DSC 40B - Quiz 07

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 n nodes, where n is even, and every node has exactly $n/2$ neighbors. What is the time complexity of a full BFS of this graph, as a function of n?

◯ $\Theta(n)$ ◯ $\Theta(n \log n)$ ● $\Theta(n^2)$ ◯ $\Theta(n^3)$

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

A full BFS takes $\Theta(V + E)$ time. Here $|V| = n$, and since each of the n nodes has $n/2$ neighbors, $|E| = \Theta(n^2)$. So the time is $\Theta(n + n^2) = \Theta(n^2)$.

This problem was similar to Problem #258 in the Quiz 07 practice problems.

Problem 2.

Suppose a *breadth* first search (BFS) is run on a graph $G = (V, E)$, and that node u is distance 5 from the source while node v is distance 3 from the source. True or False: it is possible that at some point during the BFS, both u and v are in the pending queue at the same time.

◯ True

● False

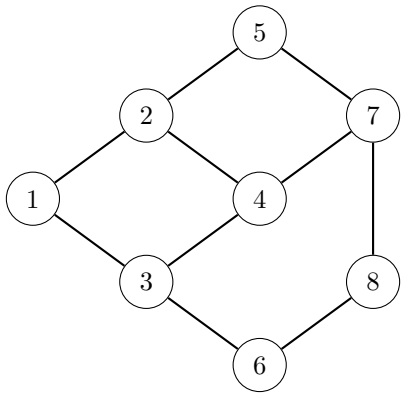
Solution: False.

BFS adds every node at distance $k - 1$ to the queue before any node at distance k, and a node at distance k is only added while a node at distance $k-1$ is being visited. So at any moment, the nodes in the queue are all at distance d or $d+1$ from the source, for some d. By the time u (distance 5) is added, every node at distance 3, including v, has already been popped. Two nodes whose distances differ by 2 can never be in the queue together.

This problem was Problem #252 from the Quiz 07 practice problems.

Problem 3.

Suppose a BFS is run on the undirected graph below using node 1 as the source, and using the convention that a node's neighbors are produced in ascending order by label. Which node will be the BFS predecessor of node 7?



◯ 3 ● 4 ◯ 5 ◯ 8

Solution: 4.

BFS visits the oldest pending node. Visiting 1 discovers 2 and 3. Visiting 2 discovers 4 and 5 (1 is already discovered), and visiting 3 discovers 6 (4 is already pending). Next, 4 is popped, before 5, since it was added to the queue first. Visiting 4 discovers 7, so 4 is the BFS predecessor of 7. By the time 5 is visited, 7 is already pending, so 5 doesn't become its predecessor. Node 3 isn't a neighbor of 7, and 8 is discovered later, from 6.

This problem was similar to Problem #246 in the Quiz 07 practice problems.

Problem 4.

`bfs_shortest_paths` (from lecture) is run on an **undirected** graph with source node s, and it returns the dictionary of BFS predecessors below. You aren't shown the graph.

```
predecessor = {
    's': None, 'a': 's', 'b': 's', 'c': 'a',
    'd': 'c', 'e': 'b', 'f': 'd', 'g': None
}
```

What is the shortest path distance from s to f?

◯ 1 ◯ 3 ● 4 ◯ 6

Solution: 4.

The BFS predecessors encode the shortest paths: following them back from f traces a shortest path from s to f, in reverse. The predecessor of f is d, whose predecessor is c, whose predecessor is a, whose predecessor is s. So the path s, a, c, d, f is a shortest path from s to f, and it has 4 edges.

This problem was similar to Problem #245 in the Quiz 07 practice problems.

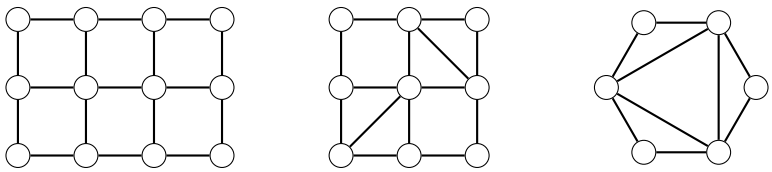
Problem 5.

The code below is `full_dfs` and `dfs` from lecture, with a `print` added at the top of `dfs`.

```
def full_dfs(graph):
    status = {node: 'undiscovered' for node in graph.nodes}
    for node in graph.nodes:
        if status[node] == 'undiscovered':
            dfs(graph, node, status)

def dfs(graph, u, status=None):
    print("Hello!")
    if status is None:
        status = {node: 'undiscovered' for node in graph.nodes}
    status[u] = 'pending'
    for v in graph.neighbors(u):
        if status[v] == 'undiscovered':
            dfs(graph, v, status)
    status[u] = 'visited'
```

Suppose `full_dfs` is run on the **undirected** graph below, which has three connected components. How many times will `"Hello!"` be printed?



○ 3 ● 27 ○ 40 ○ 80

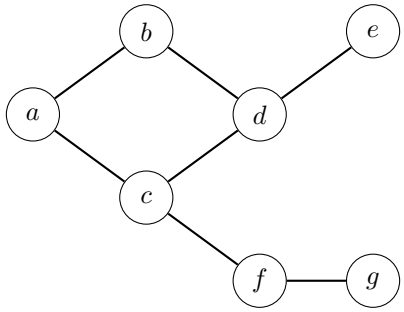
Solution: 27.

`"Hello!"` is printed once per call to `dfs`, and in a full DFS, `dfs` is called on each node exactly once: a call is only made on an undiscovered node, and the call marks it pending right away, so it is never called on again. The graph has 27 nodes, so there are 27 calls. The number of edges (40) and of connected components (3) doesn't change this. (`full_dfs` makes 3 of the calls, one for each connected component, and the other 24 are recursive calls.)

This problem was similar to Problem #281 in the Quiz 07 practice problems.

Problem 6.

Suppose a DFS is run on the undirected graph below to compute start and finish times, using node a as the source. Use the convention that the start time of a is 1, and that a node's neighbors are produced in alphabetical order.



a) What will be the **start** time of node c?

○ 2 ○ 3 ● 4 ○ 10

b) What will be the **finish** time of node c?

○ 7 ● 9 ○ 12 ○ 13

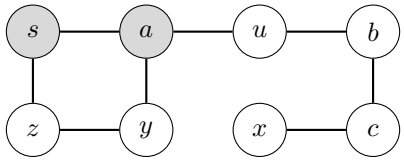
Solution: Start time: 4. Finish time: 9.

The clock goes up by one each time a node is marked pending or visited. `dfs(a)` starts at 1 and calls `dfs(b)`, which starts at 2 and calls `dfs(d)`, which starts at 3. The first undiscovered neighbor of d is c, so `dfs(c)` starts at 4. It calls `dfs(f)` (start 5), which calls `dfs(g)` (start 6, finish 7). Then f finishes at 8, and c, with no more undiscovered neighbors, finishes at 9. (After that, d calls `dfs(e)`, which starts at 10 and finishes at 11, and d, b, and a finish at 12, 13, and 14.)

This problem was similar to Problem #280 in the Quiz 07 practice problems.

Problem 7.

A DFS is run on the **undirected** graph below with s as the source, and neighbors are produced in alphabetical order. At the moment `dfs(u)` is called, s and a (shaded) are pending, and every other node is undiscovered.



How many calls to `dfs` will start *and* finish during the call to `dfs(u)`? Don't count `dfs(u)` itself.

○ 1 ● 3 ○ 5 ○ 7

Solution: 3: the calls `dfs(b)`, `dfs(c)`, and `dfs(x)`.

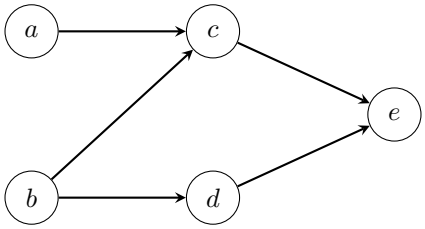
By the key property of DFS, if there is a path of undiscovered nodes from u to v when `dfs(u)` is called, then `dfs(v)` starts and finishes during `dfs(u)`. The path u, b, c, x is made of undiscovered nodes, so `dfs(b)`, `dfs(c)`, and `dfs(x)` are all nested in `dfs(u)`, even though only b is a neighbor of u.

Nodes y and z are undiscovered and reachable from u, but every path from u to them goes through the pending node a (or s), and `dfs` never calls itself on a pending node. So `dfs(u)` can't reach them: y is discovered later, after `dfs(u)` finishes and the loop in `dfs(a)` gets to it. And `dfs(s)` and `dfs(a)` started before `dfs(u)`, so they can't be nested inside it.

This problem was similar to Problem #294 in the Quiz 07 practice problems.

Problem 8.

Which one of the orderings below is a valid topological sort of this DAG?



○ a, c, e, b, d ○ b, a, c, e, d ● a, b, d, c, e ○ a, b, d, e, c

Solution: a, b, d, c, e.

In a topological sort, for every edge (u, v), u comes before v. The edges are (a, c), (b, c), (b, d), (c, e), and (d, e), and a, b, d, c, e puts the first node of each one before the second. Each of the other orderings breaks exactly one edge: a, c, e, b, d puts c before b, breaking (b, c); b, a, c, e, d puts e before d, breaking (d, e); and a, b, d, e, c puts e before c, breaking (c, e).

This problem was similar to Problem #270 in the Quiz 07 practice problems.

Problem 9.

BFS finds shortest paths. Does DFS find longest paths?

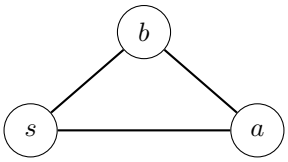
Let G be a connected, **undirected** graph, and suppose a DFS is run from a node s, recording each node's *predecessor*: the node it was discovered from. True or False: for every node v, following the DFS predecessors back from v to s gives a **longest** simple path from s to v.

○ True

● False

Solution: False.

DFS aims for depth over breadth, but that doesn't make its paths the longest ones. For example, consider the triangle below, and run a DFS from s with neighbors produced in alphabetical order.

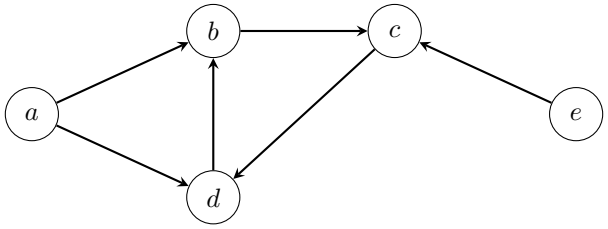


The DFS discovers a from s, then b from a. So the predecessor path to a is just s, a, with 1 edge, but s, b, a is a longer path from s to a, with 2 edges.

This problem was similar to Problem #296 in the Quiz 07 practice problems.

Problem 10.

A full DFS is run on the **directed** graph below. It loops over the nodes in alphabetical order, and neighbors are also produced in alphabetical order. Which one of these edges is found to be a **back edge**?



◯ (a, d) ◯ (b, c) ● (d, b) ◯ (e, c)

Solution: (d, b).

An edge (u, v) is a back edge if, while visiting u, we see that v is *pending*. The full DFS starts with `dfs(a)`, which calls `dfs(b)`, which calls `dfs(c)`, which calls `dfs(d)`. At that point a, b, c, and d are all pending, so when d explores the edge (d, b), b is pending: a back edge. (It closes the cycle $b \to c \to d \to b$.)

The others aren't back edges. (b, c) is explored while c is undiscovered, so it leads to the call `dfs(c)`. (a, d) is explored after `dfs(b)` returns, when d is already visited. And (e, c) is explored in the second search of the full DFS, from e, when c is visited.

No practice problem was about back edges. This problem was closest to Problem #287 in the Quiz 07 practice problems, which is about a node's neighbors being pending.