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 \( \text{None} \)), and \( d \) is the distance from \( s \) to \( u \) (the number of edges used in the shortest path).

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

2 Dijkstra’s Algorithm

Input: A weighted graph \( G = (V, E) \) and a start node \( s \in V \). Let \( \text{weight}(u, v) \) 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 \( \text{None} \)), and \( d \) is the distance from \( s \) to \( u \) (the total weight of the edges used in the shortest path).

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