
DSC 40B - Homework 06

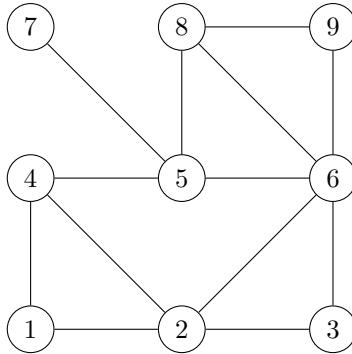
Due: Wednesday, February 25

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.

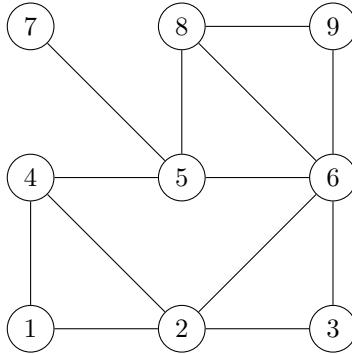
Problem 1.

For the following problems, recall that (u, v) is a *tree edge* if node v is discovered while visiting node u during a breadth-first or depth-first search. Assume the convention that a node's neighbors are produced in ascending order by label. You do not need to show your work for this problem.

a) Suppose a breadth-first search is performed on the graph below, starting at node 5. Mark every BFS tree edge with a bold arrow emanating from the predecessor.



b) Suppose a depth-first search is performed on the graph below, starting at node 5. Mark every DFS tree edge with a bold arrow emanating from the predecessor.



c) Fill in the table below so that it contains the start and finish times of each node after a DFS is performed on the above graph using node 5 as the source. Begin your start times with 5.

Node	Start	Finish
1		
2		
3		
4		
5		
6		
7		
8		
9		

Problem 2.

a) (True or False): Suppose you run the BFS algorithm as described in class from source node s . Suppose at some point during the algorithm, the status of u is 'undiscovered'. Then the status of v must be 'undiscovered'. [You don't need to justify your answer.]

b) (Fill in the blanks) Suppose the shortest path distance from s to u is 10. Then the smallest possible shortest path distance from s to v is _____

c) (True or False): Suppose you run the DFS algorithm as described in class from source node s . Suppose at some point during the algorithm, the status of v is 'pending'. Then it is possible that the status of u is 'visited'. [You don't need to justify your answer.]

Problem 3.

Consider the following code (which does not do anything meaningful) that takes an **undirected and connected** graph $G = (V, E)$ as input. Assume that the graph is represented by the adjacency list representation.

Here 'dfs' within the code is the same DFS procedure that we described in class (e.g, on slide 6 of Lecture 13).

Analyze its time complexity in terms of $|V|$ and $|E|$ using asymptotic language. Make your answer as tight as possible, and loose bound receives fewer points. [You don't need to justify your answer.]

```

def SomeGraphFunction(G):
    for u in G.nodes:
        for v in G.neighbors(u):
            status = {node: 'undiscovered' for node in G.nodes}
            dfs(G, v, status)
            print("Hello!")

```

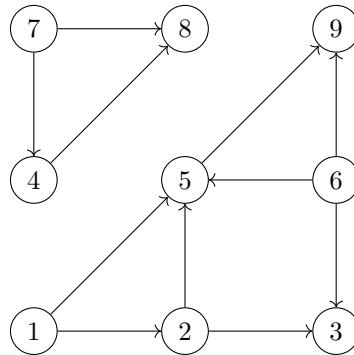
Problem 4.

Give an example of a DAG $G = (V, E)$ of $|V| = n$ nodes, yet with $|E| = \Theta(n^2)$ edges.

Problem 5.

Topologically sort the vertices of the following graph. Note that there may be multiple, equally-correct topological sorts.

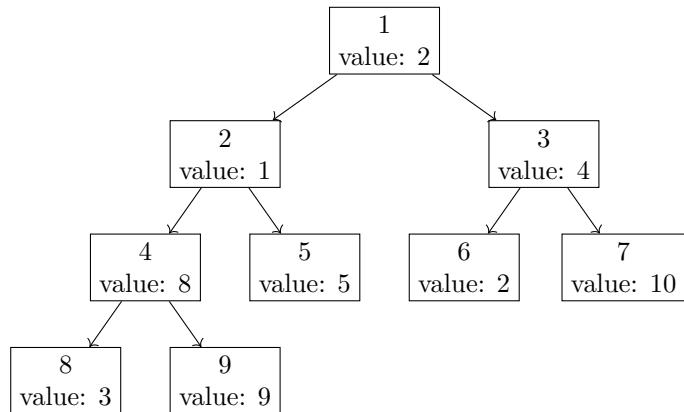
You do not need to show your work for this problem.



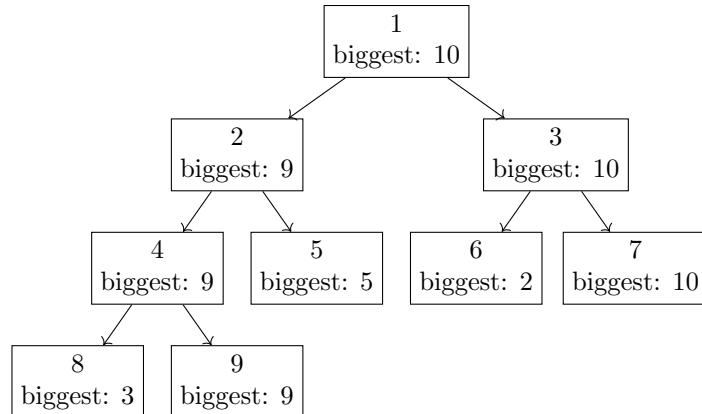
Programming Problem 1.

You are given a directed graph representing a tree and a dictionary `value` which contains a value for each node. Define the *biggest descendant value* of a node u to be the largest value of any node which is a descendent of u in the tree (for this problem, you should consider u to be a descendent of itself).

For instance, given the following tree where each node's label is replaced by its value:



The *biggest descendant value* for each node is:



In a file named `biggest_descendent.py`, write a function `biggest_descendent(graph, root, value)` which accepts the graph, the label of the root node, and the dictionary of values and returns a dictionary mapping each node in the graph to its biggest descendant value.

The input graph will be an instance of `dsc40graph.DirectedGraph()`.

Example:

```

>>> edges = [(1, 2), (1, 3), (2, 4), (2, 5), (4, 8), (4, 9), (3, 6), (3, 7)]
>>> g = dsc40graph.DirectedGraph()
>>> for edge in edges: g.add_edge(*edge)
>>> value = {1: 2, 2: 1, 3: 4, 4: 8, 5: 5, 6: 2, 7: 10, 8: 3, 9: 9}
>>> biggest_descendent(g, 1, value)
{1: 10, 2: 9, 3: 10, 4: 9, 5: 5, 6: 2, 7: 10, 8: 3, 9: 9}
  
```