My treatment of breadth-first search and Dijkstra's algorithm differs from our textbook's in two ways. First, I structure the two algorithms to emphasize their similarity. Second, I keep track of predecessor nodes. For any node u, the predecessor p is the node immediately before u on the shortest path from s to u. Predecessor nodes are useful for reconstructing the shortest path explicitly, rather than just knowing its length.

1 Breadth-First Search

Input: A graph $G = (V, E)$ and a start node $s \in V$. Output: A list of nodes in G, each tagged with information about the (or a) shortest path from s to the node. Specifically, each node will be presented in a triple $[u, p, d]$, where u is the node, p is the predecessor node (or $None$), and d is the distance from s to u (the number of edges used in the shortest path).

1. Let $frontier = [[s, None, 0]]$ and $known = []$.
2. While $frontier$ is not empty:
   (a) Remove the first item $[u, p, d]$ from the start of $frontier$.
   (b) For each neighbor v of u:
      i. If v is not in $known$ and not in $frontier$, then append $[v, u, d + 1]$ to the end of $frontier$.
   (c) Append $[u, p, d]$ to $known$.
3. Return $known$.

2 Dijkstra's Algorithm

Input: A weighted graph $G = (V, E)$ and a start node $s \in V$. Let $weight(u, v)$ denote the weight of the edge from u to v, if any. Output: A list of nodes in G, each tagged with information about the (or a) shortest path from s to the node. Specifically, each node will be presented in a triple $[u, p, d]$, where u is the node, p is the predecessor node (or $None$), and d is the distance from s to u (the total weight of the edges used in the shortest path).

1. Let $frontier = [[s, None, 0]]$ and $known = []$.
2. While $frontier$ is not empty:
   (a) Remove the triple $[u, p, d]$ from $frontier$ that has the least d.
   (b) For each neighbor v of u:
      i. If v is in $frontier$, then let $[v, q, c]$ be its triple in $frontier$; if $d + weight(u, v) < c$, then update v's triple in $frontier$ to be $[v, u, d + weight(u, v)]$.
      ii. If v is not in $known$ and not in $frontier$, then append $[v, u, d + weight(u, v)]$ to $frontier$.
   (c) Append $[u, p, d]$ to $known$.
3. Return $known$.