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 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 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 \textit{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: \textbf{Input}: A graph $G(V, E)$, starting vertex $s$
2: \textbf{for all} $v \in V$ \textbf{do}
3: \hspace{1em} $visited(v) = \text{False}$
4: \hspace{1em} $visited(s) = \text{True}$
5: \hspace{1em} $L_0 \rightarrow \{s\}$
6: \textbf{for} $i$ \textbf{from} 0 \textbf{to} $n - 1$ \textbf{do}
7: \hspace{1em} $L_{i+1} = \{\}$
8: \hspace{1em} \textbf{for} $u \in L_i$ \textbf{do}
9: \hspace{2em} \textbf{for} $(u, v) \in E$ \textbf{do}
10: \hspace{3em} \textbf{if} $visited(v) = \text{False}$ \textbf{then}
11: \hspace{4em} $L_{i+1}.\text{add}(v)$
12: \hspace{3em} $visited(v) = \text{True}$

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 \textit{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 positive integers.

We replace each edge $e$ in $G$ with \underline{} 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 Running Errands

You need to run a set of $k$ errands in Berkeley. Berkeley is represented as a directed weighted graph $G$, where each vertex $v$ is a location in Berkeley, and there is an edge $(u, v)$ with weight $w_{uv}$ if it takes $w_{uv}$ minutes to go from $u$ to $v$. The errands must be completed in order, we’ll assume the $i$th errand can be completed immediately upon visiting any vertex in the set $S_i$ (for example, if you need to buy snacks, you could do it at any grocery store). Your home in Berkeley is the vertex $h$.

Given $G$, $h$, and all $S_i$ as input, given an efficient algorithm that computes the time needed to complete all the errands starting at $h$. That is, find the shortest path in $G$ that starts at $h$ and passes through a vertex in $S_1$, then a vertex in $S_2$, then in $S_3$, etc.

Give a 3-part solution.
5 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.

6 Inequalities

Given a list of $n$ variables $x_1, \ldots, x_n$ and $m$ inequalities of the form $x_i < x_j$ or $x_i \leq x_j$ for some $i, j \in [n]$, you would like to find values for the variables such that all inequalities are satisfied, or determine that not all inequalities can be satisfied simultaneously.

(a) Design an efficient algorithm for the case that all inequalities are inequalities are strict (i.e. of the form $x_i < x_j$).

Give a concise description of the algorithm (proof of correctness or runtime analysis are not needed).

(b) Design an efficient algorithm that solves the above problem in general (when some inequalities are of the form $x_i < x_j$, and some are of the form $x_i \leq x_j$).

Give a concise description of the algorithm (proof of correctness or runtime analysis are not needed).