Although we typically speak of “the” shortest path between two nodes, a single graph could contain several minimum-length paths with the same endpoints.

Describe and analyze an algorithm to determine the number of shortest paths from a source vertex \( s \) to a target vertex \( t \) in an arbitrary directed graph \( G \) with weighted edges. You may assume that all edge weights are positive and that all necessary arithmetic operations can be performed in \( O(1) \) time.

(Hint: Compute shortest path distances from \( s \) to every other vertex. Throw away all edges that cannot be part of a shortest path from \( s \) to another vertex. What is left?)

Solution:

We start by computing shortest-path distances \( \text{dist}(v) \) from \( s \) to \( v \), for every vertex \( v \), using Dijkstra’s algorithm. Call an edge \( u \to v \) tight if \( \text{dist}(u) + w(u \to v) = \text{dist}(v) \). Every edge in a shortest path from \( s \) to \( t \) must be tight. Conversely, every path from \( s \) to \( t \) that uses only tight edges has total length \( \text{dist}(t) \) and is therefore a shortest path!

Let \( H \) be the subgraph of all tight edges in \( G \). We can easily construct \( H \) in \( O(V + E) \) time. Because all edge weights are positive, \( H \) is a directed acyclic graph. It remains only to count the number of paths from \( s \) to \( t \) in \( H \).

For any vertex \( v \), let \( \text{PathsToT}(v) \) denote the number of paths in \( H \) from \( v \) to \( t \); we need to compute \( \text{PathsToT}(s) \). This function satisfies the following simple recurrence:

\[
\text{PathsToT}(v) = \begin{cases} 
1 & \text{if } v = t \\
\sum_{w \to v} \text{PathsToT}(w) & \text{otherwise}
\end{cases}
\]

In particular, if \( v \) is a sink but \( v \neq t \) (and thus there are no paths from \( v \) to \( t \)), this recurrence correctly gives us \( \text{PathsToT}(v) = \sum \emptyset = 0 \).

We can memoize this function into the graph itself, storing each value \( \text{PathsToT}(v) \) at the corresponding vertex \( v \). Since each subproblem depends only on its successors in \( H \), we can compute \( \text{PathsToT}(v) \) for all vertices \( v \) by considering the vertices in reverse topological order,
or equivalently, by performing a depth-first search of $H$ starting at $s$. The resulting algorithm runs in $O(V + E)$ time.

The overall running time of the algorithm is dominated by Dijkstra’s algorithm in the preprocessing phase, which runs in $O(E \log V)$ time.

*Rubric:* 10 points = 5 points for reduction to counting paths in a dag + 5 points for the path-counting algorithm (standard dynamic programming rubric)