Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Short Answer

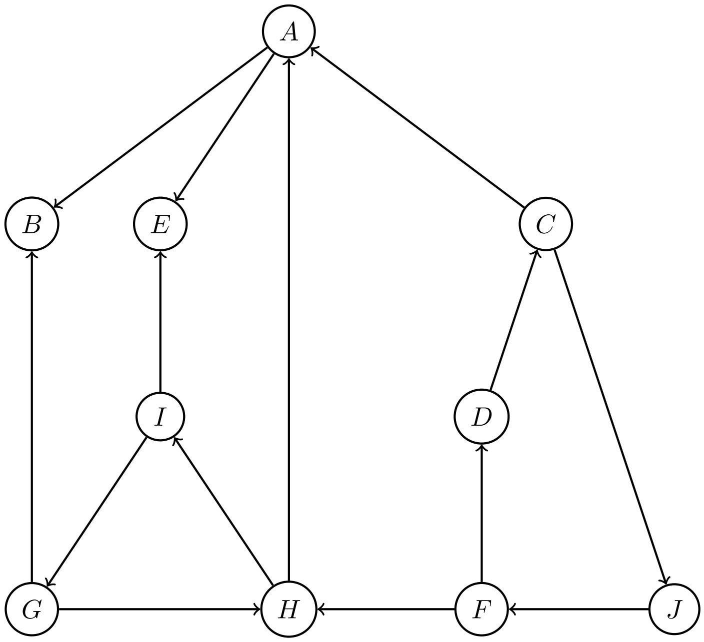
For each of the following, either prove the statement is true or give a counterexample to show it is false.

(a) If (u, v) is an edge in an undirected graph and during DFS, $\text{post}(v) < \text{post}(u)$, then u is an ancestor of v in the DFS tree.

(b) In a directed graph, if there is a path from u to v and $\text{pre}(u) < \text{pre}(v)$ then u is an ancestor of v in the DFS tree.

(c) In any connected undirected graph G there is a vertex whose removal leaves G connected.

2 Graph Traversal



(a) Recall that given a DFS tree, we can classify edges into one of four types:

- Tree edges are edges in the DFS tree,
- Back edges are edges (u, v) not in the DFS tree where v is the ancestor of u in the DFS tree
- Forward edges are edges (u, v) not in the DFS tree where u is the ancestor of v in the DFS tree
- Cross edges are edges (u, v) not in the DFS tree where u is not the ancestor of v, nor is v the ancestor of u.

For the directed graph above, perform DFS starting from vertex A, breaking ties alphabetically. As you go, label each node with its pre- and post-number, and mark each edge as **T**ree, **B**ack, **F**orward or **C**ross.

(b) What are the strongly connected components of the above graph?

(c) Draw the DAG of the strongly connected components of the graph.

3 Finding Clusters

We are given a directed graph $G = (V, E)$, where $V = \{1, \ldots, n\}$, i.e. the vertices are integers in the range 1 to n. For every vertex i we would like to compute the value $m(i)$ defined as follows: $m(i)$ is the smallest j such from which you can reach vertex i. (As a convention, we assume that i is reachable from i.)

(a) Show that the values $m(1), \ldots, m(n)$ can be computed in $O(|V| + |E|)$ time.

(b) Suppose we instead define $m(i)$ to be the smallest j that can be reached from i, instead of the smallest j from which you can reach i. How should you modify your answer to part (a) to work in this case?

4 BFS Intro

In this problem we will consider the shortest path problem: Given a graph $G(V, E)$, find the length of the shortest path from s to every vertex v in V. For an unweighted graph, the length of a path is the number of edges in the path. We can do this using the *breadth-first search* (BFS) algorithm, which we will see again in lecture this week.

BFS can be implemented just like the depth-first search (DFS) algorithm, but using a queue instead of a stack. Below is pseudo-code for another implementation of BFS, which computes for each $i \in \{0, 1, \ldots, n-1\}$ the set of vertices distance i from s, denoted L_i.

Input: A graph $G(V, E)$, starting vertex s
for all $v \in V$ **do**
    $visited(v) = False$
$visited(s) = True$
$L_0 \to \{s\}$
for i from 0 to $n-1$ **do**
    $L_{i+1} = \{\}$
    for $u \in L_i$ **do**
        for $(u, v) \in E$ **do**
            if $visited(v) = False$ **then**
                $L_{i+1}.add(v)$
                $visited(v) = True$

In other words, we start with $L_0 = \{s\}$, and then for each i, we set L_{i+1} to be all neighbors of vertices in L_i that we haven't already added to a previous L_i.

(a) Prove that BFS computes the correct value of L_i for all i (Hint: Use induction to show that for all i, L_i contains all vertices distance i from s, and only contains these vertices).

(b) Show that just like DFS, the above algorithm runs in $O(m + n)$ time.

(c) We might instead want to find the shortest *weighted* path from s to each vertex. That is, each edge has weight w_e, and the length of a path is now the sum of weights of edges in the path. The above algorithm works when all $w_e = 1$, but can easily fail if some $w_e \neq 1$.

Fill in the blank to get an algorithm computing the shortest paths when w_e are integers: We replace each edge e in G with _____ to get a new graph G', then run BFS on G' starting from s. Justify your answer.

(d) What is the runtime of this algorithm as a function of the weights w_e? How many bits does it take to write down all w_e? Is this algorithm's runtime a polynomial in the input size?