So far we've concerned ourselves with reachability only, caring not for the details of how we get from $s$ to $t$. We now want to think about the shortest path from $s$ to $t$.

**Ex)**

```plaintext
Explore neighbors alphabetically

**DFS Path**

$S A B C T$

**BFS Path**

$S D t$

**Shortest Path**

$S D t$
```

Is it coincidence that BFS found the shortest path?

No! BFS explores all vertices 1 edge from $s$, then all vertices 2 edges from $s$, etc. The shortest path here is the first $S \rightarrow t$ path BFS finds because BFS explores path in edge-distance order. What about with a general nonnegative weighting?

**Ex)**

```plaintext
Explore neighbors alphabetically

**DFS Path**

$S A B C T$

Distance = 21

**BFS Path**

$S D t$

Distance = 1001

**Shortest Path**

$S A B D t$

Distance = 14
```
One way to find the shortest path (on a nonnegative weighting) is to continually find the shortest path from \( s \) to a new vertex.

**Example**

![Graph](image)

\[
\begin{array}{c|c|c|c|c}
\text{Vertex} & \text{Distance from } s & \text{Previous} & \text{Distance from } s & \text{Previous} \\
\hline
S & 0 & \varnothing & 1 & A \\
A & 3 & S & 4 & B \\
B & 1 & C & 7 & F \\
C & 2 & D & 8 & E \\
D & 5 & E & 10 & F \\
E & 1 & F & 2 & \varnothing \\
F & 1 & \varnothing & 2 & \varnothing \\
\end{array}
\]

Dijkstra's Algorithm

**Algorithm**

Let \( D: V \to \mathbb{R}_{\geq 0} \times V \) be a map (use an array/dictionary/hash map for \( \text{dist, prev} \)).

Assign \( D(s) = (0, \varnothing) \)

Let \( Q \) be a priority queue.

For each \( u \in V \)

- If \( u \neq s \)
  - Assign \( D(u) = (\infty, \varnothing) \)
  - Enqueue \( u \) in \( Q \) with priority \( \infty \)

While \( Q \) is not empty

- Dequeue \( u \) from \( Q \) with priority \( p \)

For each neighbor \( v \) of \( u \)

- If \( D(v) \geq D(u) + \omega(u,v) \)
  - Assign \( D(v) = (D(u) + \omega(u,v), u) \)
  - Decrease the priority of \( v \) in \( Q \) to \( D(u) + \omega(u,v) \)

Return \( D \)
The runtime of Dijkstra's depends heavily on how you implement both it and your priority queue. The best known implementation uses a Fibonacci heap to achieve $O(E + V \log V)$, which is the fastest general path-finding algorithm for nonnegative weighted digraphs. Other algorithms are faster for special cases. For example, when all weights are equal, use BFS.

**Example**

Now let's prove Dijkstra is correct.

**Proof**

**W**ith input $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_{\geq 0}$,

Dijkstra$(G, s, w)$ returns a mapping

$D : V \to \mathbb{R}_{\geq 0} \times V$ such that $D(u)$ is

the length of a shortest path from $s$ to $u$ and $D(u)$ is

the previous vertex on a shortest path from $s$ to $u$ (which one

is not unique doesn't matter b/c either path backwards to $s$ is shortest).

We will need two loop invariants to do this.

$\text{LI}_0(D, Q, U) = \forall g \in U, D(g) = (l, h)$, where $l$ is the length of

a shortest path from $s$ to $g$ using only vertices in $Q$ and

$h$ is the vertex before $g$ on a shortest path (or $(\infty, l)$ if no

such path exists).

For $\text{LI}_0$, $U = V$, but we add this parameter to make $\text{LI}_1$

easier to write.
For the inner loop invariant, we're going to define $N$ to be the unprocessed neighbors of $v$ for convenience.

$LII(D, Q, N) := LIO(D, Q \cup \{u\} \setminus N) \land LIO(D, Q, \bar{N})$

(we can use $u$ as an intermediary)

Suppose $LII(D, Q, v)$.

The only thing that can possibly change is $D(v)$.

This occurs precisely when $D(v) > D(u) + W(u, v)$, hence when

* a shortest path using $v$ vertices in $\bar{Q} \cup \{u\}$ is longer than
  a shortest path using $u$ as an intermediary.*

In either case, we assign $D(v)$ to be the (length, predecessor) of a shortest path using intermediary vertices not in $Q$ as desired. Then $N' = N \setminus \{u\}$.

Thus $LII(D', Q, N')$, so $LII$ is a loop invariant of the inner loop.

* There is no shorter path with $u$ as an intermediary but not $v$.

Now suppose $LIO(D, Q, v)$.

We remove $u$ from $Q$ with lowest priority ($D(u) \leq D(v)$, $\forall v \in Q$).

Since $Q' \cup \{u\} \subseteq \bar{Q}$, it cannot gain from being an intermediary.

Since $D(u) \leq D(v)$, $D(u)$ is correct for $u$. Moreover, $u$'s only vertices that can benefit from $u$ as an intermediary currently are those adjacent to $u$. By (*), there is no detour through $Q$ that helps. As such, we know

$LII(D', Q', N(u))$ holds, where $N(u)$ are the neighbors of $u$.

The inner loop must break because $N$ loses a vertex each time and $N(u)$ is finite. When it does, $N = \emptyset$, thus $LIO(D, Q', v)$. 


Lastly, notice that the alg initializes $D, \pi$ so that $\text{LIO}(D, \pi, V)$ holds since $\pi = V$ and the only reachable vertex from $s$ via no intermediaries is $s$ with $D(s) = (0, 1)$.

When the outer loop breaks ($\pi$ finite and loses 1 vert each loop), $\pi$ is empty, so we have $\text{LIO}(D, \emptyset, V)$. But then $\forall u \in V, D(u)$ is the length of a shortest path $s \to u$ using any vertex as an intermediary, hence it is the shortest path $s \to u$ in $G$. Moreover, $D(u)$ is the predecessor on one such shortest paths.

We then return $D$ correctly.

As an aside try coming up with an algorithm for the longest path.

In general, this is an exceedingly hard problem. Unlike shortest path, (where if $u$ lies on the shortest path from $p = s \to t$, then $p = s \to u \to t$ is also the shortest paths $s \to u$ and $u \to t$) longest path does not exhibit this optimal substructure property. There do, however, exist special cases where it does, such as in DAGs. 

$\text{DAGLP}(G, w)$

Let $D : V \to \mathbb{R}_{\geq 0} \times V$ be a map with $D(v) = (\infty, 1)$ everywhere.

Let $L$ be a topological sort of $G$.

For each $u \in L$ (in topological order)

For each $v \in N(u)$

If $D(v) = (D(u) + w(u, v))$

$D(v) = (D(u) + w(u, v), u)$

Return $D$
Since L is a topological sort, we explore every possible path forward in the graph to find the longest paths.

\[ \text{TS: A B C D E} \]

\[
\begin{align*}
D(A) &= (0, A) \\
D(B) &= (2, A) \\
D(C) &= (4, A) \\
D(D) &= (5, B) \\
D(E) &= (11, C)
\end{align*}
\]

Notice that this algorithm also works to find the shortest path in a DAG if we flip the inequality since we explore every possible path forward.

Also, the distance from \( u \) to \( v \) in either case is \( D(v) - D(u) \) when \( v \) is reachable from \( u \).

Lastly, note that the runtime is \( O(V+E) \) in either case.