Inhaltsverzeichnis

* 1 Edsger W. Dijkstra (ca. ½ Seite)
* 2 Graphen (ca. 2 Seiten)
  + 2.1 Undirected
  + 2.2 Directed
  + 2.3 Weighted
* 3 Beschreibung des Dijktstra-Algorithmus (ca. 2 ½ Seiten)
  + 3.1 Problemstellung
  + 3.2 Lazy
  + 3.3 Eager
* 4 Erweiterung des Dijkstra-Algorithmus (ca. 1 Seite)
  + 4.1 A\*-Search
  + 4.2 Weitere
* 5 Anwendungsmöglichkeiten (ca. 2 Seiten)
  + 5.1 Navigation
  + 5.2 Routing

Dijkstra Lazy

Pseudocode:

function dijkstra(graph, n, s, e):

visited = [false, false, …, false) # size n

prev = [null, null, …, null] # size n

distance = [∞, ∞, …, ∞, ∞] # size n

distance[s] = 0

pq = empty priority queue

pq.insert((s, 0))

while pq.size() != 0:

index, minValue = pq.poll()

visited[index] = true

for (edge: g[index]):

if visited[edge.to]: continue

newDistance = distance[index] + edge.cost

if newDistance < distance[edge.to]:

prev[edge.to] = index

distance[edge.to] = newDistance

pq.insert((edge.to, newDistance))

if index == e:

return distance[e]

return distance

Dijkstra Eager

Pseudocode:

Function Dijkstra(g, n, s):

Visited = [false, false, …, false] # size n

prev = [null, null, …, null] # size n

distance = [∞, ∞, …, ∞, ∞] # size n

distance[s] = 0

ipq = empty index priority queue

ipq.insert((s, 0))

while pq.size() != 0:

index, minValue = pq.poll()

visited[index] = true

for (edge: g[index]):

if visited[edge.to]: continue

newDistance = distance[index] + edge.cost

if newDistance < distance[edge.to]:

prev[edge.to] = index

distance[edge.to] = newDistance

if !ipq.contains(edge.to):

ipq,insert(edge.to, newDistance)

else:

ipq.descreaseKey(edge.to, newDistance) # Prevent duplicate node indexes to be present

pq.insert((edge.to, newDistance))

if index == e:

return distance[e]

return distance

A\* Search Algorithm

Pseudocode:

make an openlist containing only the starting node

make an empty closed list

while (the destination node has not been reached):

consider the node with the lowest f score in the open list

if (this node is our destination node) :

we are finished

if not:

put the current node in the closed list and look at all of its neighbors

for (each neighbor of the current node):

if (neighbor has lower g value than current and is in the closed list) :

replace the neighbor with the new, lower, g value

current node is now the neighbor's parent

else if (current g value is lower and this neighbor is in the open list ) :

replace the neighbor with the new, lower, g value

change the neighbor's parent to our current node

else if this neighbor is not in both lists:

add it to the open list and set its g