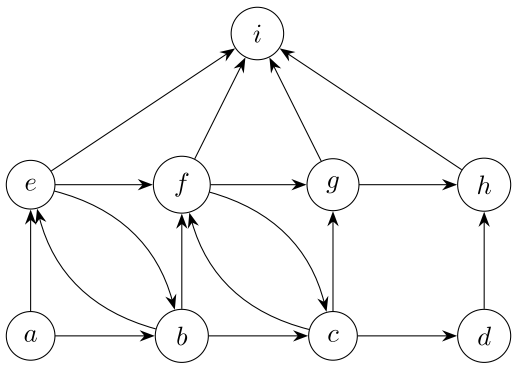
DSC 40B - Discussion 06

Problem 1.

In lecture, we saw that breadth-first search (BFS) can be used to find the shortest paths from a source node to all other nodes in an unweighted graph. Our implementation of BFS returned two dictionaries: `distance`, containing the shortest path distance from the source node to each node in the graph, and `predecessor`, containing the BFS predecessor of each node.

Consider a breadth-first search on the graph shown below, starting with node a.



$$\begin{tikzcd}
 & & & i & & & \\
e \arrow[urrr] \arrow[rr] \arrow[drr, bend left] & & f \arrow[ur] \arrow[rr] \arrow[drr, bend left] & & g \arrow[ul] \arrow[rr] & & h \arrow[ulll] \\
a \arrow[u] \arrow[rr] & & b \arrow[ull, bend left] \arrow[u] \arrow[rr] & & c \arrow[ull, bend left] \arrow[u] \arrow[rr] & & d \arrow[u]
\end{tikzcd}$$

Write down the `distance` and `predecessor` dictionaries returned by BFS on this graph.

Note: you should adopt the convention that `.neighbors()` returns the neighbors of a node in **ascending alphabetical order**.

Solution:

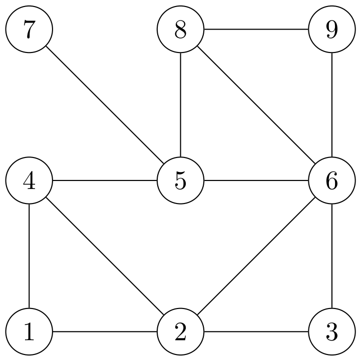
```
distance = {
    'a': 0,
    'b': 1,
    'c': 2,
    'd': 3,
    'e': 1,
    'f': 2,
    'g': 3,
    'h': 4,
    'i': 2
}

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

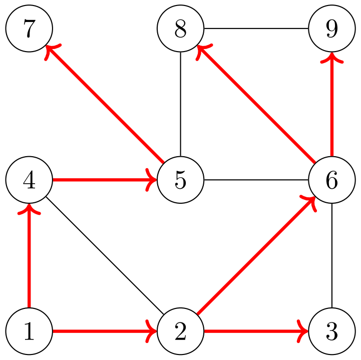
Problem 2.

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.

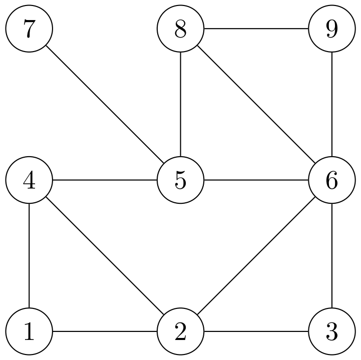
a) Suppose a breadth-first search is performed on the graph below, starting at node 1. Mark every BFS tree edge with a bold arrow emanating from the predecessor.



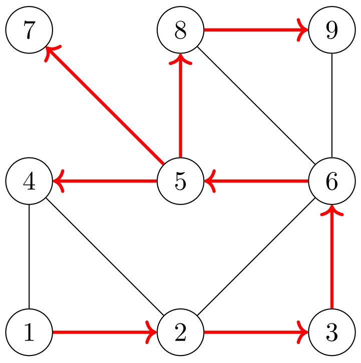
Solution:



b) Suppose a depth-first search is performed on the graph below, starting at node 1. Mark every DFS tree edge with a bold arrow emanating from the predecessor.



Solution:



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 1 as the source. Begin your start times with 1.

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

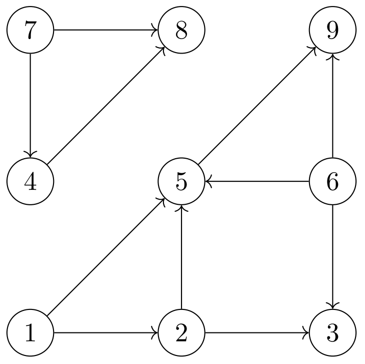
Solution:

Node	Start	Finish
1	1	18
2	2	17
3	3	16
4	6	7
5	5	14
6	4	15
7	8	9
8	10	13
9	11	12

Problem 3.

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.



Solution: Our algorithm for producing a topological sort of the nodes is to first perform a DFS in order to record finish times, and then to sort the nodes in order of decreasing finish time.

If we start a DFS at node 1 and us the convention that the neighbors are produced in increasing order of label, we will visit nodes 1, 2, 3, 5, and 9. This search didn't visit all of the nodes of the graph, so suppose we restart the search at node 6. This search will not move away from 6, and so we restart the search again at node 7, visiting nodes 4 and 8. The finish times resulting from this search are

```
times.finish = {
    1: 10,
    2: 9,
    3: 4,
    4: 17,
    5: 8,
    6: 12,
    7: 18,
    8: 16,
    9: 7
}
```

Ordering these by decreasing finish time, we obtain a topological sort of:

$$7, 4, 8, 6, 1, 2, 5, 9, 3.$$

Your topological sort may be different, but still correct. This may happen if you restarted the search at different nodes than what was used above.