Lösung 4. Aufgabe “Urlaubsfahrt”

Lösungsidee

Der zur Lösung entwickelte Algorithmus kann in drei Schritte aufgeteilt werden:

Schritt 0

Da keine explizite Garantie über das Eingabeformat vorliegt, werden die Tankstellen zunächst der Entfernung nach aufsteigend (näher am Start nach näher am Ziel) sortiert. Hierfür wird von Java ein **Quicksort** verwendet.

Schritt 1

Es werden die vom Start aus mit der Tankfüllung erreichbaren Tankstellen bestimmt. Diese sind alle diejenigen, die maximal Tankfüllung/Verbrauch*100 Kilometer entfernt sind.

Schritt 2

Die Tankstellen werden in der in Schritt 0 bestimmten Reihenfolge durchgegangen. Für jede Tankstelle werden die von ihr aus mit voller Tankfüllung erreichbaren Tankstellen (alle die maximal Volle Tankfüllung/Verbrauch*100 Kilometer entfernt sind) bestimmt. Wenn eine Tankstelle so erreicht wird, vergleicht sie die bis zu ihr erfolgten Tankvorgänge auf diesem Weg mit ihrem aktuellen “Rekordhalter” (dem bisher “besten” Weg, der zu ihr gefunden wurde), sofern er existiert - sonst wurde gerade eben ein Weg zu ihr gefunden, der als bisher Bester gespeichert wird. Sind es weniger, wird die neue Route ihre “Bestroute”, als Weg mit minimal vielen Tankvorgängen. Sind es gleich viele, werden die Kosten auf dem Weg verglichen - sind sie niedriger, haben wir einen neuen Rekordhalter. Ist von einer Tankstelle aus das Ende erreichbar, wird sie mit anderen Tankstellen verglichen, von denen aus auch das Ende erreicht werden konnte. Ist die Route / der Pfad der über diese Tankstelle zum Ende führt “besser” - d.h. weniger Stopps oder geringere Kosten - haben wir einen neuen besten Weg zum Ziel gefunden.

So finden die Tankstellen der Reihe nach jeweils “beste” Routen zu sich - und können diese dann direkt an von ihnen erreichbare Tankstellen weitergeben.

Um zu bestimmen wie viel jeweils zu tanken ist, wird zunächst nicht gespeichert, wie viel getankt wird - beim nächsten Tankvorgang wird rückwirkend bestimmt, ob die vorherige Tankstelle billiger gewesen wäre - wenn ja, hätte man volltanken sollen; dies wird dann rückwirkend “gespeichert.” Wenn nein, sollte man dort gerade so viel wie nötig tanken. An einem Beispiel:

1. Da der Weg möglichst kurz ist, kann keine Tankstelle übersprungen werden
2. Also reicht es, jeweils zwei Tankstellen zu vergleichen - es muss bei allen 3 getankt werden, nur wieviel ist die Frage

```
Start - Tankstelle A (billig) - Tankstelle B (sehr billig) - Tankstelle C (teuer) - Ziel
```

   Es lohnt sich, zuerst bei Tankstelle A nur so viel zu tanken, wie zum Erreichen von Tankstelle B notwendig - denn dort ist der Sprit billiger. Wir stellen fest A > B, also tanken wir bei A nur so viel

wie nötig. Wenn wir dann bei Tankstelle B ankommen, stellen wir fest, dass C noch teuerer ist - B < C. Also lohnt es sich, bei B vollzutanken - Sprit den wir von B noch übrig haben, müssen wir nicht bei C für einen höheren Preis erwerben. Bei C tanken wir dann nur noch soviel, wie zum Erreichen des Ziels notwendig. **Ist die zuerst kommende Tankstelle billiger, sollte man volltanken; sonst nur soviel, wie zum Erreichen der Nächsten notwendig.**

Umsetzung

Bibliotheken

- `java.io.File`, `java.io.FileReader`, `java.io.BufferedReader`: Zum Öffnen & Lesen der Dateien mit der Aufgabenstellung
- `java.util.Arrays`: Zum Sortieren der Eingabe
- `java.util.ArrayList`: Dynamisch erweiterbare Listen, um "beste Route" ([Route]-Klasse) zu speichern, oder erreichbare Tankstellen
- `java.util.Iterator`: Iterator, nützlich um `String.join` benutzen zu können

Klassen

`Tankstelle`

Als grundlegendste Klasse implementieren wir die `Tankstelle`. Diese speichert:

- Die Position auf dem Weg als Ganzzahl (`int`)
- Die Kosten in Cent als Ganzzahl (`int`)
- Den besten Weg, der zu ihr führt (`Route`)
- Die von ihr aus erreichbaren Tankstellen (`ArrayList<Tankstelle>`)

`Tankvorgang`

Speichert die Tankstelle (jedoch als Referenz, nicht als Kopie) und wie viel getankt wurde (Gleitpunktzahl, `double`).

`Route`

Hauptsächlich eine Liste von Objekten der `Tankvorgang`-Klasse. Speichert zusätzlich noch die bisher angefallenen Kosten und den übrigen Sprit (Gleitpunktzahlen (`double`s), müssen so nicht stets neu berechnet werden).

Verwendung

Ausgeben (einer) der "besten Routen" für eine gegebene Aufgabenstellung in einer Datei: `java -jar Urlaubsfahrt.jar <pfad_zur_datei>` (*angeben des **vollständigen** Pfades empfohlen*)

Beispiele

fahrt1.txt

Eine optimale Lösung ist: Bei Position 100 km, Preis €1/l, 10.0l für €14.5 getankt -> Bei Position 400 km, Preis €1/l, 48.0l für €67.2 getankt - Kosten insgesamt: €81.7 - Stopps: 2

fahrt2.txt

Eine optimale Lösung ist: Bei Position 1118 km, Preis €1/l, 239.76l für €275.724 getankt -> Bei Position 2240 km, Preis €1/l, 266.159999999999971 für €385.93199999999996 getankt -> Bei Position 3346 km, Preis €1/l, 258.96l für €300.3936 getankt -> Bei Position 4455 km, Preis €1/l, 268.08l für €351.18479999999994 getankt -> Bei Position 5534 km, Preis €1/l, 19.919999999999987l für €23.306399999999986 getankt -> Bei Position 6651 km, Preis €1/l, 253.92l für €302.1648 getankt -> Bei Position 7737 km, Preis €1/l, 274.0l für €364.42 getankt -> Bei Position 8242 km, Preis €1/l, 123.679999999999981 für €160.784 getankt -> Bei Position 9100 km, Preis €1/l, 216.0l für €278.64 getankt - Kosten insgesamt: €600.208 - Stopps: 9

fahrt3.txt

Eine optimale Lösung ist: Bei Position 465 km, Preis €1/l, 80.0l für €99.2 getankt - Kosten insgesamt: €99.2 - Stopps: 1

fahrt4.txt

Eine optimale Lösung ist: Bei Position 264 km, Preis €1/l, 88.2l für €105.84 getankt -> Bei Position 607 km, Preis €1/l, 100.799999999999981 für €131.04 getankt - Kosten insgesamt: €236.88 - Stopps: 2

fahrt5.txt

Eine optimale Lösung ist: Bei Position 194 km, Preis €1/l, 244.0l für €319.64 getankt -> Bei Position 1305 km, Preis €1/l, 225.769999999999981 für €318.3357 getankt -> Bei Position 2433 km, Preis €1/l, 231.0l für €314.16 getankt -> Bei Position 3542 km, Preis €1/l, 244.0l für €319.64 getankt -> Bei Position 4642 km, Preis €1/l, 237.72l für €290.0184 getankt -> Bei Position 5751 km, Preis €1/l, 241.709999999999981 für €335.97689999999994 getankt -> Bei Position 6883 km, Preis €1/l, 244.0l für €309.88 getankt -> Bei Position 7997 km, Preis €1/l, 216.95l für €290.71299999999997 getankt -> Bei Position 9049 km, Preis €1/l, 199.709999999999981 für €249.63749999999996 getankt - Kosten insgesamt: €583.1501 - Stopps: 9

Quellcode

Tankstelle.java

```java
package appguru;

import java.util.ArrayList;

/**
 *
 * @author lars
 */
public class Tankstelle {

    public int distanz;
    public int preis;
    public int min_stopps_bis = Integer.MAX_VALUE / 2;
    public boolean erreicht = false;
    public Route beste_route = new Route();

    public Tankstelle(int distanz, int preis) {
        this.distanz = distanz;
        this.preis = preis;
    }

    // Passt den letzten Tankvorgang an
    public static void passeLetztesTankenAn(Tankstelle vorige, Tankstelle naechste) {
        if (!naechste.beste_route.stopps.isEmpty()) {
            double zu_tanken;
            double min_zu_tanken = Urlaubsfahrt.VERBRAUCH * (naechste.distanz - vorige.distanz);
                // Genau soviel wie noch nötig tanken
            Tankvorgang letzter =
                naechste.beste_route.stopps.get(naechste.beste_route.stopps.size() - 1);
            if (letzter.tankstelle.preis < naechste.preis) {
                zu_tanken = Urlaubsfahrt.GROESSE_TANK; // Volltanken
            } else { // Fall 2
                zu_tanken = min_zu_tanken;
            }
            double sprit_uebrig = zu_tanken - min_zu_tanken; // Tank ist voll, oder es bleibt nix
                übrig
            zu_tanken -= vorige.beste_route.sprit_uebrig; // Können nicht überfüllen, und wollen
                sowieso weniger tanken in Fall 2
            naechste.beste_route.sprit_uebrig = vorige.beste_route.sprit_uebrig = sprit_uebrig;
                // Übrigen Sprit speichern
            letzter.liter_getankt = zu_tanken; // Rückwirkend hätten wir soviel tanken sollen
            naechste.beste_route.kosten += zu_tanken * vorige.preis;
        }
    }
```

```
40
41      public void zielErreicht() {
42          double noch_zu_fahren = Urlaubsfahrt.LAENGE_STRECKE - this.distanz;
43          if (!beste_route.stopps.isEmpty()) {
44              passeLetztesTankenAn(beste_route.stopps.get(beste_route.stopps.size() -
                    1).tankstelle, this);
45          }
46          double zu_tanken = noch_zu_fahren * Urlaubsfahrt.VERBRAUCH;
47          zu_tanken -= beste_route.sprit_uebrig;
48          beste_route.stopps.add(new Tankvorgang(this, zu_tanken));
49          beste_route.kosten += zu_tanken * this.preis;
50      }
51
52      // t ist von uns aus erreichbar
53      public void erreichbar(Tankstelle t) {
54          if (min_stopps_bis + 1 < t.min_stopps_bis || (min_stopps_bis + 1 == t.min_stopps_bis &&
                beste_route.kosten < t.beste_route.kosten)) {
55              t.beste_route = new Route();
56              t.beste_route.stopps = new ArrayList();
57              t.beste_route.stopps.addAll(beste_route.stopps);
58              passeLetztesTankenAn(this, t);
59              t.beste_route.stopps.add(new Tankvorgang(this, 0));
60              t.min_stopps_bis = min_stopps_bis + 1;
61              if (erreicht) {
62                  t.erreicht=true;
63              }
64          }
65      }
66
67      @Override
68      public String toString() { // Tankstelle schön formattieren
69          return "Position " + distanz + " km, Preis " + (preis / 100) + "€/l"; // "(s="+distanz+",
                p="+preis+", t="+min_steps_to+")";
70      }
71
72      @Override
73      public int hashCode() {
74          return this.distanz; // Distanz als HashCode
75      }
76
77      @Override
78      public boolean equals(Object obj) { // Vergleichsoperation
79          if (this == obj) {
80              return true;
81          }
82          if (obj == null) {
83              return false;
84          }
```

```
85          if (getClass() != obj.getClass()) {
86              return false;
87          }
88          final Tankstelle other = (Tankstelle) obj;
89          return this.distanz == other.distanz; // Positionen zu vergleichen reicht per
                Aufgabenstellung
90      }
91
92  }
```

Tankvorgang.java

```
1  package appguru;
2
3  /**
4   *
5   * @author lars
6   */
7  // SPeichert einen Tankvorgang
8  public class Tankvorgang {
9
10     public Tankstelle tankstelle; // Tankstelle (Referenz)
11     public double liter_getankt; // Getankte Liter
12
13     // Konstruktor zum Initialisieren
14     public Tankvorgang(Tankstelle tankstelle, double liter_getankt) {
15         this.tankstelle = tankstelle;
16         this.liter_getankt = liter_getankt;
17     }
18
19     @Override
20     public String toString() {
21         return "Bei " + tankstelle + ", " + liter_getankt + "l für
               "+(tankstelle.preis*liter_getankt/100)+"€ getankt";
22     }
23 }
```

Route.java

```
1  package appguru;
2
3  import java.util.ArrayList;
4  import java.util.Iterator;
5
6  /**
7   *
8   * @author lars
9   */
```

```java

// Route als Struktur zum Speichern
public class Route {
    public ArrayList<Tankvorgang> stopps=new ArrayList(); // Tankvorgänge
    public double kosten; // Kosten in Cent
    public double sprit_uebrig; // Sprit übrig in Liter

    @Override
    public String toString() {
        // Route schön formattieren - Stopps getrennt mit "->", Insgesamtkosten, Anzahl Stopps
        return String.join(" -> ", new Iterable<String>() {
            @Override
            public Iterator<String> iterator() {
                return stopps.stream().map(Tankvorgang::toString).iterator();
            }
        })+" - Kosten insgesamt: "+kosten/100+"€ - Stopps: " + stopps.size();
    }
}
```

Urlaubsfahrt.java

```java
package appguru;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.Arrays;
import java.util.ArrayList;

/**
 *
 * @author lars
 */
public class Urlaubsfahrt {

    // Selbsterklärende Variablennamen, näheres in readFile
    public static double VERBRAUCH;
    public static int GROESSE_TANK;
    public static int FUELLUNG_TANK;
    public static int LAENGE_STRECKE;
    public static double ANFANGS_REICHWEITE;
    public static double VOLLE_REICHWEITE;
    public static Tankstelle[] TANKSTELLEN;
    public static int MIN_STOPPS_BIS = Integer.MAX_VALUE;

    // Liest die Aufgabenstellung
    public static void readFile(File file) throws Exception {
        var reader = new BufferedReader(new FileReader(file)); // Datei öffnen
```

```
28          VERBRAUCH = Integer.parseInt(reader.readLine()) / 100.0; // Erste Zeile: Verbrauch pro
                100km lesen, in Verbrauch pro km umrechnen, und speichern
29          GROESSE_TANK = Integer.parseInt(reader.readLine()); // 2. Zeile: Tankgröße
30          FUELLUNG_TANK = Integer.parseInt(reader.readLine()); // 3. Zeile: Tankfüllung
31          LAENGE_STRECKE = Integer.parseInt(reader.readLine()); // 4. Zeile: Länge der Strecke
32          TANKSTELLEN = new Tankstelle[Integer.parseInt(reader.readLine())]; // 5. Zeile: Anzahl
                Tankstellen, Array initialisieren
33          for (int t = 0; t < TANKSTELLEN.length; t++) {
34              // Strecke und Preis sind pro Zeile mit einem oder mehreren Leerzeichen getrennt,
                    daher der Regex "\\s+"
35              String[] strecke_und_preis = reader.readLine().split("\\s+");
36              int strecke = Integer.parseInt(strecke_und_preis[0]);
37              int preis = Integer.parseInt(strecke_und_preis[1]);
38              Tankstelle tanke = new Tankstelle(strecke, preis); // Tankstelle erzeugen
39              TANKSTELLEN[t] = tanke; // Und speichern
40          }
41          // Tankstellen der Strecke nach sortieren (nah am Start nach weit vom Start)
42          Arrays.sort(TANKSTELLEN, 0, TANKSTELLEN.length, (Object o1, Object o2) ->
                Integer.compare(((Tankstelle) o1).distanz, ((Tankstelle) o2).distanz));
43          ANFANGS_REICHWEITE = FUELLUNG_TANK / (double) VERBRAUCH; // Anfangsreichweite:
                Tankfüllung/Verbrauch
44          VOLLE_REICHWEITE = GROESSE_TANK / (double) VERBRAUCH; // Maximale Reichweite: Volle
                Tankfüllung/Verbrauch
45      }
46
47      // Führt den Algorithmus zur Lösung durch
48      public static void findeOptimalenWeg() {
49          // Trivialfall: Ziel direkt erreichbar mit anfänglicher Tankfüllung
50          if (ANFANGS_REICHWEITE >= LAENGE_STRECKE) {
51              System.out.println("0 mal tanken notwendig, Ziel direkt erreichbar");
52              return;
53          }
54
55          // Alle vom Start aus erreichbaren Tankstellen bestimmen
56          for (int t = 0; t < TANKSTELLEN.length; t++) {
57              if (TANKSTELLEN[t].distanz > ANFANGS_REICHWEITE) {
58                  break;
59              }
60              TANKSTELLEN[t].erreicht = true; // Tankstelle erreicht
61              TANKSTELLEN[t].min_stopps_bis = 0; // Keine Stopps bis zur Tankstelle nötig
62              TANKSTELLEN[t].beste_route.sprit_uebrig = FUELLUNG_TANK - (TANKSTELLEN[t].distanz *
                    VERBRAUCH); // Bei Erreichen noch so viel Sprit übrig, wie von der
                    Anfangsfüllung bleibt
63          }
64
65          // Tankstellen durchgehen, von Tankstellen aus erreichbare Tankstellen & Routen bestimmen
66          boolean loesung_existiert = false; // existiert eine Lösung?
67          Route beste_route = new Route(); // Rekordhalter-Route initialisieren
```

```
68          beste_route.kosten = Double.MAX_VALUE;
69          for (int t = 0; t < TANKSTELLEN.length; t++) {
70              // Lässt sich von der Tankstelle aus das Ende erreichen?
71              if (LAENGE_STRECKE - TANKSTELLEN[t].distanz <= VOLLE_REICHWEITE) {
72                  // Tankstelle muss erreichbar sein
73                  if (TANKSTELLEN[t].erreicht) {
74                      loesung_existiert = true;
75                      TANKSTELLEN[t].zielErreicht();
76                      if (TANKSTELLEN[t].min_stopps_bis < MIN_STOPPS_BIS) { // Neuer Rekordhalter!
77                          MIN_STOPPS_BIS = TANKSTELLEN[t].min_stopps_bis;
78                          beste_route = TANKSTELLEN[t].beste_route;
79                      } else if (TANKSTELLEN[t].min_stopps_bis == MIN_STOPPS_BIS) { // Gleich viele
                             Stopps, aber vielleicht billiger?
80                          if (TANKSTELLEN[t].beste_route.kosten < beste_route.kosten) { // Falls
                                 billiger
81                              beste_route = TANKSTELLEN[t].beste_route; // Neuer Rekordhalter !
82                          }
83                      }
84                  }
85              } else {
86                  // Welche anderen Tankstellen lassen sich von der Tankstelle aus erreichen
87                  for (int t2 = t + 1; t2 < TANKSTELLEN.length; t2++) {
88                      if (TANKSTELLEN[t2].distanz - TANKSTELLEN[t].distanz > VOLLE_REICHWEITE) {
89                          break;
90                      }
91                      TANKSTELLEN[t].erreichbar(TANKSTELLEN[t2]);
92                  }
93              }
94          }
95          if (!loesung_existiert) { // Es existiert keine Lösung, das Ziel kann nicht erreicht
                 werden
96              System.out.println("Es existiert keine Lösung: die Tankstellen liegen zu weit
                     auseinander, das Ziel kann nicht erreicht werden");
97          } else { // Sonst: gebe gefundene Lösung aus
98              System.out.println("Eine optimale Lösung ist: " + beste_route);
99          }
100     }
101
102     public static void main(String[] args) {
103         if (args.length != 1) { // Zu viele / zu wenige Argumente
104             System.out.println("Argumente: <pfad_zur_datei>");
105             return;
106         }
107         try {
108             Urlaubsfahrt.readFile(new File(args[0])); // Datei lesen
109         } catch (Exception ex) {
110             // Fehlermeldung
111             System.out.println("Datei existiert nicht / ist nicht lesbar / ist falsch
```

```
                formattiert.");
            return;
        }
        // Algorithmus starten
        Urlaubsfahrt.findeOptimalenWeg();
    }

}
```