Exercise 12.10 Let $G = (V, E)$ be a directed graph with m edges and n vertices, where each vertex $v \in V$ is given an integer label $\ell(v) \in \mathbb{N}$. The goal is to find the length of the longest path[3] in G where the labels of the vertices are (strictly) increasing.

[3]Recall that a path is a walk that does not repeat vertices.

Exercise 12.10.1. Suppose G is a DAG. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

Solution. We claim the following polynomial-time algorithm suffices.

lip-dag(s):

/ given $s \in V$ in a DAG, computes the length of the longest strictly-increasing path starting at vertex s, measured by the number of edges. */*

1. Let $m \leftarrow 0$.
2. For $(s, v) \in \delta^+(s)$ such that $\ell(v) > \ell(s)$ do
   A. m $\leftarrow$ max $(m, 1 + \texttt{lip}(v))$
3. return m

We can solve the original problem by computing max $_{\forall v \in V}$ lip-dag(v).

Runtime. If we cache the return value of lip-dag(v) for all $v \in V$ (as well as the maximum of all these return values), our algorithm has the following runtime complexity:

$$O\left(n \cdot \left(1 + \sum_{v \in V} d^+(v)\right)\right) = O(m + n)$$

■

Correctness. The correctness of this algorithm follows from performing induction, in reverse topological order, according to the recursive specification. We claim that for all vertices $s \in V$, the algorithm correctly computes the length of the longest strictly increasing path starting with vertex s, measured in the number of edges.

In the base case, s is the last vertex in topological order, so s is a sink and the longest increasing path starting from s has length 0, as returned in the algorithm.

Now assume that the claim holds for some vertex s as well as all vertices that follow s in topological order. Take $s' \in V$ to be a vertex that immediately preceeds s in topological order, and notice that the claim holds for all v such that $(s', v) \in \delta^+(s')$. If s' has no out-neighbors with larger weight, then the algorithm correctly returns a maximum increasing path length of 0. Otherwise, the longest path starting at s' has length

$$\max_{v \,:\, (s',v) \,\in\, \delta^+(s'), \ell(v) > \ell(s')} \texttt{lip-dag}(v)$$

as computed in the algorithm.

Hence, by induction, the claim—algorithmic correctness—holds for all $s \in V$.

■

□

Exercise 12.10.2. Consider now the problem for general graphs. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

Solution. The intuition here is that, because it is not possible to have a strictly-increasing cycles, we can treat G just as we would a DAG.

We claim the same polynomial-time algorithm as in 12.10.1 solves this problem.

`lip`(s):

/ given $s \in V$, computes the length of the longest strictly-increasing path starting at vertex s, measured by the number of edges. */*

1. Let $m \leftarrow 0$.
2. For $(s, v) \in \delta^+(s)$ such that $\ell(v) > \ell(s)$ do
   A. m $\leftarrow$ max $(m, 1 + \texttt{lip}(v))$
3. return m

Again, we can solve the original problem by computing max $_{\forall v \in V}$ `lip`(v).

Runtime. If we cache the return value of `lip`(v) for all $v \in V$ (as well as the maximum of all these return values), our algorithm has the following runtime complexity:

$$O\left(n \cdot \left(1 + \sum_{v \in V} d^+(v)\right)\right) = O\left(m + n\right)$$

■

Correctness. To clarify the DAG-like nature of this traversal (i.e. why there are no cyclic dependencies in the recursive calls), consider calling `lip` on some vertex $s \in V$. We make recursive calls to adjacent vertices $v \in V$, $(s, v) \in \delta^+(s)$ only when $\ell(v) > \ell(s)$, which creates a sort of topological ordering-by-weight on G.

Explicitly: we claim that for all vertices $s \in V$, the algorithm correctly computes the length of the longest strictly increasing path starting with vertex s, measured in the number of edges. To prove this, we perform induction based on the weight $\ell(v)$ of all $v \in V$.

In the base case, consider $s \in V$ with maximal weight; that is, $\ell(s) \geq \ell(v)$ for all $v \in V$. Then since $\ell(v) > \ell(s)$ is always false, `lip`(s) makes no recursive calls, and the algorithm returns 0, as desired.

Now assume that the claim holds for some vertex s as well as all vertices $v \in V$ such that $\ell(v) \geq \ell(s)$.

Take $s' \in V$ to be a vertex that immediately preceeds s in weight order (that is, $\ell(s') \leq \ell(s)$ and there exists no $v \in V$ for which $\ell(s') < \ell(v) < \ell(s)$), and notice that the claim holds for all v such that $(s', v) \in \delta^+(s')$ and $\ell(v) > \ell(s')$ by assumption.

If s' has no out-neighbors with larger weight, then the algorithm correctly returns a maximum increasing path length of 0. Otherwise, the longest path starting at s' has length

$$\max_{v \,:\, (s',v) \,\in\, \delta^+(s'),\ \ell(v) > \ell(s')} \underline{\texttt{lip}(v)}$$

as computed in the algorithm.

By induction, we can conclude that `lip`(s) returns the length of the longest increasing path for all $s \in V$. ■

□

Exercise 12.10.3. Suppose instead we ask for the length of the longest path in G where G is a general graph and the labels of the vertices are weakly increasing.[4] For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

[4] A sequence $x_1, \ldots, x_k$ is weakly increasing if $x_1 \leq x_2 \leq \cdots \leq x_k$.

Solution. We claim that a polynomial time solution for the partition problem would imply a polynomial time solution for SAT. To see this, we present a polynomial time reduction from the longest path problem, which is known to be hard, to the longest weakly increasing path problem.

The approach is simple: given a directed graph G, we label every vertex with the same weight 1 to get a vertex-weighted graph G'. This is clearly a polynomial-time reduction.

Correctness. We claim that `longest-weakly-increasing-path`(G') computes `longest-path`(G).

If we take any path P in G, then the exact same path in G' is weakly increasing, since $1 \leq 1 \leq 1 \leq \ldots$. Conversely, take any weakly-increasing path P' in G'. Since removing vertex weights does not alter reachability, P' is also a path in G.

This correspondence preserves path lengths; hence, the maximum path length `longest-path`(G) is exactly the maximum weakly-increasing path `longest-weakly-increasing-path`(G'). ■

□