1. **DFS mechanics.** For each graph, show the “DFS forest” resulting from an execution of DFS. Whenever there is a choice of vertices, choose the one that is alphabetically first. Identify the cross, forward, and back edges, and label each vertex with its discovery and finishing time.

![Graph (i)](image1)

2. **DFS mechanics.** Run the DFS-based topological sort algorithm on the following graph. Whenever there is a choice of vertices, choose the one that is alphabetically first. Show the “DFS forest”, including discovery and finishing times. Give the resulting topological ordering of the vertices.

![Graph (ii)](image2)

3. **DFS mechanics.** Run the strongly-connected components algorithm on this graph. Show the “DFS forest”, including discovery and finishing times, for both runs of the DFS algorithm. Draw the resulting component graph. As usual, when faced with a choice among vertices, pick the one that is alphabetically first.

![Graph (iii)](image3)

4. **Dijkstra mechanics.** Run Dijkstra’s algorithm on the following graph, starting from vertex A. Whenever there is a choice of vertices, choose the one that is alphabetically first. Show the state of the algorithm (d array, π array, Q, and R) just after initialization and at the end of each loop iteration.

![Graph (iv)](image4)
5. **Back to your roots.** Let \( G = (V, E) \) be a directed graph. A *root* of \( G \) is a vertex from which every other vertex is reachable (i.e., for every \( v \in V \) there is a path from \( r \) to \( V \)).

Now suppose \( G \) is *acyclic*. Give an argument for each of the following:

(i) if \( r \) is a root, then it must have in-degree 0 (i.e., if \( r \) has in-degree > 0, then it cannot be a root);
(ii) if there is more than one node of in-degree 0, then there is no root;
(iii) if there is exactly one node \( r \) of in-degree zero, then \( r \) is a root.

*Note:* this says that in a DAG, there can be at most one root, and that the graph has a root if and only if there is a unique vertex of in-degree zero (in which case, that vertex is the root).

6. **Gotham City.** The mayor of Gotham City has made all the streets of the city one way. We can model the street layout as a directed graph \( G = (V,E) \), where the vertices \( V \) of the graph correspond to street intersections (locations), and the edges \( E \) correspond the the roads connecting intersections. So for every pair of distinct vertices \( u,v \in V \), if there is an edge from \( u \) to \( v \), then there is no edge from \( v \) to \( u \). Assume a sparse representation for \( G \).

(a) The mayor claims that one can legally drive from any one location in the city to any other. Show how to verify the mayor’s claim using a linear-time algorithm that takes as input the graph \( G \).

(b) Suppose the mayor’s claim is not necessarily true: that it may not be possible to legally drive from any one location to any other.

Let us call a location *dominant* if you can legally drive from that location to any other location. Said another way, \( v \) is dominant if the following property holds: for every location \( w \), one can legally drive from \( v \) to \( w \).

Give a linear-time algorithm that outputs all of the dominant locations, given the graph \( G \) as input.

(c) Suppose again that the mayor’s claim is not necessarily true.

Let us call a location \( v \) *safe* if wherever you legally drive starting at \( v \), you can always legally drive back to \( v \). Said another way, \( v \) is safe if the following property holds: for every location \( w \), if one can legally drive from \( v \) to \( w \), then one can legally drive back from \( w \) to \( v \).

Give a linear-time algorithm that outputs all of the safe locations, given the graph \( G \) as input.

(d) Suppose again that the mayor’s claim is not necessarily true. Nevertheless, an emergency arises, and you *must* drive from location \( s \) to location \( t \), even though there may not be a legal route from \( s \) to \( t \).

You want to plan a route that traverses some edges in the graph in the reverse (i.e., illegal) direction. In order to minimize the chance of getting caught by the police, your planned route should minimize the number of illegal edges. The number of legal edges in the route is irrelevant.

Give a linear-time algorithm that computes a route from \( s \) to \( t \) that minimizes the number of illegal edges (or reports that no route exists), given the graph \( G \) and locations \( s \) and \( t \) as input.

*Hints:* all of the above problems can be very easily solved by using standard algorithms as subroutines; for part (b), the previous exercise may be useful.

7. **Most valuable node.** You are given a directed graph \( G = (V,E) \) in which each node \( v \in V \) has an associated value \( c[v] \), which is a positive number. You are also given a node \( s \in V \).

Your goal is to find the most valuable node \( v \) that is reachable from \( s \). That is, among all nodes \( v \) reachable from \( s \), find one for which \( c[v] \) is largest.

Your algorithm should run in linear time.

*Hint:* solve the problem first assuming the graph is acyclic; use standard algorithms as subroutines.

8. **Patriotic paths.** Let \( G = (V,E) \) be a directed graph. Each edge has a *weight*, which is a non-negative number, and a color, which is either *red*, *white*, or *blue*. Let \( s,t \in V \). A path from \( s \) to \( t \) is called *patriotic* if it starts with a sequence of one or more *red* edges, followed by a sequence of one or more *white* edges, and ends with a sequence of one or more *blue* edges.

Your task is to design an efficient algorithm that finds a patriotic path from \( s \) to \( t \) of minimum weight (or reports that there is no patriotic path). What is the running time of your algorithm, stated in terms of \( n := |V| \) and \( m := |E| \)?

*Hint:* use standard algorithms as subroutines.
9. **Running on empty.** In this problem, we want to determine how to drive a car from point $s$ to point $t$ without running out of gas. Our car has a gas tank that is initially filled up to its capacity $c$. We may have to stop to refuel along the way: we can never allow the amount of gas in the tank to become negative, and we can never fill our tank beyond its capacity. Let us model this problem using a directed graph $G = (V, E)$ with non-negative edge weights $w_t : E \rightarrow \mathbb{R}_{\geq 0}$, along with nodes $s, t \in V$. For an edge $(u, v) \in E$, $w_t(u, v)$ represents the amount of gas required to drive from $u$ to $v$. Also, a subset $F \subseteq V$ represents those points where we may stop to refuel (assume each node $v$ is marked with a bit $f(v)$ indicating whether there is a gas station located at $v$).

Give an efficient algorithm to solve this problem. Your algorithm should determine if there is a viable path from $s$ to $t$, and if so, output a path that uses the minimum amount of gas. The running time of your algorithm should be $O(|F|(|V| + |E|) \log |V|)$.

*Hint:* use Dijkstra’s shortest path algorithm as a subroutine. You will have to call this subroutine many times.

10. **Dijkstra ascending.** Referring to Dijkstra’s algorithm as presented in class:

   (a) Show that at any point in time, we have
   
   $$\delta(s, x) \leq \delta(s, y)$$
   
   for all $x \in R$ and $y \in V \setminus R$.
   
   *Hint:* this can be done with an induction argument very similar to what was done in the proof of correctness of Dijkstra’s algorithm.

   (b) Show that if $u$ is removed from $Q$, then the next node $v$ removed from $Q$ satisfies $\delta(s, u) \leq \delta(s, v)$.
   
   *Note:* this says that nodes are removed from $Q$ in order of increasing distance from $s$.
   
   *Hint:* this follows very easily from part (a).