Dijkstra's Algorithm

One of the best-known and most important algorithms in graph theory/complex networks relates to finding the most efficient (i.e., shortest) path between two vertices.

The problem, for instance, of finding the most efficient route between two cities (i.e., the "Google Maps" problem) is something we have each, inevitably, encountered.

A less explicit (but nonetheless equally ubiquitous) instance of the "shortest path" problem pertains to message routing on the internet. In this way, we can think of vertices as representing computers (or servers), so that neighboring vertices are in some sense "linked" via their ability to send/receive messages. The commonly deployed link-state routing protocol uses a shortest path variant algorithm (i.e., Dijkstra's).

In the exposition that follows, we present a version of Dijkstra's Algorithm (1959) that simultaneously solves the shortest path problem for all vertices in a graph with respect to a "root" vertex.
Formal Algorithm (Dijkstra):

Let $G$ be an undirected, simple, weighted graph. Edge weights are required to be non-negative. Consider a vertex $u$. We introduce the following sets of labels:

1) Let $S_t(u)$ be the set of vertices to which a shortest path from vertex $u$ has been found after step $t$.

2) Each vertex $v$ is assigned a label $L(v) = (L_1(v), L_2(v))$ in which $L_1(v)$ is the vertex preceding $v$ in the shortest $(u,v)$-path found so far, and $L_2(v)$ is the total weight of that path.

3) Let $R_t(u) = S_t(u) \cup N(v)$, with $N(v)$ denoting the neighbor sets of $v$. Put another way, $R_t(u)$ consists of vertices in $S_t(u)$ & their neighbors.

Notation Summary:

$S_t(u)$: set of vertices with known shortest path from $u$.

$L_1(v), L_2(v)$: label for previous vertex in path (left) & path length (right).

$R_t(u)$: set of "known" vertices & their neighbors.
The Algorithm (in 3 steps):

1. **Initialization.**
   - Set $T \leftarrow \emptyset$, $S_0(u) = \{u\}$ and for all $v \in V(G)$,
     $$L(v) \rightarrow \begin{cases} (u, 0) & \text{if } v = u \\ (-\infty) & \text{else} \end{cases}$$

2. For every vertex $y \in R_T(u) \setminus S_T(u)$, consider the vertices $N(y)$ that are neighbors of $y$ that lie in $S_T(u)$, i.e., $N(y) = N(y) \cap S_T(u)$. Select $x \in N(y)$ for which $L_2(x) + w(x, y)$ is minimal. Finally, set $L(y) \rightarrow (x, L_2(x) + w(x, y))$.

3. Let $z \in R_T(u) \setminus S_T(u)$ for which $L_2(z)$ is minimal. Set $S_{T+1}(u) \rightarrow S_T(u) \cup \{z\}$. If $S_{T+1}(u) = V(G)$ we are done. Otherwise, update $T \leftarrow T + 1$, compute $R_T(u)$ & repeat (2).

**Algorithm Summary:**

1. Initialize parameters; set labels $L(v)$ to $\infty$ to all vertices except $u$.

2. Update labels based on minimal neighbors.

3. Update set of known vertices $S_T(u)$, $T$. 
we now apply Dijkstra's algorithm to a directed graph. Following this example, we present a succinct, pseudo-code variant of the algorithm.

Ex. Let \( G \) be the following digraph.

Step 1: Set \( t = 0 \), \( S(v_0) = \{v_0, v_3\} \), label each vertex in \( G \) with appropriate \( L(v) = (L_1(v), L_2(v)) \) label.

Step 2: We consider the vertex closest to the "root" vertex \( v_0 \). This vertex is \( v_3 \). Next we add \( v_3 \) to the set \( S(v_0) = \{v_0, v_3, v_3\} \); note that the label for \( v_3 \) is accordingly updated to: \((3, 1)\) (previous vertex in path \( (1) \)).

Step 3: Continuing, we identify the vertex closest to \( v_0 \) that can be reached from any vertex in \( S(v_0) \). This vertex is \( v_2 \), which is then added to \( S(v_0) = \{v_0, v_3, v_3\} \). Next we update the label for \( v_3 \) to \((0, 3)\).
After 3 steps:

After 4 steps: The next vertex to add is $v_4$: with $S(v_0) = \{v_0\}$ not set to: $\{v_0, v_2, v_3, v_6\}$, it receives label: $(2, 4)$.

Steps: At this juncture the set of "reachable" vertices (neighbor-wise) for $S(v_0) = \{v_0, v_2, v_3, v_6\}$ consists of the set: $\{v_1, v_4, v_6, v_7\}$. The respective minimum distance from $v_0$ for each of these vertices is as follows:

$v_1 = (5 \text{ via } v_2)$, $v_4 = (6 \text{ via } v_0)$, $v_6 = (5 \text{ via } v_3)$, $v_7 = (8 \text{ via } v_3)$. Either minimum reachable vertex will do; we choose $v_1$, making $S(v_3) = \{v_0, v_1, v_2, v_3, v_6\}$ and apply the label: $(2, 5)$.
Step 6: As previously mentioned, vertex $v_6$ is also now a minimal distance from $v_0$, so we add it to the queue, giving: $S(v_0) = \{ v_0, v_1, v_2, v_3, v_4, v_6 \}$. Applying the label: $(3, 5)$.

Step 7: Only two unvisited vertices remain: $v_4, v_7$. The minimum distance to each is: $v_4 = (6 \text{ via } v_3)$, $v_7 = (7 \text{ via } v_0)$, respectively. Consequently, we add $v_4$ to $S(v_0)$ with label: $(0, 6)$.

Step 8: Lastly, we add $v_7$ with label: $(6, 7)$. This completes the algorithm.

Result of Dijkstra's algorithm:

Note that Dijkstra's algorithm results in a spanning tree rooted at $v_0$. We now know the minimum distance between $v_0$ and any other vertex; in addition, this minimum path can be "read" from the Dijkstra encoding.
Pseudo-code version of The Algorithm:

\[ S(u) \rightarrow \emptyset \]
\[ L(v) \rightarrow (v, 0), \text{ else } L(v) \rightarrow (-, \infty) \]

\[ \text{While } S(u) \neq V(G) \text{ do:} \]
\[ R(u) \rightarrow S(u) \cup N(u) \]
\[ \text{for all } y \in R(u) \setminus S(u) \text{ do:} \]
\[ \text{for all } x \in N(y) \cap S(u) \text{ do:} \]
\[ \text{if } L_z(x) + w(<x, y>) < L_z(y) \text{ then:} \]
\[ L(y) \rightarrow (x, L_z(x) + w(<x, y>)) \]
\[ \text{end if} \]
\[ \text{end for} \]
\[ \text{end for} \]
\[ \text{select } v \in S(u) \text{ where } L_z(v) \text{ is minimal} \]
\[ S(u) \rightarrow S(u) \cup \{v\} \]

end while.