Practice Problems 2

Problem 1. In this problem you will simulate the strong-components algorithm given in class on the digraph shown in Fig. 1.

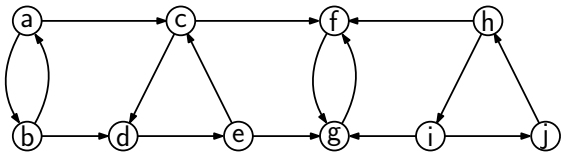


Figure 1: Computing strong components.

(a) Draw the reverse graph G^R. (Be careful to note the directions of the edges.)

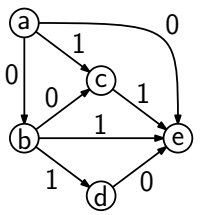
(b) Show the result of applying DFS to G^R. Label each vertex u with its discovery and finish times ($d[u]/f[u]$). You need only show the *final* DFS tree, not the intermediate results.

To make the grader's life easier please draw your DFS forest in the same manner as in Fig. 5 of Lecture 3. Whenever you have a choice of which vertex to visit next, select the lowest vertex in alphabetic order.

(c) Show the result of running DFS to G, where the main program selects the next vertex to visit based on decreasing order of finish times from part (b).

(d) Illustrate (e.g., by circling them or listing them out) the strong components of G.

Problem 2. You are given a DAG (directed acyclic graph) $G = (V, E)$ in which each edge has a label label(u, v) which is either 0 or 1. An *alternating path* is defined to be a path containing at least one edge, such that the edges along the path alternate in label between 0 and 1. The path may begin with a 0-label edge or a 1-label edge. **Note:** The empty path is *not* an alternating path, and a path with a single edge is always an alternating path. Paths need not be of maximal length. For each vertex $u \in V$, let alt(u) denote the number of alternating paths that start at u. For example, in Fig. 2, alt$(b) = 5$ because of the paths $\{\langle b, c\rangle, \langle b, c, e\rangle, \langle b, d\rangle, \langle b, d, e\rangle, \langle b, e\rangle\}$.



u	alt(u)
a	6
b	5
c	1
d	1
e	0

Figure 2: Alternating paths in a DAG.

Present an algorithm which, given a DAG $G = (V, E)$ with 0-1 edge labels, computes $\text{alt}(u)$ for all $u \in V$. Briefly justify your algorithm's correctness and derive its running time. Ideally, your algorithm should run in $O(n + m)$ time.

Hint: Use DFS, where each vertex stores the two counts for the number of alternating paths, one for paths starting with 0 and the other with 1.

Problem 3. For both parts below, assume that graphs and digraphs are represented using an adjacency list. Given a graph or digraph $G = (V, E)$, let $n = |V|$ and $m = |E|$.

(a) Present an algorithm which, given an undirected graph $G = (V, E)$, determines whether G contains a cycle in $O(n)$ time. If the graph has a cycle, it should output the vertices of any cycle.

It is important that the running time of your algorithm is independent of the number of edges m, which may generally be as high as $\Omega(n^2)$. This means that your algorithm will need to terminate with the correct answer, possibly even before reading the entire adjacency list. You may assume that the graph is presented to your algorithm in constant time as reference to its existing adjacency list.

(b) Prove that there is no corresponding algorithm for digraphs. In particular, prove that any algorithm that correctly determines whether a digraph has a cycle may need to inspect $\Omega(n^2)$ edges.

Hint: Prove this by contradiction. Suppose that such an algorithm existed. Show that there exists a digraph having $\Omega(n^2)$ edges, such that any correct algorithm algorithm must inspect *every* edge of this graph.

Problem 4. A digraph $G = (V, E)$ is said to be *semi-connected* if, given any two vertices, $u, v \in V$, there exists a path from u to v, or there exists a path from v to u (possibly both). Give an efficient algorithm which, given a directed acyclic graph (DAG), determines whether it is weakly connected.

Briefly justify your algorithm's correctness and derive its running time. Ideally, you algorithm should run in $O(n+m)$ time. (**Hint:** Try drawing a few semi-connected DAGs. What special structure do these DAGs have in common?)

Problem 5. Dijkstra's algorithm assumes that all the edge weights in a digraph are nonnegative. For this problem, assume the version of Dijkstra's algorithm that was given in class.

(a) Present an (ideally small) example that shows that Dijkstra's algorithm may fail to produce a correct result if the digraph has even a single negative-weight edge. (In particular, on termination, the d-value for at least one vertex is incorrect.)

(b) Suppose that your digraph has negative-weight edges, but these edges all emanate from the source vertex. Also, the digraph has not negative-cost cycles. Prove that Dijkstra's algorithm produces a correct result on such a digraph.