Problem 1. Suppose $G = (V, E)$ is a directed acyclic graph represented by adjacency lists. Devise a linear time algorithm that, given such a $G$, returns the length of the longest path in $G$. Prove your algorithm runs in $O(|V| + |E|)$-time.

**SOLUTION:** Note that the longest path from vertex $u$ is $\max\{1 + \text{the length of the longest path from } v : (u, v) \in E\}$ where, by convention, $\max(\emptyset) = 0$. Now, to solve the problem, first do a topological sort of $G$ and number $G$’s vertices 1 through $n$ where $i < j$ implies that vertex $i$ comes before vertex $j$ in the topological sort. Next compute:

1. for $i = n, n - 1, \ldots, 1$ do
2. $\text{longest}[i] \leftarrow 0$
3. for each $j$ adjacent to $i$ do // so $(i, j) \in E$
4. $\text{longest}[i] \leftarrow \max(\text{longest}[i], 1 + \text{longest}[j])$
5. return $\max\{\{\text{longest}[i] : i = 1, \ldots, n\}\}$

Since we are considering vertices in reverse-top-sort order, we know that in line 4, each $\text{longest}[j]$ has its correct value, hence when we are done with the for-loop of lines 3–4, $\text{longest}[i]$ has its correct value.

**Runtime analysis:** Line 2 is done once for each $i \in V$, line 4 is done once for the $(i, j) \in E$ and line 5 takes $O(n)$ time. Hence, the entirety takes $O(|V| + |E|)$-time.

Problem 2. Suppose $G = (V, E)$ is a directed graph represented by adjacency lists. Devise a linear time algorithm that, given such a $G$, returns a list of all the source vertices of $G$. Prove your algorithm runs in $O(|V| + |E|)$-time.

**SOLUTION:** The in-degree of a vertex $v$ = the number of edges with end-point $v$. So, $v$ is a source iff $\text{in-degree}(v) = 0$.

Now, we can compute in-degrees and identify sources as follows:

1. for each $u \in V$ do $\text{inDegree}[u] \leftarrow 0$
2. for each $(u, v) \in E$ do
3. $\text{inDegree}[v] \leftarrow \text{inDegree}[v] + 1$
4. return $\{v : v \in V, \text{inDegree}[v] = 0\}$

Clearly, this is $O(|V| + |E|)$.

Problem 3. DPV Problem 3.18

Starting at the root vertex do a DFS with pre and post numbering. Then: $u$ is an ancestor of $v$ in the tree $\iff$ \textbf{Corrected}

$$ (\text{pre}[u], \text{post}[u]) \supset (\text{pre}[v], \text{post}[v]) \tag{1} $$

and we can test (1) in constant time.

Problem 4. DPV Problem 3.22

In the case where we know that $G$ is a directed acyclic graph, there is such a vertex in $G$ if and only if $G$ has exactly one source vertex. Problem 2 showed how to determine the source vertex of a dag in linear time.

In the case where $G$ is not necessarily a dag, compute the strongly connected component dag of $G$ (which takes linear time) and then test if that dag has just one source.

Problem 5. DPV Problem 3.23

This is a variation of what we did for Problem 1. First do a topological sort of $G$. Remove all vertices that are either before $s$ or after $t$ in the topological-sort (because they cannot be in any path from $s$ to $t$). Number the remaining vertices 1 through $n$ in topological sort order. (So $s = 1$ and $t = n$.) Next compute:

1. $\text{paths}[n] \leftarrow 1$
2. for $i = n - 1, \ldots, 1$ do
3. $\text{paths}[i] \leftarrow 0$
4. for each $j$ adjacent to $i$ do // so $(i, j) \in E$
5. $\text{paths}[i] \leftarrow \text{paths}[i] + \text{paths}[j]$
6. return $\text{paths}[1]$

Problem 6. DPV Problem 4.1

(a) Here is the table. \textbf{Corrected}

<table>
<thead>
<tr>
<th>Step</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

(b) Here is the tree. Solid edges make up the tree; edges not in the tree are dashed.