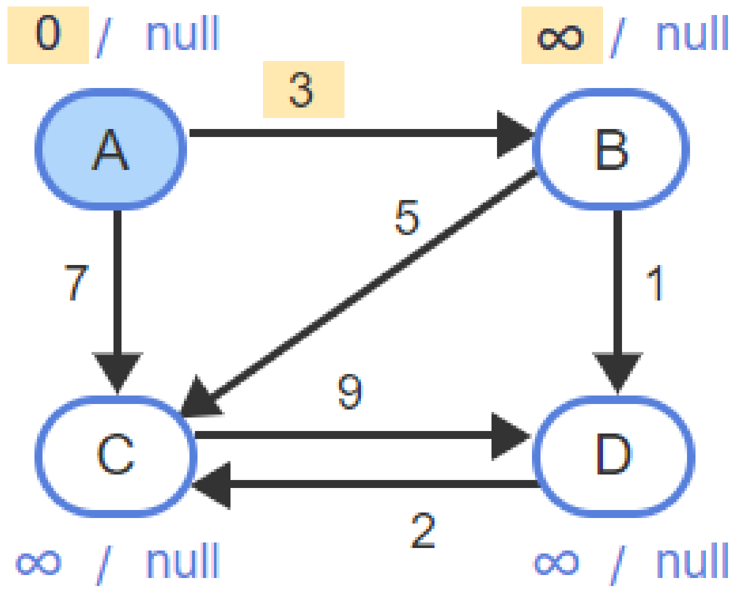
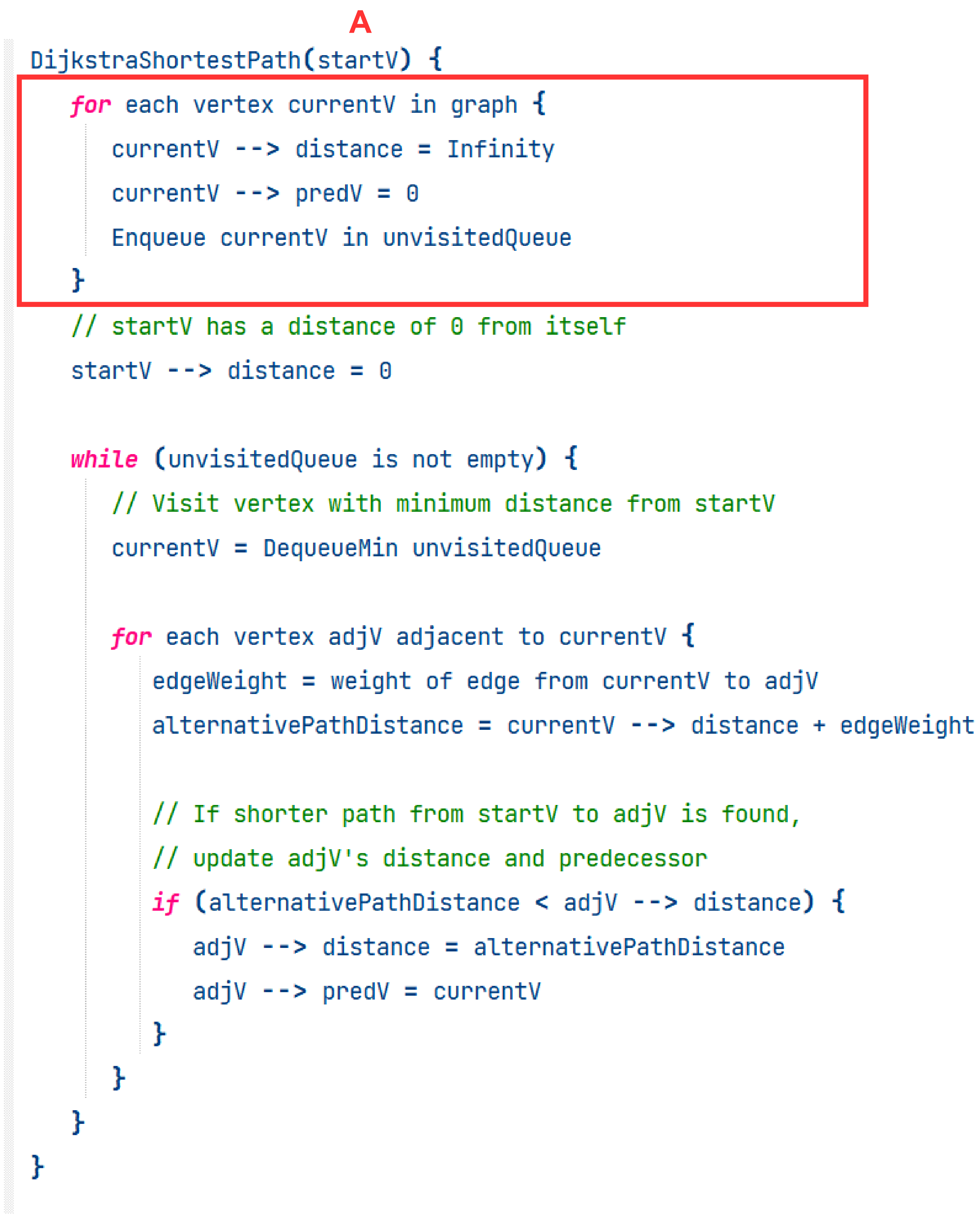
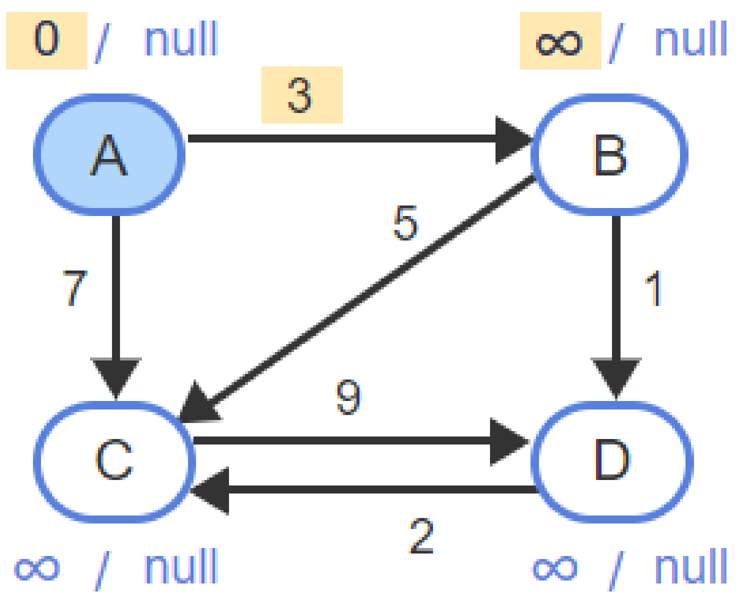


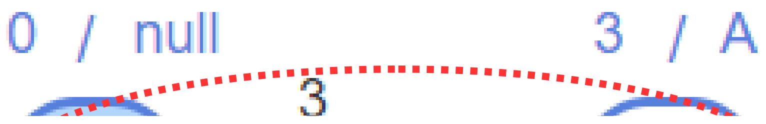
currentV	
unvisitedQueue	A, B, C, D

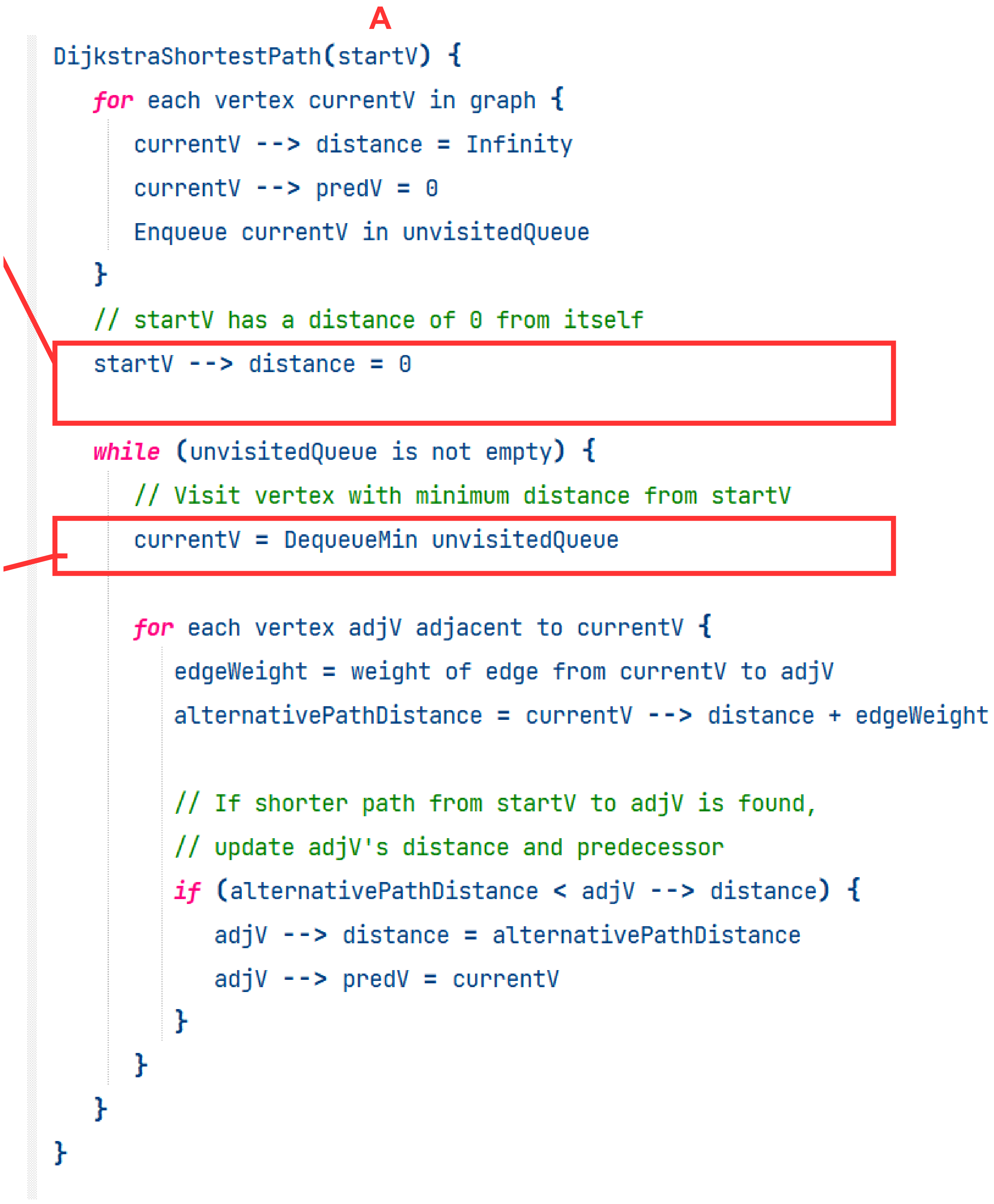


```
DijkstraShortestPath(startV) {
   for each vertex currentV in graph {
      currentV --> distance = Infinity
      currentV --> predV = 0
      Enqueue currentV in unvisitedQueue
   }
   // startV has a distance of 0 from itself
   startV --> distance = 0

   while (unvisitedQueue is not empty) {
      // Visit vertex with minimum distance from startV
      currentV = DequeueMin unvisitedQueue

      for each vertex adjV adjacent to currentV {
         edgeWeight = weight of edge from currentV to adjV
         alternativePathDistance = currentV --> distance + edgeWeight

         // If shorter path from startV to adjV is found,
         // update adjV's distance and predecessor
         if (alternativePathDistance < adjV --> distance) {
            adjV --> distance = alternativePathDistance
            adjV --> predV = currentV
         }
      }
   }
}
```

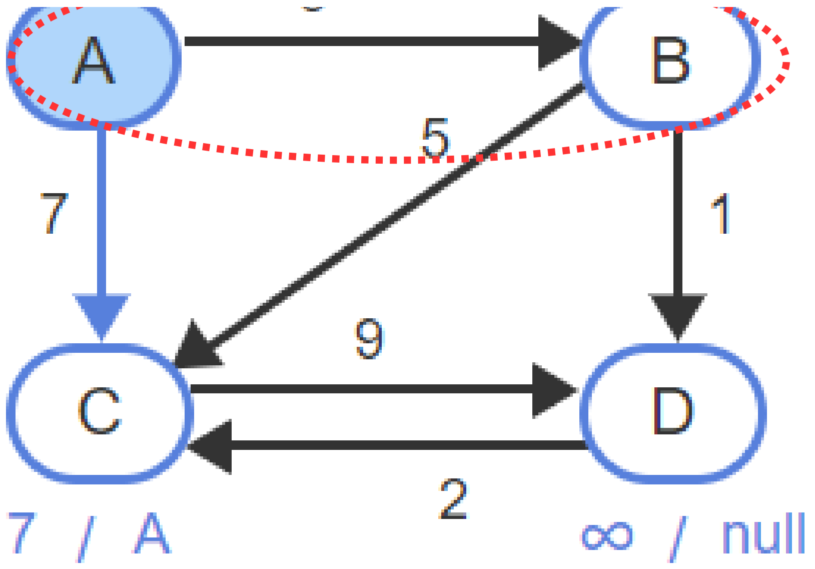


currentV	A
unvisitedQueue	B, C, D





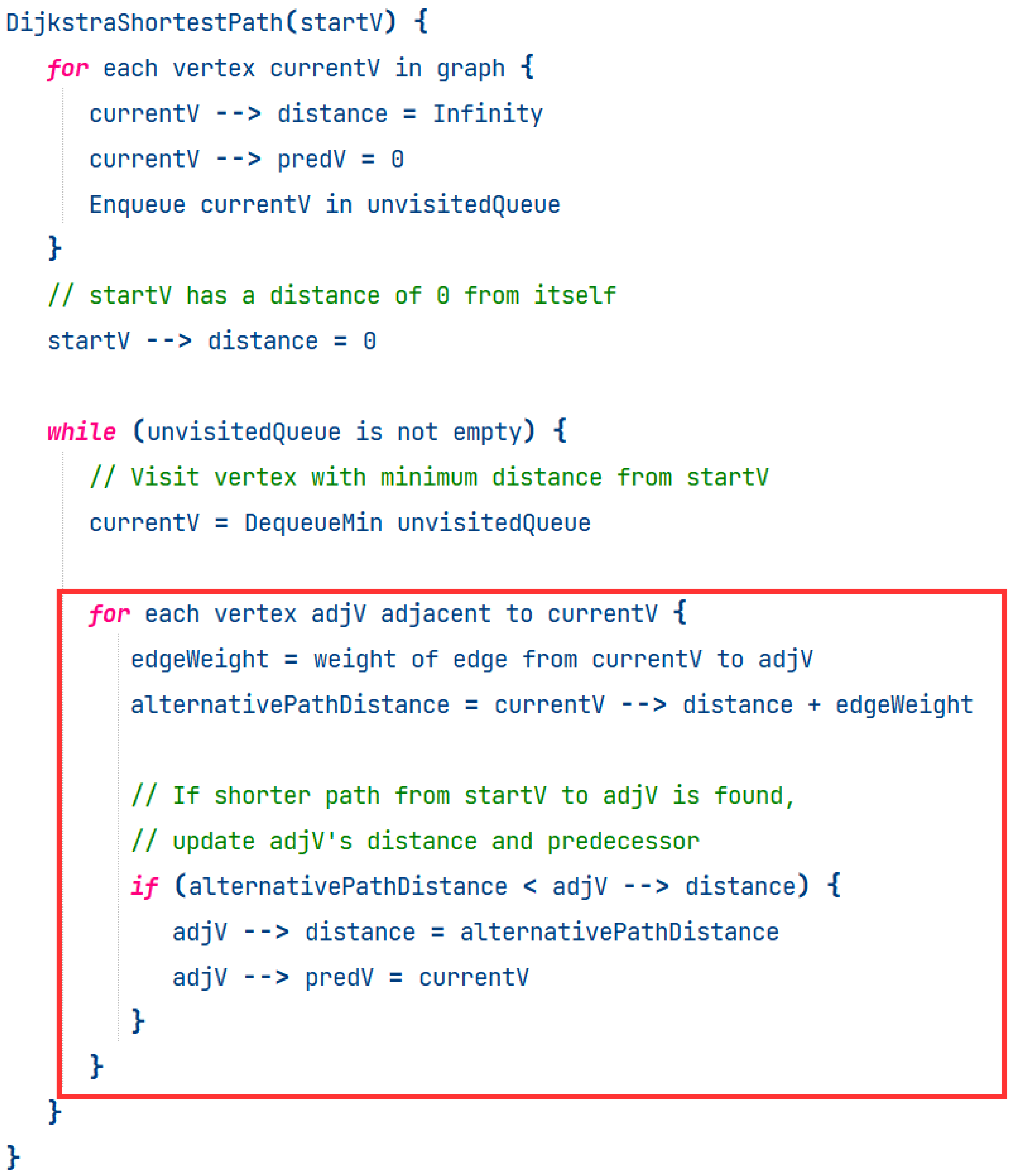
```
DijkstraShortestPath(startV) {
   for each vertex currentV in graph {
      currentV --> distance = Infinity
      currentV --> predV = 0
      Enqueue currentV in unvisitedQueue
   }
   // startV has a distance of 0 from itself
   startV --> distance = 0

   while (unvisitedQueue is not empty) {
      // Visit vertex with minimum distance from startV
      currentV = DequeueMin unvisitedQueue

      for each vertex adjV adjacent to currentV {
         edgeWeight = weight of edge from currentV to adjV
         alternativePathDistance = currentV --> distance + edgeWeight

         // If shorter path from startV to adjV is found,
         // update adjV's distance and predecessor
         if (alternativePathDistance < adjV --> distance) {
            adjV --> distance = alternativePathDistance
            adjV --> predV = currentV
         }
      }
   }
}
```



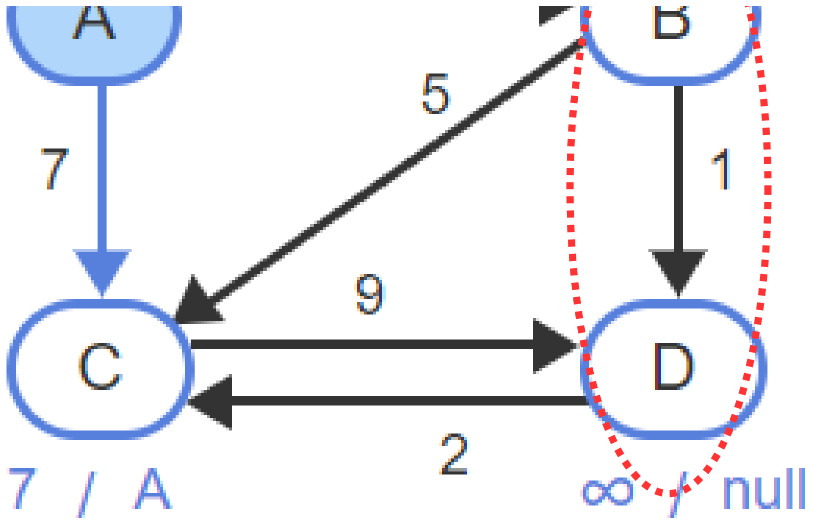
currentV	A
unvisitedQueue	B, C, D

A --> B = 3
A --> C = 7

A



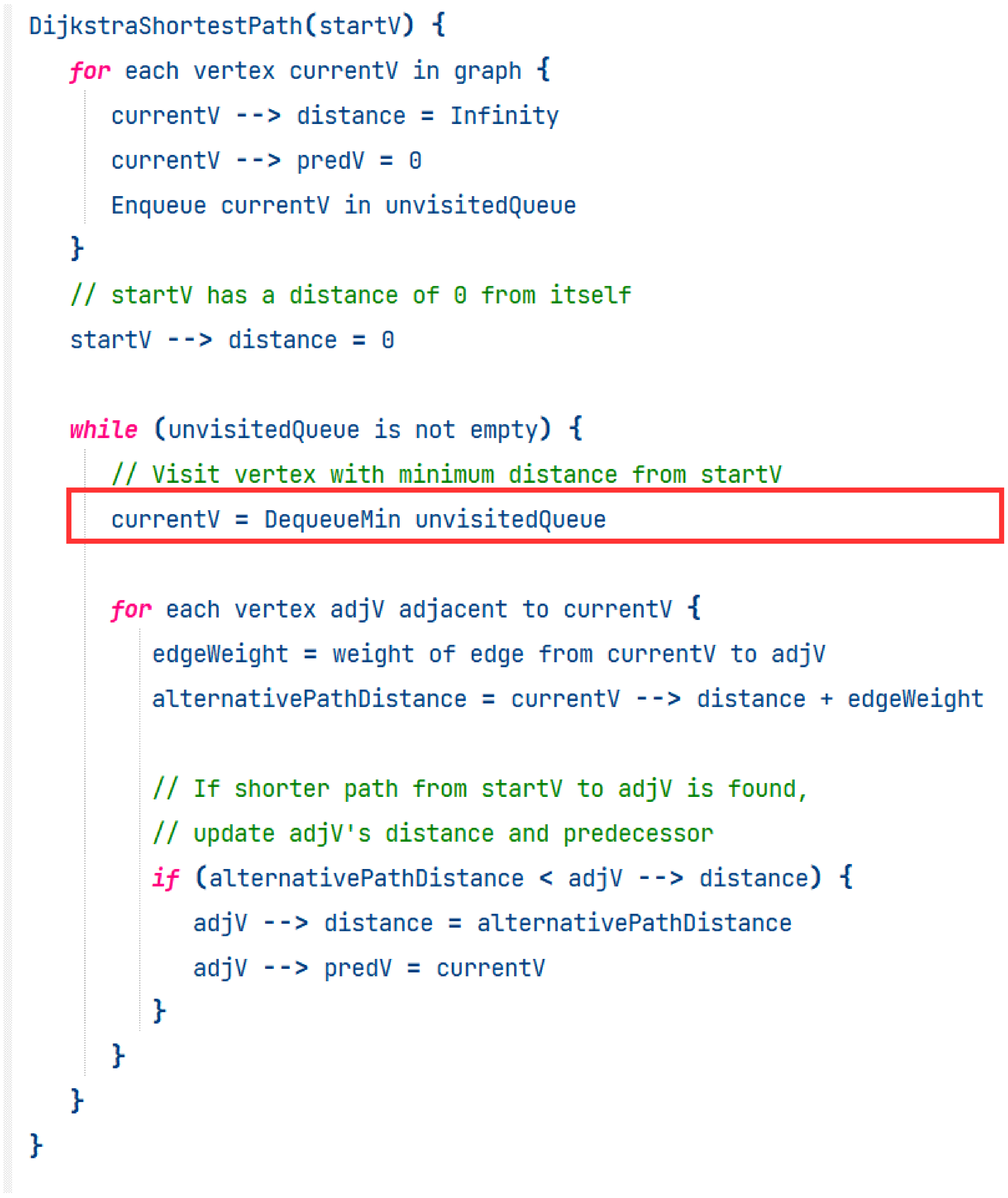
```
DijkstraShortestPath(startV) {
   for each vertex currentV in graph {
      currentV --> distance = Infinity
      currentV --> predV = 0
      Enqueue currentV in unvisitedQueue
   }
   // startV has a distance of 0 from itself
   startV --> distance = 0

   while (unvisitedQueue is not empty) {
      // Visit vertex with minimum distance from startV
      currentV = DequeueMin unvisitedQueue

      for each vertex adjV adjacent to currentV {
         edgeWeight = weight of edge from currentV to adjV
         alternativePathDistance = currentV --> distance + edgeWeight

         // If shorter path from startV to adjV is found,
         // update adjV's distance and predecessor
         if (alternativePathDistance < adjV --> distance) {
            adjV --> distance = alternativePathDistance
            adjV --> predV = currentV
         }
      }
   }
}
```



currentV	B
unvisitedQueue	C, D

B --> C = 5
B --> D = 1

A



```
DijkstraShortestPath(startV) {
   for each vertex currentV in graph {
      currentV --> distance = Infinity
      currentV --> predV = 0
      Enqueue currentV in unvisitedQueue
   }
   // startV has a distance of 0 from itself
   startV --> distance = 0

   while (unvisitedQueue is not empty) {
      // Visit vertex with minimum distance from startV
      currentV = DequeueMin unvisitedQueue

      for each vertex adjV adjacent to currentV {
         edgeWeight = weight of edge from currentV to adjV
         alternativePathDistance = currentV --> distance + edgeWeight

         // If shorter path from startV to adjV is found,
         // update adjV's distance and predecessor
         if (alternativePathDistance < adjV --> distance) {
            adjV --> distance = alternativePathDistance
            adjV --> predV = currentV
         }
      }
   }
}
```