Shortest Path Algorithmus von Edsger Dijkstra

Michael Dienert

4. April 2017

Inhaltsverzeichnis

1 Shortest Path Algorithmus

1.1 Graphen

Der Shortest Path Algorithmus von Edsger Dijkstra sucht in einem *gewichteten Graphen* (= Netz, bestehend aus Knoten und Verbindungen mit Kosten) nach den günstigsten Pfaden von einem Startknoten zu allen anderen Knoten.

Das Beispielnetz in diesem Dokument sieht so aus:

Es enthält die vier Knoten **a**,**b**,**c**,**d**. Die Ziffern stellen die **Kosten** der Verbindung dar. In der Netzwerktechnik werden die Kosten aus der Bandbreite brechnet, verwendet man den Dijkstra-Algorithmus um Wegstrecken zu opimieren, entsprechen die Kosten der Entfernung zwischen den Knoten.

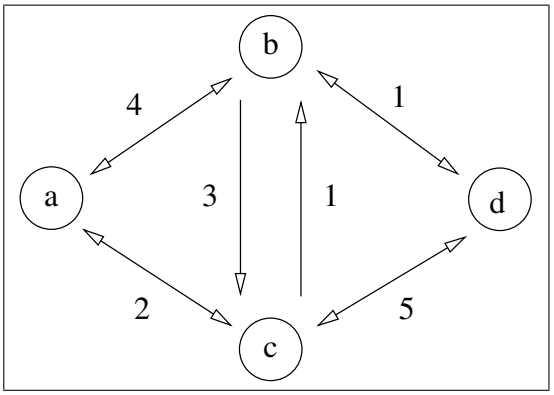

Abbildung 1: Ein Beispielnetz mit 5 Knoten

1.2 Knoten

Jeder Knoten des Graphen hat folgende Eigenschaften:

Distanz: Die beste angenommene Distanz eines Knoten zum Startpunkt

Vorgänger: Der Name des Vorgängerknotens auf dem bislang besten Pfad zum Startort

Offen/Erledigt: Diese Eigenschaft (im Folgenden Variable `qs`) eines Knotens kann 3 Werte annehmen:

- **-** der Knoten wurde noch nicht besucht
- **q** der Knoten wurde besucht, seine kürzeste Distanz zum Startort ist aber noch nicht bekannt
- **s** der Knoten ist erledigt (*settled*), d.h. seine kürzeste Distanz zum Start ist bekannt.

1.3 Pseudocode

```
//initialisieren
setze die Distanzen aller Knoten auf unendlich: z.B. $dist = 1000000$
setze den Vorgaengerknoten aller Knoten auf "" (leerer String)
setze die offen/erledigt-Eigenschaft (qs) aller Knoten auf "-"

setze qs des Startknotens auf "q"
setze die Distanz dist des Startknotens auf dist=0

while( es gibt noch knoten, bei denen qs=="q" ist ){
   knoten u = extractMinimum();
   relaxNeighbors(u);
}
```

Methode extractMinimum():

```
extractMinimum(){
   finde den Knoten u mit minimaler Distanz dist aus der Menge
      aller Knoten, bei denen qs=="q" ist ;
   u.setQs("s");
   return u;
}
```

Methode relaxNeighbors(u):

```
relaxNeighbors(knoten u){
   foreach Knoten v, v.getQs() ist nicht "s"  &&
                     v ist nachbar von u{
      if (v.getDist() > u.getDist() + getDistanz( u,v) ){
         v.setDist(  u.getDist() + getDistanz(u,v)  );
         v.setPre(u);
         v.setQs("q");
      }
   }
}
```

Für die Realisierung als Java-Programm kann man u und v als Objekte der Klasse `KnotenZustand` darstellen. Das Klassendiagramm von KnotenZustand ist in Abb. 2 (Angang) gezeigt.

Die beiden Methoden `relaxNeighbors(String u)` und `extractMinimum()` sind in der Klasse Dijkstra realisiert. Die Abb. 3 zeigt deren Klassendiagramm und ist ebenfalls im Anhang.

2 Programmierung

2.1 Aufbau der Link-States-Matrix

Die Link-States-Matrix besteht aus einer Hash-Map, deren Schlüssel die Knotennamen bilden (String) und deren zugeordnete Werte wieder eine Hash-Map sind (Zeilen Tab. 1).

Die Schlüssel-Werte-Paare der Hash-Maps, die die Zeilen der folgenden Tabelle (Tab. 1) bilden, enthalten einen Zielknoten (Schlüssel) mit seiner Entfernung (Wert).

Die beiden Abbildungen in Tabelle 1 sind gleichwertig: Die inneren Hash-Maps wurden zunächst als Tabellen (links) und dann vereinfacht als Menge von Wertepaaren dargestellt (rechts).

String	Hash-Map	
a	b	4
	c	2
b	a	4
	c	3
	d	1
c	a	2
	b	1
	d	5
d	b	1
	c	5

String	Hash-Map			
a		(b,4)	(c,2)	
b	(a,4)		(c,3)	(d,1)
c	(a,2)	(b,1)		(d,5)
d		(b,1)	(c,5)	

Tabelle 1: Link-States-Matrix mit Hash-Maps

A Programmieraufgabe

Die beiden Klassendiagramme und das Codegerüst sollen unter **Einhaltung der Vorgaben** (Klassen, Bezeichner) ausprogrammiert werden, so dass das Beispielnetz geroutet wird.

Die mit **tar** [1] (und nur mit tar und nichts ausser tar und keinesfalls irgend was anderem) archivierten NB-Projektverzeichnisse sind per e-mail an **dienert@wara.de** noch dieses Jahr einzureichen. Die Projektverzeichnisse müssen eine direkt ausführbare jar-Datei enthalten (Build Project nicht vergessen).

Und wehe, die geschweiften Klammern sind da, wo sie nicht hingehören!!!!

B Klassendiagramme und Codegerüst

B.1 Klassse KnotenZustand

B.2 Klasse Dijkstra

B.3 Code-Gerüst

[1] Für wintendo-Spieler gibt es izarc oder tugzip

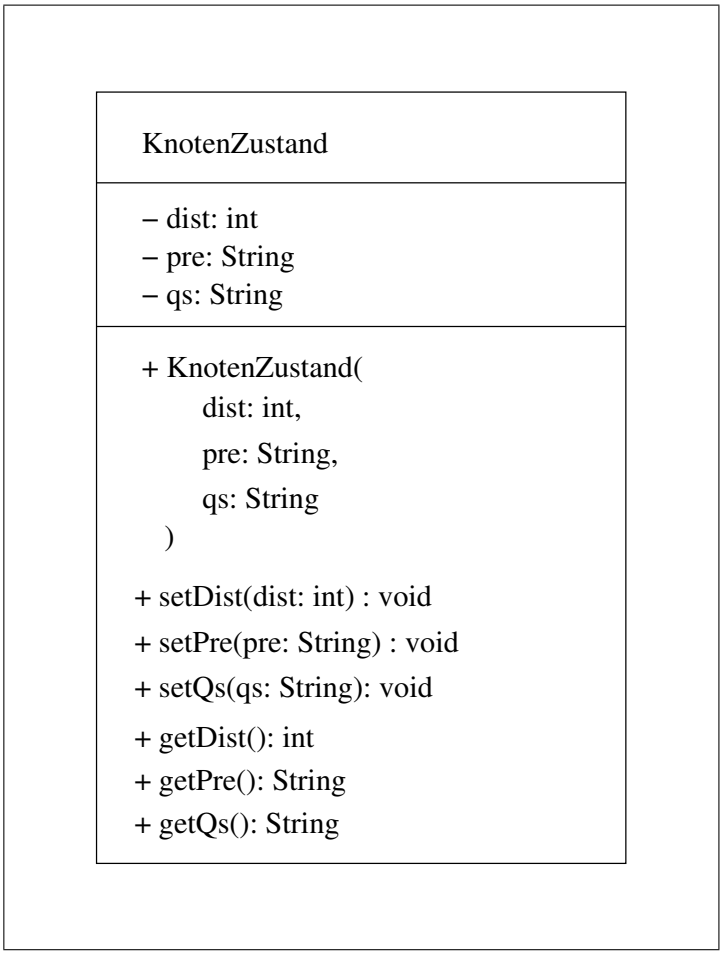

Abbildung 2: Klassendiagramm

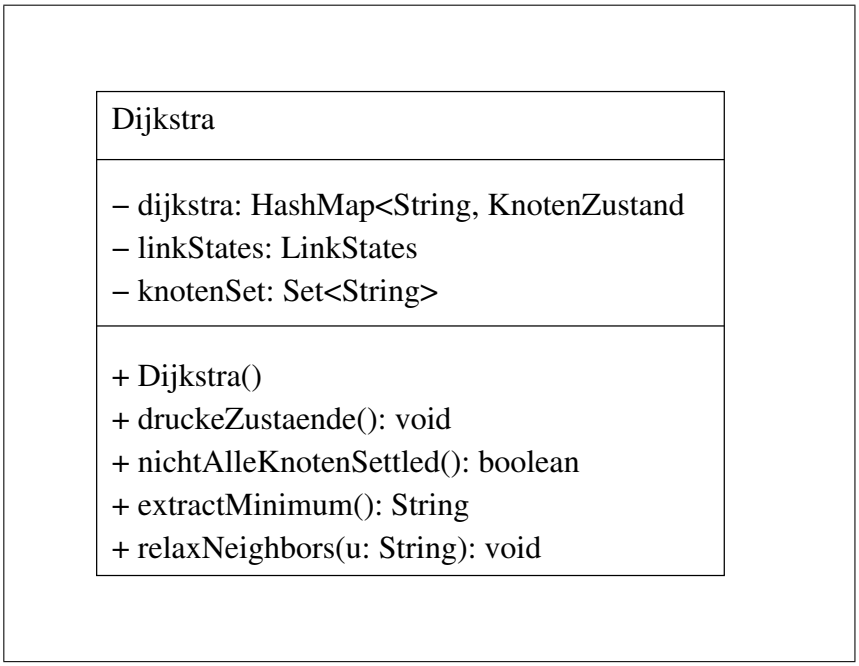

Abbildung 3: Klassendiagramm

```
/**
 * linkStates-objekte speichern in einer hashMap zu jedem knoten eine
 * hashMap, die zu jedem knoten die distanz enthaelt
 * etwa so:
 * a, ( (a,0), (b,4), (c,2), (d,1000000) )
 * b, ( (a,4), (b,0), (c,3), (d,1)  )
 * usw.
 **/

package dijkstra;

import java.util.HashMap;
import java.util.Set;

class LinkStates{

    HashMap<String, HashMap> matrix;
    HashMap<String, Integer> zeile;

    //link-state hinzufuegen:

    public void addLinkState(String startKnoten, String nachbar, int distanz){
      // vervollstaendigen
   }

    public Set<String> getNachbarn(String knoten){
      //vervollstaendigen
   }

    public Set<String> getAlleKnoten(){
      //vervollstaendigen
   }

    public int getDistanz(String u, String v){
      //vervollstaendigen
   }

    public void matrixDrucken(){
      //vervollstaendigen
   }

    public LinkStates(){
        this.addLinkState("a","b",4);
        this.addLinkState("b","a",4);
        this.addLinkState("b","c",3);
        this.addLinkState("b","d",1);
        this.addLinkState("a","c",2);
        this.addLinkState("c","a",2);
        this.addLinkState("c","b",1);
        this.addLinkState("c","d",5);
        this.addLinkState("d","b",1);
        this.addLinkState("d","c",5);

        //this.matrixDrucken();
    }

    public static void main(String[] args){
        new LinkStates();
    }
}
```

C Beispiel eines Programmlaufs

```
-------------------------------------
dijkstra: d:( dist=1000000, vorgaenger=, qs=- )
dijkstra: b:( dist=1000000, vorgaenger=, qs=- )
dijkstra: c:( dist=1000000, vorgaenger=, qs=- )
dijkstra: a:( dist=1000000, vorgaenger=, qs=- )
-------------------------------------
*****************************************************
knoten u ist der knoten: a
dessen zustand ist: ( dist=0, vorgaenger=, qs=s )
die hashMap mit den nachbarn von a :
{b=4, c=2}
-------------------------------------
dijkstra: d:( dist=1000000, vorgaenger=, qs=- )
dijkstra: b:( dist=4, vorgaenger=a, qs=q )
dijkstra: c:( dist=2, vorgaenger=a, qs=q )
dijkstra: a:( dist=0, vorgaenger=, qs=s )
-------------------------------------
*****************************************************
knoten u ist der knoten: c
dessen zustand ist: ( dist=2, vorgaenger=a, qs=s )
die hashMap mit den nachbarn von c :
{d=5, b=1, a=2}
-------------------------------------
dijkstra: d:( dist=7, vorgaenger=c, qs=q )
dijkstra: b:( dist=3, vorgaenger=c, qs=q )
dijkstra: c:( dist=2, vorgaenger=a, qs=s )
dijkstra: a:( dist=0, vorgaenger=, qs=s )
-------------------------------------
*****************************************************
knoten u ist der knoten: b
dessen zustand ist: ( dist=3, vorgaenger=c, qs=s )
die hashMap mit den nachbarn von b :
{d=1, c=3, a=4}
-------------------------------------
dijkstra: d:( dist=4, vorgaenger=b, qs=q )
dijkstra: b:( dist=3, vorgaenger=c, qs=s )
dijkstra: c:( dist=2, vorgaenger=a, qs=s )
dijkstra: a:( dist=0, vorgaenger=, qs=s )
-------------------------------------
*****************************************************
knoten u ist der knoten: d
dessen zustand ist: ( dist=4, vorgaenger=b, qs=s )
die hashMap mit den nachbarn von d :
{b=1, c=5}
-------------------------------------
dijkstra: d:( dist=4, vorgaenger=b, qs=s )
dijkstra: b:( dist=3, vorgaenger=c, qs=s )
dijkstra: c:( dist=2, vorgaenger=a, qs=s )
dijkstra: a:( dist=0, vorgaenger=, qs=s )
-------------------------------------
```

D Übung zum Dijkstra-Algorithmus

Vervollständige die Tabellen anhand des Pseudo-Codes:

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			

u= u.getDist()=

v= v.getDist()=

getDistanz(u,v)=

knoten	dist	pre	qs
a			
b			
c			
d			