Design an efficient algorithm to find the longest path in a directed acyclic graph. (Partial credit will be given for a solution where each edge has weight 1; full credit for solutions that handle general real-valued weights on the edges, including negative values.)

Solution
First, we perform a topological sort on the vertices, which is possible because the graph is a DAG. As we saw in lecture, this can be done using DFS in $O(|V| + |E|)$ time. Now, we initialize two arrays $\text{dist}[\cdot]$ and $\text{prev}[\cdot]$ both of length $|V|$.

- $\text{dist}[v]$ stores the length of the longest path that ends at $v$.
- $\text{prev}[v]$ stores the previous node in the longest path that ends at $v$

We initialize $\text{dist}[\cdot]$ to be all 0’s and $\text{prev}[\cdot]$ to be all NULL.

Now, we will calculate $\text{dist}[v]$ and $\text{prev}[v]$ by iterating through the vertices $v \in V$ in topological order. For a particular node $v$, we perform the following update:

$$\text{dist}[v] = \max_u (0, \text{dist}[u] + \omega(u, v))$$

In the update above, if we choose 0 as the maximum, then set $\text{prev}[v] = \text{NULL}$. Otherwise, if we find a $u$, we set $\text{prev}[v] = u$. Intuitively, the longest path either (a) begins and ends at $v$ (in which case it has length 0), or (b) is an extension of the longest path ending at $u$ for some neighbor $u \to v$. In the second case, we take the maximum out of all such neighbors $u$ of the length of the extended path.

Once we have done the above for every vertex, we can scan through the list $\text{dist}$ for the index $v$ with the maximum value. This implies that the longest path in the graph ends at $v$. If we want the length of the longest path, we can output $\text{dist}[v]$. If we want the path itself, we can look at $\text{prev}[v]$ to find the second-to-last node in the longest path, and then look at $\text{prev}[\text{prev}[v]]$ to find the third-to-last node...etc until we find that $\text{prev}$ of something is NULL (this implies there is no more predecessor in the longest path ending at this node, so this longest path must have begun at this node).

Proof of Correctness
Since we fill in the values of the two arrays in topological ordering, we can use induction on this ordering to prove correctness. Suppose that we are trying to calculate $\text{dist}[v]$ and $\text{prev}[v]$. If $v$ is a source, then $\text{dist}[v] = 0, \text{prev}[v] = \text{NULL}$ is correct as initialized at the beginning. Otherwise, assume that we have correctly calculated $\text{dist}[u]$ and $\text{prev}[u]$ for all vertices $u$ that come before $v$ in the topological ordering. The longest path ending at $v$ either began at $v$ or is an extension from the longest path ending at a neighbor of $v$. Then, our update is correct because it finds the maximum possible length out of these possibilities, which exhaustively cover all possible paths that end at $v$. Thus, our algorithm is correct.
**Run-time Analysis** The initial topological sort takes $O(|V| + |E|)$ by DFS. Iterating through each of the nodes, at each node $v$, we take the maximum of a number of elements equal to the in-degree of $v$. Over the course the entire algorithm, for every edge $(u, v)$, that edge contributes to finding the maximum of another element when we are trying to calculate $\text{dist}[v]$ and $\text{prev}[v]$. For each $v \in V$, we also do a small fixed cost of calculation (such as assignment of the array). Therefore, the run-time of this part of the algorithm is also $O(|V| + |E|)$, which means the overall run-time is $O(|V| + |E|)$. 