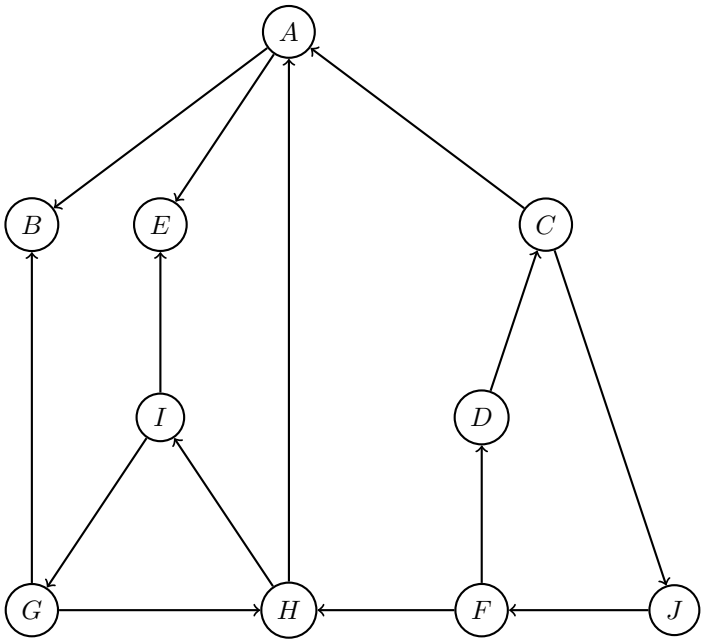
Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 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.

2 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 $i \in \{0, 1, \ldots, |V| - 1\}$ the set of vertices distance i from s, denoted L_i.

```
1: Input: A graph G(V, E), starting vertex s
2: for all v ∈ V do
3:     visited(v) = False
4: visited(s) = True
5: L_0 → {s}
6: for i from 0 to n − 1 do
7:     L_{i+1} = {}
8:     for u ∈ L_i do
9:         for (u, v) ∈ E do
10:            if visited(v) = False then
11:                L_{i+1}.add(v)
12:                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, where n is the number of nodes and m is the number of edges.

(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?

3 Counting Shortest Paths

Given an undirected unweighted graph G and a vertex s, let $p(v)$ be the number of distinct shortest paths from s to v. We will use the convention that $p(s) = 1$ in this problem. Give an $O(|V|+|E|)$-time algorithm to compute $p(v) \bmod 1337$ for all vertices. Only the main idea and runtime analysis are needed.

(Hint: For any v, how can we express $p(v)$ as a function of other $p(u)$?)

Note: As a secondary question, you should ask yourself whether the runtime would remain the same if we were computing $p(v)$ rather than $p(v) \bmod 1337$.

4 More Graph Proofs

Only prove the following statements for simple graphs (i.e. graphs that do not have any parallel edges or self-loops).

(a) An undirected graph G is called *bipartite* if we can separate its vertices into two subsets A and B, such that every edge in G must cross between A and B. Show that a graph is bipartite if and only if it has no odd cycles.
Hint: Consider a *spanning tree* of the graph, which is a subset of the graph's edges which forms a tree on all of its vertices.

(b) A directed acyclic graph G is *semiconnected* if for any two vertices A and B, there is either a path from A to B or a path from B to A. Show that G is semiconnected if and only if there is a directed path that visits all of the vertices of G.

5 Preorder, Postorder

Suppose we just ran DFS on a directed (not necessarily strongly connected) graph G starting from vertex r, and have the pre-visit and post-visit numbers $pre(v), post(v)$ for every vertex. We now delete vertex r and all edges adjacent to it to get a new graph G'. Given *just* the arrays $pre(v), post(v)$, describe how to modify them to arrive at new arrays $pre'(v), post'(v)$ such that $pre'(v), post'(v)$ are a valid pre-visit and post-visit ordering for some DFS of G'.