DSC 40B - Quiz 06

Name: | Student ID:

About the quizzes:

- Quizzes in DSC 40B are *optional* and graded pass/fail.
- A score of 70% or higher earns a "pass" and 1.5 credits toward your final grade.
- If you don't pass, no credits are earned, but it doesn't hurt your grade.
- You have 30 minutes to complete the quiz.
- Unfortunately, we can't answer clarifying questions during the quiz. If you think a question has a bug or is unclear, please let us know in a private post on Campuswire after the quiz, and we'll take it into account when grading.

Problem 1.

An **undirected** graph has 7 nodes. What is the greatest number of edges it can have?

◯ 7 ● 21 ◯ 42 ◯ 49

Solution: 21.

An undirected graph can't have self-loops, and (u, v) and (v, u) are the same edge. So the most edges it can have is one for each pair of distinct nodes, which is $|V|(|V|-1)/2 = 7 \cdot 6/2 = 21$. (A *directed* graph with 7 nodes could have up to $7^2 = 49$ edges, since there it can have self-loops and opposite edges.)

This problem was similar to Problem #201 in the Quiz 06 practice problems.

Problem 2.

Let $V = \{a, b, c, d\}$ and $E = \{(a, b), (a, a), (a, c), (d, b), (c, a)\}$ be the nodes and edges of a **directed** graph. How many predecessors does node b have?

◯ 1 ● 2 ◯ 3 ◯ 4

Solution: 2.

The predecessors of b are the nodes at the start of an edge entering b. The edges entering b are (a, b) and (d, b), so its predecessors are a and d.

This problem was Problem #192 from the Quiz 06 practice problems.

Problem 3.

Consider the **undirected** graph $G = (V, E)$ with

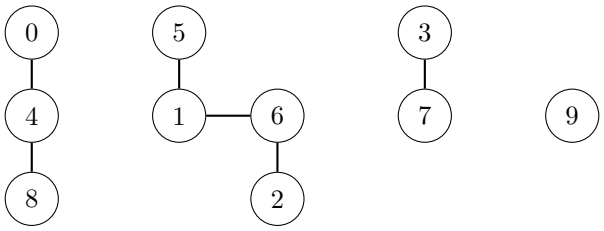
$$V = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$
$$E = \{(0, 4), (4, 8), (1, 5), (2, 6), (6, 1), (3, 7)\}$$

How many connected components does G have?

◯ 3 ● 4 ◯ 5 ◯ 6

Solution: 4.

Drawing the graph makes the components easy to see:



The connected components are $\{0, 4, 8\}$, $\{1, 2, 5, 6\}$, $\{3, 7\}$, and $\{9\}$. Node 9 has no edges, but it is reachable from itself, so it is a connected component by itself.

This problem was similar to Problem #197 in the Quiz 06 practice problems.

Problem 4.

Suppose an **undirected** graph $G = (V, E)$ has two connected components, C_1 and C_2, where C_1 has 5 nodes and C_2 has 3 nodes. What is the largest that $|E|$ can possibly be?

◯ 8 ● 13 ◯ 16 ◯ 28

Solution: 13.

There can't be any edges between C_1 and C_2, or they would be one connected component. So the most edges we can have is to put every possible edge inside each component. A component with 5 nodes can have at most $5 \cdot 4/2 = 10$ edges, and one with 3 nodes can have at most $3 \cdot 2/2 = 3$, for a total of $10 + 3 = 13$.

This problem was similar to Problem #207 in the Quiz 06 practice problems.

Problem 5.

Let $G = (V, E)$ be an undirected graph, and suppose that u, v, and w are three nodes in V. Suppose that u and v are in the same connected component, but u and w are not in the same connected component. True or False: it is possible that v and w are in the same connected component.

◯ True

● False

Solution: False.

If v and w were in the same connected component, then there would be a path from u to v and a path from v to w, and putting them together gives a path from u to w. That would put u and w in the same component, which contradicts what we were told.

In other words, connected components are **disjoint**: no node belongs to more than one of them.

This problem was Problem #200 from the Quiz 06 practice problems.

Problem 6.

An *isolated* node in an undirected graph is a node that has no neighbors.

a) Suppose a graph $G = (V, E)$ is stored as an **adjacency matrix**. What is the best possible worst-case time complexity of an efficient algorithm that determines whether a **given node** v is isolated?

◯ $\Theta(1)$ ● $\Theta(V)$ ◯ $\Theta(V + E)$ ◯ $\Theta(V^2)$

Solution: $\Theta(V)$.

We only need to look at the one node v, not the whole graph. With an adjacency matrix, v is isolated exactly when every entry in its row is zero. In the worst case (for example, when v really is isolated), we have to check all $|V|$ entries in the row, which takes $\Theta(V)$ time.

b) Now suppose G is stored as an **adjacency list**. What is the best possible worst-case time complexity of an efficient algorithm that determines whether a **given node** v is isolated?

● $\Theta(1)$ ◯ $\Theta(V)$ ◯ $\Theta(V + E)$ ◯ $\Theta(V^2)$

Solution: $\Theta(1)$.

With an adjacency list, v is isolated exactly when its list of neighbors is empty. Checking whether a list is empty (or finding its length) takes $\Theta(1)$ time.

This problem was similar to Problem #223 in the Quiz 06 practice problems.

Problem 7.

Suppose a **connected**, **undirected** graph $G = (V, E)$ is stored as an adjacency list in the variable `adj`. What is the time complexity of the code below?

```
total = 0
seen = {u: False for u in range(len(adj))}
for u in range(len(adj)):
    for v in adj[u]:
        total += 1
        if not seen[v]:
            seen[v] = True
            for w in range(len(adj)):
                total += 1
    print(u, total)
```

◯ $\Theta(V)$ ◯ $\Theta(V + E)$ ● $\Theta(V^2)$ ◯ $\Theta(V \cdot E)$

Solution: $\Theta(V^2)$.

This is an aggregate analysis, as in lecture. Creating `seen` takes $\Theta(V)$ time, and the outer loop runs $|V|$ times. Over all of the iterations of the outer loop, the loop over `adj[u]` runs once for each entry in the adjacency list, which is $2|E|$ times, since the graph is undirected. So far, that's $\Theta(V + E)$.

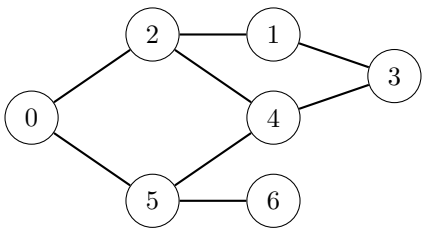
The innermost loop over all of the nodes takes $\Theta(V)$ time, but it only runs when `seen[v]` is `False`. Once it runs for a node v, `seen[v]` becomes `True`, so it runs at most once per node. And since the graph is connected, every node has a neighbor (if $|V| > 1$), so every node shows up in some adjacency list and the innermost loop runs exactly once per node: $|V|$ times. In total, it takes $|V| \cdot \Theta(V) = \Theta(V^2)$ time.

It isn't $\Theta(V \cdot E)$: although the innermost loop is inside the loop over the $2|E|$ entries, it runs on only $|V|$ of them. The total is $\Theta(V + E + V^2) = \Theta(V^2)$, since $|E|$ is at most $|V|(|V| - 1)/2$.

This problem was similar to Problem #221 in the Quiz 06 practice problems.

Problem 8.

Suppose a breadth-first search is run on the undirected graph below, starting at node 0, and using the convention that a node's neighbors are produced in ascending order by label. In which order are the nodes visited?



◯ 0, 1, 2, 3, 4, 5, 6 ◯ 0, 2, 1, 3, 4, 5, 6 ● 0, 2, 5, 1, 4, 6, 3 ◯ 0, 5, 2, 6, 4, 1, 3

Solution: 0, 2, 5, 1, 4, 6, 3.

BFS always visits the oldest pending node. Visiting 0 makes 2 and 5 pending. Visiting 2 makes 1 and 4 pending (0 is already discovered). Visiting 5 makes 6 pending (4 is already pending). Visiting 1 makes 3 pending. Visiting 4, 6, and 3 discovers nothing new. So the queue hands out the nodes in the order 0, 2, 5, 1, 4, 6, 3.

This problem was similar to Problem #228 in the Quiz 06 practice problems.

Problem 9.

The adjacency matrix of a **directed** graph with nodes $0, 1, \ldots, 5$ is shown below. As in lecture, the entry in row i and column j is 1 if $(i, j) \in E$ and 0 otherwise.

	0	1	2	3	4	5
0	0	**1**	0	0	0	0
1	0	0	**1**	0	0	0
2	**1**	0	0	**1**	0	0
3	0	**1**	0	0	0	0
4	0	0	0	**1**	0	**1**
5	**1**	0	0	0	0	0

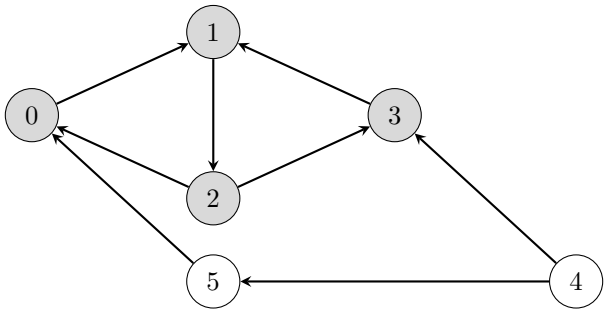
True or False: there is a path from node 0 to node 5.

○ True

● False

Solution: False.

The easiest way to see this is to draw the graph. Row i lists the edges leaving node i, so the graph is the one shown below. The nodes reachable from 0 are shaded.



Looking at this graph, we can see that there is no path from 0 to 5.

There *is* a path from 5 to 0, since $(5, 0)$ is an edge. But the graph is directed, so that doesn't give a path from 0 to 5.

This problem was similar to Problem #226 in the Quiz 06 practice problems.

Problem 10.

The code below is the full BFS from lecture with a new `print` statement, marked with a comment.

```
def bfs(graph, source, status=None):
    if status is None:
        status = {node: 'undiscovered' for node in graph.nodes}
    status[source] = 'pending'
    pending = deque([source])

    while pending:
        u = pending.popleft()
        for v in graph.neighbors(u):
            if status[v] == 'undiscovered':
                status[v] = 'pending'
                pending.append(v)
        status[u] = 'visited'

def full_bfs(graph):
    status = {node: 'undiscovered' for node in graph.nodes}
    for node in graph.nodes:
        if status[node] == 'undiscovered':
            print("bar") # <--- this is new
            bfs(graph, node, status)
```

Suppose `full_bfs` is run on an **undirected** graph and prints `"bar"` 4 times. Which of the following must be true about the graph?

- ● It has exactly 4 connected components.
- ○ It has exactly 5 connected components.
- ○ It has exactly 4 nodes.
- ○ It has exactly 4 edges.

Solution: It has exactly 4 connected components.

`"bar"` is printed each time `full_bfs` calls `bfs`. In an undirected graph, each call discovers exactly the nodes in one connected component, so if there are k connected components, `bfs` is called exactly k times. So there are 4 connected components. The printouts don't tell us how many nodes or edges the graph has, since each component can be any size.

This problem was similar to Problem #240 in the Quiz 06 practice problems.