1. 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.

![Graphs](image)

2. 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](image)

3. Here is another algorithm for performing a topological sort. On a high level, for a graph $G$ with vertices $V$ and edges $E$, it works like this:

   while (V not empty)
   find a source vertex $v \in V$
   output $v$
   remove $v$ from $V$ and all edges of the form $v \rightarrow w$ from $E$

Here, a source vertex is a vertex with no oncoming edges.

(a) Illustrate the execution of this algorithm on the graph in the previous exercise — use the usual alphabetical rule to break ties.

(b) Show how to implement this algorithm in linear time, i.e., time $O(|V|+|E|)$, assuming an adjacency list representation for $G$. Give a detailed description of your algorithm and its data structures. What does your algorithm do if $G$ contains a cycle?
4. 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.

5. **Income.** You are given a directed (possibly cyclic) graph $G = (V, E)$ with $n$ vertices and $m$ edges, along with two distinguished vertices $s, t \in V$. Each edge $e \in E$ has a cost $c(e)$, which is a nonnegative integer. Each vertex $v \in V$ has an income $I(v) \in E$, which is also a nonnegative integer. For a path $p = \langle v_0, v_1, \ldots, v_k \rangle$, we define the net income of $p$ to be the sum of the income derived from the distinct vertices in $p$, minus the sum of all the edges (counted with multiplicities) in $p$.

Intuitively, you can think of the problem as follows: on each vertex there is some money, which you can pick up if you visit that vertex; however, once you pick up the money, it’s gone. In addition, every time you cross an edge, you pay a toll, and if you cross the same edge several times, you pay the toll each and every time.

Your goal is to find a path from $s$ to $t$ in $G$ that generates the most net income. For simplicity, you can just focus on the problem of calculating the maximum net income, rather than the path. Assume the the graph is given in adjacency list form, and that you can fetch $I$ and $c$ values in constant time.

(a) Show how to solve this problem in time $O(n + m)$ assuming that $G$ is acyclic. *Hint:* use a topological sorting algorithm as a subroutine.

(b) Show how to solve this problem in time $O(n + m)$, assuming that $c(e) = 0$ for all $e \in E$ (but $G$ may be cyclic). *Hint:* use an SCC algorithm as a subroutine.

6. **Carrying stones.** You are given a directed acyclic graph $G = (V, E)$ with $n$ vertices and $m$ edges, along with two distinguished vertices $s, t \in V$. At each vertex $v \in V$, there are a number of stones $q(v)$ (so $q(v)$ is a nonnegative integer). Your goal is to find a path from $s$ to $t$, picking up stones along the way, and arrive at $t$ carrying as many stones as possible. At each node $v$ along the path, you are allowed to pick up at most $q(v)$ stones (and for simplicity, assume that $q(t) = 0$). However, there is a complication: each edge $e \in E$ has a capacity $c(e)$ (which is a positive integer), and you are not allowed to carry more than $c(e)$ stones across that edge.

Show how to solve this problem in time $O(n + m)$. The output should be an optimal path from $s$ to $t$, and for every vertex along the path, your output should include the number of stones to be picked up at that vertex. Assume the the graph is given in adjacency list form, and that you can fetch $q$ and $c$ values in constant time.

*NOTE:* You may want to first solve a variant of the above problem in which at each vertex, in addition to picking up stones, you may also throw some stones away. It should be easy to modify your solution to this variant to obtain a solution to the original problem.

*Hint:* first do a topological sort on the reversed graph.