```
breadthFirstSearch(startV) {
   create an empty queue
   create an empty discoveredSet

   enqueue startV in queue
   add startV to discoveredSet
   startV.dist = 0

   while queue is not empty {
      curV = dequeue for queue
      curV.visited = true
      for each vertex adjV adjacent to curV {
         if adjV is not in discoveredSet {
            enqueue adjV in queueue
            add adjV to discoveredSet
            adjV.dist = curV.dist + 1
         }
      }
   }
}
```

```
depthFirstSearch(startV) {
   create an empty stack
   create an empty visitedSet

   for all vertices v in the graph {
      v.dist = infinity
      v.visited = false
   }

   push startV on stack
   startV.dist = 0

   while stack is not empty {
      curV = pop from stack
      for all adjacent vertices adjV to curV {
         if (adjV.dist > curV.dist + 1) {
            adjV.dist = curV.dist + 1
         }
      }

      if curV is not in visitedSet {
         curV.visited = true
         add curV to visitedSet
         for each vertex adjV adjacent to curV {
            push adjV to stack
         }
      }
   }
}
```

```
dijkstraShortestPath(startV) {
   create an empty unvisitedQueue

   for each vertex curV in graph {
      curV.dist = infinity
      curV.predecessor = 0
      enqueue curV in unvisitedQueue
   }

   startV.dist = 0

   while unvisitedQueue not empty {
      curV = dequeue from unvisitedQueue vertex with
               minimum dist
      for each adjacent vertex adjV to curV {
         edgeW = weight of edge from curV to adjV
         altDist = curV.dist + edgeW
         if altDist < adjV.dist {
            adjV.dist = altDist
            adjV.predecessor = curV
         }
      }
   }
}
```

```
topologicalSort(graph) {
   for all vertices v in graph {
      v.incoming = number of incoming edges
   }
   resultList = empty list of vertices
   noIncoming = list of all vertices with no incoming edges
   edgesList = list of all edges in the graph

   while noIncoming is not empty {
      curV = remove any vertex from noIncoming
      add curV to resultList
      for all outgoing edges outE from curV {
         remove outE from edgesList
         destV = vertex on on other end of outE
         destV.incoming = destV.incoming - 1
         if (destV.incoming == 0) {
            add destV to noIncoming
         }
      }
   }
   return resultList
}
```