Pseudocode

```
function Dijkstra(Graph, source):
    create vertex set Q
>>  for each vertex v in Graph: // Initialization
        dist[v] ← INFINITY   // Unknown dist from source to v
        prev[v] ← UNDEFINED // Prev. node in optimal path from source
        add v to Q  // ALL nodes initially in Q (unvisited nodes)
    dist[source] ← 0  // Dist from source to source
>>  while Q is not empty:
        u ← vertex in Q w/ min dist[u] // node w/ least dist
        remove u from Q                // will be selected first
>>> 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
                dist[v] ← alt   // been found
                prev[v] ← u
>>  return dist[], prev[]
```

min priority queue (heap) - ADT w/ 3 operations
- add_with_priority()
- decrease_prio()
- extract_min()

USING A PRIORITY QUEUE

```
function Dijkstra(Graph, source):
>> dist[source] ← 0   // init
>> create vertex set Q
>> for each vertex v in Graph:
>>>> if v ≠ source
>>>>> dist[v] ← INFINITY // unknown dist from source to v
>>>>> prev[v] ← UNDEFINED // predecessor of v
>>>> Q.add_with_priority(v, dist[v])
>> while Q is not empty: // main loop
>>>> u ← Q.extract_min() // remove & return best vertex
>>>> for each neighbor v of u: // only v that is still in Q
>>>>>> alt ← dist[u] + length(u, v)
>>>>>> if alt < dist[v]
>>>>>>> dist[v] ← alt
>>>>>>> prev[v] ← u
>>>>>>> Q.decrease_priority(v, alt)
>> return dist[], prev[]
```