Der Dijkstra-Algorithmus

Klasse Algorithmus

```
1  public class Algorithmus {
2
3    Knoten[] graph;
4
5    Knoten startknoten = null;
6    Knoten zielknoten = null;
7
8    public Algorithmus() {
9      this.initialisiere();
10   }
11
12   public void initialisiere() {
13     graph = new Knoten[19];
14     graph[1] = new Knoten("A", 0);
15     graph[15].fuegeKanteHinzu(new Kante(graph[14], 91));
16     graph[18].fuegeKanteHinzu(new Kante(graph[17], 23));
17     // ...
18   }
19
20   public void berechneWeg() {
21     Warteschlange ws = new Warteschlange();
22
23     Knoten aktuellerKnoten = this.startknoten;
24
25     while (aktuellerKnoten != zielknoten) {
26       aktuellerKnoten.expandiereKnoten(ws);
27       aktuellerKnoten = ws.gibErstesElementZurueck();
28     }
29   }
30
31   public static void main(String[] argv) {
32     Algorithmus algo = new Algorithmus();
33     algo.startknoten = algo.graph[9];
34     algo.zielknoten = algo.graph[14];
35     algo.berechneWeg();
36     algo.zielknoten.gibPfadAus();
37   }
38 }
```

Klasse Kante

```
1  public class Kante {
2
3    public Knoten zielknoten;
4    public int entfernung;
5
6    public Kante(Knoten knoten, int entfernung) {
7      this.zielknoten = knoten;
8      this.entfernung = entfernung;
9    }
10 }
```

Klasse Knoten

```
1  public class Knoten {
2
3    public int entfernung = 0;
4    public Knoten nachfolger;
5    public String name;
6    public Knoten vorgaengerImPfad;
7    public Kante[] kanten = new Kante[10];
8    public int kantenanzahl = 0;
9    public boolean istMarkiert = false;
10
11
12   public Knoten(String name, int entfernung) {
13     this.name = name;
14     this.entfernung = entfernung;
15   }
16
17
18   public void fuegeKnotenEin(Knoten knoten) {
19     if (nachfolger == null) {
20       nachfolger = knoten;
21     } else {
22       if (knoten.entfernung < nachfolger.entfernung) {
23         knoten.nachfolger = nachfolger;
24         nachfolger = knoten;
25       } else {
26         nachfolger.fuegeKnotenEin(knoten);
27       }
28     }
29   }
30
31
32   public void entferneKnoten(Knoten knoten) {
33     if (nachfolger != null) {
34       if (nachfolger == knoten) {
35         nachfolger = knoten.nachfolger;
36       } else {
37         nachfolger.entferneKnoten(knoten);
38       }
39     }
40   }
41
42
43   public void fuegeKanteHinzu(Kante kt) {
44     this.kanten[kantenanzahl] = kt;
45     this.kantenanzahl = this.kantenanzahl + 1;
46   }
47
48   public void gibRestWarteschlangeAus() {
49     System.out.println("Knoten: " + name);
50     if (nachfolger != null) {
51       nachfolger.gibRestWarteschlangeAus();
52     }
53   }
54
55   public void gibPfadAus() {
56     if (this.vorgaengerImPfad != null) {
57       this.vorgaengerImPfad.gibPfadAus();
58     }
59     System.out.println("Knoten: " + name);
60   }
61
```

```
62    public void expandiereKnoten(Warteschlange ws) {
63      for (int i = 0; i < this.kantenanzahl; i++) {
64        Knoten nachbarknoten = kanten[i].zielknoten;
65        int kantenentfernung = kanten[i].entfernung;
66        if (!nachbarknoten.istMarkiert) {
67          if (nachbarknoten.entfernung == 0) {
68            nachbarknoten.entfernung = kantenentfernung
69                + this.entfernung;
70            nachbarknoten.vorgaengerImPfad = this;
71            ws.fuegeKnotenEin(nachbarknoten);
72          } else {
73            if (nachbarknoten.entfernung > kantenentfernung
74                + this.entfernung) {
75              ws.entferneKnoten(nachbarknoten);
76              nachbarknoten.entfernung = kantenentfernung
77                  + this.entfernung;
78              nachbarknoten.vorgaengerImPfad = this;
79              ws.fuegeKnotenEin(nachbarknoten);
80            }
81          }
82        }
83      }
84      this.istMarkiert = true;
85    }
86  }
```

Klasse Warteschlange

```
1   public class Warteschlange {
2
3     private Knoten kopf;
4
5     public Warteschlange() {
6       kopf = new Knoten("Kopf", -1);
7     }
8
9     public void fuegeKnotenEin(Knoten knoten) {
10      kopf.fuegeKnotenEin(knoten);
11    }
12
13    public void entferneKnoten(Knoten knoten) {
14      kopf.entferneKnoten(knoten);
15    }
16
17    public Knoten gibErstesElementZurueck() {
18      Knoten hilfsknoten = kopf.nachfolger;
19      kopf.entferneKnoten(kopf.nachfolger);
20      return hilfsknoten;
21
22    }
23
24    public void gibWarteschlangeAus() {
25      kopf.gibRestWarteschlangeAus();
26    }
27  }
```