**Dijkstra’s Algorithm**

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

{

create min-priority queue PQ

for each vertex v in the graph:

dist[v] = infinity

prev[v] = undefined

dist[start] = 0  
PQ.insert(start, 0)

while PQ is not empty:

u = PQ.extract\_minimum() # We now “visit” vertex u.

if u == finish: break

for each neighbor v of u: # 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

if PQ.contains(v)

PQ.change\_priority(v, alt)

else

PQ.insert(v, alt)

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.