

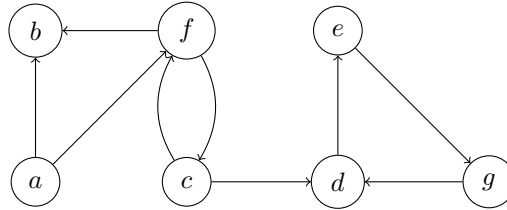
---

## DSC 40B - Discussion 07

---

### Problem 1.

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



- a) Suppose you call `bfs_shortest_paths(graph, 'c')` on the graph above. This function returns dictionaries `distance` and `predecessor`. Write down the contents of these dictionaries as they are when the function exits.

```
def bfs_shortest_paths(graph, source):
    status = {node: 'undiscovered' for node in graph.nodes}
    distance = {node: float('inf') for node in graph.nodes}
    predecessor = {node: None for node in graph.nodes}
    status[source] = 'pending'
    distance[source] = 0
    pending = deque([source])
    # while there are still pending nodes
    while pending:
        u = pending.popleft()
        for v in graph.neighbors(u):
            # explore edge (u,v)
            if status[v] == 'undiscovered':
                status[v] = 'pending'
                distance[v] = distance[u] + 1
                predecessor[v] = u
                # append to right
                pending.append(v)
        status[u] = 'visited'
    return predecessor, distance
```

#### Solution:

```
distance: {
    'a': ∞,
    'b': 2,
    'c': 0,
    'd': 1,
    'e': 2,
    'f': 1,
    'g': 3
}
predecessor: {
    'a': None,
    'b': 'f',
```

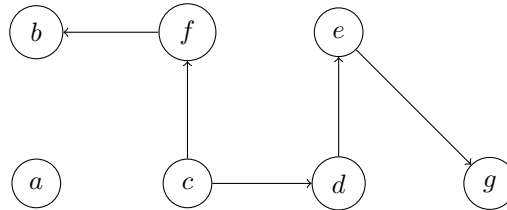
```

'c':None,
'd': 'c',
'e': 'd',
'f': 'c',
'g': 'e'
}

```

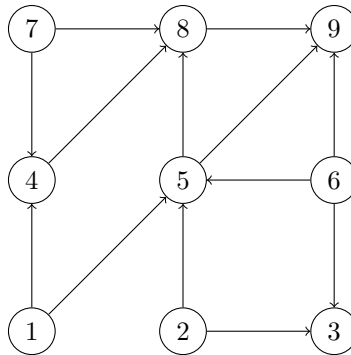
- b) Mark the BFS trees produced on executing BFS on this graph.

**Solution:**



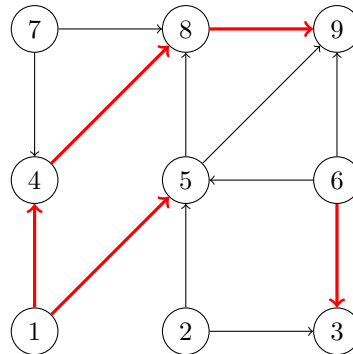
## Problem 2.

Consider the following directed graph.



- a) Make a bold arrow from node  $u$  to node  $v$  if  $u$  is the predecessor of node  $v$  in DFS. Use the convention that a node's neighbors are processed in ascending order by label.

**Solution:**



- b) 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	1	10
2	17	18
3	14	15
4	2	7
5	8	9
6	13	16
7	11	12
8	3	6
9	4	5

c) Topologically sort the vertices of the graph.

**Solution:** 2,6,3,7,1,5,4,8,9

### Problem 3.

State whether the following statements are true or false.

- a) Breadth first search on a directed graph always produces same number of BFS trees irrespective of order in which vertices are given and the neighbouring nodes are visited.

**Solution:** False.

- b) Breadth first search on an undirected graph always produces same number of BFS trees irrespective of order in which vertices are given and the neighbouring nodes are visited.

**Solution:** True.

- c) Both BFS and DFS require atleast  $\Omega(V)$  memory.

**Solution:** True.

- d) Consider a graph G on which BFS is run with node s as the source. Assume that BFS visits a node u in the graph before node v. Then  $d(s, u) < d(s, v)$

**Solution:** False.  $d(s, u) \leq d(s, v)$

- e) Every directed acyclic graph has exactly one topological ordering.

**Solution:** False. There can be multiple topological orderings depending on the order of nodes chosen during DFS.

#### Problem 4.

Given an undirected graph  $G=(V,E)$ , give an algorithm to find if the graph is disconnected.

**Solution:** Run BFS or DFS on a node in  $G$  (since  $G$  is undirected, any node will do). If the size of the set of visited nodes does not equal  $|V|$ , then the graph is disconnected.

Figure 1: "Full" DFS

```
from dataclasses import dataclass
@dataclass
class Times:
    clock: int
    start: dict
    finish: dict

def full_dfs_times(graph):
    status = {node: 'undiscovered' for node in graph.nodes}
    predecessor = {node: None for node in graph.nodes}
    times = Times(clock=0, start={}, finish={})
    for u in graph.nodes:
        if status[u] == 'undiscovered':
            dfs_times(graph, u, status, times)
    return times, predecessor

def dfs_times(graph, u, status, predecessor, times):
    times.clock += 1
    times.start[u] = times.clock
    status[u] = 'pending'
    for v in graph.neighbors(u):
        # explore edge (u, v)
        if status[v] == 'undiscovered':
            predecessor[v] = u
            dfs_times(graph, v, status, times)
    status[u] = 'visited'
    times.clock += 1
    times.finish[u] = times.clock
```