

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

add v to Q ~~// dist from source to source~~

// ALL nodes initially in Q (unvisited nodes)

~~dist[Q] is not empty~~

dist[source] ← 0 // Dist from source to source

>> while Q is not empty:

u ← vertex in Q w/ min dist[u] // node u / 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[]

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[]

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

- add_with_priority()
- decrease_priority()
- extract_min()

12:50
13:00
13:10
13:20
13:30
13:40
13:50
14:00
14:10
14:20
14:30
14:40
14:50
15:00
15:10
15:20
15:30
15:40
15:50
16:00
16:10
16:20
16:30
16:40
16:50
17:00
17:10
17:20
17:30
17:40
17:50
18:00
18:10
18:20
18:30
18:40
18:50
19:00
19:10
19:20
19:30
19:40
19:50
20:00
20:10
20:20
20:30
20:40
20:50
21:00
21:10
21:20
21:30
21:40
21:50
22:00
22:10
22:20
22:30
22:40
22:50
23:00
23:10
23:20
23:30
23:40
23:50
24:00