**Dijkstra(S, G):**

**Initialize: PriorityQueue (PQ), visited HashSet, parent   
 HashMap, and distances to infinity**

**Add (S, 0) onto the PriorityQueue**

**while PQ is not empty:**

**dequeue node curr from front of queue**

**if(curr is not visited)**

**add curr to the visited set**

**if(curr == G) return parent map**

**for each of curr’s neighbors, n, not in visited set:**

**if(path through curr to n is shorter)**

**update n’s distance**

**update curr as n’s parent in parent map**

**enqueue (n, distance) onto the PQ**

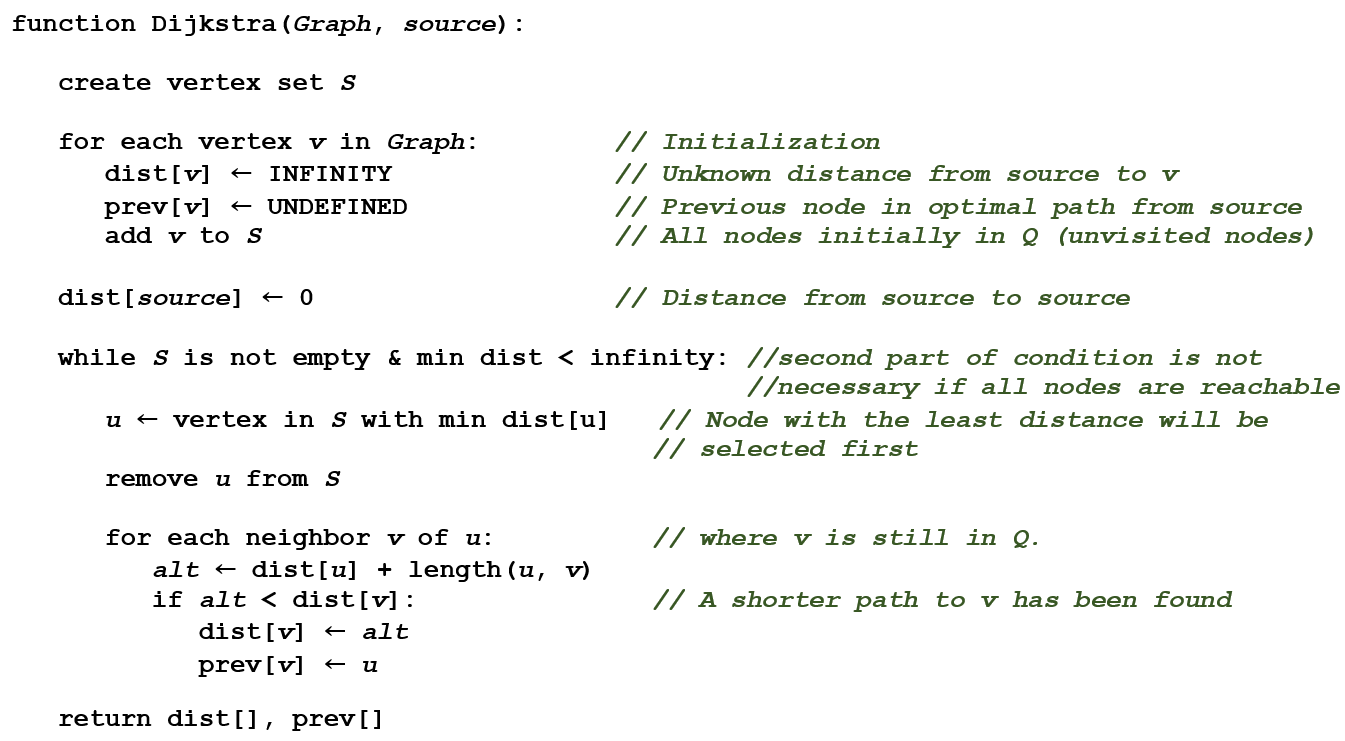
**// if we made it here then there’s no path**

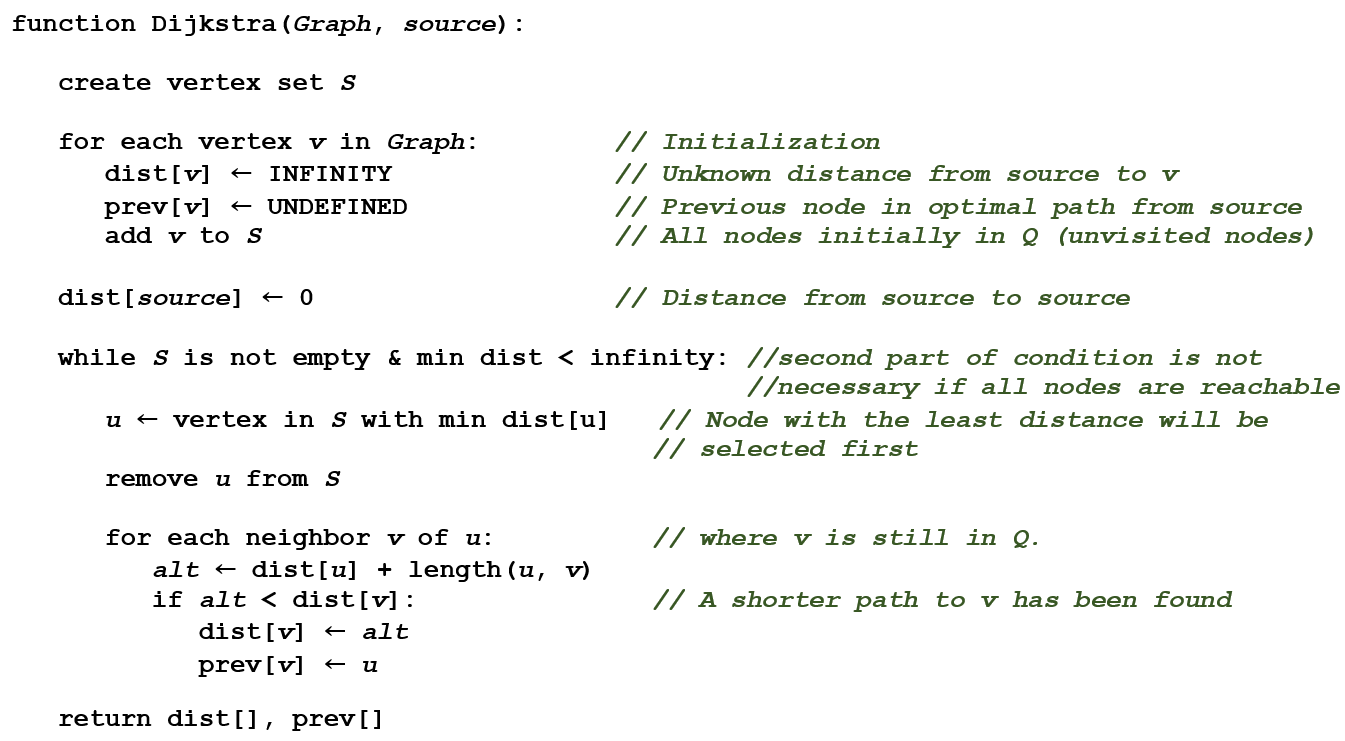
**Dikstra’s Algorithm – finding the shortest path**

Let the node at which we are starting be called the **initial node**. Let the **distance of node *Y*** be the distance from the **initial node** to *Y*. Dijkstra’s algorithm will assign some initial distance values and will try to improve them step by step.

* 1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

1. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
3. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
4. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
5. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new “current node”, and go back to step





**function Dijkstra(*Graph*, *source*):**

**create vertex set *S***

**for each vertex *v* in *Graph*: *// Initialization***

**dist[*v*] ← INFINITY *// Unknown distance from source to v***

**prev[*v*] ← UNDEFINED *// Previous node in optimal path from source***

**add *v* to *S* *// All nodes initially in Q (unvisited nodes)***

**dist[*source*] ← 0 *// Distance from source to source***

**while *S* is not empty & min dist < infinity: *//second part of condition is not  
 //necessary if all nodes are reachable***

***u* ← vertex in *S* with min dist[u] *// Node with the least distance will be   
 // selected first***

**remove *u* from *S***

**for each neighbor *v* of *u*: *// where v is still in Q.***

***alt* ← dist[*u*] + length(*u*, *v*)**

**if *alt* < dist[*v*]: *// A shorter path to v has been found***

**dist[*v*] ← *alt***

**prev[*v*] ← *u***

**return dist[], prev[]**