(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 Tree, Back, Forward or Cross.

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

(c) Draw the DAG of the strongly connected components of the above 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 $v \in V$ do
3:   visited($v$) = False
4:   visited($s$) = True
5:   $L_0 \rightarrow \{s\}$
6: for $i$ from 0 to $n-1$ do
7:   $L_{i+1} \rightarrow \{\}$
8: for $u \in L_i$ do
9:   for $(u,v) \in E$ 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 Dijkstra’s Algorithm Fails on Negative Edges

Draw a graph with five vertices or fewer, and indicate the source where Dijkstra’s algorithm will be started from.

(a) Draw a graph with no negative cycles for which Dijkstra’s algorithm produces the wrong answer.

(b) Draw a graph with at least two negative weight edges for which Dijkstra’s algorithm produces the correct answer.

4 Waypoint

You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights, and there is a special node $v_0 \in V$. Give an efficient algorithm that computes, for all node pairs $s, t$, the length of the shortest path from $s$ to $t$ that passes through $v_0$. Your algorithm should take $O(|V|^2 + |E| \log |V|)$ time.

5 Dijkstra Tiebreaking

We are given a directed graph $G$ with positive weights on its edges. We wish to find a shortest path from $s$ to $t$, and, among all shortest paths, we want the one in which the longest edge is as short as possible. How would you modify Dijkstra’s algorithm to this end? Just a description of your modification is needed.

(If there are multiple shortest paths where the longest edge is as short as possible, outputting any of them is fine).