**Dijkstra’s Algorithm**

void dijkstra(Graph g, Vertex start, Vertex finish)

{

Q = new data structure to hold vertices (can be list, set, …)

for each vertex v in the graph:

dist[v] = infinity

prev[v] = undefined

dist[start] = 0  
Q.add(start)

while Q is not empty:

u = vertex in Q with minimum dist[u] value // we now “visit” u

remove u from Q

if u == finish: break

for each neighbor v of u still in Q: // all the nodes "v" we can go to from "u"

alt = dist[u] + weight(u, v)

if alt < dist[v]

dist[v] = alt

prev[v] = u

Final path length is dist[finish].

Traverse prev[] array starting from prev[finish] in reverse order back

to start vertex to get final path from start to finish.

}

Note: during the for each neighbor v of u step, the algorithm will reconsider nodes it has already visited before (thereby opening the possibility of a cycle). However, for a situation like this, dist[u] + weight(u, v) will always be bigger than dist[v], so the cycles will be ignored anyway. However, some Dijktra’s Algorithm implementations explicitly keep track of which vertices have been visited already and modify the for each neighbor v of u step to skip over any vertex v that has already been visited earlier.