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, start, destination):

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

prev = [null, null, …, null] # size n

distance = [∞, ∞, …, ∞, ∞] # size n

distance[start] = 0

pq = empty priority queue

pq.insert((start, 0))

while pq.size() != 0:

index, min\_value = pq.poll()

visited[index] = true

for edge in g[index]:

if visited[edge.to]: continue

new\_distance = distance[index] + edge.cost

if new\_distance < distance[edge.to]:

prev[edge.to] = index

distance[edge.to] = new\_distance

pq.insert((edge.to, new\_distance))

if index == destination:

return distance[destination]

return ∞

Dijkstra Eager

Pseudocode:

function dijkstra(g, n, start, destination):

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

prev = [null, null, …, null] # size n

distance = [∞, ∞, …, ∞, ∞] # size n

distance[start] = 0

ipq = empty index priority queue

ipq.insert((start, 0))

while pq.size() != 0:

index, min\_value = pq.poll()

visited[index] = true

for edge in g[index]:

if visited[edge.to]: continue

new\_distance = distance[index] + edge.cost

if new\_distance < distance[edge.to]:

prev[edge.to] = index

distance[edge.to] = new\_distance

if edge.to not in ipq:

ipq.insert(edge.to, new\_distance)

else:

ipq.descreaseKey(edge.to, new\_distance)

if index == destination:

return distance[destination]

return ∞

A\* Search Algorithm

Pseudocode:

function a\_star\_search(graph, n, start, end):

open\_set = {} # size n

closed = {} # size n

distance = [∞, ∞, …, ∞, ∞] # size n

distance[start] = 0

pq = empty priority queue

pq.insert((start, 0))

while pq.size() != 0:

index, min\_value = pq.poll()

visited[index] = true

for edge in g[index]:

if visited[edge.to]: continue

new\_distance = distance[index] + edge.cost

if new\_distance < distance[edge.to]:

prev[edge.to] = index

distance[edge.to] = new\_distance

pq.insert((edge.to, new\_distance))

if index == e:

return distance[e]

return ∞