



test

Max Mustermann

Fachhochschule Trier

19. Juli 2015

Problemstellung

test

- ▶ Bilder: Flugzeug, Bus- / Zugfahrplan



Table of content

eins

Komplexität

Implementierung

eins



HOCHSCHULE **TRIER**
Trier University of Applied Sciences
Informatik - Computer Science

test

Komplexität

Ursprüngliche Implementierung

Ursprüngliche Implementierung

- ▶ Einzelschritte:

Ursprüngliche Implementierung

- ▶ Einzelschritte:
 - ▶ Initialisieren der Arrays je $O(m)$

Ursprüngliche Implementierung

- ▶ Einzelschritte:
 - ▶ Initialisieren der Arrays je $O(m)$
 - ▶ Abarbeiten der grauen Knoten $O(m)$

Ursprüngliche Implementierung

- ▶ Einzelschritte:
 - ▶ Initialisieren der Arrays je $O(m)$
 - ▶ Abarbeiten der grauen Knoten $O(m)$
 - ▶ Bestimmen des Minimums $O(m)$

Ursprüngliche Implementierung

- ▶ Einzelschritte:
 - ▶ Initialisieren der Arrays je $O(m)$
 - ▶ Abarbeiten der grauen Knoten $O(m)$
 - ▶ Bestimmen des Minimums $O(m)$
 - ▶ Aktualisieren der Nachfolger $O(\deg(v)) \rightarrow O(k)$

Ursprüngliche Implementierung

- ▶ Einzelschritte:
 - ▶ Initialisieren der Arrays je $O(m)$
 - ▶ Abarbeiten der grauen Knoten $O(m)$
 - ▶ Bestimmen des Minimums $O(m)$
 - ▶ Aktualisieren der Nachfolger $O(\deg(v)) \rightarrow O(k)$
- ▶ insgesamt Komplexität $O(m^2)$

Implementierung mit Heap

- ▶ Vorteile:

Implementierung mit Heap

- ▶ Vorteile:
 - ▶ Heapoperationen in $O(\log m)$

Implementierung mit Heap

- ▶ Vorteile:
 - ▶ Heapoperationen in $O(\log m)$
 - ▶ Bestimmen des Minimums in $O(\log m)$

Implementierung mit Heap

- ▶ Vorteile:
 - ▶ Heapoperationen in $O(\log m)$
 - ▶ Bestimmen des Minimums in $O(\log m)$
- ▶ insgesamt Komplexität $O(k * \log m)$

Eigenschaften

Eigenschaften

- ▶ Programmiersprache: Python

Eigenschaften

- ▶ Programmiersprache: Python
- ▶ Umsetzung mit Heap (Priority Queue)

Eigenschaften

- ▶ Programmiersprache: Python
- ▶ Umsetzung mit Heap (Priority Queue)
- ▶ keine Speicherung der Farbstufen wie bei Dijkstra

Eigenschaften

- ▶ Programmiersprache: Python
- ▶ Umsetzung mit Heap (Priority Queue)
- ▶ keine Speicherung der Farbstufen wie bei Dijkstra
 - ▶ kürzerer und übersichtlicherer Code

Implementierung

Kompletter Code

```

1 from heapq import heappush, heappop
2
3 def dijkstra_pq(G,s):
4     m = len(G) #priority queue #O(1)
5     pq = [] #O(1)
6     d = [None]*m #kosten #O(m)
7     p = [None]*m #vorgaenger #O(m)
8     d[s] = 0 #O(1)
9     heappush(pq, (0,s)) #O(log(m))
10    while pq: #O(m)
11        (_,v) = heappop(pq) #O(log m)
12        for u in G[v]: #O(deg(v)) -> O(k)
13            alt = d[v] + G[v][u] #O(1)
14            if d[u]== None or alt < d[u]: #O(1)
15                d[u] = alt #O(1)
16                p[u] = v #O(1)
17                heappush(pq, (alt,u)) #O(log m)
18    return d,p
19
20 def shortest_path(s,v,p):
21     if v == None:
22         return []
23     else:
24         return shortest_path(s,p[v],p) + [v]
25
26 #Graph:
27
28 def define_G():
29     G = [ {1:1, 4:4, 2:2}, # Nachfolger von s
30           {3:3, 4:1}, # von u
31           {4:2, 5:3}, # von x
32           {}, # von y
33           {3:1, 5:2}, # von v
34           {} # von z
35         ]
36     return G
37
38
39 G = define_G()
40 d,p = dijkstra_pq(G,0)
41 print( shortest_path(0,5,p))
42 print( shortest_path(0,3,p))

```

Implementierung

Eingabe

- ▶ leer

Algorithmus

- ▶ Initialisierung

Algorithmus

- ▶ Initialisierung
- ▶ Erweitern

Algorithmus

- ▶ Initialisierung
- ▶ Erweitern
- ▶ Aktualisieren

Rekursives Bestimmen des Pfades

- ▶ leer

Implementierung

Aufruf

- ▶ leer