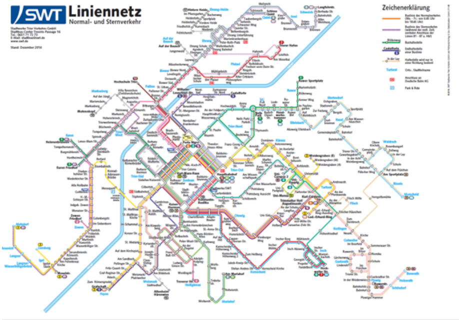
test

Max Mustermann

Fachhochschule Trier

20. Juli 2015

Problemstellung



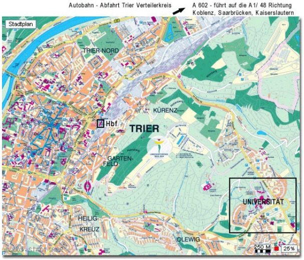
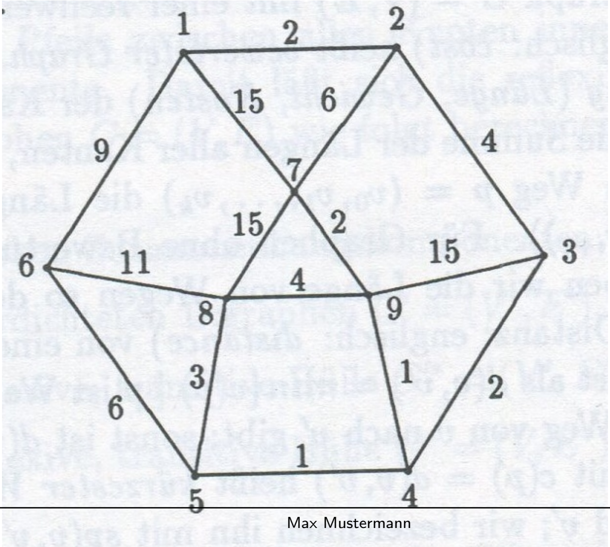


Table of content

Definition

Ein **Graph** G besteht aus einer Menge X [deren Elemente Knotenpunkte genannt werden] und einer Menge U, wobei jedem Element u $\in$ U in eindeutiger Weise ein geordnetes oder ungeordnetes Paar von [nicht notwendig verschiedenen] Knotenpunkten, x,y $\in$ X zugeordnet ist. Ist jedem u $\in$ U ein geordnetes Paar von Knoten zugeordnet, so heißt der Graph **gerichtet**, und wir schreiben $G = (X, U)$. Die Elemente von U werden in diesem Fall als **Bögen** bezeichnet.
Ist jedem u $\in$ U ein ungeordnetes Paar von Knotenpunkten zugeordnet, so heißt der Graph **ungerichtet** und wir schreiben $G = [X, U]$. Die Elemente von U bezeichnen wir dann als **Kanten.**

Beispiel



Dijkstra

HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

white node

Ein “white node” ist ein Knoten über den noch nichts bekannt ist.

Dijkstra

white node

Ein “white node” ist ein Knoten über den noch nichts bekannt ist.

grey node

Ein “grey node” ist ein Knoten zu dem bereits ein Weg gefunden wurde. Ob dies der ideale Weg ist kann man allerdings noch nicht sagen.

Dijkstra

white node
Ein “white node” ist ein Knoten über den noch nichts bekannt ist.

grey node
Ein “grey node” ist ein Knoten zu dem bereits ein Weg gefunden wurde. Ob dies der ideale Weg ist kann man allerdings noch nicht sagen.

black node
Ein “black node” ist ein Knoten zu dem bereits der optimale Weg gefunden wurde.

Berechnung von Knotenwerten

Der Knotenwert k_A eines Knotens K_A kann berechnet werden, wenn es einen Knoten K_V gibt, dessen Knotenwert k_V bekannt ist und eine gerichtete Kante von K_V nach K_A mit bekannten Kosten p existiert.

Berechnung: $k_A = k_V + p$

Anfang

Der Startknoten wird als “black node” mit dem Wert 0 definiert. Alle seine Nachfolgerknotenwerte werden bestimmt. Diese werden zu “grey node”

Schritt

Der “grey node”, welcher den niedrigsten Wert besitzt, wird zu einem “black nodes”. Bei allen seinen Nachfolgern wird deren Wert von dem aktuell ausgewählten “grey node” neu berechnet. Wenn der Wert kleiner als der Wert des Knoten ist, so wird dieser Wert überschrieben. Knoten, welche das erste mal einen Wert zugeordnet bekommen, sind jetzt “grey nodes”.
relaxieren benennen.

Dijkstra

Ende

Wenn der Zielknoten ein “black node” wird, hat man den idealen Weg vom Startknoten und Endknoten gefunden.

Komplexität

Komplexität

Ursprüngliche Implementierung

Komplexität

Ursprüngliche Implementierung

- Einzelschritte:

Ursprüngliche Implementierung

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

Komplexität

Ursprüngliche Implementierung

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

Komplexität

Ursprüngliche Implementierung

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

Komplexität

Ursprüngliche Implementierung

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

Komplexität

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 $O(m^2)$

Komplexität

Implementierung mit Heap

- Vorteile:

Komplexität

Implementierung mit Heap

- Vorteile:
  - Heapoperationen in $O(logm)$

Komplexität

Implementierung mit Heap

- Vorteile:
  - Heapoperationen in $O(logm)$
  - Bestimmen des Minimums in $O(logm)$

Komplexität

Implementierung mit Heap

- Vorteile:
  - Heapoperationen in $O(logm)$
  - Bestimmen des Minimums in $O(logm)$
- insgesamt Komplexität $O(k * logm)$

Implementierung

Eigenschaften

Implementierung

HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

Eigenschaften

- Programmiersprache: Python

Implementierung

Eigenschaften

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

Implementierung

Eigenschaften

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

Implementierung

Eigenschaften

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

Kompletter Code

```
from heapq import heappush, heappop

def dijkstra_pq(G,s):
    m = len(G)                                      #O(1)
    pq = []         #priority queue                 #O(1)
    d = [None]*m    #kosten                         #O(m)
    p = [None]*m    #vorgaenger                     #O(m)
    d[s] = 0                                        #O(1)
    heappush(pq, (0,s))                             #O(log(m))
    while pq:                                       #O(m)
        (_,v) = heappop(pq)                         #O(log m)
        for u in G[v]:                              #O(deg(v)) --> O(k)
            alt = d[v] + G[v][u]                    #O(1)
            if d[u]== None or alt < d[u]:           #O(1)
                d[u] = alt                          #O(1)
                p[u] = v                            #O(1)
                heappush(pq, (alt,u))               #O(log m)
    return d,p

def shortest_path(s,v,p):
    if v == None:
        return []
    else:
        return shortest_path(s,p[v],p) + [v]

#Graph:

def define_G():
    G = [   {1:1, 4:4, 2:2},    # Nachfolger von s
            {3:3, 4:1},         # von u
            {4:2, 5:3},         # von x
            {},                 # von y
            {3:1, 5:2},         # von v
            {}                  # von z
        ]
    return G


G = define_G()
d,p = dijkstra_pq(G,0)
print( shortest_path(0,5,p))
print( shortest_path(0,3,p))
```

Eingabe

```
#Graph:

def define_G():
    G = [     {1:1, 4:4, 2:2},    # Nachfolger von s
              {3:3, 4:1},         # von u
              {4:2, 5:3},         # von x
              {},                 # von y
              {3:1, 5:2},         # von v
              {}                  # von z
        ]
    return G
```

Implementierung

Algorithmus

- Initialisierung

Implementierung

Algorithmus

- Initialisierung
- Erweitern

Implementierung

Algorithmus

- Initialisierung
- Erweitern
- Aktualisieren

Implementierung

Rekursives Bestimmen des Pfades

- leer

Implementierung

HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

Aufruf

- leer