1. **Solo Journeys: The Love Lives of 374 Students** A directed graph $G = (V, E)$ is *singly connected* if $u \rightsquigarrow v$ implies that G contains at most one simple path from u to v for all vertices $u, v \in V$. Give an efficient algorithm to determine whether a directed graph is singly connected.

Solution: To determine whether a directed graph $G = (V, E)$ is singly connected, we can use the following approach:

Initialization: For each vertex $u \in V$, initialize a visited array and start a DFS from u.

DFS-VISIT: During the DFS traversal, classify each edge into one of the following categories: **Tree Edge**: An edge (u, v) where v is first discovered. **Back Edge**: An edge (u, v) where v is an ancestor of u in the DFS tree. **Forward Edge**: An edge (u, v) where v is a descendant of u but not a tree edge. **Cross Edge**: Any other edge that is neither a tree, back, nor forward edge. **Edge Classification**: If any edge is classified as a forward or cross edge during any DFS execution, the graph is not singly connected. If all edges in every DFS execution are either tree edges or back edges, the graph is singly connected.

Termination: The algorithm stops early if a forward or cross edge is found.

Proof of Correctness

Lemma: Directed graph $G = (V, E)$ is not singly connected if and only if for some vertex $u \in V$, a call of DFS-VISIT(G, u) yields a forward or cross edge.

Proof:

If G is not singly connected, suppose G is not singly connected. Then there exist vertices u and v such that there are at least two simple paths from u to v. Let p_1 be one simple path from u to v formed by tree edges during the DFS. Let p_2 be another simple path from u to v. Since p_2 is different from p_1, it must contain at least one edge e that is not in p_1. Edge e cannot be a tree edge, as it would contradict the uniqueness of the tree path. Edge e cannot be a back edge, as p_2 would not be simple (it would contain a cycle). Therefore, e must be either a forward or cross edge.

If a call of DFS-VISIT(G, u) yields a forward or cross edge, suppose during the DFS from u, an edge (v, w) is classified as a forward or cross edge. If (v, w) is a forward edge, there are two paths from v to w: the direct edge (v, w) and the path from v to w through the DFS tree. If (v, w) is a cross edge, there are two paths from u to w: one path $u \to w$ through the DFS tree and another path $u \to v \to w$, where v is an ancestor of w in the DFS tree. In both cases, the existence of multiple simple paths from u to w implies that G is not singly connected.

By proving the contrapositive, we conclude that G is singly connected if and only if no call of DFS-VISIT yields a forward or cross edge.

Time Complexity

The algorithm involves running DFS from each vertex, leading to a time complexity of $O(V \times (V + E))$. In the worst case, this is $O(V^2 + VE)$, which simplifies to $O(VE)$ when considering dense graphs.

Thus, the algorithm efficiently determines whether a directed graph is singly connected by utilizing DFS and edge classification.

■

2. **Premature Termination** Find diagram 8.7 in chapter 8 under additional resources of Lecture 16. Suppose we change the first line of diagram 8.7 to read "for the first |V|-1 vertices in topological order" instead. Will this algorithm still work? Why or why not? Justify your answer. (That way if your wrong I can still give you pity points.)

Solution: The last vertex in the topological ordering of a Directed Acyclic Graph (DAG) is always a sink. Sink vertices have no outgoing edges. Therefore, since there are no outgoing edges from the last vertex in the topological ordering, there are no edges to relax. Thus, this adjustment will work. ■

3. **When Dijkstra F*s Up** Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces an incorrect answer.

Solution: See Dijkstra's Algorithm on Negative-weighted Graphs for an example. ■

4. **Bitonic Befuddlement** A sequence is *bitonic* if it monotonically increases and then monotonically decreases, or if by a circular shift it monotonically increases and then monotonically decreases. For example the sequences $\langle 1, 4, 6, 8, 3, -2\rangle$, $\langle 9, 2, -4, -10, -5\rangle$, and $\langle 1, 2, 3, 4\rangle$ are bitonic, but $\langle 1, 3, 12, 4, 2, 10\rangle$ is not bitonic.

Suppose that you are given a directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, where all edge weights are unique, and you wish to find single-source shortest paths from a source vertex s. You are given one additional piece of information: for each vertex $v \in V$, the weights of the edges along any shortest path from s to v form a bitonic sequence.

Solution: $v.d$ is the shortest-path estimate, which is an upper bound on the weight of a shortest path from the source s to v.
$v.\pi$ is the predecessor of v on the shortest path from s to v.

Observe that a bitonic sequence can increase, then decrease, then increase, or it can decrease, then increase, then decrease. That is, there can be at most two changes of direction in a bitonic sequence. Any sequence that increases, then decreases, then increases, then decreases has a bitonic sequence as a subsequence.

Now, let us suppose that we had an even stronger condition than the bitonic property given in the problem: for each vertex $v \in V$, the weights of the edges along any shortest path from s to v are increasing. Then we could call `INITIALIZE-SINGLE-SOURCE` and then just relax all edges one time, going in increasing order of weight. Then the edges along every shortest path would be relaxed in order of their appearance on the path. (We rely on the uniqueness of edge weights to ensure that the ordering is correct.) The path-relaxation property (Lemma 22.15) would guarantee that we would have computed correct shortest paths from s to each vertex.

If we weaken the condition so that the weights of the edges along any shortest path increase and then decrease, we could relax all edges one time, in increasing order of weight, and then one more time, in decreasing order of weight. That order, along with the uniqueness of edge weights, would ensure that we had relaxed the edges of every shortest path in order, and again the path-relaxation property would

guarantee that we would have computed correct shortest paths.

To make sure that we handle all bitonic sequences, we do as suggested above. That is, we perform four passes, relaxing each edge once in each pass. The first and third passes relax edges in increasing order of weight, and the second and fourth passes in decreasing order. Again, by the path-relaxation property and the uniqueness of edge weights, we have computed correct shortest paths.

The total time is $O(V + E \lg V)$, as follows. The time to sort $|E|$ edges by weight is $O(E \lg E) = O(E \lg V)$ (since $|E| = O(V^2)$). `INITIALIZE-SINGLE-SOURCE` takes $O(V)$ time. Each of the four passes takes $O(E)$ time. Thus, the total time is $O(E \lg V + V + 4E) = O(V + E \lg V)$. ■

5. **Ugh Yuck! A proof. Gross.** Prove the following statement. If G has a topological ordering, then G is a DAG. [Hint: contradiction is your friend ;)]

Solution: Proof. Suppose, by way of contradiction, that G has a topological ordering $v_1, v_2, \ldots, v_n$, and also has a cycle C. Let v_i be the lowest-indexed node on C, and let v_j be the node on C just before v_i—thus (v_j, v_i) is an edge. But by our choice of i, we have $j > i$, which contradicts the assumption that $v_1, v_2, \ldots, v_n$ was a topological ordering. ■