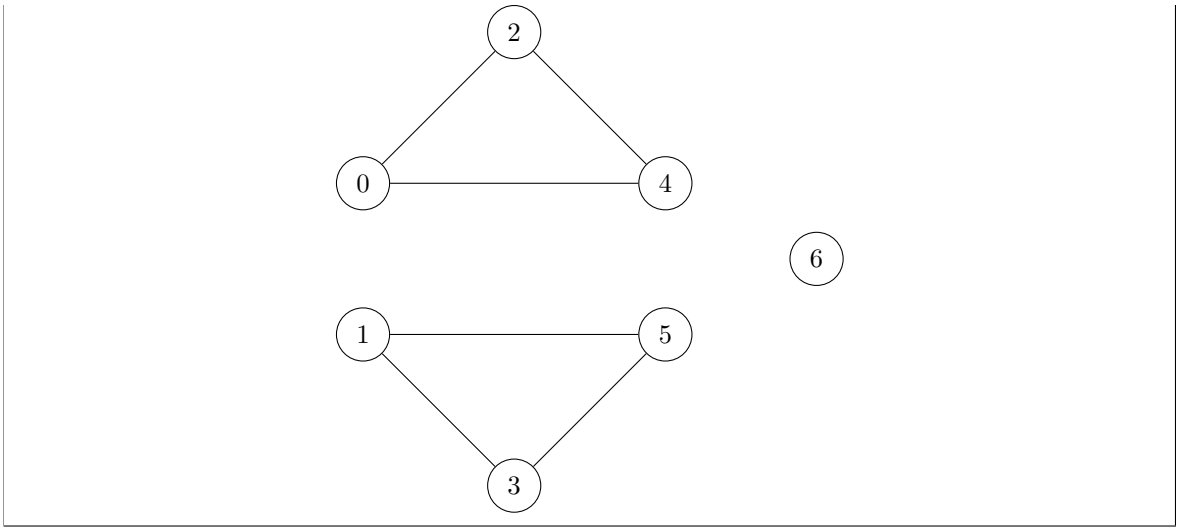
In the following, let

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

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.

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:

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

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

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

Problem 4.

Suppose A is the adjacency matrix of an undirected graph. Let A^2 be the *squared* matrix, obtained by matrix multiplying A by itself. Prove 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 $(A_{ij})^2$.

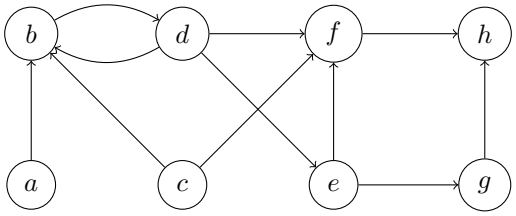


Figure 1: Graph for Problems 1 and 2.

Problem 5. *(Eligible for Redemption)*

Consider a *breadth*-first search on the graph shown in Figure 1, starting with node c.

Which nodes are visited, and in what order? Use the convention that `graph.neighbors()` produces successors in ascending order of label.

Solution: Nodes c, b, f, d, h, e, g are visited, in that order.

Problem 6.

Consider the following modified BFS which has two new lines of code: lines 21 and 22. What is the asymptotic time complexity of `foo` in terms of $|V|$ and $|E|$?

```
1  from collections import deque
2
3  def foo(graph):
4      status = {'undiscovered' for node in graph.nodes}
5      for u in graph.nodes:
6          if status[u] == 'undiscovered':
7              bar(graph, u, status)
8
9  def bar(graph, source, status):
10     status[source] = 'pending'
11     pending = deque([source])
12
13     # while there are still pending nodes
14     while pending:
15         u = pending.popleft()
16         for v in graph.neighbors(u):
17             # explore edge (u,v)
```

```
18              if status[v] == 'undiscovered':
19                  status[v] = 'pending'
20                  pending.append(v)
21              for v in graph.nodes:
22                  print(v)
23          status[u] = 'visited'
24
25      return predecessor, distance
```

Solution: The new lines of code take $\Theta(V)$ time every time that an edge is visited. This means that the total time complexity has an extra $\Theta(VE)$. So the time is $\Theta(V + E + VE)$. This simplifies to $\Theta(V + VE)$, since $|E| \leq |V| \cdot |E|$.

Note that it isn't quite right to say that $|V| \cdot |E|$ is larger than $|V|$ and thus the time complexity is $\Theta(VE)$. For instance, suppose $|E| = 0$. Then $|V| \cdot |E|$ is zero, and therefore not larger than $|V|$!.